

AFEM: Axisymmetric Project

Derek Lontine, Stuart Childs, Alex Bailey

April 28, 2016

Chapter 1

Introduction

The focus of this project is to implement different types of axisymmetric elements for stress displacement loadings, and compare error raised by different material properties, specifically, incompressible materials to analytical solutions for those loadings.

In order to accomplish that, the analytical, and computational implementation of the elements are examined and developed for use. The loading scenarios are described along with the explicit analytical solutions to those loadings. Armed with elements to test, and solutions to test against, error analysis of the elements is performed and compared to any trends, with some of the behavior being well behaved, while other behavior seems unpredictable.

Chapter 2

Analytical Formulation

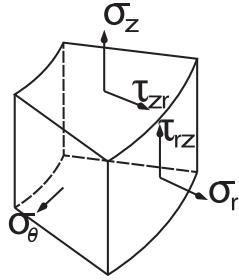


Figure 2.1: An infinitesimal volume unit within the axisymmetric solid of revolution.

The analytical formulation for this problem is very straight forward as it follows the Felliqa formulation. We begin with the displacement.

$$\mathbf{u}(r, z) = \begin{bmatrix} u_r(r, z) \\ u_z(r, z) \end{bmatrix} \quad (2.1)$$

We note that the strain is defined as:

$$[\mathbf{e}] = \begin{bmatrix} e_{rr} & e_{rz} & 0 \\ e_{rz} & e_{zz} & 0 \\ 0 & 0 & e_{\theta\theta} \end{bmatrix} \quad (2.2)$$

where

$$e_{rr} = \frac{\partial u_r}{\partial r}, \quad e_{zz} = \frac{\partial u_z}{\partial z}, \quad e_{\theta\theta} = \frac{\partial u_\theta}{\partial \theta}, \quad \gamma_{rz} = \frac{\partial u_r}{\partial z} + \frac{\partial u_z}{\partial r} = e_{rz} + e_{zr} = 2e_{rz} \quad (2.3)$$

then

$$\mathbf{e} = \begin{bmatrix} e_{rr} \\ e_{zz} \\ e_{\theta\theta} \\ \gamma_{rz} \end{bmatrix} = \begin{bmatrix} e_{rr} \\ e_{zz} \\ e_{\theta\theta} \\ 2e_{rz} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial r} & 0 \\ 0 & \frac{\partial}{\partial z} \\ \frac{1}{2} & 0 \\ \frac{\partial}{\partial z} & \frac{\partial}{\partial r} \end{bmatrix} \begin{bmatrix} u_r(r, z) \\ u_z(r, z) \end{bmatrix} = \mathbf{D}\mathbf{u} \quad (2.4)$$

The off axis zero terms are in the place of the torsional shear terms in the strain, as there are no shear loads in the simple axisymmetric loading scenarios encountered. The displacement interpolation is what we're used to:

$$\begin{bmatrix} u_r \\ u_z \end{bmatrix} = \begin{bmatrix} N_1^e & 0 & N_2^e & 0 & \dots & N_n^e & 0 \\ 0 & N_1^e & 0 & N_2^e & \dots & 0 & N_n^e \end{bmatrix} \mathbf{u}^e = \mathbf{N}\mathbf{u}^e \quad (2.5)$$

The Strain-Displacement Matrix, \mathbf{B} , is therefore:

$$\mathbf{e} = \begin{bmatrix} e_{rr} \\ e_{zz} \\ e_{\theta\theta} \\ \gamma_{rz} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1^e}{\partial r} & 0 & \frac{\partial N_2^e}{\partial r} & 0 & \dots & \frac{\partial N_n^e}{\partial r} & 0 \\ 0 & \frac{\partial N_1^e}{\partial z} & 0 & \frac{\partial N_2^e}{\partial z} & \dots & 0 & \frac{\partial N_n^e}{\partial z} \\ \frac{N_1^e}{r} & 0 & \frac{N_2^e}{r} & 0 & \dots & \frac{N_n^e}{r} & 0 \\ \frac{\partial N_1^e}{\partial z} & \frac{\partial N_1^e}{\partial r} & \frac{\partial N_2^e}{\partial z} & \frac{\partial N_2^e}{\partial r} & \dots & \frac{\partial N_n^e}{\partial z} & \frac{\partial N_n^e}{\partial r} \end{bmatrix} \mathbf{u}^e = \mathbf{B} \mathbf{u}^e \quad (2.6)$$

The Fellipa finite element formulation is:

$$\mathbf{K}^e = \sum_{k=1}^p \sum_{l=1}^p w_k w_l \mathbf{B}^T \mathbf{E} \mathbf{B} r J_\Omega \quad (2.7)$$

$$\mathbf{f} = \int_{\Omega^e} r \mathbf{N}^T \mathbf{b} d\Omega + \int_{\Gamma^e} r \mathbf{N}^T \hat{\mathbf{t}} d\Gamma \quad (2.8)$$

The system is solved using a simply implemented linsolve function since:

$$[\mathbf{K}]^e \{\mathbf{u}\}^e = \{\mathbf{f}\}^e \quad (2.9)$$

2.0.1 Integration Methods

For the two elements which will be discussed, there was a choice of full integration, or reduced integration. As we know, reduced integration is the reduction of the number of Gauss Points to evaluate the \mathbf{K} matrix over. As one professor's course notes stated: "Displacement-based FE formulations always over-estimate the stiffness matrix and the use of fewer integration points should produce a less stiff element. Therefore, in some cases, particularly non-linear problems such as plasticity, creep or incompressible materials, it is actually advisable to use reduced integration instead of full integration. The slight loss of accuracy is counteracted by the improvement in approximation to real-life behaviour."

Chapter 3

Computational Implementation

The axisymmetric elements produced are fundamentally quite simple. They modify the basic quad element found in pyfem2. They were developed and tested using the latest version of pyfem2 as given in update bea5af7 (4/19/2016).

Each of these elements has several things in common. Each of these elements modifies the $[B]$ matrix so that the third row ($B[2,0::2]$) includes the shape functions divided by the radius from the axis of symmetry.

Additionally, each of these elements has a group of “formulation setting” functions to access the different modifications to the stiffness matrix as defined in chapter 2 in the Galerkin and Petrov-Galerkin formulations of axisymmetry.

3.1 Full integration

The full integration element was not developed by the authors, however, it is used heavily in the verification problems. This element is the base element and other elements such as the selective reduced and reduced integration elements are only slight modifications of this the base element.

The full integration element applies gauss points at $(x, y) = (\pm\sqrt{\frac{1}{3}}, \pm\sqrt{\frac{1}{3}})$ inside the element. This is applying 2 gauss points for each dimension. The gauss weight for each of the gauss points is equal to 1.

As will be discussed in great detail, this element can exhibit problems such as shear and volume locking in different scenarios.

```
1 from numpy import *
2 from .isop2_4 import CSDIsoParametricQuad4 as BaseElement
3 # -----
4 # ----- Axisymmetric Quad Element ----- #
5 #
6 class AxiSymmetricQuad4(BaseElement):
7     ndir = 3
8     nshr = 1
9     integration = 4
10    elefab = {'formulation': 1}
11    gaussw = ones(4)
12    gaussp = array([[-1., -1.], [ 1., -1.], [-1.,  1.], [ 1.,  1.]]) / sqrt(3.)
13    @property
14    def formulation(self):
15        return self.axisymmetric
16    @formulation.setter
17    def formulation(self, arg):
18        assert arg in (0, 1, 2)
19        self.axisymmetric = arg
20    def bmatrix(self, dN, N, xi, *args):
21        rp = dot(N, self.xc[:,0])
22        B = zeros((4, 8))
23        B[0, 0::2] = B[3, 1::2] = dN[0, :]
24        B[1, 1::2] = B[3, 0::2] = dN[1, :]
25        B[2, 0::2] = N / rp
```

```
26     return B
```

3.2 Reduced Integration with Hourglass Control

```
1 from numpy import *
2 from .isop2_4 import CSDIsoParametricQuad4 as BaseElement
3 #
4 # ----- Axisymmetric Reduced Integration With Hourglass Element ----- #
5 #
6 class AxiSymmetricQuad4Reduced( BaseElement ):
7     ndir = 3
8     nshr = 1
9     integration = 1
10    elefab = { 'formulation': 1}
11    gaussw = array([4.])
12    gaussp = array([[0.,0.]])
13    hourglass_control = True
14    #HOURGLASS CONTROL PARAMETERS
15    hglassp = array([[0.,0.,0.]])
16    hglassv = array([[1.,-1.,1.,-1.]])
17    #REST
18    @property
19    def formulation(self):
20        return self.axisymmetric
21    @formulation.setter
22    def formulation(self, arg):
23        assert arg in (0, 1, 2)
24        self.axisymmetric = arg
25    def bmatrix(self, dN, N, xi, *args):
26        rp = dot(N, self.xc[:,0])
27        B = zeros((4, 8))
28        B[0, 0::2] = B[3, 1::2] = dN[0, :]
29        B[1, 1::2] = B[3, 0::2] = dN[1, :]
30        B[2, 0::2] = N / rp
31        return B
```

3.3 Implementation

The following listing is required for `__init__.py` in order for the elements above to be implemented into pyfem2. Note that the primary changes to the previously generated pyfem2 in the commit mentioned above is found in lines 4,5,22,23. The AxisymmetricQuad4 element was implemented in pyfem2 without any contribution of the authors.

```
1 __all__ = [ 'PlaneStrainTria3' , 'PlaneStressTria3' ,
2            'PlaneStrainQuad4BBar' , 'PlaneStrainQuad4' , 'PlaneStrainQuad4Reduced' ,
3            'PlaneStrainQuad4SelectiveReduced' , 'PlaneStressQuad4' ,
4            'AxiSymmetricQuad4' , 'AxiSymmetricQuad4SelectiveReduced' ,
5            'AxiSymmetricQuad4Reduced' ,
6            'PlaneStressQuad4Incompat' , 'PlaneStrainQuad8BBar' ,
7            'PlaneStrainQuad8' , 'PlaneStrainQuad8Reduced' , 'PlaneStressQuad8' ,
8            'CSDIsoParametricElement' , 'IsoPElement' ]
9
10 from .isolib import CSDIsoParametricElement
11 IsoPElement = CSDIsoParametricElement
12
13 from .CSDT3EF import PlaneStrainTria3
14 from .CSDT3SF import PlaneStressTria3
15
16 from .CSDQ4EB import PlaneStrainQuad4BBar
17 from .CSDQ4EF import PlaneStrainQuad4
18 from .CSDQ4ER import PlaneStrainQuad4Reduced
19 from .CSDQ4ES import PlaneStrainQuad4SelectiveReduced
20
21 from .CSDAX4F import AxiSymmetricQuad4
```

```
22 from .CSDAX4S import AxiSymmetricQuad4SelectiveReduced
23 from .CSDAX4R import AxiSymmetricQuad4Reduced
24
25 from .CSDQ4SF import PlaneStressQuad4
26 from .CSDQ4SI import PlaneStressQuad4Incompat
27
28 from .CSDQ8EB import PlaneStrainQuad8BBar
29 from .CSDQ8EF import PlaneStrainQuad8
30 from .CSDQ8ER import PlaneStrainQuad8Reduced
31
32 from .CSDQ8SF import PlaneStressQuad8
```

Chapter 4

Verification Problems

These are the test load scenarios the axisymmetric elements were subjected to, while ranging constants E and ν . For solid plates and washers, the constant D is defined as:

$$D = \frac{Et^3}{12(1 - \nu^2)}$$

4.1 Solid Plates

4.1.1

Simply supported circular plate with a point load at the center. The maximum deflection of the plate, occurs

