fluidfoam Documentation

Release 0.2.0

Cyrille Bonamy

Feb 18, 2020

Contents:

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

CHAPTER 1

What is this repository for?

- Openfoam Tools
- Version : 0.2.0
- Supported OpenFoam Versions : 2.4.0, 4.1 to 7, v1712plus to v1912plus
- Supported Python Versions : 2.7.x, >= 3.4

CHAPTER 2

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 3

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 4

Example Usage

- http://servforge.legi.grenoble-inp.fr/pub/soft-sedfoam/

CHAPTER 5

Core Developers

- Cyrille.Bonamy@legi.cnrs.fr
- Julien.Chauchat@grenoble-inp.fr
- Antoine.Mathieu@univ-grenoble-alpes.fr

CHAPTER 6

Emeritus Core Developers

- Pierre.Augier@legi.cnrs.fr

CHAPTER 7

Emeritus Developers

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

CHAPTER 8

License

fluidfoam is distributed under the GNU General Public License v2 (GPLv2).

CHAPTER 9

Modules Reference

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

fluidfoam.readof	Read OpenFoam Files for Python
fluidfoam.processing1d	Write, Read and Plot 1D input files for swak4foam
fluidfoam.readpostpro	Read OpenFoam PostProcessing Files for Python
fluidfoam.meshdesign	Compute mesh grading and cell sizes

9.1 fluidfoam.readof

9.1.1 Read OpenFoam Files for Python

This module provides functions to read OpenFoam Files:

fluidfoam.readof.**readmesh**(*rep*, *structured=False*, *boundary=None*, *order='F'*, *precision=15*)

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

Args: rep: str

structured: False or True

boundary: None or str

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

precision : Number of decimal places to round to (default: 15). If decimals is negative, it specifies the number of positions to the left of the decimal point.

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)

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

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

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

precision : Number of decimal places to round to (default: 15). If decimals is negative, it specifies the number of positions to the left of the decimal point.

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*, *order='F'*, *precision=15*, *mode=None*)

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

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

precision : Number of decimal places to round to (default: 15). If decimals is negative, it specifies the number of positions to the left of the decimal point.

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*, *order='F'*, *precision=15*)

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

order: “F” (default) or “C”

precision : Number of decimal places to round to (default: 15). If decimals is negative, it specifies the number of positions to the left of the decimal point.

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*, *order='F'*, *precision=15*)

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

order: “F” (default) or “C”

precision : Number of decimal places to round to (default: 15). If decimals is negative, it specifies the number of positions to the left of the decimal point.

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*, *order='F'*, *precision=15*)

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

order: “F” (default) or “C”

precision : Number of decimal places to round to (default: 15). If decimals is negative, it specifies the number of positions to the left of the decimal point.

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.**typefield**(*path*, *time_name=None*, *name=None*)

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

readfield(path[, time_name, name, ...])	Read OpenFoam field and reshape if necessary (structured mesh) and possible (not uniform field).
readmesh(rep[, structured, boundary, order, ...])	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])	Read OpenFoam field and returns type of field.

Classes

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

9.2 fluidfoam.processing1d

9.2.1 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).
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.

9.3 fluidfoam.readpostpro

9.3.1 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

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_of_OpenFoam_case', '0', 'forces')

It will create and fill the force variables:

['T','Fpx','Fpy','Fpz','Fvx','Fvy','Fvz' ,'Fpox','Fpoy','Fpoz','Mpx','Mpy','Mpz' ,'Mvx','Mvy','Mvz','Mpox','Mpoy','Mpoz']

`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 .
varinforce()	return the var included in postProcessing force files.

9.4 fluidfoam.meshdesign

9.4.1 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.

Continued on next page

Table 6 – continued from previous page

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.

