pflow.py a Potential Flow Solver and Visualizer

Mechanical Engineering Technical Report 2017/12
Ingo Jahn
School of Mechanical and Mining Engineering
The University of Queensland.

July 28, 2018

Abstract

`plow.py` is a simple teaching and analysis tool for 2-D Potential Flows. It is a collection of code, that allows the construction of simple flow fields that meet the Potential Flow governing Equations. A range of plotting and visulisation tools are included.

This report is a brief userguide and example book.

Contents

1 Introduction

Potential Flow is a simple but powerful analysis approach to simulate inviscid flow. This report is the userguide for `pflow.py` a tool to analyse simple 2-D flows together with a selection of plotting and post-processing tools. The code allows flow-fields, consisting of the following building blocks to be analysed: Uniform Flow, Sink/Source, Irrotational Vortex, Doublet, and other user-define options.
The post-processing tool allows plotting of stream functions ψ, velocity vectors, velocity contour plots, and pressure contours. In addition post-processing tools are included to extract point data and data along lines.

1.1 Getting the Code

The code is distributed as part of the *HTDT - Heat engine & Turbomachinery Design Tools* code under a GNU General Public License 3 license. The code can be downloaded from `https://bitbucket.org/uqturbine/htdt/src`. The source code and examples are located in the `htdt/src/PotentialFlow/` and `htdt/examples/PotentialFlow/` sub-directories respectively.

1.2 Dependencies and Installation

`pflow.py` is written in python. The following packages are required:

- `python 2.7` or `python 3.x`- any standard distribution
- `numpy`
- `matplotlib`

There are two options to run and install `pflow.py`.

1.2.1 Easy Installation - Windows Users

1. Copy the file `pflow.py` and `jobfile.py` and any other files you want to use into the same directory.
2. Open spyder3 (available to download from `https://anaconda.org/anaconda/python`)
3. Navigate to the directory containing the files.
4. Run `pflow.py` with the following command in the spyder console:
   `%run pflow.py --job=jobfile.py`
5. The results will appear in the spyder console

1.2.2 More Complex Install - Linux Users & OS X users

This installation path is slighty more complex, but allows easier use of the code.
For OS X:
From terminal, execute the following command: `$$ vi ~/.profile` and add the following two lines:

```
export PYTHONPATH=${PYTHONPATH}:${HOME}/path-to-file
export PATH=${PATH}:${HOME}/path-to-file
```

followed by restarting the terminal.

For Linux (e.g. UBUNTU):
From terminal, execute the following command: `$$ gedit ~/.profile` and add the following two lines:
```
export PYTHONPATH=${PYTHONPATH}:${HOME}/path-to-file
export PATH=${PATH}:${HOME}/path-to-file
```
followed by restarting the terminal.

The above steps ensure that `pflow.py` is on the search path for executables and python modules. This allows the code to be executed from any directory using the command:
```
$$  pflow.py --job=jobfile.py
```

1.3 Citing this tool

When using the tool in simulations that lead to published works, it is requested that the following works are cited:

- Jahn, I. (2017), pflow.py a Potential Flow Solver and Visualizer, *Mechanical Engineering Technical Report 2017/12*, The University of Queensland, Australia

2 Distribution

`pflow.py` is distributed as part of the code collection maintained by the *CFCFD Group* at the University of Queensland [1]. This collection is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version. This program collection is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details `http://www.gnu.org/licenses/`.
Alternatively the code is included in the Appendix.

2.1 Modifying the code

You can modify and adjust the code to your heart's content. If you want to share the updated and modified code, please email an updated version together with a short description of the changes to the authors. Once reviewed the changes will be included in future versions of the code.

3 Using the Tool: 5-minute version for experienced python Users

If you understand python and know how the potential flow building blocks work, this is for you.