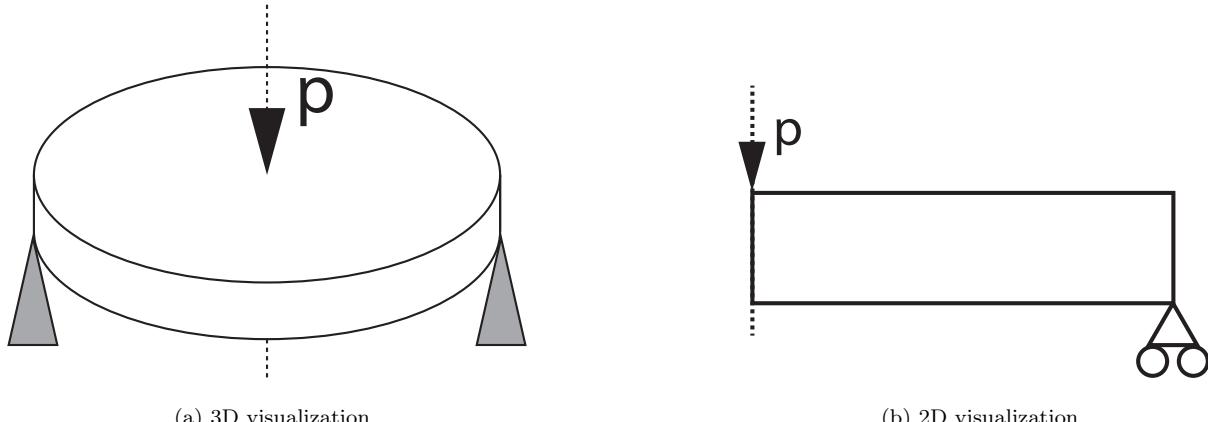


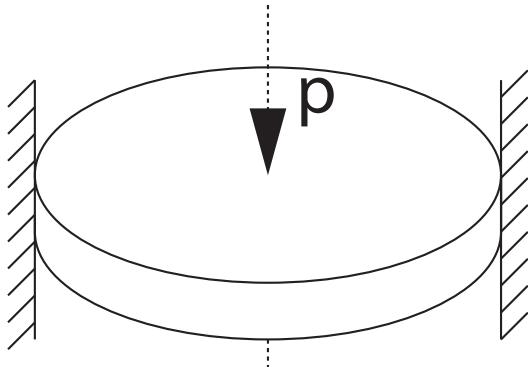
Figure 4.1: A pinned plate under a point load.

at the center of the plate and is defined by:

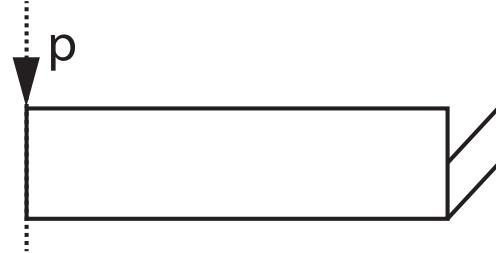
$$y_{max} = \frac{-pR^2}{16\pi D} \frac{3 + \nu}{1 + \nu}$$

4.1.2

A circular plate with a fixed edge boundary condition, and point load at the center.



(a) 3D visualization



(b) 2D visualization

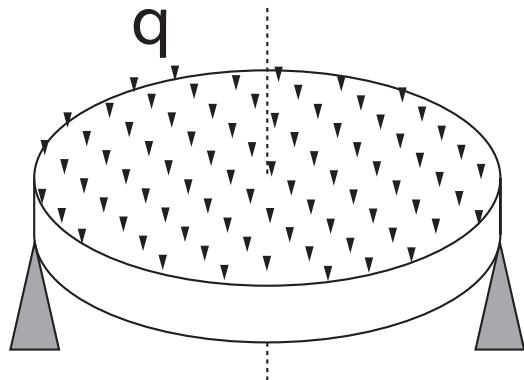
Figure 4.2: A fixed edge plate under a point load.

The maximum deflection of the plate, occurs at the center of the plate and is defined by:

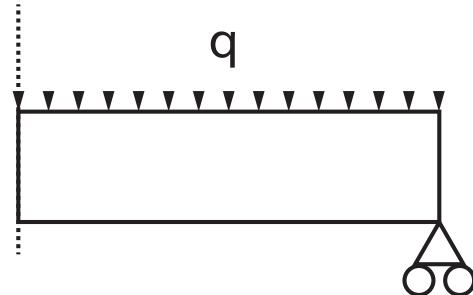
$$y_{max} = \frac{-pR^2}{16\pi D}$$

4.1.3

Simply supported circular plate with a distributed load over the entire plate surface.



(a) 3D visualization



(b) 2D visualization

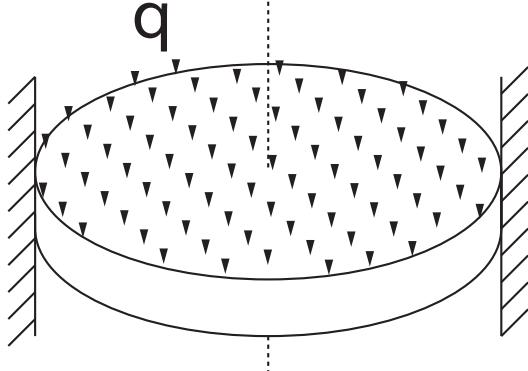
Figure 4.3: A pinned plate under a pressure load.

The maximum deflection of the plate, occurs at the center of the plate and is defined by:

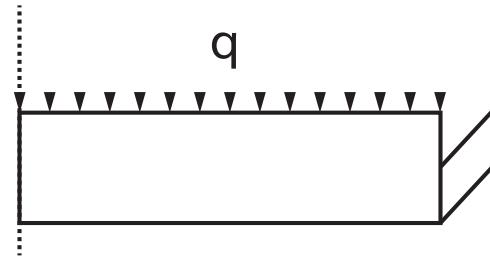
$$y_{max} = \frac{-qR^4(5 + \nu)}{64D(1 + \nu)}$$

4.1.4

Simply supported circular plate with a distributed load over the entire plate surface.



(a) 3D visualization



(b) 2D visualization

Figure 4.4: A fixed edge plate under a pressure load.

The maximum deflection of the plate, occurs at the center of the plate and is defined by:

$$y_{max} = \frac{-qrR^4}{64D}$$

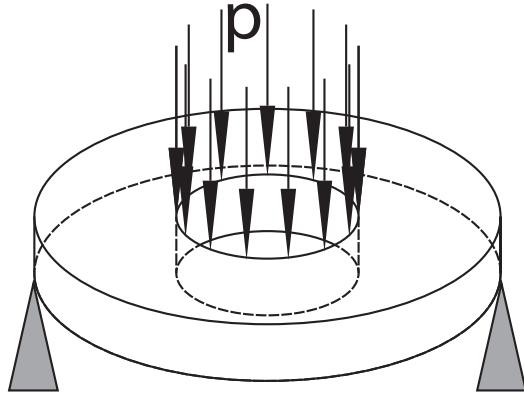
4.2 Plate With a Hole (Washers)

The following values are used repetitively through the y_{max} equations for washers:

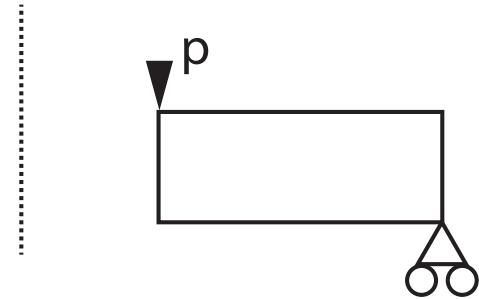
$$\begin{aligned} C_1 &= \frac{1+\nu}{2} \frac{r}{R} \ln \frac{R}{r} + \frac{1-\nu}{4} \left(\frac{R}{r} - \frac{r}{R} \right) \\ C_4 &= \frac{1}{2} \left((1+\nu) \frac{r}{R} + (1-\nu) \frac{R}{r} \right) \\ C_7 &= \frac{1}{2} (1-\nu^2) \left(\frac{R}{r} - \frac{r}{R} \right) \\ L_3 &= \frac{r}{4R} \left(\left(\left(\frac{r}{R} \right)^2 + 1 \right) \ln \frac{R}{r} + \left(\frac{r}{R} \right)^2 - 1 \right) \\ L_6 &= \frac{r}{4R} \left(\left(\frac{r}{R} \right)^2 - 1 + 2 \ln \frac{R}{r} \right) \\ L_9 &= \frac{r}{R} \left(\frac{1+\nu}{2} \ln \frac{R}{r} + \frac{1-\nu}{4} \left(1 - \left(\frac{r}{R} \right)^2 \right) \right) \\ L_{11} &= \frac{1}{64} \left(1 + 4 \left(\frac{r}{R} \right)^2 - 5 \left(\frac{r}{R} \right)^4 - 4 \left(\frac{r}{R} \right)^2 \left[2 + \left(\frac{r}{R} \right)^2 \right] \ln \frac{r}{R} \right) \\ L_{14} &= \frac{1}{16} \left(1 - \left(\frac{r}{R} \right)^4 - 4 \left(\frac{r}{R} \right)^2 \ln \frac{R}{r} \right) \\ L_{17} &= \frac{1}{4} \left(1 - \frac{1-\nu}{4} \left(1 - \left(\frac{r}{R} \right)^4 \right) - \left(\frac{r}{R} \right)^2 \left(1 + (1+\nu) \ln \frac{R}{r} \right) \right) \end{aligned}$$

4.2.1

Simply supported circular washer with a line load on the inner circumference. The maximum deflection of



(a) 3D visualization



(b) 2D visualization. The dotted line is the axis of symmetry running through the center of the washer.

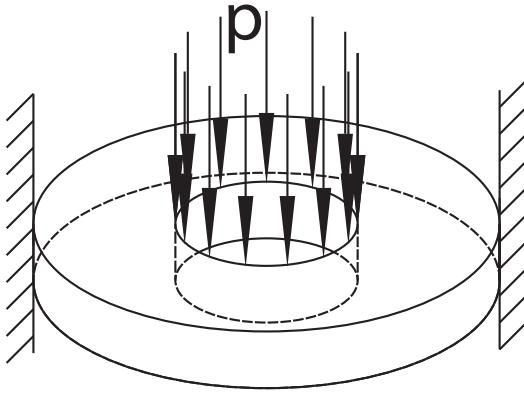
Figure 4.5: A pinned washer under a line load on the inner circumference.

the plate, occurs at the inner circumference of the washer and is defined by:

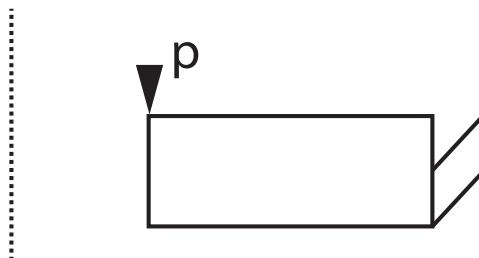
$$y_{max} = \frac{-pR^3}{D} \left(\frac{C_1 L_9}{C_7} - L_3 \right)$$

4.2.2

A circular washer under a fixed edge boundary condition with a line load on the inner circumference. The



(a) 3D visualization



(b) 2D visualization. The dotted line is the axis of symmetry running through the center of the washer.

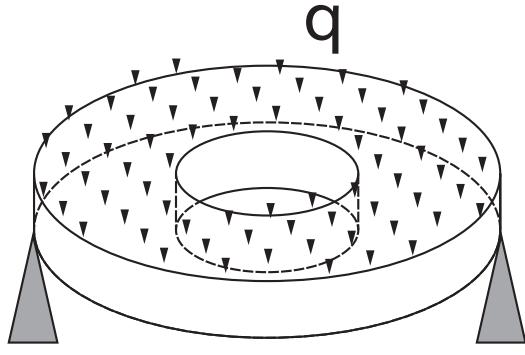
Figure 4.6: A fixed edge washer under a line load on the inner circumference.

maximum deflection of the plate, occurs at the inner circumference of the washer and is defined by:

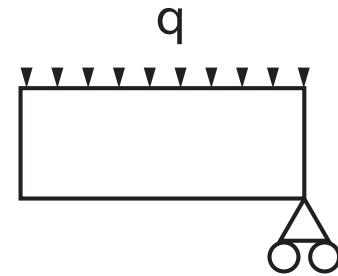
$$y_{max} = \frac{-pR^3}{D} \left(\frac{C_1 L_6}{C_4} - L_3 \right)$$

4.2.3

A circular washer under a fixed edge boundary condition with a line load on the inner circumference. The



(a) 3D visualization



(b) 2D visualization. The dotted line is the axis of symmetry running through the center of the washer.

