2021 ASIAA SUMMER PROGRAM

HIGH ORDER FINITE DIFFERENCE WENO SCHEMES: IMPLEMENTATION & VALIDATION FOR 2D EULER EQUATIONS

STUDENT: HO WAN CHENG

SUPERVISOR: DR. SHANG, HSIEN

MENTOR: MR. BANDOPADHYAY, SOMDEB

Project Objectives

- **To test 5 different WENO schemes**
  - WENO-JS
  - WENO-Z
  - WENO-Z+ (implemented in previous code)
  - WENO-YC
  - WENO-M

- **To validate the implemented schemes**
  - 2D isentropic vortex problem

- **To evaluate the computational cost of each implemented scheme**

2D Euler equations

$$\frac{\partial}{\partial t}\begin{Bmatrix}\rho\\ \rho v_x\\ \rho v_y\\ \rho v_z\\ E\end{Bmatrix}+\frac{\partial}{\partial x}\begin{Bmatrix}\rho v_x\\ \rho v_x^{\mathbf{2}}+P_t\\ \rho v_x v_y\\ \rho v_x v_z\\ (E+P_t)v_x\end{Bmatrix}+\frac{\partial}{\partial y}\begin{Bmatrix}\rho v_y\\ \rho v_y v_x\\ \rho v_y^{\mathbf{2}}+P_t\\ \rho v_y v_z\\ (E+P_t)v_y\end{Bmatrix}=0$$

$$\frac{\partial \mathbf{U}}{\partial t}+\frac{\partial \mathbf{F}}{\partial x}+\frac{\partial \mathbf{G}}{\partial y}=0$$

Consider $\frac{\partial \mathbf{U}}{\partial t}$ along x- and y- direction:

$$\frac{\partial \mathbf{U}_i}{\partial t}=-\frac{1}{\Delta x}\left(\hat{\mathrm{F}}_{i+\frac{1}{2},j}-\hat{\mathrm{F}}_{i-\frac{1}{2},j}\right)$$

$$\frac{\partial \mathbf{U}_j}{\partial t}=-\frac{1}{\Delta y}\left(\hat{\mathrm{G}}_{i,j+\frac{1}{2}}-\hat{\mathrm{G}}_{i-\frac{1}{2},j}\right)$$

Lax-Friedrichs flux splitting

$$f^{\pm}_{i+\frac{1}{2}} = \frac{1}{2}(\mathrm{F} \pm \lambda \mathrm{U})_{i+\frac{1}{2}}$$

where $\lambda = \begin{Bmatrix} v_x - c_s \\ v_x \\ v_x \\ v_x \\ v_x + c_s \end{Bmatrix}$ is the characteristic speed, c_s is the speed of sound. The following can then be defined:

$$\mathcal{F}^{\pm} = \mathcal{L}\left(\widetilde{U}_{i+\frac{1}{2}}\right)\left(f^{\pm}_{i+\frac{1}{2}}\right)$$

where $\mathcal{L}$ is the left eigenvector of the flux Jacobian and $\widetilde{U}_{i+\frac{1}{2}} = \frac{1}{2}(U_i + U_{i+1})$

Hence the numerical flux is expressed as

$$\hat{f}_{i+\frac{1}{2}} = \mathcal{R}\left(\widetilde{U}_{i+\frac{1}{2}}\right)\left[(\mathcal{F}^{+})^{L}_{i+\frac{1}{2}} + (\mathcal{F}^{-})^{R}_{i+\frac{1}{2}}\right]$$

where $\mathcal{R}$ is the right eigenvector of the flux Jacobian and $\mathcal{F}^{\pm}$ can be found using the WENO schemes

Right and left eigenvectors of flux Jacobian

Following the formulation by Stone JM, Gardiner TA, Teuben P, Hawley JF & Simon JB (2008), the right and left eigenvectors:

$$\mathcal{R} = \begin{Bmatrix} 1 & 0 & 0 & 1 & 1 \\ v_x - c_s & 0 & 0 & v_x & v_x + c_s \\ v_y & 1 & 0 & v_y & v_y \\ v_z & 0 & 1 & v_z & v_z \\ H - v_x c_s & v_y & v_z & \dfrac{|v^2|}{2} & H + v_x c_s \end{Bmatrix}$$

$$\mathcal{L} = \begin{Bmatrix} \left(\dfrac{1}{2c_s^2}\right)\left[(\gamma-1)\dfrac{|v|^2}{2} + v_x c_s\right] & -\dfrac{1}{2c_s^2}[(\gamma-1)v_x + c_s] & -\dfrac{(\gamma-1)v_y}{2c_s^2} & -\dfrac{(\gamma-1)v_z}{2c_s^2} & \dfrac{\gamma-1}{2c_s^2} \\ -v_y & 0 & 1 & 0 & 0 \\ -v_z & 0 & 0 & 1 & 0 \\ 1 - \dfrac{(\gamma-1)|v|^2}{2c_s^2} & \dfrac{(\gamma-1)v_x}{c_s^2} & \dfrac{(\gamma-1)v_y}{c_s^2} & \dfrac{(\gamma-1)v_z}{c_s^2} & -\dfrac{(\gamma-1)}{c_s^2} \\ \left(\dfrac{1}{2c_s^2}\right)\left[(\gamma-1)\dfrac{|v|^2}{2} - v_x c_s\right] & -\dfrac{1}{2c_s^2}[(\gamma-1)v_x - c_s] & -\dfrac{(\gamma-1)v_y}{2c_s^2} & -\dfrac{(\gamma-1)v_z}{2c_s^2} & \dfrac{\gamma-1}{c_s^2} \end{Bmatrix}$$

