fluidfoam Documentation

Release 0.2.6

Cyrille Bonamy

Apr 16, 2024

CONTENTS:

The fluidfoam package provides Python classes useful to perform some plot with OpenFoam data.

CHAPTER

ONE

WHAT IS THIS REPOSITORY FOR?

- Openfoam Tools
- Version : 0.2.6
- Supported OpenFoam Versions : 2.4.0, 4.1 to 9, v1712plus to v2312plus
- Supported Python Versions : >= 3.8

CHAPTER

TWO

DEPLOYMENT INSTRUCTIONS

The simplest way to install fluidfoam is by using pip:

```
pip install fluidfoam --user
```

You can get the source code from github or from the Python Package Index.

The development mode is often useful. From the root directory, run:

```
python setup.py develop --user
```

CHAPTER

THREE

COMMITTING INSTRUCTIONS (IN DEVELOPMENT MODE)

A good starting point is to follow this forking tutorial.

To clone your fork of fluidfoam repository:

```
git clone https://github.com/your_username/fluidfoam
```

To get the status of the repository:

```
git status
```

In case of new/modified file(s):

```
git add new_file
```

To commit a revision on the local repository:

```
git commit -m "comment on the revision"
```

To push the revision on your github fluidfoam repository:

```
git push
```

To propose your changes into the main fluidfoam project, follow again the forking tutorial.

CHAPTER

FOUR

EXAMPLE USAGE

- https://sedfoam.github.io

CHAPTER

FIVE

CORE DEVELOPERS

- Cyrille.Bonamy@univ-grenoble-alpes.fr

CHAPTER

SIX

OTHER CONTRIBUTORS

- Julien.Chauchat@univ-grenoble-alpes.fr
- amathieu@udel.edu
- Remi.Chassagne@univ-grenoble-alpes.fr
- Quentin.Clemencot@univ-grenoble-alpes.fr
- Matthias.Renaud@univ-grenoble-alpes.fr
- Alban.Gilletta.De.Saint.Joseph@france-energies-marines.org
- Gabriel Goncalves

CHAPTER

SEVEN

EMERITUS CORE DEVELOPERS

- Pierre.Augier@legi.cnrs.fr

CHAPTER

EIGHT

EMERITUS DEVELOPERS

- Guillaume.Maurice@univ-grenoble-alpes.fr
- Tim.Nagel@legi.cnrs.fr

CHAPTER

NINE

LICENSE

fluidfoam is distributed under the GNU General Public License v3 (GPLv3 or newer).

CHAPTER

TEN

MODULES REFERENCE

Here is presented the general organization of the package and the documentation of the modules, classes and functions.

fluidfoam.readof	Reading OpenFoam Files with Python This module provides functions to read OpenFoam Files:
fluidfoam.processing1d	Write, Read and Plot 1D input files for swak4foam This module allows to read OpenFoam output of one dimensional computation and then write, plot and read input files for Boundary and Initial Conditions imposition in 3D computation (via swak4foam):
fluidfoam.readpostpro	Read OpenFoam PostProcessing Files for Python This module provides functions to list and read OpenFoam PostProcessing Files:
fluidfoam.meshdesign	Compute mesh grading and cell sizes This module provides functions to design OpenFoam mesh using blockMesh:
fluidfoam.meshvisu	Visualisation of 2D OpenFoam Mesh with Python This module provides functions to read 2D OpenFoam Mesh:
fluidfoam.openfoamsimu	Class to load all data saved at timeStep of an openFoam simulation

10.1 fluidfoam.readof

10.1.1 Reading OpenFoam Files with Python

This module provides functions to read OpenFoam Files:

fluidfoam.readof.**readmesh**(*path*, *time_name=None*, *structured=False*, *boundary=None*, *sets=None*, *region=None*, *order='F'*, *precision=15*, *verbose=True*)

Read OpenFoam mesh and reshape if necessary (in cartesian structured mesh).

Args:

path: str

time_name: str ('latestTime' is supported)

structured: False or True

boundary: None or str

sets: None or str

region: None or str

order: “F” (default) or “C”

precision : Number of decimal places to round to (default: 15)

verbose : True or False (default: True).

Returns:

array: array of vector (Mesh X, Y, Z); size of the array is the size of the interior domain (or of the size of the boundary in case of not None boundary)

A way you might use me is:

X, Y, Z = fluidfoam.readmesh(‘path_of_OpenFoam_case’) So X, Y and Z are 1D numpy array with size = nb_cell

If you play with structured mesh you can shape the X, Y and Z output :

X, Y, Z = fluidfoam.readmesh(‘path_of_OpenFoam_case’, structured=True) So X, Y and Z are 3D numpy array with shape = (nx, ny, nz)

And if you play with dynamic mesh the time_name option is for you

fluidfoam.readof.**readfield**(*path*, *time_name=None*, *name=None*, *structured=False*, *boundary=None*, *sets=None*, *region=None*, *order='F'*, *precision=15*, *datatype=None*, *verbose=True*)

Read OpenFoam field and reshape if necessary (structured mesh) and possible (not uniform field).

Args:

path: str

time_name: str (‘latestTime’ is supported)

name: str

structured: False or True

boundary: None or str

sets: None or str

region: None or str

order: “F” (default) or “C”

precision : Number of decimal places to round to (default: 15)

datatype: None (default) or str (“scalar”, “vector”...) necessary in case of files without header

verbose : True or False (default: True).

Returns:

array: array of type of the field; size of the array is the size of the interior domain (or of the size of the boundary in case of not None boundary)

A way you might use me is:

field = fluidfoam.readfield(‘path_of_OpenFoam_case’, ‘0’, ‘alpha’)

fluidfoam.readof.**readscalar**(*path*, *time_name=None*, *name=None*, *structured=False*, *boundary=None*, *sets=None*, *region=None*, *order='F'*, *precision=15*, *mode=None*, *verbose=True*)

Read OpenFoam scalar field and reshape if necessary and possible (not uniform field).

Args:

path: str

time_name: str ('latestTime' is supported)

name: str

structured: False or True

boundary: None or str

sets: None or str

region: None or str

order: "F" (default) or "C"

precision : Number of decimal places to round to (default: 15)

verbose : True or False (default: True).

Returns:

array: array of scalar field; size of the array is the size of the interior domain (or of the size of the boundary in case of not None boundary)

A way you might use me is:

scalar_a = fluidfoam.readscalar('path_of_OpenFoam_case', '0', 'alpha')

fluidfoam.readof.**readvector**(*path*, *time_name=None*, *name=None*, *structured=False*, *boundary=None*, *sets=None*, *region=None*, *order='F'*, *precision=15*, *verbose=True*)

Read OpenFoam vector field and reshape if necessary and possible (not uniform field).

Args:

path: str

time_name: str ('latestTime' is supported)

name: str

structured: False or True

boundary: None or str

sets: None or str

region: None or str

order: "F" (default) or "C"

precision : Number of decimal places to round to (default: 15)

verbose : True or False (default: True).

Returns:

array: array of vector field; size of the array is the size of the interior domain (or of the size of the boundary in case of not None boundary)

A way you might use me is:

U = fluidfoam.readvector('path_of_OpenFoam_case', '0', 'U')

fluidfoam.readof.**readsymmtensor**(*path*, *time_name=None*, *name=None*, *structured=False*, *boundary=None*, *sets=None*, *region=None*, *order='F'*, *precision=15*, *verbose=True*)

Read OpenFoam symmetrical tensor field and reshape if necessary and possible (not uniform field).

Args:

path: str

time_name: str ('latestTime' is supported)

name: str

structured: False or True

boundary: None or str

sets: None or str

region: None or str

order: "F" (default) or "C"

precision : Number of decimal places to round to (default: 15)

verbose : True or False (default: True).

Returns:

array: array of symmetrical tensor field; size of the array is the size of the interior domain (or of the size of the boundary in case of not None boundary)

A way you might use me is:

sigma = fluidfoam.readsymmtensor('path_of_OpenFoam_case', '0', 'sigma')

`fluidfoam.readof.`**`readtensor`**(*path*, *time_name=None*, *name=None*, *structured=False*, *boundary=None*, *sets=None*, *region=None*, *order='F'*, *precision=15*, *verbose=True*)

Read OpenFoam tensor field and reshape if necessary and possible (not uniform field).

Args:

path: str

time_name: str ('latestTime' is supported)

name: str

structured: False or True

boundary: None or str

sets: None or str

region: None or str

order: "F" (default) or "C"

precision : Number of decimal places to round to (default: 15)

verbose : True or False (default: True).

Returns:

array: array of tensor field; size of the array is the size of the interior domain (or of the size of the boundary in case of not None boundary)

A way you might use me is:

tens = fluidfoam.readtensor('path_of_OpenFoam_case', '0', 'tens')

`fluidfoam.readof.`**`getVolumes`**(*path*, *time_name=None*, *structured=False*, *boundary=None*, *sets=None*, *region=None*, *order='F'*, *precision=15*, *verbose=True*, *box=None*)

Reads OpenFoam mesh and returns the cell centroids and cell volumes of a given box.

Args:

path: str

time_name: str ('latestTime' is supported)

structured: False or True

boundary: None or str

sets: None or str

region: None or str

order: "F" (default) or "C"

precision : Number of decimal places to round to (default: 15)

verbose : True or False (default: True) box : tuple of box's dimension: ((xmin, ymin, zmin), (xmax, ymax, zmax))

(if None, includes the whole mesh)

Returns:

array: two arrays that contain the cell centroids and cell volumes

A way you might use me is:

centroidList,vol = fluidfoam.getVolumes('path_of_OpenFoam_case') So centroidList and vol are the cell centroid and cell volume arrays.

`fluidfoam.readof.`**`typefield`**(*path*, *time_name=None*, *name=None*, *verbose=True*)

Read OpenFoam field and returns type of field.

Args:

path: str

time_name: str ('latestTime' is supported)

name: str

Returns:

str: type of field

A way you might use me is:

print("type of alpha field is", fluidfoam.typefield('path_of_OpenFoam_case', '0', 'alpha'))

Functions

getVolumes(path[, time_name, structured, ...])	Reads OpenFoam mesh and returns the cell centroids and cell volumes of a given box.
readfield(path[, time_name, name, ...])	Read OpenFoam field and reshape if necessary (structured mesh) and possible (not uniform field).
readmesh(path[, time_name, structured, ...])	Read OpenFoam mesh and reshape if necessary (in cartesian structured mesh).
readscalar(path[, time_name, name, ...])	Read OpenFoam scalar field and reshape if necessary and possible (not uniform field).
readsymmtensor(path[, time_name, name, ...])	Read OpenFoam symmetrical tensor field and reshape if necessary and possible (not uniform field).
readtensor(path[, time_name, name, ...])	Read OpenFoam tensor field and reshape if necessary and possible (not uniform field).
readvector(path[, time_name, name, ...])	Read OpenFoam vector field and reshape if necessary and possible (not uniform field).
typefield(path[, time_name, name, verbose])	Read OpenFoam field and returns type of field.

Classes

OpenFoamFile(path[, time_name, name, ...])	OpenFoam file parser.

10.2 fluidfoam.processing1d

10.2.1 Write, Read and Plot 1D input files for swak4foam

This module allows to read OpenFoam output of one dimensional computation and then write, plot and read input files for Boundary and Initial Conditions imposition in 3D computation (via swak4foam):

fluidfoam.processing1d.**create1dprofil**(*pathr*, *pathw*, *timename*, *axis*, *varlist*)

This function provides way to read 1D profiles at time timename of pathr and write them in OpenFoam Format in the 1d_profil folder of pathw (for BC imposition in 2D or 3D case for example).

Args:

pathr: str

pathw: str

timename: str

axis: str

varlist: list of str

Returns:

status: 'create 1D profiles: done' if ok

A way you might use me is:

status = fluidfoam.create1dprofil("path_of_case", "pathw", time, 'Y', ['Ua', 'Ub'])

Please note that the 1d_profil directory must be existing in the pathw directory

fluidfoam.processing1d.**read1dprofil**(*file_name*)

This function provides way to read and return 1D profil created by the create1dprofil function. file_name can be a complete path.

Args:
filename: str

Returns:
z: 1d mesh corresponding to 1d profil

field: scalar value of the field specified via filename

size1d: size of the 1d profil

A way you might use me is:

z, a, size1d = fluidfoam.read1dprofil("path_of_case/1d_profil/a.xy")

fluidfoam.processing1d.**plot1dprofil**(*pathr*, *varlist*)

This function provides way to plot 1D profiles created by the create1dprofil function.

Args:
pathr: str (must be the full path of the 1d_profil directory)

varlist: list of str

A way you might use me is:

fluidfoam.plot1dprofil("path_of_case/1d_profil", ['Ua', 'Ub', 'alpha'])

Functions

create1dprofil(pathr, pathw, timename, axis, ...)	This function provides way to read 1D profiles at time timename of pathr and write them in OpenFoam Format in the 1d_profil folder of pathw (for BC imposition in 2D or 3D case for example).
create1dprofilDFSEM(pathr, pathw, ...)	This function provides way to read 1D profiles at time timename of pathr and write them in OpenFoam Format in the 1d_profil folder of pathw (for BC imposition in 2D or 3D case for example).
create1dprofil_spe(pathw, waxis, var, ...)	This function provides way to write specific 1D profil (var array) in OpenFoam Format in the 1d_profil folder of pathw (for BC imposition in 2D or 3D case for example).
plot1dprofil(pathr, varlist)	This function provides way to plot 1D profiles created by the create1dprofil function.
read1dprofil(file_name)	This function provides way to read and return 1D profil created by the create1dprofil function.
read1dprofilDFSEM(file_name, boundary_name, ...)	This function provides way to read and return 1D profil created by the create1dprofilDFSEM function.

10.3 fluidfoam.readpostpro

10.3.1 Read OpenFoam PostProcessing Files for Python

This module provides functions to list and read OpenFoam PostProcessing Files:

`fluidfoam.readpostpro.`**`readforce`**(*path*, *namepatch='forces'*, *time_name='0'*, *name='forces'*)

read the data contained in the force file . create the forces variables in the Forcesfile object

Args:

path: str

namepatch: str

time_name: str ('latestTime' and 'mergeTime' are supported)

name: str

Returns:

array: array of force field; size of the array is the size of the time

A way you might use me is:

force = readforce(path='path_of_OpenFoam_case', namepatch='forces',
time_name='0', name='forces')

`fluidfoam.readpostpro.`**`readprobes`**(*path*, *probes_name='probes'*, *time_name='0'*, *name='U'*)

read the data contained in the force file . create the forces variables in the Forcesfile object

Args:

path: str

probes_name: str

time_name: str ('latestTime' and 'mergeTime' are supported)

name: str

Returns:

array: array of time values and array of probes data;

A way you might use me is:

probe_data = read('path_of_OpenFoam_case', '0', 'probes', 'U')

Functions

readforce(path[, namepatch, time_name, name])	read the data contained in the force file .
readprobes(path[, probes_name, time_name, name])	read the data contained in the force file .

