Investigation 004
Applied Element Method

A.H. Kotze

M.Eng student, University of Pretoria, South Africa
Pyrometalurgical Modelling Group
B.Eng(Hons) Metallurgical, B.Eng Mech

July 29, 2016

Contents

1 Introduction **2**

2 Math **3**
2.1 Static Monotonic Small Deformations of Elastic Material 4
2.1.1 Poisson Effect 8

3 Stresses and Strains **11**

4 Python Coding **12**
4.1 AEM_basic 14
References19

List of Figures

1 Modelling of structure to AEM [Meguro K. 2000] 3
2 Breakdown of Papers published with modelling methods using AEM [Meguro K. 2000] 3
3 Degrees of Freedom and Element shape with Contact location shown [Meguro K. 2000] 4
4 Two elements with normal and shear spring connection [Prashidha 2014] 5
5 Effects of number of springs on rotational stiffness [Meguro K. 2000] 6
6 Flow chart of an elastic loading condition AEM program [Meguro K. 2000] 7
7 Element and element edge numbering used in Poisson effect inclusion [Meguro K. 2000] 8
8 Horizontal displacement of element 0 with Poisson effect taken into account [Meguro K. 2000] 9
9 Values to add to stiffness matrix for unit displacement in all DOF for element one in Figure 8 [Meguro K. 2000] 10

Listings

1 Python AEM basic program 14

Document Purpose

The purpose of this document is to explain the Applied Element Method (AEM). It includes the first principals of the mathematics governing the method and can be used when coding a solver using the AEM. This document will also explain in more detail the mathematics of the method found in a series of papers starting with [Meguro K. 2000]. It combines the information from the papers with additional information obtained from other sources, easing the implementation of the method.

Target Audience

Herman Kotze To use this document when coding an AEM solver.

Johan Zietsman For verification of my work and act as an introduction to AEM

Schalk Kok For verification of my work and act as an introduction to AEM

1 Introduction

There are several cases where other modelling methods proved to be insufficient. These cases include modelling of masonry walls with independent bricks under different loading cases. These cases are usually high in contact areas and need to be able to handle crack initiation and growth. In my personal experience rigid body motion easily occurs when several objects are in contact with one another resulting in methods such as FEM failing. In the past attempts have been made to model these sort of cases with FEM and DEM however both proved to be computationally very expensive and requires special treatment of methods. Also specifically for DEM poor constitutive laws are used for brick interfaces [Bishnu Hari Pandey 2004].

Hereafter follows a breakdown of the mathematics used in AEM. I also include sections with possible coding implications and classes that might be used. This work is mostly based on the work and series of papers starting with [Meguro K. 2000]. I use other sources for easier application or additional information. In blue text throughout the document I include my thoughts on certain aspects of the method and how I plan to use it.

2 Math

The two types of springs used in AEM are normal springs and shear springs.

For the normal springs

$$K_n = \frac{EdT}{a} \tag{1}$$

and for the shear springs

$$K_s = \frac{GdT}{a} \tag{2}$$

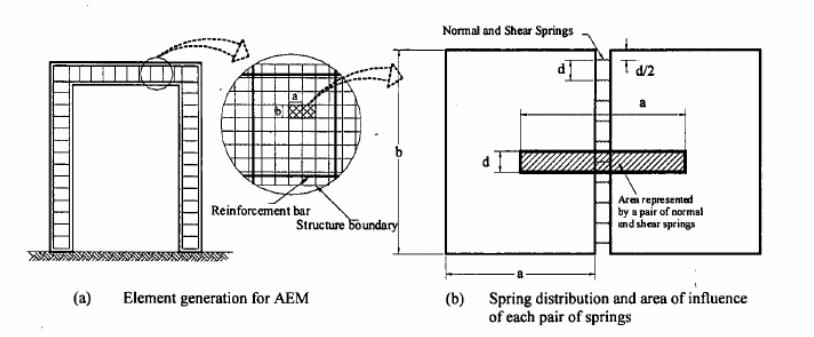

Figure 1: Modelling of structure to AEM [Meguro K. 2000]

As can be seen in the figure the Applied Element Method makes use of springs to model the interactions between small elements. The elements are assumed to be connected with pairs of normal and shear springs, these springs are located at contact locations which are distributed across the element edges. A pair of springs represent stresses and deformations of a certain area. in Figure 1 this area is the darker hatched area. Equations 1 and 2 show the spring stiffness for the pair of springs. a is the length of the area represented by the spring, d is both the distance between the springs and the height of the represented area. E and G are the Young's and Shear modulus of the material. Finally T is the thickness of the material. [Meguro K. 2000]

The next section of mathematical breakdown of the method is extracted from [Meguro K. 2000]. This paper was published as part of a series of papers explaining various ways to use the AEM method to model different situations. These paper lists and the modelling methods using AEM is shown in Figure 2 which is a table extracted from [Meguro K. 2000].

Table 2 Organization of research results

		Static		Dynamic	
Geometry	***Material***	***Monotonic***	***Cyclic***	***Monotonic***	***Cyclic***
Small deformation (linear)	***Elastic***	I(This paper)	III[12]	V[13]	VI[14]
	Nonlinear	II[10]			
Large deformation (nonlinear)	***Elastic***	IV[11]			
	Nonlinear	Covered in dynamics			
Collapse process		No meaning			

Figure 2: Breakdown of Papers published with modelling methods using AEM [Meguro K. 2000]

2.1 Static Monotonic Small Deformations of Elastic Material

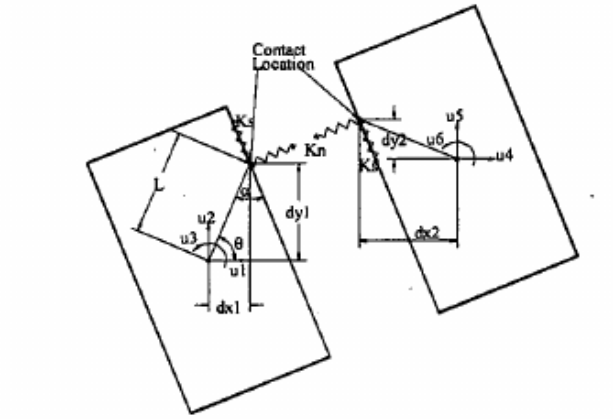

Figure 3: Degrees of Freedom and Element shape with Contact location shown [Meguro K. 2000]