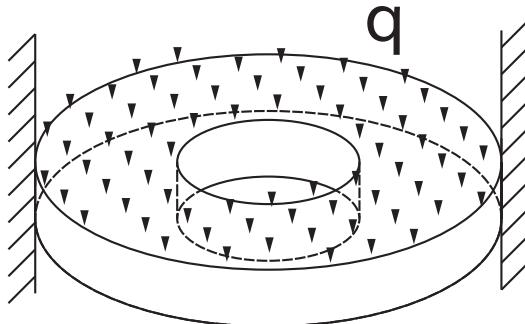
Figure 4.7: A pinned washer under a distributed load over the surface of the washer.

maximum deflection of the plate, occurs at the inner circumference of the washer and is defined by:

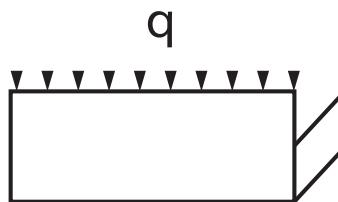
$$y_{max} = \frac{-qR^4}{D} \left(\frac{C_1 L_{17}}{C_7} - L_{11} \right)$$

4.2.4

A circular washer under a fixed edge boundary condition with a line load on the inner circumference. The



(a) 3D visualization



(b) 2D visualization. The dotted line is the axis of symmetry running through the center of the washer.

Figure 4.8: A pinned washer under a distributed load over the surface of the washer.

maximum deflection of the plate, occurs at the inner circumference of the washer and is defined by:

$$y_{max} = \frac{-qR^4}{D} \left(\frac{C_1 L_{14}}{C_4} - L_{11} \right)$$

Chapter 5

Performance Evaluation

The following section evaluates each of the elements developed (or attempted to be developed) and interrogates their performance relative to the following criteria:

- Convergence of a metric to a solution
- Convergence of a metric toward the analytical solution
- Stability of element in several loading states
- Suitability of element under various material parameters (E, ν)

As mentioned above, the primary elements of interest are the CSDAX4F (full integration) and CSDAX4R (reduced integration) elements. We attempted to develop a selective reduced integration element (CSDAX4S), however, as will be shown, the performance metrics failed so miserably, that the element is useless much more development work must take place in order for such an element to be functional.

5.1 Determining Differences Between Analytical and Numerical Solutions

Although much more rigorous methods could likely be applied to determining the difference between the analytical and numerical solution for FEM problems, the more simple the metric, the more easily understood it is. Therefore, we determined that an acceptable metric is the maximum displacement of the plate. This maximum deflection has well defined solutions (cataloged in great detail in Roark's solution manual).

In order to extract the maximum deflection from the FEM solution, it is critical to interrogate each node individually for its displacement in the Z axis. Therefore, the following function was constructed in order to interrogate each node for its displacement.

```
1 def get_max_disp(V,**kwargs):
2     uy_max=0
3     uy_max2=0
4     Xi=array(V.mesh.coord)
5     ui=array(V.steps.last.dofs.reshape(Xi.shape))
6     for ym in ui[:,1]:
7         asym=abs(ym)
8         if uy_max < asym:
9             uy_max=asym
10            uy_max2=ym
11    return uy_max2
```

If interrogating each individual point, and comparing each FEM point to an analytical solution, a more rigorous function that returns the x & y displacements as well as the original positions X and Y is:

```
1 def get_disp_pos(GP,V,**kwargs):
2     Xi=array(V.mesh.coord)
```

```

3     ui=array(V.steps.last.dofs.reshape(Xi.shape))
4     ux = ui[GP,0]
5     uy = ui[GP,1]
6     X = Xi[GP,0]
7     Y = Xi[GP,1]
8     return ux,uy,X,Y

```

Finally, if an $nx4$ array is desired to pass to testing functions for analysis, the following function may come in use. We did not end up implementing these functions into any of the testing framework that we developed, however, if one desired to test all points against an analytical solution, this function would be useful.

```

1 def get_all_disp_pos(V,**kwargs):
2     nnodes=V.numnod
3     data = array(zeros((nnodes,4)))
4     for ii in range(0,nnodes):
5         data[ii,:]=get_disp_pos(ii,V)
6     return data

```

5.2 Convergence

An error metric was developed which was the error between the numerical solution Y_{num} and the analytical solution Y_a . Although there are many error metrics, the one determined most useful for the purposes of determining convergence upon a *solution* is a simplistic difference error. This error, which will be utilized in the vertical axis of the plots in this section and can be summarized as:

$$error = Y_{FEM} - Y_a \quad (5.1)$$

The following listing provides a function for which one can pass an analysis system into in order to determine the errors associated with that analysis system. Note that this code also returns the number of elements and the percent error.

```

1 def Comp_Analysis(E,v,P,OD,h,inD=None,
2                     NinX=None,NinY=None,eletyp=None,formula=None,
3                     Model_Comparison_Function=None,A_Mod.Comp.Fun=None,**kwargs):
4     d=dict({'E':E,'v':v,'P':P,'OD':OD,'h':h,'inD':inD,'NinX':NinX,'NinY':NinY,
5             'eletyp':eletyp,'formula':formula})
6     V=Model_Comparison_Function(**d)
7     zFEM = get_max_disp(V)
8     zANA = A_Mod.Comp_Fun(**d)
9     err=(zFEM-zANA)#/zANA*100.
10    err2=(zFEM-zANA)/zANA*100.
11    print(Model_Comparison_Function)
12    print(eletyp)
13    print('Numerical Sol is: ',zFEM)
14    print('Analytical Sol is: ',zANA)
15    print('Absolute Error is: ',err)
16    print('Percent Error is: ',err2)
17    nele=V.numele
18    return err,err2,nele

```

5.2.1 CSDAX4F- Full Integration

The full integration element was the most successful in converging in nearly all loading conditions. As can be seen from the series of figures following, the element seems to follow excellent h-refinement convergence. The primary driver in this h-refinement is the generic mesh density. However, if greater detail is applied to ensuring a very square element, instead of rectangular elements, convergence is much more rapid.

In addition to this mesh dependence, it is important to note that if the mesh density in the Z direction is increased without an increase in the R mesh density, the model fails to converge with h-refinement. However, when both are applied (as shown in the images below), h-refinement convergence is much more rapid.

It was generally observed that fewer than 4 elements in the Z directions were unsatisfactory for producing a mesh that gave believable results for the full integration element. Not a great deal of work was exerted toward interrogating this fact, but it is notable nonetheless.

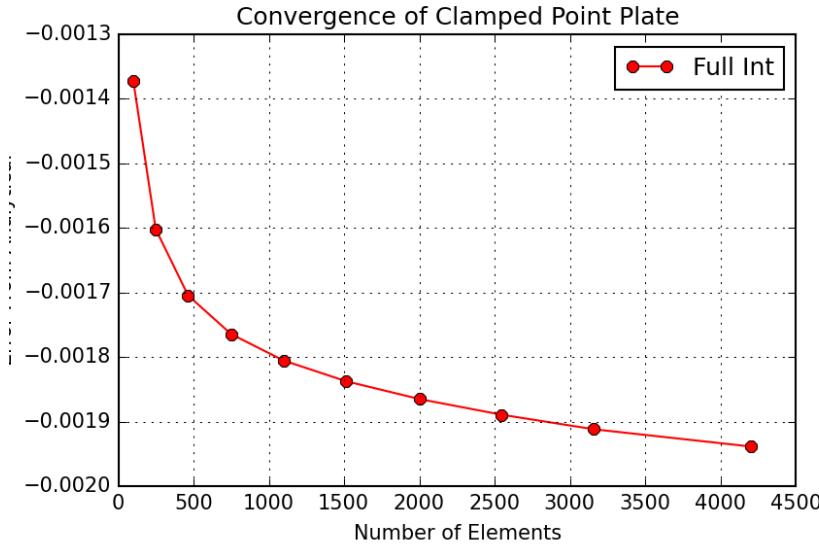


Figure 5.1: H refinement of CSDAX4F under the loading condition described above in section 4.1.2. This loading condition was the axisymmetric plate with a central point load and clamped outer boundary.

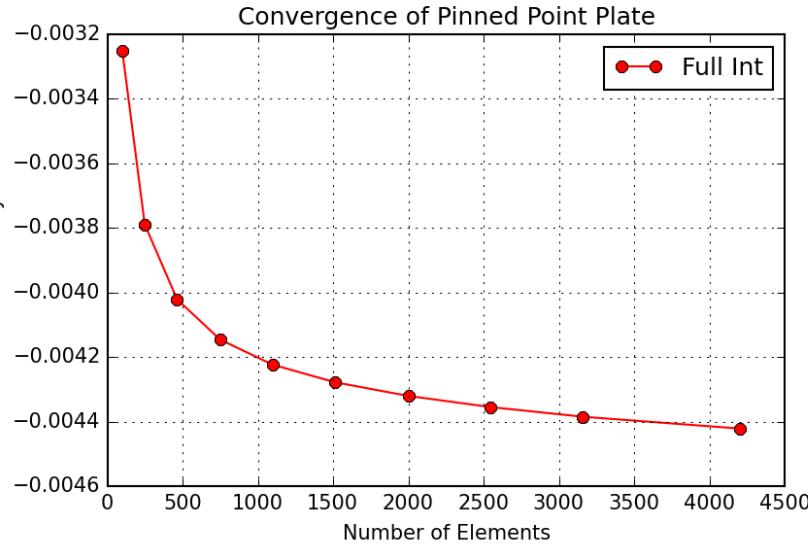


Figure 5.2: H refinement of CSDAX4F under the loading condition described above in section 4.1.1. This loading condition was the axisymmetric plate with a central point load and pinned outer boundary.

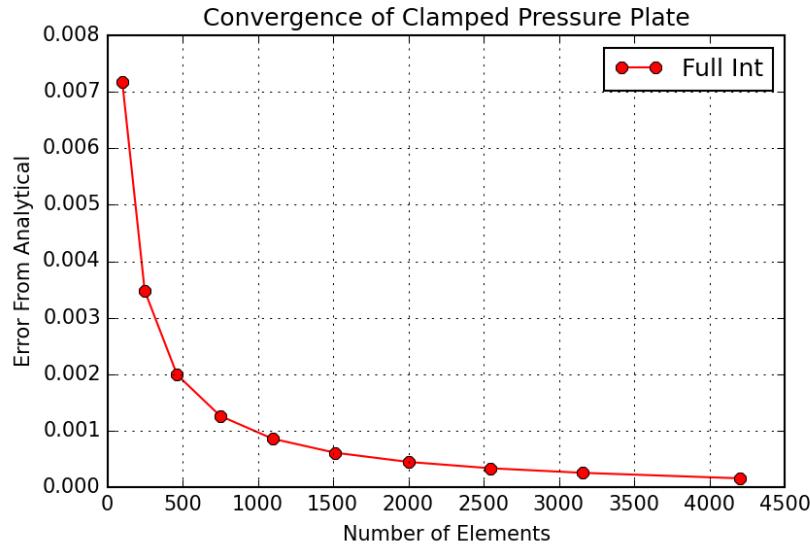


Figure 5.3: H refinement of CSDAX4F under the loading condition described above in section 4.1.4. This loading condition was the axisymmetric plate with a uniform pressure load and clamped outer boundary.

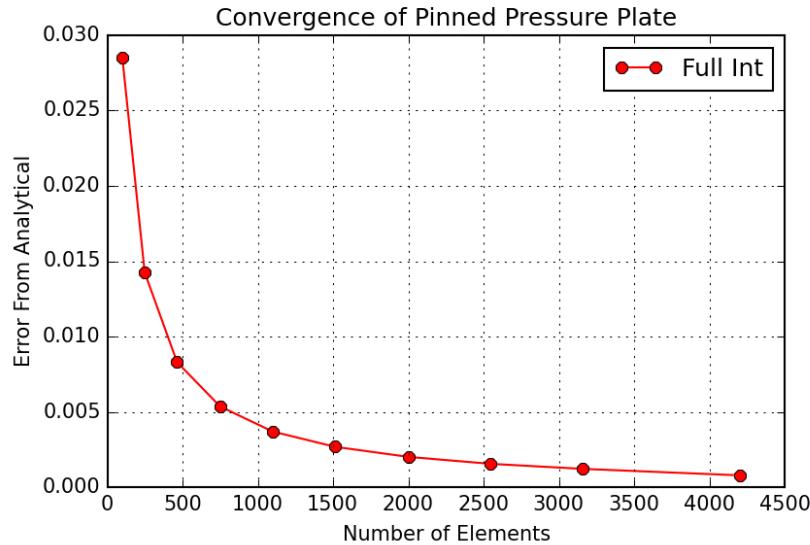


Figure 5.4: H refinement of CSDAX4F under the loading condition described above in section 4.1.3. This loading condition was the axisymmetric plate with a uniform pressure load and pinned outer boundary.

