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 $x = 4$m and radius $R = 0.5$m. It is loaded with a uniform traction of $\sigma = 10$kPa over its left and right faces as can be seen in Figure 1.

A mesh sensitivity study is performd for a plate with side length $x = 4$m, and an effect of the plate length study is performed for plates of length $x = 3$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:

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

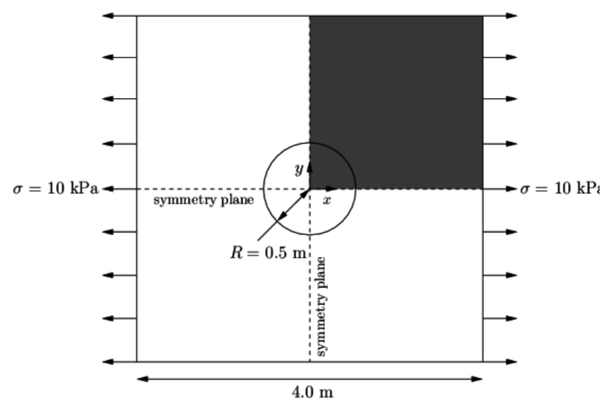

Fig. 1. Geometry of the plate with a hole [2]

A Python script was created to automatically generate the configuration files, calculate the resulting steady-state stress through the plate using OpenFOAM [1], 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 multiple of the characteristic number of points, n, in the x_i and y_i directions. 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 $x = 4$m. The effect of plate length study looks at plate lengths of $x = 3$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

The computational result for normal stress along the vertical symmetry can be compared to the analytical result above, Equation 1. The Root Mean Square error,

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[\sigma_i - \sigma_i^*]^2}, \tag{2}$$

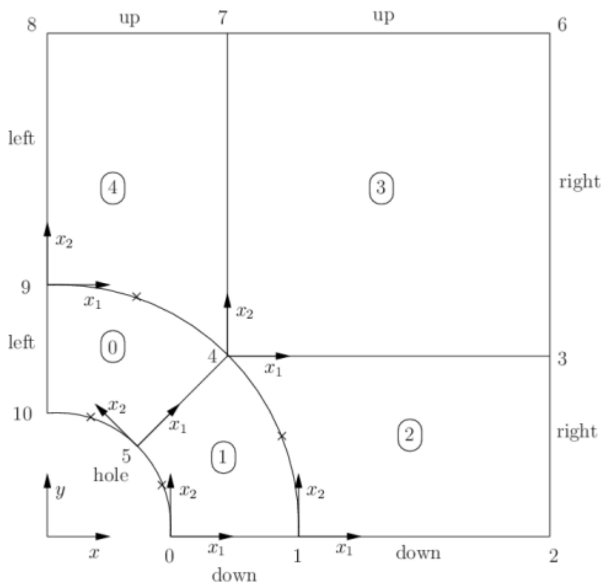

Fig. 2. Block structure of the mesh for the plate with a hole [2]

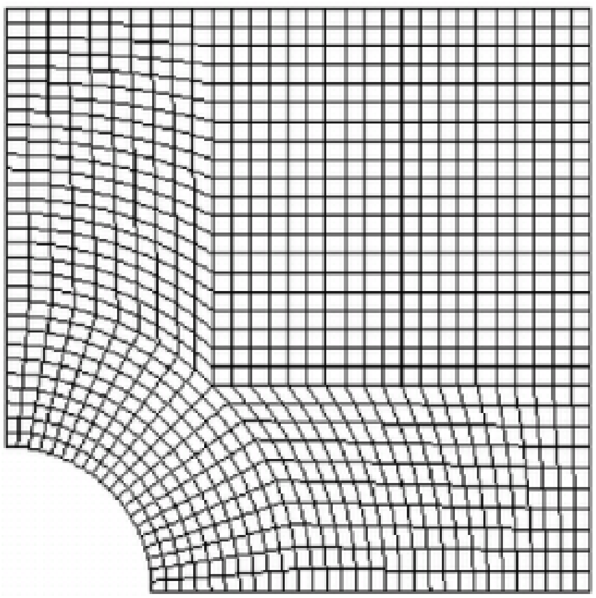

Fig. 3. Mesh of the hole in a plate problem with $n = 10$ [2]

and the Normalized Root Mean Square error,

$$\text{NRMS} = \frac{RMSE}{max(\sigma^*) - min(\sigma^*)}, \tag{3}$$

can be calculated. Here σ_i is the computational result for the the stress for each point along the boundary, σ_i^* is the analytical solution, and N is the number of points along the boundary.

3.1 Selected key results for the base case

The base case described in the OpenFOAM tutorial involves a plate of width $x = 4$ with a mesh resulting from $n = 10$ [2]. The NRMS from the analytical solution for this case is approximately 9.41%. The results for the normal stress and the deviation from the analytical solution for this case can be seen in Figure 4.

3.2 Mesh sensitivity study

The mesh sensitivity study looks at meshes resulting from $n = 10, 100$, and 1000 on a plate width of $x = 4$m.

$x(m)$	n	$NRMS$
4	10	9.41%
4	100	9.72%
4	1000	9.01%

Table 1. NRMS results from the numerical simulations for the mesh sensitivity study

3.3 Effect of the plate length study

The effect of plate length study looks at plate lengths of $x = 3$m, 4m, 5m, and 100m with a mesh of $n = 10$.

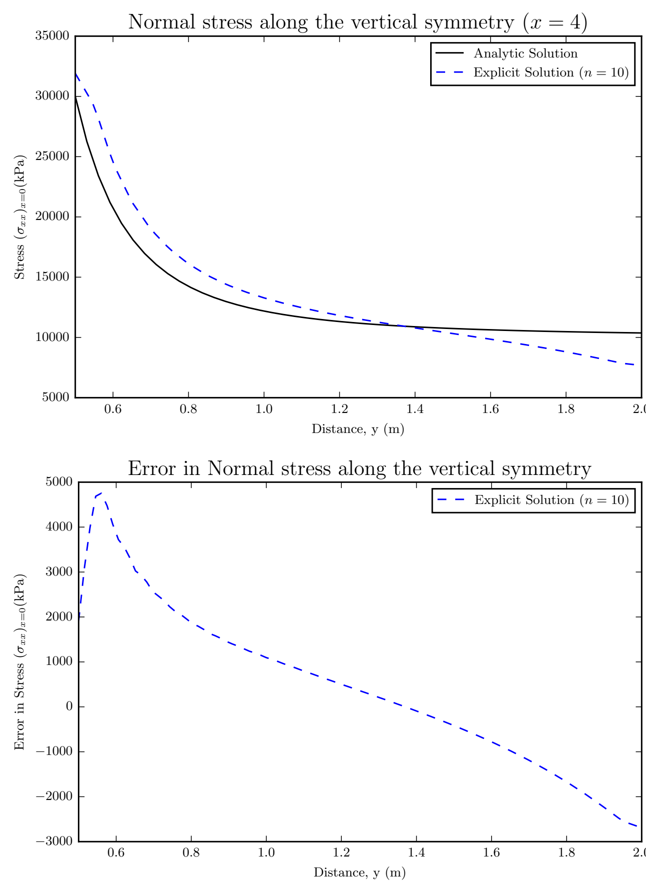

Fig. 4. Results of base case

$x(m)$	n	$NRMS$
3	10	14.58%
4	10	9.41%
5	10	6.22%
100	10	3.11%

Table 2. NRMS results from the numerical simulations for the effect of plate length study

4 Conculsion

References

[1] Jasak, H., Jemcov, A., and Tukovic, Z., 2007. "Openfoam: A c++ library for complex physics simulations". In International workshop on coupled methods in numerical dynamics, Vol. 1000, pp. 1–20.

[2] OpenFOAM_Foundation, 2014. Stress analysis of a plate with a hole. http://www.openfoam.org/docs/user/plateHole.php.

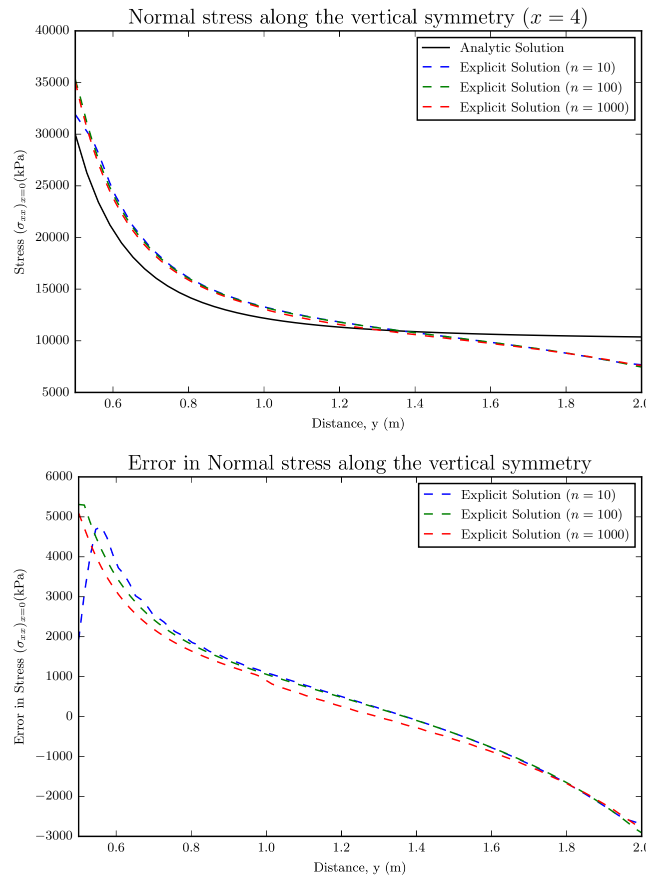

Fig. 5. Results of mesh sensitivity study

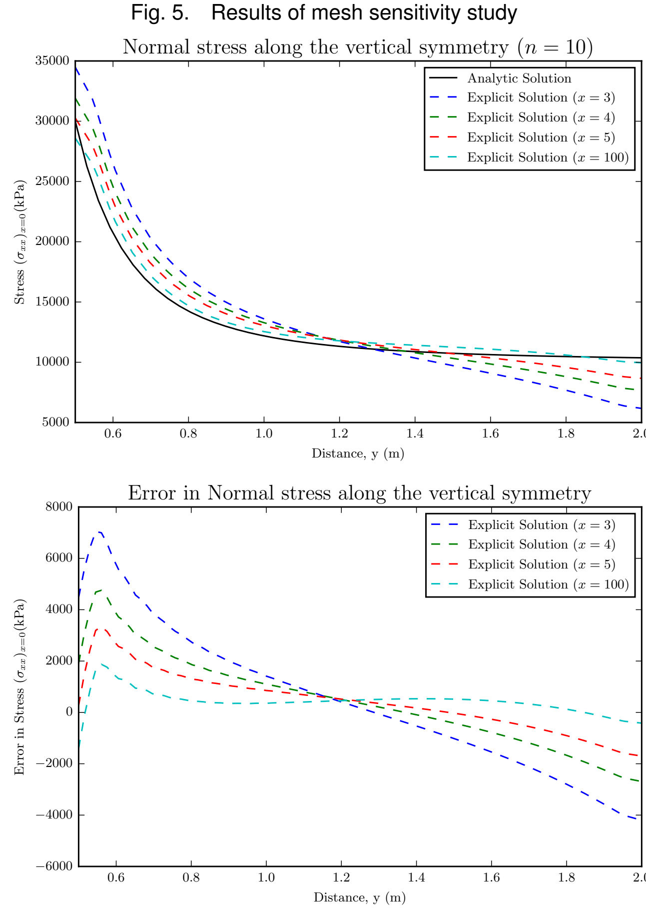

Fig. 6. Results of plate length study

Appendix A: Python Code

```
import subprocess
import numpy as np
import matplotlib.pyplot as plt
import os
import time
import datetime


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


def generate_folders(widths, meshes):
    for width, mesh in zip(widths, meshes):
        run = "Run" + str(width) + '-' + str(mesh)
        if not os.path.exists(run):
            command = "cp -rf base/ " + run + "/; "
            subprocess_cmd(command)

    print ('Folders generated.')


def create_config_file(width, mesh):
    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 1;


    vertices
    (
        (0.5 0 0)
        (1 0 0)
        (''' + str(width) + ''' 0 0)
        (''' + str(width) + ''' 0.707107 0)
        (0.707107 0.707107 0)
        (0.353553 0.353553 0)
        (''' + str(width) + ''' 2 0)
        (0.707107 2 0)
```

```
        (0 2 0)
        (0 1 0)
        (0 0.5 0)
        (0.5 0 0.5)
        (1 0 0.5)
        (''' + str(width) + ''' 0 0.5)
        (''' + str(width) + ''' 0.707107 0.5)
        (0.707107 0.707107 0.5)
        (0.353553 0.353553 0.5)
        (''' + str(width) + ''' 2 0.5)
        (0.707107 2 0.5)
        (0 2 0.5)
        (0 1 0.5)
        (0 0.5 0.5)
    );

    blocks
    (
        hex (5 4 9 10 16 15 20 21) (''' + str(mesh) + ' ' + str(mesh) + ''' 1) simpleGrading (1 1 1)
        hex (0 1 4 5 11 12 15 16) (''' + str(mesh) + ' ' + str(mesh) + ''' 1) simpleGrading (1 1 1)
        hex (1 2 3 4 12 13 14 15) (''' + str(mesh * 2) + ' ' + str(mesh) + ''' 1) simpleGrading (1 1 1)
        hex (4 3 6 7 15 14 17 18) (''' + str(mesh * 2) + ' ' + str(mesh * 2) + ''' 1) simpleGrading (1 1 1)
        hex (9 4 7 8 20 15 18 19) (''' + str(mesh) + ' ' + str(mesh * 2) + ''' 1) simpleGrading (1 1 1)
    );

    edges
    (
        arc 0 5 (0.469846 0.17101 0)
        arc 5 10 (0.17101 0.469846 0)
        arc 1 4 (0.939693 0.34202 0)
        arc 4 9 (0.34202 0.939693 0)
        arc 11 16 (0.469846 0.17101 0.5)
        arc 16 21 (0.17101 0.469846 0.5)
        arc 12 15 (0.939693 0.34202 0.5)
        arc 15 20 (0.34202 0.939693 0.5)
    );

    boundary
    (
        left
        {
            type symmetryPlane;
            faces
            (
                (8 9 20 19)
                (9 10 21 20)
            );
        }
        right
        {
            type patch;
            faces
            (
                (2 3 14 13)
                (3 6 17 14)
            );
        }
        down
        {
            type symmetryPlane;
            faces
            (
                (0 1 12 11)
                (1 2 13 12)
            );
        }
        up
        {
            type patch;
```

```
            faces
            (
                (7 8 19 18)
                (6 7 18 17)
            );
        }
        hole
        {
            type patch;
            faces
            (
                (10 5 16 21)
                (5 0 11 16)
            );
        }
        frontAndBack
        {
            type empty;
            faces
            (
                (10 9 4 5)
                (5 4 1 0)
                (1 4 3 2)
                (4 7 6 3)
                (4 9 8 7)
                (21 16 15 20)
                (16 11 12 15)
                (12 13 14 15)
                (15 14 17 18)
                (15 18 19 20)
            );
        }
    );

    mergePatchPairs
    (
    );

    // ************************************************************************* //

    '''

    return config


def update_dimensions(widths, meshes):
    for width, mesh in zip(widths, meshes):
        run = "Run" + str(width) + '-' + str(mesh)
        path = run + '/constant/polyMesh/blockMeshDict'
        with open(path, 'w') as config_file:
            config_file.write(create_config_file(width, mesh))

    print ('Config generated.')


def run_simulations(widths, meshes):
    for width, mesh in zip(widths, meshes):
        run = "Run" + str(width) + '-' + str(mesh)
        if not os.path.exists(run + '/100/'):
            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 += "solidDisplacementFoam > log; "
            command += "foamCalc components sigma; "
            command += "sample"
            subprocess_cmd(command)
```

```python
        print(run + ' complete.')

    print('Simulations complete.')


def sigma_xx(x):
    return 1E4*(1+(0.125/(x**2))+(0.09375/(x**4)))


# this has not been found analytically
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.')


def main(widths, meshes):
    print('Running widths ' + str(widths) + ' with meshes ' + str(meshes) + '.')
    generate_folders(widths, meshes)
    update_dimensions(widths, meshes)
    run_simulations(widths, meshes)
    generate_plots(widths, meshes)
    print('Done!')

if __name__ == "__main__":
    # Base case
    widths = [2]
    meshes = [10]
    main(widths, meshes)

    # Increasing mesh resolution
    widths = [2, 2, 2]
    meshes = [10, 100, 1000]
    main(widths, meshes)

    # Changing the plate size
    widths = [1.5, 2, 2.5, 50]
    meshes = [10 for _ in widths]
    main(widths, meshes)
```