Aplicación de los cuaterniones en el análisis matricial

Implementación de pyFEM

Cristian Ramirez Martín Estrada

Facultad de ingeniería
Universidad Nacional de Colombia

IX seminario permanente de divulgación de resultados de investigación
grupo INDETEC, Noviembre de 2020

Tabla de contenido

Introducción

Definición

pyFEM: programa de computador para el análisis de estructuras en tres dimensiones tipo pórtico sometidas a cargas estáticas.

```
pyFEM
├── pyFEM
│   ├── classtools.py
│   ├── core.py
│   └── primitives.py
├── test
│   ├── space_frame.py
│   └── trusses.py
├── LICENSE
└── README.md
```

github.com/rvcristiand/pyFEM

Introducción

Defincián

FEM.js: programa de computador para visualizar estructuras tridimensionales.

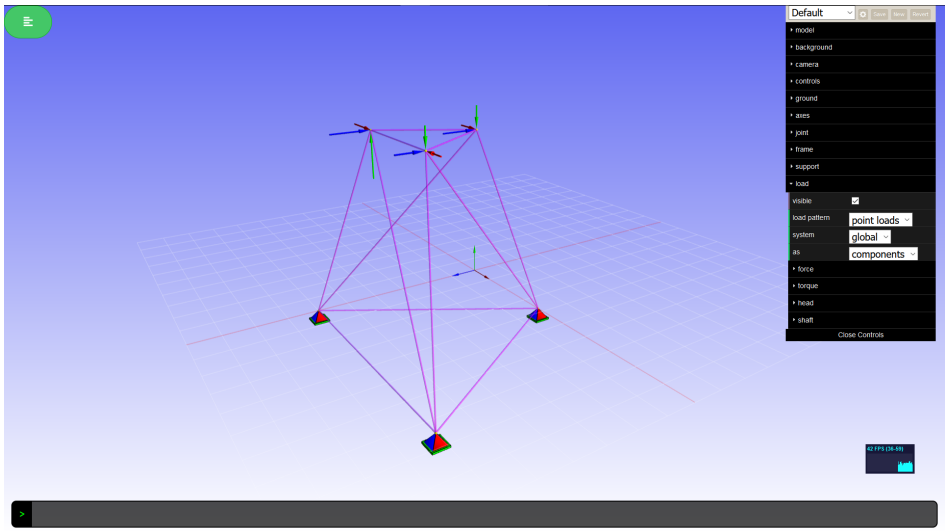


Figura: FEM.js ejecutandose en el navegador de internet.