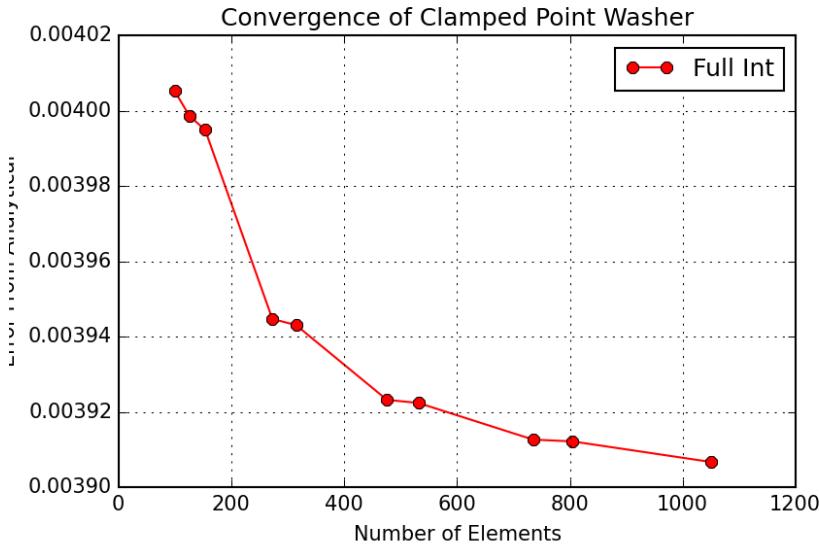


Figure 5.5: H refinement of CSDAX4F under the loading condition described above in section 4.2.2. This loading condition was the axisymmetric washer with a point load on the inside diameter of the washer and clamped outer boundary.

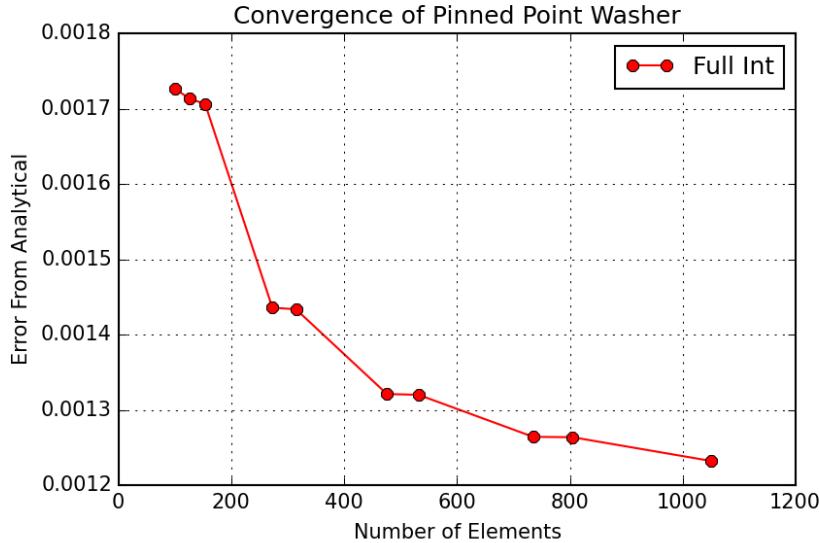


Figure 5.6: H refinement of CSDAX4F under the loading condition described in section 4.2.1. This loading condition was the axisymmetric washer with a point load on the inside diameter of the washer and pinned outer boundary.

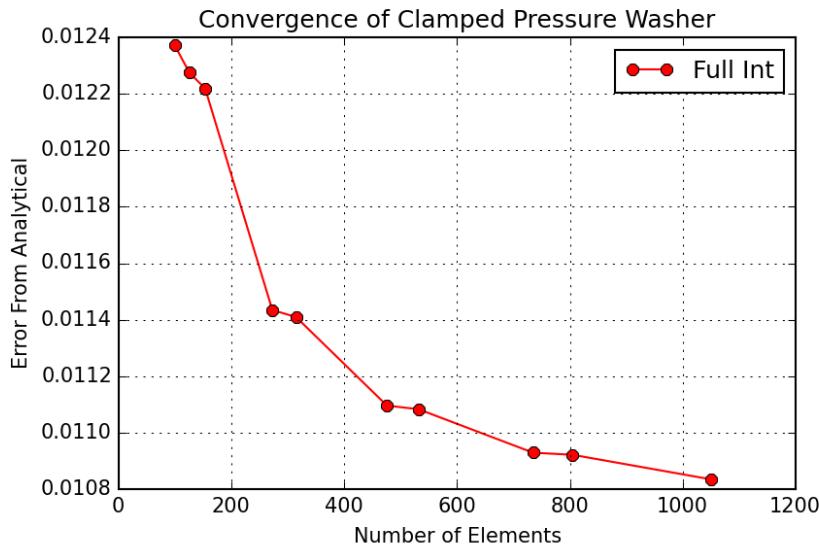


Figure 5.7: H refinement of CSDAX4F under the loading condition described in section 4.2.4. This loading condition was the axisymmetric washer with a uniform pressure load and clamped outer boundary.

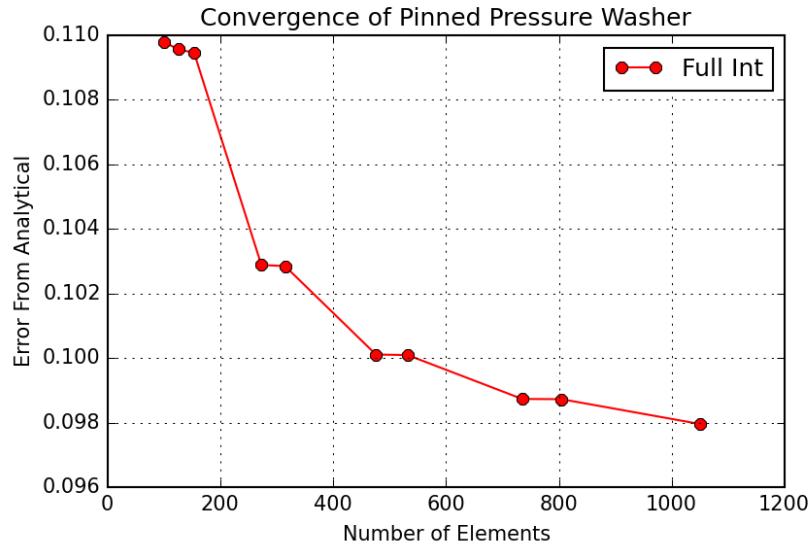


Figure 5.8: H refinement of CSDAX4F under the loading condition described in section 4.2.1. This loading condition was the axisymmetric washer with a uniform pressure load and pinned outer boundary.

5.2.2 CSDAX4R- Reduced Integration

The reduced integration element struggled to converge in all loading situations. The root cause of this inconsistency is not clear, however, it is clear that whenever the pinned boundary condition is applied to the reduced integration element, the model fails to converge on a solution with h-refinement.

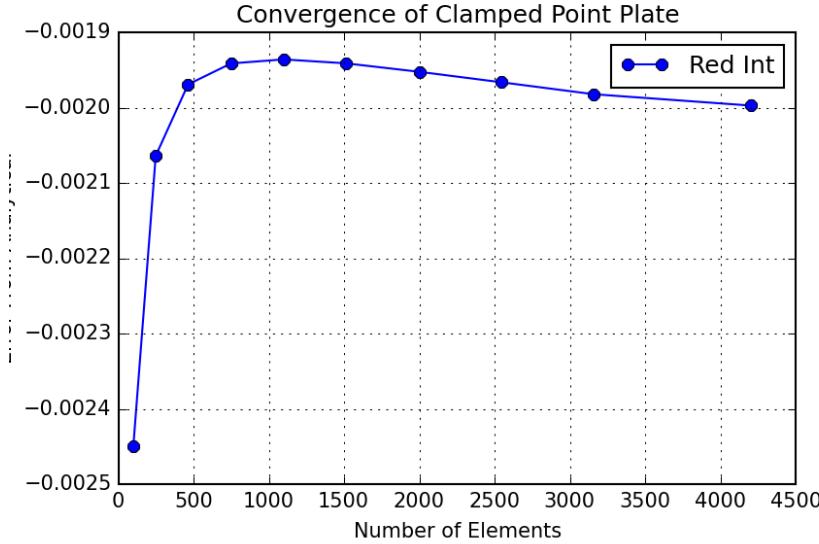


Figure 5.9: H refinement of CSDAX4R under the loading condition described above in section 4.1.2. This loading condition was the axisymmetric plate with a central point load and clamped outer boundary.

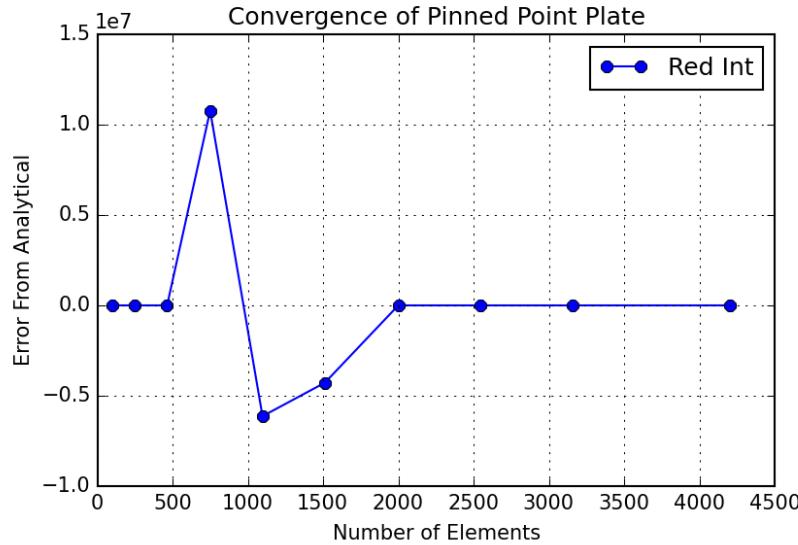


Figure 5.10: H refinement of CSDAX4R under the loading condition described above in section 4.1.1. This loading condition was the axisymmetric plate with a central point load and pinned outer boundary.

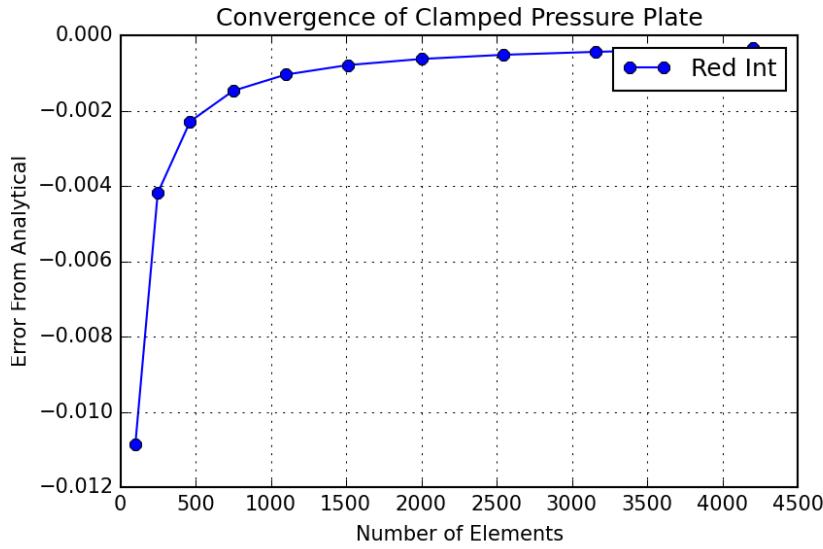


Figure 5.11: H refinement of CSDAX4R under the loading condition described above in section 4.1.4. This loading condition was the axisymmetric plate with a uniform pressure load and clamped outer boundary.