10.4 fluidfoam.meshdesign

10.4.1 Compute mesh grading and cell sizes

This module provides functions to design OpenFoam mesh using blockMesh:

`fluidfoam.meshdesign.`**`getgz`**(*h*, *dz1*, *N*)

Given a domain size h, a first grid size dz1 and a number of points N this function returns the common ratio, the grading gz to enter in blockMesk and the z and dz vectors. Usage: z,dz,gz=getgz(h,dz1,N)

`fluidfoam.meshdesign.`**`getdzs`**(*h*, *gz*, *N*)

Given a domain size h, a grading factor gz and a number of points N this function returns the grid size of the first and the last points. Usage: dz1,dzN=getdzs(h,gz,N)

Functions

getdzs(h, gz, N)	Given a domain size h, a grading factor gz and a number of points N this function returns the grid size of the first and the last points.
getgz(h, dz1, N)	Given a domain size h, a first grid size dz1 and a number of points N this function returns the common ratio, the grading gz to enter in blockMesk and the z and dz vectors.

10.5 fluidfoam.meshvisu

10.5.1 Visualisation of 2D OpenFoam Mesh with Python

This module provides functions to read 2D OpenFoam Mesh:

class `fluidfoam.meshvisu.`**`MeshVisu`**(*path*, *box=None*, *plane='xy'*, *time_name=None*, *verbose=True*)

Read OpenFoam mesh of 2D planar simulation and list all the edges contained in a box.

Args:

path: str

box: tuple of box's dimension: ((xmin, ymin, zmin), (xmax, ymax, zmax))

(if None, includes the whole mesh)

plane: str plane in which the mesh is contained, either:

'xy': the xy-plane of outgoing normal z (default value)

'xz': the xz-plane of outgoing normal -y

'yz': the yz-plane of outgoing normal x

time_name: str ('latestTime' is supported)

verbose : True or False (default: True)

A way you might use me is:

MyMesh = fluidfoam.MeshVisu(path = 'path_of_OpenFoam_case')

Then a minimal example to generate and save vectorial mesh figure could be:

```
import matplotlib.pyplot as plt

from matplotlib.collections import LineCollection

fig, ax = plt.subplots()

ln_coll = LineCollection(MyMesh.get_all_edgesInBox())

ax.add_collection(ln_coll, autolim=True)

plt.savefig('./myMesh.svg', dpi=fig.dpi, transparent = True, bbox_inches = 'tight')
```

MeshVisu.**get_xlim**()

returns the x limits of the mesh visualization box.

Returns:
tuple of floats: (xmin, xmax)

MeshVisu.**get_ylim**()

returns the y limits of the mesh visualization box.

Returns:
tuple of floats: (ymin, ymax)

MeshVisu.**get_zlim**()

returns the z limits of the mesh visualization box.

Returns:
tuple of floats: (zmin, zmax)

MeshVisu.**get_all_edgesInBox**()

return the list of all edges in box.

Eatch edge is describe by a tuple of tuples of float: ((x0, y0), (x1, y1)).

(x0, y0) being the coordonates of the fisrt point, (x1, y1), the coordinates of the second point.

This list can be given as an argument to the matplotlib LineCollection function, which allows to display a large number of segments on an image.

Returns:
list of tuples

MeshVisu.**update_box**(*box*, *verbose=True*)

updates the mesh visualization box

Args:
box: tuple ((xmin, ymin, zmin), (xmax, ymax, zmax))

A way you might use me is:

```
MyMesh.update(box = ((0, 0, -1), (0.05, 0.05, 1)))
```

MeshVisu.**get_box**()

return the mesh visualization box

Returns:
tuple: ((xmin, ymin, zmin), (xmax, ymax, zmax))

MeshVisu.**set_box_to_mesh_size**(*verbose=False*)

Set the mesh visualization box to mesh size.

Classes

MeshVisu(path[, box, plane, time_name, verbose])	Read OpenFoam mesh of 2D planar simulation and list all the edges contained in a box.

10.6 fluidfoam.openfoamsimu

10.6.1 Class to load all data saved at timeStep of an openFoam simulation

class fluidfoam.openfoamsimu.**OpenFoamSimu**(*path*, *simu=None*, *timeStep=None*, *structured=False*)

Class to load all data saved at timeStep of an openFoam simulation

Args:

path: str, reference path where simulations are stored.

You may want to provide path if all your simulations are located inside path and subfolders of path. You can do it by modifying in the __init__ path='/path/to/the/simulations/'

simu: str, name of the simu that has to be loaded.

If simu=None, it will lists all existing simulation names in path and ask you to choose.

timeStep: str, timeStep to load. If None, load the last time step

structured: bool, true if the mesh is structured

OpenFoamSimu.**keys**()

Print the name of all variables loaded from simulation results

OpenFoamSimu.**readopenfoam**(*timeStep=None*, *structured=True*)

Reading SedFoam results Load the last time step saved of the simulation

Args:

timeStep : str or int, timeStep to load. If None, load the last time step

structured : bool, true if the mesh is structured

Classes

OpenFoamSimu(path[, simu, timeStep, structured])	Class to load all data saved at timeStep of an openFoam simulation

Exceptions

DirectorySimuError(simu)
Error

10.7 Gallery of Examples

10.7.1 First examples

This gallery consists of introductory examples to read and plot OpenFoam output files with Python.

First example

This example doesn't do much, it just reads and makes a simple plot of OpenFoam field

First read a scalar field

Note: It just reads a scalar field and store it in alpha variable

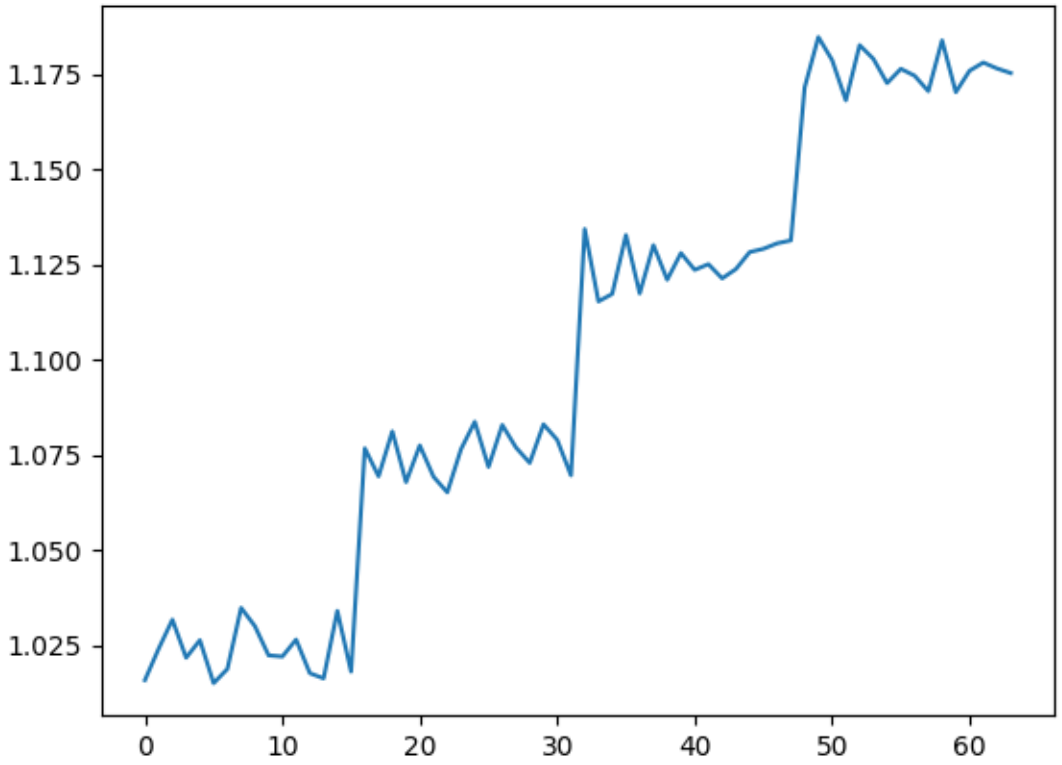
```
# import readscalar function from fluidfoam package
from fluidfoam import readscalar

sol = '../output_samples/bin/'
timename = '0'

alpha = readscalar(sol, timename, 'alpha')
```

```
Reading file ../output_samples/bin/0/alpha
```

Now plot this scalar field

In this example, we haven't read the mesh, and can be structured or unstructured

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(alpha)
```




```
[<matplotlib.lines.Line2D object at 0x7f00e2224f50>]
```

Total running time of the script: (0 minutes 0.137 seconds)

output field of files without header (sampling)

This example doesn't do much, it just reads and makes a simple plot of OpenFoam field in case of files without header (as for example the output of sampling library)

Read a scalar sampled field and the associated mesh

Note: It reads a scalar sampled field and the associated mesh

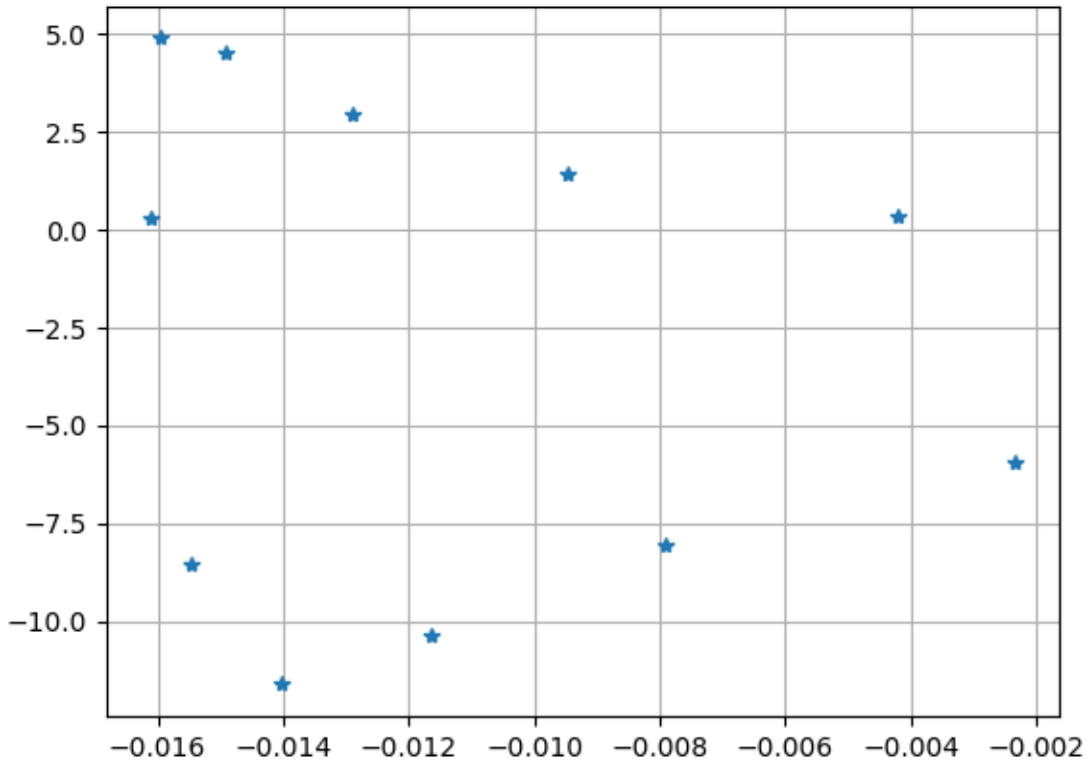
```
# import readscalar, readvector function from fluidfoam package
from fluidfoam import readscalar, readvector

sol = '../output_samples/ascii/wohead'

X, Y, Z = readvector(sol, 'faceCentres')
pressure = readscalar(sol, 'p')
```

```
Reading file ../output_samples/ascii/wohead/faceCentres
Reading file ../output_samples/ascii/wohead/p
```

Now plot this scalar field

In this example it is the pressure coefficient around an airfoil. It can be useful to sort the data in order to plot a line and not stars

```
import matplotlib.pyplot as plt
plt.figure()
plt.plot(X, pressure, '*')
plt.grid()
plt.show()
```



Total running time of the script: (0 minutes 0.117 seconds)

Time series of postProcessing force

This example reads and plots a series of postProcessing force

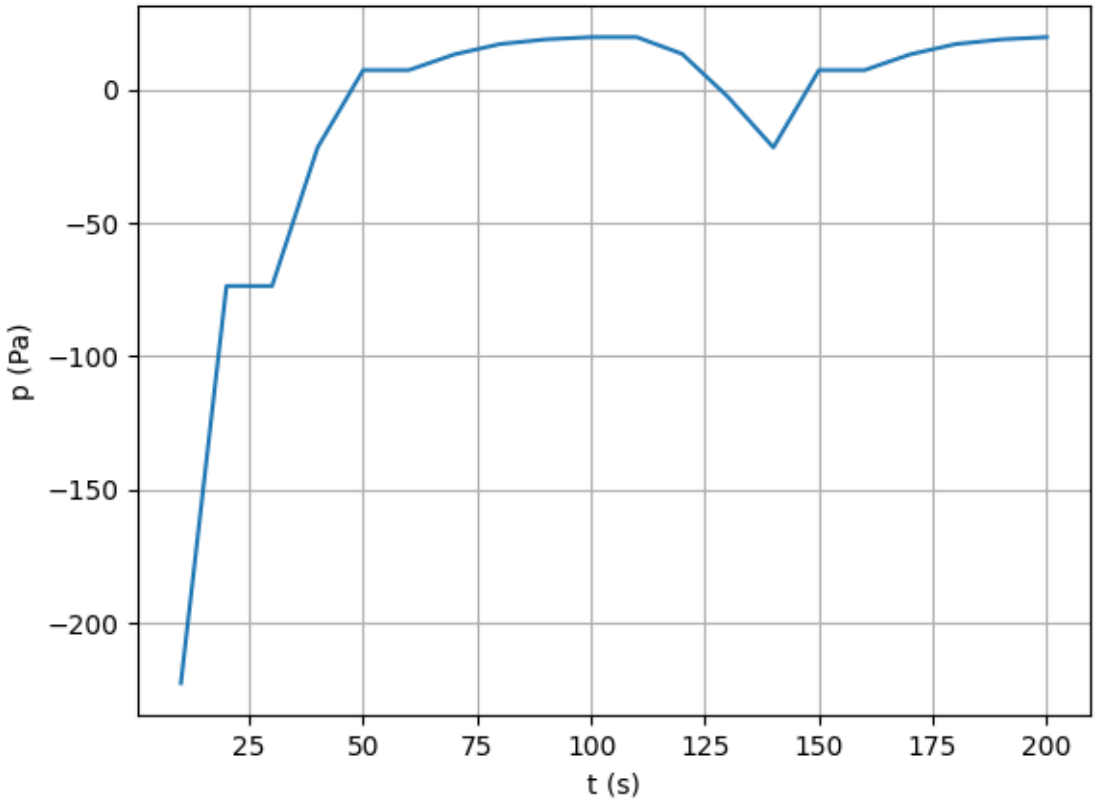
Read the postProcessing files

Note: In this example it reads and merges two postProcessing files automatically (with the 'mergeTime' option)

```
# import readforce function from fluidfoam package
from fluidfoam.readpostpro import readforce

sol = '../output_samples/ascii/'

force = readforce(sol, time_name = 'mergeTime')
```

Now plots the pressure force

```
import matplotlib.pyplot as plt

plt.figure()

plt.plot(force[:, 0], force[:, 1])