1. Find an existing example file, e.g. `jobfile.py`.
2. Change the number and type of building blocks used to assemble your potential flow field, e.g. `a1 = UniformFlow(1.0,0.0)`. A full list of options is available in section 4.1.1.
3. Change the settings for how the flow will be visualised.
   E.g. `plot.psi()`
4. (Optional) Ask the code for an on-screen display of local flow conditions.
   E.g. `screen.locations([[0.,-0.5], [0.,0.5]])`
5. Save the job file.
6. Run the code:
   If using spyder:

   - Start spyder
   - Navigate to the directory containing `pflow.py` and your input file.
   - Run the code from the spyder console with: `%run pflow.py --job=jobfile.py`
     (Replace `jobfile.py` with the name of your input file)

   If running from terminal in Linux or OS X

   - Open a terminal
   - Navigate to the directory containing your input file (and `pflow.py` if you haven't added it to PATH and PYTHONPATH).
   - Run the code from the terminal with the command `pflow.py --job=jobfile.py`
     (Replace `jobfile.py` with the name of your input file)
7. If you want to save your data, add the option `--out-file=data.txt` to your file

4 Using the Tool: Detailed explanation

4.1 Creating your Flow field

In potential flow, different flow features, *building blocks*, that full-fill Laplace's equation by themselves, are superimposed (added) in order to generate complex flow-field solutions. The first step in `pflow.py` is to define these building blocks, which can be combined to create the complex flow-field.

Step: 1

Create a file that details the simulation you want to perform. In this document the file is called `jobfile.py`. Within this file start by defining the model reference data. The following variables are available. If skipped the

mdata.name : (default = "") name of simulation case.

mdata.dimensions : (default = 2) dimensions of domain. Currently limited to 2.

mdata.Pinf : (default = 0 Pa) free-stream pressure used for calculations

mdata.rho : (default = 1.225 kg/m^3) density used for flow calculations

mdate.Uinf : (default = np.nan') If 'nan', this parameter will be calculated based on vector sum of all uniform flow fields that are defined. Alternatively if an integer or float is defined, this value will be used.

Step: 2

Within the file define the building blocks you are using to construct your flow solution. The following code section defines Uniform Flow with $U = 5$ m/s, $V = 0$ m/s and a Source located at (0.5, 0.) with a net flow rate of 5 kg/s/m:

```
# Uniform Flows
A1 = UniformFlow(5.,0.,label='U-Flow')
# Sources
D1 = Source(0.5,0., 5.,label='Source 1')
```

See section 4.1.1 for a list of possible options and detailed descriptions.

4.1.1 Building Blocks

Currently the following Building Blocks are supported.

Uniform Flow: `UniformFlow(u,v,label=')'`

This creates a uniform flow with the velocity components u and v in the x- and y-direction respectively. The velocity vectors for the flow-field are shown in Fig. 1a.
The streamfunction is defined as:

$$\Psi = u\,y - v\,x \tag{1}$$

Source: `Source(x0,y0,m,label=')'`

This generates a source (use -ve m for sink) located at the position defined by $(x0, y0)$. Velocity vectors for the flow-field are shown Fig. 1b.

The streamfunction is defined as:

$$\theta = \tan^{-} 1 \left(\frac{y - y_0}{x - x_0} \right) \tag{2}$$

$$\Psi = \theta \frac{m}{2\,\pi} \tag{3}$$

Vortex: `Vortex(x0,y0,Gamma=Gamma,label=')'` or `Vortex(x0,y0,K=K,label=')'`

This generates an irrotational vortex of circulation Γ or strength K, depending on whether `Gamma=` or `K=` is specified, with the core locates at $(x0, y0)$ (If K is supplied, circulation is calculated as $\Gamma = 2\,\pi\,K$). Velocity vectors for the flow-field are shown Fig. 1c.
The streamfunction is defined as:

$$r = \left[(x - x_0)^2 + (y - y_0)^2 \right]^{\frac{1}{2}} \tag{4}$$

$$\Psi = -\frac{\Gamma}{2\,\pi} \ln r = -K\, \ln r \tag{5}$$

Doublet: `Doublet(x0,y0,a,U_inf,label=')'`

This generates the flow field known as a doublet. This is generated if a source and sink are brought very close together with a separation $s = \frac{a^2\,\pi U_\infty}{m}$ in the flow direction. $\pm m$ is the strength of the source and sink. The center of the doublet is located at (x_0, y_0). Velocity vectors for the flow-field are shown Fig. 1d.
The streamfunction is defined as:

$$\Psi = U_\infty\, (y - y_0)\, \frac{-a^2}{(x - x_0)^2 + (y - y_0)^2} \tag{6}$$

This doublet works only for flow in the $+x$ directions. For other flows modify the code or manually generate a doublet by bringing together a source and sink, aligned with the flow direction.

