15/07/2021

NeuraSim MANUAL

Ekhi Ajuria, Antonio Giménez Nadal

Institut Supérieur de l'Aéronautique et de l'Espace
SUPAERO

Index

- Fluid Solver
- Deep Learning
- Code Structure
- Folder Structure
- Configuration Files
- Command Interface

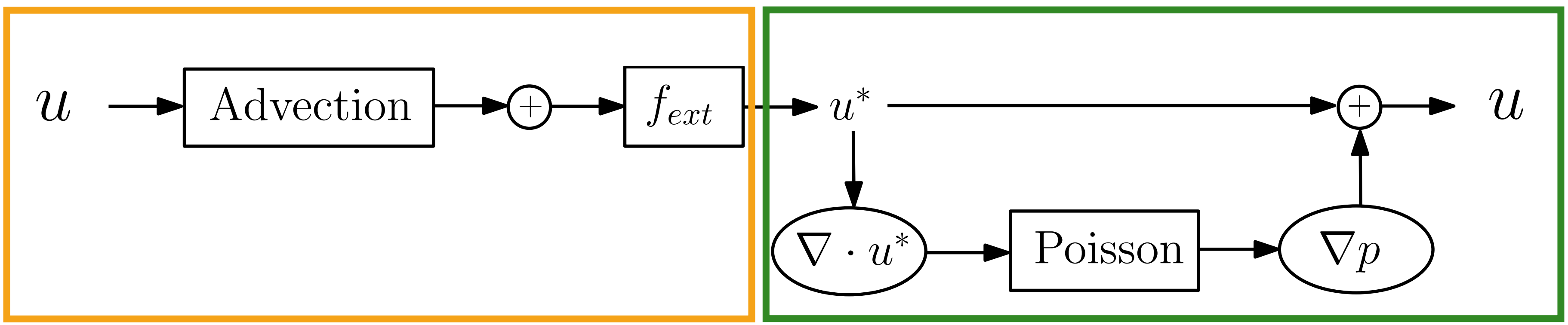
- Field Operate Functions
- Analyze Functions
- Plot Functions

Fluid Solver

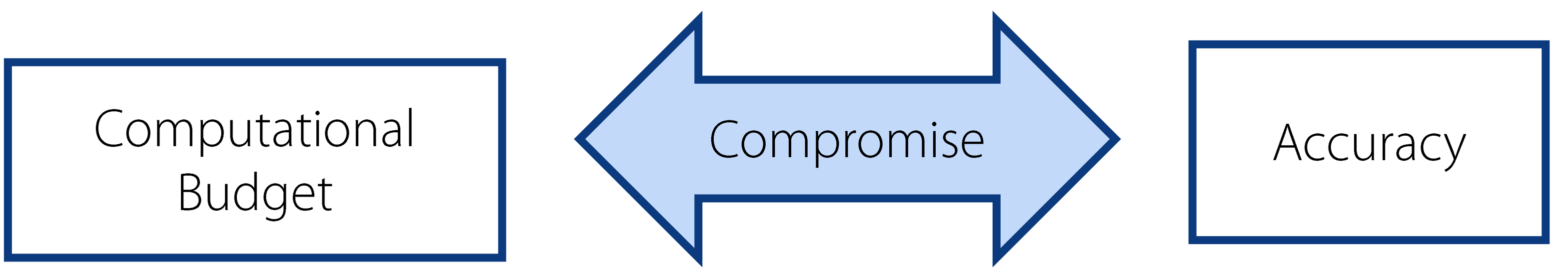
NeuraSim Manual

- Incompressible Navier Stokes
- Based on PhiFlow (python version of Mantaflow (Thuerey,2016))
  - Computer-vision orientated
- Two main steps:
  - Advection
  - Pressure projection

$$\begin{cases} \dfrac{\partial u}{\partial t} = -u \cdot \nabla u - \dfrac{1}{\rho} \nabla p + f \\ \nabla \cdot u = 0 \end{cases}$$



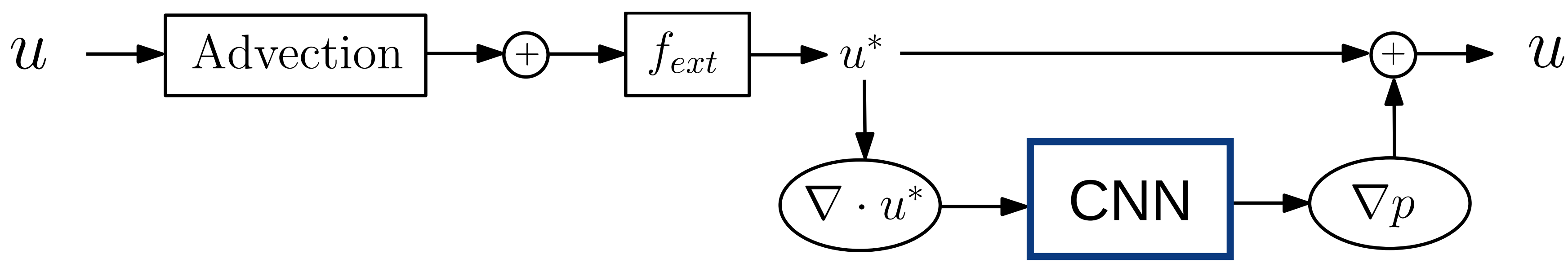
- **Resolution** methods for Eulerian approach of the **Poisson Step** (iterative methods):
  - **Jacobi** methods
    - $\uparrow$ iterations = $\uparrow$ accuracy = $\uparrow$ computational cost
  - Preconditioned **Conjugate Gradient** (PCG)
- <u>It takes up to 80% of the calculation time!</u>



Deep Learning

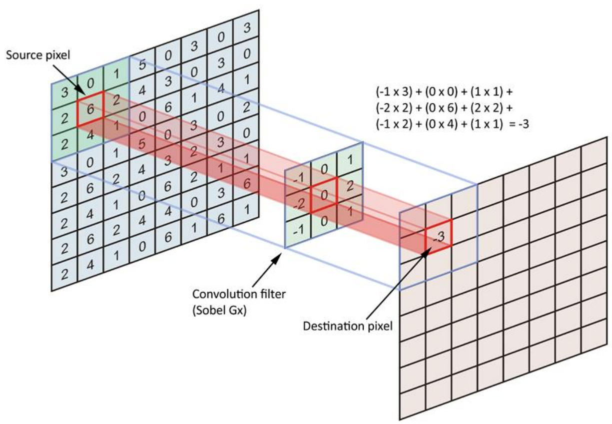
NeuraSim Manual

- Instead of using those methods, solve the **Poisson step** using **Convolutional Neural Networks** (following Tompson's work)

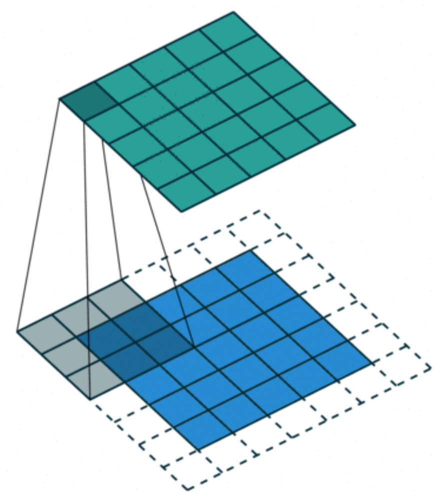


NeuraSim Manual

- Convolutional layers:
  - For each input we apply a series of filters of “small” size (typically kernels of 3x3)
  - These filters perform a linear operation, sliding through the whole domain