9.5 Gallery of Examples

Some examples

Note: Click *here* to download the full example code

9.5.1 Read and Plot OpenFoam output field

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')
```

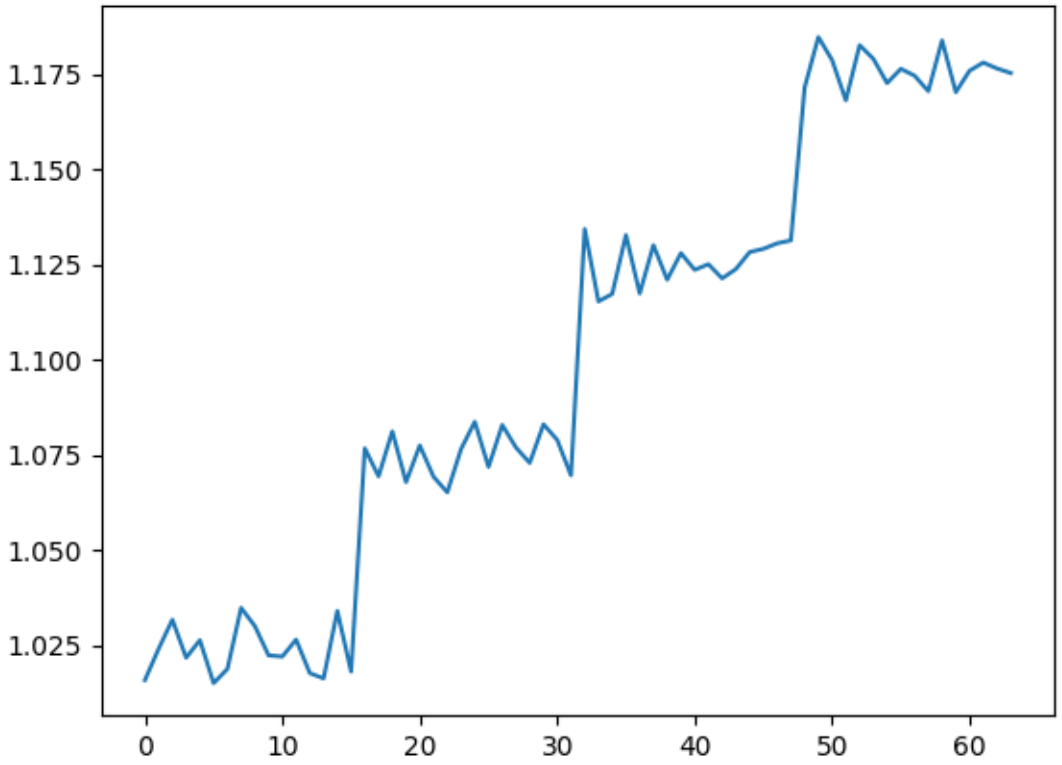
Out:

```
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)
```



Out:

```
[<matplotlib.lines.Line2D object at 0x7f66f4211c88>]
```

Total running time of the script: (0 minutes 0.208 seconds)

Note: Click *here* to download the full example code

9.5.2 Read and Plot a time series of OpenFoam 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