User_defined: `User_Defined(x0,y0,n,label=')'`

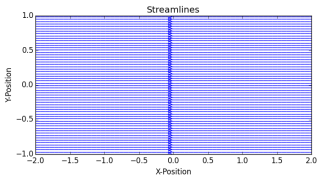
This generates the streamlines for flow around a 90° corner, located at position. Velocity vectors for the flow-field are shown Fig. 1e.
The streamfunction is defined as:

$$\Psi = n\, (x - x_0)\, (y - y_0) \tag{7}$$

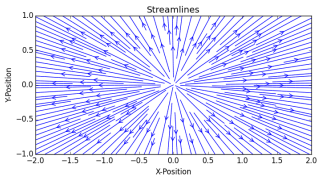
Name `Name(x0,y0,Var1,Var2,Var3,label=')'`

This is a template for future building blocks that you may want to implement. The block class must have the following functions and components:

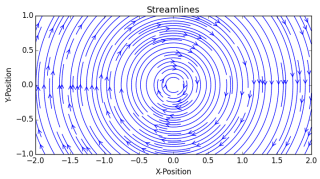
- `instances = []` This ensures all instances of the class are collected.
- `__init__(self,....)` This initializes the function and reads in the initial settings `self.__class__.instances.append(self)` This appends the current instance to the *instances* list.
- `evaP(self,x,y)` This returns the value of the stream function, Ψ at the point defined by the coordinates (x, y)
- `eval(self,x,y)` This returns the value of the u and v velocity at the point defined by the coordinates (x, y). This should be the analytical solution to $\frac{d\Psi}{dy}$ and $-\frac{d\Psi}{dx}$.



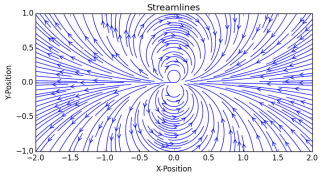
(a) Uniform Flow



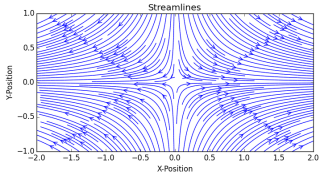
(b) Source



(c) Vortex



(d) Doublet



(e) User Defined

Figure 1: Building Blocks available to generate Potential Flow solutions.

4.2 Define the range and resolution for your simulation

After the building blocks have been defined, the next step is to define the range over which the data will be solved and how the data is visualised and plotted. This is done by adjusting the settings of the `visual` class.

Step: 3 (optional)

Use the following settings to adjust how much (x-range and y-range) is plotted and the resolution of the data:

visual.xmin : (default = -1.) sets x_{min} for plots
visual.xmax : (default = 1.) sets x_{max} for plots
visual.ymin : (default = -1.) sets y_{min} for plots
visual.ymax : (default = 1.) sets y_{max} for plots
visual.Nx : (default = 50) number of points used for discretisation in x-direction. Larger values will create better plots, but this will slow down the code.
visual.Ny : (default = 50) number of points used for discretisation in y-direction
visual.subplot : (default = 0) sets whether to plot individual graphs or to plot all graphs in a single figure.
0 - all individual graphs;
1 - subplots in single figure.

4.3 Plotting the results for PSI, U, V, ...

Once the area and visualisation settings have been defined, you now must define what is plotted.