With the assumption of three degrees of freedom for each element namely displacement in the x and y direction and rotation in the x-y plane. These degrees of freedom represent the rigid body motion of each element. What should be noted now is that each element moves as a rigid body, the deformations experienced by the element as well as the stresses are calculated according to the spring deformations around each element. Because each element already moves as a rigid body contact and contact separation is easy to model

Note how elements in AEM acts similarly to nodes in FEM with the springs acting similar to the elements in FEM. Seeing as in FEM elements connect nodes and in AEM springs connect the elements [Prashidha 2014]

For the simple case of explaining the maths the two elements shown in Figure 3 are assumed to be connected with one set of springs only. Note that on each element the dx and dy are specified. This si the x and y distances from the contact point to the centroid of the element. Each entry nt eh stiffness matrix corresponds to a degree of freedom, thus the stiffness matrix will be 6x6 in size.

The stiffness matrix components are computed by assuming a unit displacement in the studied direction and the computing the forces at the centroid of each element generated by the springs due to this displacement. I decided to follow the approach of Prashidha 2014 where the stiffness matrix is computed for the local co-ordinate system first and then translated to the global system using a translation matrix. The following equations are based on [Prashidha 2014] and Figure 4.

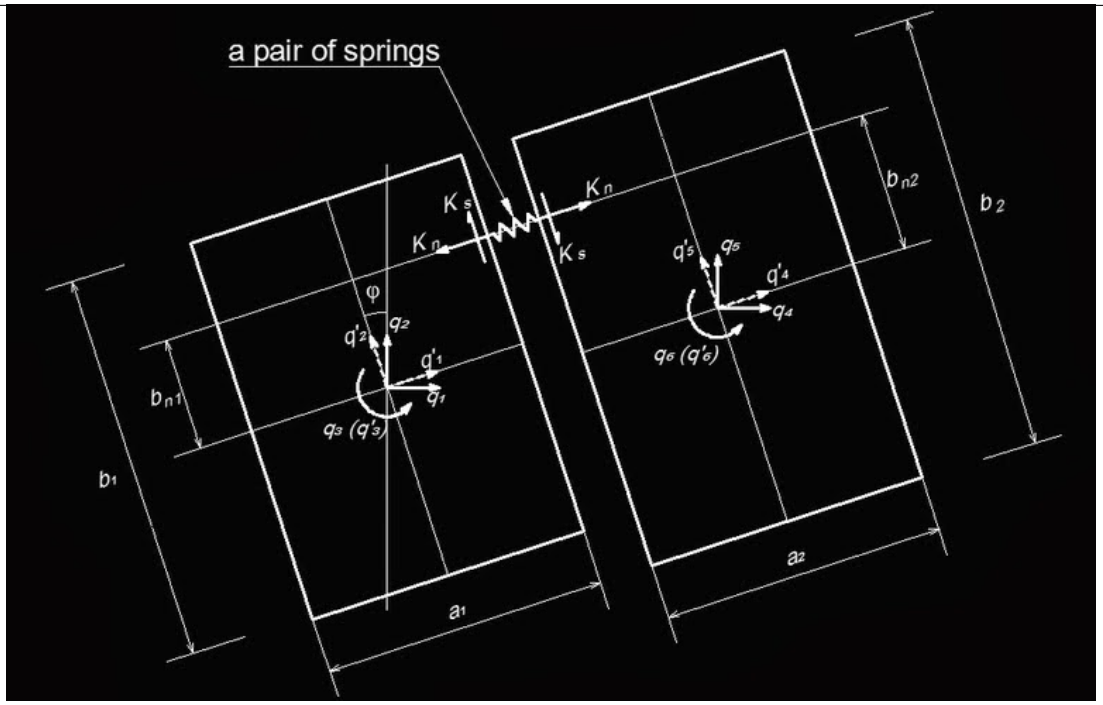

Figure 4: Two elements with normal and shear spring connection [Prashidha 2014]

The translation matrix are:

$$L = \begin{bmatrix} cos(\varphi) & sin(\varphi) & 0 & 0 & 0 & 0 \\ -sin(\varphi) & cos(\varphi) & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & cos(\alpha) & sin(\alpha) & 0 \\ 0 & 0 & 0 & -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

The stiffness matrix can then be calculated by the above mentioned method of a unit displacement in each degree of freedom and calculating the forces at the centroid of the elements. This is the stiffness matrix for the local co-ordinate system:

$$K^{te} = \begin{bmatrix} K_n & 0 & -K_n b_{n1} & -K_n & 0 & K_n b_{n2} \\ 0 & K_s & K_s \frac{a_1}{2} & 0 & -K_s & K_s \frac{a_2}{2} \\ -K_n b_{n1} & K_s \frac{a_1}{2} & K_n (b_{n1})^2 + K_s (\frac{a_1}{2})^2 & K_n b_{n1} & -K_s \frac{a_1}{2} & -K_n (b_{n1} b_{n2}) + K_s (\frac{a_1}{2} \frac{a_2}{2}) \\ -K_n & 0 & K_n b_{n1} & K_n & 0 & -K_n b_{n2} \\ 0 & -K_s & -K_s \frac{a_1}{2} & 0 & K_s & -K_s \frac{a_2}{2} \\ K_n b_{n2} & K_s \frac{a_2}{2} & -K_n (b_{n1} b_{n2}) + K_s (\frac{a_1}{2} \frac{a_2}{2}) & -K_n b_{n2} & -K_s \frac{a_2}{2} & K_n (b_{n2})^2 + K_s (\frac{a_2}{2})^2 \end{bmatrix} \tag{4}$$

Now the stiffness matrix can be calculated as

$$K^e = L^T K^{te} L \tag{5}$$

End of section extracted from [Prashidha 2014].