```

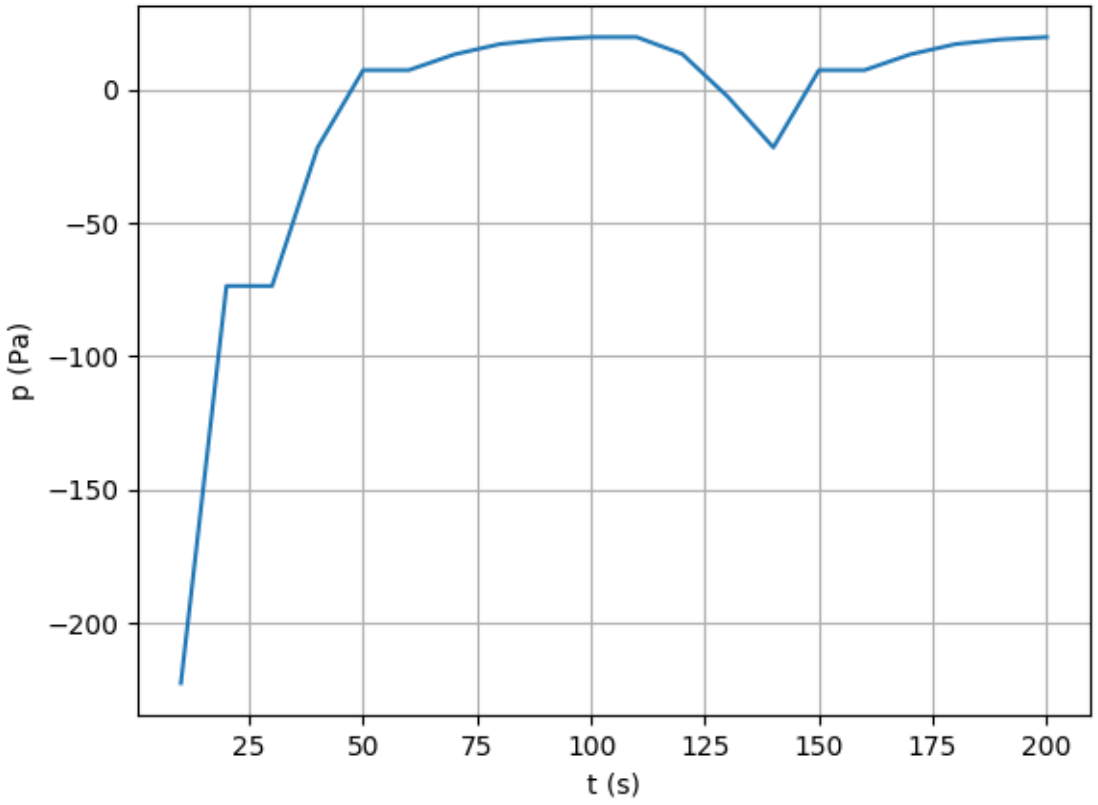
(continues on next page)

(continued from previous page)

```
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 0.139 seconds)

Note: Click *here* to download the full example code

9.5.3 Read and Plot a time series of OpenFoam 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)

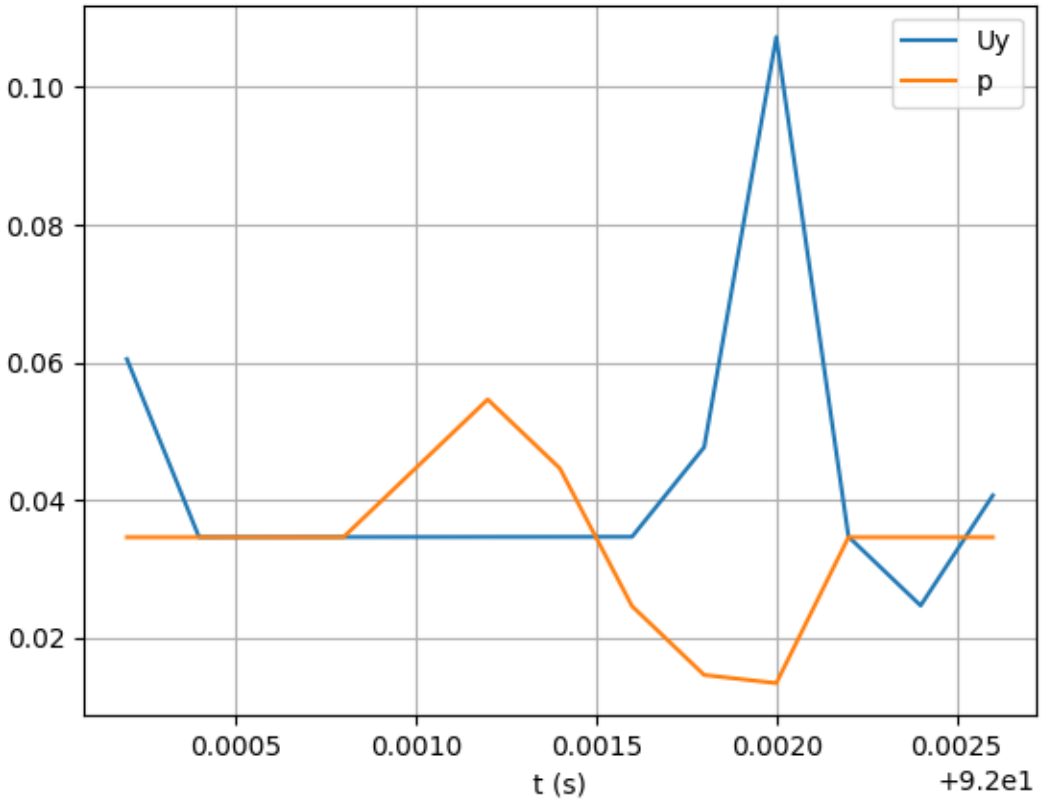
```
# import readprobes function from fluidfoam package
from fluidfoam.readpostpro import readprobes