# Setting axis labels
plt.xlabel('t (s)')
plt.ylabel('p (Pa)')

# add grid
plt.grid()
```



Total running time of the script: (0 minutes 0.117 seconds)

Time series of postProcessing probe

This example reads and plots a series of postProcessing probe

Read the postProcessing files

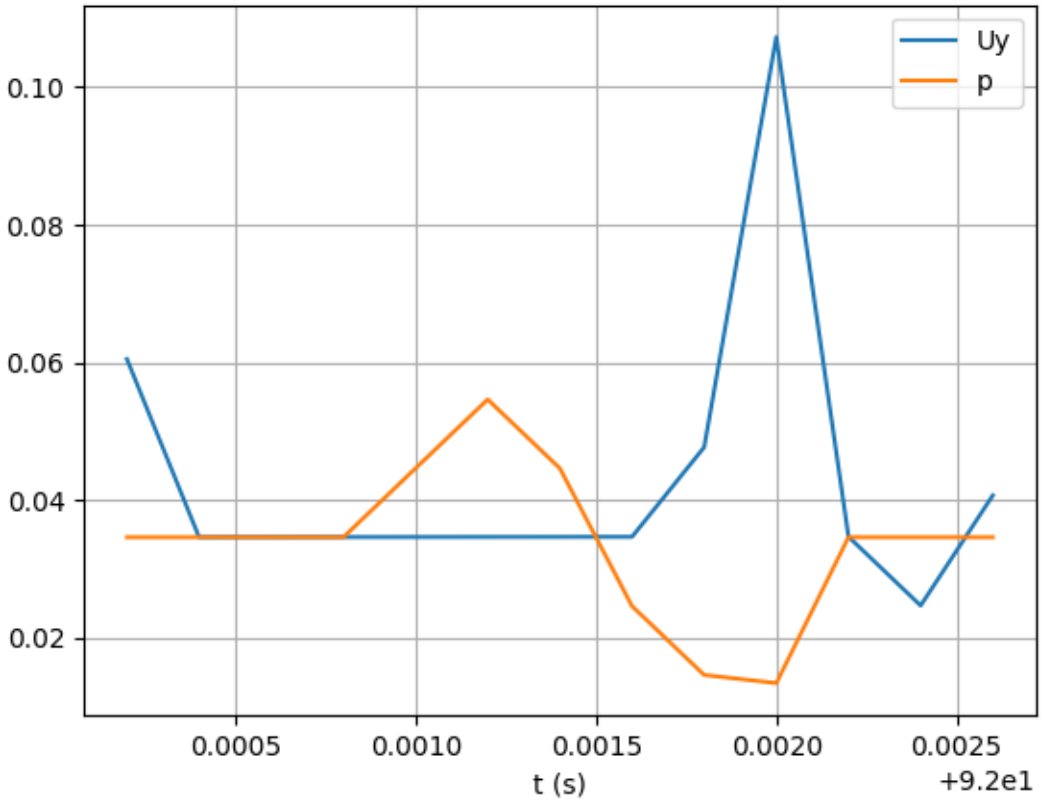
Note: In this example it reads and merges two postProcessing files automatically (with the 'mergeTime' option)

```
# import readprobes function from fluidfoam package
from fluidfoam.readpostpro import readprobes

sol = '../output_samples/ascii/'

# import readprobes function from fluidfoam package
probes_locU, timeU, u = readprobes(sol, time_name = 'mergeTime', name = 'U')
probes_locP, timeP, p = readprobes(sol, time_name = 'mergeTime', name = 'p')
```

```
Reading file ../output_samples/ascii/postProcessing/probes/0/U
4 probes over 10 timesteps
Reading file ../output_samples/ascii/postProcessing/probes/0.2/U
4 probes over 6 timesteps
Reading file ../output_samples/ascii/postProcessing/probes/0/p
1 probes over 10 timesteps
Reading file ../output_samples/ascii/postProcessing/probes/0.2/p
1 probes over 6 timesteps
```

Now plots the pressure and y velocity for the first probe

```
import matplotlib.pyplot as plt

plt.figure()

plt.plot(timeU, u[:, 0, 1])
plt.plot(timeP, p[:, 0])

# Setting axis labels
plt.xlabel('t (s)')

# add grid and legend
plt.grid()
plt.legend(["Uy", "p"])

# show
plt.show()
```



Total running time of the script: (0 minutes 0.129 seconds)

Time series of OpenFoam scalar field

This example reads and plots a time series of an OpenFoam scalar field

Gets the time directories

Note: Tries if directory is a number and adds it in the time array

```
import os
import numpy as np

sol = '../output_samples/box/'
dir_list = os.listdir(sol)
time_list = []

for directory in dir_list:
    try:
        float(directory)
```

(continues on next page)

(continued from previous page)

```
        time_list.append(directory)
    except:
        pass
time_list.sort(key=float)
time_list=np.array(time_list)
```

Reads a scalar value at a given position for different times

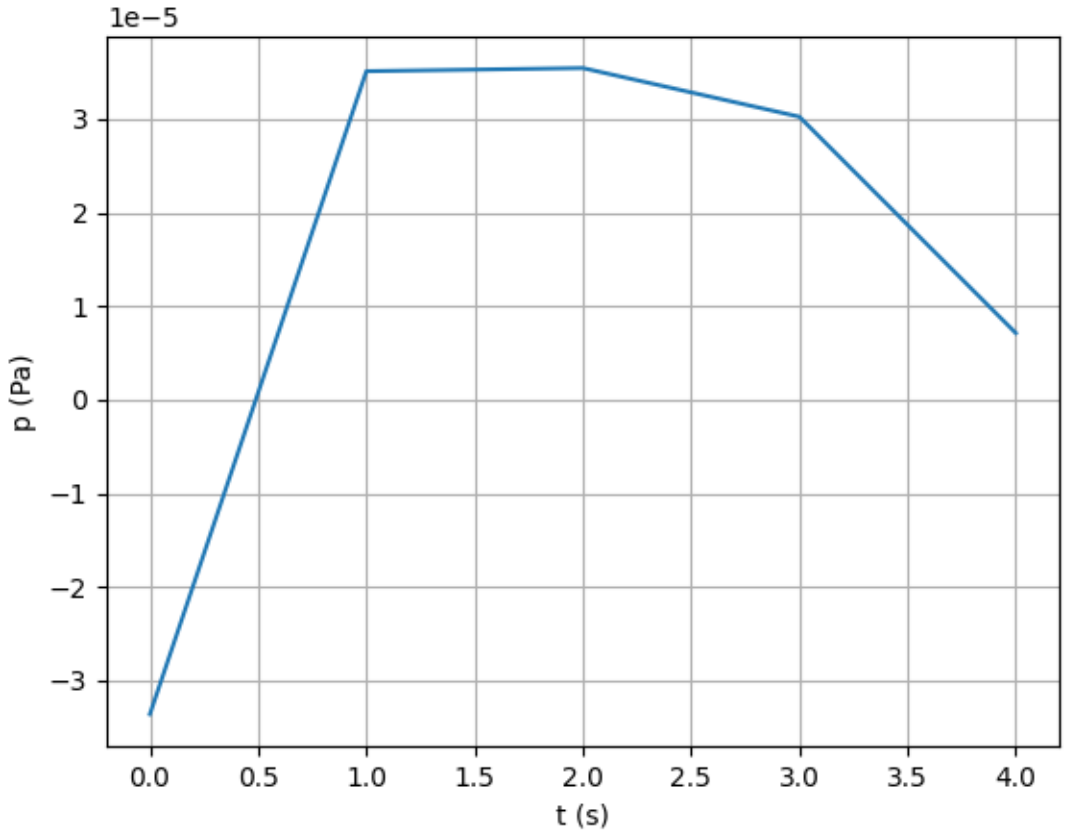
Note: It reads the scalar field p at position 20 and stores it in the numpy array time_series

```
# import readvector function from fluidfoam package
from fluidfoam import readscalar

sol = '../output_samples/box/'

time_series = np.empty(0)

for timename in time_list:
    p = readscalar(sol, timename, 'p')
    time_series = np.append(time_series, p[20])
```

```
Reading file ../output_samples/box/0/p
Reading file ../output_samples/box/1/p
Reading file ../output_samples/box/2/p
Reading file ../output_samples/box/3/p
Reading file ../output_samples/box/4/p
```

Now plots the time series

```
import matplotlib.pyplot as plt

plt.figure()

# Converts strings to float for plot
time_list = [float(i) for i in time_list]
plt.plot(time_list, time_series)

# Setting axis labels
plt.xlabel('t (s)')
plt.ylabel('p (Pa)')

# add grid
plt.grid()
```



Total running time of the script: (0 minutes 0.152 seconds)

Time series of postProcessing coefficient force

This example reads and plots one postProcessing coefficient force and compute and plot the fft of the lift coefficient force

Read the postProcessing files

Note: In this example it reads one postProcessing file for the statistically steady state of the flow around a rectangular cylinder.

```
from fluidfoam.readpostpro import readforce
import numpy as np

# ***************Selection of case, repository and parameters*************** #
rep = '../output_samples/'
case = 'ascii'

time = '100'                        # Simulation start time for fft
```

(continues on next page)

(continued from previous page)

```
force_file_name = 'coefficient'     # Variable on which fft is applied
index_Cl = 2                        # Lift force
index_Cd = 1                        # Drag force
n0 = 0                              # Start point for the fft
nfft = 500                          # Number of point for the fft
calculdeltat = 0.01                 # Delta t used in the calculation
fftdeltat = 0.2                     # Sampling Delta t for the fft, note that Tfft =␣
↪nfft * fftdeltat =< Ttot = endTime - time
timestart = 0                       # Start time for sampling from time
display_temp = True

# ************************************************************************** #

# reading Data
force = readforce(rep + case, namepatch = 'forces', time_name = time, name = force_file_
↪name)
timevec = np.zeros(len(force))
varC = np.zeros((len(force),2))
for i in range(len(force)):
    timevec[i] = force[i,0]
    varC[i,0] = force[i,index_Cl]
    varC[i,1] = force[i,index_Cd]

# Sampling
i = 0

varCi = np.zeros((nfft,2))
vart = np.zeros(nfft)
while i < nfft:
    for k in range(2):
        varCi[i,k] = varC[int(i*fftdeltat/calculdeltat+timestart/calculdeltat), k]
    vart[i] = timevec[int(i*fftdeltat/calculdeltat+timestart/calculdeltat)]
    i += 1

# fft calculation
y1 = np.zeros((nfft,2),dtype = np.complex128)
for k in range(2):
# Division by nfft to correct the spectral amplitude
    y1[:, k] = np.fft.fft(varCi[n0:n0+nfft,k] - np.mean(varCi[n0:n0+nfft,k]))/nfft
```

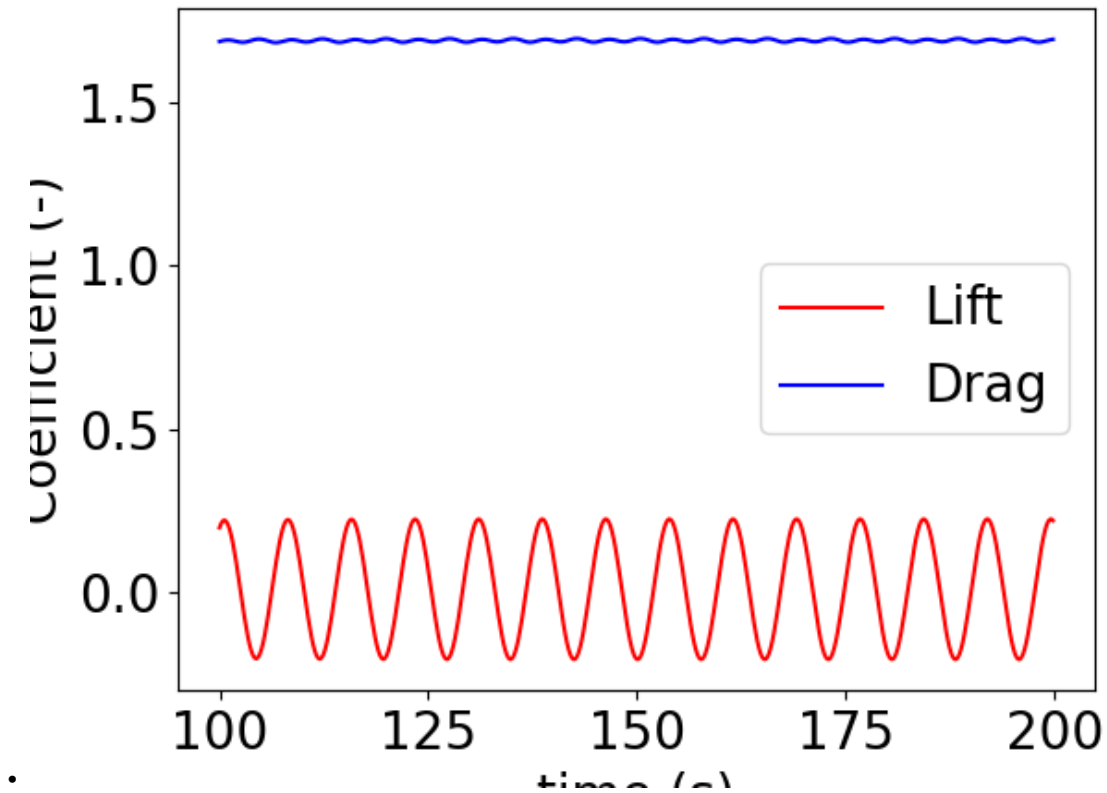
Now plots the lift and drag coefficient forces

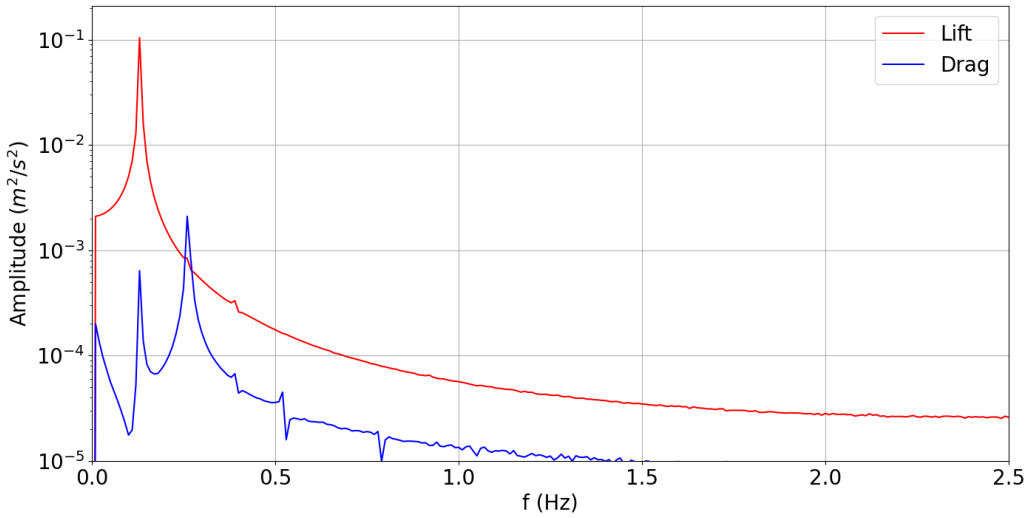
```
import matplotlib.pyplot as plt

plt.rcParams.update({'font.size': 20})
# Temporal plot section
colorList = ['r', 'b']
labelList = ['Lift', 'Drag']
if display_temp:
    for k in range(2):
```

(continues on next page)

(continued from previous page)