https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

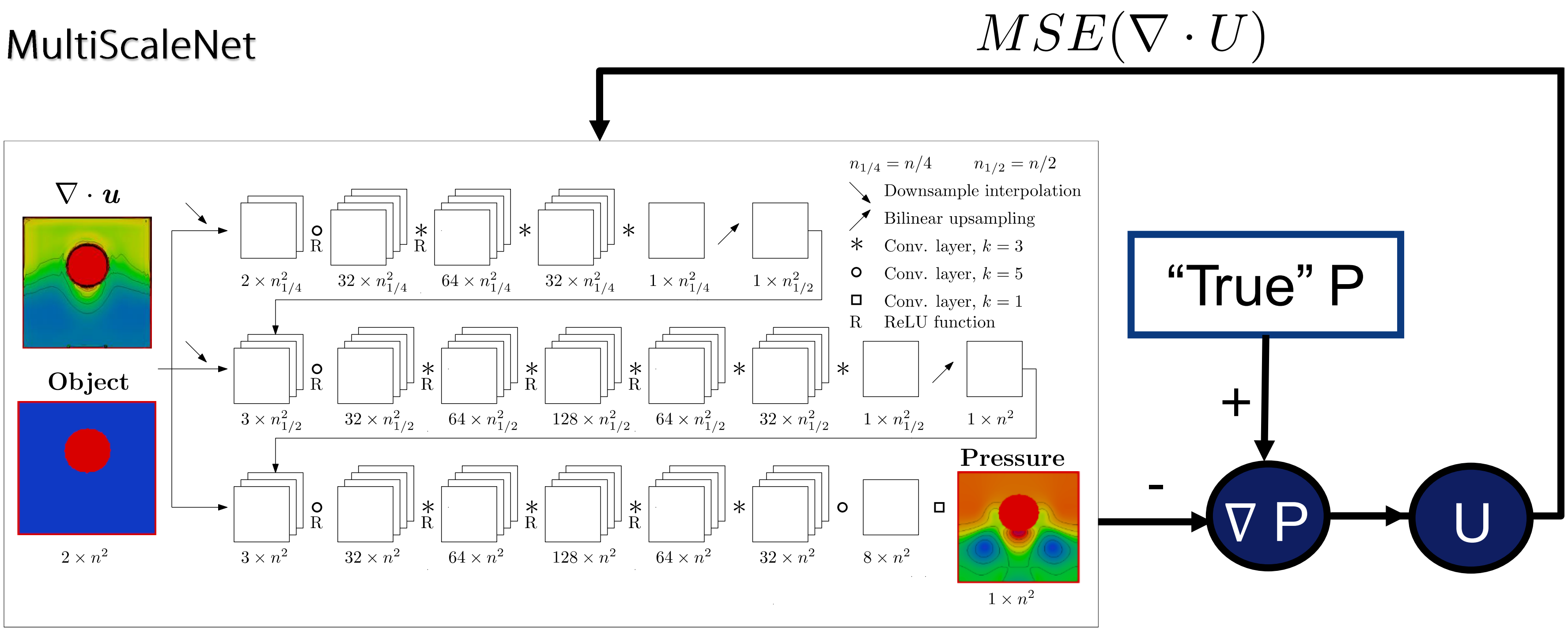


https://github.com/vdumoulin/conv_arithmetic

NeuraSim Manual

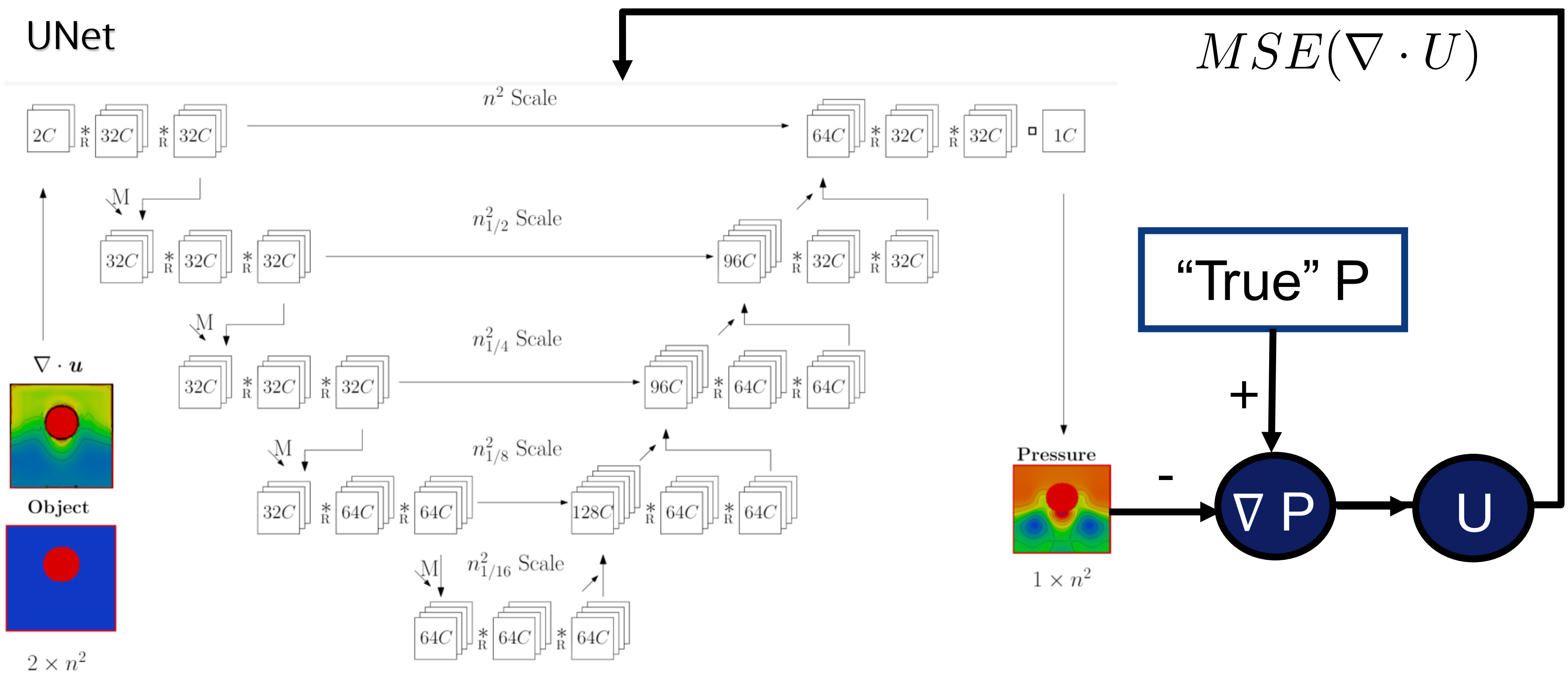
- The specific architecture chosen is:
  - **MultiScaleNet** (Mathieu, 2015) or **UNet** (Ronnenberger, 2015)
  - Account for the **physical loss**

MultiScaleNet



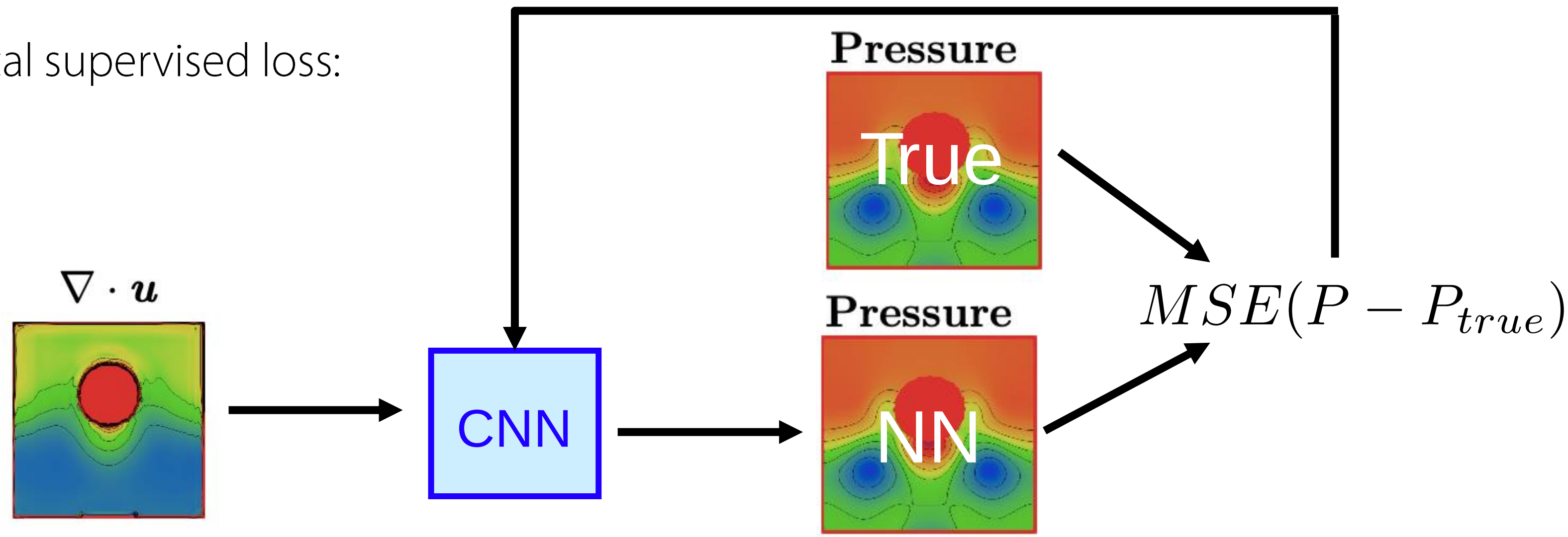
NeuraSim Manual

- The specific architecture chosen is:
  - **MultiScaleNet** (Mathieu, 2015) or **UNet** (Ronnenberger, 2015)
  - Account for the **physical loss**

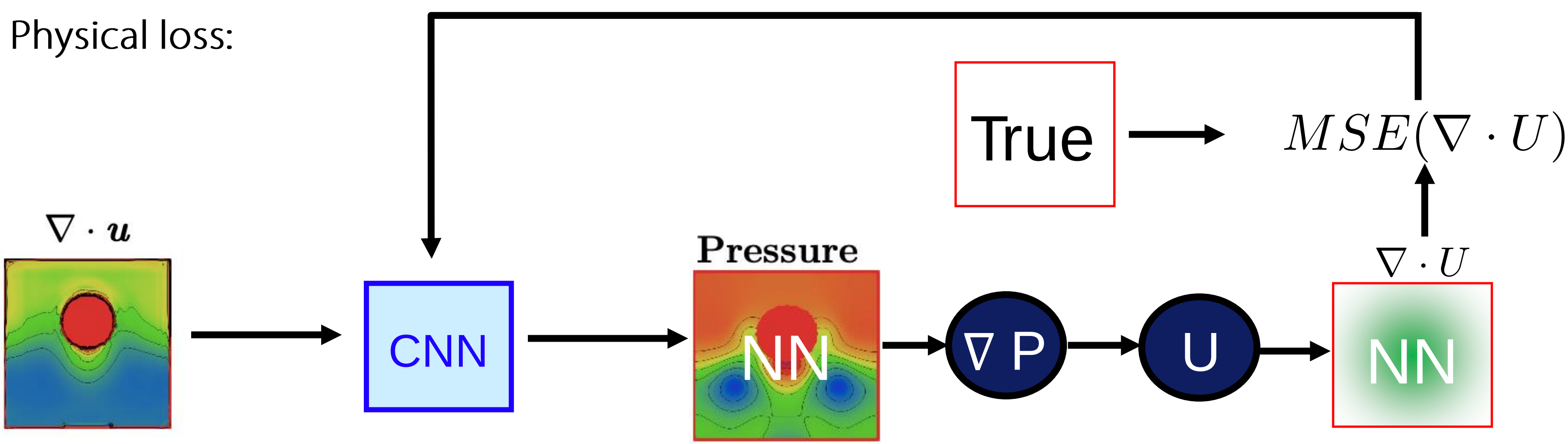


NeuraSim Manual

- Typical supervised loss:

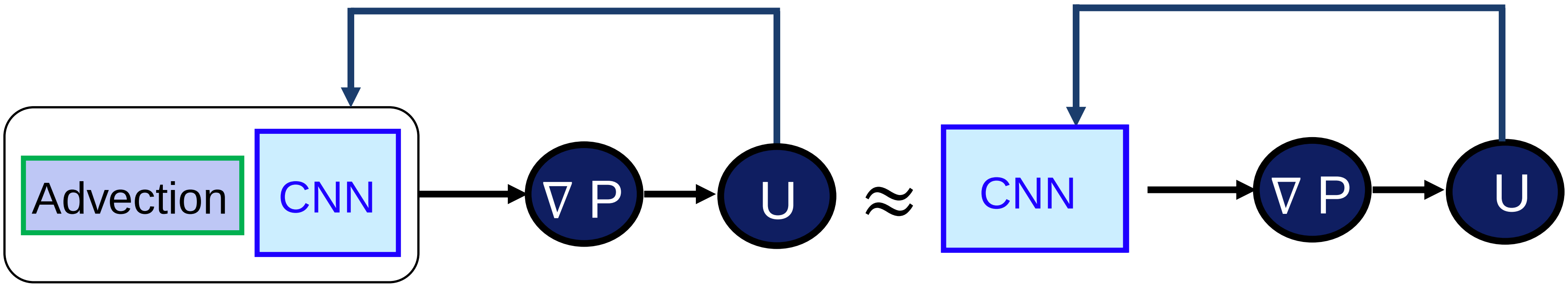


- Physical loss:

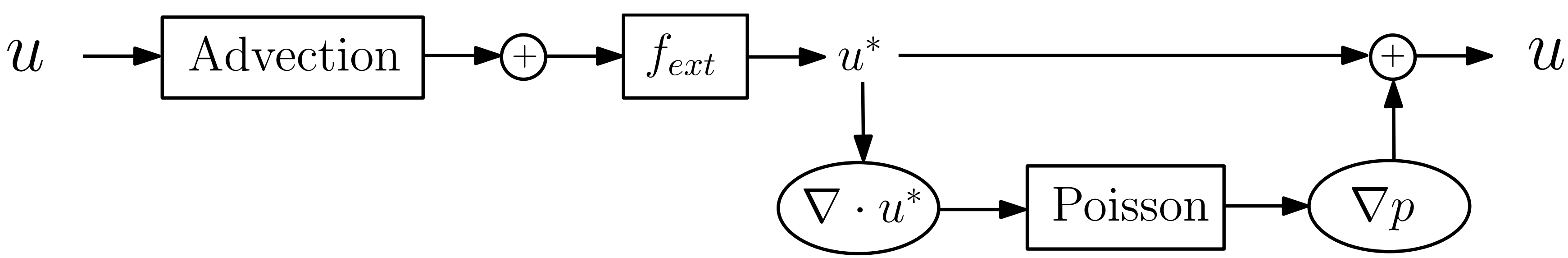


NeuraSim Manual

- Initially, network can be **treated independently to solver during training**

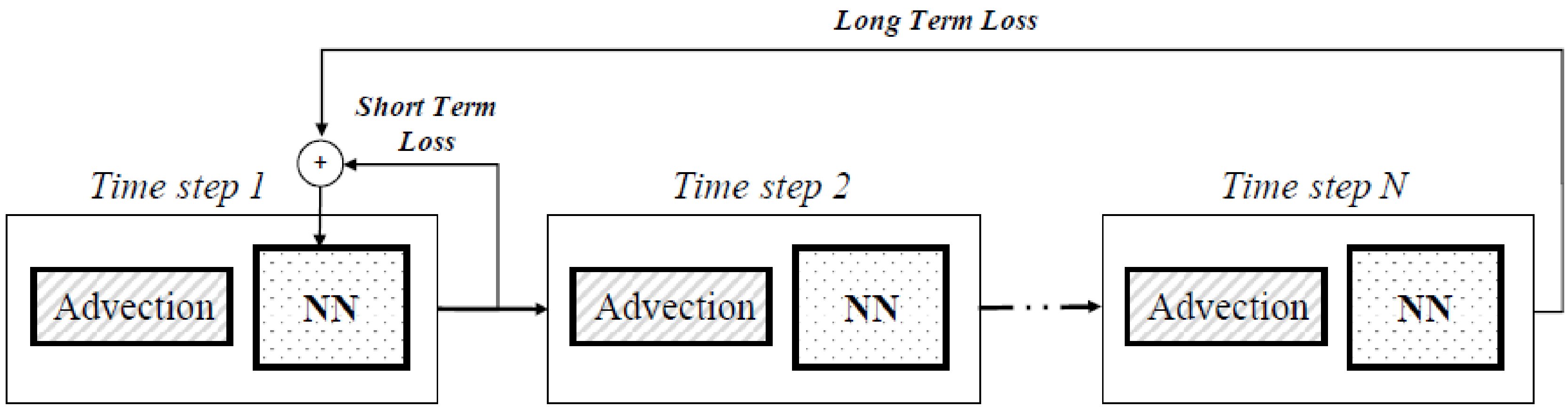


- During testing, or inference, use the regular configuration



NeuraSim Manual

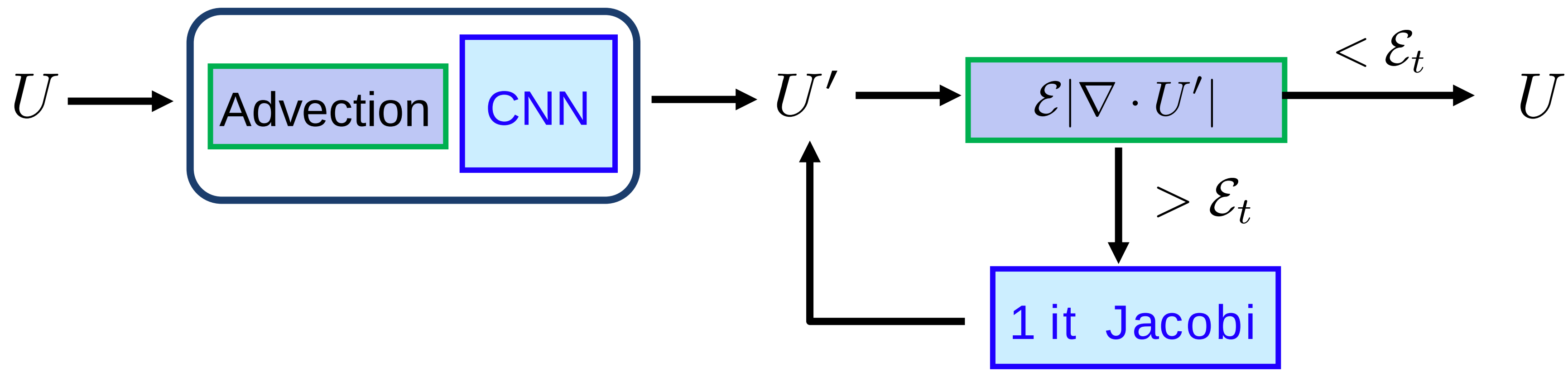
- To **reduce** the **error** of the CNN, we use the **Short and Long term loss**:



- Each being the physical loss: $MSE(\nabla \cdot U)$

NeuraSim Manual

- How can an AI-based solver be **robust** and **reliable**?
  - Improve the learning (loss, architecture, etc)
  - Bigger database
  - New **hybrid CFD-AI strategy** -> Ensures **reliability**

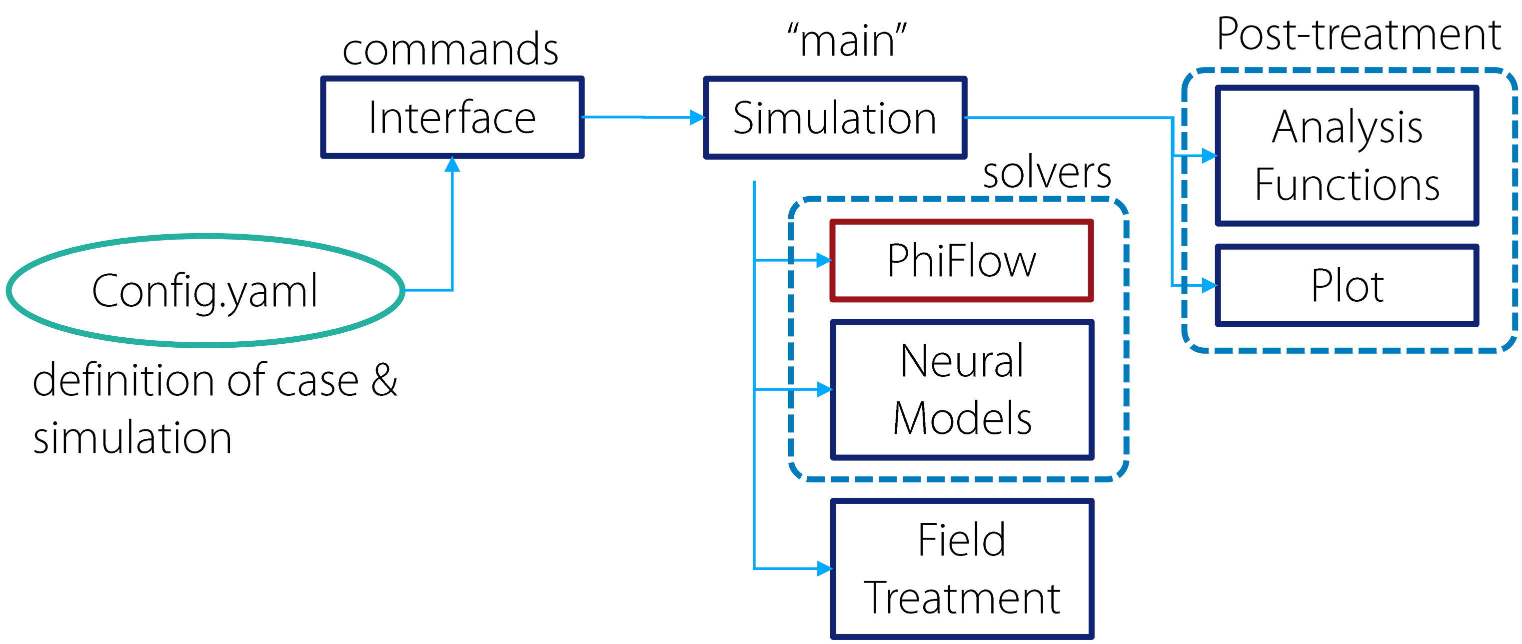


Code Structure

NeuraSim Manual

- In house-made software base on PhiFlow
- Characteristics:
  - Python based
  - GPU accelerated using torch tensors + differentiable
  - Works around PhiFlow types. Mainly StaggeredGrid and CenteredGrid

- Structure of the code:



Folder Structure

NeuraSim Manual

- Main Folders Structure

  - cases/ -> config_simulation.yaml + launcher.slurm
  - trainings/ -> saved models
  - neurasim/
    - analysis/ -> análisis functions
    - doc/ -> documentation (under construction)
    - engines/phi/ -> phiflow
    - interface/ -> commands.py, parsers, I/O files, etc
    - neural_models/ -> training, architecture classes, etc
    - simulation/ -> VonKarman.py, etc
    - util/
      - operations/ -> field operations
      - plot/ -> plot field, distribution, GIF, etc
      - unit_test/ -> if some unit test of future utility put here on its folder
      - temp/ -> temporal files

Configuration Files

NeuraSim Manual

- config_simulation.yaml

Type of Simulation Class

```yaml
####################
#    SIMULATION    #
####################
simClass: VonKarman_rotative
GPU: True
save_field: False
sim_method: PHI
```

solver

```yaml
#in_dir: './'
out_dir: './results_pre_1e-3/'

####################
#  DISCRETIZATION  #
####################
Nx: 448 #[] number of control volumes in x direction
Ny: 448 #[] number of control volumes in y direction
Nt: 15000  #[] number of time steps to simulate

# CFL
CFL: 0.2
```

```yaml
####################
#     GEOMETRY     #
####################
#Domain
Lx: 224 #[m]  or in mm if consistently changed
Ly: 224 #[m]

#BC
BC_domain_x: OPEN
BC_domain_y: STICKY

#Cilinder
D: 15
xD: 75

####################
# PHYSICAL FORCES  #
####################
Reynolds: 100.0
Alpha: 1.5
```

NeuraSim Manual

- Packaging of a meta simulation/training caller used for the iterate command.

```
EXECUTE = 'rotating_cylinder.py'
```

Accepts function or script

```
#CONSTANT CASE PARAMETERS
CONSTANT = [['Lx', 150],
            ['Ly', 150],
            ['Nt', 300]
        ]
```

Passed as parameters if function, or as CLI arguments if script

```
#ITERABLE CASE PARAMETERS
ITERABLE=[ ['A', [0, 2]],
           ['Nx', [150,450]],
           ['Re', [200,300,500]] ,
        ]
```

Accepts list, arrays, and range types

- Packaging of a meta simulation/training caller used for the iterate command.

```
#ITERABLE CASE PARAMETERS
ITERABLE=[ ['A', [0, 2]],
           ['Nx', [150,450]],
           ['Re', [200,300,500]] ,
         ]
```

Notice it is not a trivial problem since it consist of an undetermined number of nested for loops inside the previous ones. And managing the corresponding indexes, etc.