Note that this stiffness matrix computed above is only for one spring. To calculate the stiffness matrix for more springs the same procedure is followed and then the matrices are added together to form the stiffness matrix. Note that logic then dictates that the stiffness matrix size is not dependant on the amount of springs but rather the amount of elements. The final stiffness matrix is thus an average for the element according to the stress situation around the particular element. In this case pure rigid body motion of each element is considered and thus the stress is proportional to the displacement between the two ends of the springs. The strain is firstly calculated using:

$$\epsilon_x = \frac{d_x}{a} \tag{6}$$

$$\epsilon_y = \frac{d_y}{a} \tag{7}$$

For each spring either the x displacement or the y displacement can be calculated. Note that d_x and d_y are relative displacements of spring ends. The stresses is then simply calculated as

$$\sigma_x = E\epsilon_x \tag{8}$$

$$\sigma_y = E\epsilon_y \tag{9}$$

Note how for purely translational purposes a single spring would be adequate as the stiffness is calculated using the area represented by the spring and summing a bunch of springs together for a purely translational load leads to the same result as using one spring for the entire area. However using more springs leads to a greater accuracy for rotational loading to prove this the theoretical stiffness using only normal springs were calculated and then compared to the stiffness computed using a finite amount of normal springs. The process can easily be proved by writing out all the equations, I summarized the results below:

Referring to Figure 5 the following should be noted:

- The elements are square thus a as used in Equations 1 and 2 are the same as b in Figure 5
- The number of connecting springs are $2n$ thus the minimum number of springs are 2
- The rotational stiffness contribution of the normal springs are $K_r = K_n z^2$

Theoretically thus the stiffness for rotation are:

$$K_r = \int_{y=-\frac{b}{2}}^{y=\frac{b}{2}} \frac{ET}{b} z^2 dz \tag{10}$$

$$K_r = \frac{ETb^2}{12} \tag{11}$$

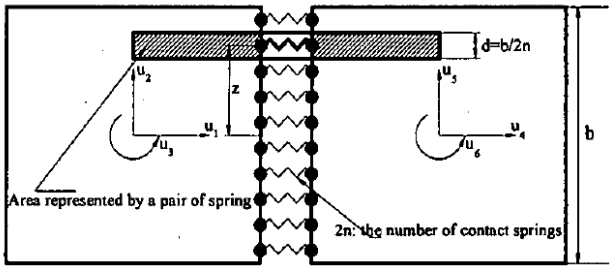

Figure 5: Effects of number of springs on rotational stiffness [Meguro K. 2000]

The stiffness calculated from $2n$ springs are then:

$$K_r = \frac{ETb^2}{4n^3} \Sigma_{i=1}^{n} (i - 0.5)^2 \tag{12}$$

The resulting comparison shows that more springs result in smaller errors when comparing equations 11 and 12. The same error difference was obtained in my own comparison as in the paper namely:

- 25% for 2 springs
- 2.8% for 6 springs
- 1.0% for 10 springs
- 0.3% for 20 springs

The final equation to solve, as with FEM, is:

$$K_G U = F \tag{13}$$

Here K_G is the global stiffness matrix, U the displacement vector and F the applied forces. The same method is used for both load and displacement control cases. For load cases the vector F is already known and we simply solve for U. "In displacement control cases, the load is applied by unit virtual displacement for one or more degrees of freedom." [Meguro K. 2000] I am unsure how exactly to implement this

Note that the matrix in Equation 4 is symmetrical. Thus after computing the global stiffness matrix a vector can be created for half the matrix containing only the non zero elements, thus saving memory and computing time. The layout of the program is shown in the Figure 6. Note how the stiffness matrix is recomputed for each increment.

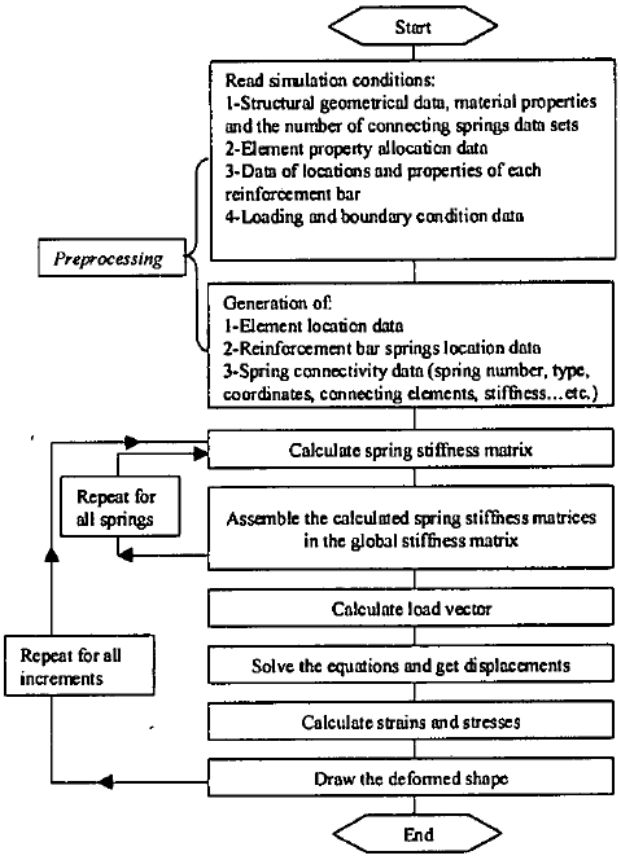

Figure 6: Flow chart of an elastic loading condition AEM program [Meguro K. 2000]

The number of elements and their influence was studied in [Meguro K. 2000] included was also the amount of springs used between elements. Final conclusions were that more elements with lower number of springs lead to more accuracy and less CPU time than high number of springs between

elements. To improve the accuracy in the case of elastic modelling it was advised to use higher number of elements with less springs for example 10 springs rather than 20. It was found that the same accuracy was obtained for 10 springs as for 20 springs but with half the CPU time. In conclusion it was found that larger elements are sufficient for cases where normal stresses are the most important but smaller and more elements are needed to accurately predict the shear stresses.

2.1.1 Poisson Effect

According to [Meguro K. 2000] there are two ways to model for the Poisson effect. I think these might be used to model the thermal expansion as well if adapted

First Method The first method introduces two additional degrees of freedom to the 2D model. Thus each element now has two additional degrees of freedom and results in a total of 5 degrees of freedom namely, x and y displacement, rotational and the expansion in the y and x direction. This however increases CPU time by approximately 2.78. The other additional problem is that DOF's now have a coupling effect making it harder to calculate the stresses and strains in the elements.