github.com/rvcristiand/FEM.js
rvcristiand.github.io/FEM.js

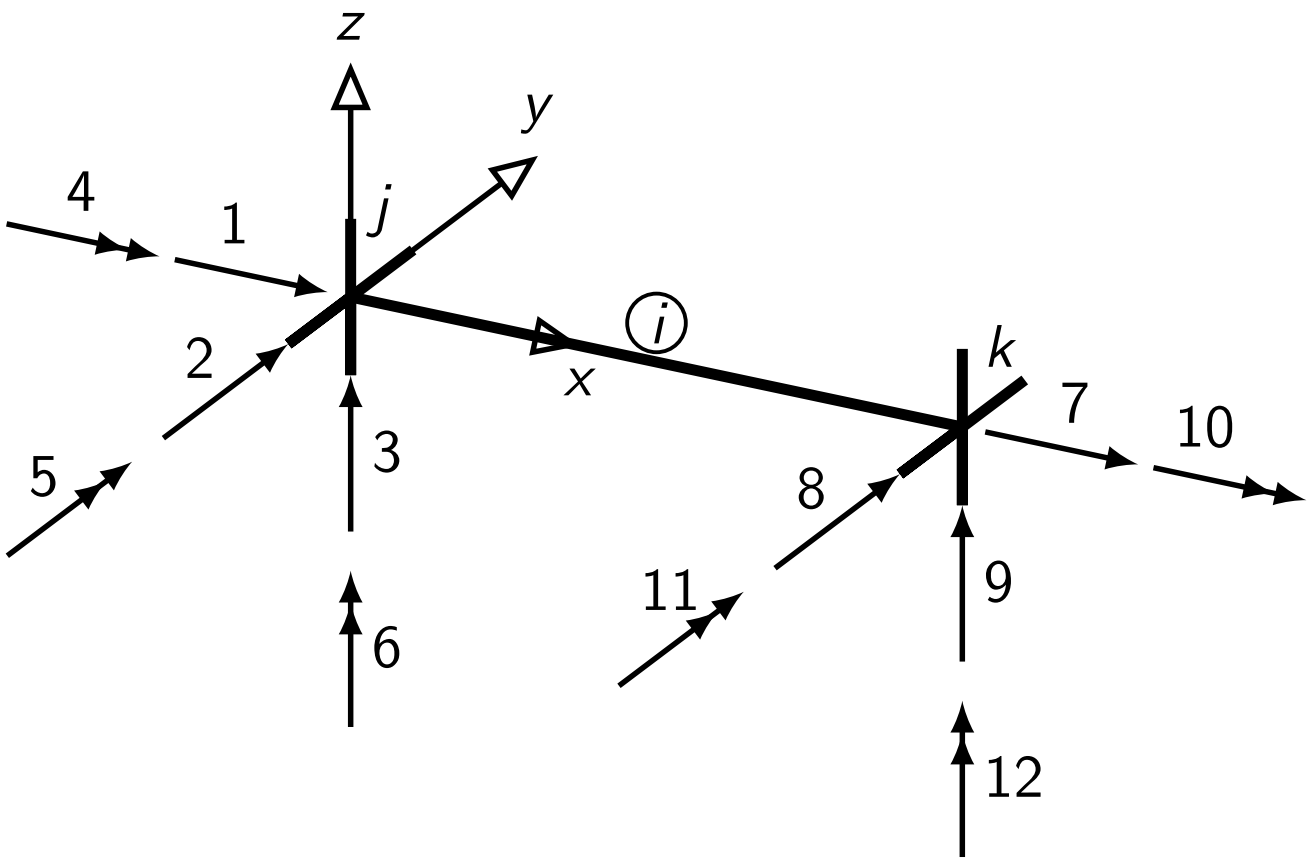
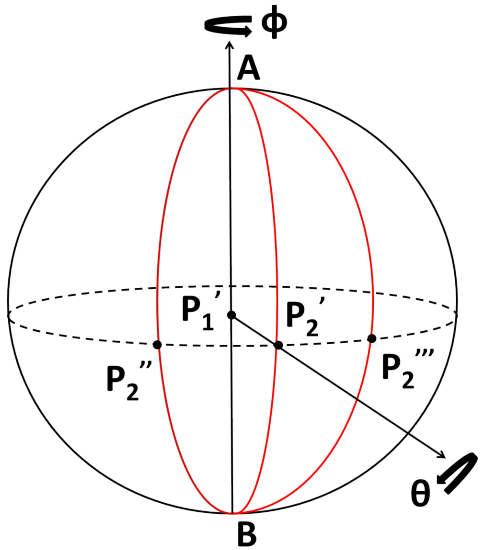


Figura: Elemento tipo pórtico en coordenadas locales.

$$
\begin{array}{c}
 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \\ 11 \\ 12
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12
\end{array} \\
\left[\begin{array}{cccccccccccc}
\frac{EA_X}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{EA_X}{L} & 0 & 0 & 0 & 0 & 0 \\
 & \frac{12EI_Z}{L^3} & 0 & 0 & 0 & \frac{6EI_Z}{L^2} & 0 & -\frac{12EI_Z}{L^3} & 0 & 0 & 0 & \frac{6EI_Z}{L^2} \\
 & & \frac{12EI_Y}{L^3} & 0 & -\frac{6EI_Y}{L^2} & 0 & 0 & 0 & -\frac{12EI_Y}{L^3} & 0 & -\frac{6EI_Y}{L^2} & 0 \\
 & & & \frac{GI_X}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GI_X}{L} & 0 & 0 \\
 & & & & \frac{4EI_Y}{L} & 0 & 0 & 0 & \frac{6EI_Y}{L^2} & 0 & \frac{2EI_Y}{L} & 0 \\
 & & & & & \frac{4EI_Z}{L} & 0 & -\frac{6EI_Z}{L^2} & 0 & 0 & 0 & \frac{2EI_Z}{L} \\
 & & & & & & \frac{EA_X}{L} & 0 & 0 & 0 & 0 & 0 \\
 & & & & & & & \frac{12EI_Z}{L^3} & 0 & 0 & 0 & -\frac{6EI_Z}{L^2} \\
 & & & & & & & & \frac{12EI_Y}{L^3} & 0 & \frac{6EI_Y}{L^2} & 0 \\
 & & & & & & & & & \frac{GI_X}{L} & 0 & 0 \\
 & & & & & & & & & & \frac{4EI_Y}{L} & 0 \\
\text{sim.} & & & & & & & & & & & \frac{4EI_Z}{L}
\end{array}\right]
\end{array}
\tag{1}
$$