where H is the total enthalpy and γ is the adiabatic index

Time discretization

The ordinary differential equation for the discretization of the spatial operator $L(u)$:

$$\frac{du}{dt} = L(u)$$

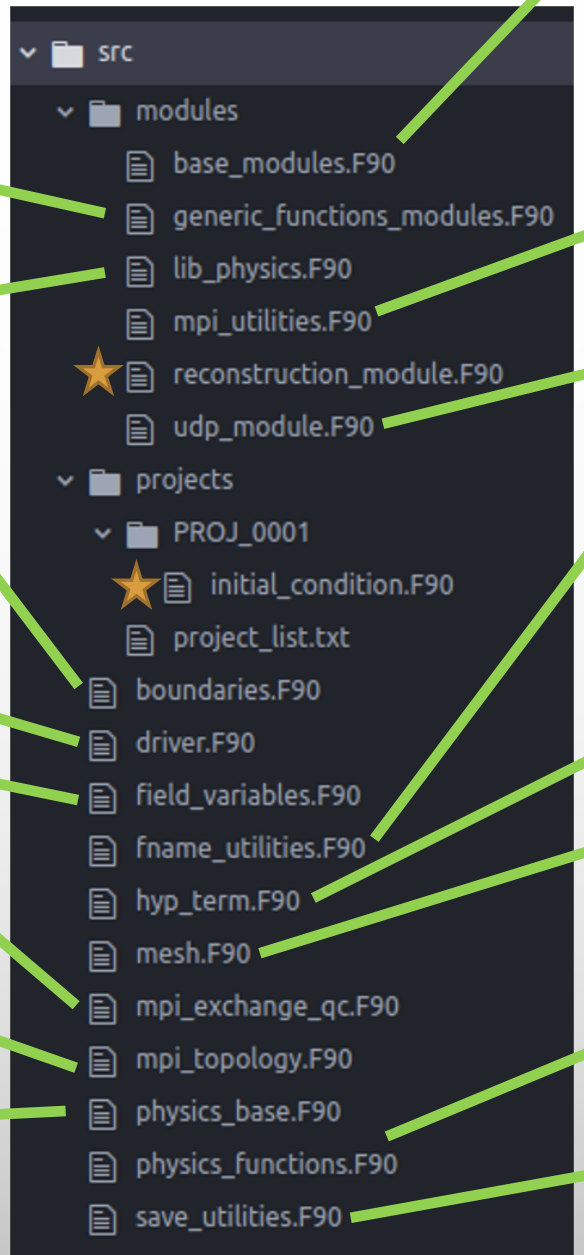
The third-order TVD Runge-Kutta scheme:

$$u^{(1)} = u^n + \Delta t L(u^n)$$

$$u^{(2)} = \frac{3}{4}u^n + \frac{1}{4}u^{(1)} + \frac{1}{4}\Delta t L(u^{(1)})$$

$$u^{(n+1)} = \frac{1}{3}u^n + \frac{2}{3}u^{(2)} + \frac{2}{3}\Delta t L(u^{(2)})$$

Code Structure



Holds arrays and flags for general code

Holds generic function pointers

Holds generic data structure for MPI_DECOMPOSITION

Holds physics specific arrays, constants and provides specific functions

Uses defined parameters, selected functions

Main driver for the code

Boundary condition

Create files and folders

Variable allocation and deallocation

Calculate $\frac{\partial F}{\partial x}$ and $\frac{\partial G}{\partial y}$ for given $\vec{u}$