Second Method The second method involves still using only three degrees of freedom, but takes advantage of the assembly of the elements which is deformable. The stiffness matrix of each element is correlated by those of adjacent elements. This is the preferred method according to [Meguro K. 2000] because it avoids the problems mentioned for the previous method. Both should be investigated as possible solutions for the temperature expansion though

Firstly a factor needs to be introduced which will govern whether forces due to Poisson ratio expansion should be included in the stiffness matrix or not. This factor is calculated according to numbering of elements and of element edges. This is illustrated in Figure 7.

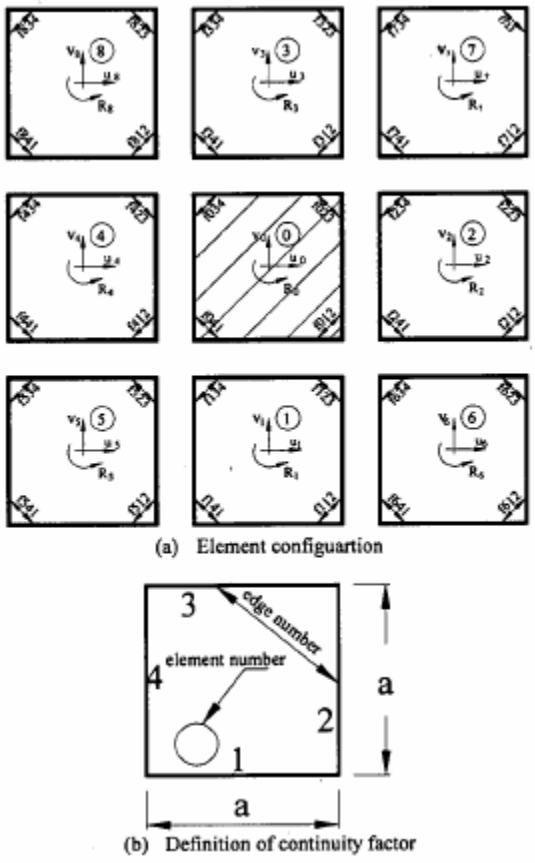

Figure 7: Element and element edge numbering used in Poisson effect inclusion [Meguro K. 2000]

The factor is then calculated using equation 14. In this case the element is divided into four sections with i indicating element number and j and k the edge numbers. If there is no element connected (springs need to be active) to element i at edge j then $fij = 0$. If there is a connecting element then $fij = 1$.

$$fijk = fij \times fik \tag{14}$$

Remember that for each spring a stiffness matrix is developed and then all these are summed together to form the stiffness matrix for the element. Now additional terms need to be added, these are calculated by assuming a corresponding displacement in the direction of the studied DOF and then calculating the response at the centre of the element. Thus these terms are added to the already assembled stiffness matrix of each element.

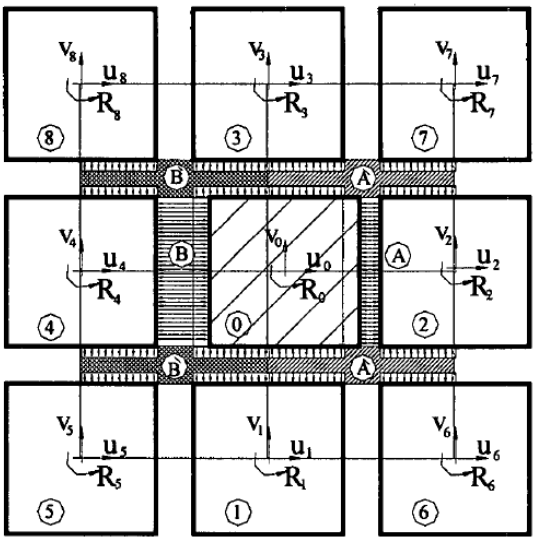

Figure 8: Horizontal displacement of element 0 with Poisson effect taken into account [Meguro K. 2000]

From Figure 8 it can be seen that due to horizontal displacement of element 0 region A is compressed and region B experiences tension. This means that a secondary stress is induced in areas A‘ and B‘ due to lateral displacement which is a function of the Poisson ratio. Note that if the lateral displacement is not prevented no additional stiffness matrix terms will have to be included. This is why the factor for continuity was introduced in equation 14.

In the case of element rotation, the secondary stresses induced are assumed to be evenly distributed across element faces. From [Meguro K. 2000] the additions to the global stiffness matrix, for unit displacements in all DOF for element one, are recorded in Figure 9. As this is a system with 9 elements each having 3 DOF the final matrix size will be 27×27. Those shown in Figure 9 are the first three rows.

Table 4 Added stiffness matrix values due to unit displacement of each degree of freedom of element (0)

(0)			(1)			(2)		
u0	**v0**	**R0**	**u1**	**v1**	**R1**	**u2**	**v2**	**R2**
0	p0 x (-f012+f023 -f034+f041)	m0 x (-f012+f023 +f034-f041)	0	p0 x (+f012-f041)	m0 x (+f012+f041)	0	p2 x (-f241+f234)	m2 x (+f241-f234)
p0 x (-f012+f023 -f034+f041)	0	m0 x (-f012-f023 +f034+f041)	p1 x (-f123+f134)	0	m1 x (+f123-f134)	p0 x (+f012-f023)	0	m0 x (+f012+f023)
m0 x (-f012+f023 +f034-f041)	m0 x (-f012-f023 +f034+f041)	-2 x m0 x a/4 (f012+f023 +f034+f041)	m1 x (-f123-f134)	m0 x (+f012-f041)	a/4 x (+m0 x f012 +m0 x f041 +m1 x f123 +m1 x f134)	m0 x (f012-f023)	m2 x (-f241-f234)	a/4 x (+m2 x f241 +m2 x f234 +m0 x f012 +m0 x f023)