The solution adopted consist of using a recursive algorithm to create each combination possible.

Command Interface

- Interface commands

Simulate

*Usage: simulate [-key <arg>] [–long_key <arg>]

Argument Description	**Usage (Short)**	**Usage (long)**	**Default Value**
Path of the configuration yaml files directory	-cd <'path'>	–conf_dir <'path'>	'./'
Path of the output results files directory	-co <'path'>	–out_dir <'path'>	'./results/'
Path of the input results files directory	-id <'path'>	–in_dir <'path'>	'./results/'

- Interface commands

Analyze

*Usage: analyze [-key <arg>] [–long_key <arg>]

Argument Description	**Usage (Short)**	**Usage (long)**	**Default Value**
Path of the configuration yaml files directory	-cd <'path'>	–conf_dir <'path'>	'./'
Path of the output results files directory	-co <'path'>	–out_dir <'path'>	'./results/'
Path of the input results files directory	-id <'path'>	–in_dir <'path'>	'./results/'
Type of analysis to perform. Append as many as needed	-a <type>	–analysis_type <type>	–

- Interface commands

Train

*Usage: train [-key <arg>] [–long_key <arg>]

Argument Description	Usage (Short)	Usage (long)	Default Value
Path of the configuration yaml files directory	-cd <’path’>	–conf_dir <’path’>	‘./’
Path of the output results files directory	-co <’path’>	–out_dir <’path’>	‘./results/’
Path of the input results files directory	-id <’path’>	–in_dir <’path’>	‘./results/’

NeuraSim Manual

- Interface commands

Iterate

*Usage: iterate [-key <arg>] [–long_key <arg>]

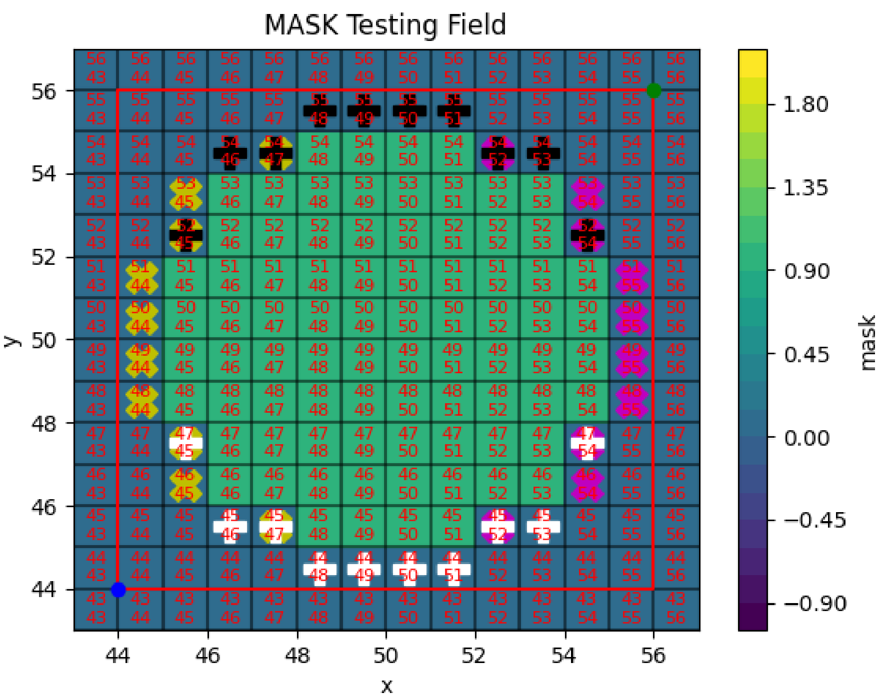
Argument Description	Usage (Short)	Usage (long)	Default Value
Path of the configuration yaml files directory	-cd <’path’>	–conf_dir <’path’>	‘./’
Path of the output results files directory	-co <’path’>	–out_dir <’path’>	‘./results/’
Path of the input results files directory	-id <’path’>	–in_dir <’path’>	‘./results/’
Callable(function, command) or script to iterate over	-e <callable>	–execute <callable>	simulate

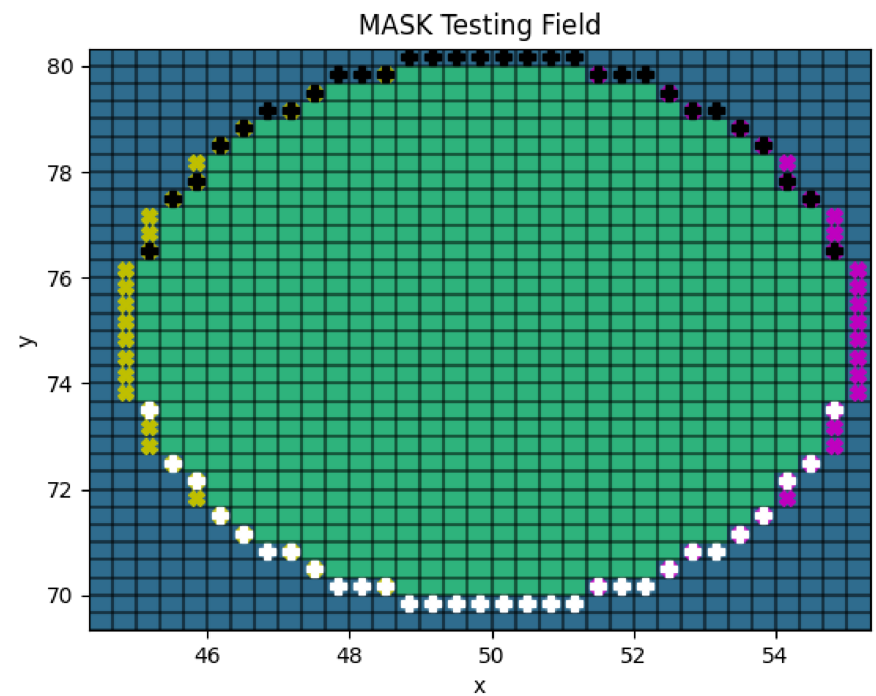
*Example: iterate -e “analyze -a velocity”

Field Operate Functions

NeuraSim Manual

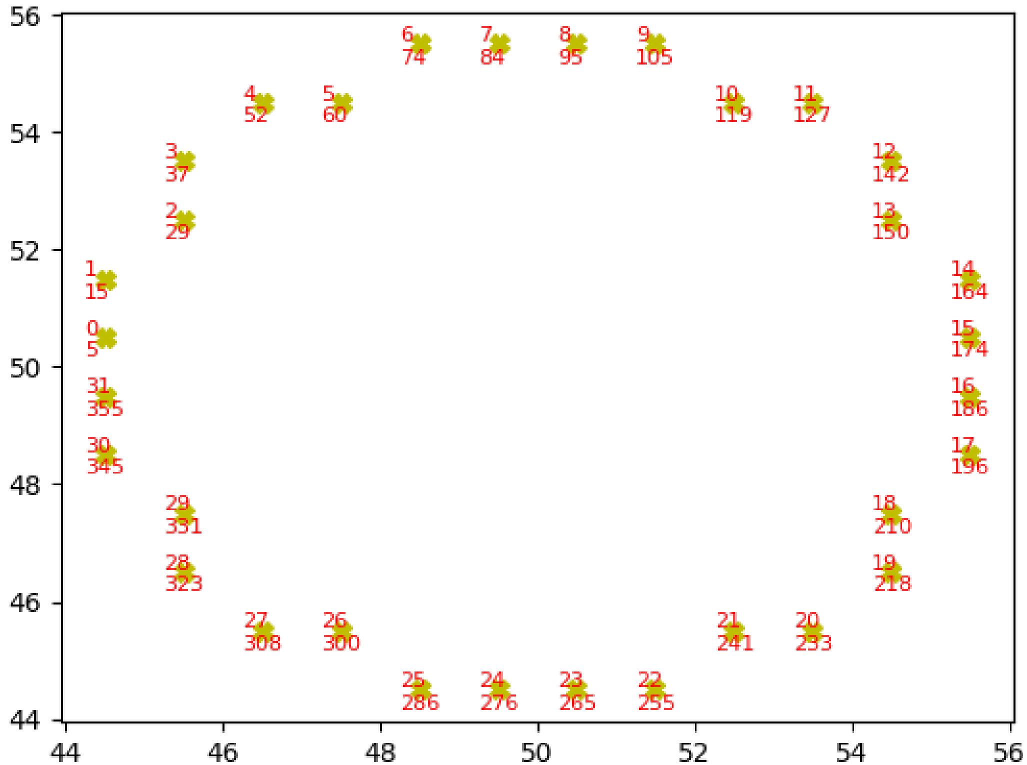
- get_exterior_edges(object_mask)





NeuraSim Manual

- get_line_distribution(object_mask=None, edges=None)

