Final: Incompressible, Laminar Flow over a Rectangular Cavity

John Karasinski
Graduate Student Researcher
Center for Human/Robotics/Vehicle Integration and Performance
Department of Mechanical and Aerospace Engineering
University of California
Davis, California 95616
Email: karasinski@ucdavis.edu

1 Problem Description

2 Numerical Solution Approach

The pisoFoam solver from OpenFOAM 2.3.0 was used to model the solution of this problem. pisoFoam is a transient solver for incompressible flow which supports multiple forms of turbulence modelling. A Reynolds-average simulation (RAS) turbulence model is employed with a standard k-ε model.

The solver is initialized with the following initial conditions:

	internal	lid	fixedWalls
U [m/s]	(0 0 0)*	(1 0 0)*	(0 0 0)*
p [m^2/s^2]	0*	zeroGradient	zeroGradient
ε [m^2/s^3]	0.000765*	0.000765*	0.000765*
k [m^2/s^2]	0.00325*	0.00325*	0.00325*
ν_t [m^2/s]	0*	0*	0*

Table 1: Initial conditions for simulation (*: uniform)

The boundary field for the inlet and outlet boundaries is set to "zeroGradient" for the initial. Additionally, the boundary field for frontAndBack is set to "empty" for all initial fields, turning this into a 2D problem.

3 Results Discussion

4 Conclusion

References

[1] Mehta, U. B., and Lavan, Z., 1969. "Flow in a two-dimensional channel with a rectangular cavity". *Journal of Applied Mechanics,* **36**(4), pp. 897–901.

Appendix A: Python Code

```
import subprocess
import os
from PrettyPlots import *


def inplace_change(filename, old_string, new_string):
    with open(filename) as f:
        s = f.read()

    if old_string in s:
        # print 'Changing "{old_string}" to "{new_string}"'.format(**locals())
        s = s.replace(old_string, new_string)
        with open(filename, 'w') as f:
            f.write(s)
    # else:
        # print 'No occurances of "{old_string}" found.'.format(**locals())


def subprocess_cmd(command):
    process = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
    proc_stdout = process.communicate()[0].strip()
    # print proc_stdout


def generate_folders(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + '-' + str(Re)
        if not os.path.exists(run):
            command = "cp -rf base/ " + run + "/; "
            subprocess_cmd(command)

    print ('Folders generated.')


def create_mesh_file(path, AR, Re):
    mesh = 80.  # where 40 is a mesh size that gives results at the necessary spacing
    # print AR, mesh, AR*mesh, str(int(mesh*AR))

    YMESH = str(int(mesh * AR))
    XMESH = str(int(mesh))
    AR    = str(-AR)

    inplace_change(path, 'AR',       AR)
    inplace_change(path, 'XMESH', XMESH)
    inplace_change(path, 'YMESH', YMESH)
    inplace_change(path, 'MESH',  XMESH)


def create_properties_file(path, AR, Re):
    d = 12.           # depth of chasm
    NU = str(d / Re)  # where 20 is the width of the domain

    inplace_change(path, 'NU', NU)


def update_dimensions(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + '-' + str(Re)
        path = run + '/constant/polyMesh/blockMeshDict'
        create_mesh_file(path, AR, Re)
        # with open(path, 'w') as config_file:
            # config_file.write(create_mesh_file(AR, Re))

        path = run + '/constant/transportProperties'
        create_properties_file(path, AR, Re)
        # with open(path, 'w') as config_file:
            # config_file.write(create_properties_file(AR, Re))
```

```
68
69      print ('Config generated.')
70
71
72  def run_simulations(ARs, Res):
73      for AR, Re in zip(ARs, Res):
74          run = "Run" + str(AR) + '-' + str(Re)
75          if not os.path.exists(run + '/log'):
76              print(run + ' running now.')
77              command = "hdiutil attach -quiet -mountpoint $HOME/OpenFOAM OpenFOAM.sparsebundle; "
78              command += "sleep 1; "
79              command += "source $HOME/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc; "
80              command += "cd " + run + "; "
81              command += "blockMesh; "
82              command += "decomposePar; "
83              command += "mpirun -np 4 pisoFoam -parallel > log; "
84              command += "reconstructPar; "
85              command += "streamFunction; "
86              # command += 'paraFoam --script="../paraFoam.py" '
87
88              subprocess_cmd(command)
89          print(run + ' complete.')
90
91      print('Simulations complete.')
92
93
94  def main(ARs, Res):
95      print('Running ARs ' + str(ARs) + ' with Res ' + str(Res) + '.')
96      generate_folders(ARs, Res)
97      update_dimensions(ARs, Res)
98      run_simulations(ARs, Res)
99      # generate_plots(ARs, Res)
100     print('Done!')
101
102 if __name__ == "__main__":
103     # Base case
104     ARs = [  0.5]
105     Res = [100.0]
106     #          o                     Broken=x Working=o
107     main(ARs, Res)
108
109     # Additional cases
110     # ARs = [0.5,    0.5,   2.0,   5.0]
111     # Res = [1.0, 2000.0, 100.0, 100.0]
112     #        x       o      o      o  Broken=x Working=o
113
114     main(ARs, Res)
```