Cell-centered variables exchange via MPI

Create cartesian mesh

Create the domain decomposition and neighbor list for processes

Assign physics specific functions to function parameters

Create custom string arrays for holding the names of physics variables

Output functions from the code

Implementation of WENO-Z+

reconstruction_module.F90

```fortran
!-----------------------------------------------------------------------
subroutine reconstruct_weno5_zp(dh,n_item,ui,uiph)
implicit none
real(kind=8),intent(in) :: dh
integer, intent(in) :: n_item
real(kind=8),dimension(n_item,5), intent(in) :: ui
real(kind=8),dimension(n_item), intent(out) :: uiph
real(kind=8) :: eps_zp5 = 1.0e-40
real(kind=8) :: d1=1.d+00/10.d+00, d2=3.d+00/5.d+00, d3=3.d+00/10.d+00
real(kind=8) :: um2, um1, u00, up1, up2
real(kind=8) :: beta1, beta2, beta3
real(kind=8) :: tau_zp5, lambda
real(kind=8) :: w1,w2,w3, u1,u2,u3
integer :: i
do i=1,n_item
     um2 = ui(i,1)
     um1 = ui(i,2)
     u00 = ui(i,3)
     up1 = ui(i,4)
     up2 = ui(i,5)
     beta1 = (13.d+00/12.d+00)*((um2 - 2.d+00*um1 + u00)**2) &
           + ( 1.d+00/ 4.d+00)*((um2 - 4.d+00*um1 + 3.d+00*u00)**2)
     beta2 = (13.d+00/12.d+00)*((um1 - 2.d+00*u00 + up1)**2) &
           + ( 1.d+00/ 4.d+00)*((um1 - up1)**2)
     beta3 = (13.d+00/12.d+00)*((u00 - 2.d+00*up1 + up2)**2) &
           + ( 1.d+00/ 4.d+00)*((3.d+00*u00 - 4.d+00*up1 + up2)**2)
     tau_zp5= sqrt((beta1 - beta3)*(beta1 - beta3))
     lambda = dh**(2.d+00/3.d+00)
      w1 = d1 * (1.d+00 + (((tau_zp5+eps_zp5)/(eps_zp5+beta1))**2) &
                        + lambda*(eps_zp5+beta1)/(tau_zp5+eps_zp5))
      w2 = d2 * (1.d+00 + (((tau_zp5+eps_zp5)/(eps_zp5+beta2))**2) &
                        + lambda*(eps_zp5+beta2)/(tau_zp5+eps_zp5))
      w3 = d3 * (1.d+00 + (((tau_zp5+eps_zp5)/(eps_zp5+beta3))**2) &
                        + lambda*(eps_zp5+beta3)/(tau_zp5+eps_zp5))
      u1 =  (1.d+00/3.d+00)*um2 - (7.d+00/6.d+00)*um1 + (11.d+00/6.d+00)*u00
      u2 = -(1.d+00/6.d+00)*um1 + (5.d+00/6.d+00)*u00 + ( 1.d+00/3.d+00)*up1
      u3 =  (1.d+00/3.d+00)*u00 + (5.d+00/6.d+00)*up1 - ( 1.d+00/6.d+00)*up2
      uiph(i) = (w1 * u1 + w2 * u2 + w3 * u3)/(w1 + w2 + w3)
end do
!-----------------------------------------------------------------------
end subroutine reconstruct_weno5_zp
```

Sample Initial Condition (Isentropic vortex)

```fortran
! Set initial condition for isentropic vortex
mach = 0.5d+00
alpha = 45.0d+00
beta = 5.0d+00
r2 = x*x + y*y
q(i_rho) =  (1.0d+00 - &
      (gam-1.0d+00)*(beta*beta)/(8.0*gam*pi_value*pi_value) &
      *exp(1.0d+00-r2))**(1.0d+00/(gam-1.0d+00))
q(i_vx1) =  mach*dcos(alpha*pi_value/180.d+00) &
          - beta/(2.d+00*pi_value)*y*dexp(0.5d+00*(1.d+00-r2))
q(i_vx2) =  mach*dsin(alpha*pi_value/180.d+00) &
         + beta/(2.d+00*pi_value)*x*dexp(0.5d+00*(1.d+00-r2))
q(i_vx3) =  0.0d+00
q(i_prs) =  q(i_rho)**gam
!-----------------------------------------------------------------
end subroutine initial_condition
```

initial_condition.F90

```fortran
!=================================================================
subroutine initial_condition(x, y, q)
! Initial condition as a function of x
!-----------------------------------------------------------------
use param_physics_base
!-----------------------------------------------------------------
implicit none
!-----------------------------------------------------------------
real(kind=8),intent(in)    :: x, y
real(kind=8),intent(inout) :: q(nvar)
! Local variables
real(kind=8) :: mach, alpha, beta, r2
real(kind=8),parameter :: pi_value = 4.0d+00*atan(1.0d+00)
!-----------------------------------------------------------------
```

Formulation of nonlinear weights in WENO-Z+

The nonlinear weights in WENO-Z+:

$$\omega_k^{zp} = \frac{\alpha_k^{zp}}{\sum_{q=0}^{2} \alpha_q^{zp}}, \qquad k = 1, 2, 3$$

and

$$\alpha_k^{zp} = d_k \left(1 + \left[\frac{\tau_5}{\beta_k + \epsilon} \right]^2 + \Delta x^{\frac{2}{3}} \left(\frac{\beta_k + \epsilon}{\tau_5 + \epsilon} \right) \right)$$

where $d_1 = 1/10$, $d_2 = 6/10$, $d_3 = 3/10$ are the ideal weights and $\epsilon = 10^{-40}$. The local smoothness indicators β_k and global smoothness indicator τ_5 are given as follows:

- $\beta_0 = \frac{13}{12}(f_{i-2} - 2f_{i-1} + f_i)^2 + \frac{1}{4}(f_{i-2} - 4f_{i-1} + 3f_i)^2$
- $\beta_1 = \frac{13}{12}(f_{i-1} - 2f_i + f_{i+1})^2 + \frac{1}{4}(f_{i+1} - f_{i-1})^2$
- $\beta_2 = \frac{13}{12}(f_i - 2f_{i+1} + f_{i+2})^2 + \frac{1}{4}(3f_i - 4f_{i+1} + f_{i+2})^2$
- $\tau_5 = |\beta_0 - \beta_2|$

2D isentropic vortex problem

The problem has been described by Yee, Vinokur & Djomehri (2000):

$$\mathbf{Q} = \left\{ \begin{array}{c} \rho = \left[1 - \frac{(\gamma - 1)\beta}{8\gamma\pi} e^{1-r^2}\right]^{\frac{1}{\gamma-1}} \\ \mathbf{v} = \mathbf{v}_0 + \delta\mathbf{v} \\ P_t = \rho^\gamma \end{array} \right\}$$

where the parameter $\beta = 5$ and $\gamma = 7/5$ is the adiabatic index

Initial conditions

Considering $\mathbf{v} = \mathbf{v}_0 + \delta\mathbf{v}$

$$\mathbf{v_0} = \begin{Bmatrix} v_x = 1 \\ v_y = 1 \\ v_z = 0 \end{Bmatrix}, \qquad \delta\mathbf{v} = \begin{Bmatrix} \delta v_x = \dfrac{\beta}{2\pi} e^{\frac{1-r^2}{2}} (y - y_0) \\ \delta v_y = \dfrac{\beta}{2\pi} e^{\frac{1-r^2}{2}} (x - x_0) \\ \delta v_z = 0 \end{Bmatrix}$$

- The background velocity $\mathbf{v}_0$ is non-zero
- The initial coordinate (x_0, y_0) of the vortex is at (0, 0)
- Domain defined as $-5 \leq x \leq 5$ and $-5 \leq y \leq 5$
- Periodic boundary condition
- CFL = 0.4
- Simulation time = 10

The L-p norms

$$E^N = Q^N - q^N$$

where E is the difference between the numerical solution Q and the exact solution q at time interval $N = \frac{T}{\Delta t}$

The error of the numerical scheme can then be quantified by p-norms as the following:

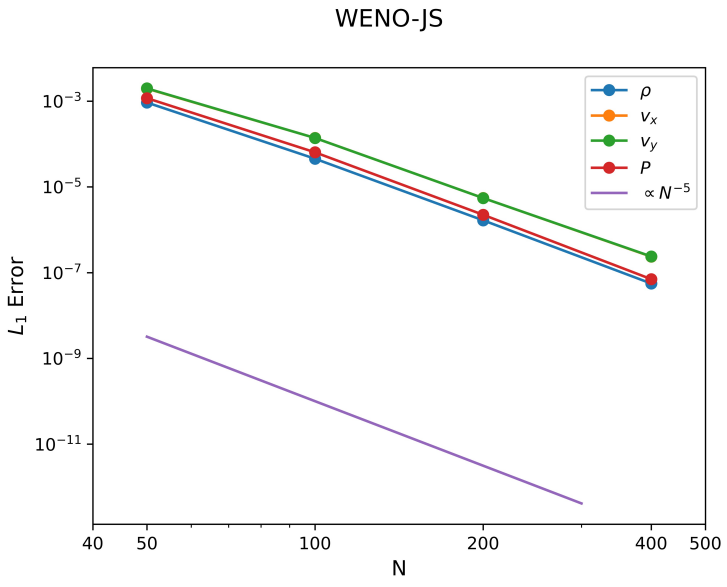
$$\|E\|_p = \left(\Delta x \sum_{i=-\infty}^{\infty} |E_i|^p\right)^{\frac{1}{p}}$$

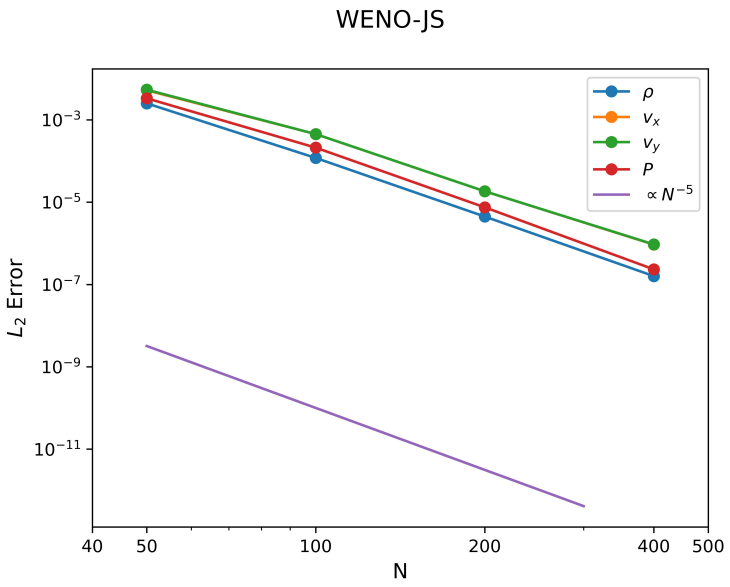
L-1 and L-2 norms correspond to $p = 1$ and 2 respectively.

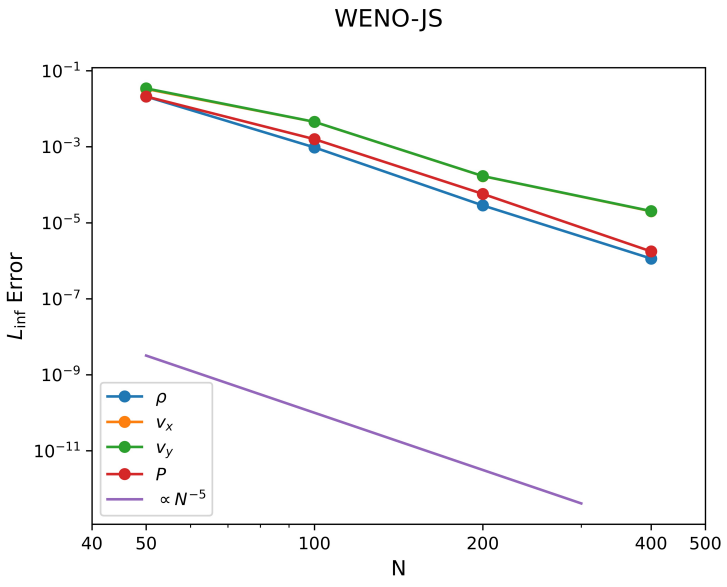
For L-∞ norm,

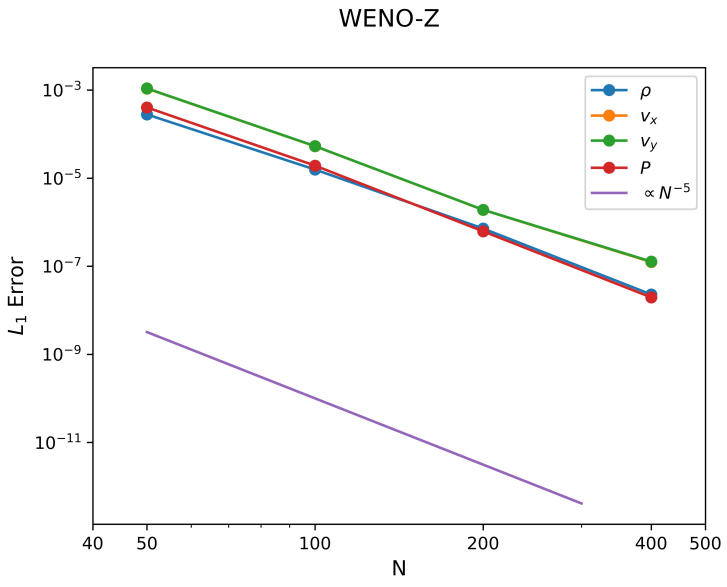
$$\|E\|_\infty = \max_{-\infty < i < \infty} |E_i|$$

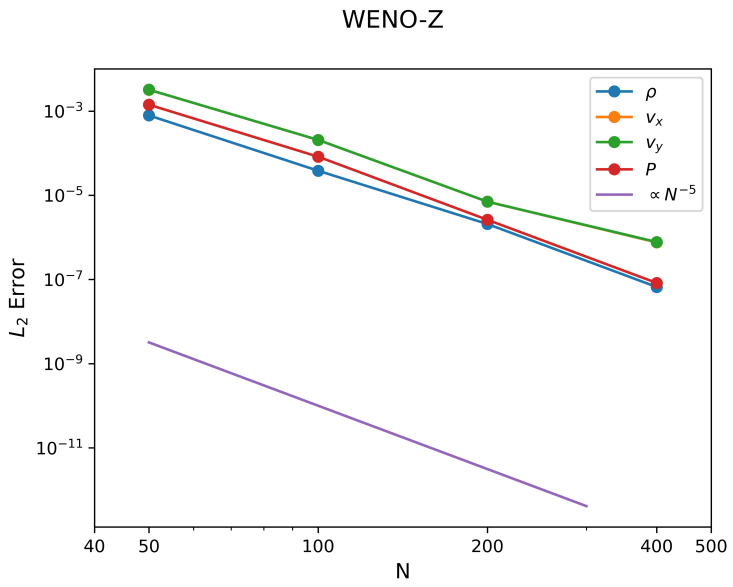
Results

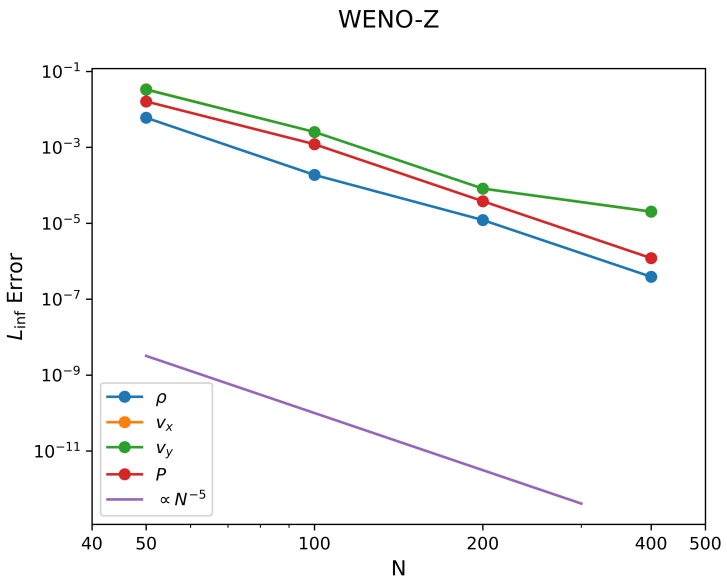

WENO-JS
L_1 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

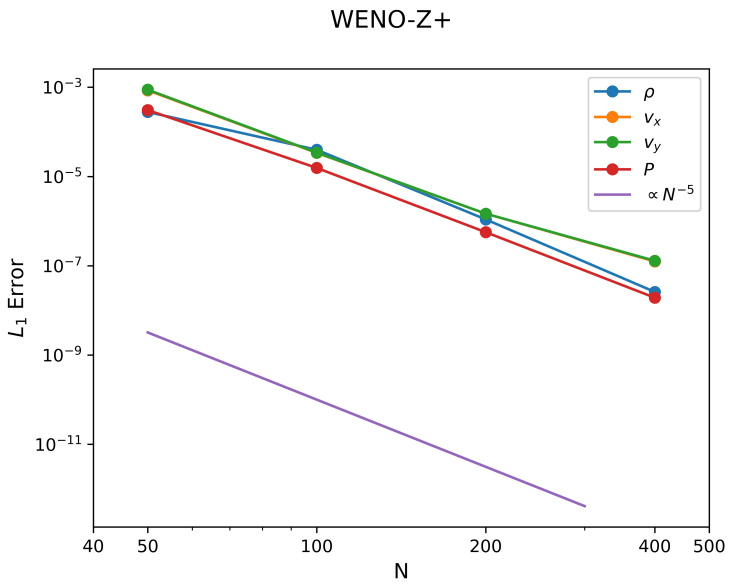

WENO-JS
L_2 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

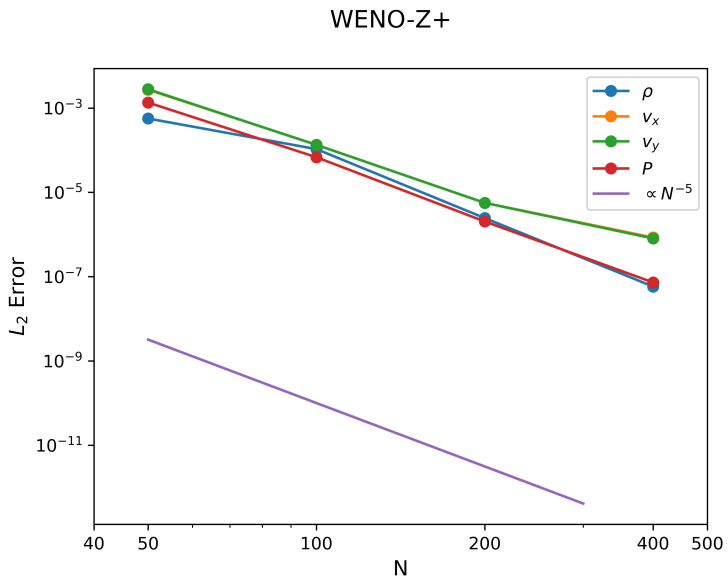

WENO-JS
L_{inf} Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

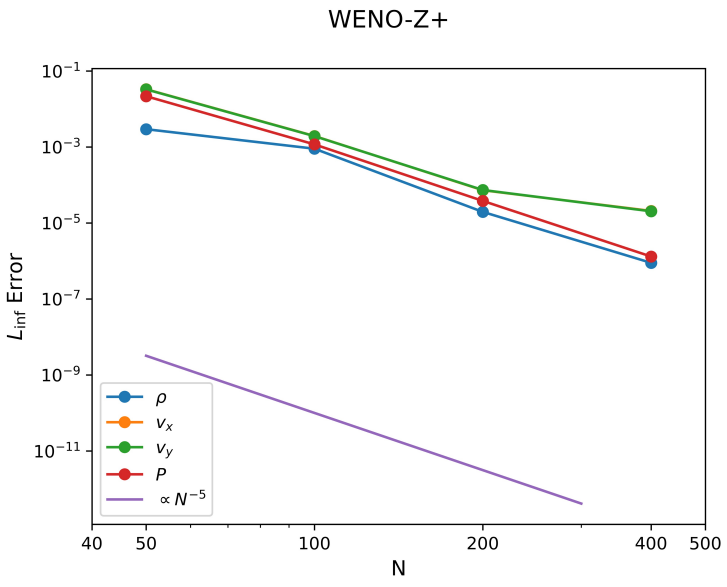

WENO-Z
L_1 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

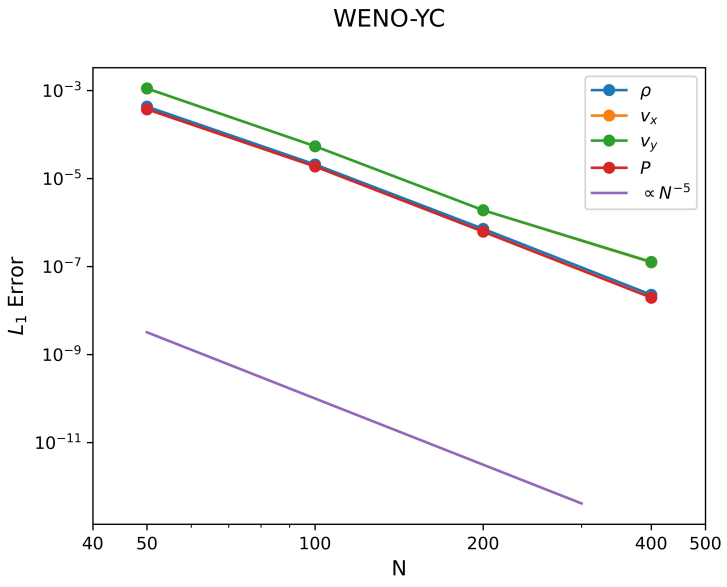

WENO-Z
L_2 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

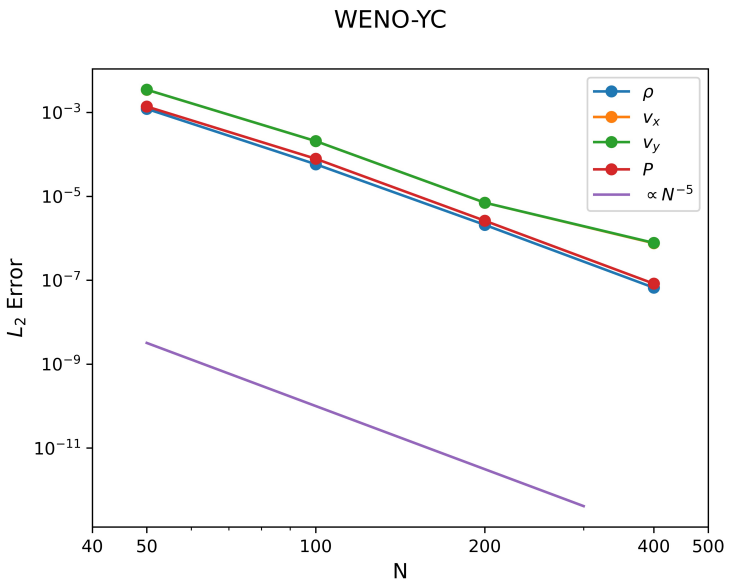

WENO-Z
L_{inf} Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

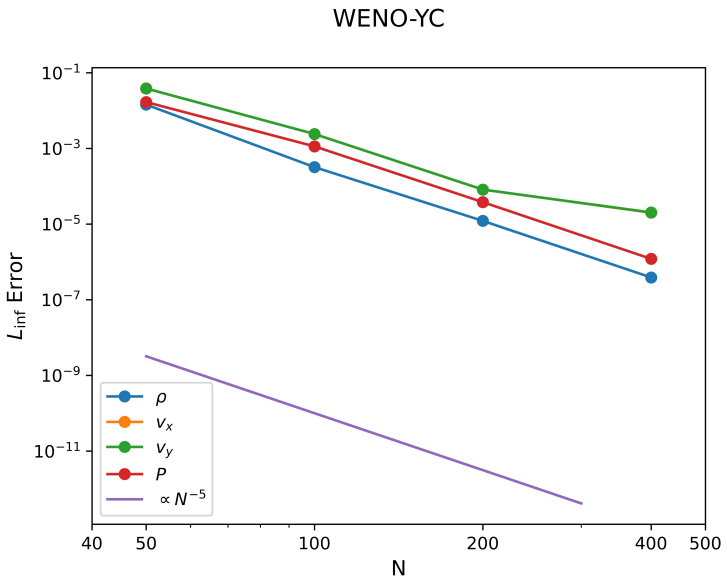

WENO-Z+
L_1 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

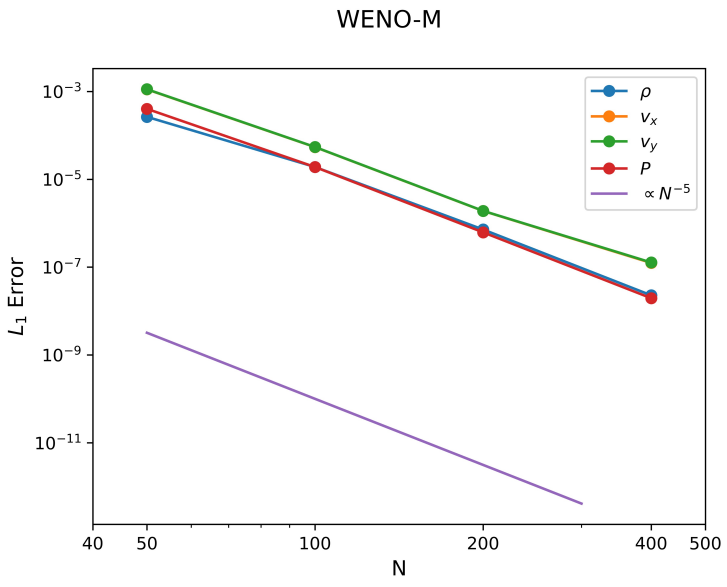

WENO-Z+
L_2 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

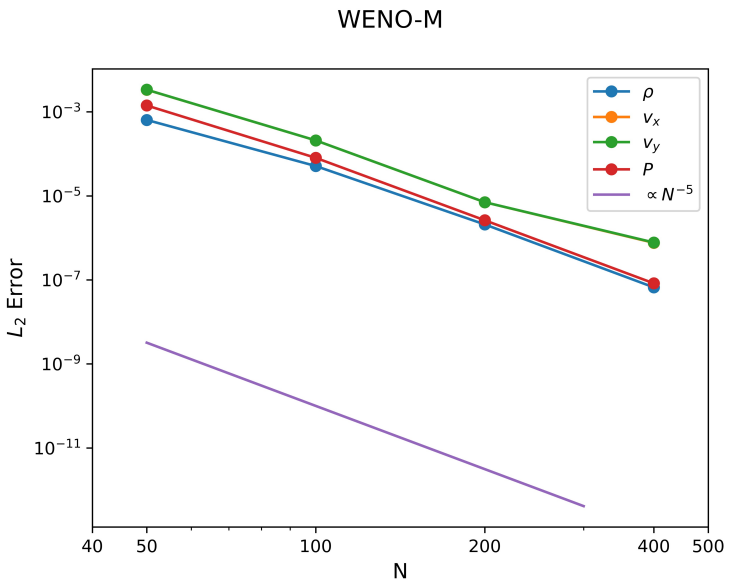

WENO-Z+
L_{inf} Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

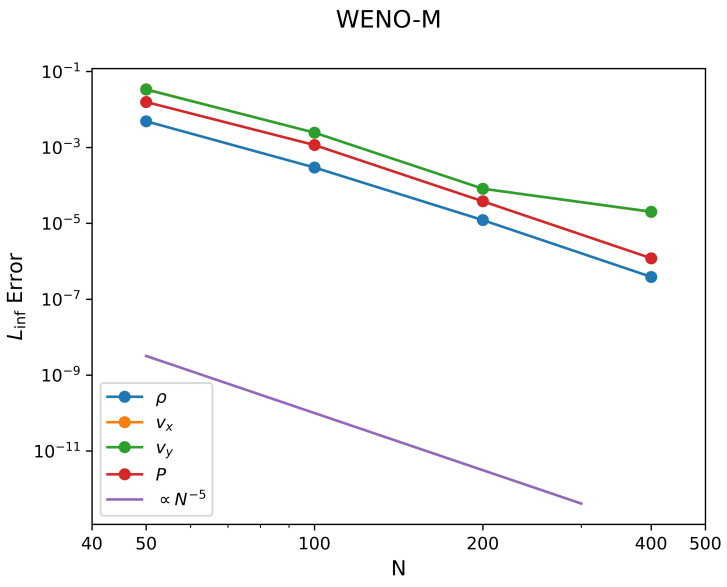

WENO-YC
L_1 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$


WENO-YC
L_2 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$


WENO-YC
L_{inf} Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$


WENO-M
L_1 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$


WENO-M
L_2 Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$


WENO-M
L_{inf} Error
N
ρ
v_x
v_y
P
$\propto N^{-5}$

Summary

- Error of the 5 WENO schemes has been quantified by the L-norms under 4 different resolutions
- All the WENO schemes are of fifth-order convergence
- WENO-YC in general has the largest L-norms error
- WENO-Z tends to have lower L-1 and L-2 error than WENO-Z+ at intermediate resolution
- WENO-Z+ has larger L- ∞ error than WENO-Z except under low resolution
- WENO-YC and WENO-M have very similar L-norm error
- WENO-M outperforms WENO-YC at low resolution

Conclusion & Future studies

- The performance of WENO-M to solve 2D Euler equations is the best among the different versions of WENO schemes

- To take the tradeoff between the performance of the WENO schemes and their computational cost into account (part of the project objectives but data has lost due to technical issues)