```
        plt.plot(vart[n0:n0+nfft], varCi[n0:n0+nfft,k],
             color = colorList[k], label = labelList[k])
    plt.legend(loc='best')
    plt.xlabel('time (s)')
    plt.ylabel('Coefficient (-)')
    plt.show()

# frequency plot section
f = np.arange(nfft)*1/(fftdeltat*nfft)

plt.figure(figsize=(16, 8))
for k in range(2):
    plt.semilogy(f[n0:n0+nfft], abs(y1[n0:n0+nfft,k]), color = colorList[k], label =␣
↪labelList[k])

plt.axis([0.0, 1/(2*fftdeltat), 1e-5,  2*np.max(abs(y1[n0:n0+nfft]))])
plt.grid()
plt.legend(loc='upper right')
plt.xlabel(r'f (Hz)')
plt.ylabel(r'Amplitude ($m^2/s^2$)')
plt.show()
```





-

Total running time of the script: (0 minutes 0.709 seconds)

10.7.2 Examples for structured mesh

This gallery consists of examples for structured mesh.

10.7.3 Examples for unstructured mesh

This gallery consists of examples for unstructured mesh.

10.7.4 Advanced examples

This gallery consists of advanced examples.

Examples for structured mesh

This gallery consists of examples for structured mesh.

Spatially averaged profile

This example reads and plots a spatially averaged profile of the first component of an OpenFoam vector field from a structured mesh

First reads the mesh

Note: It reads the mesh coordinates for a structured mesh (argument True) and stores them in variables x, y and z

```
# import readmesh function from fluidfoam package
from fluidfoam import readmesh

sol = '../../output_samples/box/'

x, y, z = readmesh(sol, structured=True)
```

```
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
```

Reads a vector field

Note: It reads a vector field from a structured mesh and stores it in vel variable

```
# import readvector function from fluidfoam package
from fluidfoam import readvector

timename = '0'
vel = readvector(sol, timename, 'U', structured=True)
```

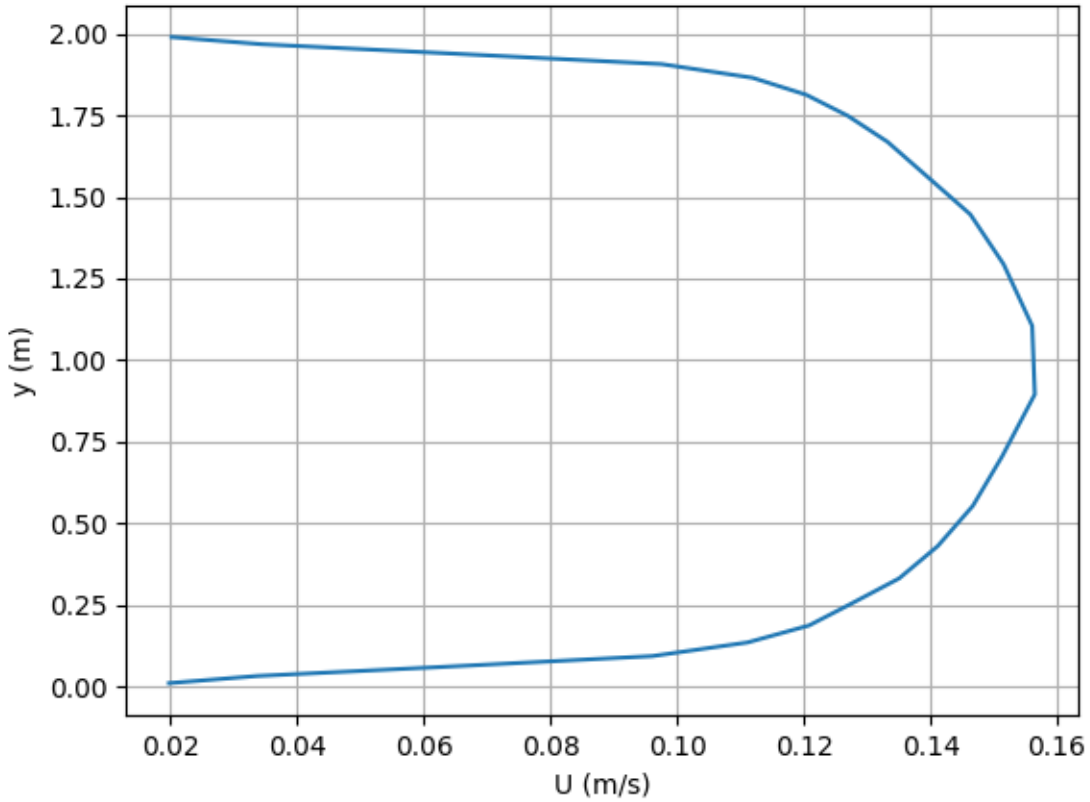
```
Reading file ../../output_samples/box/0/U
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
```

Averaging along x and z axis (1 and 3)

```
import numpy as np

vel_averaged = np.mean(np.mean(vel, 3), 1)
```

Now plots the profile of the averaged first velocity component

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(vel_averaged[0], y[0, :, 0])

#Setting axis labels
plt.xlabel('U (m/s)')
plt.ylabel('y (m)')

# add grid
plt.grid()
```



Total running time of the script: (0 minutes 1.602 seconds)

Contour from a slice in domain

This example reads and plots a contour of the first component of an OpenFoam vector field from a structured mesh

First reads the mesh and print the shape/size of the mesh

Note: It reads the mesh coordinates for a structured mesh (argument True) and stores them in variables x, y and z

```
# import readmesh function from fluidfoam package
from fluidfoam import readmesh

sol = '../../output_samples/box/'

x, y, z = readmesh(sol, structured=True)

nx, ny, nz = x.shape
print("Nx = ", nx, "Ny = ", ny, "Nz = ", nz)
```

```
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
Nx =  20 Ny =  24 Nz =  15
```

Reads a vector field

Note: It reads a vector field from a structured mesh and stores it in vel variable

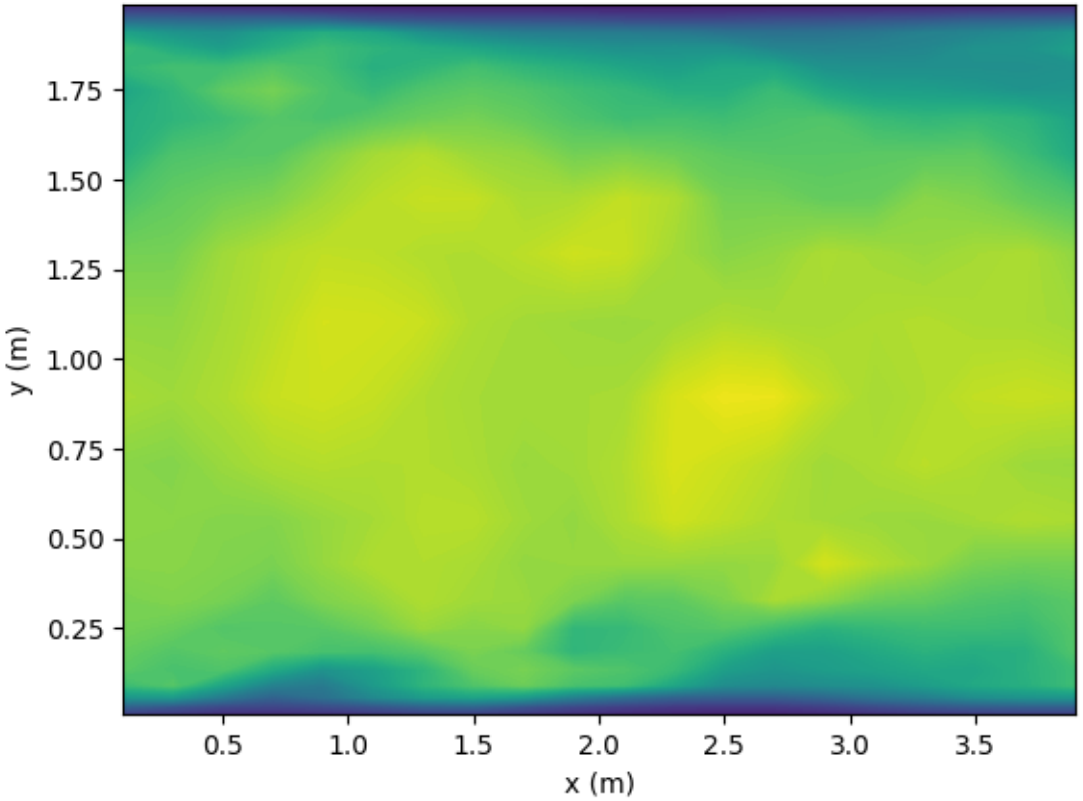
```
# import readvector function from fluidfoam package
from fluidfoam import readvector

timename = '0'
vel = readvector(sol, timename, 'U', structured=True)
```

```
Reading file ../../output_samples/box/0/U
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
```

Now plots the contour of the first velocity component at a given z position

Note: Here the position z is the middle (// is used to have an integer)

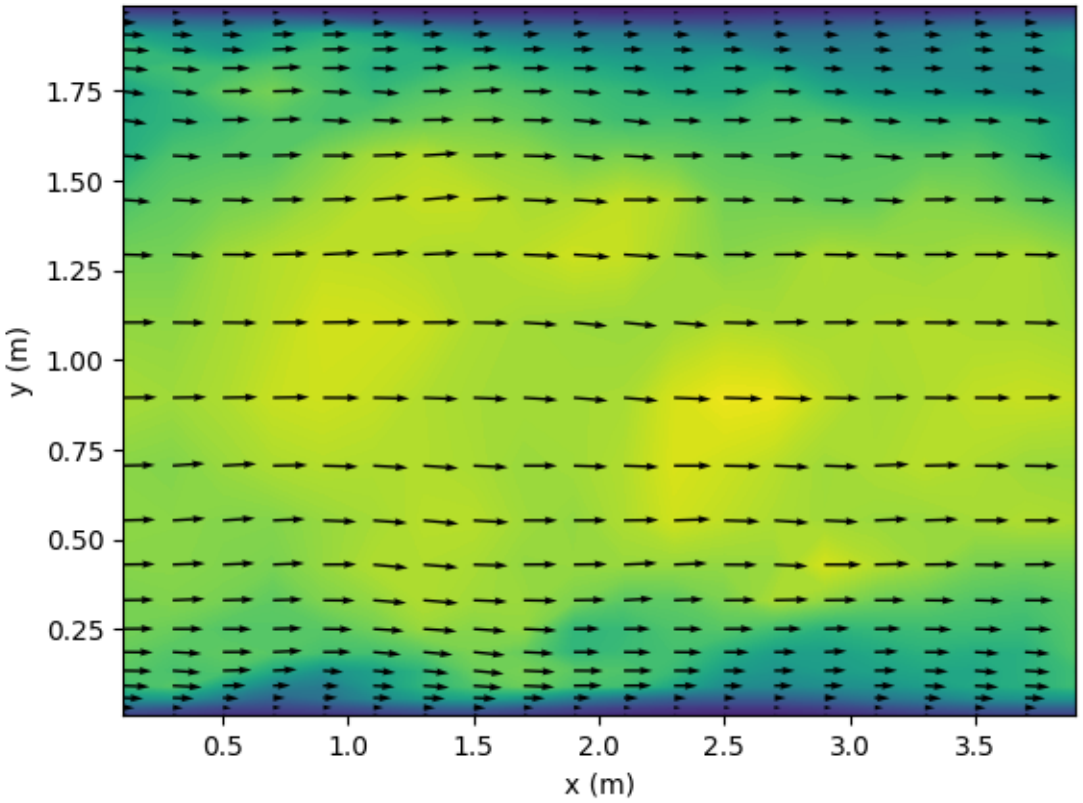
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()
levels = np.arange(0, 0.178, 0.001)
plt.contourf(x[:, :, nz//2], y[:, :, nz//2], vel[0, :, :, nz//2],
             levels=levels)

# Setting axis labels
plt.xlabel('x (m)')
plt.ylabel('y (m)')
```




```
Text(33.972222222222214, 0.5, 'y (m)')
```

Now add on the same plot the velocity vectors

```
plt.figure()
plt.contourf(x[:, :, nz//2], y[:, :, nz//2], vel[0, :, :, nz//2],
             levels=levels)

# Setting axis labels
plt.xlabel('x (m)')
plt.ylabel('y (m)')

plt.quiver(x[:, :, nz//2], y[:, :, nz//2],
           vel[0, :, :, nz//2], vel[1, :, :, nz//2])
```




```
<matplotlib.quiver.Quiver object at 0x7f00e22268a0>
```

Total running time of the script: (0 minutes 1.755 seconds)

Contour and scatter from a boundary/patch

This example reads and plots the first component of an OpenFoam vector field from a boundary (patch) of a structured mesh

First reads the mesh and print the shape/size of the mesh boundary

Note: It reads the mesh coordinates of a boundary for a structured mesh and stores them in variables x, y and z

```
# import readmesh function from fluidfoam package
from fluidfoam import readmesh


sol = "../../output_samples/box/"

x, y, z = readmesh(path=sol, structured=True, boundary="topWall")

nface = x.shape
print("Boundary shape = ", nface)
```

```
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/boundary
Boundary shape =  (20, 1, 15)
```

Reads a vector field

Note: It reads a vector field of a boundary from a structured mesh and stores it in vel variable

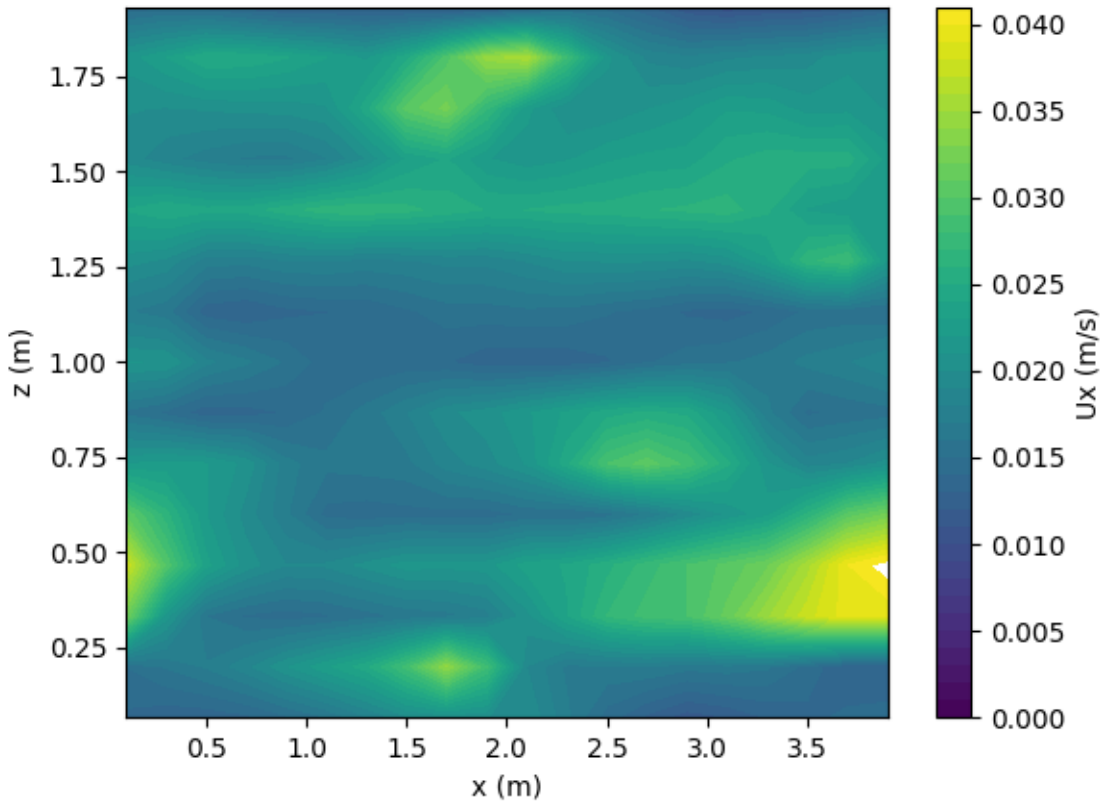
```
# import readvector function from fluidfoam package
from fluidfoam import readvector

timename = "0"
vel = readvector(sol, timename, "U", structured=True, boundary="topWall")
```