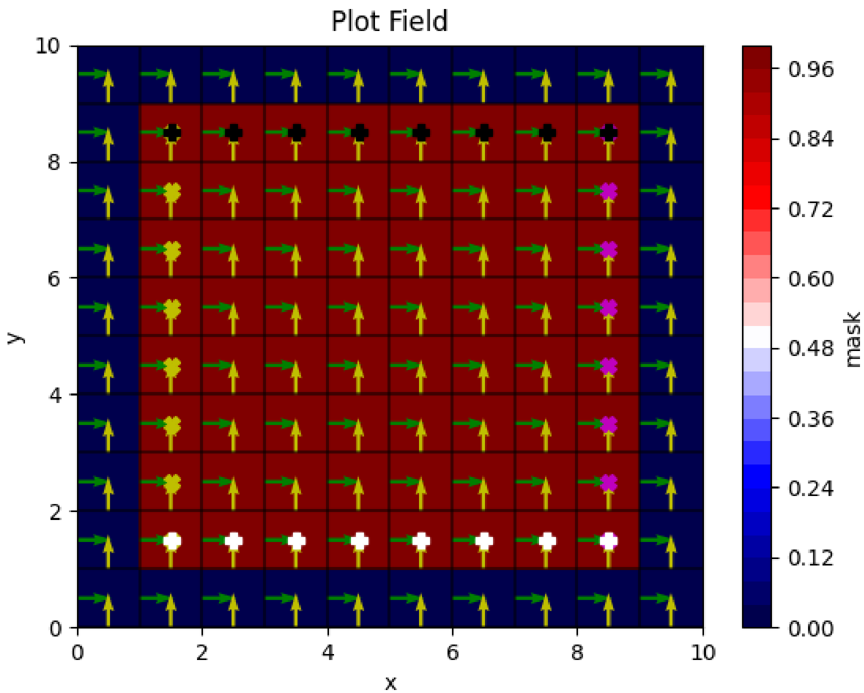


node
degrees

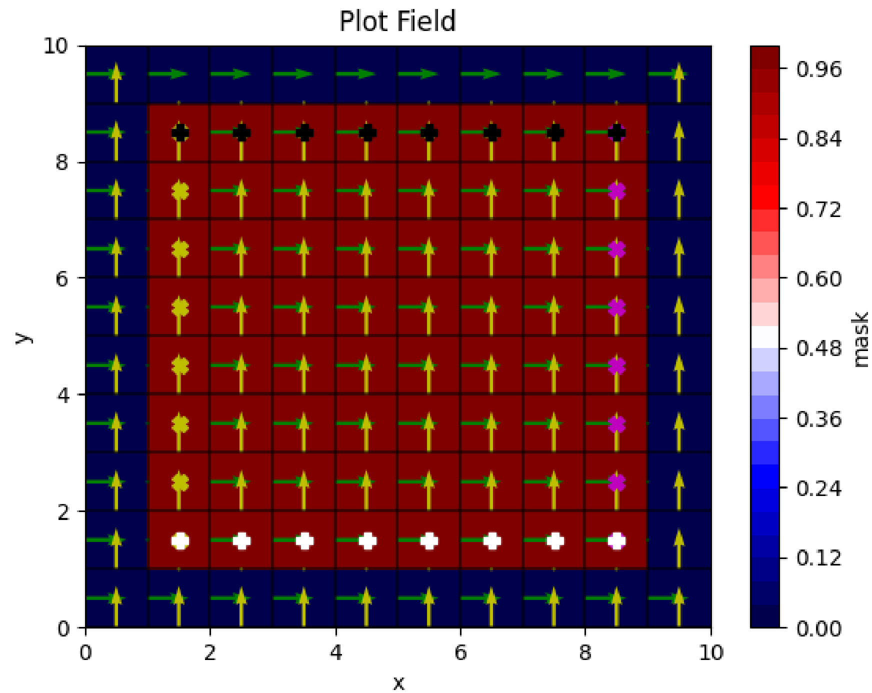
NeuraSim Manual

- set_normal_bc()

NOTICE: there are two types of set_XX_bc(). Indicated by a 2 at the end. Being the later more efficient but with more inputs precomputed.



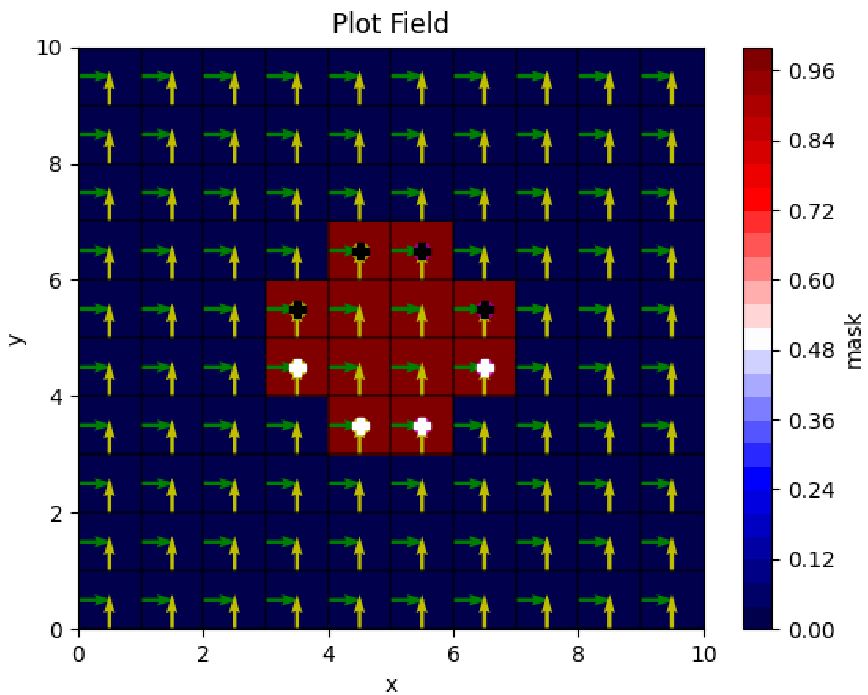
Input velocity field + object mask



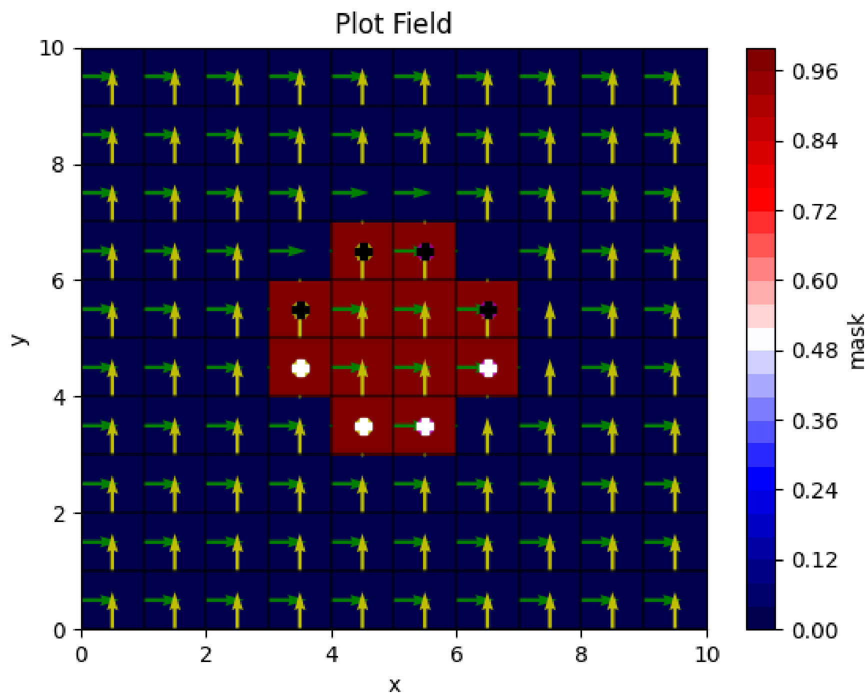
Output velocity field + object mask

NeuraSim Manual

- set_normal_bc()