(3)			(4)			(5)		
u3	**v3**	**R3**	**u4**	**v4**	**R4**	**u5**	**v5**	**R5**
0	p0 x (-f023+f034)	m0 x (-f023-f034)	0	p4 x (+f412-f423)	m4 x (+f412-f423)	0	-p4 x f412	-m4 x f412
p3 x (+f312-f341)	0	m3 x (+f312-f341)	p0 x (-f041+f034)	0	m0 x (-f041-f034)	-p1 x f134	0	m1 x f134
m3 x (+f312+f341)	m0 x (+f023-f034)	a/4 x (+m0 x f023 +m0 x f034 +m3 x f312 +m3 x f341)	m0 x (+f041-f034)	m4 x (+f412+f423)	a/4 x (+m4 x f412 +m4 x f423 +m0 x f041 +m0 x f034)	m1 x f134	m4 x -f412	a/4 x (-m4 x f412 -m1 x f134)

(6)			(7)			(8)		
u6	**v6**	**R6**	**u7**	**v7**	**R7**	**u8**	**v8**	**R8**
0	+p2 x f241	-m2 x f241	0	-p2 x f234	+m2 x f234	0	+p4 x f423	+m4 x f423
+p1 x f123	0	-m1 x f123	-p3f312	0	-m3 x f312	+p3f341	0	+m3 x f341
+m1 x f123	+m2 x f241	a/4 x (-m2 x f241 -m1 x f123)	-m3 x f312	+m2 x f234	a/4 x (-m2 x f234 -m3 x f312)	-m3 x f341	-m4 x f423	a/4 x (-m4 x f423 -m3 x f341)

Figure 9: Values to add to stiffness matrix for unit displacement in all DOF for element one in Figure 8 [Meguro K. 2000]

The force and moments imposed are

$$pi = \frac{\nu E_i t_i}{4(1-\nu^2)} \tag{15}$$

$$mi = \frac{\nu E_i t_i}{4(1-\nu^2)} \times \frac{a}{4} \tag{16}$$

ν is the Poisson ratio with E and t being the Young's Modulus and the element thickness respectively. i indicates the element number. Note the role he continuity factor plays in the assembly of the matrix, if a neighbouring element does not exist or if contact springs aren't in place due to cracking the effect is not taken into account. Also notice that only the forces and moments due to the normal springs are adjusted.

To calculate the stresses some alterations need to be made to equations 8 and 9. Firstly an average strain need to be calculated for each element namely ϵ_{xa} and ϵ_{ya} and then the new stress equations are:

$$\sigma_x = E \times \frac{\epsilon_x + \nu\epsilon_{ya}}{1-\nu^2} \tag{17}$$

$$\sigma_y = E \times \frac{\epsilon_y + \nu\epsilon_{xa}}{1-\nu^2} \tag{18}$$

Notice that when the Poisson ratio is ignored, $\nu = 0$ then we obtain equations 8 and 9 again.

3 Stresses and Strains

I had trouble with the implementation of the stresses and strains in the 2D code so the study recorded below was conducted. I first looked at the basics of stresses and strains and the ground principals of internal resultant loadings in a body. I then investigated the implementation of these stress and strain principals in continuum mechanics and in previous FEM codes. Finally I propose an implementation of the stresses and strains in the basic AEM implementation and do verifications of stresses and strains with analytical results and FEM obtained results.

Essentially the calculation of the stresses and strains are explained in a very limited way in the articles explaining the principals of AEM. Currently my understanding is that the normal springs can only provide stiffness in the normal direction from the element and the shear springs in the shear direction. It is thus my understanding that the displacements should be divided into the normal and shear for each spring pair and then used to calculate the strains and stresses.

4 Python Coding

As a first attempt to understand the implementation of the method, I hard coded a python program to solve two elements. I did not make use of any functions or classes. During this implementation several questions arose. Some of them can easily be answered and just serves as a reminder of factors that should be considered. Others I am unsure of the implications and how to solve the possible errors. I included the code for python program *AEM_ basic.py*. In order to make the impact of the problems and questions listed below easier to identify and understand. The numbers of the questions correlate with the number in the code where the problem might occur.

1. Impact on d and z values if element sizes are different, specifically in height.
2. Do different elements have different Kn and Ks values since it seems that d might change?
3. How exactly do I determine break strength for springs?
4. Do I just add the effects of different element pairs on the same DOF in the global stiffness matrix?
5. Should Kte be multiplied with L for each independent spring or can all Kte first be added together?

   - According to matrix properties:

   $$A(B + C) = AB + AC \tag{19}$$
   $$(A + B)C = AC + BC \tag{20}$$

   - It follows that:

   $$L^T K_1^{te} L + L^T K_2^{te} L = L^T (K_1^{te} L + K_2^{te} L) \tag{21}$$
   $$L^T (K_1^{te} L + K_2^{te} L) = L^T (K_1^{te} + K_2^{te}) L \tag{22}$$

   Thus all K^{te} matrices can be added together before the transformation matrix calculations are preformed.
6. What if angle between elements aren't the same any more, don't think will happen with small deformations but for larger?
7. The moment caused by normal springs are pretty much always zero when even number of springs are used?

Notes on Free and prescribed degrees of freedom: In order to include the effects of prescribed displacements I followed the same approach as explained by [Wilke 2015]. I divided the displacement and corresponding forces as well as the stiffness matrix into the free and prescribed displacement fields. In order to do this a system had to be in place in order to identify the prescribed degrees of freedom in the displacement vector.

Notes on element pair interaction and stiffness matrix calculations: The stiffness matrix is computed per element pair interaction rather than per element as is the case for FEM codes. I have opted to accomplish this by using two for loops to loop over the rows and columns of a grid. This code will need to be adjusted if the mesh does not take the form of a grid with hex elements. These loops can be seen in row numbers 80 to 150. First and foremost loops are placed over rows

and then columns, the interaction between two horizontal elements are then computed and added to the global stiffness matrix, then the interaction between elements vertically connected are computed. These are once again added to the global stiffness matrix. An interesting observation can be made here where it can be shown that the stiffness matrix vertically is the same as the horizontal stiffness matrix rotated through 90 degrees using $K^e = L^T K^{te} L$, if the following values are set to be the same $\{a_1 = b_1; a_2 = b_2; a_n1 = b_n1; a_n2 = b_n2\}$ by either using square elements or changing the values of the original variables. The proof is shown below:

$$K^{te}_{horizontal} = \begin{bmatrix} K_n & 0 & -K_n b_{bn1} & -K_n & 0 & K_n b_{bn2} \\ 0 & K_s & K_s \frac{a_1}{2} & 0 & -K_s & K_s \frac{a_2}{2} \\ -K_n b_{bn1} & K_s \frac{a_1}{2} & K_n (b_{n1})^2 + K_s (\frac{a_1}{2})^2 & K_n b_{n1} & -K_s \frac{a_1}{2} & -K_n (b_{n1} b_{n2}) + K_s (\frac{a_1}{2} \frac{a_2}{2}) \\ -K_n & 0 & K_n b_{bn1} & K_n & 0 & -K_n b_{n2} \\ 0 & -K_s & -K_s \frac{a_1}{2} & 0 & K_s & -K_s \frac{a_2}{2} \\ K_n b_{bn2} & K_s \frac{a_2}{2} & -K_n (b_{n1} b_{n2}) + K_s (\frac{a_1}{2} \frac{a_2}{2}) & -K_n b_{n2} & -K_s \frac{a_2}{2} & K_n (b_{n2})^2 + K_s (\frac{a_2}{2})^2 \end{bmatrix} \tag{23}$$

$$K^{te}_{vertical} = \begin{bmatrix} K_s & 0 & -K_s \frac{b_1}{2} & -K_s & 0 & -K_s \frac{b_2}{2} \\ 0 & K_n & -K_n a_{n1} & 0 & -K_n & K_n a_{n2} \\ -K_s \frac{b_1}{2} & -K_n a_{n1} & K_n (a_{n1})^2 + K_s (\frac{b_1}{2})^2 & K_s \frac{b_1}{2} & K_n a_{n1} & -K_n (a_{n1} a_{n2}) + K_s (\frac{b_1}{2} \frac{b_2}{2}) \\ -K_s & 0 & K_s \frac{b_1}{2} & K_s & 0 & K_s \frac{b_2}{2} \\ 0 & -K_n & K_n a_{n1} & 0 & K_n & -K_n a_{n2} \\ -K_s \frac{b_2}{2} & K_n a_{n2} & -K_n (a_{n1} a_{n2}) + K_s (\frac{b_1}{2} \frac{b_2}{2}) & K_s \frac{b_2}{2} & -K_n a_{n2} & K_n (a_{n2})^2 + K_s (\frac{b_2}{2})^2 \end{bmatrix} \tag{24}$$

$$L_{\varphi,\alpha=90^0} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{25}$$

$$L^T K^{te} L = \begin{bmatrix} K_s & 0 & -K_s \frac{a_1}{2} & -K_s & 0 & -K_s \frac{a_2}{2} \\ 0 & K_n & -K_n b_{n1} & 0 & -K_n & K_n b_{n2} \\ -K_s \frac{a_1}{2} & -K_n b_{n1} & K_n (b_{n1})^2 + K_s (\frac{a_1}{2})^2 & K_s \frac{a_1}{2} & K_n b_{n1} & -K_n (b_{n1} b_{n2}) + K_s (\frac{a_1}{2} \frac{a_2}{2}) \\ -K_s & 0 & K_s \frac{a_1}{2} & K_s & 0 & K_s \frac{a_2}{2} \\ 0 & -K_n & K_n b_{n1} & 0 & K_n & -K_n b_{n2} \\ -K_s \frac{a_2}{2} & K_n b_{n2} & -K_n (b_{n1} b_{n2}) + K_s (\frac{a_1}{2} \frac{a_2}{2}) & K_s \frac{a_2}{2} & -K_n b_{n2} & K_n (b_{n2})^2 + K_s (\frac{a_2}{2})^2 \end{bmatrix} \tag{26}$$

I however did not implement this just yet as I am unsure if it would yield faster CPU time as opposed to re-entering the matrix as I have currently done.

4.1 AEM_basic

Listing 1: Python AEM basic program

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 17 13:47:36 2016

@author: herman
"""

# Basic AEM for two elements
    # element numbering from 1,2...
    # Note  all comments marked as #& indicate changes to be made in AEM_2D.py
    #       all comments marked as #! are concerns to be addressed
    #       all comments marked as ## indicates start and end of sections
    #       all comments marked as #@ working comments

import numpy as np
from math import cos, sin

## Start of input deck      #@ change for beam input

# Material properties
E = 210e+9
nu = 0
G = E/(2*(1+nu))
T = 1

# Element properties
#& Should be adjusted to load from hex_mesher
grid = [5, 3, 0]
num_ele = grid[0]*grid[1]

a1 = 5          # half of width of element 1
b1 = 10         # height of element 1
theta1 = 0      # radians (for time being not applicable)

a2 = 5          # half of width of element 2
b2 = 10         # height of element 2
theta2 = 0      # radians (for time being not applicable)

gap = [0.001, 0.001, 0]         # size of gap between elements

# Spring properties
num_spring = 2

# Prescribed displacements
    # element number, DOF, value

# Prescribed forces
    # element number, DOF, value

## End of input deck

## Start of calculating stiffness matricies [Prashidha 2014]

#! Include [if] for element with smallest height
#! Question 1&2: Are Kn and Ks different if element sizes are different?
```

```
    # which element determines the value of d and z if element sizes differ?
d = float(b1)/(num_spring)
a = a1/2 + a2/2
Kn = (E*d*T)/a
Ks = (G*d*T)/a
#! need to adjust this to include theta1 and theta2 not really don't know
#@ input of element for loops
x_co = np.zeros(shape=(2, num_spring))
y_co = np.zeros(shape=(2, num_spring))
L0_row = a + gap[0]
L0_column = b1 + gap[1]     # only valid for elements of same height in grid

for co in range(0, num_spring):
    x_co[0, co] = a1*2
    x_co[1, co] = a1*2 + gap
    y_co[0, co] = d/2+d*(co)
    y_co[1, co] = d/2+d*(co)