```
Reading file ../../output_samples/box/0/U
Warning : No data on boundary/patch
Using the values of the nearest cells
Reading file ../../output_samples/box//constant/polyMesh/boundary
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/boundary
```

Now plots the contour of the first velocity component on the topWall boundary

Note: Here the topWall boundary is in (x, z) plane

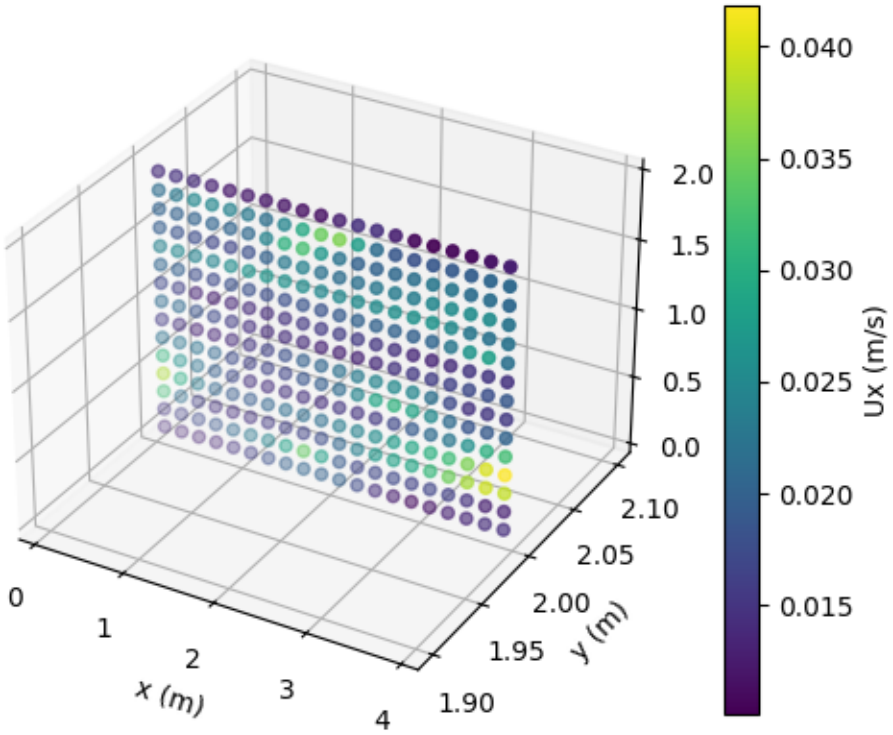
```
import matplotlib.pyplot as plt
import numpy as np

plt.figure()

levels = np.arange(0, np.max(vel[0]), 0.001)
ax = plt.contourf(x[:, 0, :], z[:, 0, :], vel[0, :, 0, :], levels=levels)
cbar = plt.colorbar(ax)
cbar.set_label("Ux (m/s)")

# Setting axis labels
plt.xlabel("x (m)")
plt.ylabel("z (m)")
```




```
Text(33.972222222222214, 0.5, 'z (m)')
```

If you don't know the plane

```
fig = plt.figure()
ax = fig.add_subplot(111, projection="3d")

ax3d = ax.scatter(x, y, z, c=vel[0, :, :, :])

# Setting axis labels
ax.set_xlabel("x (m)")
ax.set_ylabel("y (m)")
ax.set_zlabel("z (m)")
cbar = plt.colorbar(ax3d)
cbar.set_label("Ux (m/s)")
```



Total running time of the script: (0 minutes 0.645 seconds)

Examples for unstructured mesh

This gallery consists of examples for unstructured mesh.

Contour from an unstructured mesh (no interpolation)

This example reads and plots a contour of an OpenFoam vector field from an unstructured mesh by triangulation WITHOUT interpolation on a structured grid

Reads the mesh

Note: It reads the mesh coordinates and stores them in variables x, y and z

```
# import readmesh function from fluidfoam package
from fluidfoam import readmesh

sol = '../../output_samples/pipeline/'

x, y, z = readmesh(sol)
```

```
Reading file ../../output_samples/pipeline//constant/polyMesh/owner
Reading file ../../output_samples/pipeline//constant/polyMesh/faces
Reading file ../../output_samples/pipeline//constant/polyMesh/points
Reading file ../../output_samples/pipeline//constant/polyMesh/neighbour
```

Reads vector and scalar field

Note: It reads volume scalar field from an unstructured mesh and stores it

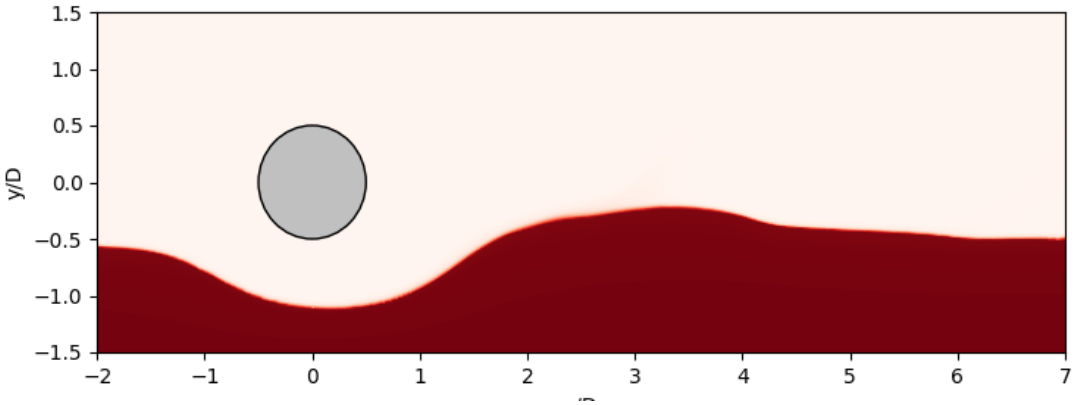
```
# import readvector and readscalar functions from fluidfoam package
from fluidfoam import readvector, readscalar

timename = '25'
vel = readvector(sol, timename, 'Ub')
alpha = readscalar(sol, timename, 'alpha')
```

```
Reading file ../../output_samples/pipeline/25/Ub
Reading file ../../output_samples/pipeline/25/alpha
```

Plots the contour of the volscalarfield alpha and a patch

Note: The scalar field alpha reprensents the concentration of sediment in in a 2D two-phase flow simulation of erosion below a pipeline

```
import numpy as np
import matplotlib.pyplot as plt

# Define plot parameters
fig, ax = plt.subplots(figsize=(8.5, 3), dpi=100)
plt.rcParams.update({'font.size': 10})
plt.xlabel('x/D')
plt.ylabel('y/D')
d = 0.05
# Add a cuircular patch representing the pipeline
circle = plt.Circle((0, 0), radius=0.5, fc='silver', zorder=10,
                    edgecolor='k')
plt.gca().add_patch(circle)

# Plots the contour of sediment concentration
levels = np.arange(0.0, 0.63, 0.001)

plt.tricontourf(x/d, y/d, alpha, cmap=plt.cm.Reds, levels=levels)

ax.set(xlim=(-2, 7), ylim=(-1.5, 1.5))
plt.show()
```



Total running time of the script: (0 minutes 14.213 seconds)

Contour and streamlines from an unstructured mesh

This example reads and plots a contour of an OpenFoam vector field from an unstructured mesh by interpolation on a structured grid

Reads the mesh

Note: It reads the mesh coordinates and stores them in variables x, y and z

```
# import readmesh function from fluidfoam package
from fluidfoam import readmesh

sol = '../../output_samples/pipeline/'

x, y, z = readmesh(sol)
```

```
Reading file ../../output_samples/pipeline//constant/polyMesh/owner
Reading file ../../output_samples/pipeline//constant/polyMesh/faces
Reading file ../../output_samples/pipeline//constant/polyMesh/points
Reading file ../../output_samples/pipeline//constant/polyMesh/neighbour
```

Reads vector and scalar field

Note: It reads vector and scalar field from an unstructured mesh and stores them in vel and alpha variables

```
# import readvector and readscalar functions from fluidfoam package
from fluidfoam import readvector, readscalar

timename = '25'
vel = readvector(sol, timename, 'Ub')
alpha = readscalar(sol, timename, 'alpha')
```

```
Reading file ../../output_samples/pipeline/25/Ub
Reading file ../../output_samples/pipeline/25/alpha
```

Interpolate the fields on a structured grid

Note: The vector and scalar fields are interpolated on a specified structured grid

```
import numpy as np
from scipy.interpolate import griddata

# Number of division for linear interpolation
```

(continues on next page)

(continued from previous page)

```
ngridx = 500
ngridy = 180

# Interpolation grid dimensions
xinterpmin = -0.1
xinterpmax = 0.35
yinterpmin = -0.075
yinterpmax = 0.075

# Interpolation grid
xi = np.linspace(xinterpmin, xinterpmax, ngridx)
yi = np.linspace(yinterpmin, yinterpmax, ngridy)

# Structured grid creation
xinterp, yinterp = np.meshgrid(xi, yi)

# Interpolation of scalra fields and vector field components
alpha_i = griddata((x, y), alpha, (xinterp, yinterp), method='linear')
velx_i = griddata((x, y), vel[0, :], (xinterp, yinterp), method='linear')
vely_i = griddata((x, y), vel[1, :], (xinterp, yinterp), method='linear')
```

Plots the contour of the interpolted scalarfield alpha, streamlines and a patch

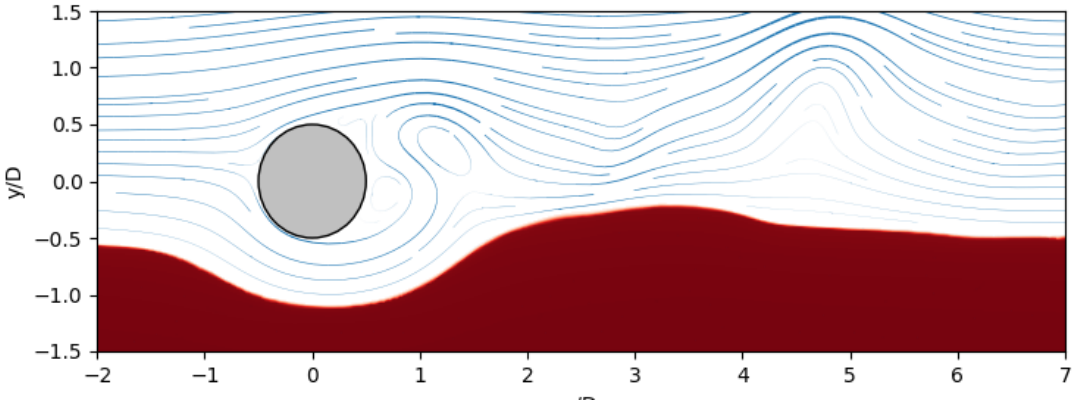
Note: The scalar field alpha reprensents the concentration of sediment in in a 2D two-phase flow simulation of erosion below a pipeline

```
import matplotlib.pyplot as plt

# Define plot parameters
fig = plt.figure(figsize=(8.5, 3), dpi=100)
plt.rcParams.update({'font.size': 10})
plt.xlabel('x/D')
plt.ylabel('y/D')
d = 0.05

# Add a cuircular patch representing the pipeline
circle = plt.Circle((0, 0), radius=0.5, fc='silver', zorder=10,
                    edgecolor='k')
plt.gca().add_patch(circle)

# Plots the contour of sediment concentration
levels = np.arange(0.1, 0.63, 0.001)
plt.contourf(xi/d, yi/d, alpha_i, cmap=plt.cm.Reds, levels=levels)

# Calculation of the streamline width as a function of the velociy magnitude
vel_i = np.sqrt(velx_i**2 + vely_i**2)
lw = pow(vel_i, 1.5)/vel_i.max()

# Plots the streamlines
```

(continues on next page)

(continued from previous page)

```
plt.streamplot(xi/d, yi/d, velx_i, vely_i, color='C0', density=[2, 1],
               linewidth=lw, arrowsize=0.05)
```




```
<matplotlib.streamplot.StreamplotSet object at 0x7f00dfb03860>
```

Total running time of the script: (0 minutes 17.583 seconds)

Get the cell centroids and cell volumes of a given box

This example shows how to extract the cell volumes inside a given box

First import the getVolumes function and other relevant libraries

```
#import the class MeshVisu, numpy library and getVolumes function
from fluidfoam import MeshVisu
from fluidfoam.readof import getVolumes
import numpy as np
# path to the simulation to load
path = '../../output_samples/pipeline'

# Load mesh and create an object called myMesh
# The box by default is equal to the mesh dimension
myMesh = MeshVisu( path =  path)
```

```
Reading file ../../output_samples/pipeline/constant/polyMesh/faces
Reading file ../../output_samples/pipeline/constant/polyMesh/points
Box set to mesh size:
 (minx, miny, minz) = (-0.75, -0.1, -0.001)
 (maxx, maxy, maxz) = (1.0, 0.205, 0.0)
```

We are going to extract the cell volumes and cell centroids of two given boxes

```
#tuple of box's dimension: ((xmin, ymin, zmin), (xmax, ymax, zmax))
mybox_A = ((0, 0, -1), (0.03, 0.03, 1))
mybox_B = ((0.022, 0.0221, -1), (0.0274, 0.0275, 1))

#getVolumes function returns arrays containing the centroids and volume of the
#cells inside boxes A and B
centroidList_box_A,vol_box_A = getVolumes( path =  path, box = mybox_A)
centroidList_box_B,vol_box_B = getVolumes( path =  path, box = mybox_B)

vol_box_A_total = sum(vol_box_A)
vol_box_B_total = sum(vol_box_B)

print("Total cell volume inside the box A:", vol_box_A_total)
print("Total cell volume  inside the box B:", vol_box_B_total)
```