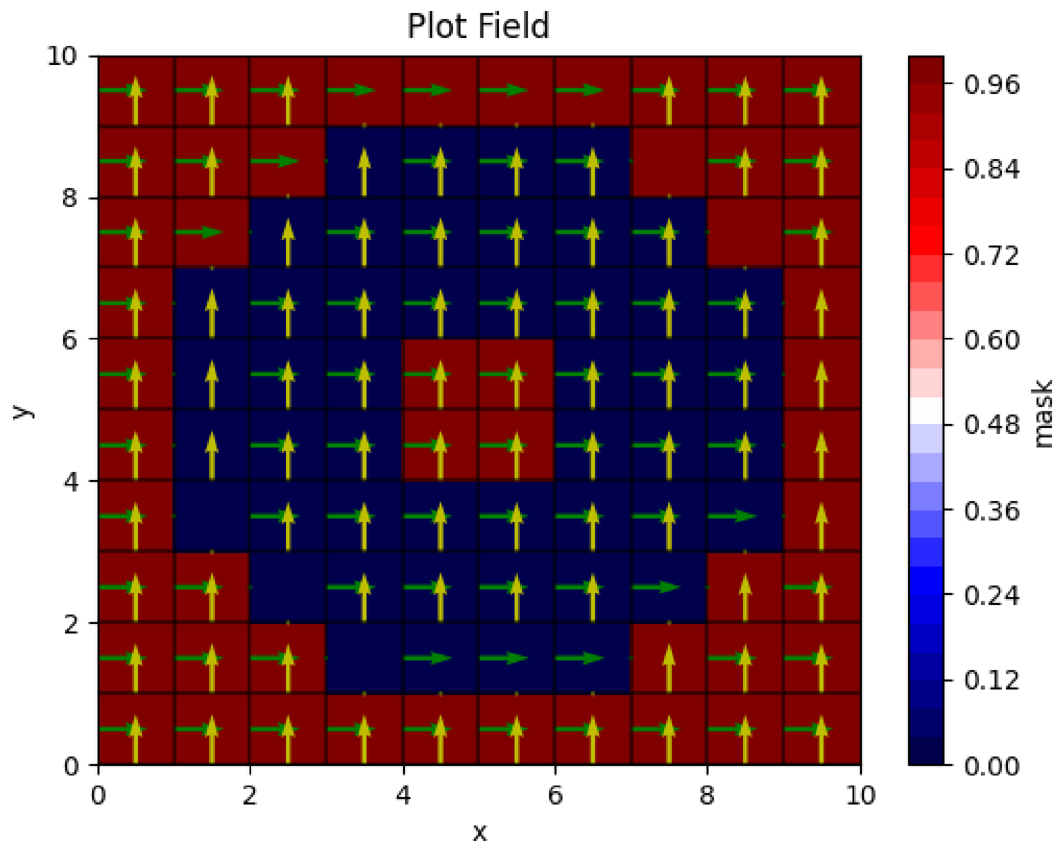


Input velocity field + object mask

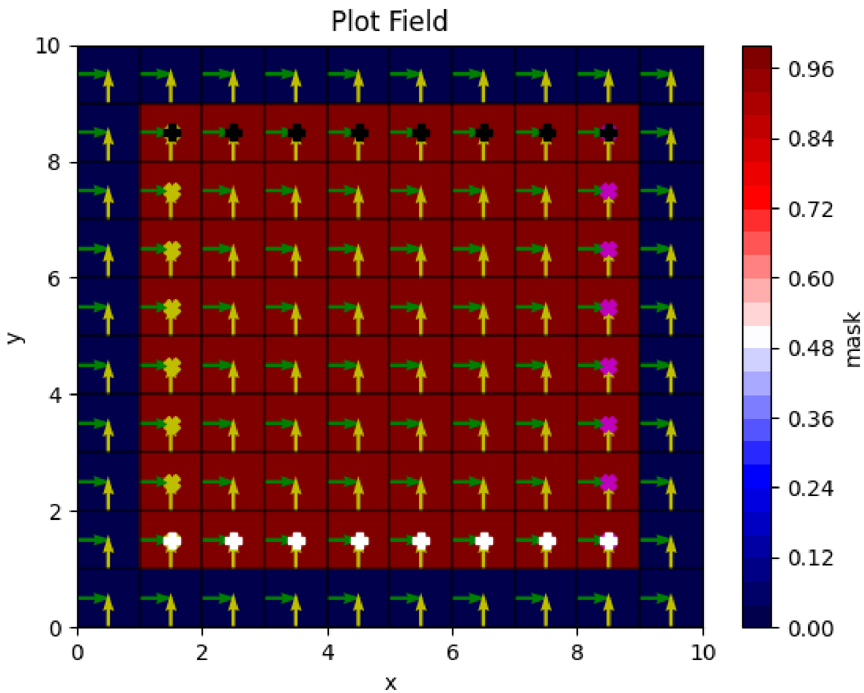


Output velocity field + object mask

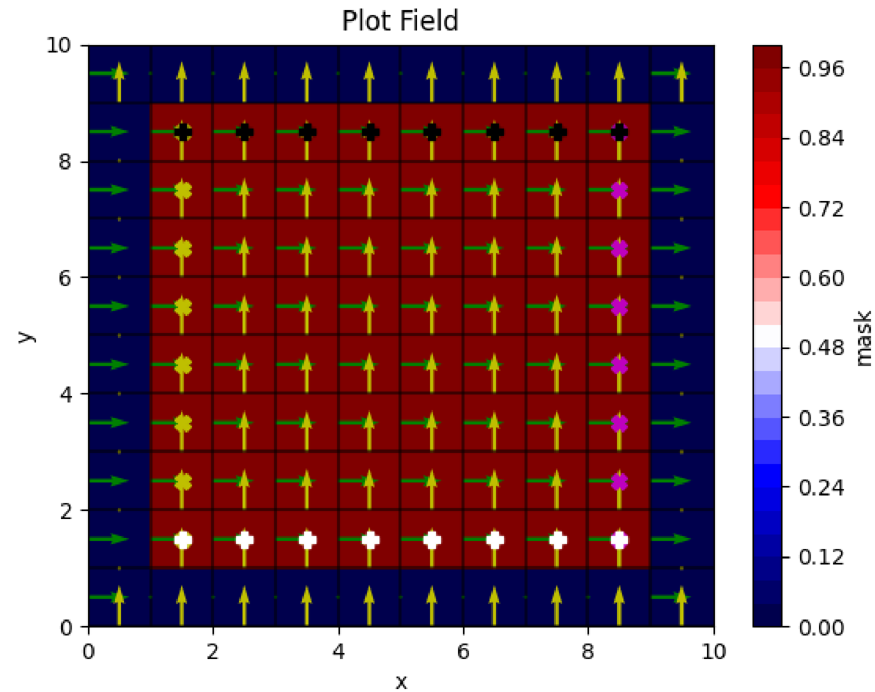
- set_normal_bc()   for **Inverse geometry**



- set_tangential_bc()



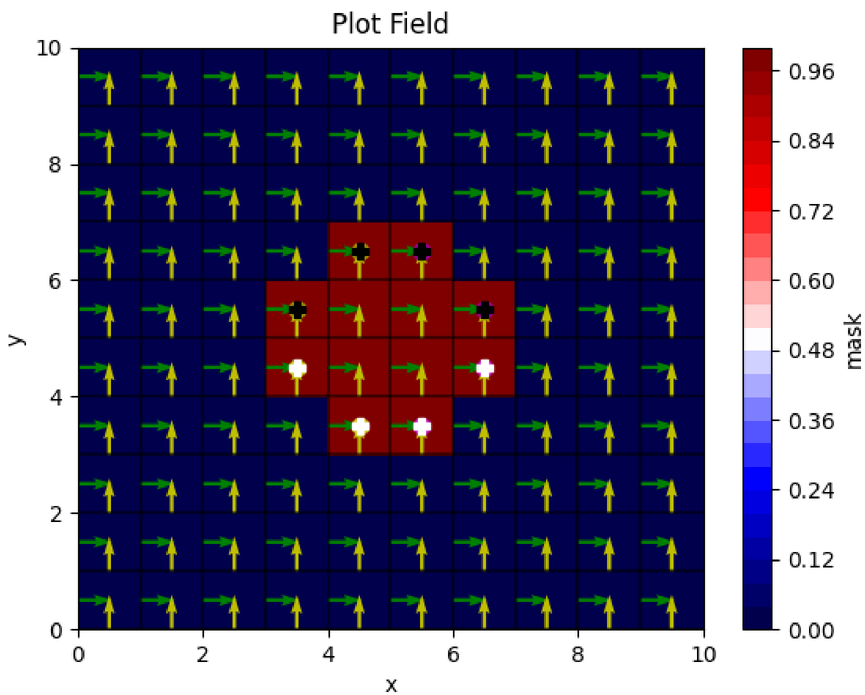
Input velocity field + object mask



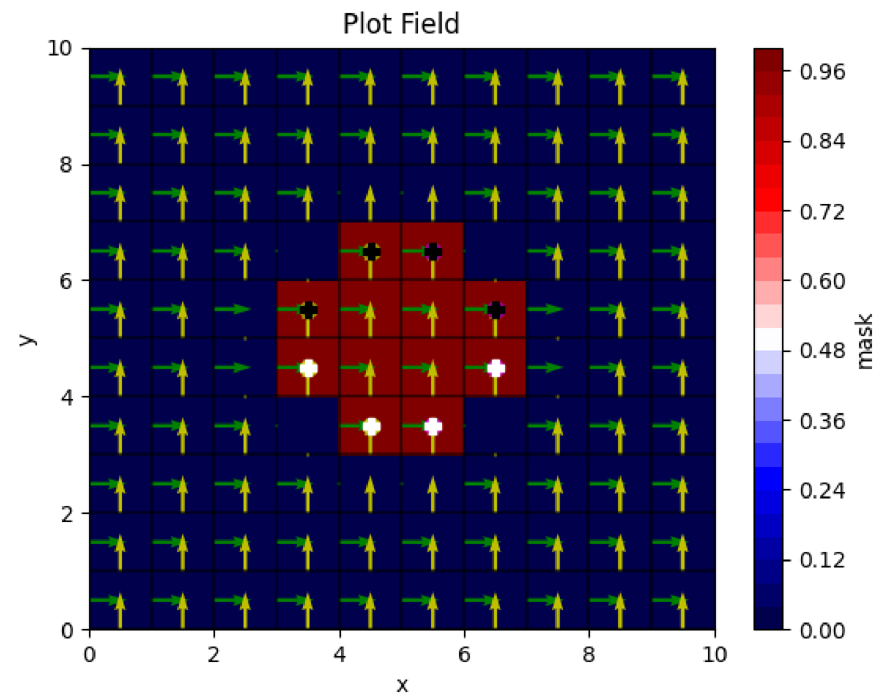
Output velocity field + object mask

NeuraSim Manual

- set_tangential_bc()

