3D Space Truss Analysis Vector Example

February 23, 2017

1 3D Truss Analysis Example

This document demonstrates using Jupyter for vector operations by determining the axial forces in a simple 3D space truss.

1.1 Initialization

The code below imports the required modules into the notebook. We use NumPy for its arrays. The numpy.linalg gives us norm to determine a vectors magnitude. The set_printoptions sets the array default printing format. I use `X=0, Y=1, Z=2` as index values to return the components of a vector.

```
In [1]: from numpy import *
        from numpy.linalg import *
        set_printoptions(precision=3, suppress=True)
        X=0; Y=1; Z=2
```

1.2 Problem Statement

For the image shown to the right, determine all axial forces as well as all reaction forces. The applied force is $F = (0, 40kN, 0)$

We define all the ponits for use later in computing unit vectors. We also define a function to make this a trivial task.

```
In [2]: a=array((1.1, -0.4, 0))
        b=array((1,0,0))
        c=array((0,0,0.6))
        d=array((0,0,-0.4))
        e=array((0,0.8,0))

        Fapp=array((0, 40, 0))

        def λ(a,b):
            return (b-a)/norm(b-a)
```

1.3 Joint A

I should have an FBD of joint A here, but I will save that for another update.

In vector form, force equilibrium at joint A gives $T_{AB}\lambda_{AB} + T_{AC}\lambda_{AC} + T_{AD}\lambda_{AD} + F_{app} = 0$. We need to calculate the unit vectors.

```
In [3]: λab=λ(a,b)
        λac=λ(a,c)
        λad=λ(a,d)
        print('The unit vectors λab, λac, λad are:', '\n',λab, '\n',λac,'\n', λad)
```

```
The unit vectors λab, λac, λad are:
 [-0.243  0.97   0.   ]
 [-0.836  0.304  0.456]
 [-0.889  0.323 -0.323]
```

We can write the above equilibrium equation in matrx form as $M \cdot T = -F_{app}$.

In this case,

$$T = \begin{bmatrix} T_{AB} \\ T_{AC} \\ T_{AD} \end{bmatrix}$$

and M is composed of the unit vectors as defined in the cell below.

```
In [4]: m=matrix(((λab[X],λac[X],λad[X]),
                  (λab[Y],λac[Y],λad[Y]),
                  (λab[Z],λac[Z],λad[Z])))
        print(m)
```

```
[[-0.243 -0.836 -0.889]
 [ 0.97   0.304  0.323]
 [ 0.     0.456 -0.323]]
```

The solution is then:

$$\begin{bmatrix} T_{AB} \\ T_{AC} \\ T_{AD} \end{bmatrix} = M^{-1} \cdot (-F_{app})$$

which we calculate in the following cells.

```
In [5]: minv=inv(m)
        print(minv)
```

```
[[ 0.412  1.134 -0.   ]
 [-0.526 -0.132  1.315]
 [-0.742 -0.186 -1.237]]
```

```
In [6]: answer=minv.dot(-Fapp)
        (TAB, TAC, TAD)=answer.tolist()[0]
        print("The results are TAB={:.3f}, TAC={:.3f} and TAD={:.3f}.".format(TAB,
```

```
The results are TAB=-45.354, TAC=5.261 and TAD=7.422.
```

1.4 Joint B

We proceed in the exact same way at this joint. The only difference is that the known force comes from TAB, not an externally applied force.

```
In [9]: λba=λ(b,a)
        λbc=λ(b,c)
        λbd=λ(b,d)
        λbe=λ(b,e)
        print(λba, '\n',λbc,'\n', λbd,'\n', λbe)

[ 0.243 -0.97   0.   ]
 [-0.857  0.     0.514]
 [-0.928  0.    -0.371]
 [-0.781  0.625  0.   ]
```

```
In [10]: m=matrix(((λbc[X],λbd[X],λbe[X]),
                   (λbc[Y],λbd[Y],λbe[Y]),
                   (λbc[Z],λbd[Z],λbe[Z])))
         print(m)

[[-0.857 -0.928 -0.781]
 [ 0.     0.     0.625]
 [ 0.514 -0.371  0.   ]]
```

```
In [11]: minv=inv(m)
         print(minv)

[[-0.466 -0.583  1.166]
 [-0.646 -0.808 -1.077]
 [ 0.     1.601  0.   ]]
```

```
In [12]: FonBfromA=TAB * λba #note this is the unit vector from B to A, not A to B.
         print(FonBfromA)

[-11.  44.  -0.]
```

```
In [13]: answer=minv.dot(-FonBfromA)
         (TBC, TBD, TBE)=answer.tolist()[0]
         print("The results are TBC={:.3f}, TBD={:.3f} and TBE={:.3f}.".format(TBC,

The results are TBC=20.525, TBD=28.434 and TBE=-70.434.
```

1.5 Joint E

In this case, the equilibrium equation only has the known axial force from BE and the three reaction forces at E. This makes the equilibrium equation very simple.

```
In [15]: λeb=-λbe
         m=identity(3)
         print(m)
```

```
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

```
In [16]: minv=inv(m)
         minv
```

```
Out[16]: array([[ 1.,  0.,  0.],
                [ 0.,  1.,  0.],
                [ 0.,  0.,  1.]])
```

```
In [17]: FonEfromBE= TBE * λeb
         print(FonEfromBE)
```

```
[-55.  44.   0.]
```

```
In [19]: E=dot(minv,-FonEfromBE)
         print(E)
```

```
[ 55. -44.   0.]
```

1.6 Joint C

```
In [21]: λca=-λac
         λcb=-λbc
```

```
In [24]: FonCfromAC=TAC * λca
         FonCfromBC=TBC * λcb
         print('The force on C from member AC is {} \n''The force on C from member
         .format(FonCfromAC, FonCfromBC))
```

```
The force on C from member AC is [ 4.4 -1.6 -2.4]
The force on C from member BC is [ 17.6   -0.   -10.56]
```

```
In [26]: C=dot(minv,-(FonCfromAC+FonCfromBC))
         print(C)
```

```
[-22.     1.6   12.96]
```

1.7 Joint D

In [28]:
```
λda=-λad
λdb=-λbd
```

In [29]:
```
D=dot(minv,-(TAD * λda + TBD * λdb))
print(D)
```

```
[-33.      2.4   -12.96]
```

We can now check our work. The sum of the reaction forces should equal the applied force.

In [30]:
```
Freaction=C + D + E
print(Freaction)
```

```
[  0. -40.   0.]
```

This is the negative of the applied force.

In [32]:
```
print(Fapp)
```

```
[ 0 40  0]
```