

# 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

```
1 import subprocess
2 import os
3 from PrettyPlots import *
4
5
6 def subprocess_cmd(command):
7     process = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
8     proc_stdout = process.communicate()[0].strip()
9     # print proc_stdout
10
11
12 def generate_folders(ARs, Res):
13     for AR, Re in zip(ARs, Res):
14         run = "Run" + str(AR) + '-' + str(Re)
15         if not os.path.exists(run):
16             command = "cp -rf base/ " + run + "/; "
17             subprocess_cmd(command)
18
19     print ('Folders generated.')
20
21
22 def create_mesh_file(AR, Re):
23     mesh = 200.
24
25     config = '''
26     /*-----*- C++ -*-----*/
27     | ===== |
28     | \\      / F ield      | OpenFOAM: The Open Source CFD Toolbox |
29     | \\      / O peration   | Version: 2.3.0 |
30     | \\      / A nd         | Web:      www.OpenFOAM.org |
31     |  \\\\    M anipulation  | |
32     /*-----*/
33     FoamFile
34     {
35         version      2.0;
36         format       ascii;
37         class        dictionary;
38         object       blockMeshDict;
39     }
40     // * * * * *
41
42     convertToMeters 0.1;
43
44     vertices
45     (
46         (0 0 0)
47         (1 0 0)
48         (1 ''' + str(AR) + ''' 0)
49         (0 ''' + str(AR) + ''' 0)
50         (0 0 0.1)
51         (1 0 0.1)
52         (1 ''' + str(AR) + ''' 0.1)
53         (0 ''' + str(AR) + ''' 0.1)
54     );
55
56     blocks
57     (
58         hex (0 1 2 3 4 5 6 7) (''' + str(int(mesh)) + ' ' + str(int(mesh * AR)) + ''' 1) simpleGrading (1 1 1)
59     );
60
61     edges
62     (
63     );
64
65     boundary
66     (
67         movingWall
```

```

68     {
69         type wall;
70         faces
71         (
72             (3 7 6 2)
73         );
74     }
75     fixedWalls
76     {
77         type wall;
78         faces
79         (
80             (0 4 7 3)
81             (2 6 5 1)
82             (1 5 4 0)
83         );
84     }
85     frontAndBack
86     {
87         type empty;
88         faces
89         (
90             (0 3 2 1)
91             (4 5 6 7)
92         );
93     }
94 );
95
96 mergePatchPairs
97 (
98 );
99
100 // ***** //
101 '''
102
103 return config
104
105
106 def create_properties_file(AR, Re):
107     nu = 0.1 / Re
108
109     config = '''
110     /*-----*- C++ -*-----*/\
111     | ===== |
112     | \\      / F ield      | OpenFOAM: The Open Source CFD Toolbox |
113     | \\      / O peration   | Version: 2.3.0 |
114     | \\      / A nd         | Web:      www.OpenFOAM.org |
115     |  \\//    M anipulation  | |
116     \*-----*/\
117     FoamFile
118     {
119         version      2.0;
120         format        ascii;
121         class          dictionary;
122         location       "constant";
123         object          transportProperties;
124     }
125     // * * * * * //
126
127     nu                nu [ 0 2 -1 0 0 0 0 ] ''' + str(nu) + ''';
128
129
130 // ***** //
131 '''
132
133 return config
134
135
136 def update_dimensions(ARs, Res):

```

```

137 for AR, Re in zip(ARs, Res):
138     run = "Run" + str(AR) + '-' + str(Re)
139     path = run + '/constant/polyMesh/blockMeshDict'
140     with open(path, 'w') as config_file:
141         config_file.write(create_mesh_file(AR, Re))
142
143     path = run + '/constant/transportProperties'
144     with open(path, 'w') as config_file:
145         config_file.write(create_properties_file(AR, Re))
146
147     print ('Config generated.')
148
149
150 def run_simulations(ARs, Res):
151     for AR, Re in zip(ARs, Res):
152         run = "Run" + str(AR) + '-' + str(Re)
153         if not os.path.exists(run + '/log'):
154             print(run + ' running now.')
155             command = "hdiutil attach -quiet -mountpoint $HOME/OpenFOAM OpenFOAM.sparsebundle; "
156             command += "sleep 1; "
157             command += "source $HOME/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc; "
158             command += "cd " + run + "; "
159             command += "blockMesh; "
160             command += "icoFoam > log; "
161             command += "streamFunction"
162             subprocess_cmd(command)
163             print(run + ' complete.')
164
165     print('Simulations complete.')
166
167
168 def main(ARs, Res):
169     print('Running ARs ' + str(ARs) + ' with Res ' + str(Res) + '.')
170     generate_folders(ARs, Res)
171     update_dimensions(ARs, Res)
172     run_simulations(ARs, Res)
173     # generate_plots(ARs, Res)
174     print('Done!')
175
176 if __name__ == "__main__":
177     # Base case
178     ARs = [ 0.5]
179     Res = [100.0]
180     # o Broken=x Working=o
181     main(ARs, Res)
182
183     # Additional cases
184     # ARs = [0.5, 0.5, 2.0, 5.0]
185     # Res = [1.0, 2000.0, 100.0, 100.0]
186     # x o o o Broken=x Working=o
187
188     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)))
20
21
22 # def sigma_yy(x):
23 #     return 1E4*((0.125/(x**2))-(0.09375/(x**4)))
24
25
26 # def plot_xx(widths, meshes):
27 #     # Format plot
28 #     plt.figure(figsize=fig_dims)
29 #     plt.xlabel('Distance, y (m)')
30 #     plt.ylabel('Stress ($\sigma_{xx}$)_{x=0}$ (kPa)')
31 #     title = 'Normal stress along the vertical symmetry'
32 #     x = np.linspace(0.5, 2)
33 #     sigmaxx = sigma_xx(x)
34
35 #     plt.plot(x, sigmaxx, '-k', label='Analytic Solution')
36 #     plt.xlim(0.5, 2)
37
38 #     for width, mesh in zip(widths, meshes):
39 #         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/leftPatch_sigmaxx_sigmayy.xy'
40 #         data = np.loadtxt(path)
41
42 #         if widths.count(widths[0]) == len(widths):
43 #             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
44 #         else:
45 #             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
46 #         plt.plot(data[:, 0], data[:, 1], '--', markersize=5, label=label)
47
48 #     if widths.count(widths[0]) == len(widths):
49 #         title += ' ($x=' + str(int(2*width)) + '$)'
50 #     else:
51 #         title += ' ($n=' + str(int(mesh)) + '$)'
52
53 #     plt.title(title)
54 #     plt.legend(loc='best')
55
56 #     # Save plots
57 #     save_name = 'result-x-' + str(widths) + str(meshes) + '.pdf'
58 #     try:
59 #         os.mkdir('figures')
60 #     except Exception:
61 #         pass
62
63 #     plt.savefig('figures/' + save_name, bbox_inches='tight')
64 #     plt.clf()
65
66
67 # def plot_xx_err(widths, meshes):
68 #     # Format plot
69 #     plt.figure(figsize=fig_dims)
70 #     plt.xlabel('Distance, y (m)')
71 #     plt.ylabel('Error in Stress ($\sigma_{xx}$)_{x=0}$ (kPa)')
72 #     plt.title('Error in Normal stress along the vertical symmetry')
73 #     plt.xlim(0.5, 2)
74
75 #     for width, mesh in zip(widths, meshes):
76 #         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/leftPatch_sigmaxx_sigmayy.xy'
77 #         data = np.loadtxt(path)
78
79 #         if widths.count(widths[0]) == len(widths):
80 #             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
81 #         else:
82 #             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
83

```

```

84 #         x = data[:, 0]
85 #         sigma_xx = sigma_xx(x)
86 #         err = data[:, 1] - sigma_xx
87
88 #         RMS = np.sqrt(np.mean(np.square(err)))/(max(sigma_xx) - min(sigma_xx))
89 #         print('x err', width, mesh, '{0:.3e}'.format(RMS))
90
91 #         plt.plot(x, err, '--', markersize=5, label=label)
92
93 #     plt.legend(loc='best')
94
95 #     # Save plots
96 #     save_name = 'error-x-' + str(widths) + str(meshes) + '.pdf'
97 #     try:
98 #         os.mkdir('figures')
99 #     except Exception:
100 #         pass
101
102 #     plt.savefig('figures/' + save_name, bbox_inches='tight')
103 #     plt.clf()
104
105
106 # def plot_yy(widths, meshes):
107 #     # Format plot
108 #     plt.figure(figsize=fig_dims)
109 #     plt.xlabel('Distance, x (m)')
110 #     plt.ylabel('Stress ( $\sigma_{yy}$ ) ( $\sigma_{y=0}$ ) (kPa)')
111 #     title = 'Normal stress along the horizontal symmetry'
112 #     y = np.linspace(0.5, 2)
113 #     sigma_yy = sigma_yy(y)
114
115 #     plt.plot(y, sigma_yy, '-k', label='Analytic Solution')
116 #     plt.xlim(0.5, 2)
117
118 #     for width, mesh in zip(widths, meshes):
119 #         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/downPatch_sigmaxx_sigmayy.xy'
120 #         data = np.loadtxt(path)
121
122 #         if widths.count(widths[0]) == len(widths):
123 #             label = 'Explicit Solution ( $n=$  + str(int(mesh)) + '$)'
124 #         else:
125 #             label = 'Explicit Solution ( $x=$  + str(int(2*width)) + '$)'
126 #         plt.plot(data[:, 0], data[:, 2], '--', markersize=5, label=label)
127
128 #     if widths.count(widths[0]) == len(widths):
129 #         title += ' ( $x=$  + str(int(2*width)) + '$)'
130 #     else:
131 #         title += ' ( $n=$  + str(int(mesh)) + '$)'
132
133 #     plt.title(title)
134 #     plt.legend(loc='best')
135
136 #     # Save plots
137 #     save_name = 'result-y-' + str(widths) + str(meshes) + '.pdf'
138 #     try:
139 #         os.mkdir('figures')
140 #     except Exception:
141 #         pass
142
143 #     plt.savefig('figures/' + save_name, bbox_inches='tight')
144 #     plt.clf()
145
146
147 # def plot_yy_err(widths, meshes):
148 #     # Format plot
149 #     plt.figure(figsize=fig_dims)
150 #     plt.xlabel('Distance, x (m)')
151 #     plt.ylabel('Error in Stress ( $\sigma_{yy}$ ) ( $\sigma_{y=0}$ ) (kPa)')
152 #     plt.title('Error in Normal stress along the horizontal symmetry')

```

```

153 # plt.xlim(0.5, 2)
154
155 # for width, mesh in zip(widths, meshes):
156 #     path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/downPatch_sigmaxx_sigmayy.xy'
157 #     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