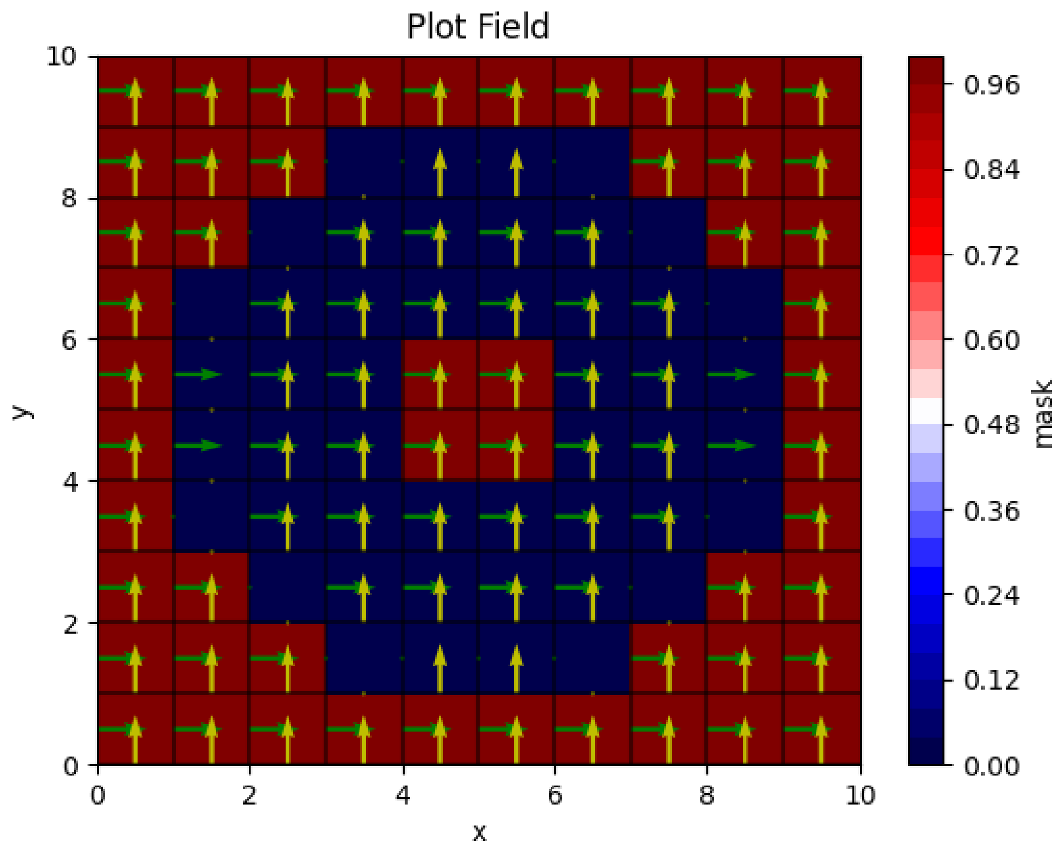


Input velocity field + object mask



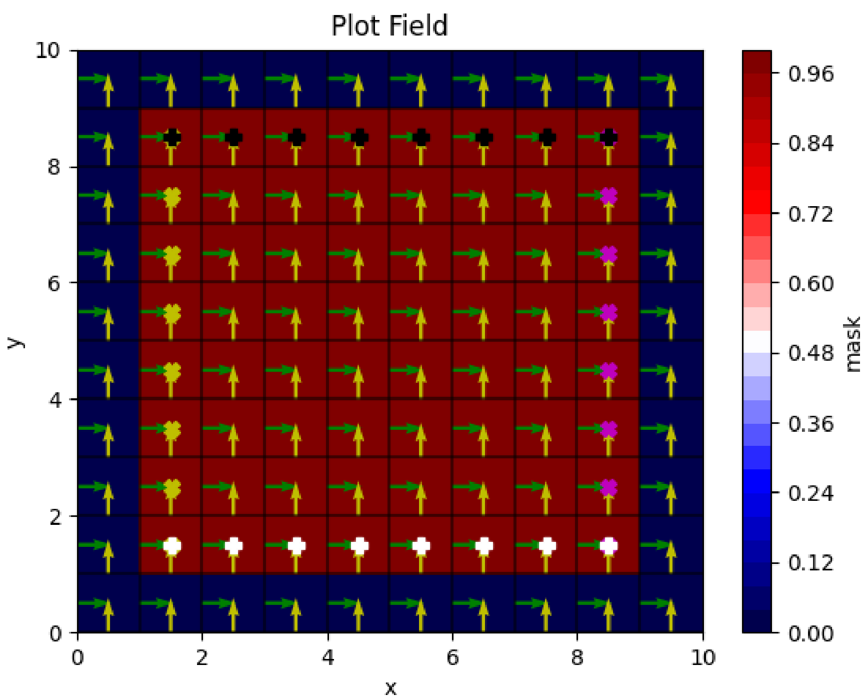
Output velocity field + object mask

- set_tangential_bc() for **Inverse geometry**

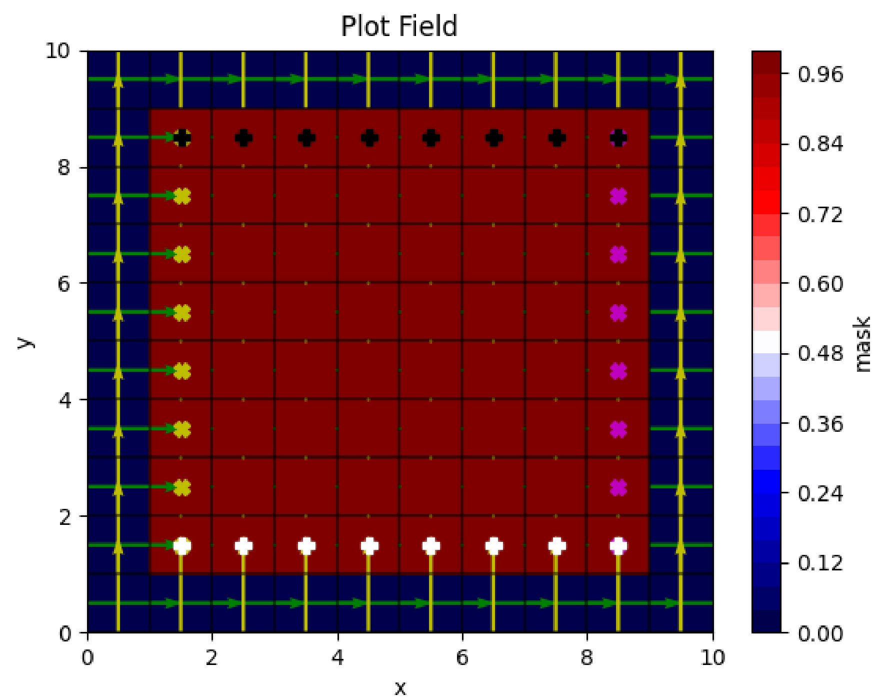


NeuraSim Manual

- set_internal_bc()



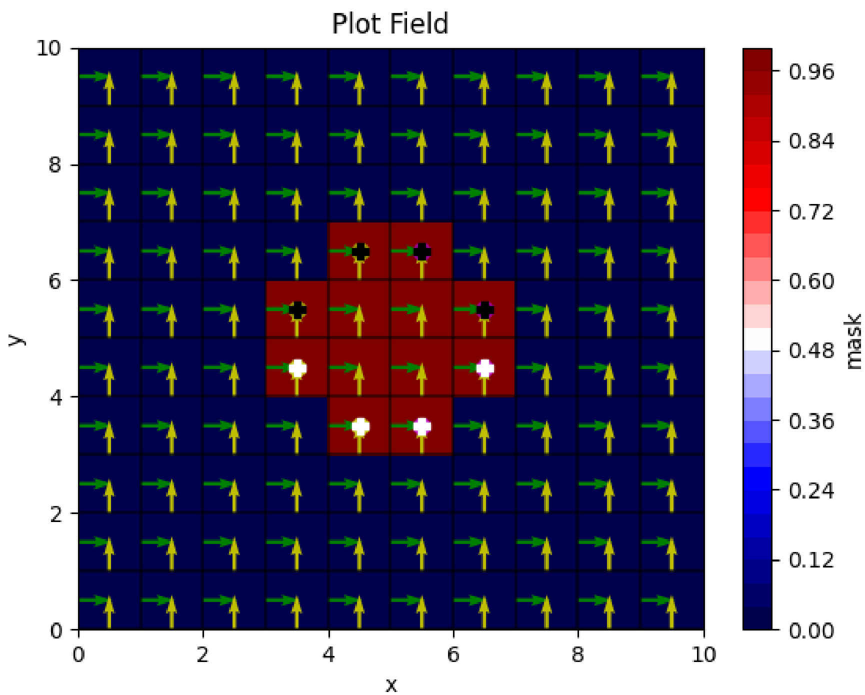
Input velocity field + object mask



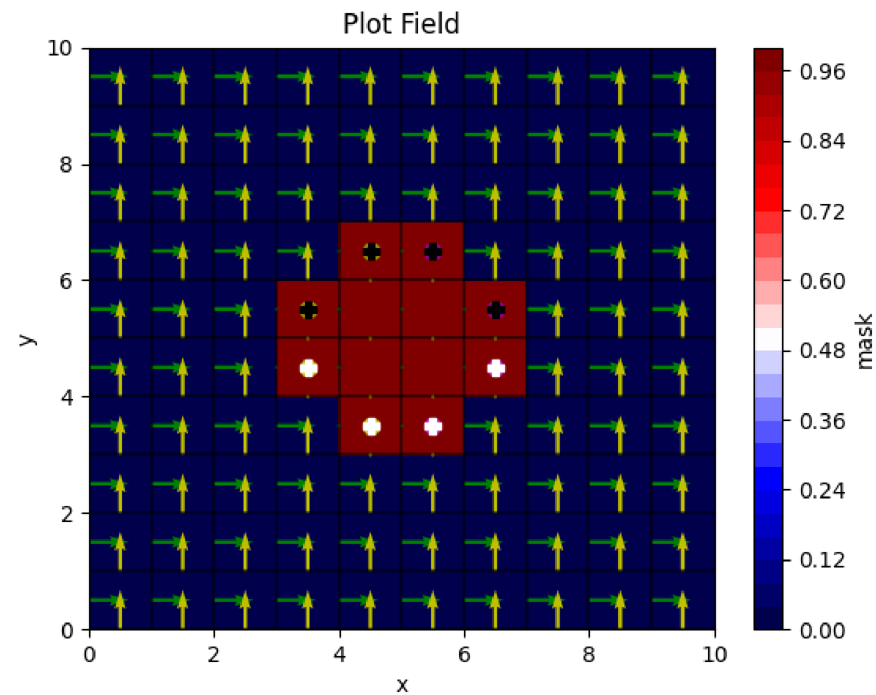
Output velocity field + object mask

NeuraSim Manual

- set_internal_bc()



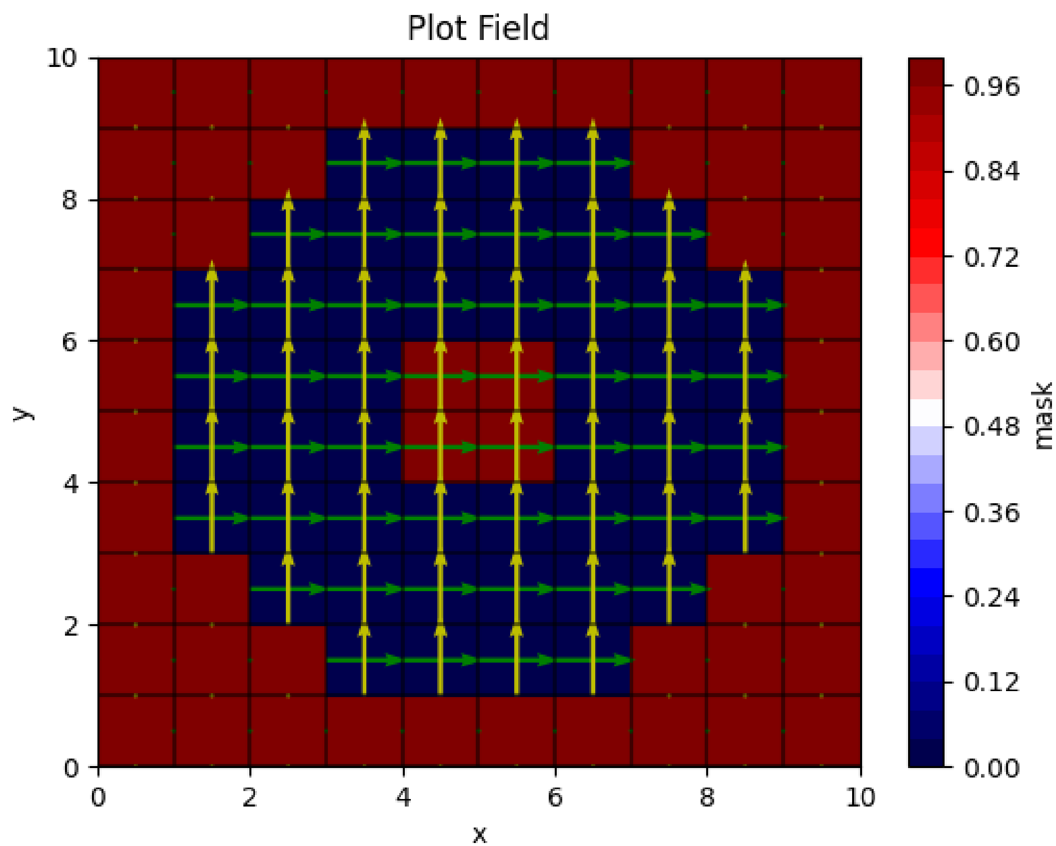
Input velocity field + object mask



Output velocity field + object mask

NeuraSim Manual

- set_internal_bc() for **Inverse geometry**

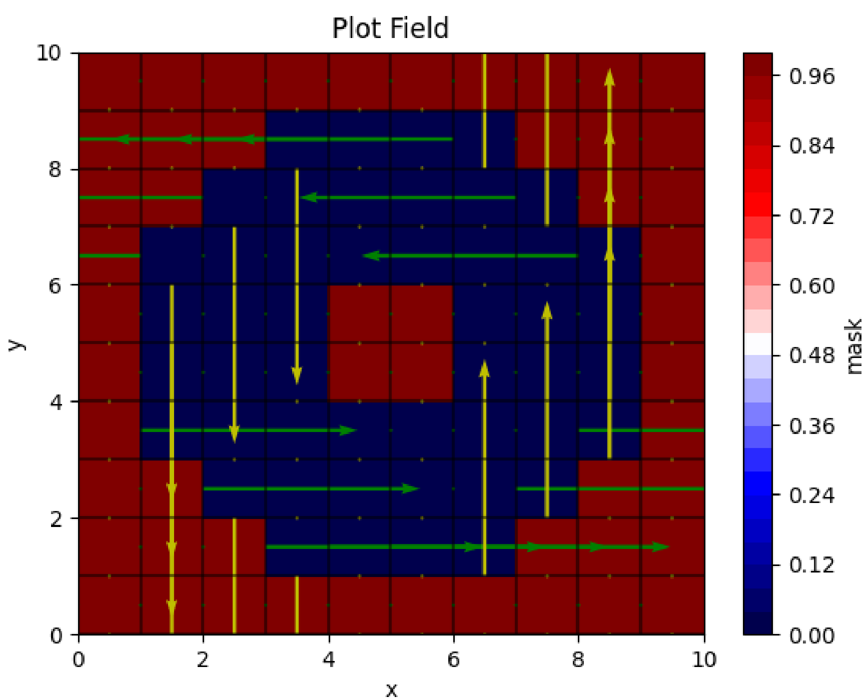


NeuraSim Manual

- get_obstacles_bc(obstacles)

