AFEM: Axisymmetric Project

Derek Lontine, Stuart Childs, Alex Bailey

April 27, 2016

Chapter 1

Introduction

Chapter 2

Analytical Formulation

The finite element formulation

Fellipa

$$\boldsymbol{K}^e = \sum_{k=1}^{p}\sum_{l=1}^{p} w_k w_i \boldsymbol{B}^T \boldsymbol{E}\boldsymbol{B} r J_\Omega \tag{2.1}$$

$$\boldsymbol{f} = \int_{\Omega^e} r\boldsymbol{N}^T\boldsymbol{b}\, d\Omega + \int_{\Gamma^e} r\boldsymbol{N}^T\hat{\boldsymbol{t}}\, d\Gamma \tag{2.2}$$

Santos:

$$[\boldsymbol{K}^e] = \underbrace{\int_\Omega [\boldsymbol{B}]^T[\boldsymbol{D}][\boldsymbol{B}]dA}_{PlaneStrain} - \int_r \frac{1}{r}(\sigma_{33}\delta_{1i} - \sigma_{i1}) w_i dr \tag{2.3}$$

$$\{\boldsymbol{f}\} = \int_{A^e} \rho\{\boldsymbol{b}\}[\boldsymbol{P}]dA + \int_S \{\boldsymbol{t}\}[\boldsymbol{P}]ds \tag{2.4}$$

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 $[\boldsymbol{B}]$ 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.

```
from numpy import *
from .isop2_4 import CSDIsoParametricQuad4 as BaseElement
# --------------------------------------------------------------------------- #
# ------------------------- Axisymmetric Quad Element ----------------------- #
# --------------------------------------------------------------------------- #
class AxiSymmetricQuad4(BaseElement):
    ndir = 3
    nshr = 1
    integration = 4
    elefab = {'formulation': 1}
    gaussw = ones(4)
    gaussp = array([[-1., -1.], [ 1., -1.], [-1.,  1.], [ 1.,  1.]]) / sqrt(3.)
    @property
    def formulation(self):
        return self.axisymmetric
    @formulation.setter
    def formulation(self, arg):
        assert arg in (0, 1, 2)
        self.axisymmetric = arg
    def bmatrix(self, dN, N, xi, *args):
        rp = dot(N, self.xc[:,0])
        B = zeros((4, 8))
        B[0, 0::2] = B[3, 1::2] = dN[0, :]
        B[1, 1::2] = B[3, 0::2] = dN[1, :]
        B[2, 0::2] = N / rp
        return B
```

3.2 Reduced Integration with Hourglass Control

```
from numpy import *
from .isop2_4 import CSDIsoParametricQuad4 as BaseElement
# --------------------------------------------------------------------------- #
# --------- Axisymmetric Reduced Integration With Hourglass Element --------- #
# --------------------------------------------------------------------------- #
class AxiSymmetricQuad4Reduced(BaseElement):
    ndir = 3
    nshr = 1
    integration = 1
    elefab = {'formulation': 1}
    gaussw = array([4.])
    gaussp = array([[0.,0.]])
    hourglass_control = True
    #HOURGLASS CONTROL PARAMETERS
    hglassp = array([[0.,0.,]])
    hglassv = array([[1.,-1.,1.,-1.]])
    #REST
    @property
    def formulation(self):
        return self.axisymmetric
    @formulation.setter
    def formulation(self, arg):
        assert arg in (0, 1, 2)
        self.axisymmetric = arg
    def bmatrix(self, dN, N, xi, *args):
        rp = dot(N, self.xc[:,0])
        B = zeros((4, 8))
        B[0, 0::2] = B[3, 1::2] = dN[0, :]
        B[1, 1::2] = B[3, 0::2] = dN[1, :]
        B[2, 0::2] = N / rp
        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 .isoplib 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
```

3.4 Aba Meshing

Chapter 4

Verification Problems

Chapter 5

Performance Evaluation

```
xforce=...
c = J * self.gaussw[p]
if self.axisymmetric == 1:
  rp = dot(Ne, self.xc[:,0])
  c *= rp
xforce += c * dot(Pe.T, dloadx)
```