```
Reading file ../../output_samples/pipeline/constant/polyMesh/owner
Reading file ../../output_samples/pipeline/constant/polyMesh/faces
Reading file ../../output_samples/pipeline/constant/polyMesh/points
Reading file ../../output_samples/pipeline/constant/polyMesh/neighbour
Reading file ../../output_samples/pipeline/constant/polyMesh/owner
Reading file ../../output_samples/pipeline/constant/polyMesh/faces
Reading file ../../output_samples/pipeline/constant/polyMesh/points
Reading file ../../output_samples/pipeline/constant/polyMesh/neighbour
Total cell volume inside the box A: 4.0746335288354696e-07
Total cell volume  inside the box B: 2735.079298833823
```

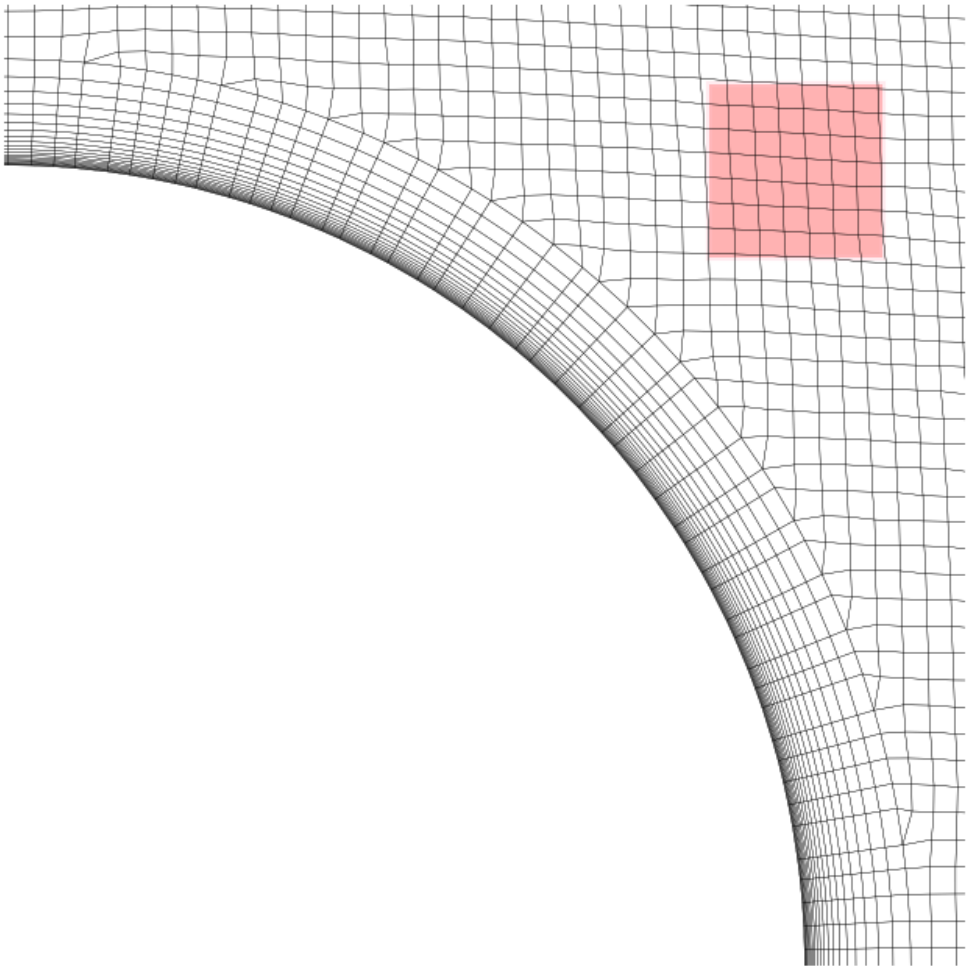
Visualisation of the two boxes

```
myMesh.update_box(mybox_A)

import matplotlib.pyplot as plt
from  matplotlib.collections import LineCollection
from matplotlib.patches import Rectangle

fig, ax = plt.subplots( figsize = (8,8))

# create a collection with edges and print it
ln_coll = LineCollection(myMesh.get_all_edgesInBox(), linewidths = 0.20, colors = 'black
↪')
ax.add_collection(ln_coll, autolim=True)

# Set box dimensions as the figures's limits
ax.set_xlim(myMesh.get_xlim())
ax.set_ylim(myMesh.get_ylim())

# Add rectangle to plot, which corresponds to box B
ax.add_patch(Rectangle((mybox_B[0][0], mybox_B[0][1]), mybox_B[1][0]-mybox_B[0][0],␣
↪mybox_B[1][1]-mybox_B[0][1],
             edgecolor = 'pink',
```

(continues on next page)

(continued from previous page)

```
            facecolor = 'red',
            alpha=0.3,
            fill=True,
            lw=3))


# to avoid distorting the mesh:
ax.set_aspect('equal')

# to don't print axis:
ax.axis('off')

# to save the figure in pdf or svg format, uncomment one of the following two lines:
# plt.savefig('./myCylinderCellVolumes.pdf', dpi=fig.dpi, transparent = True, bbox_inches␣
↪= 'tight')
# plt.savefig('./myCylinderZomm.svg', dpi=fig.dpi, transparent = True, bbox_inches = 'tight
↪')
```



```
(0.0, 0.03, 0.0, 0.03)
```

Total running time of the script: (0 minutes 35.764 seconds)

Create vectorised visualisations of the mesh

This example shows how to use MeshVisu object to plot vectorised images of 2D planar meshes.

First create a visualisable mesh object with MeshVisu

Note: This class allows you to create a list of edges contained inside a box. This list of edges will then be ploted.

```
# import the class MeshVisu
from fluidfoam import MeshVisu

# path to the simulation to load
path = '../../output_samples/pipeline'

# Load mesh and create an object called myMesh
# The box by default is egal to the mesh dimension
myMesh = MeshVisu( path =  '../../output_samples/pipeline')
```

```
Reading file ../../output_samples/pipeline/constant/polyMesh/faces
Reading file ../../output_samples/pipeline/constant/polyMesh/points
Box set to mesh size:
 (minx, miny, minz) = (-0.75, -0.1, -0.001)
 (maxx, maxy, maxz) = (1.0, 0.205, 0.0)
```

Plot the whole mesh

```
import matplotlib.pyplot as plt
from  matplotlib.collections import LineCollection

# compute mesh aspect ratio:
xmin, xmax = myMesh.get_xlim()
ymin, ymax = myMesh.get_ylim()
AR = (ymax - ymin) / (xmax - xmin)

fig, ax = plt.subplots( figsize = (8,8*AR))
# create a collection with edges and print it
ln_coll = LineCollection(myMesh.get_all_edgesInBox(), linewidths = 0.25, colors = 'brown
↪')
ax.add_collection(ln_coll, autolim=True)

# impose the dimensions of the box as the limits of the figure
ax.set_xlim(myMesh.get_xlim())
ax.set_ylim(myMesh.get_ylim())

# to avoid distorting the mesh:
ax.set_aspect('equal')
```

(continues on next page)

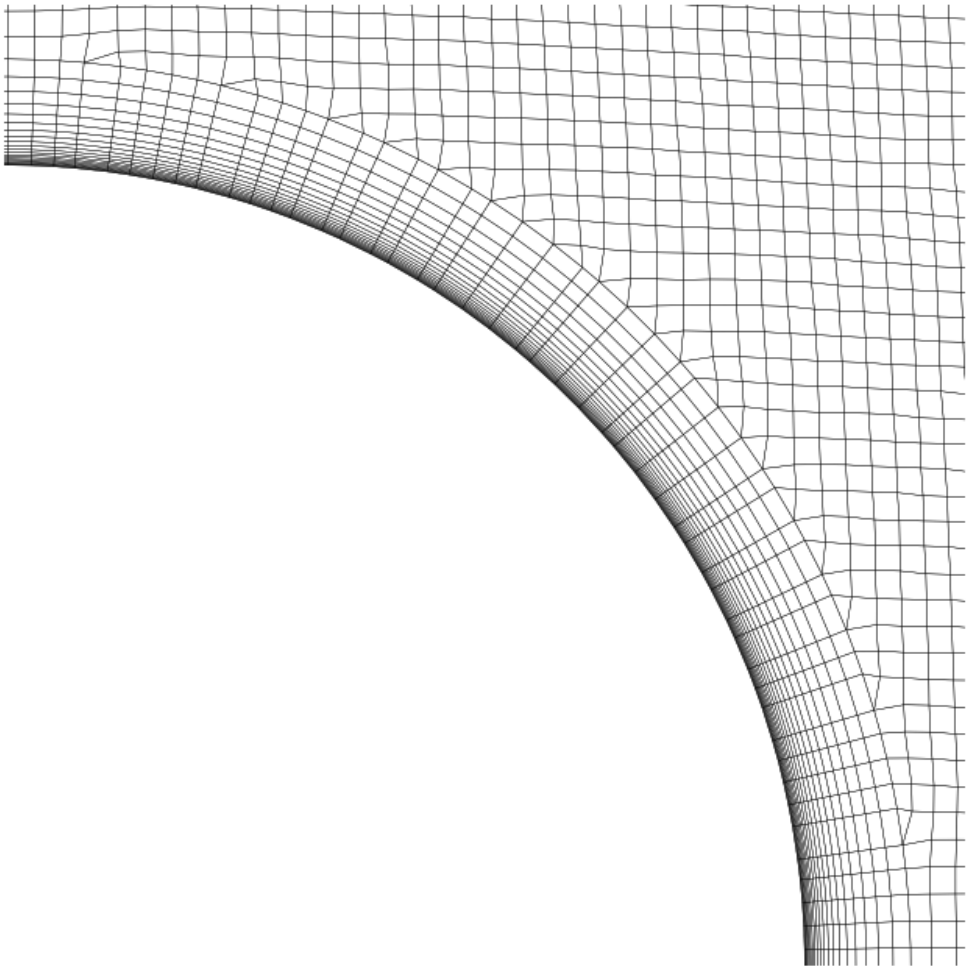
(continued from previous page)

```
# to don't print axis:
ax.axis('off')
```

```
(-0.75, 1.0, -0.1, 0.205)
```

Update the box to zoom on the cylinder and save figure

```
myMesh.update_box(((0, 0, -1), (0.03, 0.03, 1)))

fig, ax = plt.subplots( figsize = (8,8))
# create a collection with edges and print it
ln_coll = LineCollection(myMesh.get_all_edgesInBox(), linewidths = 0.25, colors = 'black
↪')
ax.add_collection(ln_coll, autolim=True)

# Set box dimensions as the figures's limits
ax.set_xlim(myMesh.get_xlim())
ax.set_ylim(myMesh.get_ylim())

# to avoid distorting the mesh:
ax.set_aspect('equal')

# to don't print axis:
ax.axis('off')

# to save the figure in pdf or svg format, uncomment one of the following two lines:
# plt.savefig('./myCylinderZomm.pdf', dpi=fig.dpi, transparent = True, bbox_inches = 'tight
↪')
# plt.savefig('./myCylinderZomm.svg', dpi=fig.dpi, transparent = True, bbox_inches = 'tight
↪')
```



```
(0.0, 0.03, 0.0, 0.03)
```

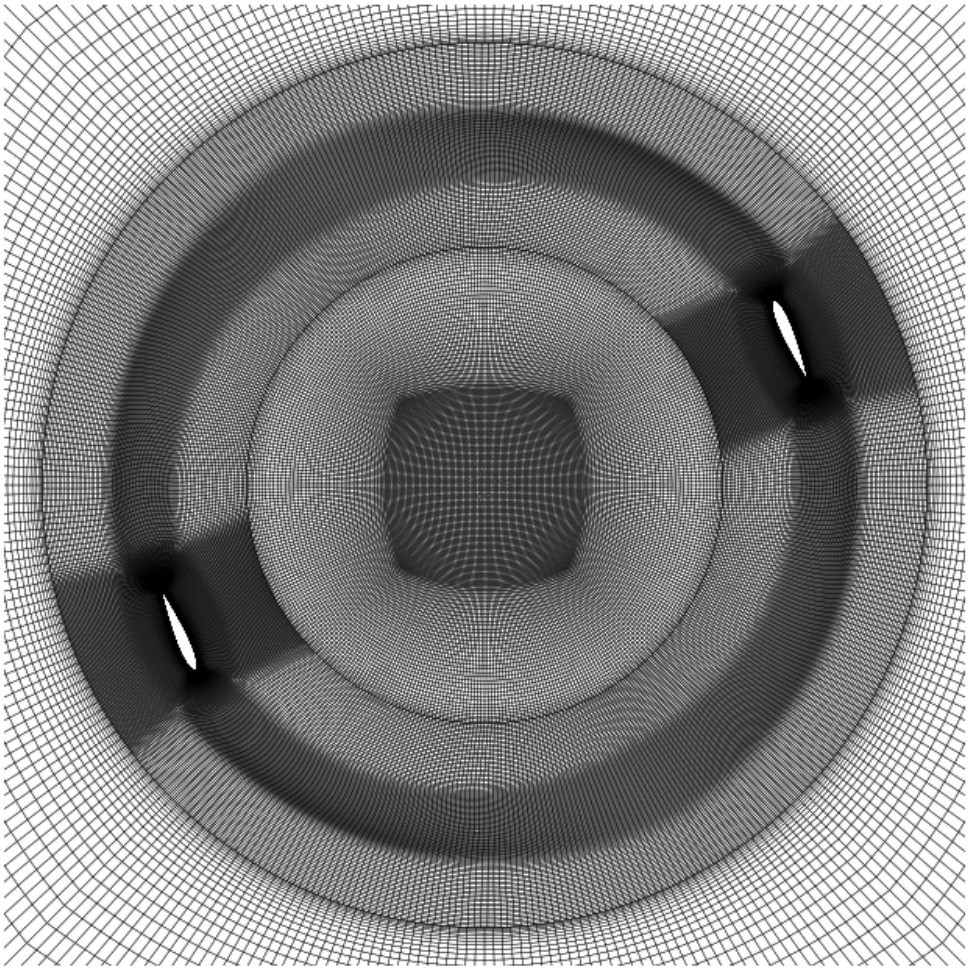
Visualisation of dynamic case in xz plane

```
# path to the simulation to load:
mypath =  '../../output_samples/darrieus'

# time folder for which you want to display the mesh:
mytime =  '0.1'

# plane in which the mesh is contained, either:
```

(continues on next page)

(continued from previous page)

```
# 'xy': the xy-plane of outgoing normal z (default value)
# 'xz': the xz-plane of outgoing normal -y
# 'yz': the yz-plane of outgoing normal x
myplane =  'xz'

# box to zoom in on for mesh display:
mybox = ((-1.2, -1, -1.2), (1.2, 1, 1.2))

# Load mesh and create an object called myOtherMesh:
myOtherMesh = MeshVisu(path = mypath, box = mybox, time_name = mytime, plane = myplane)

# The next line sets the thumbnail for the last figure in the gallery
# sphinx_gallery_thumbnail_number = -1
fig, ax = plt.subplots( figsize = (8,8))
# create a collection with edges and print it
ln_coll = LineCollection(myOtherMesh.get_all_edgesInBox(), linewidths = 0.25, colors =
↪'black')
ax.add_collection(ln_coll, autolim=True)

# Set box dimensions as the figures's limits
ax.set_xlim(myOtherMesh.get_xlim())
ax.set_ylim(myOtherMesh.get_zlim())

# to avoid distorting the mesh:
ax.set_aspect('equal')

# to don't print axis:
ax.axis('off')
```



```
Reading file ../../output_samples/darrieus/constant/polyMesh/faces
Reading file ../../output_samples/darrieus/0.1/polyMesh/points

(-1.2, 1.2, -1.2, 1.2)
```

Total running time of the script: (0 minutes 41.128 seconds)

Advanced examples

This gallery consists of advanced examples.

Create and plot boundaryData for DFSEM from a 1D RANS simulation

This example shows how to create a boundary data for turbulentDFSEM inlet boundary condition. In addition, the script also plots the profiles for verification.

