class data_saver.**FIEFS_Output(input_fname: str)**

Bases: object

Class for producing the desired output

Produces the desired output variables in the desired format - set in the problem input

input_fname

File name for the output file

TYPE: str

data_preferences(pin: FIEFS_Input) → None

This function is called in *FIEFS.py* and sets the data preferences specified in the problem input (pin).

PARAMETERS:

pin (*FIEFS_Input*) – Contains the problem information stored in the FIEFS_Input object

save_data(pmesh: FIEFS_Array, t: float, tmax: float, gamma: float, iter: float) → None

Saves the data to to the desired format

PARAMETERS:

- **pmesh** (*FIEFS_Array*) – FIEFS_Array mesh which contains all of the mesh information and the conserved variables, Un
- **t** (*float*) – The current time
- **tmax** (*float*) – The maximum time or the time the simulations runs until
- **gamma** (*float*) – Specific heat ratio

class input.**FIEFS_Input(input_fname: str)**

Bases: object

Class containing the input information

PARAMETERS:

input_fname (*str*) – The name of the input file

input_fname

The name of the input file

TYPE: str

value_dict

Dictionary containing the problem input information (after parsing the input file)

TYPE: dict

parse_input_file() → None

Parses and stores information from input file

Loops through parameter file and stores values from file to be used in problem generator

class mesh.**FIEFS_Array(pin: FIEFS_Input, dtype: dtype)**

Bases: object

Class which contains the mesh and conserved variables

PARAMETERS:

- **pin** (*FIEFS_Input*) – Contains the problem information stored in the FIEFS_Input object
- **dtype** (*dtype*) – Specify the dtype for the conserved variables to be stored in the PyschoArray

nvar: Number of variables to be stored

TYPE: int

nx1: Number of cells in the x1 direction

TYPE: int

nx2: Number of cells in the x2 direction

TYPE: int

ng: Number of ghost cells

TYPE: int

x1min, x2min: Min x1 and x2 values

TYPE: float

x1max, x2max: Max x1 and x2 values

TYPE: float

dx1, dx2: Step size in the x1 and x2 directions

TYPE: float

Un: Conserved variables

TYPE: ndarray[dtype]

enforce_bcs(pin: FIEFS_Input) → None

Implements the desired boundary conditions

Will enforce the boundary conditions set in the FIEFS_Input class on the conserved variables, Un.

pinFIEFS_Input

Contains the problem information stored in the FIEFS_Input object

print_value(indvar: int, indx1: int, indx2: int) → None

eos.**e_EOS(rho: Union[float, ndarray], p: Union[float, ndarray], gamma: float) → Union[float, ndarray]**

Equation of state for internal energy

PARAMETERS:

- **rho** (*Union[float, ndarray]*) – Density
- **p** (*Union[float, ndarray]*) – Pressure
- **gamma** (*float*) – Specific heat ratio

RETURNS:

Internal Energy

RETURN TYPE:

Union[float, ndarray]

eos.**p_EOS(rho: Union[float, ndarray], e: Union[float, ndarray], gamma: float) → Union[float, ndarray]**

Equation of state for pressure

PARAMETERS:

- **rho** (*Union[float, ndarray]*) – Density
- **e** (*Union[float, ndarray]*) – Internal energy
- **gamma** (*float*) – Specific heat ratio

RETURNS:

Pressure

RETURN TYPE:

Union[float, ndarray]

mesh.**get_interm_array(nvar: int, nx1: int, nx2: int, dtype: dtype) → ndarray**

Generates empty scratch array for intermediate calculations

PARAMETERS:

- **nvar** (*int*) – Number of variables
- **nx1** (*int*) – Number of cells in the x1 direction
- **nx2** (*int*) – Number of cells in the x2 direction
- **dtype** (*dtype*) – Type for the values to have in the scratch array

RETURNS:

Scratch array with provided dtype

RETURN TYPE:

ndarray

reconstruct.**get_limited_slopes(U_i_j: ndarray, U_ip1_j: ndarray, U_im1_j: ndarray, U_i_jp1: ndarray, U_i_jm1: ndarray, beta: float)**

Minmod slope limiter to handle discontinuities

Minimod slope limiter based on page 508 in [1], necessary for handling discontinuities

PARAMETERS:

- **U_i_j** (*ndarray[float]*) – Values of U not shifted by any index
- **U_ip1_j** (*ndarray[float]*) – Values of U shifted by i+1
- **U_im1_j** (*ndarray[float]*) – Values of U shifted by i-1
- **U_i_jp1** (*ndarray[float]*) – Values of U shifted by j+1
- **U_i_jm1** (*ndarray[float]*) – Values of U shifted by j-1
- **beta** (*float*) – Weight value determining type of limiter. Default value is 1.0 which represents a minmod limiter

RETURNS:

- **delta_i** (*ndarray[float]*) – Slope of the conserved variables in the x-direction
- **delta_j** (*ndarray[float]*) – Slope of the conserved variables in the y-direction

REFERENCES

[1] Toro, E. F. (2011). Riemann solvers and Numerical Methods for fluid dynamics: A practical introduction. Springer.

reconstruct.**get_unlimited_slopes(U_i_j: ndarray, U_ip1_j: ndarray, U_im1_j: ndarray, U_i_jp1: ndarray, U_i_jm1: ndarray, w: float)**

Find the slopes between grid cells without a limiter. Can potentially lead to oscillations if there are discontinuities present in flow.

PARAMETERS:

- **U_i_j** (*ndarray[float]*) – Values of U not shifted by any index
- **U_ip1_j** (*ndarray[float]*) – Values of U shifted by i+1
- **U_im1_j** (*ndarray[float]*) – Values of U shifted by i-1
- **U_i_jp1** (*ndarray[float]*) – Values of U shifted by j+1
- **U_i_jm1** (*ndarray[float]*) – Values of U shifted by j-1
- **w** (*float*) – Weight parameter determining whether slopes are centered or biased in a particular direction

RETURNS:

- **delta_i** (*ndarray[float]*) – Slope of the conserved variables in the x-direction
- **delta_j** (*ndarray[float]*) – Slope of the conserved variables in the y-direction

REFERENCES

[1] Toro, E. F. (2011). Riemann solvers and Numerical Methods for fluid dynamics: A practical introduction. Springer.

riemann.**solve_riemann(U_l: ndarray, U_r: ndarray, gamma: float, direction: str) → ndarray**

Solve the Riemann problem

Solves the Riemann problem using a HLLC Riemann solver - outlined in Toro adapted from page 322 (see [1])

PARAMETERS:

- **U_l** (*ndarray[float]*) – Conserved variables at the left cell face
- **U_r** (*ndarray[float]*) – Conserved variables at the right cell face
- **gamma** (*float*) – Specific heat ratio
- **direction** (*str*) – Specify the 'x' or 'y' direction

RETURNS:

F – The flux in the specified direction returned from the Riemann problem

RETURN TYPE:

ndarray[float]

tools.**calculate_timestep(pmesh: FIEFS_Array, cfl: float, gamma: float) → float**

Calculates the maximum timestep allowed for a given CFL to remain stable

PARAMETERS:

- **pmesh** (*FIEFS_Array*) – FIEFS_Array mesh which contains all of the current mesh information and the conserved variables Un
- **cfl** (*float*) – Courant-Freidrichs-Lewy condition necessary for stability when choosing the timestep
- **gamma** (*float*) – Specific heat ratio

RETURNS:

The calculated timestep for the provided conditions

RETURN TYPE:

float

tools.**get_fluxes_1d(Un: ndarray, gamma: float, direction: str) → ndarray**

Returns fluxes provided the conserved variables, Un, at a point

This function returns the fluxes in the x or y direction at one specified cell provided the conserved variables, Un, at the specified cell

PARAMETERS:

- **Un** (*ndarray[float]*) – Conserved variables
- **gamma** (*float*) – Specific heat ratio
- **direction** (*str*) – Specify the 'x' or 'y' direction

RETURNS:

F – The computed flux vector at one cell in the specified direction

RETURN TYPE:

ndarray[float]

tools.**get_fluxes_2d(Un: ndarray, gamma: float, direction: str) → ndarray**

Returns fluxes provided the conserved variables, Un, for all cells

This function returns the fluxes in the x or y direction for all cells provided the conserved variables, Un.