Listing 1: Code to create solutions

```
1  # import numpy as np
2  # import matplotlib.pyplot as plt
3  # import os
4
5
6  # # Configure figures for production
7  # WIDTH = 495.0  # width of one column
8  # FACTOR = 1.0   # the fraction of the width the figure should occupy
9  # fig_width_pt  = WIDTH * FACTOR
10
11 # inches_per_pt = 1.0 / 72.27
12 # golden_ratio  = (np.sqrt(5) - 1.0) / 2.0       # because it looks good
13 # fig_width_in  = fig_width_pt * inches_per_pt   # figure width in inches
14 # fig_height_in = fig_width_in * golden_ratio    # figure height in inches
15 # fig_dims      = [fig_width_in, fig_height_in]  # fig dims as a list
16
17
18 # def sigma_xx(x):
19 #     return 1E4*(1+(0.125/(x**2))+(0.09375/(x**4)))
```

```
# def sigma_yy(x):
#     return 1E4*((0.125/(x**2))-(0.09375/(x**4)))


# def plot_xx(widths, meshes):
#     # Format plot
#     plt.figure(figsize=fig_dims)
#     plt.xlabel('Distance, y (m)')
#     plt.ylabel('Stress ($\sigma_{xx}$)$_{x=0}$(kPa)')
#     title = 'Normal stress along the vertical symmetry'
#     x = np.linspace(0.5, 2)
#     sigmaxx = sigma_xx(x)

#     plt.plot(x, sigmaxx, '-k', label='Analytic Solution')
#     plt.xlim(0.5, 2)

#     for width, mesh in zip(widths, meshes):
#         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/leftPatch_sigmaxx_sigmayy.xy'
#         data = np.loadtxt(path)

#         if widths.count(widths[0]) == len(widths):
#             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
#         else:
#             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
#         plt.plot(data[:, 0], data[:, 1], '--', markersize=5, label=label)

#     if widths.count(widths[0]) == len(widths):
#         title += ' ($x=' + str(int(2*width)) + '$)'
#     else:
#         title += ' ($n=' + str(int(mesh)) + '$)'

#     plt.title(title)
#     plt.legend(loc='best')

#     # Save plots
#     save_name = 'result-x-' + str(widths) + str(meshes) + '.pdf'
#     try:
#         os.mkdir('figures')
#     except Exception:
#         pass

#     plt.savefig('figures/' + save_name, bbox_inches='tight')
#     plt.clf()


# def plot_xx_err(widths, meshes):
#     # Format plot
#     plt.figure(figsize=fig_dims)
#     plt.xlabel('Distance, y (m)')
#     plt.ylabel('Error in Stress ($\sigma_{xx}$)$_{x=0}$(kPa)')
#     plt.title('Error in Normal stress along the vertical symmetry')
#     plt.xlim(0.5, 2)

#     for width, mesh in zip(widths, meshes):
#         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/leftPatch_sigmaxx_sigmayy.xy'
#         data = np.loadtxt(path)

#         if widths.count(widths[0]) == len(widths):
#             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
#         else:
#             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'

#         x = data[:, 0]
#         sigmaxx = sigma_xx(x)
#         err = data[:, 1] - sigmaxx

#         RMS = np.sqrt(np.mean(np.square(err)))/(max(sigmaxx) - min(sigmaxx))
```

```