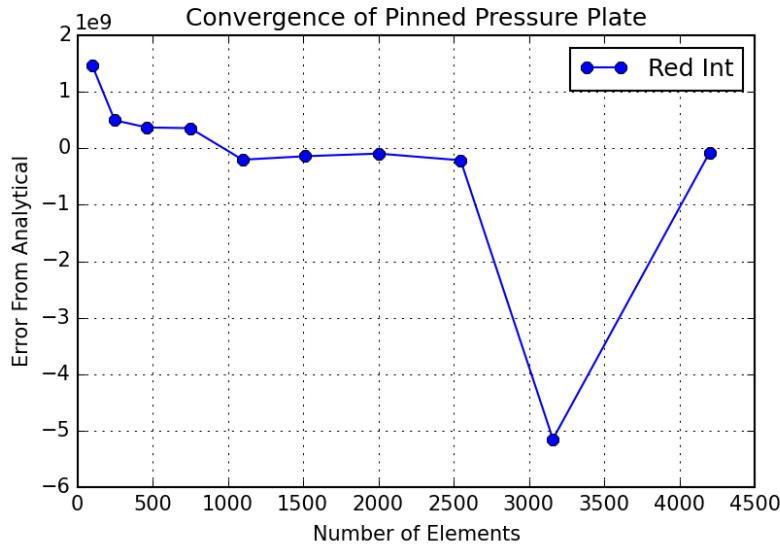


Figure 5.12: H refinement of CSDAX4R under the loading condition described above in section 4.1.3. This loading condition was the axisymmetric plate with a uniform pressure load and pinned outer boundary.

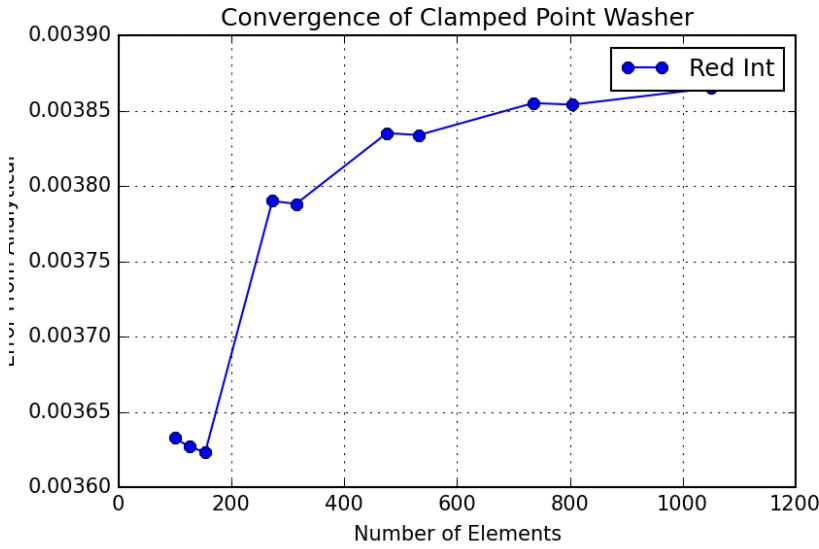


Figure 5.13: H refinement of CSDAX4R under the loading condition described above in section 4.2.2. This loading condition was the axisymmetric washer with a point load on the inside diameter of the washer and clamped outer boundary.

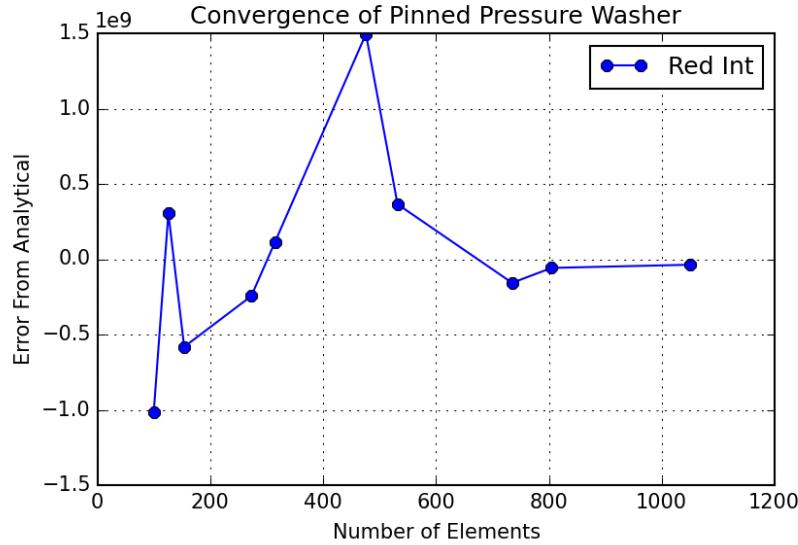


Figure 5.14: H refinement of CSDAX4R under the loading condition described in section 4.2.1. This loading condition was the axisymmetric washer with a point load on the inside diameter of the washer and pinned outer boundary.

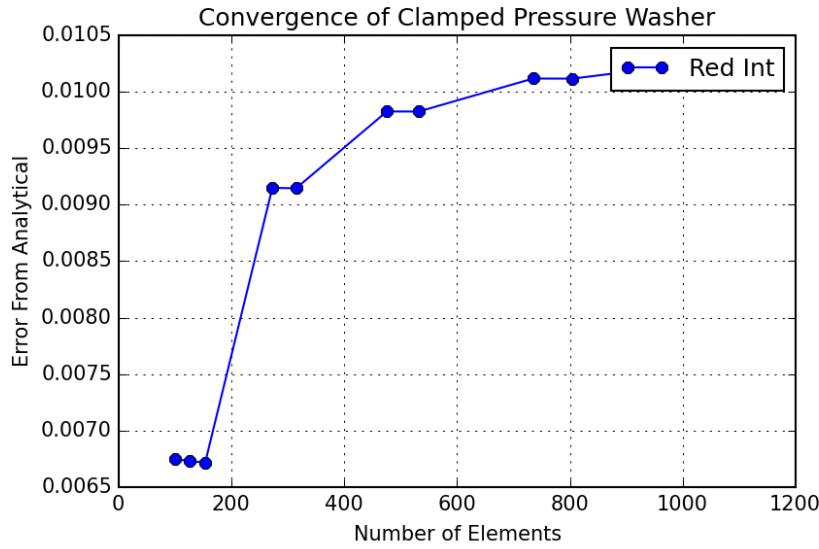


Figure 5.15: H refinement of CSDAX4R under the loading condition described in section 4.2.4. This loading condition was the axisymmetric washer with a uniform pressure load and clamped outer boundary.

5.2.3 Comparing Convergence of Both Elements

The following plots compare the convergence of the CSDAX4F and CSDAX4R element types. Note that some plots (particularly the plots with pinned joints) have extreme scales due to the errors and non-convergence associated with the reduced integration element. This shows that there is clearly some malfunction in the operation of the pin constraint when operating on the reduced integration element. As mentioned earlier, there is no known solution to this as of yet.

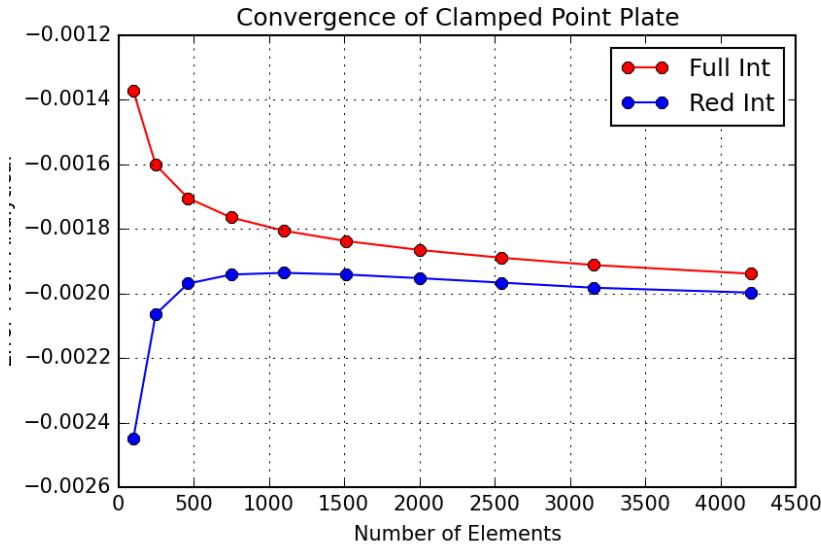


Figure 5.16: Comparison of CSDAX4F and CSDAX4R under the loading condition described above in section 4.1.2. This loading condition was the axisymmetric plate with a central point load and clamped outer boundary.

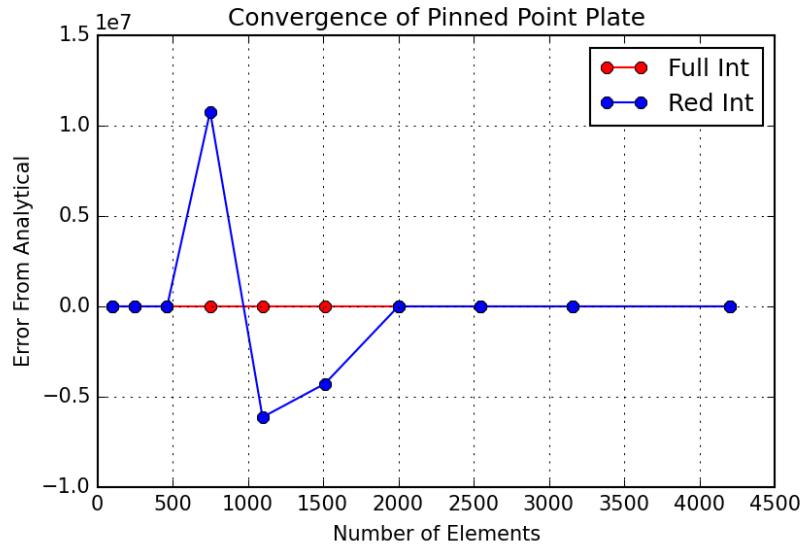


Figure 5.17: Comparison of CSDAX4F and CSDAX4R under the loading condition described above in section 4.1.1. This loading condition was the axisymmetric plate with a central point load and pinned outer boundary.

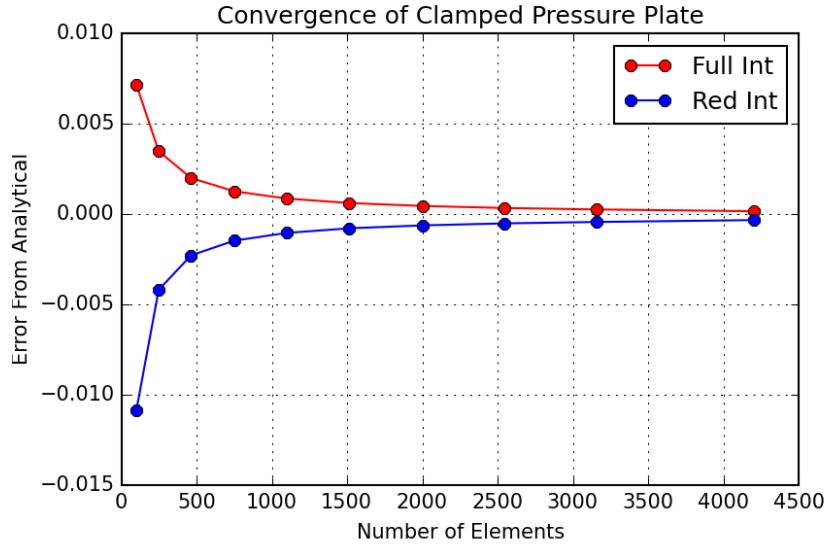


Figure 5.18: Comparison of CSDAX4F and CSDAX4R under the loading condition described above in section 4.1.4. This loading condition was the axisymmetric plate with a uniform pressure load and clamped outer boundary.