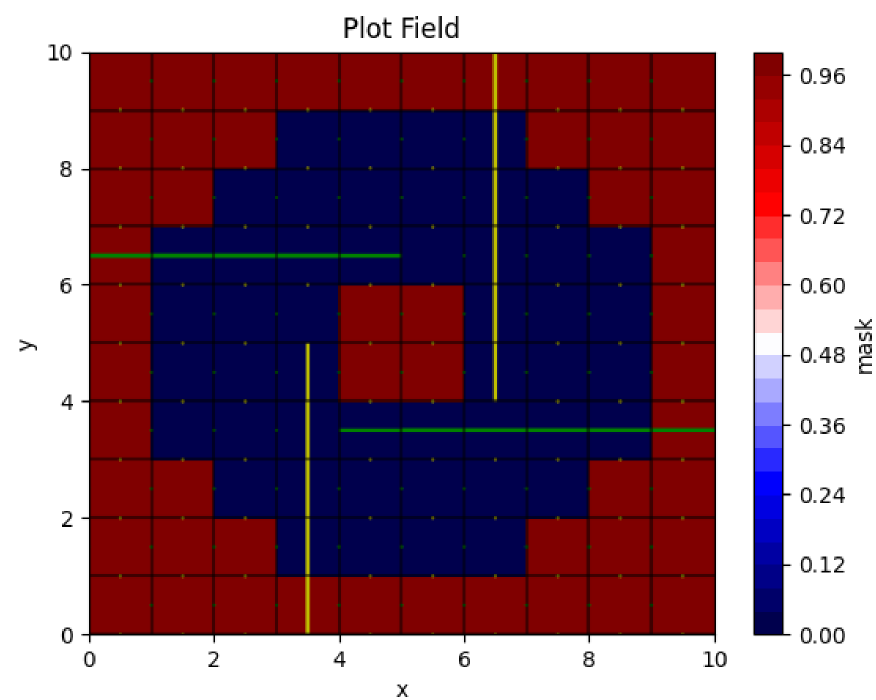
<u>bool</u>

obstacles: [[obstacle_mask, 'wr', 'inv_geom'], [obstacle_mask, 'wr', 'inv_geom'], ...]



`[[EXT, WR, True],[INT, 0, False]]`

Input (0,0)

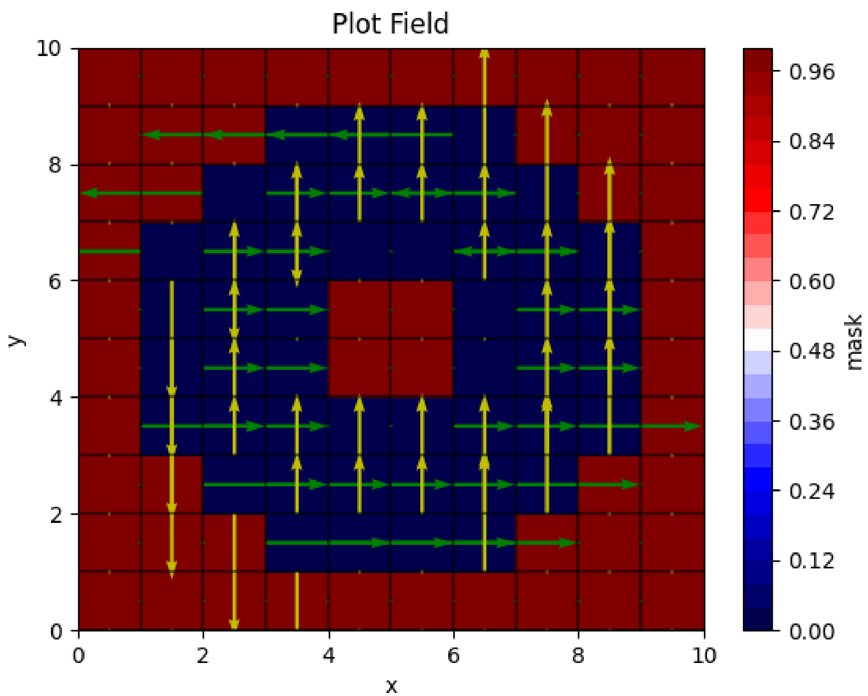


`[[EXT, 0, True],[INT, WR , False]]`

Input (0,0)

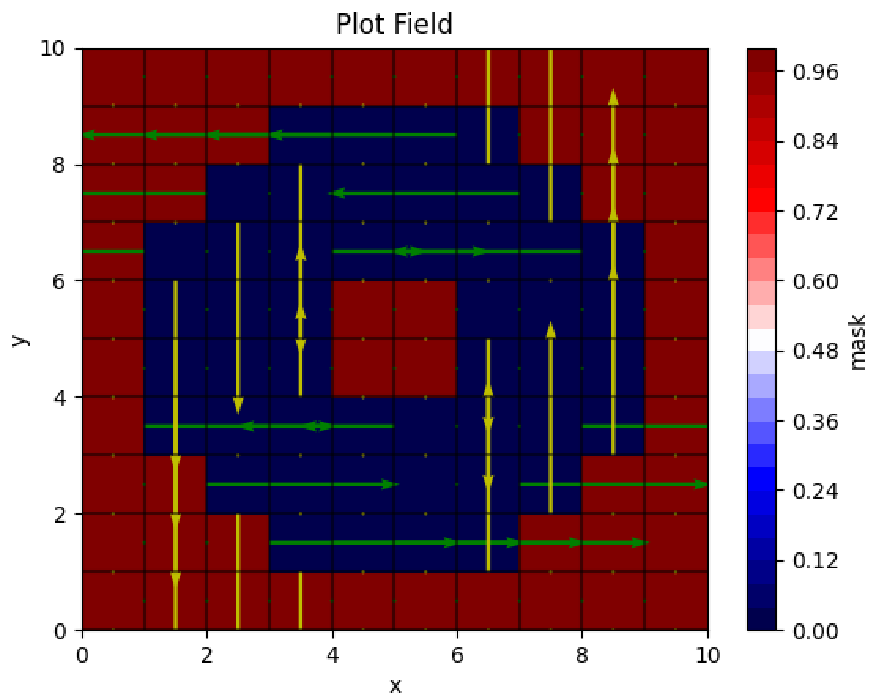
NeuraSim Manual

- get_obstacles_bc(obstacles)



`[[EXT, WR, True],[INT, 0, False]]`

Input (1,1)



`[[EXT, WR, True],[INT,-0.5*WR, False]]`

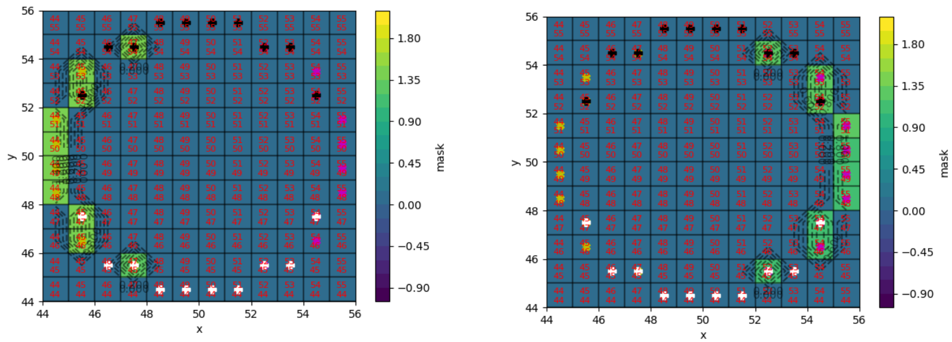
Input (0,0)

Analyze Functions

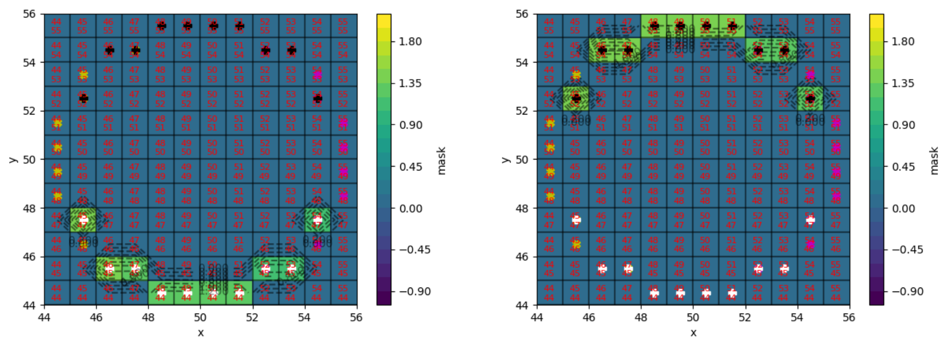
NeuraSim Manual

- calculate_forces(pressure, object_mask, dx, dy)

  - Force_h = np.sum(pressure[left] - pressure[right])*dy



  - Force_v = np.sum(pressure[bottom] - pressure[top])*dx



NeuraSim Manual

- calculate_forces_with_momentum(pressure, velocity, object_mask, factor=1, rho=1, dx=1, dy=1)

Plot Functions

- plot_field()

plot_field(field, plot_type=['surface’], options=[], Lx=None, Ly=None, dx=None, dy=None, lx='x', ly='y', lbar='field', ltitle='Plot Field', save=False, filename='./field.png’, fig=None, ax=None)

```
general_plot_options:
  -edges -> [ [edge_hl_x, edge_hl_y], [edge_hr_x, edge_hr_y], [edge_vb_x, edge_vb_y], [edge_vt_x, edge_vt_y] ]
  -square -> [x1,x2,y1,y2]
  -aux_contourn -> True/False

   -limits -> [min, max]
   -vector_axis -> 0/1

   -indeces -> True/False
   -grid -> True/False
   -velocity -> StaggeredGrid

   -zoom_position -> [x1,x2,y1,y2]
   -full_zoom -> True/False
```

- GIF

```
gif = GIF(gifname=f'./results/…', total_frames=Nt)

for ite in range(Nt):
        …
        gif.add_frame(ite,…, pressure, plots=['surface’], …, ltitle='…')


gif.build_gif()
```

field

Types of plots

```
plots=['surface’,mask, contourn,
streamlines, etc]
```

Institut Supérieur de l'Aéronautique et de l'Espace
10, avenue Édouard-Belin – BP 54032
31055 Toulouse Cedex 4 – France
T +33 5 61 33 80 80
www.isae-supaero.fr