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

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

```python
import subprocess
import os
from PrettyPlots import *


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(AR, Re):
    mesh = 200.

    config = '''
    /*--------------------------------*- C++ -*----------------------------------*\
    | =========                 |                                                 |
    | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
    |  \\    /   O peration     | Version:  2.3.0                                 |
    |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
    |    \\/     M anipulation  |                                                 |
    \*---------------------------------------------------------------------------*/
    FoamFile
    {
        version     2.0;
        format      ascii;
        class       dictionary;
        object      blockMeshDict;
    }
    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    convertToMeters 0.1;

    vertices
    (
        (0 0 0)
        (1 0 0)
        (1 ''' + str(AR) + ''' 0)
        (0 ''' + str(AR) + ''' 0)
        (0 0 0.1)
        (1 0 0.1)
        (1 ''' + str(AR) + ''' 0.1)
        (0 ''' + str(AR) + ''' 0.1)
    );

    blocks
    (
        hex (0 1 2 3 4 5 6 7) (''' + str(int(mesh)) + ' ' + str(int(mesh * AR)) + ''' 1) simpleGrading (1 1 1)
    );

    edges
    (
    );

    boundary
    (
        movingWall
        {
            type wall;
            faces
            (
                (3 7 6 2)
            );
        }
        fixedWalls
        {
            type wall;
            faces
            (
                (0 4 7 3)
                (2 6 5 1)
                (1 5 4 0)
            );
        }
        frontAndBack
        {
            type empty;
            faces
            (
                (0 3 2 1)
                (4 5 6 7)
            );
        }
    );

    mergePatchPairs
    (
    );

    // ************************************************************************* //
    ’’’

    return config


def create_properties_file(AR, Re):
    nu = 0.1 / Re

    config = ’’’
    /*--------------------------------*- C++ -*----------------------------------*\
    | =========                 |                                                 |
    | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
    |  \\    /   O peration     | Version:  2.3.0                                 |
    |   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
    |    \\/     M anipulation  |                                                 |
    \*---------------------------------------------------------------------------*/
    FoamFile
    {
        version     2.0;
        format      ascii;
        class       dictionary;
        location    "constant";
        object      transportProperties;
    }
    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    nu              nu [ 0 2 -1 0 0 0 0 ] ’’’ + str(nu) + ’’’;


    // ************************************************************************* //
    ’’’

    return config


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

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

    print (’Config generated.’)


def run_simulations(ARs, Res):
    for AR, Re in zip(ARs, Res):
        run = "Run" + str(AR) + ’-’ + str(Re)
        if not os.path.exists(run + ’/log’):
            print(run + ’ running now.’)
            command = "hdiutil attach -quiet -mountpoint $HOME/OpenFOAM OpenFOAM.sparsebundle; "
            command += "sleep 1; "
            command += "source $HOME/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc; "
            command += "cd " + run + "; "
            command += "blockMesh; "
            command += "icoFoam > log; "
            command += "streamFunction"
            subprocess_cmd(command)
        print(run + ’ complete.’)

    print(’Simulations complete.’)


def main(ARs, Res):
    print(’Running ARs ’ + str(ARs) + ’ with Res ’ + str(Res) + ’.’)
    generate_folders(ARs, Res)
    update_dimensions(ARs, Res)
    run_simulations(ARs, Res)
    # generate_plots(ARs, Res)
    print(’Done!’)

if __name__ == "__main__":
    # Base case
    ARs = [  0.5]
    Res = [100.0]
    #          o                      Broken=x Working=o
    main(ARs, Res)

    # Additional cases
    # ARs = [0.5,    0.5,   2.0,   5.0]
    # Res = [1.0, 2000.0, 100.0, 100.0]
    #        x       o      o      o  Broken=x Working=o

    main(ARs, Res)
```

Listing 1: Code to create solutions

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


# # Configure figures for production
# WIDTH = 495.0  # width of one column
# FACTOR = 1.0   # the fraction of the width the figure should occupy
# fig_width_pt  = WIDTH * FACTOR

# inches_per_pt = 1.0 / 72.27
# golden_ratio  = (np.sqrt(5) - 1.0) / 2.0       # because it looks good
# fig_width_in  = fig_width_pt * inches_per_pt   # figure width in inches
# fig_height_in = fig_width_in * golden_ratio    # figure height in inches
# fig_dims      = [fig_width_in, fig_height_in]  # fig dims as a list


# def sigma_xx(x):
#     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)

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

#         y = data[:, 0]
#         sigmayy = sigma_yy(y)
#         err = data[:, 2] - sigmayy

#         RMS = np.sqrt(np.mean(np.square(err)))/(max(sigmayy) - min(sigmayy))
#         print(’y err’, width, mesh, ’{0:.3e}’.format(RMS))

#         plt.plot(y, err, ’--’, markersize=5, label=label)

#     plt.legend(loc=’best’)

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

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


# def generate_plots(widths, meshes):
#     plot_xx(widths, meshes)
#     plot_xx_err(widths, meshes)
#     plot_yy(widths, meshes)
#     plot_yy_err(widths, meshes)

#     print(’Plots generated.’)
```

Listing 2: Code to generate pretty plots