PARAMETERS:

- **Un** (*ndarray[float]*) – Conserved variables
- **gamma** (*float*) – Specific heat ratio
- **direction** (*str*) – Specify the 'x' or 'y' direction

RETURNS:

F – The computed flux vector at all cells in the specified direction

RETURN TYPE:

ndarray[float]

tools.**get_primitive_variables_1d(Un: ndarray, gamma: float)**

Returns the primitive variables at a point provided Un.

This function returns the primitive variables at a single point provided the conserved variables Un are provided at the same point.

PARAMETERS:

- **Un** (*ndarray[float]*) – Conserved variables
- **gamma** (*float*) – Specific heat ratio

RETURNS:

- **rho** (*float*) – Density
- **u** (*float*) – Horizontal velocity
- **v** (*float*) – Vertical velocity
- **p** (*float*) – Pressure

tools.**get_primitive_variables_2d(Un: ndarray, gamma: float)**

Returns the primitive variables for all points provided Un.

This function returns the primitive variables for all points provided the conserved variables Un are provided.

PARAMETERS:

- **Un** (*ndarray[float]*) – Conserved variables
- **gamma** (*float*) – Specific heat ratio

RETURNS:

- **rho** (*ndarray[float]*) – Density
- **u** (*ndarray[float]*) – Horizontal velocity
- **v** (*ndarray[float]*) – Vertical velocity
- **p** (*ndarray[float]*) – Pressure

sample.**sampleProblemGenerator(pin: FIEFS_Input, pmesh: FIEFS_Array) → None**

Generates the problem in by inputting the information to the problem mesh
This function is called in *main.py* and sets the initial conditions specified in the problem input (pin) onto the problem mesh (pmesh).
Needs to exist for each problem type in order for everything to work.
PARAMETERS:

- **pin** (*FIEFS_Input*) – Contains the problem information stored in the FIEFS_Input object
- **pmesh** (*FIEFS_Array*) – FIEFS_Array mesh which contains all of the current mesh information and the conserved variables Un

kh.**ProblemGenerator(pin: FIEFS_Input, pmesh: FIEFS_Array) → None**

Generates the problem in by inputting the information to the problem mesh

This function is called in *main.py* and sets the initial conditions specified in the problem input (pin) onto the problem mesh (pmesh).

Needs to exist for each problem type in order for everything to work.

PARAMETERS:

- **pin** (*FIEFS_Input*) – Contains the problem information stored in the FIEFS_Input object
- **pmesh** (*FIEFS_Array*) – FIEFS_Array mesh which contains all of the current mesh information and the conserved variables Un

class plotter.**Plotter(pmesh: FIEFS_Array)**

Bases: object

Creates plots for the desired variables

PARAMETERS:

pmesh (*FIEFS_Array*) – FIEFS_Array mesh which contains all of the current mesh information and the conserved variables Un

ng: Number of ghost cells

TYPE: int

rho: Array containing the rho (density) value at each point in space at a given time

TYPE: ndarray[float]

u: Array containing the u (horizontal velocity) value at each point in space at a given time

TYPE: ndarray[float]

v: Array continaing the v (vertical velocity) value at each point in space at a given time

TYPE: ndarray[float]

et: Array containing the et (total energy) value at each point in space at a given time

TYPE: ndarray[float]

primitives: Dictionary containing each primitive (rho, u, v, et) with a corresponding key

TYPE: dict

x1: Vector containing the x1 values

TYPE: ndarray[float]

x2: Vector containing the x2 values

TYPE: ndarray[float]

x1_plot: Array of x1 values after using meshrid for plotting

TYPE: ndarray[float]

x2_plot: Array of x2 values after using meshgrid for plotting

TYPE: ndarray[float]

check_path_exists() → None

Checks output path and creates it if necessary

create_plot(variables_to_plot: list[str], labels: list[str], cmaps: list[str], stability_name: str, style_mode: bool, iter: int, time: float) → None

Creates desired plots for provided input variables at a given time

PARAMETERS:

- **variables_to_plot** (*list[str]*) – A list of strings containing the variables to be plotted
- **labels** (*list[str]*) – The corresponding labels for the variables (can be in LaTeX form)
- **cmaps** (*list[str]*) – The desired cmaps for each variable - entered in the corresponding order
- **stability_name** (*str*) – The name of the instability
- **style_mode** (*bool*) – Style_mode will turn off all extra text and display only the desired variable in full in the figure
- **iter** (*int*) – The current iteration
- **time** (*float*) – The current time