```
from fluidfoam import create1dprofilDFSEM, read1dprofilDFSEM
import os

basepath = "../../output_samples//DFSEM/"

case3d = "3D/"
case1d = "1D/"

sol1d = os.path.join(basepath, case1d)
sol3d = os.path.join(basepath, case3d)

boundary_name = "inlet/"
axis = "Y"

create1dprofilDFSEM(sol1d, sol3d, boundary_name, "200", axis,
                          "U","k","omega","turbulenceProperties:R","Y")
# if turbulenceProperties:R does not exist type:
# "pimpleFoam -postProcess -func R -time 200"
# in a terminal

Y, U, L, R, ny = read1dprofilDFSEM(sol3d, boundary_name, "0", axis)

import matplotlib.pyplot as plt

dummy, axarr = plt.subplots(1, 3, sharey=True)
axarr[0].set_ylabel("Y (m)")

axarr[0].plot(U[:], Y)
axarr[0].set_title("U")
axarr[0].set_xlabel("U (m/s)")

axarr[1].plot(L[:], Y)
axarr[1].set_title("L")
axarr[1].set_xlabel("L (m)")

axarr[2].plot(R[:,1], Y)
axarr[2].set_title("R")
axarr[2].set_xlabel(r"R ($m^2/s^2$)")

plt.show()
```

OpenFoamSimu to post-process simulation

This example shows how to organize the results of a simulation into an OpenFoamSimu object that contains all the results of one simulation.

First create a simulation object with OpenFoamSimu

Note: This class allows you to create an object associated to a simulation

```
# import the class OpenFoamSimu
from fluidfoam import OpenFoamSimu

#path were all simulations are located
path = '../../output_samples'
#Name of the simulation to load, if not given, the program lists all simulations
#located in path and ask you to choose which one to load
simu = 'box'
#time step to load, if not given, load last time step
timeStep = '4'

#Load simulation and create an object called mySimu that contain the results of
#the simulation, structured=True indicates that the mesh is structured
mySimu = OpenFoamSimu(path=path, simu=simu, timeStep=timeStep, structured=True)
# .. note:: Each file saved at timeStep of the simulation  are
#           automatically loaded as variables of object mySimu.
#           You can know what variables have been loaded using function keys() of
#           the class

mySimu.keys()

# .. note:: function keys() indicates that variables named U, nut and p have
#           been loaded. You can access them simply by typing for example mySimu.U
#           for the velocity field

mySimu.U
```

```
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
Reading file ../../output_samples/box/4/nut
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
Reading file ../../output_samples/box/4/p
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
```

(continues on next page)

(continued from previous page)

```
Reading file ../../output_samples/box/4/U
Reading file ../../output_samples/box//constant/polyMesh/owner
Reading file ../../output_samples/box//constant/polyMesh/faces
Reading file ../../output_samples/box//constant/polyMesh/points
Reading file ../../output_samples/box//constant/polyMesh/neighbour
Loaded available variables are :
['nut', 'p', 'U']

array([[[[ 3.18992e-02,  1.79405e-02,  1.85248e-02, ...,  2.07018e-02,
           2.18473e-02,  4.01484e-02],
         [ 5.45512e-02,  2.20373e-02,  3.37097e-02, ...,  3.63537e-02,
           3.65411e-02,  7.03408e-02],
         [ 9.64050e-02,  5.14858e-02,  6.60540e-02, ...,  7.79280e-02,
           8.04698e-02,  1.16315e-01],
         ...,
         [ 3.96455e-02,  5.08090e-02,  1.15117e-01, ...,  6.56267e-02,
           7.19338e-02,  4.01245e-02],
         [ 2.11214e-02,  3.61928e-02,  7.55611e-02, ...,  3.47472e-02,
           3.71378e-02,  2.26417e-02],
         [ 1.20526e-02,  1.23910e-02,  4.44112e-02, ...,  2.04318e-02,
           2.16031e-02,  1.34423e-02]],

        [[ 3.26646e-02,  2.00377e-02,  1.94394e-02, ...,  1.87730e-02,
           2.49159e-02,  3.52195e-02],
         [ 5.80944e-02,  2.46078e-02,  3.17509e-02, ...,  3.15628e-02,
           4.35643e-02,  6.52370e-02],
         [ 1.05022e-01,  6.52484e-02,  5.84404e-02, ...,  6.88878e-02,
           8.10161e-02,  1.19481e-01],
         ...,
         [ 3.87072e-02,  4.75148e-02,  1.17276e-01, ...,  6.59377e-02,
           7.26811e-02,  4.35011e-02],
         [ 2.10653e-02,  2.99974e-02,  7.03449e-02, ...,  3.41233e-02,
           3.69766e-02,  2.45663e-02],
         [ 1.23754e-02,  1.43732e-02,  3.77137e-02, ...,  2.01094e-02,
           2.26093e-02,  1.46607e-02]],

        [[ 3.16753e-02,  2.09653e-02,  1.87914e-02, ...,  1.90877e-02,
           2.41407e-02,  2.70096e-02],
         [ 5.62772e-02,  3.82019e-02,  3.03069e-02, ...,  3.11023e-02,
           4.60455e-02,  4.49955e-02],
         [ 1.04903e-01,  5.13360e-02,  5.04449e-02, ...,  5.33476e-02,
           8.86924e-02,  1.04078e-01],
         ...,
         [ 3.72512e-02,  5.42535e-02,  1.14961e-01, ...,  6.76110e-02,
           7.31055e-02,  4.58660e-02],
         [ 2.10945e-02,  2.67728e-02,  5.22082e-02, ...,  3.38472e-02,
           4.01594e-02,  2.47811e-02],
         [ 1.28640e-02,  1.59485e-02,  2.52787e-02, ...,  1.88499e-02,
           2.29339e-02,  1.42788e-02]],

        ...,
```

(continues on next page)

(continued from previous page)

```
       [[ 2.10110e-02,  1.58239e-02,  1.71158e-02, ...,  1.93919e-02,
          2.18244e-02,  3.20522e-02],
        [ 3.32812e-02,  2.23170e-02,  3.11740e-02, ...,  3.43311e-02,
          4.73305e-02,  5.14514e-02],
        [ 5.47667e-02,  4.05091e-02,  6.10295e-02, ...,  6.81035e-02,
          8.04988e-02,  7.97360e-02],
        ...,
        [ 3.67845e-02,  5.27457e-02,  9.30992e-02, ...,  7.62478e-02,
          6.10209e-02,  3.17175e-02],
        [ 2.09944e-02,  2.74006e-02,  6.02779e-02, ...,  4.04399e-02,
          2.75262e-02,  1.82754e-02],
        [ 1.25483e-02,  1.55108e-02,  3.68839e-02, ...,  2.33335e-02,
          1.96690e-02,  1.04692e-02]],

       [[ 2.45636e-02,  1.52846e-02,  1.74154e-02, ...,  2.17054e-02,
          2.08747e-02,  3.60227e-02],
        [ 3.92183e-02,  2.07494e-02,  3.30393e-02, ...,  3.69546e-02,
          3.77244e-02,  6.03639e-02],
        [ 6.25909e-02,  3.91597e-02,  7.29379e-02, ...,  6.51149e-02,
          8.80845e-02,  8.50655e-02],
        ...,
        [ 3.75764e-02,  5.39175e-02,  1.00008e-01, ...,  7.48765e-02,
          6.36199e-02,  3.31145e-02],
        [ 2.23479e-02,  4.69155e-02,  7.06649e-02, ...,  3.67162e-02,
          2.82184e-02,  1.93503e-02],
        [ 1.37299e-02,  1.44704e-02,  4.15440e-02, ...,  2.15754e-02,
          1.95848e-02,  1.13547e-02]],

       [[ 2.85475e-02,  1.59666e-02,  1.80183e-02, ...,  2.29304e-02,
          2.03935e-02,  3.87283e-02],
        [ 4.92924e-02,  2.54433e-02,  3.42915e-02, ...,  3.97380e-02,
          3.39281e-02,  6.79486e-02],
        [ 7.63345e-02,  3.72513e-02,  7.51616e-02, ...,  7.20266e-02,
          8.53194e-02,  1.04755e-01],
        ...,
        [ 3.76133e-02,  8.87271e-02,  1.09970e-01, ...,  6.65887e-02,
          6.62168e-02,  3.58228e-02],
        [ 2.19142e-02,  4.24531e-02,  7.37609e-02, ...,  3.47382e-02,
          3.56335e-02,  2.01614e-02],
        [ 1.29698e-02,  1.21604e-02,  4.37596e-02, ...,  1.97192e-02,
          1.98143e-02,  1.18009e-02]]],


      [[[ 7.84279e-04,  5.47106e-04, -8.21631e-04, ..., -1.11664e-04,
          2.08206e-04, -5.19400e-04],
        [ 1.39026e-03,  9.00218e-04, -1.76744e-03, ..., -5.17103e-04,
          4.37167e-04, -1.43493e-03],
        [ 1.46458e-03,  2.79096e-03, -4.10754e-03, ..., -1.51365e-03,
         -1.64057e-03, -4.25778e-03],
        ...,
        [-1.56645e-03, -4.01416e-03,  3.25295e-03, ..., -1.62787e-03,
          7.39715e-05,  1.78847e-04],
```

(continues on next page)

(continued from previous page)

```
        [-5.17724e-04, -9.69123e-04,  9.90531e-04, ..., -5.14880e-04,
         -1.14499e-04,  2.57391e-04],
        [-1.95333e-04, -5.23292e-04,  5.60509e-04, ..., -2.18410e-04,
         -4.90563e-05,  1.63185e-04]],

       [[ 4.44843e-04,  9.04593e-04, -1.15291e-03, ..., -4.36677e-04,
          5.50935e-04, -7.08876e-04],
        [ 7.25699e-04,  1.61511e-03, -1.91988e-03, ..., -7.77636e-04,
          1.13815e-03, -1.42614e-03],
        [ 9.08462e-04,  3.25245e-03, -4.42358e-03, ..., -1.65696e-03,
         -1.60247e-03, -3.86300e-03],
        ...,
        [-1.50284e-03, -2.00554e-03,  1.35983e-03, ..., -1.07225e-03,
          1.37933e-04, -4.74335e-04],
        [-4.18433e-04, -7.45759e-04,  4.91048e-04, ..., -4.87104e-04,
         -4.67593e-04, -8.59092e-05],
        [-1.31601e-04, -6.05762e-05,  2.79449e-04, ..., -3.00915e-04,
         -6.36434e-05, -1.90407e-05]],

       [[ 1.97729e-05,  8.08211e-04, -6.48533e-04, ..., -1.23010e-04,
          6.97042e-04, -4.93388e-04],
        [-4.20549e-05,  1.64312e-03, -1.17454e-03, ..., -8.63789e-05,
          1.18361e-03, -5.41687e-04],
        [-2.36747e-04,  4.12082e-03, -2.19435e-03, ..., -3.92024e-04,
         -1.83017e-03, -1.66362e-03],
        ...,
        [-1.81059e-03, -3.11806e-03,  4.82269e-04, ..., -1.68911e-04,
          4.22794e-04, -1.00459e-03],
        [-5.29763e-04, -1.00495e-03,  4.36633e-04, ..., -2.06172e-04,
          2.29263e-04, -4.10152e-04],
        [-1.89720e-04, -1.81242e-04,  3.18442e-04, ..., -1.69489e-04,
          4.02346e-05, -1.81635e-04]],

       ...,

       [[ 1.46079e-04, -8.25166e-04,  6.09599e-04, ..., -2.35004e-04,
          2.42891e-04, -7.51496e-05],
        [ 1.20524e-04, -3.11857e-04,  1.04892e-03, ..., -4.87344e-04,
          3.43499e-04, -6.75418e-04],
        [-2.78686e-04,  1.95781e-05,  1.39853e-03, ..., -8.46590e-04,
          9.31528e-04, -3.42571e-03],
        ...,
        [ 4.96518e-04, -3.48306e-03,  2.17695e-03, ...,  8.87406e-04,
         -5.75233e-04,  1.93982e-04],
        [ 4.16109e-04, -1.92033e-03,  9.48353e-04, ...,  3.12548e-04,
         -4.58189e-04,  1.26472e-04],
        [ 2.53736e-04, -6.88038e-04,  3.39592e-04, ...,  1.26221e-04,
         -1.58602e-04,  1.00377e-04]],

       [[ 6.19570e-04, -4.15901e-04,  2.07242e-04, ..., -3.77105e-04,
          1.09699e-04, -1.83479e-04],
        [ 1.13080e-03,  1.80459e-04,  2.04156e-04, ..., -7.53989e-04,
```

(continues on next page)

(continued from previous page)

```
          4.09488e-04, -8.32334e-04],
        [ 1.16754e-03,  5.21672e-04, -7.78524e-04, ..., -1.55456e-03,
         -1.13852e-04, -3.89561e-03],
        ...,
        [-1.46650e-05, -6.04571e-03,  3.87313e-03, ...,  1.03792e-03,
         -6.57619e-04, -2.99752e-04],
        [ 6.11913e-05, -1.26989e-03,  1.73945e-03, ...,  2.74474e-04,
         -6.51257e-04,  4.25862e-05],
        [ 2.19103e-04, -1.16641e-03,  7.31279e-04, ...,  9.97285e-05,
         -3.03561e-04,  5.61145e-05]],

       [[ 6.65368e-04, -4.43151e-06, -2.01946e-04, ..., -1.33367e-04,
          2.22934e-05, -2.33348e-04],
        [ 1.32565e-03,  1.36880e-04, -6.54239e-04, ..., -3.98235e-04,
          2.00215e-04, -1.03888e-03],
        [ 1.06385e-03,  1.41124e-03, -1.67270e-03, ..., -1.44708e-03,
         -4.56157e-04, -3.82760e-03],
        ...,
        [-1.14635e-03, -2.81073e-03,  3.70703e-03, ...,  5.03686e-04,
         -3.10570e-04, -8.05578e-05],
        [-2.74711e-04, -1.71081e-03,  1.61273e-03, ..., -7.04135e-05,
         -2.20637e-04,  1.66443e-04],
        [-7.68263e-05, -1.18094e-03,  6.73337e-04, ...,  4.97206e-05,
         -1.69600e-04,  1.08526e-04]]],


      [[[ 6.63459e-03, -3.37437e-03, -3.35515e-03, ..., -9.49669e-04,
         -2.68579e-03,  6.50514e-03],
        [ 9.12968e-03, -7.22581e-03, -4.24501e-03, ..., -1.74703e-03,
         -3.16699e-03,  7.67820e-03],
        [ 7.71626e-03, -5.52429e-03, -5.33153e-03, ..., -5.28505e-03,
         -4.05202e-03,  2.99451e-04],
        ...,
        [ 6.25965e-04, -1.46304e-02, -1.00215e-02, ...,  1.30336e-02,
          1.25806e-02,  2.51707e-03],
        [ 1.05628e-03, -1.25478e-02, -6.49182e-03, ...,  8.77716e-03,
          8.51097e-03,  9.86680e-04],
        [ 8.46178e-04, -5.89148e-03, -4.56450e-03, ...,  5.49236e-03,
          5.21531e-03,  6.13888e-04]],