Step: 4

Use the following commands to create different types of graphs.

plot.psi(levels=20) : plots 'real' streamlines, contours of psi. Use levels to set number of contours.

plot.psi_magU(min=[], max=[], levels=20) : create contour plot of velocity magnitude with overlaid stream functions. Use min and max to specify range and levels to set numbers of contours.

plot.vectors() : plots nice looking velocity vectors. Note these are not equi-potentials of streamfunction, Ψ.

plot.vectors_magU(min=[], max=[], levels=20) : create contour plot of velocity magnitude with overlaid velocity vectors. Use min and max to specify range and levels to set numbers of contours.

plot.magU(min=[], max=[], levels=20) : create contour plot of velocity magnitude. Use min and max to specify range and levels to set numbers of contours.

plot.U(min=[], max=[], levels=20) : create contour plot of U velocity. Use min and max to specify range and levels to set numbers of contours.

plot.V(min=[], max=[], levels=20) : create contour plot of V velocity. Use min and max to specify range and levels to set numbers of contours.

plot.P(min=[], max=[], levels=20) : create contour plot of pressure. $P = P_{inf} - \frac{1}{2}\,\rho\,|U|^2$. Use min and max to specify range and levels to set numbers of contours. P_{inf} and ρ are defined as part of the model settings (see section 4.1).

plot.Cp(min=[], max=[], level=20) : create contorus of pressure coefficient C_p. $C_p = 1. - P/(\frac{1}{2}\,\rho\,U_{inf}^2)$. Use min and max to specify range and levels to set numbers of contours. U_{inf} and ρ are defined as part of the model settings (see section 4.1).

For example, to create two graphs, one showing stream-functions, Ψ, and one showing velocity vectors super-imposed with velocity magnitude (velocity range limted to $[0., 10.]$), use the following code:

```
plot.vectors_magU(min=[0.], max=[10.], levels=20)
plot.psi(levels=20)
```

The results are shown in Fig. 2.

4.4 Extracting data

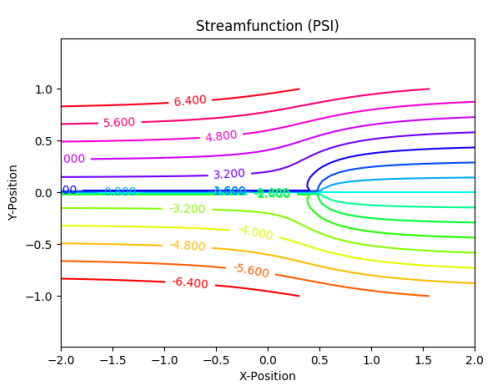
In addition to plotting the data it is also possible to evaluate the properties at single points or along lines.

Step: 5

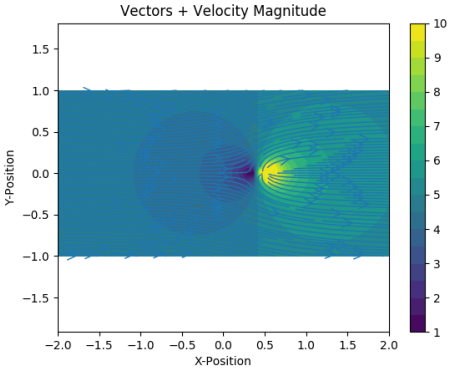
The display of data on screen is achieved using the following commands

screen.variables(['Psi', 'magU', 'P']) : This defines the parameters that will be evaluated. The following variables are available:

Psi : Value of streamfunction
magU : Velocity magnitude
U : x-component of velocity



(a) Streamlines



(b) Streamlines superimposed with contours of Velocity magnitude (magnitude capped at $10\,\mathrm{m\,s^{-1}}$).

Figure 2: Streamline and Streamline + Velocity magnitude plots generated for flow-field generated by *UniformFlow* and *Source*

V : y-component of velocity
P : pressure
Cp : pressure coefficient

screen.locations([[x0,y0], [x1,y1], ...]) : Extracts the local values at the points with coordinates (x0, y0) and (y0, y1).

screen.Lineval([x0,y0], [x1,y1], N=10) : Extracts the local values at $N = 10$ equally spaced points between (x0, y0) and (y0, y1).

The following code extracts stream function, Ψ, velocity magnitude, and pressure at the points $(-0.5, -0.5)$ and $(-0.5, 0.5)$, and also along the line between $(-1.0, 0.0)$ and $(1.0, 0.0)$.

```
# Extract data along lines
screen.locations([ [-0.5,-0.5], [-0.5,0.5] ])
screen.Lineval( [-1.,0.], [1.,0.], N=9)
```

A screen grab of the result is shown in Fig. 3. As expected, pressure and velocity are infinite at the center of the source.

4.5 Saving data

The setup of a simulation and the data displayed on screen can also be written to an output file. To achieve this run `pflow.py` with the `--out-file=datafile.txt`, where *datafile.txt* can be any file name. For example:

`%run pflow.py --job=jobfile.py --out-file=Test1.txt` (For use in spyder)
`pflow.py --job=jobfile.py --out-file=Test1.txt` (For use in from terminal)

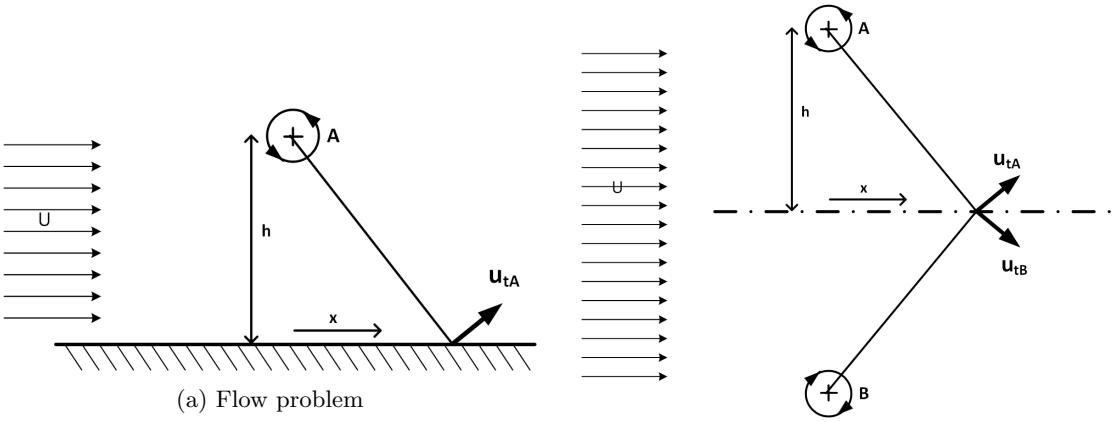
will save your data to *Test1.txt*, which can be opened with any text editor.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Output Data:
Density = 1.225 kg/s
Uinf = 5.0 m/s

Points:
     X-loc     Y-loc     Psi       magU      P
     -0.50     -0.50     -4.63      4.37    -11.72
     -0.50      0.50      4.63      4.37    -11.72

Lines:
     X-loc     Y-loc     Psi       magU      P
     -1.00      0.00      2.50      4.47    -12.24
     -0.75      0.00      2.50      4.36    -11.66
     -0.50      0.00      2.50      4.20    -10.83
     -0.25      0.00      2.50      3.94     -9.50
      0.00      0.00      2.50      3.41     -7.12
      0.25      0.00      2.50      1.82     -2.02
      0.50      0.00      0.00       inf      -inf
      0.75      0.00      0.00      8.18    -41.01
      1.00      0.00      0.00      6.59    -26.61
```

Figure 3: Stream function value, velocity magnitude and pressure evaluated at discrete points.

5 Example - Vortex near wall

This example shows how `Potential_Flow.py` can be used to analyse the flow field generated by a uniform flow parallel to a wall and a vortex positioned at a distance of 0.5 m from the wall. The problem is illustrated in Fig. 4a.



(a) Flow problem

(b) Using symmetry, in the form of a second vortex positioned behind the wall, a straight streamline representing the wall is created.

Figure 4: Example case, consisting of uniform flow and a vortex positioned near a wall.

In order to generate the effect of a wall (straight streamline) one can use the principle of symmetry. Thus the problem we actually solve using Potential Flow theory is the one shown in Fig. 4b, which consists of three building blocks. The Uniform Flow, the Vortex at $(0.0, 0.5)$ and a mirror image (about the x-axis) of the Vortex, located at $(0.0, -0.5)$. By symmetry this generates a straight streamline along the x-axis, which is equivalent to a wall (by definition no flow crosses a streamline, making it equivalent to a wall).

The appropriate code, defining the Uniform Flow, with a strength of 5.0 and vortices with a strength of ± 5.0 is given below. First the building blocks are generated as variables `A1`, `C1`, and `C2`. Then the extend of the grid over which we solve the flow-field is defined as ± 2 in the x-direction and ± 1 in the y-direction.

The results from the plotting functions, showing field data and data along the wall, are shown in Fig. 5. The obtained velocity in the wall parallel direction equals the analytical solution to the problem, given by

$$\begin{aligned} U_T(x) &= U_\infty + \frac{\Gamma\, h}{\pi\,(x^2 + h^2)} \\ &= 5.0 + \frac{5.0 \times 0.5}{\pi\left(x^2 + 0.5^2\right)} \\ U_T(0) &= 5.0 + 3.18 = 8.18 \end{aligned} \tag{8}$$

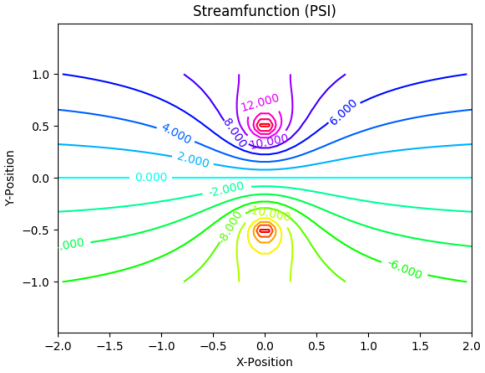
```
# set model parameters
mdata.name = 'Vortex + Uniform Flow adjacent to a wall'
mdata.dimensions = 2
```

```
# Define the Components
a1=Vortex(0.0, 0.5, Gamma=-5., label='Vortex')
a2=Vortex(0.0,-0.5, Gamma=5., label='Vortex')
b=UniformFlow(5.,0.)

# Define how the solution will be visualised.
visual.xmin =-2.
visual.xmax =2.
visual.subplot = 0

plot.psi(levels = 20)
plot.vectors_magU(min=0., max=20.)
plot.P(min=-100, max=0)

# Define what is printed to screen
screen.variables(['Psi', 'U', 'V', 'P'])
screen.Lineval([-2.0,0.0], [2.0,0.0], N=19)
```

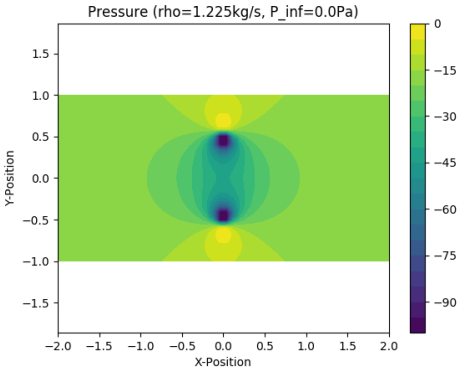
(see section 7.1 for a more detailed input template)

As expected along the wall ($y = 0.0$) the flow velocity in the vertical direction (V) is zero, as one would expect for a wall. Similarly the component of fluid velocity along the wall (U) is seen to accelerate, which results in a pressure reduction.

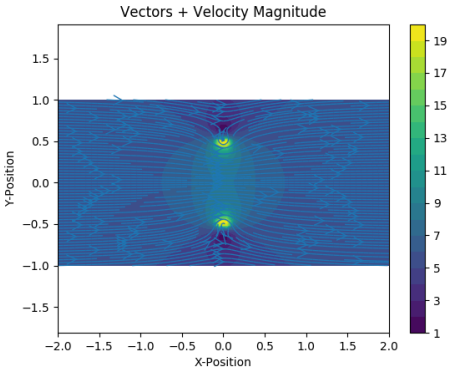
These results show the ability of potential flow to create simple flow fields, but it also highlights that potential flow is inviscid. If a *real* fluid, with viscosity, was used the x-component of velocity would be zero also. Nevertheless solutions created using potential flow are accurate for conditions and geometries, where the effects of viscosity are small.



(a) Streamlines



(b) Pressure contours



(c) U-Velocity, extracted along the x-axis

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Output Data:
Density = 1.225 kg/s
Uinf = 5.0 m/s

Lines:
```

X-loc	Y-loc	Psi	U	V	P
-2.00	0.00	0.00	5.19	0.00	-16.48
-1.78	0.00	0.00	5.23	0.00	-16.78
-1.56	0.00	0.00	5.30	0.00	-17.19
-1.33	0.00	0.00	5.39	0.00	-17.81
-1.11	0.00	0.00	5.54	0.00	-18.77
-0.89	0.00	0.00	5.77	0.00	-20.36
-0.67	0.00	0.00	6.15	0.00	-23.14
-0.44	0.00	0.00	6.78	0.00	-28.14
-0.22	0.00	0.00	7.66	0.00	-35.92
0.00	0.00	0.00	8.18	0.00	-41.01
0.22	0.00	0.00	7.66	0.00	-35.92
0.44	0.00	0.00	6.78	0.00	-28.14
0.67	0.00	0.00	6.15	0.00	-23.14
0.89	0.00	0.00	5.77	0.00	-20.36
1.11	0.00	0.00	5.54	0.00	-18.77
1.33	0.00	0.00	5.39	0.00	-17.81
1.56	0.00	0.00	5.30	0.00	-17.19
1.78	0.00	0.00	5.23	0.00	-16.78
2.00	0.00	0.00	5.19	0.00	-16.48

(d) Screen output

Figure 5: Flow field and flow properties obtained from a uniform flow with u-velocity of 5.0 m/s and a vortex with a strength of -5.0 positioned 0.5 m from a wall running along the x-axis.

6 References

References

[1] CFCFD, *The Compressible Flow Project* `http://cfcfd.mechmining.uq.edu.au` The University of Queensland