#! Question 3: Setting up the stiffness matrices, these do not change unless a
                # spring is no longer active
# Note stiffness matrices are compiled per spring thus per element pair
## Start loops over element pairs
elemn = 0
for row in range(0, grid[1]-1):
    for column in range(0, grid[0]-1):
        ## Start of horizontal element pair interaction
        Kel = np.zeros(shape=(num_ele*3, num_ele*3))
        for sign in range(1, 3):
            for n in range(1, (num_spring/2)+1):
                bn1 = ((-1)**sign)*(d/2 + d*(n-1))
                bn2 = ((-1)**sign)*(d/2 + d*(n-1))
                Kte = np.array([[Kn, 0, -Kn*bn1, -Kn, 0, Kn*bn2],
                                [0, Ks, Ks*(a1/2), 0, -Ks, Ks*(a2/2)],
                                [-Kn*bn1, Ks*(a1/2), Kn*bn1**2+Ks*(a1/2)**2, Kn*bn1,
                                 -Ks*(a1/2), -Kn*bn1*bn2+Ks*(a1/2)*(a2/2)],
                                [-Kn, 0, Kn*bn1, Kn, 0, -Kn*bn2],
                                [0, -Ks, -Ks*(a1/2), 0, Ks, -Ks*(a2/2)],
                                [Kn*bn2, Ks*(a2/2), (Kn*bn1*bn2)+Ks*(a1/2)*(a2/2),
                                 -Kn*bn2, -Ks*(a2/2), Kn*bn2**2+Ks*(a2/2)**2]])
                Kel = Kel + Kte     # adding each spring's contribution

        # Translation matrix:
        #! Check use of theta2
        L = np.array([[cos(theta1), sin(theta1), 0, 0, 0, 0],
                      [-sin(theta1), cos(theta1), 0, 0, 0, 0],
                      [0, 0, 1, 0, 0, 0],
                      [0, 0, 0, cos(theta2), sin(theta2), 0],
                      [0, 0, 0, -sin(theta2), cos(theta2), 0],
                      [0, 0, 0, 0, 0, 1]])
        # Setting up stiffness matrix for element pair interaction
        LT = L.T
        Ke = LT.dot(Kel.dot(L))
        ## End of horizontal element pair interaction

        ## Start of adding element pair stiffness to global stiffness
        pg_r = ([3*elemn, 3*elemn+1, 3*elemn+2,
                 3*(elemn+1), 3*(elemn+1)+1, 3*(elemn+1)+2])
        Kg[ix_(pg_r, pg_r)] = Kg[ix_(pg_r, pg_r)] + Ke

```

```
116         ## Start of vertical element pair interaction
117         Kel = np.zeros(shape=(num_ele*3, num_ele*3))
118         for sign in range(1, 3):
119             for n in range(1, (num_spring/2)+1):
120                 bn1 = ((-1)**sign)*(d/2 + d*(n-1))
121                 bn2 = ((-1)**sign)*(d/2 + d*(n-1))
122                 Kte = np.array([[Kn, 0, -Kn*bn1, -Kn, 0, Kn*bn2],
123                                 [0, Ks, Ks*(a1/2), 0, -Ks, Ks*(a2/2)],
124                                 [-Kn*bn1, Ks*(a1/2), Kn*bn1**2+Ks*(a1/2)**2, Kn*bn1,
125                                  -Ks*(a1/2), -Kn*bn1*bn2+Ks*(a1/2)*(a2/2)],
126                                 [-Kn, 0, Kn*bn1, Kn, 0, -Kn*bn2],
127                                 [0, -Ks, -Ks*(a1/2), 0, Ks, -Ks*(a2/2)],
128                                 [Kn*bn2, Ks*(a2/2), (Kn*bn1*bn2)+Ks*(a1/2)*(a2/2),
129                                  -Kn*bn2, -Ks*(a2/2), Kn*bn2**2+Ks*(a2/2)**2]])
130                 Kel = Kel + Kte     # adding each spring's contribution
131
132         # Translation matrix:
133         #! Check use of theta2
134         L = np.array([[cos(theta1), sin(theta1), 0, 0, 0, 0],
135                       [-sin(theta1), cos(theta1), 0, 0, 0, 0],
136                       [0, 0, 1, 0, 0, 0],
137                       [0, 0, 0, cos(theta2), sin(theta2), 0],
138                       [0, 0, 0, -sin(theta2), cos(theta2), 0],
139                       [0, 0, 0, 0, 0, 1]])
140         # Setting up stiffness matrix for element pair interaction
141         LT = L.T
142         Ke = LT.dot(Kel.dot(L))
143         ## End of vertical element pair interaction
144
145         ## Start of adding element pair stiffness to global stiffness
146         pg_c = ([3*elemn, 3*elemn+1, 3*elemn+2,
147                  3*(elemn+grid[0]), 3*(elemn+grid[0])+1, 3*(elemn+grid[0])+2])
148         Kg[ix_(pg_c, pg_c)] = Kg[ix_(pg_c, pg_c)] + Ke
149
150
151 ## End of calculating stiffness matrices
152
153 ## Start of calculating unknown degrees of freedom and reaction forces
154
155 # Next the stiffness matrix will be devided so that the free and
156     # prescribed degree of freedoms can be idependantly used to calculate
157     # the unknown variables
158 # This section is copied and adjusted from
159             # quad4_ls_2013.m Prof S Kok and Dr D Wilke
160 # Calculating the load vector
161 # Loads are applied at the centre of an element on the degrees of freedom
162 # Displacements are arranged with the first three entries belonging
163     # to the first element and so forth
164     # resulting in the loading vector having the same arangement
165
166 F = np.zeros(shape=(num_ele*3, 1))    # need to add prescribed forces
167 F[3, 0] = 1.05e+12
168
169 # Solve displacements
170 U = np.zeros(shape=(num_ele*3, 1))
171 # U =np.linalg.solve(Kg,F)
172
173 # To solve for the free displacements the stiffness matrix needs to be devided
174     # this is done by clasifying the free and prescribed degrees of freedom
175 dof = np.ones(shape=(num_ele*3, 1))
```

```
Up = np.array([0, 0, 0])
dof[0, 0] = 0
dof[1, 0] = 0
dof[2, 0] = 0
#& Use a find function to find where dof is zero
pdof = np.array([0, 1, 2])
#& Use a find function to find where dof is not zero
fdof = np.array([3, 4, 5])

Kff = Ke[fdof][:, fdof]
Kfp = Ke[fdof][:, pdof]
Kpp = Ke[pdof][:, pdof]