#         print('x err', width, mesh, '{0:.3e}'.format(RMS))

#         plt.plot(x, err, '--', markersize=5, label=label)

#     plt.legend(loc='best')

#     # Save plots
#     save_name = 'error-x-' + str(widths) + str(meshes) + '.pdf'
#     try:
#         os.mkdir('figures')
#     except Exception:
#         pass

#     plt.savefig('figures/' + save_name, bbox_inches='tight')
#     plt.clf()


# def plot_yy(widths, meshes):
#     # Format plot
#     plt.figure(figsize=fig_dims)
#     plt.xlabel('Distance, x (m)')
#     plt.ylabel('Stress ($\sigma_{yy}$)$_{y=0}$(kPa)')
#     title = 'Normal stress along the horizontal symmetry'
#     y = np.linspace(0.5, 2)
#     sigmayy = sigma_yy(y)

#     plt.plot(y, sigmayy, '-k', label='Analytic Solution')
#     plt.xlim(0.5, 2)

#     for width, mesh in zip(widths, meshes):
#         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/downPatch_sigmaxx_sigmayy.xy'
#         data = np.loadtxt(path)

#         if widths.count(widths[0]) == len(widths):
#             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
#         else:
#             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
#         plt.plot(data[:, 0], data[:, 2], '--', markersize=5, label=label)

#     if widths.count(widths[0]) == len(widths):
#         title += ' ($x=' + str(int(2*width)) + '$)'
#     else:
#         title += ' ($n=' + str(int(mesh)) + '$)'

#     plt.title(title)
#     plt.legend(loc='best')

#     # Save plots
#     save_name = 'result-y-' + str(widths) + str(meshes) + '.pdf'
#     try:
#         os.mkdir('figures')
#     except Exception:
#         pass

#     plt.savefig('figures/' + save_name, bbox_inches='tight')
#     plt.clf()


# def plot_yy_err(widths, meshes):
#     # Format plot
#     plt.figure(figsize=fig_dims)
#     plt.xlabel('Distance, x (m)')
#     plt.ylabel('Error in Stress ($\sigma_{yy}$)$_{y=0}$(kPa)')
#     plt.title('Error in Normal stress along the horizontal symmetry')
#     plt.xlim(0.5, 2)

#     for width, mesh in zip(widths, meshes):
#         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/downPatch_sigmaxx_sigmayy.xy'
#         data = np.loadtxt(path)
```

```
158
159 #         if widths.count(widths[0]) == len(widths):
160 #             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
161 #         else:
162 #             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
163
164 #         y = data[:, 0]
165 #         sigmayy = sigma_yy(y)
166 #         err = data[:, 2] - sigmayy
167
168 #         RMS = np.sqrt(np.mean(np.square(err)))/(max(sigmayy) - min(sigmayy))
169 #         print('y err', width, mesh, '{0:.3e}'.format(RMS))
170
171 #         plt.plot(y, err, '--', markersize=5, label=label)
172
173 #     plt.legend(loc='best')
174
175 #     # Save plots
176 #     save_name = 'error-y-' + str(widths) + str(meshes) + '.pdf'
177 #     try:
178 #         os.mkdir('figures')
179 #     except Exception:
180 #         pass
181
182 #     plt.savefig('figures/' + save_name, bbox_inches='tight')
183 #     plt.clf()
184
185
186 # def generate_plots(widths, meshes):
187 #     plot_xx(widths, meshes)
188 #     plot_xx_err(widths, meshes)
189 #     plot_yy(widths, meshes)
190 #     plot_yy_err(widths, meshes)
191
192 #     print('Plots generated.')
```

Listing 2: Code to generate pretty plots