

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 $x = 3\text{m}$, 4m , 5m , and 100m . 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 is included 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' command, and the resulting mesh for $n = 10$ can be seen in Figure 3.

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

$x = 3\text{m}$, 4m , 5m , and 100m with a mesh of $n = 10$. Once the meshes have been generated, OpenFOAM's 'solidDisplacementFoam' solver runs the simulation, and σ_{xx} is calculated and sampled by the OpenFOAM commands 'foamCalc components sigma' and 'sample'. The 'solidDisplacementFoam' solver is a transient segregated finite-volume solver for linear-elastic, small-strain deformation of a solid body, and is well suited to solve this problem.

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

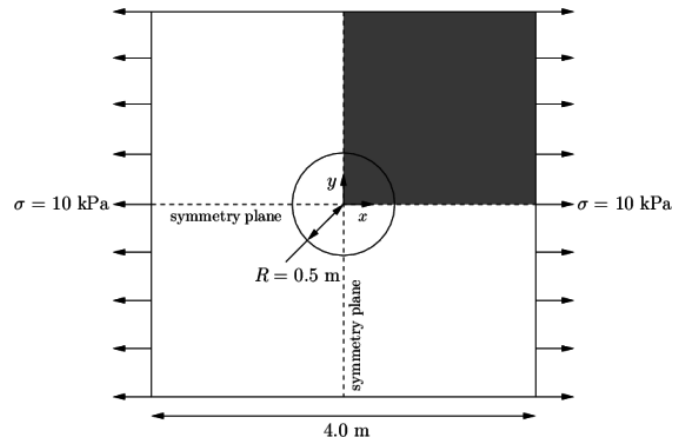


Fig. 1. Geometry of the plate with a hole

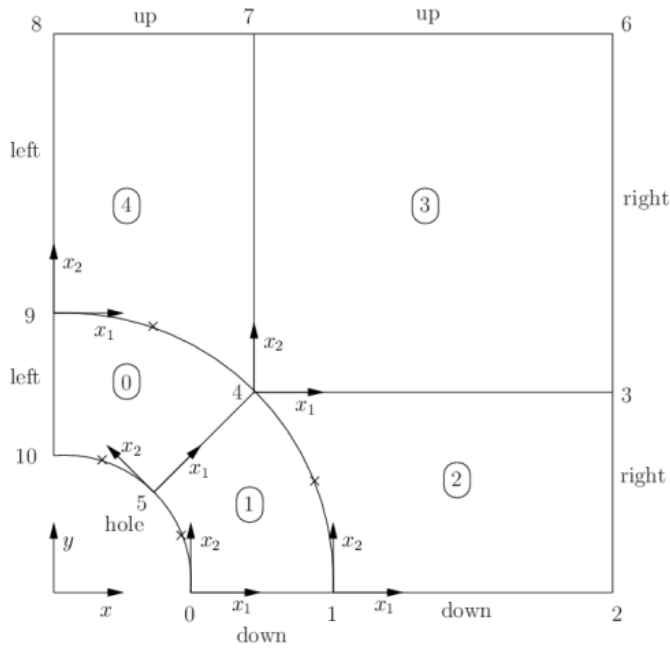


Fig. 2. Block structure of the mesh for the plate with a hole

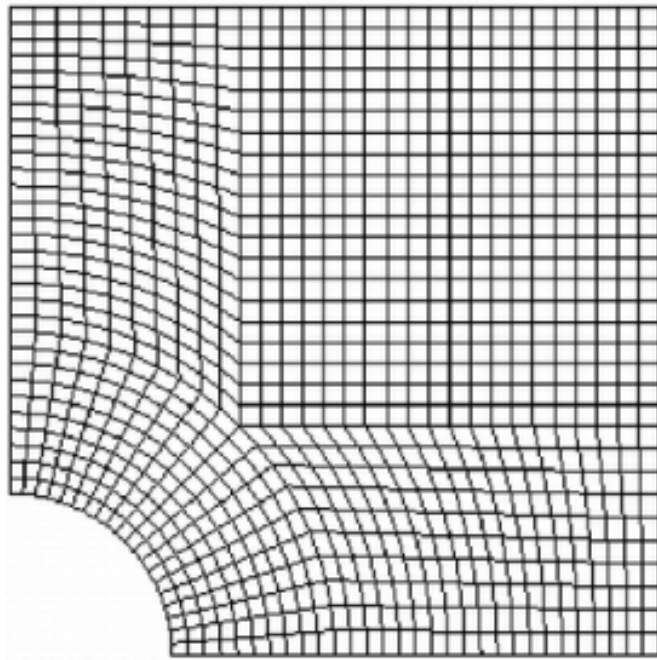


Fig. 3. Mesh of the hole in a plate problem with $n = 10$

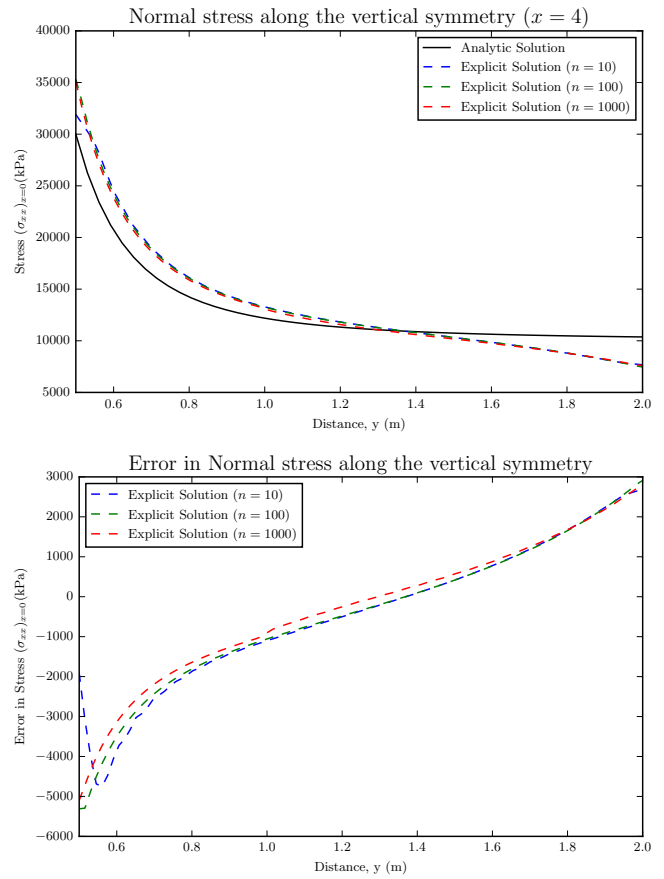