sol = '../output_samples/ascii/'

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

Out:

```
Reading file ../output_samples/ascii/postProcessing/probes/0/U
4 probes over 10 timesteps
Reading file ../output_samples/ascii/postProcessing/probes/0.1/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/0.1/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 0.199 seconds)

Note: Click *here* to download the full example code

9.5.4 Read and Plot a spatially averaged profile from a structured mesh

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, True)
```

Out:

```
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

sol = '../output_samples/box/'
timename = '0'
vel = readvector(sol, timename, 'U', True)
```

Out:

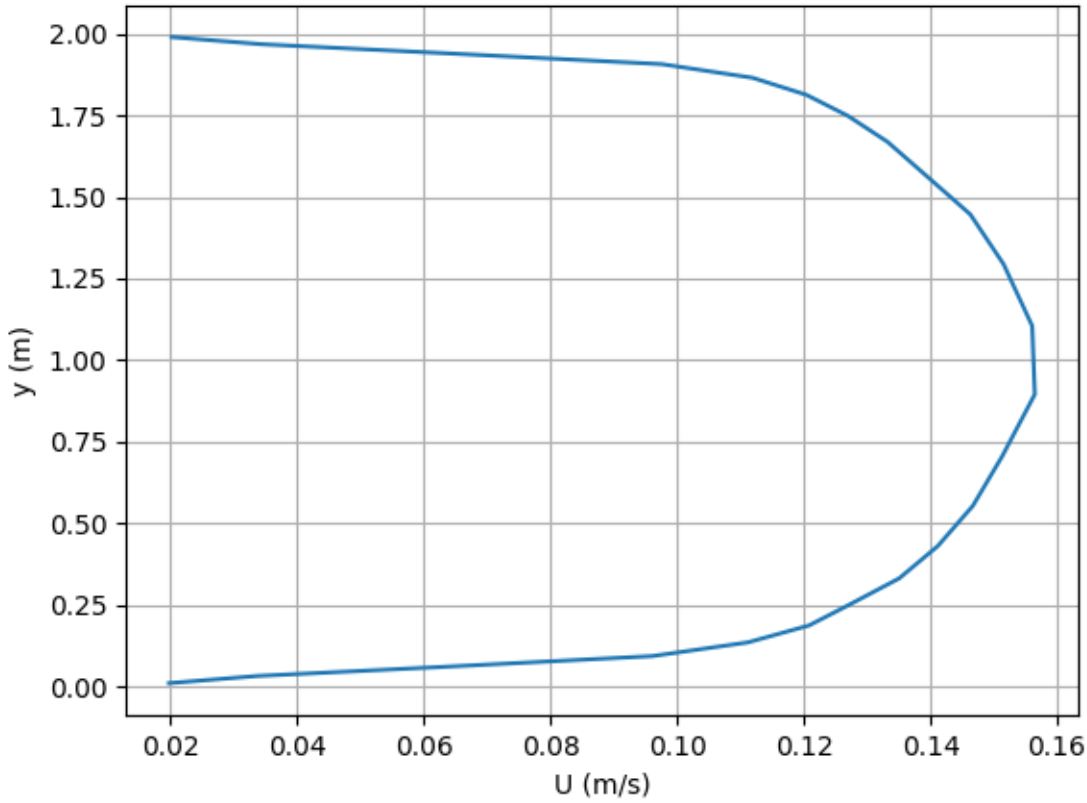
```
Reading file ../output_samples/box/0/U
Reading file ../output_samples/box//constant/polyMesh/owner
Reading file ../output_samples/box//constant/polyMesh/faces
Reading file ../output_samples/box//constant/polyMesh/points
Reading file ../output_samples/box//constant/polyMesh/neighbour
```

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 2.685 seconds)

Note: Click *here* to download the full example code

9.5.5 Read and Plot a 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 = []
```

(continues on next page)

(continued from previous page)

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

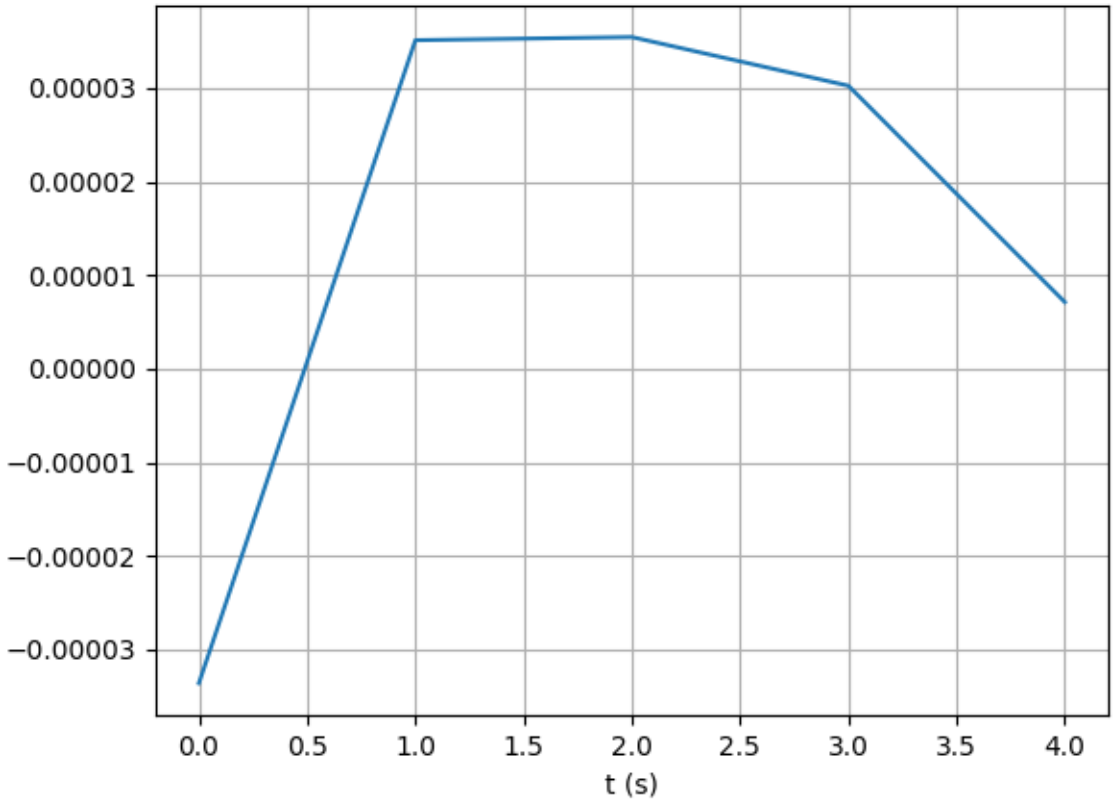
```
# import readvector function from fluidfoam package
from fluidfoam import readscalar

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])
```

Out:

```
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 0.209 seconds)

Note: Click *here* to download the full example code

9.5.6 Read and Plot a contour of OpenFoam output from a structured mesh

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, True)
```

(continues on next page)

(continued from previous page)

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

Out:

```
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

sol = '../output_samples/box/'
timename = '0'
vel = readvector(sol, timename, 'U', True)
```

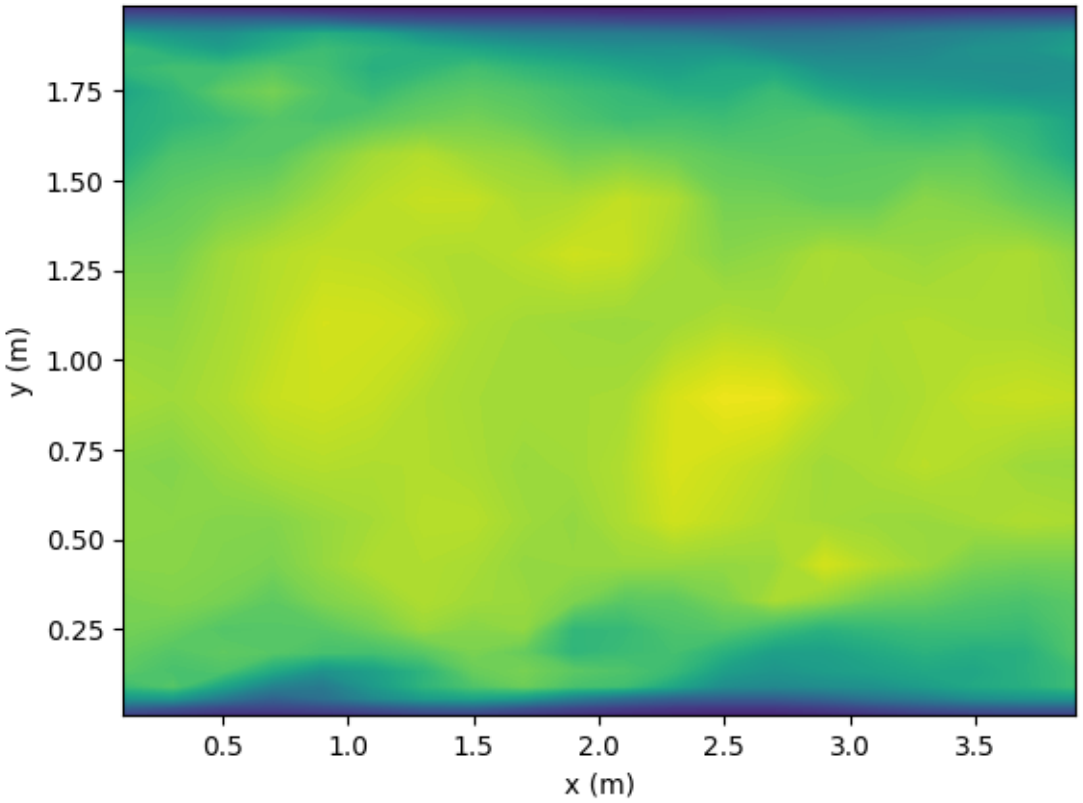
Out:

```
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)')
```

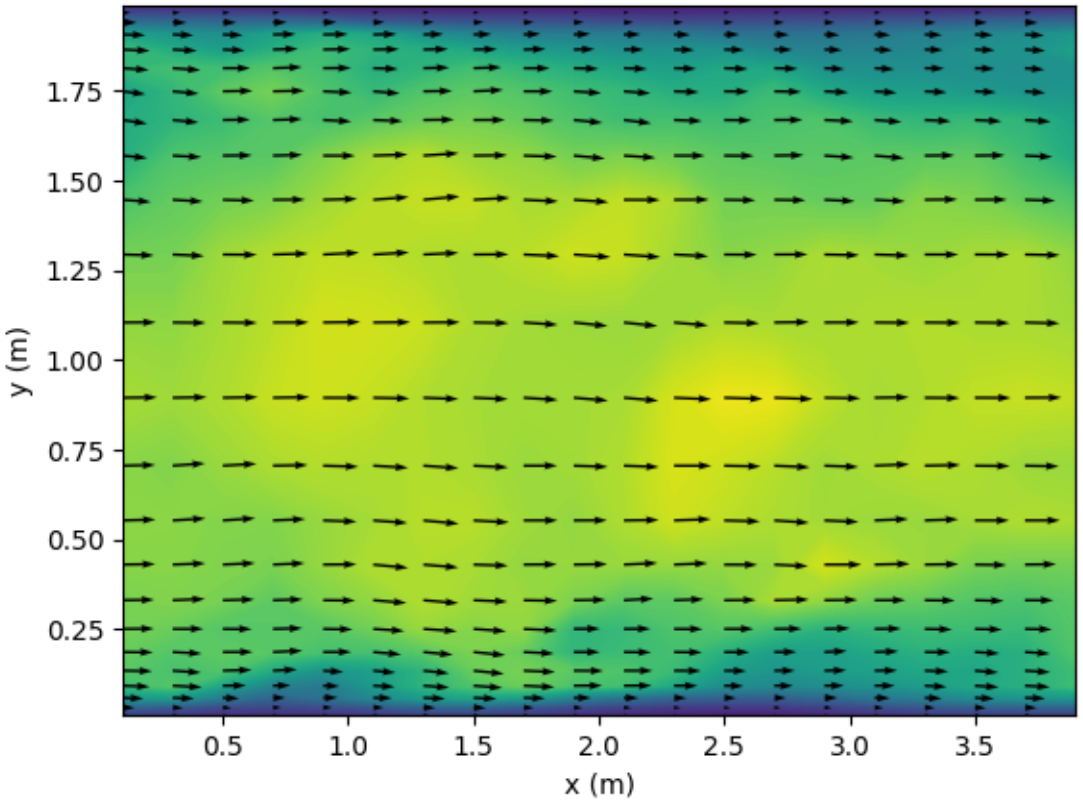


Out:

```
Text(0, 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])
```



Out:

```
<matplotlib.quiver.Quiver object at 0x7f66efac4898>
```

Total running time of the script: (0 minutes 3.230 seconds)

Note: Click *here* to download the full example code

9.5.7 Read and Plot a contour of OpenFoam output 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
```

(continues on next page)

(continued from previous page)

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

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

Out:

```
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 phi variables

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


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

Out:

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

Interpolate the fields on a structured grid

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

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

# Number of division for linear interpolation
ngridx = 500
ngridy = 180

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

# Interpolation grid
```

(continues on next page)

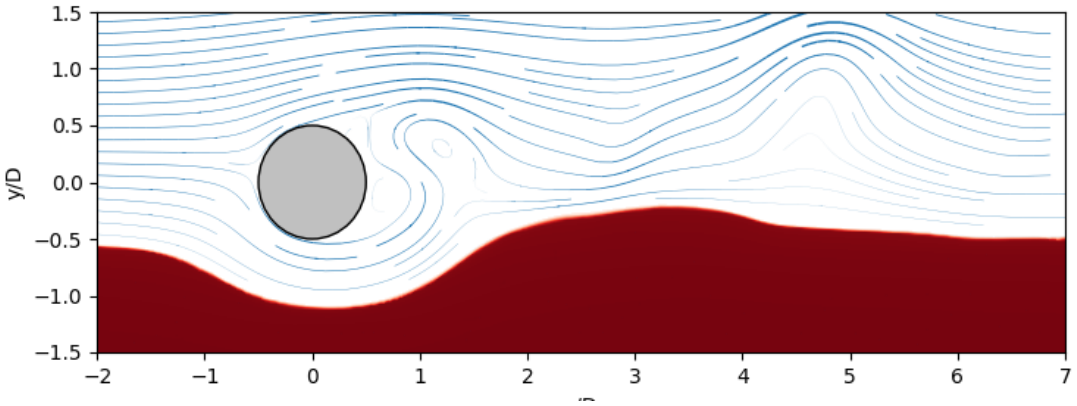
(continued from previous page)

```
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
phi_i = griddata((x, y), phi, (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 phi, streamlines and a patch

Note: The scalar field phi 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, phi_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
plt.streamplot(xi/d, yi/d, velx_i, vely_i, color='C0', density=[2, 1],
               linewidth=lw, arrowsize=0.05)
```



Out:

```
<matplotlib.streamplot.StreamplotSet object at 0x7f66e7944e48>
```

Total running time of the script: (0 minutes 30.936 seconds)

CHAPTER 10

More

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

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index