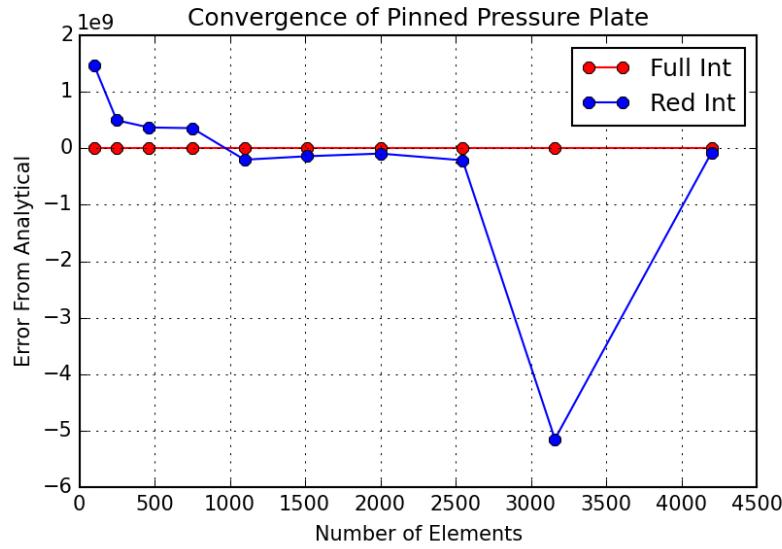


Figure 5.19: Comparison of CSDAX4F and CSDAX4R under the loading condition described above in section 4.1.3. This loading condition was the axisymmetric plate with a uniform pressure load and pinned outer boundary.

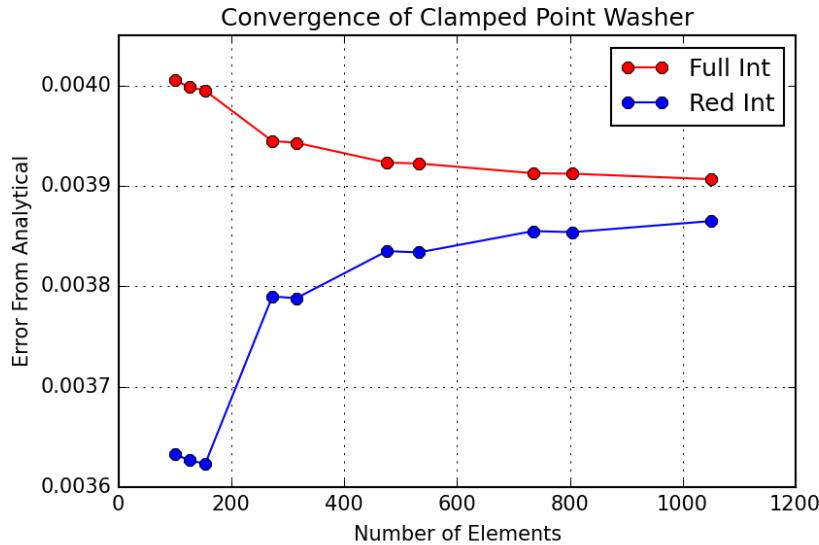


Figure 5.20: Comparison of CSDAX4F and CSDAX4R under the loading condition described above in section 4.2.2. This loading condition was the axisymmetric washer with a point load on the inside diameter of the washer and clamped outer boundary.

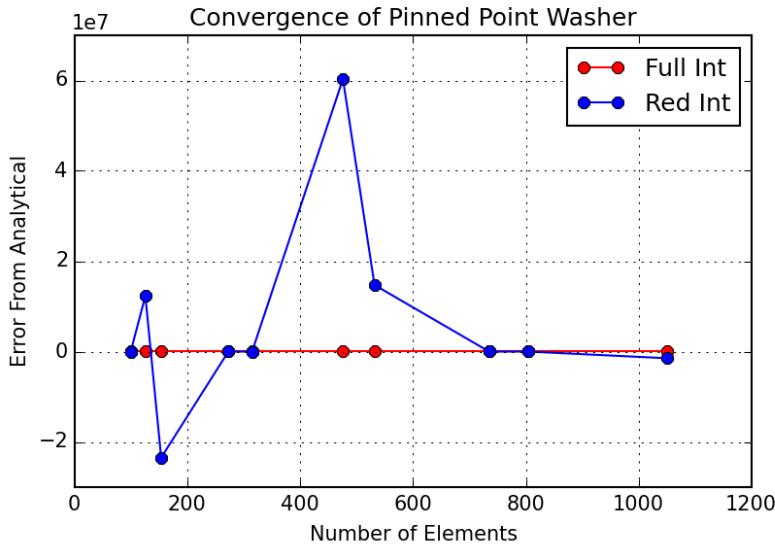


Figure 5.21: Comparison of CSDAX4F and CSDAX4R under the loading condition described in section 4.2.1. This loading condition was the axisymmetric washer with a point load on the inside diameter of the washer and pinned outer boundary.

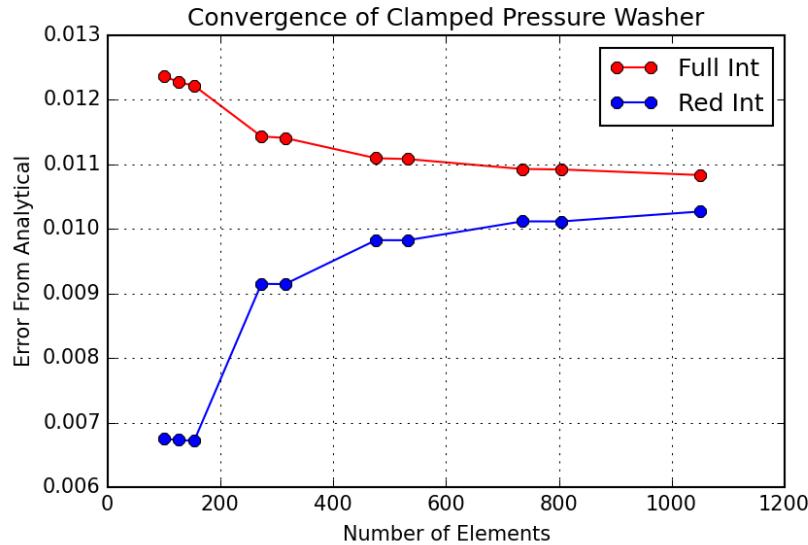


Figure 5.22: Comparison of CSDAX4F and CSDAX4R under the loading condition described in section 4.2.4. This loading condition was the axisymmetric washer with a uniform pressure load and clamped outer boundary.

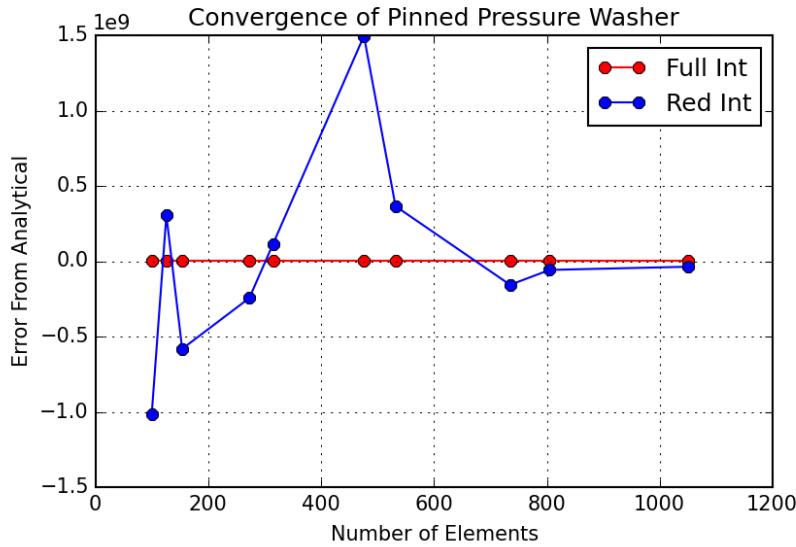


Figure 5.23: Comparison of CSDAX4F and CSDAX4R under the loading condition described in section 4.2.3. This loading condition was the axisymmetric washer with a uniform pressure load and pinned outer boundary.

5.3 Convergence toward analytical solutions

Convergence toward analytical solutions is somewhat sketchy at this point. This is primarily because we are unsure if the analytical solutions are providing the appropriate response to what we are observing with the finite element solutions. Because the convergence plots above do not all go to zero, this shows that there is some problems with at least a few of the analytical solutions. The analytical solutions that look very good are the clamped and pinned boundary conditions on the uniform plate (no hole) with a uniform pressure load applied.

We are unsure, why the concentrated loads are not functioning properly, however, we suspect it has something to do with applying a concentrated load at the center of the axisymmetric element. At this point, $r=0$ and some rather strange things can begin to happen. Because the radius of the gauss integration point is very small, this may cause the FEM displacement to be too small. This is reflected in the h-refinement images shown in figures 5.1 and 5.2. In fact, in both of those figures, the error seems to converge to a slope rather than a level.

5.4 Functions utilized to determine convergence etc

The following listings are functions that are utilized in determining the convergence of the model.

```

1 import sys
2 import pdb
3 import matplotlib
4 sys.path.insert(0, '../')
5 from pyfem2 import *
6 from Definitions import *
7 from Definitions2 import *
8 from mpl_toolkits.mplot3d import Axes3D
9 from matplotlib import cm
10 from matplotlib import pyplot as plt
11
12 def find_convergence(Model_Comparison_Function,
13                      A_Mod_Comp_Fun,
14                      nstep=None, xmax=None, title=None,
```

```

15             saveas1=None, saveas2=None,
16             saveas3=None, saveas4=None, **kwargs):
17     if nstep is None:
18         nstep=7
19     if xmax is None:
20         xmax=200
21     if title is None:
22         title='Convergence on Model'
23     if saveas1 is None:
24         saveas1='Convergence_A.png'
25     if saveas2 is None:
26         saveas2='Convergence_F.png'
27     if saveas2 is None:
28         saveas2='Convergence_R.png'
29     if saveas2 is None:
30         saveas2='Convergence_S.png'
31     pconv=dict({'P':40,
32                 'OD':23,
33                 'h':.4,
34                 'formula':1,
35                 'inD':23/2,
36                 'E':5e7,
37                 'v':0.4,
38                 'Model_Comparison_Function':Model_Comparison_Function,
39                 'A_Mod_Comp_Fun':A_Mod_Comp_Fun})
40     ymax=int(pconv['h']*xmax/pconv['OD'])*3
41     ninx=linspace(50,xmax,nstep)
42     niny=linspace(3,ymax,nstep)
43
44     er_f=zeros(nstep)
45     ne_f=zeros(nstep)
46     erp_f=zeros(nstep)
47     for ii ,nx in enumerate(ninx):
48         pconv['NinX']=int(ninx[ii])
49         pconv['NinY']=int(niny[ii])
50         pconv['eletyp']=AxiSymmetricQuad4
51         #er_f[ii], ne_f[ii]=Model_Comparison_Function(**pconv)
52         er_f[ii], erp_f[ii], ne_f[ii]=Comp_Analysis(**pconv)
53
54     er_r=zeros(nstep)
55     ne_r=zeros(nstep)
56     erp_r=zeros(nstep)
57     for ii ,nx in enumerate(ninx):
58         pconv['NinX']=int(ninx[ii])
59         pconv['NinY']=int(niny[ii])
60         pconv['eletyp']=AxiSymmetricQuad4Reduced
61         #er_r[ii], ne_r[ii]=Model_Comparison_Function(**pconv)
62         er_r[ii], erp_r[ii], ne_r[ii]=Comp_Analysis(**pconv)
63
64     er_s=zeros(nstep)
65     ne_s=zeros(nstep)
66     erp_s=zeros(nstep)
67     for ii ,nx in enumerate(ninx):
68         pconv['NinX']=int(ninx[ii])
69         pconv['NinY']=int(niny[ii])
70         pconv['eletyp']=AxiSymmetricQuad4SelectiveReduced
71         #er_s[ii], ne_s[ii]=Model_Comparison_Function(**pconv)
72         er_s[ii], erp_s[ii], ne_s[ii]=Comp_Analysis(**pconv)
73
74     plt.plot(ne_f,er_f,marker='o', linestyle='--', color='r',label='Full Int')
75     plt.plot(ne_r,er_r,marker='o', linestyle='--', color='b',label='Red Int')
76     #plt.plot(ne_s,er_s,marker='o', linestyle='--', color='g',label='Sel Red Int')
77     plt.xlabel('Number of Elements')
78     plt.ylabel('Error From Analytical')
79     plt.title(title)
80     plt.legend()
81     plt.grid(True)
82     plt.savefig(saveas1)

```

```

83     plt.show()
84
85
86     plt.plot(ne_f,er_f,marker='o', linestyle='-', color='r',label='Full Int')
87     #plt.plot(ne_r,er_r,marker='o', linestyle='-', color='b',label='Red Int')
88     #plt.plot(ne_s,er_s,marker='o', linestyle='-', color='g',label='Sel Red Int')
89     plt.xlabel('Number of Elements')
90     plt.ylabel('Error From Analytical')
91     plt.title(title)
92     plt.legend()
93     plt.grid(True)
94     plt.savefig(saveas2)
95     plt.show()
96
97     #plt.plot(ne_f,er_f,marker='o', linestyle='-', color='r',label='Full Int')
98     plt.plot(ne_r,er_r,marker='o', linestyle='-', color='b',label='Red Int')
99     #plt.plot(ne_s,er_s,marker='o', linestyle='-', color='g',label='Sel Red Int')
100    plt.xlabel('Number of Elements')
101    plt.ylabel('Error From Analytical')
102    plt.title(title)
103    plt.legend()
104    plt.grid(True)
105    plt.savefig(saveas3)
106    plt.show()
107
108    #plt.plot(ne_f,er_f,marker='o', linestyle='-', color='r',label='Full Int')
109    #plt.plot(ne_r,er_r,marker='o', linestyle='-', color='b',label='Red Int')
110    plt.plot(ne_s,er_s,marker='o', linestyle='-', color='g',label='Sel Red Int')
111    plt.xlabel('Number of Elements')
112    plt.ylabel('Error From Analytical')
113    plt.title(title)
114    plt.legend()
115    plt.grid(True)
116    plt.savefig(saveas4)
117    plt.show()

```

The following listing describes one potential dictionary and function call of the above definition:

```

1 pdict=dict({ 'Model_Comparison_Function':Plate_Point_Pinned ,
2             'A_Mod_Comp_Fun':A_Plate_Point_Pinned ,
3             'nstep':nsteps ,
4             'xmax': 350 ,
5             'title':'Convergence of Pinned Point Plate' ,
6             'saveas1':'Conv_PlPoPi_1.png' ,
7             'saveas2':'Conv_PlPoPi_2.png' ,
8             'saveas3':'Conv_PlPoPi_3.png' ,
9             'saveas4':'Conv_PlPoPi_4.png'})}
10 find_convergence(**pdict)

```

5.5 FEM code

The following listing provides functions for each of the analysis systems described above. The naming convention is quite plain. Several FEM function definitions were not utilized in this report, however, they are given for future reference. One notable function is the thick walled pressure vessel function.

```

1 def Plate_Point_Pinned(E,v,P,OD,h,
2                         NinX=None ,NinY=None ,eletyp=None ,formula=None ,
3                         **kwargs):
4     if eletyp is None:
5         eletyp = AxiSymmetricQuad4
6     if NinX is None:
7         NinX = 100 #Number of elements in I (Diameter)
8     if NinY is None:
9         NinY = 4 #Number elements in J (Thickness)
10    if formula is None:
11        formula=1

```

```

12 R=OD/2.0
13 kp1=NinY*(NinX+1)+1 #Central point
14 kp2=NinX+1           #Bottom outside edge
15 mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=R, ly=h)
16 mat = Material('Material-1', elastic={'E':E, 'Nu':v})
17
18 V = FiniteElementModel(mesh=mesh, jobid='PlatePointPinned')
19 V.ElementBlock('ElementBlock1', ALL)
20 V.AssignProperties('ElementBlock1', eletyp, mat, formulation=formula)
21
22 #V.PrescribedBC(kp2,Y)
23 step = V.StaticStep()
24 step.PinNodes(kp2)
25 #step.FixNodes(kp2)
26 #step.PrescribedBC(kp2,Y)
27 step = V.StaticStep()
28 step.ConcentratedLoad(kp1, Y, -P)
29 step.run()
30 V.WriteResults()
31 if not os.environ.get('NOGRAPHICS'):
32     V.Plot2D(show=1, deformed=1)
33 return V
34
35 def Plate_Point_Clamped(E,v,P,OD,h,
36                         NinX=None, NinY=None, eletyp=None, formula=None,
37                         **kwargs):
38     if eletyp is None:
39         eletyp = AxiSymmetricQuad4
40     if NinX is None:
41         NinX=100 #Number of elements in I (Diameter)
42     if NinY is None:
43         NinY=4 #Number elements in J (Thickness)
44     if formula is None:
45         formula=1
46 R=OD/2.0
47 #Find Keypoints of interest
48 kp1=NinY*(NinX+1)+1 #Central point
49 kp2=NinX+1           #Bottom outside edge
50
51 mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=R, ly=h)
52 mat = Material('Material-1', elastic={'E':E, 'Nu':v})
53
54 V = FiniteElementModel(mesh=mesh, jobid='PlatePointClamped')
55 V.ElementBlock('ElementBlock1', ALL)
56 V.AssignProperties('ElementBlock1', eletyp, mat, formulation=formula)
57
58 step = V.StaticStep()
59 step.FixNodes(IHI)
60 step = V.StaticStep()
61 step.ConcentratedLoad(kp1, Y, -P)
62 step.run()
63 V.WriteResults()
64 if not os.environ.get('NOGRAPHICS'):
65     V.Plot2D(show=1, deformed=1)
66 return V
67
68 def Plate_Pressure_Pinned(E,v,P,OD,h,
69                           NinX=None, NinY=None, eletyp=None, formula=None,
70                           **kwargs):
71     if eletyp is None:
72         eletyp = AxiSymmetricQuad4
73     if NinX is None:
74         NinX = 100 #Number of elements in I (Diameter)
75     if NinY is None:
76         NinY = 4 #Number elements in J (Thickness)
77     if formula is None:
78         formula=1
79 R=OD/2.0

```

```

80 kp1=NinY*(NinX+1)+1 #Central point
81 kp2=NinX+1           #Bottom outside edge
82 mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=R, ly=h)
83 mat = Material('Material-1', elastic={'E':E, 'Nu':v})
84
85 V = FiniteElementModel(mesh=mesh, jobid='PlatePressurePinned')
86 V.ElementBlock('ElementBlock1', ALL)
87 V.AssignProperties('ElementBlock1', eletyp, mat, formulation=formula)
88
89 #V.PrescribedBC(kp2,Y)
90 step = V.StaticStep()
91 step.PinNodes(kp2)
92 #step.FixNodes(kp2)
93 #step.PrescribedBC(kp2,Y)
94 step = V.StaticStep()
95 step.Pressure(JHI, P)
96 step.run()
97 V.WriteResults()
98 if not os.environ.get('NOGRAPHICS'):
99     V.Plot2D(show=1, deformed=1)
100 return V
101
102
103 def Plate_Pressure_Clamped(E,v,P,OD,h,
104                             NinX=None, NinY=None, eletyp=None, formula=None,
105                             **kwargs):
106     if eletyp is None:
107         eletyp = AxiSymmetricQuad4
108     if NinX is None:
109         NinX = 100 #Number of elements in I (Diameter)
110     if NinY is None:
111         NinY = 4 #Number elements in J (Thickness)
112     if formula is None:
113         formula=1
114     R=OD/2.0
115     kp1=NinY*(NinX+1)+1 #Central point
116     kp2=NinX+1           #Bottom outside edge
117     mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=R, ly=h)
118     mat = Material('Material-1', elastic={'E':E, 'Nu':v})
119
120     V = FiniteElementModel(mesh=mesh, jobid='PlatePointPinned')
121     V.ElementBlock('ElementBlock1', ALL)
122     V.AssignProperties('ElementBlock1', eletyp, mat)
123
124     step = V.StaticStep()
125     step.FixNodes(IHI)
126     step = V.StaticStep()
127     step.Pressure(JHI, P)
128     step.run()
129     V.WriteResults()
130     if not os.environ.get('NOGRAPHICS'):
131         V.Plot2D(show=1, deformed=1)
132     return V
133
134 def Washer_Point_Pinned(E,v,P,OD,h,
135                         NinX=None, NinY=None, eletyp=None, inD=None, formula=None,
136                         **kwargs):
137     if eletyp is None:
138         eletyp = AxiSymmetricQuad4
139     if NinX is None:
140         NinX = 50 #Number of elements in I (Diameter)
141     if NinY is None:
142         NinY = 4 #Number elements in J (Thickness)
143     if inD is None:
144         inD = OD/2.0
145     if formula is None:
146         formula=1
147     R = OD/2.0

```

```

148 Ri = inD/2.0
149 w = R-Ri
150 kp1=NinY*(NinX+1)+1 #Central point
151 kp2=NinX+1           #Bottom outside edge
152 mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=w, ly=h, shift=[Ri,0])
153 mat = Material('Material-1', elastic={'E':E, 'Nu':v})
154
155 V = FiniteElementModel(mesh=mesh, jobid='WasherPointPinned')
156 V.ElementBlock('ElementBlock1', ALL)
157 V.AssignProperties('ElementBlock1', eletyp, mat)
158
159 #V.PrescribedBC(kp2,Y)
160 step = V.StaticStep()
161 step.PinNodes(kp2)
162 #step.FixNodes(kp2)
163 #step.PrescribedBC(kp2,Y)
164 step = V.StaticStep()
165 step.ConcentratedLoad(kp1, Y, -P)
166 step.run()
167 V.WriteResults()
168 if not os.environ.get('NOGRAPHICS'):
169     V.Plot2D(show=1, deformed=1)
170 return V
171
172 def Washer_Point_Clamped(E,v,P,OD,h,
173                           NinX=None, NinY=None, eletyp=None, inD=None, formula=None,
174                           **kwargs):
175     if eletyp is None:
176         eletyp = AxiSymmetricQuad4
177     if NinX is None:
178         NinX = 50 #Number of elements in I (Diameter)
179     if NinY is None:
180         NinY = 4 #Number elements in J (Thickness)
181     if inD is None:
182         inD = OD/2.0
183     if formula is None:
184         formula=1
185     R = OD/2.0
186     Ri = inD/2.0
187     w = R-Ri
188     kp1=NinY*(NinX+1)+1 #Central point
189     kp2=NinX+1           #Bottom outside edge
190     mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=w, ly=h, shift=[Ri,0])
191     mat = Material('Material-1', elastic={'E':E, 'Nu':v})
192
193     V = FiniteElementModel(mesh=mesh, jobid='WasherPointClamped')
194     V.ElementBlock('ElementBlock1', ALL)
195     V.AssignProperties('ElementBlock1', eletyp, mat)
196
197     step = V.StaticStep()
198     step.FixNodes(IHI)
199     step = V.StaticStep()
200     step.ConcentratedLoad(kp1, Y, -P)
201     step.run()
202     V.WriteResults()
203     if not os.environ.get('NOGRAPHICS'):
204         V.Plot2D(show=1, deformed=1)
205     return V
206
207 def Washer_Pressure_Pinned(E,v,P,OD,h,
208                           NinX=None, NinY=None, eletyp=None, inD=None, formula=None,
209                           **kwargs):
210     if eletyp is None:
211         eletyp = AxiSymmetricQuad4
212     if NinX is None:
213         NinX = 50 #Number of elements in I (Diameter)
214     if NinY is None:
215         NinY = 4 #Number elements in J (Thickness)