7 Appendix

7.1 Template for jobfile.py

```
"""
Template input file for HX_solver.py
"""

# set model parameters
"""
Set parameters that define model conditions (optional)
mdata.Pinf      - (defualt = 0 Pa)          sets P-infinity used in calculations
mdata.rho       - (default = 1.225 kg/s)    sets density used for calculations
mdata.Uinf      - (default = np.nan m/s)        sets U-infinity used in calculations.
                        If NaN this will be calculated automatically.
"""
mdata.name = 'Vortex + Uniform Flow adjacent to a wall'
mdata.dimensions = 2

# Define the Building Blocks
"""
The following is a short summary of the supported components. See the User-Guide
for more detailed instructions and detailed definition.

A = UniformFlow(Vx,Vy,label='UFlow1')
--> creates a uniform with velocity magnitude and direction defined by the
        x and y components Vx and Vy

B = Vortex(Cx,Cy,K=K,label='Vortex1')
--> creates a irrotational vortex located at (Cx, Cy) with a strength
        K = Gamma / (2 * pi). A positive Gamma results in a vortex rotating in
        the anticlockwise direction.

B = Vortex(Cx,Cy,Gamma=Gamma,label='Vortex1')
--> creates a irrotational vortex located at (Cx, Cy) with a strength Gamma. A
        positive K results in a vortex rotating in the anticlockwise direction.

C = Source(Cx,Cy,m,label='Source1')
--> creates a source/sink located at (Cx,Cy). m is the mass flow rate (per unit
        depth) coming out of the source. Use +ve m for source and -ve m for sinks

D = Doublet(Cx,Cy,R,Uinf,label='Doublet1')
--> creates a doublet (co-located source and sink) located at (Cx, Cy). R sets
        the radius of the enclosing streamline that is generated. When used in
        conjunction with a uniform flow to show the flow around a cylinder, set
        Ux and Uy to match the x and y components of the uniform flow.

E = User_Defined(Cx,Cy,n,label='user1')
--> Secret. Try it out and see if you can work out what it is.
```

```
46
47 """
48
49 a1=Source(0.5, 0.0,10.,label='Source')
50 b=UniformFlow(5.,0.)
51
52 # Define how the solution will be visualised.
53 """
54 By use the following settings to adjust the visaulisation.
55 ------ Define plotting Window ------
56 visual.xmin      - (default = -1.) sets x_min for plots
57 visual.xmax      - (default =  1.) sets x_max for plots
58 visual.ymin      - (default = -1.) sets y_min for plots
59 visual.ymax      - (default =  1.) sets y_max for plots
60 visual.Nx        - (default = 50) number of points used for discretisation
61                                   in x-direction
62 visual.Ny        - (default = 50) number of points used for discretisation
63                                   in x-direction
64 visual.subplot   - (default = 0)  0 - all individual graphs; 1 - subplots in
65                                   single figure
66
67 ------ Define what is plotted ------
68 plot.psi(levels=20) - plots 'real' streamlines, contours of psi. Use levels to
69                       set number of contours.
70 plot.psi_magU(min=[], max=[], levels=20) - create contour plot of velocity
71                       magnitude with overlaid stream functions. Use min and
72                       max to specify range and levels sets numbers of contours.
73 plot.vectors()  - plots nice looking streamlines. Note these are not
74                       euqipotentials of psi.
75 plot.vectors_magU(min=[], max=[], levels=20) - create contour plot of velocity
76                       magnitude with overlaid velocity vectors. Use min and
77                       max to specify range and levels sets numbers of contours.
78 plot.magU(min=[], max=[], levels=20) - create contour plot of velocity magnitude.
79                       Use min and max to specify range and levels sets numbers
80                       of contours.
81 plot.U(min=[], max=[], levels=20) - create contour plot of U velocity. Use min
82                       and max to specify range and levels sets numbers of
                             contours.
83 plot.V(min=[], max=[], levels=20) - create contour plot of V velocity. Use min
84                       and max to specify range and levels sets numbers of
                             contours.
85 plot.P(P_inf=0., rho=1.225, min=[], max=[], levels=20) - create contour plot of
86                       pressure, using P_inf and rho to perform the calculation.
87                       P = P_inf - 1/2 * rho * magU**2. Use min and max to
88                       specify range and levels sets numbers of contours.
89 plot.Cp(U_inf=1., rho=1.225, min=[], max=[]) - create contorus of pressure
90                       coefficient Cp, using U_inf and rho to perform the
91                       calculation.
92                       Cp = P / ( 1/2 * rho * U_inf**2 )
93 """
94
95 visual.xmin=-2.
96 visual.xmax =2.
97 visual.subplot = 0
98
99 plot.psi(levels = 20)
100 plot.vectors_magU(min=0., max=20.)
101 plot.P(min=-100, max=0)
102
103
104 # Define what is printed to screen
105 """
```

```
106 ----- Define what is displayed -----
107 screen.variables(['Psi','magU','U','V','P','Cp'])  - provide list of parameters
        that
108                             will be evaluated. (default ['Psi','P', 'magU'])
109 screen.locations([ [x0,y0], [x1,y1], [x2,y2], ...] ) - provide list of points
110                             where to evaluate data
111 screen.Lineval([x0,y0], [x1,y1], N=5)      - evaluates conditions at N equally
112                             spaced points between (x0,y0) and (x1,y1)
113 """
114
115 screen.variables(['Psi', 'U', 'V', 'P'])
116 screen.Lineval([-2.0,0.0], [2.0,0.0], N=19)
```

7.2 Source Code: pflow.py

The source code for `pflow.py` is available from the htdt repository. `https://bitbucket.org/uqturbine/htdt/src` and located in the `htdt/src/PotentialFlow/` directory