Fig. 4. Results of mesh sensitivity study

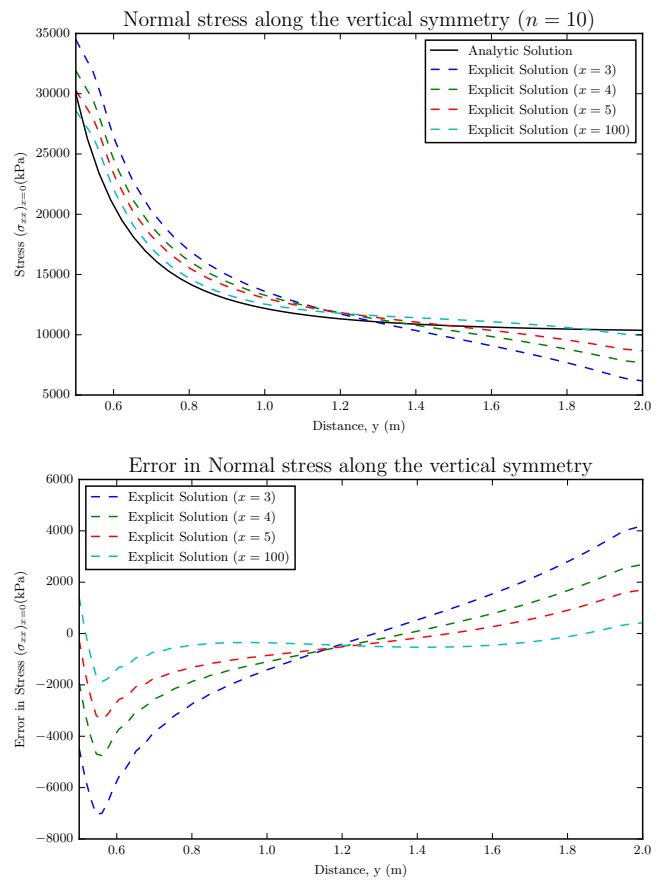


Fig. 5. Results of plate length study

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     save_name = 'result-x-' + str(widths) + str(meshes) + '.pdf'
252     try:
253         os.mkdir('figures')
254     except Exception:
255         pass
256
257     plt.savefig('figures/' + save_name, bbox_inches='tight')
258     plt.clf()
259
260
261 def plot_xx_err(widths, meshes):
262     # Format plot
263     plt.figure(figsize=fig_dims)
264     plt.xlabel('Distance, y (m)')
265     plt.ylabel('Error in Stress ( $\sigma_{xx}$ ) $_{x=0}$  (kPa)')
266     plt.title('Error in Normal stress along the vertical symmetry')
267     plt.xlim(0.5, 2)
268
269     for width, mesh in zip(widths, meshes):
270         path = "Run" + str(width) + '-' + str(mesh) + '/postProcessing/sets/100/leftPatch_sigmaxx_sigmayy.xy'
271         data = np.loadtxt(path)
272
273         if widths.count(widths[0]) == len(widths):
274             label = 'Explicit Solution ( $n=$  + str(int(mesh)) + '$)'

```

```

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

```

```

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

```