```

```

216     if inD is None:
217         inD = OD/2.0
218     if formula is None:
219         formula=1
220     R = OD/2.0
221     Ri = inD/2.0
222     w = R-Ri
223     kp1=NinY*(NinX+1)+1 #Central point
224     kp2=NinX+1           #Bottom outside edge
225     mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=w, ly=h, shift=[Ri, 0])
226     mat = Material('Material-1', elastic={'E':E, 'Nu':v})
227
228     V = FiniteElementModel(mesh=mesh, jobid='WasherPressurePinned')
229     V.ElementBlock('ElementBlock1', ALL)
230     V.AssignProperties('ElementBlock1', eletyp, mat)
231
232     #V.PrescribedBC(kp2,Y)
233     step = V.StaticStep()
234     step.PinNodes(kp2)
235     #step.FixNodes(kp2)
236     #step.PrescribedBC(kp2,Y)
237     step = V.StaticStep()
238     step.Pressure(JHI, P)
239     step.run()
240     V.WriteResults()
241     if not os.environ.get('NOGRAPHICS'):
242         V.Plot2D(show=1, deformed=1)
243     return V
244
245 def Washer_Pressure_Clamped(E,v,P,OD,h,
246                               NinX=None, NinY=None, eletyp=None, inD=None, formula=None,
247                               **kwargs):
248     if eletyp is None:
249         eletyp = AxiSymmetricQuad4
250     if NinX is None:
251         NinX = 50 #Number of elements in I (Diameter)
252     if NinY is None:
253         NinY = 4 #Number elements in J (Thickness)
254     if inD is None:
255         inD = OD/2.0
256     if formula is None:
257         formula=1
258     R = OD/2.0
259     Ri = inD/2.0
260     w = R-Ri
261     kp1=NinY*(NinX+1)+1 #Central point
262     kp2=NinX+1           #Bottom outside edge
263     mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=w, ly=h, shift=[Ri, 0])
264     mat = Material('Material-1', elastic={'E':E, 'Nu':v})
265
266     V = FiniteElementModel(mesh=mesh, jobid='WasherPressurePinned')
267     V.ElementBlock('ElementBlock1', ALL)
268     V.AssignProperties('ElementBlock1', eletyp, mat)
269
270     step = V.StaticStep()
271     step.FixNodes(IHI)
272     step = V.StaticStep()
273     step.Pressure(JHI, P)
274     step.run()
275     V.WriteResults()
276     if not os.environ.get('NOGRAPHICS'):
277         V.Plot2D(show=1, deformed=1)
278     return V
279
280 def Universal_Plate(E,v,P,OD,h,Plate=None,Pin=None,Point=None,
281                      NinX=None, NinY=None, eletyp=None, inD=None,
282                      formula=None, job=None, **kwargs):
283     if eletyp is None:

```

```

284     eletyp = AxiSymmetricQuad4
285     if NinX is None:
286         NinX = 50 #Number of elements in I (Diameter)
287     if NinY is None:
288         NinY = 4 #Number elements in J (Thickness)
289     if inD is None:
290         inD = OD/2.0
291     if formula is None:
292         formula=1
293     if Plate is None:
294         Plate = True
295     if Pin is None:
296         Pin = True
297     if Point is None:
298         Point = True
299     if job is None:
300         job = 'AxiSymPlate'
301 R = OD/2.0
302 Ri = inD/2.0
303 w = R-Ri
304 kp1=NinY*(NinX+1)+1 #Central point
305 kp2=NinX+1           #Bottom outside edge
306 mat = Material('Material-1', elastic={'E':E, 'Nu':v})
307
#Case Structure for Plate vs Washer
308 if Plate:
309     #If Plate
310     mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=R, ly=h)
311 else:
312     #If Washer
313     mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=w, ly=h, shift=[Ri,0])
315
V = FiniteElementModel(mesh=mesh, jobid=job)
317 V.ElementBlock('ElementBlock1', ALL)
318 V.AssignProperties('ElementBlock1', eletyp, mat)
319 step = V.StaticStep()
# Case Structure for Pined vs Fixed Outside Diameter
320 if Pin:
321     step.PrescribedBC(kp2, Y)
322 else:
324     step.FixNodes(IHI)
325
step = V.StaticStep()
# Case Structure for Pressure vs Point loading
327 if Point:
328     step.ConcentratedLoad(kp1, Y, -P)
329 else:
330     step.Pressure(JHI, P)
332
step.run()
334 #V.WriteResults()
335 if not os.environ.get('NOGRAPHICS'):
336     V.Plot2D(show=1, deformed=1)
337 return V
338
339
340 def Thick_Infinite_Cyl(E,v,P,OD,h,
341                         NinX=None, NinY=None, eletyp=None, inD=None, formula=None,
342                         **kwargs):
343     if eletyp is None:
344         eletyp = AxiSymmetricQuad4
345     if NinX is None:
346         NinX = 60 #Number of elements in I (Diameter)
347     if NinY is None:
348         NinY = 10 #Number elements in J (Thickness)
349     if inD is None:
350         inD = OD/2.0
351     if formula is None:

```

```

352     formula=1
353     R = OD/2.0
354     Ri = inD/2.0
355     w = R-Ri
356     kp1=NinY*(NinX+1)+1 #Central point
357     kp2=NinX+1           #Bottom outside edge
358     mesh = RectilinearMesh2D(nx=NinX, ny=NinY, lx=w, ly=h, shift=[Ri,0])
359     mat = Material('Material-1', elastic={'E':E, 'Nu':v})
360
361     V = FiniteElementModel(mesh=mesh, jobid='InfiniteCylinder')
362     V.ElementBlock('ElementBlock1', ALL)
363     V.AssignProperties('ElementBlock1', eletyp, mat)
364
365
366     V.PrescribedBC(JLO,Y)
367     V.PrescribedBC(JHI,Y)
368
369 #step = V.StaticStep()
370 #step.FixNodes(IHI)
371 step = V.StaticStep()
372 step.Pressure(ILO, P)
373 step.run()
374 V.WriteResults()
375 if not os.environ.get('NOGRAPHICS'):
376     V.Plot2D(show=1, deformed=1)
377 return V

```

Chapter 6

Unit Testing

In general, due to non-convergence poor fitting with the analytical solutions that we were unable to resolve, unit testing was not completed. However, many of the commented assertion statements could be uncommented if the analytical solutions and the numerical solutions were in the same ballpark. If run as is, all tests would fail.

Many more tests were planned, including tests of various poisson ratio, young's modulus, element type etc for each of the 8 loading conditions studied in this paper. The primary framework exists, however, is ultimately unsuccessful at this point.

```
1 import os
2 import numpy as np
3 import subprocess
4 import sys
5 import pdb
6 sys.path.insert(0, '../')
7 from pyfem2 import *
8 from Definitions import *
9 from Definitions2 import *
10
11 #This code utilizes the testing framework within python to run verification tests on the FEM
12 # solver that we have developed.
13 def test_1():
14     # TEST CASE 1
15     # Geometry: Flat circular plate with no holes
16     # Supports: Simply supported at radius r
17     # Loads: Uniform pressure load across entire plate
18     # ELEMENT TYPE: AxiSymmetricQuad4
19     #
20     # Objective of test:
21     # Verify that element in development produces appropriate maximum
22     # deflection at the center of the plate.
23     # See schematic in documentation for more information.
24
25 ##### Problem Setup #####
26 problem = dict({ 'E':1e6,
27                  'v':0.3,
28                  'P': 10,
29                  'OD':23,
30                  'h' : .4,
31                  'eletyp':AxiSymmetricQuad4,
32                  'formula':1 })
33
34 ##### FEM #####
35 V=Plate_Pressure_Pinned(**problem)
36 zFEM=get_max_disp(V)
37 ##### Exact #####
38 zANA=A_Plate_Pressure_Pinned(**problem)
```

```

40     print(zFEM)
41     print(zANA)
42     err=(zFEM-zANA)/zANA*100.
43     print(err)
44     #Compare the solutions
45     #Assert a tolerable error to pass/fail test
46     #assert np.allclose(zFEM,zANA, atol=1e-5)
47
48 def test_2():
49     # TEST CASE 2
50     # Geometry: Flat circular plate with no holes
51     # Supports: Cantilever or clamped edges at radius r
52     # Loads: Uniform pressure load across entire plate
53     #
54     # Objective of test:
55     # Verify that element in development produces appropriate maximum deflection at the
56     # center of the plate.
57     # See schematic in documentation (XXXX) for more information.
58
59     ##### Problem Setup #####
60     problem = dict({'E':1e6,
61                     'v':0.3,
62                     'P': 10,
63                     'OD':23,
64                     'h' : .4,
65                     'eletyp ':AxiSymmetricQuad4,
66                     'formula ':1})
67
68     ##### FEM #####
69     V=Plate_Pressure_Clamped(**problem)
70     zFEM=get_max_disp(V)
71     ##### Exact #####
72     zANA=A_Plate_Pressure_Clamped(**problem)
73
74     print(zFEM)
75     print(zANA)
76
77     err=(zFEM-zANA)/zANA*100.
78     print(err)
79     #Compare the solutions
80     #Assert a tolerable error to pass/fail test
81     #assert np.allclose(zFEM,zmax, atol=1e-10)
82
83 def test_3():
84     # TEST CASE 3: Plate Point Pinned
85     # Geometry: Flat circular plate with no holes
86     # Supports: Simply supported at Ds
87     # Loads: Force per circumference applied at Dl
88     #
89     # Objective of test:
90     # Verify that element in development produces appropriate maximum deflection at the
91     # center of the plate.
92     # See schematic in documentation (XXXX) for more information.
93
94     ##### Problem Setup #####
95     problem = dict({'E':1e6,
96                     'v':0.3,
97                     'P': 100,
98                     'OD':23,
99                     'h' : .4,
100                    'eletyp ':AxiSymmetricQuad4,
101                    'formula ':1})
102
103     ##### FEM #####
104     V = Plate_Point_Pinned(**problem)
105     zFEM = get_max_disp(V)
106
107     ##### Exact #####

```

```

106 zANA = -A_Plate_Point_Pinned(**problem)
107
108 print(zFEM)
109 print(zANA)
110 err=(zFEM-zANA)/zANA*100.
111 print(err)
112 #Compare the solutions
113 #Assert a tolerable error to pass/fail test
114 #assert np.allclose(zFEM,zmax,atol=1e-10)
115
116 def test_4():
117     # TEST CASE 4 *****
118     # Geometry: Flat circular plate with no holes
119     # Supports: Cantilever or clamped edges at radius r
120     # Loads: Point load at center of the plate
121     #
122     # Objective of test:
123     # Verify that element in development produces appropriate maximum deflection at the
124     # center of the plate.
125     # See schematic in documentation (XXXX) for more information.
126
127     ##### Problem Setup #####
128     problem = dict({'E':1e6,
129                     'v':0.3,
130                     'P': 100,
131                     'OD':23,
132                     'h' : .4,
133                     'eletyp ':AxiSymmetricQuad4,
134                     'formula ':1})
135
136     ##### FEM #####
137     V = Plate_Point_Clamped(**problem)
138     zFEM = get_max_disp(V)
139
140     ##### Exact #####
141     zANA = A_Plate_Point_Clamped(**problem)
142
143     ##### Error #####
144     err=(zFEM-zANA)/zANA*100.
145     #assert np.allclose(zFEM,zANA,atol=2)
146
147 def test_5():
148     # TEST CASE 4: WITH HOLE
149     # Geometry: Flat circular plate with no holes
150     # Supports: Cantilever or clamped edges at radius r
151     # Loads: Point load at center of the plate
152     #
153     # Objective of test:
154     # Verify that element in development produces appropriate maximum deflection at the
155     # center of the plate.
156     # See schematic in documentation (XXXX) for more information.
157
158     ##### Problem Setup #####
159     problem = dict({'E':1e6,
160                     'v':0.3,
161                     'P': 100,
162                     'OD':23,
163                     'h' : .4,
164                     'eletyp ':AxiSymmetricQuad4,
165                     'formula ':1})
166
167     ##### FEM #####
168     V = Washer_Point_Pinned(**problem)
169     zFEM = get_max_disp(V)
170
171     ##### Exact #####
172     zANA = A_Washer_Point_Pinned(**problem)

```

```
172 ##### Error #####
173 err=(zFEM-zANA)/zANA*100.
174 #assert np.allclose(zFEM,zANA, atol=2)
```