# Now we include the effects of prescribed displacements
Fp = F[fdof, 0] - Kfp.dot(Up)
# Solve unknown degrees of freedom
Uf = np.linalg.solve(Kff, Fp)
# Finally solve for the support reactions
KfpT = Kfp.T
Fp = KfpT.dot(Uf) + Kpp.dot(Up)
# Placing calculated values back into global F and U
U[fdof, 0] = Uf
U[pdof, 0] = Up
F[pdof, 0] = Fp

## End of calculating unknown degrees of freedom and reaction forces

## Start of calculating stresses and strains
#& write function to compute co-ordinates of springs
x_co_n = np.zeros(shape=(2, num_spring))  # [ele,spring]
y_co_n = np.zeros(shape=(2, num_spring))  # [ele,spring]

#! Need to adapt this to handle elements not in global co-ordinates
for elem in range(0, grid[0]*grid[1]):
    dof1 = elem*3
    x_co_n[elem, :] = x_co[elem, :] + U[dof1, 0]
    y_co_n[elem, :] = y_co[elem, :] + U[dof1+1, 0]
    for ele_spr in range(0, num_spring):
        x_co_n[elem, ele_spr] = (x_co_n[elem, ele_spr]*cos(U[dof1+2, 0])
                                 - y_co_n[elem, ele_spr]*sin(U[dof1+2, 0]))
        y_co_n[elem, ele_spr] = (x_co_n[elem, ele_spr]*sin(U[dof1+2, 0])
                                 + y_co_n[elem, ele_spr]*cos(U[dof1+2, 0]))

ele = 0
Li_row = np.zeros(shape=(2, num_spring))
dL_row = np.zeros(shape=(2, num_spring))
strain_row = np.zeros(shape=(3, num_spring))

Li_column = np.zeros(shape=(2, num_spring))
dL_column = np.zeros(shape=(2, num_spring))
strain_column = np.zeros(shape=(3, num_spring))
strain_ele = np.zeros(shape=(3, grid[0]*grid[1]))
## Loop over elements
#@ Check flow of for loops note error only computing forward influence!!
for row in range(0, grid[1]-1):
    for column in range(0, grid[0]-1):
        Li_row[0, :] = x_co_n[ele+1, :] - x_co_n[ele, :]
        Li_row[1, :] = y_co_n[ele+1, :] - y_co_n[ele, :]
        dL_row[0, :] = Li_row[0, :] - L0_row
        dL_row[1, :] = Li_row[1, :] - L0_row
```

```
236         strain_row[0, :] = dL_row[0, :]/L0_row
237         strain_row[2, :] = dL_row[1, :]/L0_row
238         Li_column[0, :] = x_co_n[ele+grid[0], :] - x_co_n[ele, :]
239         Li_column[1, :] = y_co_n[ele+grid[0], :] - y_co_n[ele, :]
240         dL_column[0, :] = Li_column[0, :] - 0
241         dL_column[1, :] = Li_column[1, :] - L0_column
242         strain_column[1, :] = dL_column[1, :]/L0_column
243         strain_column[2, :] = dL_column[0, :]/L0_column
244 
245         strain_ele[0, ele] = ((np.average(strain_row[0, :])
246                                + np.average(strain_column[0, :]))/2)
247         strain_ele[1, ele] = ((np.average(strain_row[1, :])
248                                + np.average(strain_column[1, :]))/2)
249         strain_ele[2, ele] = ((np.average(strain_row[2, :])
250                                + np.average(strain_column[2, :]))/2)
251         ele += 1
252 
253     Li_column[0, :] = x_co_n[ele+grid[0], :] - x_co_n[ele, :]
254     Li_column[1, :] = y_co_n[ele+grid[0], :] - y_co_n[ele, :]
255     dL_column[0, :] = Li_column[0, :] - 0
256     dL_column[1, :] = Li_column[1, :] - L0_column
257     strain_column[1, :] = dL_column[1, :]/L0_column
258     strain_column[2, :] = dL_column[0, :]/L0_column
259     strain_ele[0, ele] = ((np.average(strain_row[0, :])
260                            + np.average(strain_column[0, :]))/2)
261     strain_ele[1, ele] = ((np.average(strain_row[1, :])
262                            + np.average(strain_column[1, :]))/2)
263     strain_ele[3, ele] = ((np.average(strain_row[3, :])
264                            + np.average(strain_column[3, :]))/2)
265     ele += 1
266 
267 #@ Redo calculation of stresses here to fit with what was doen above
268 stress_Pstrain = np.zeros(shape=(3, grid[0]*grid[1]))
269 stress_Pstress = np.zeros(shape=(3, grid[0]*grid[1]))
270 C_strain = np.array([[1-nu, nu, 0],
271                      [nu, 1-nu, 0],
272                      [0, 0, 1-2*nu]])
273 C_stress = np.array([[1, nu, 0],
274                      [nu, 1, 0],
275                      [0, 0, 1-nu]])
276 
277 stress_Pstrain = (E/((1+nu)*(1-2*nu))) * C_strain.dot(strain_ele)
278 stress_Pstress = (E/(1-nu**2)) * C_stress.dot(strain_ele)
279 ## End of calculating stresses and strains
280 
281 # Draw deformed shape
282 
283 print(U)
284 print(F)
285 print(strain_ele)
286 print(stress_Pstrain)
287 print(stress_Pstress)
```

*Refrences

Bishnu Hari Pandey, Kimiro Meguro (2004). "SIMULATION OF BRICK MASONRY WALL BEHAVIOR UNDER IN- PLANE LATERAL LOADING USING APPLIED ELEMENT METHOD". In: *13th World Conference on Earthquake Engineering*.

Meguro K., Tagel-Din H. (2000). "Applied Element Method for Structural Analysis: Theory and Application for Linear Materials". In: *Structural Eng. Earthquake Eng. JCSE* 17.

Prashidha, Kharel (2014). *Formulating the Applied Element Method: Linear 2D (Part I)*. URL: `http://prashidha.blogspot.co.za/2014/03/formulating-applied-element-method.html`.

Wilke, Daniel N. (2015). *Short overview of finite element method for linear elasticity*.