       [[ 9.18522e-03, -3.34208e-04, -2.39649e-03, ...,  3.60521e-04,
         -5.05526e-03,  5.14981e-03],
        [ 1.26161e-02, -2.14244e-03, -2.56397e-03, ...,  3.53038e-04,
         -6.92290e-03,  6.06733e-03],
        [ 1.33491e-02,  2.36725e-03, -2.92752e-03, ..., -3.35320e-03,
         -2.51424e-03,  5.33454e-03],
        ...,
        [ 6.41317e-04, -1.11011e-02, -7.15145e-03, ...,  1.18647e-02,
          1.32795e-02,  1.04717e-03],
        [ 4.36028e-05, -8.66245e-03, -5.07126e-03, ...,  4.11300e-03,
          5.72037e-03,  6.54708e-04],
        [-4.09412e-04, -4.54788e-03, -3.41828e-03, ...,  2.83627e-03,
```

(continues on next page)

(continued from previous page)

```
          4.41921e-03,  1.58301e-04]],

       [[ 8.31052e-03,  4.22921e-03, -2.25483e-03, ...,  1.62446e-03,
         -8.43620e-03,  4.50279e-03],
        [ 1.12922e-02,  7.32371e-03, -2.67745e-03, ...,  9.30869e-04,
         -1.05722e-02,  6.20631e-03],
        [ 1.51287e-02,  7.26134e-03, -2.06695e-03, ..., -6.27430e-04,
         -3.00904e-03,  3.32246e-03],
        ...,
        [ 3.06024e-03, -1.37916e-02, -5.65459e-03, ...,  9.70211e-03,
          1.33434e-02,  3.95776e-03],
        [ 8.89846e-04, -9.20753e-03, -5.94054e-03, ...,  2.57700e-03,
          6.30693e-03,  3.13410e-03],
        [ 2.43406e-04, -3.55492e-03, -4.87791e-03, ...,  1.14833e-03,
          3.77411e-03,  1.79569e-03]],

       ...,

       [[-3.61441e-03,  5.61957e-04, -1.76350e-03, ..., -3.86098e-03,
         -7.64555e-04, -1.81228e-03],
        [-5.77974e-03,  3.37529e-03, -2.58588e-03, ..., -5.50819e-03,
         -5.59090e-03, -6.50543e-03],
        [-1.03968e-02,  2.45157e-03,  1.56374e-03, ..., -8.25061e-03,
         -1.14900e-02, -1.65243e-02],
        ...,
        [ 1.58778e-03, -6.07027e-03, -1.43842e-02, ..., -6.30615e-04,
          4.46550e-03,  2.52009e-03],
        [ 1.97764e-03, -4.89974e-03, -5.45858e-03, ...,  1.29130e-03,
          3.02918e-03,  1.56723e-03],
        [ 1.81484e-03, -1.97926e-03, -3.17147e-03, ...,  1.43749e-03,
          3.16560e-03,  1.32316e-03]],

       [[ 4.75758e-04, -9.07907e-04, -4.15461e-03, ..., -2.57788e-03,
         -8.36229e-04,  4.90096e-03],
        [ 1.58048e-04, -1.19511e-03, -5.86699e-03, ..., -3.95150e-03,
         -2.93053e-03,  2.74922e-03],
        [-4.47718e-03, -1.55299e-03, -6.32151e-03, ..., -5.88115e-03,
         -1.04109e-02, -1.16235e-02],
        ...,
        [ 7.47676e-05, -1.12330e-02, -6.18615e-03, ...,  8.57690e-04,
          7.60502e-03,  2.27585e-03],
        [ 2.03538e-03, -1.20773e-02, -5.51093e-03, ...,  5.80025e-03,
          4.67897e-03,  1.90197e-03],
        [ 2.04935e-03, -3.69004e-03, -3.95621e-03, ...,  3.49550e-03,
          4.45277e-03,  1.03283e-03]],

       [[ 3.08374e-03, -2.43060e-03, -4.58880e-03, ..., -1.95476e-03,
         -1.10620e-03,  7.17587e-03],
        [ 5.43264e-03, -3.69010e-03, -6.30989e-03, ..., -3.34521e-03,
         -1.40561e-03,  7.79084e-03],
        [-3.47941e-04, -7.73706e-03, -8.57942e-03, ..., -4.39765e-03,
         -6.43783e-03, -4.53129e-03],
```

(continues on next page)

(continued from previous page)

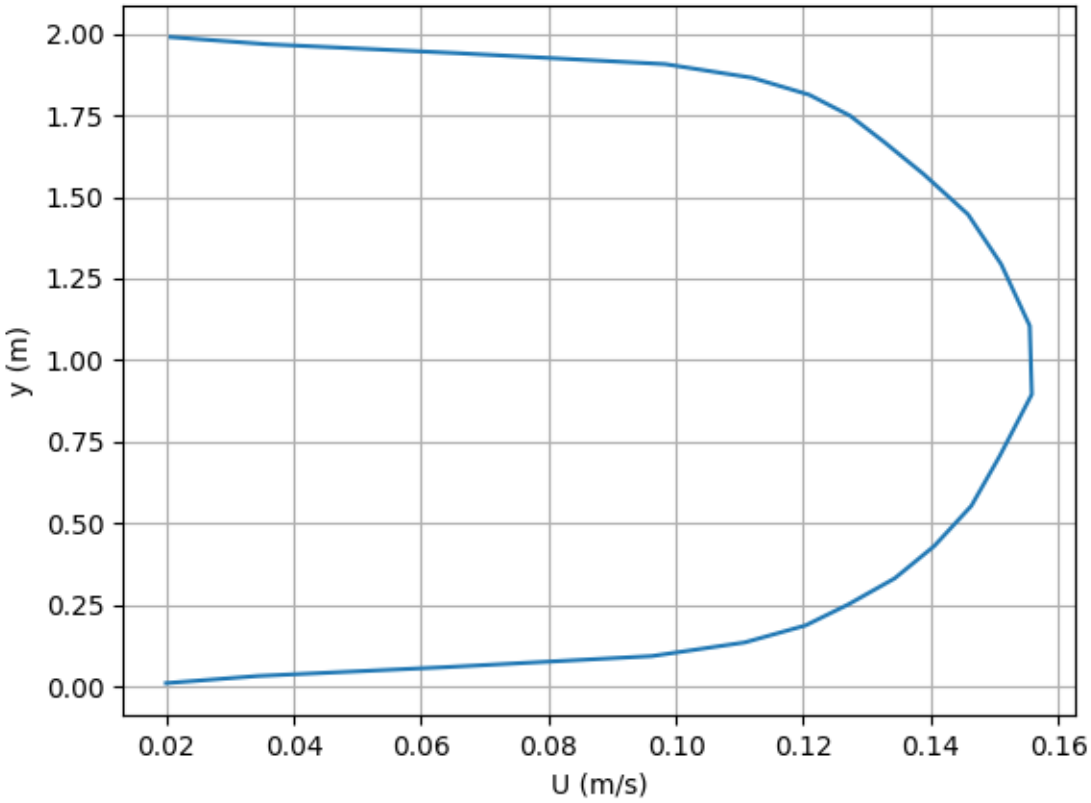
```
        ...,
        [ 5.35697e-04, -1.75389e-02, -5.19963e-03, ...,  9.27936e-03,
          1.02390e-02,  3.56798e-03],
        [ 2.43026e-03, -1.46963e-02, -7.67725e-03, ...,  9.30935e-03,
          8.80783e-03,  1.99104e-03],
        [ 2.36526e-03, -5.37717e-03, -5.44804e-03, ...,  5.87688e-03,
          5.58616e-03,  1.34276e-03]]])
```

Averaging along x and z axis (1 and 3)

```
import numpy as np

mySimu.vel_averaged = np.mean(np.mean(mySimu.U, 3), 1)
```

Now plot the profile of the averaged first velocity component

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(mySimu.vel_averaged[0], mySimu.y[0, :, 0])

#Setting axis labels
plt.xlabel('U (m/s)')
plt.ylabel('y (m)')

# add grid
plt.grid()

# show figure
plt.show()
```



Total running time of the script: (0 minutes 3.015 seconds)

Plot bed interface from unstructured mesh

This example shows how to create a pointer list of profiles from an unstructured mesh and to extract the bed interface elevation. The results are plotted as a scatter plot.

```
#path where the simulation is located
path = '../../output_samples'
#Name of the simulation to load
simu = '3dscour'
#time step to load
timeStep = '0'
#Switch to save postprocessed data on output
saveOutput = 1
```

Read the mesh and extract the pointerList

import python packages

```
from fluidfoam import readmesh,readfield
import os
import numpy as np
#
#---read mesh (in folder constant, if not done type writeCellCenters -time constant in a␣
↪terminal)
#
sol = os.path.join(path, simu)
Xb, Yb, Zb = readfield(sol, 'constant', 'C', precision=12)
#
#---create a list of (x,y) positions correpsonding to Zb = min(Zb)
#
print("Generate pointer list")
pbed = np.where(Zb == np.amin(Zb))[0]

n2d = np.size(pbed)
nz = 0
profileList = []
for i in range(n2d):
    #Extract the list of points having the same (Xb, Yb) values
    indices = np.where(np.logical_and(Xb == Xb[pbed[i]],
                                      Yb == Yb[pbed[i]]))[0]
    nz = max(nz, np.size(indices))
    profile = list(indices)
    #Sort the list of points by increasing Zb values (profile)
    profileSorted = [x for _, x in sorted(zip(Zb[profile], profile))]
    #Append the list to the profileList
    profileList.append(profileSorted)

# Convert profileList to numpy.array
pointer = np.array(profileList)
```

```
Reading file ../../output_samples/3dscour/constant/C
Generate pointer list
```

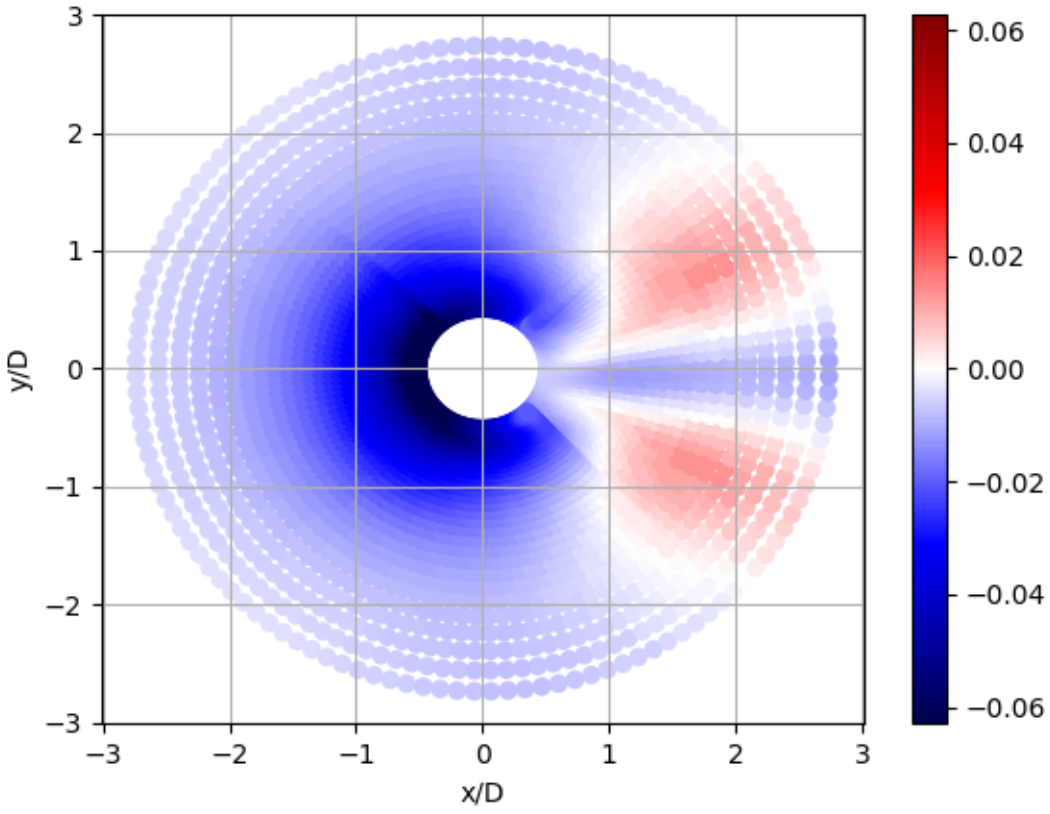
Extract bed surface from alpha.a profiles: min{ Zb | alpha.a <= 0.57 }

```
print("Extract bed surface")
a    = readfield(sol, timeStep, 'alpha.a')

zbed = np.zeros(n2d)
for i in range(n2d):
    bed_surface=np.where(a[pointer[i,:]] <= 0.57)[0]
    zbed[i] = np.min(Zb[pointer[i,bed_surface]])
```

```
Extract bed surface
Reading file ../../output_samples/3dscour/0/alpha.a
```

Now plot the zbed elevation

```
import matplotlib.pyplot as plt

D = 0.1
zmin = np.min([np.min(zbed), -np.max(zbed)])
zmax = np.max([-np.min(zbed), np.max(zbed)])
zlevels = np.linspace(zmin, zmax, 51)

plt.figure()
plt.scatter(Xb[pbed]/D, Yb[pbed]/D, c=zbed,
            vmin=zmin, vmax=zmax, cmap = 'seismic')
plt.colorbar()

#Setting axis labels
plt.xlabel('x/D')
plt.ylabel('y/D')

# add grid
plt.grid()

# show figure
plt.show()
```



save NetCDF files in constant and timeStep folders

```
if saveOutput == 1:
    from netCDF4 import Dataset

    postProcFile = os.path.join(sol, 'constant/pointerpostproc.nc')
    print ("save postprocFile in:",postProcFile)

    # NetCDF file creation
    rootgrp = Dataset(postProcFile, 'w')

    # Dimensions creation
    rootgrp.createDimension('XY', n2d)
    rootgrp.createDimension('Z', nz)

    # Variables creation
    pb_file = rootgrp.createVariable('pbed', np.int64, 'XY')
    x_file  = rootgrp.createVariable('x', np.float64, 'XY')
    y_file  = rootgrp.createVariable('y', np.float64, 'XY')
    z_file  = rootgrp.createVariable('z', np.float64, ('XY','Z'))
    p_file  = rootgrp.createVariable('p', np.int64, ('XY','Z'))

    # Writing variables
    pb_file[:]  = pbed
    x_file[:]   = Xb[pbed]
    y_file[:]   = Yb[pbed]
    z_file[:,:] = Zb[pointer[:,:]]
    p_file[:,:] = pointer[:,:]

    # File closing
    rootgrp.close()

    postProcFile2 = os.path.join(sol, timeStep, 'zbed.nc')
    print ("save zbed file in:",postProcFile2)

    # NetCDF file creation
    rootgrp = Dataset(postProcFile2, 'w')

    # Dimensions creation
    rootgrp.createDimension('XY', n2d)
    rootgrp.createDimension('Z', nz)

    # Variables creation
    zb_file = rootgrp.createVariable('zbed', np.float64, 'XY')

    # Writing variables
    zb_file[:]  = zbed

    # File closing
    rootgrp.close()
```

```
save postprocFile in: ../../output_samples/3dscour/constant/pointerpostproc.nc
save zbed file in: ../../output_samples/3dscour/0/zbed.nc
```

Total running time of the script: (0 minutes 9.848 seconds)

CHAPTER

ELEVEN

MORE

- FluidFoam forge on github
- FluidFoam in PyPI
- Unittest coverage

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX