

# Case Study # 3: Structural Analysis: Perforated Plate in Tension

**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

This case study involves a linear-elastic, steady-state stress analysis on a square plate with a circular hole at its center. The plate dimensions are: side length  $x = 4\text{m}$  and radius  $R = 0.5\text{m}$ . It is loaded with a uniform traction of  $\sigma = 10\text{kPa}$  over its left and right faces as can be seen in Figure 1. A mesh sensitivity study is performed for a plate with side length  $x = 4\text{m}$ , and an effect of the plate length study is performed for plates of length  $x = 3\text{m}$ ,  $4\text{m}$ ,  $5\text{m}$ , and  $100\text{m}$ .

The stress normal to the vertical plane of symmetry is calculated for each case, and the results are compared to the analytical solution:

$$(\sigma_{xx})_{x=0} = \begin{cases} \sigma(1 + \frac{R^2}{2y^2} + \frac{3R^4}{2y^4}) & \text{for } |y| \geq R \\ 0 & \text{for } |y| < R \end{cases} \quad (1)$$

A Python script was created to automatically generate the configuration files, calculate the resulting steady-state stress through the plate using OpenFOAM, and plot the results for both the sensitivity and plate length studies. This script can be seen in the Appendix.

## 2 Numerical Solution Approach

Two symmetry planes can be identified for this geometry and therefore the solution domain need only cover a quarter of the geometry, shown by the shaded area in Figure 1. The quarter plate is then broken into five blocks of varying sizes, as can be seen in Figure 2. These blocks have a characteristic number of points,  $n$ . Blocks 0 and 1 consist of  $n$  by  $n$  points, block 2 consists of  $2n$  by  $n$  points, block 3 consists of  $2n$  by  $2n$  points, and block 4 consists of  $n$  by  $2n$  points. The mesh is generated with OpenFOAM's 'blockMesh' and Figure 3 shows the resulting mesh for  $n = 10$ .

The mesh sensitivity study looks at meshes of  $n = 10$ ,  $100$ , and  $1000$  and a plate width of  $x = 4\text{m}$ . The effect of plate length study looks at plate lengths of  $x = 3\text{m}$ ,  $4\text{m}$ ,  $5\text{m}$ , and

$100\text{m}$  with a mesh of  $n = 10$ . Once the meshes have been generated, OpenFOAM's 'solidDisplacementFoam' solver runs the simulation, and  $\sigma_{xx}$  is calculated and sampled by the OpenFOAM commands 'foamCalc components sigma' and 'sample'.

## 3 Results Discussion

Mesh sensitivity study

Selected key results for the base case

Effect of the plate length study

Summary of the difficulties that you have encountered in running the various cases and how you have addressed them.

## 4 Conclusion

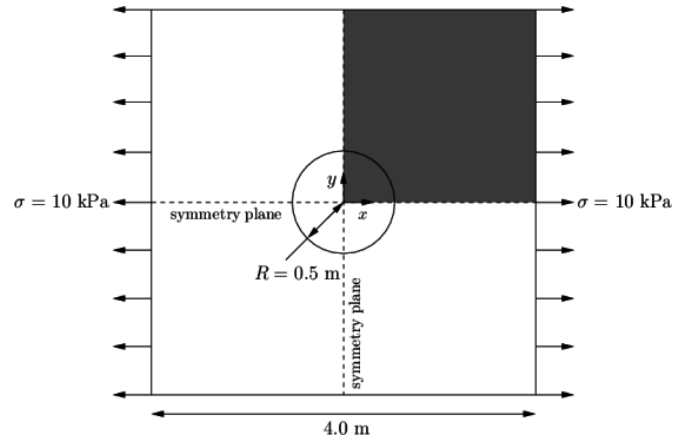


Fig. 1. Geometry of the plate with a hole

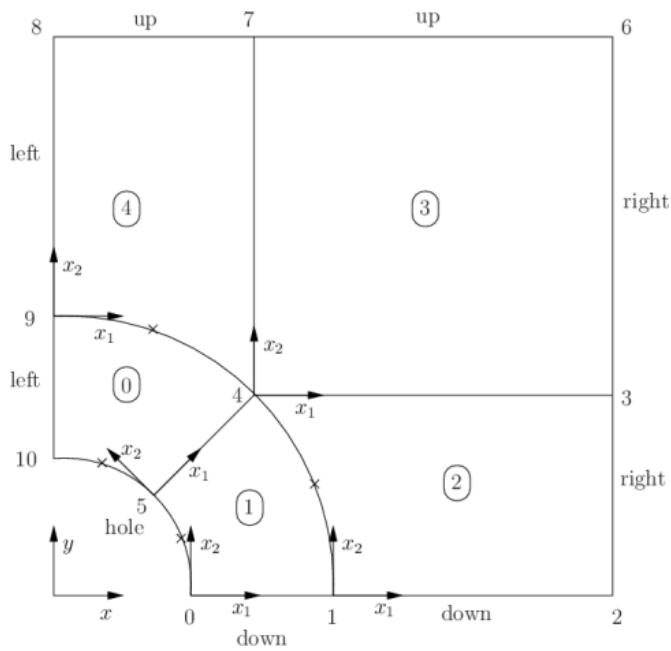


Fig. 2. Block structure of the mesh for the plate with a hole

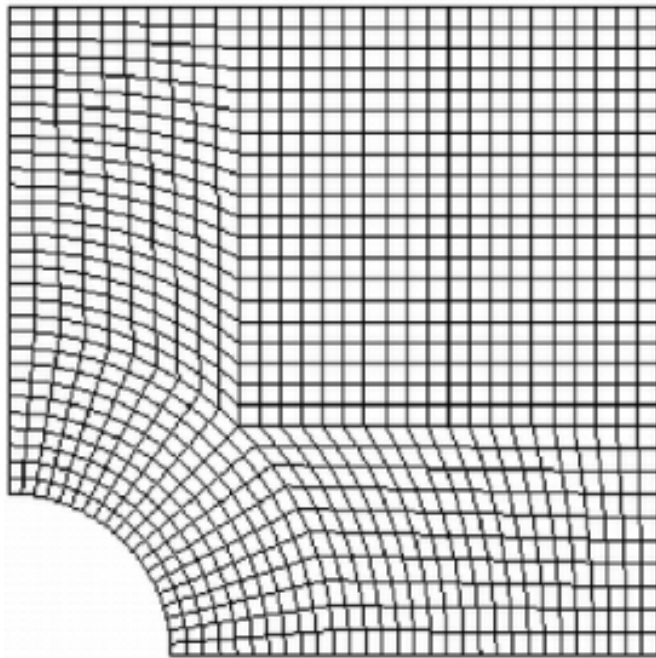


Fig. 3. Mesh of the hole in a plate problem with  $n = 10$

## Appendix A: Python Code

```
1 import subprocess
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import os
5 import time
6 import datetime
7
8
9 # Configure figures for production
10 WIDTH = 495.0 # the number latex spits out
11 FACTOR = 1.0 # the fraction of the width the figure should occupy
12 fig_width_pt = WIDTH * FACTOR
13
14 inches_per_pt = 1.0 / 72.27
15 golden_ratio = (np.sqrt(5) - 1.0) / 2.0 # because it looks good
16 fig_width_in = fig_width_pt * inches_per_pt # figure width in inches
17 fig_height_in = fig_width_in * golden_ratio # figure height in inches
18 fig_dims = [fig_width_in, fig_height_in] # fig dims as a list
19
20
21 def subprocess_cmd(command):
22     process = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
23     proc_stdout = process.communicate()[0].strip()
24     # print proc_stdout
25
26
27 def generate_folders(widths, meshes):
28     for width, mesh in zip(widths, meshes):
29         run = "Run" + str(width) + '-' + str(mesh)
30         if not os.path.exists(run):
31             command = "cp -rf base/ " + run + "/; "
32             subprocess_cmd(command)
33
34     print ('Folders generated.')
35
36
37 def create_config_file(width, mesh):
38     config = '''
39 /*-----*- C++ -*-----*/
40 | ===== |
41 | \\ / F ield | OpenFOAM: The Open Source CFD Toolbox |
42 | \\ / O peration | Version: 2.3.0 |
43 | \\ / A nd | Web: www.OpenFOAM.org |
44 | \\ / M anipulation |
45 /*-----*-*/
46 FoamFile
47 {
48     version 2.0;
49     format ascii;
50     class dictionary;
51     object blockMeshDict;
52 }
53 // * * * * *
54
55 convertToMeters 1;
56
57
58 vertices
59 (
60     (0.5 0 0)
61     (1 0 0)
62     ('' + str(width) + '' 0 0)
63     ('' + str(width) + '' 0.707107 0)
64     (0.707107 0.707107 0)
65     (0.353553 0.353553 0)
66     ('' + str(width) + '' 2 0)
67     (0.707107 2 0)
```

```

68     (0 2 0)
69     (0 1 0)
70     (0 0.5 0)
71     (0.5 0 0.5)
72     (1 0 0.5)
73     (''' + str(width) + ''' 0 0.5)
74     (''' + str(width) + ''' 0.707107 0.5)
75     (0.707107 0.707107 0.5)
76     (0.353553 0.353553 0.5)
77     (''' + str(width) + ''' 2 0.5)
78     (0.707107 2 0.5)
79     (0 2 0.5)
80     (0 1 0.5)
81     (0 0.5 0.5)
82 );
83
84 blocks
85 (
86     hex (5 4 9 10 16 15 20 21) (''' + str(mesh) + ' ' + str(mesh) + ''' 1) simpleGrading (1 1 1)
87     hex (0 1 4 5 11 12 15 16) (''' + str(mesh) + ' ' + str(mesh) + ''' 1) simpleGrading (1 1 1)
88     hex (1 2 3 4 12 13 14 15) (''' + str(mesh * 2) + ' ' + str(mesh) + ''' 1) simpleGrading (1 1 1)
89     hex (4 3 6 7 15 14 17 18) (''' + str(mesh * 2) + ' ' + str(mesh * 2) + ''' 1) simpleGrading (1 1 1)
90     hex (9 4 7 8 20 15 18 19) (''' + str(mesh) + ' ' + str(mesh * 2) + ''' 1) simpleGrading (1 1 1)
91 );
92
93 edges
94 (
95     arc 0 5 (0.469846 0.17101 0)
96     arc 5 10 (0.17101 0.469846 0)
97     arc 1 4 (0.939693 0.34202 0)
98     arc 4 9 (0.34202 0.939693 0)
99     arc 11 16 (0.469846 0.17101 0.5)
100    arc 16 21 (0.17101 0.469846 0.5)
101    arc 12 15 (0.939693 0.34202 0.5)
102    arc 15 20 (0.34202 0.939693 0.5)
103 );
104
105 boundary
106 (
107     left
108     {
109         type symmetryPlane;
110         faces
111         (
112             (8 9 20 19)
113             (9 10 21 20)
114         );
115     }
116     right
117     {
118         type patch;
119         faces
120         (
121             (2 3 14 13)
122             (3 6 17 14)
123         );
124     }
125     down
126     {
127         type symmetryPlane;
128         faces
129         (
130             (0 1 12 11)
131             (1 2 13 12)
132         );
133     }
134     up
135     {
136         type patch;

```

```

137         faces
138         (
139             (7 8 19 18)
140             (6 7 18 17)
141         );
142     }
143     hole
144     {
145         type patch;
146         faces
147         (
148             (10 5 16 21)
149             (5 0 11 16)
150         );
151     }
152     frontAndBack
153     {
154         type empty;
155         faces
156         (
157             (10 9 4 5)
158             (5 4 1 0)
159             (1 4 3 2)
160             (4 7 6 3)
161             (4 9 8 7)
162             (21 16 15 20)
163             (16 11 12 15)
164             (12 13 14 15)
165             (15 14 17 18)
166             (15 18 19 20)
167         );
168     }
169 );
170
171 mergePatchPairs
172 (
173 );
174
175 // *****
176
177 '''
178
179 return config
180
181
182 def update_dimensions(widths, meshes):
183     for width, mesh in zip(widths, meshes):
184         run = "Run" + str(width) + '-' + str(mesh)
185         path = run + '/constant/polyMesh/blockMeshDict'
186         with open(path, 'w') as config_file:
187             config_file.write(create_config_file(width, mesh))
188
189     print ('Config generated.')
190
191
192 def run_simulations(widths, meshes):
193     for width, mesh in zip(widths, meshes):
194         run = "Run" + str(width) + '-' + str(mesh)
195         if not os.path.exists(run + '/100/'):
196             print(run + ' running now.')
197             command = "hdiutil attach -quiet -mountpoint $HOME/OpenFOAM OpenFOAM.sparsebundle; "
198             command += "sleep 1; "
199             command += "source $HOME/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc; "
200             command += "cd " + run + "; "
201             command += "blockMesh; "
202             command += "solidDisplacementFoam > log; "
203             command += "foamCalc components sigma; "
204             command += "sample"
205             subprocess_cmd(command)

```

```

206         print(run + ' complete.')
207
208     print('Simulations complete.')
209
210
211 def sigma_xx(x):
212     return 1E4*(1+(0.125/(x**2))+(0.09375/(x**4)))
213
214
215 # this has not been found analytically
216 def sigma_yy(x):
217     return 1E4*(-(0.125/(x**2))-(0.09375/(x**4)))
218
219
220 def plot_xx(widths, meshes):
221     # Format plot
222     plt.figure(figsize=fig_dims)
223     plt.xlabel('Distance, y (m)')
224     plt.ylabel('Stress ($\sigma_{xx}$)_{x=0} (kPa)')
225     title = 'Normal stress along the vertical symmetry'
226     x = np.linspace(0.5, 2)
227     sigma_xx = sigma_xx(x)
228
229     plt.plot(x, sigma_xx, '-k', label='Analytic Solution')
230     plt.xlim(0.5, 2)
231
232     for width, mesh in zip(widths, meshes):
233         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/leftPatch_sigmaxx_sigmayy.xy'
234         data = np.loadtxt(path)
235
236         if widths.count(widths[0]) == len(widths):
237             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
238         else:
239             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
240         plt.plot(data[:, 0], data[:, 1], '--', markersize=5, label=label)
241
242     if widths.count(widths[0]) == len(widths):
243         title += ' ($x=' + str(int(2*width)) + '$)'
244     else:
245         title += ' ($n=' + str(int(mesh)) + '$)'
246
247     plt.title(title)
248     plt.legend(loc='best')
249
250     # Save plots
251     ts = time.time()
252     st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d-%H-%M-%S')
253     save_name = 'result-x-' + st + '.pdf'
254     try:
255         os.mkdir('figures')
256     except Exception:
257         pass
258
259     plt.savefig('figures/' + save_name, bbox_inches='tight')
260     plt.clf()
261
262
263 def plot_xx_err(widths, meshes):
264     # Format plot
265     plt.figure(figsize=fig_dims)
266     plt.xlabel('Distance, y (m)')
267     plt.ylabel('Error in Stress ($\sigma_{xx}$)_{x=0} (kPa)')
268     plt.title('Error in Normal stress along the vertical symmetry')
269     plt.xlim(0.5, 2)
270
271     for width, mesh in zip(widths, meshes):
272         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/leftPatch_sigmaxx_sigmayy.xy'
273         data = np.loadtxt(path)
274

```

```

275     if widths.count(widths[0]) == len(widths):
276         label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
277     else:
278         label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
279
280     x = data[:, 0]
281     sigma_xx = sigma_xx(x)
282     err = sigma_xx - data[:, 1]
283
284     RMS = np.sqrt(np.mean(np.square(err)))
285     print(width, RMS)
286
287     plt.plot(x, err, '--', markersize=5, label=label)
288
289 plt.legend(loc='best')
290
291 # Save plots
292 ts = time.time()
293 st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d-%H-%M-%S')
294 save_name = 'error-x-' + st + '.pdf'
295 try:
296     os.mkdir('figures')
297 except Exception:
298     pass
299
300 plt.savefig('figures/' + save_name, bbox_inches='tight')
301 plt.clf()
302
303
304 def plot_yy(widths, meshes):
305     # Format plot
306     plt.figure(figsize=fig_dims)
307     plt.xlabel('Distance, x (m)')
308     plt.ylabel('Stress ($\sigma_{yy}$) $_{y=0}$ (kPa)')
309     title = 'Normal stress along the horizontal symmetry'
310     y = np.linspace(0.5, 2)
311     sigmayy = sigma_yy(y)
312
313     plt.plot(y, sigmayy, '-k', label='Analytic Solution')
314     plt.xlim(0.5, 2)
315
316     for width, mesh in zip(widths, meshes):
317         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/downPatch_sigmaxx_sigmayy.xy'
318         data = np.loadtxt(path)
319
320         if widths.count(widths[0]) == len(widths):
321             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
322         else:
323             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
324         plt.plot(data[:, 0], data[:, 2], '--', markersize=5, label=label)
325
326     if widths.count(widths[0]) == len(widths):
327         title += ' ($x=' + str(int(2*width)) + '$)'
328     else:
329         title += ' ($n=' + str(int(mesh)) + '$)'
330
331     plt.title(title)
332     plt.legend(loc='best')
333
334     # Save plots
335     ts = time.time()
336     st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d-%H-%M-%S')
337     save_name = 'result-y-' + st + '.pdf'
338     try:
339         os.mkdir('figures')
340     except Exception:
341         pass
342
343     plt.savefig('figures/' + save_name, bbox_inches='tight')

```

```

344 plt.clf()
345
346
347 def plot_yy_err(widths, meshes):
348     # Format plot
349     plt.figure(figsize=fig_dims)
350     plt.xlabel('Distance, x (m)')
351     plt.ylabel('Error in Stress ( $\sigma_{yy}$ ) $_{y=0}$  (kPa)')
352     plt.title('Error in Normal stress along the horizontal symmetry')
353     plt.xlim(0.5, 2)
354
355     for width, mesh in zip(widths, meshes):
356         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/downPatch_sigmaxx_sigmayy.xy'
357         data = np.loadtxt(path)
358
359         if widths.count(widths[0]) == len(widths):
360             label = 'Explicit Solution ($n=' + str(int(mesh)) + '$)'
361         else:
362             label = 'Explicit Solution ($x=' + str(int(2*width)) + '$)'
363
364         y = data[:, 0]
365         sigmayy = sigma_yy(y)
366         err = sigmayy - data[:, 2]
367
368         RMS = np.sqrt(np.mean(np.square(err)))
369         print(width, RMS)
370
371         plt.plot(y, err, '--', markersize=5, label=label)
372
373     plt.legend(loc='best')
374
375     # Save plots
376     ts = time.time()
377     st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d-%H-%M-%S')
378     save_name = 'error-y-' + st + '.pdf'
379     try:
380         os.mkdir('figures')
381     except Exception:
382         pass
383
384     plt.savefig('figures/' + save_name, bbox_inches='tight')
385     plt.clf()
386
387
388 def generate_plots(widths, meshes):
389     plot_xx(widths, meshes)
390     plot_xx_err(widths, meshes)
391     plot_yy(widths, meshes)
392     plot_yy_err(widths, meshes)
393
394     print('Plots generated.')
395
396
397 def main(widths, meshes):
398     print('Running widths ' + str(widths) + ' with meshes ' + str(meshes) + '.')
399     generate_folders(widths, meshes)
400     update_dimensions(widths, meshes)
401     run_simulations(widths, meshes)
402     generate_plots(widths, meshes)
403     print('Done!')
404
405 if __name__ == "__main__":
406     # Increasing mesh resolution
407     widths = [2, 2, 2]
408     meshes = [10, 100, 1000]
409     main(widths, meshes)
410
411     # Changing the plate size
412     widths = [1.5, 2, 2.5, 50]

```



```
413 meshes = [10 for _ in widths]
414 main(widths, meshes)
```