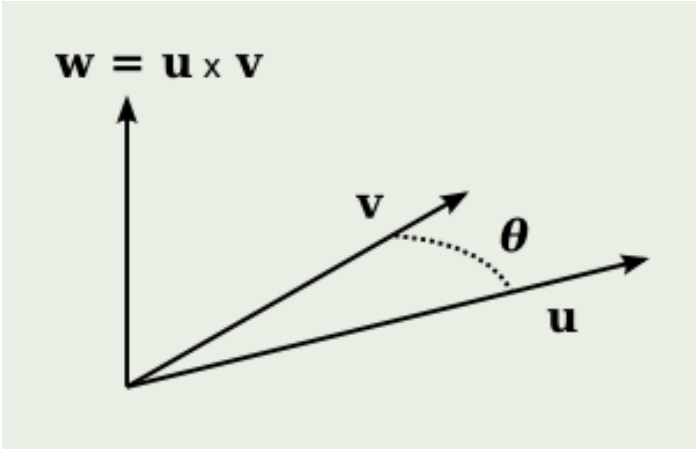
Definición

Según el *teorema de rotación de Euler* (véase Akademiia nauk SSSR., 1763), siempre es posible encontrar un diámetro de una esfera cuya posición es la misma después de rotarla alredor de su centro, por lo que cualquier secuencia de rotaciones de un sistema coordenado tridimensional es equivalente a una única rotación alrededor de un eje que pase por el origen.



El ángulo θ y el vector n que definen la rotación del eje x del sistema de coordenadas global hacia el eje x_m del sistema de coordenadas de un elemento se puede calcular como

$$\begin{aligned} \mathrm{n} &= (1,0,0) \times \mathrm{x_m} \\ \theta &= \operatorname{arc\,sen}((1,0,0) \cdot \mathrm{x_m}) \end{aligned} \tag{2}$$



Según Dunn, 2002, la rotación de un sistema de coordenadas tridimensionales alrededor del eje $\boldsymbol{n}$ una cantidad $\boldsymbol{\theta}$ se puede describir mediante un *cuaternión* como

$$\boldsymbol{q} = \begin{bmatrix} cos(\theta/2) & sin(\theta/2)\boldsymbol{n} \end{bmatrix} = \begin{bmatrix} w & x & y & z \end{bmatrix} \tag{3}$$

y se puede obtener la matriz de rotación a partir de un cuaternión de la siguiente manera

$$\mathrm{R} = \begin{bmatrix} 1-2y^2-2z^2 & 2xy+2wz & 2xz-2wy \\ 2xy-2wz & 1-2x^2-2z^2 & 2yz+2wx \\ 2xz+2wy & 2yz-2wx & 1-2x^2-2y^2 \end{bmatrix} \tag{4}$$

Implementación de los cuaterniones

```
class Frame(AttrDisplay, metaclass=
    UniqueInstances):
     __slots__ = ('joint_j', 'joint_k', 'material
         ', 'section')

     def __init__(self, joint_j=None, joint_k=
         None, material=None, section=None):
          self.joint_j = joint_j
          self.joint_k = joint_k
          self.material = material
          self.section = section
```

Implementación de los cuaterniones

```
class Frame(AttrDisplay, metaclass=
    UniqueInstances):
     def get_length(self):
         return distance.euclidean(self.joint_j.
             get_coordinate(), self.joint_k.
             get_coordinate())

     def get_direction_cosines(self):
         vector = self.joint_k.get_coordinate() -
              self.joint_j.get_coordinate()

         return vector / linalg.norm(vector)
```

Implementación de los cuaterniones

```
def get_rotation(self):
    v_from = np.array([1, 0, 0])
    v_to = self.get_direction_cosines()
    if np.all(v_from == v_to):
        return Rotation.from_quat([0, 0, 0, 1])
    elif np.all(v_from == -v_to):
        return Rotation.from_quat([0, 0, 1, 0])
    else:
        w = np.cross(v_from, v_to)
        w = w / linalg.norm(w)
        theta = np.arccos(np.dot(v_from, v_to))
        return Rotation.from_quat([x * np.sin(
            theta/2) for x in w] +
                [np.cos(theta/2)])
```

Conclusiones

Durante el ejercicio de la profesión como ingenieros civiles es habitual que usemos diferentes programas comerciales para el análisis y diseño de estructuras, con la imposibilidad de saber exactamente que rutinas siguen para llegar a los resultados que nos presentan.

La intención con este trabajo es dar el primer paso para crear una herramienta a *código abierto* para permitirle a los ingenieros depurar el programa para constatar que los cálculos realizados por el programa son los correctos.

Bibliografía

Akademiia nauk SSSR. (1763). *Novi comementarii Academiae scientiarum imperialis petropolitanae*. Typis Academiae Scientarum.

Dunn, F. (2002). *3D math primer for graphics and game development*. Plano, Tex, Wordware Pub.

Gracias