3° Práctica de Cálculo por Elementos Finitos - MC516

Josue Huaroto Villavicencio - 20174070I
Sección: E

14 de julio de 2020

1 Diagrama de flujo

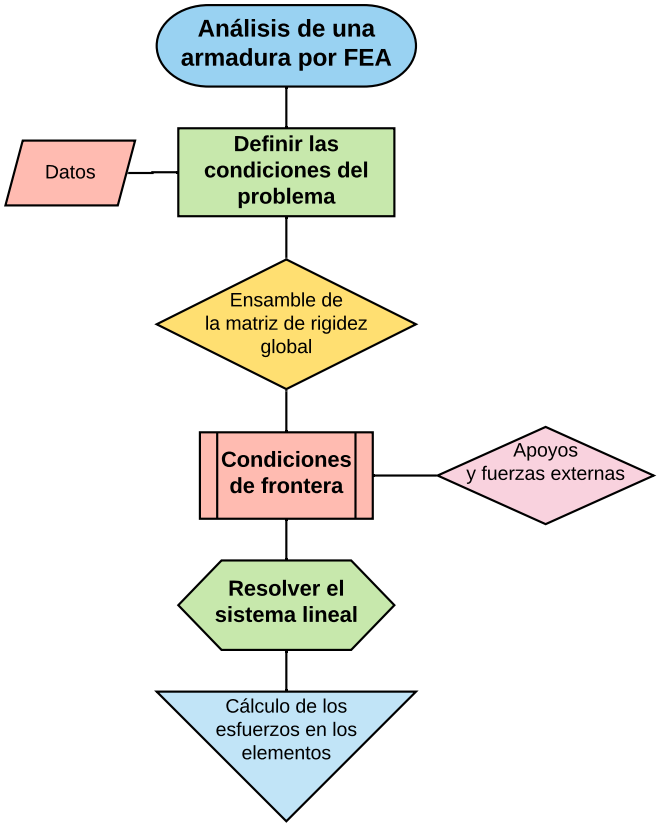

Figura 1: Diagrama de flujo

2 Ejecución del código

Del solver principal, se modifica algunas funciones para implementar ahora los elementos de la armadura.

Código 1: Cálculo de la longitud y el ángulo entre nodos

```
def DistNodes(f,s):
    if(s[0] == f[0]):
        aux = np.pi/2
        if(s[1] < f[1]):
```

```
            aux *= -1
        return (np.sqrt((s[0]-f[0])**2+(s[1]-f[1])**2),aux)
    else:
        aux = np.arctan((s[1]-f[1])/(s[0]-f[0]))
    if(aux < 0 and s[1] > f[1]):
        aux += np.pi
    if(s[1] < f[1]):
        aux += np.pi
        if(s[0] > f[0]):
            aux += np.pi

    return (np.sqrt((s[0]-f[0])**2+(s[1]-f[1])**2),aux)
```

Ahora, es necesario modificar la inserción de las matrices de rigidez de los elementos a la matriz de rigidez global.

Código 2: Ensamble de la matriz de rigidez

```
def AssemblyStiffness(nStiffnessMatrix,k,i,j):
    for p in range(0,2):
        for m in range(0,2):
            nStiffnessMatrix[2*i+p][2*i+m] += k[p][m]
            nStiffnessMatrix[2*i+p][2*j+m] += k[p][2+m]
            nStiffnessMatrix[2*j+p][2*i+m] += k[p+2][m]
            nStiffnessMatrix[2*j+p][2*j+m] += k[p+2][2+m]

def Initialize(nStiffnessMatrix,nU,nF):
    for i in range(0,Nodes):
        nU[i][0] = 0
        nF[i][0] = 0

    for i in range(0,NumberOfElement):
        AssemblyStiffness(nStiffnessMatrix,K[i],
        int(Elements[i][0]),int(Elements[i][1]))
```

Todos los demás elementos del código permanecen igual; ahora solo se necesita definir las condiciones del problema a resolver.

Código 3: Condiciones del problema

```
NodesCondition = []
Nodes = 5
Nodes *= 2
NumberOfElement = 6

h = 1500e-3#m
E = 3.2e8 #kPA
K = []
A = (0.25*np.pi*(50e-3)**2)*np.ones(Nodes) #m^2
L = []
P_A = 5000e-3 #kN
P_B = 4200e-3 #kN
P_C = 2500e-3 #kN
```

```
P_E = 3000e-3 #kN

PosNodes = np.array([(0,0),(h,0),(0,h),(h,h),(h,2*h)])
Elements = np.array([(0,2),(1,2),(1,3),(2,3),(2,4),(3,4)])

for i in range(0,NumberOfElement):
    L.append(DistNodes(PosNodes[Elements[i][0]],PosNodes[Elements[i][1]]))

L = np.array(L)

for i in range(0,NumberOfElement):
    aux = np.zeros((4,4))
    angle = L[i][1]
    rows = [np.cos(angle),np.sin(angle),-np.cos(angle),-np.sin(angle)]
    cols = [np.cos(angle),np.sin(angle),-np.cos(angle),-np.sin(angle)]
    for j in range(0,4):
        for k in range(0,4):
            aux[j][k] = rows[j]*cols[k]
    aux = aux*E*A[i]/L[i][0]
    K.append(aux)

StiffnessMatrix = np.zeros((Nodes,Nodes))

U = np.zeros(Nodes).reshape(Nodes,1)
F = np.zeros(Nodes).reshape(Nodes,1)

Initialize(StiffnessMatrix,U,F)

#Node in UBoundary = Node*2+(x=0,y=1)
UBoundaryCondition(U,0,2*0+0) #Nodo 0 en X
UBoundaryCondition(U,0,2*0+1) #Nodo 0 en Y
UBoundaryCondition(U,0,2*1+0) #Nodo 1 en X
UBoundaryCondition(U,0,2*1+1) #Nodo 1 en Y

FBoundaryCondition(F,-P_C,2*2+0) #Nodo 2 en X
FBoundaryCondition(F,-P_E,2*3+0) #Nodo 3 en X
FBoundaryCondition(F,-P_B,2*4+0) #Nodo 4 en X
FBoundaryCondition(F,P_A,2*4+1) #Nodo 4 en Y

U,F=Solve(StiffnessMatrix,U,F)

print("Stiffness Matrix:\n",StiffnessMatrix,'\n')

print("Displacements:\n",U,'\n')

print("Forces:\n",F)
```

La notación para las condiciones del problema es muy similar a los problemas anteriores de barras, siendo la diferencia más notable que no hay una relación directa entre nodos y cantidad de elementos. La representación esquemática del modelo a resolver se muestra a continuación:

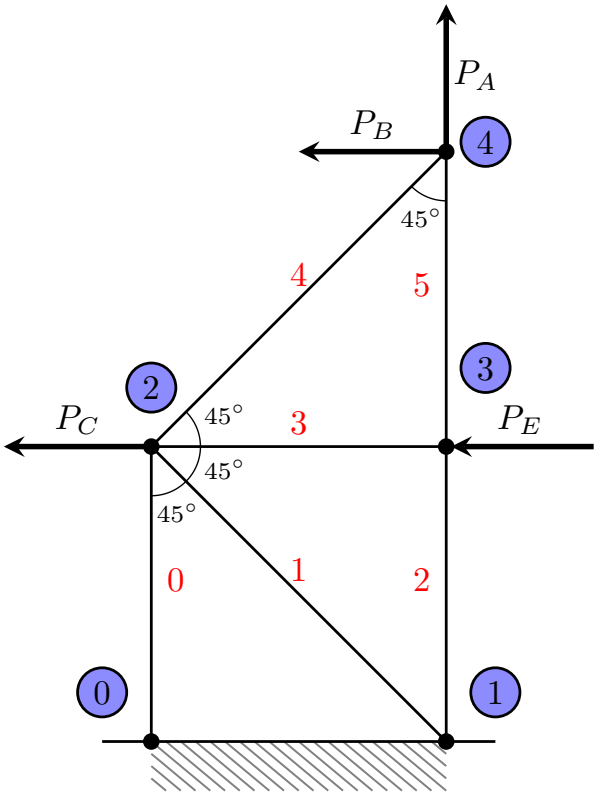

Figura 2: Representación esquemática

La geometría del problema ha sido rotada 90° para que los apoyos sean definidos correctamente. También en la figura (2) se ha denotado los nodos con color azul y los elementos con color rojo.

3 Resultados del problema

Al finalizar la ejecución del solver, obtenemos la matriz de rigidez, las fuerzas y desplazamientos:

```
Stiffness Matrix:
 [[ 1.57054477e-27  2.56489426e-11  0.00000000e+00  0.00000000e+00
  -1.57054477e-27 -2.56489426e-11  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 2.56489426e-11  4.18879020e+05  0.00000000e+00  0.00000000e+00
  -2.56489426e-11 -4.18879020e+05  0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  1.48096098e+05 -1.48096098e+05
  -1.48096098e+05  1.48096098e+05 -1.57054477e-27 -2.56489426e-11
   0.00000000e+00  0.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00 -1.48096098e+05  5.66975118e+05
   1.48096098e+05 -1.48096098e+05 -2.56489426e-11 -4.18879020e+05
   0.00000000e+00  0.00000000e+00]
 [-1.57054477e-27 -2.56489426e-11 -1.48096098e+05  1.48096098e+05
   7.15071216e+05  2.91038305e-11 -4.18879020e+05  0.00000000e+00
  -1.48096098e+05 -1.48096098e+05]
 [-2.56489426e-11 -4.18879020e+05  1.48096098e+05 -1.48096098e+05
   2.91038305e-11  7.15071216e+05  0.00000000e+00  0.00000000e+00
  -1.48096098e+05 -1.48096098e+05]
 [ 0.00000000e+00  0.00000000e+00 -1.57054477e-27 -2.56489426e-11
  -4.18879020e+05  0.00000000e+00  4.18879020e+05  5.12978852e-11
  -1.57054477e-27 -2.56489426e-11]
 [ 0.00000000e+00  0.00000000e+00 -2.56489426e-11 -4.18879020e+05
   0.00000000e+00  0.00000000e+00  5.12978852e-11  8.37758041e+05
  -2.56489426e-11 -4.18879020e+05]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -1.48096098e+05 -1.48096098e+05 -1.57054477e-27 -2.56489426e-11
   1.48096098e+05  1.48096098e+05]
 [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  0.00000000e+00
  -1.48096098e+05 -1.48096098e+05 -2.56489426e-11 -4.18879020e+05
   1.48096098e+05  5.66975118e+05]]
```

```
Displacements:
 [[ 0.00000000e+00]
 [ 0.00000000e+00]
 [ 0.00000000e+00]
 [ 0.00000000e+00]
 [-9.86818176e-05]
 [-3.31838056e-05]
 [-1.05843790e-04]
 [ 2.19633821e-05]
 [-2.04152352e-04]
 [ 4.39267643e-05]]

Forces:
 [[ 8.51129525e-16]
 [ 1.39000000e+01]
 [ 9.70000000e+00]
 [-1.89000000e+01]
 [-2.50000000e+00]
 [ 6.68762823e-11]
 [-3.00000000e+00]
 [ 3.22373239e-11]
 [-4.20000000e+00]
 [ 5.00000000e+00]]
```

Figura 3: Matriz de rigidez, desplazamientos y fuerzas

Cada desplazamiento y fuerza corresponde a un nodo en una dirección (x o y); por lo que al nodo i le corresponde las reacciones $2i$ (x) y $2i+1$ (y).
Organizamos los datos en una tabla para cada nodo/elemento para tener una mejor compresión de los resultados:

Nodo	Fuerza x (kN)	Fuerza y (kN)	Elemento	Esfuerzo (MPa)
0	0	13.9	0	-7.07921187
1	9.7	-18.9	1	6.98645461
2	-2.5	0	2	4.68552152
3	-3	0	3	-1.52788745
4	-4.2	5	4	-3.02506282
			5	4.68552152

Cuadro 1: Reacciones y esfuerzos sobre cada nodo y elemento

4 Problema generalizado para n elementos

Es posible incrementar la cantidad de elementos en el problema; imagine cada elemento inicial de la armadura que se subdivide en m partes y cada parte está unida mediante un nuevo nodo. Sin embargo, los resultados son muy similares aún incrementando la cantidad de elementos.

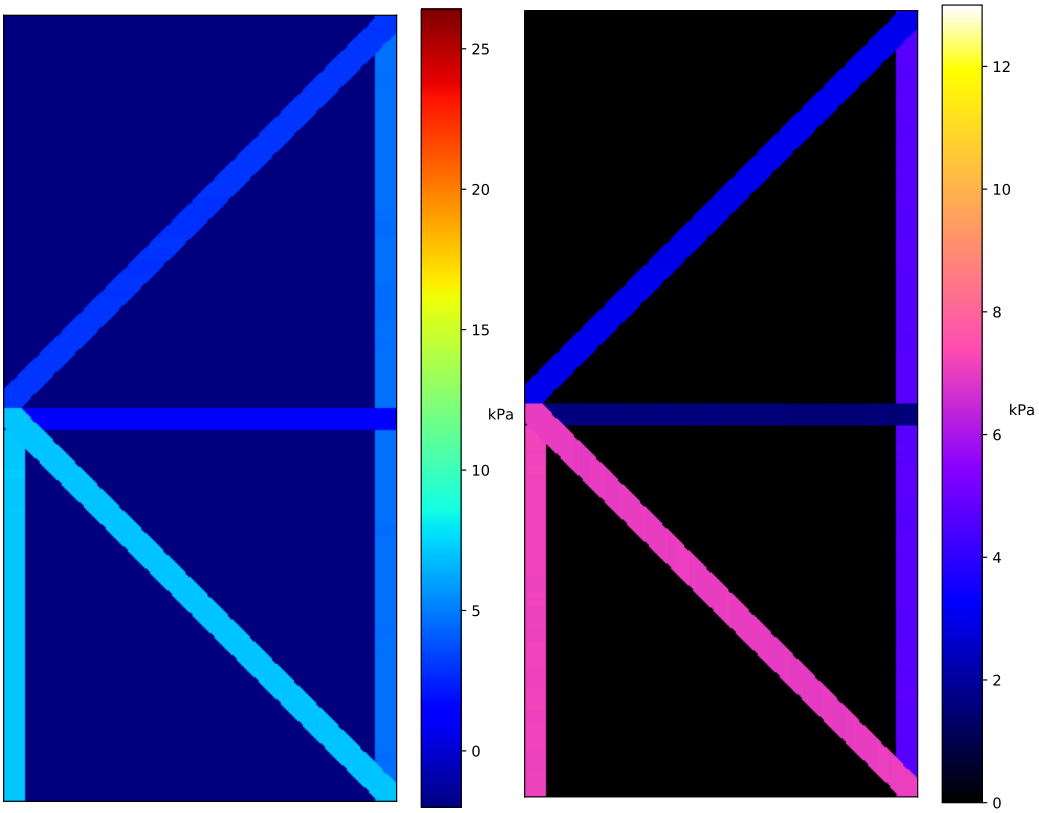

Figura 4: Esfuerzos para 1500 elementos

Observe que, ambas gráficas son las mismas, solo se modificó la escala para que sea más fácil notar la variación de esfuerzos en los elementos.

5 Verificación de resultados

Para la verificación de los cálculos se utilizó el software Autodesk Fusion 360.

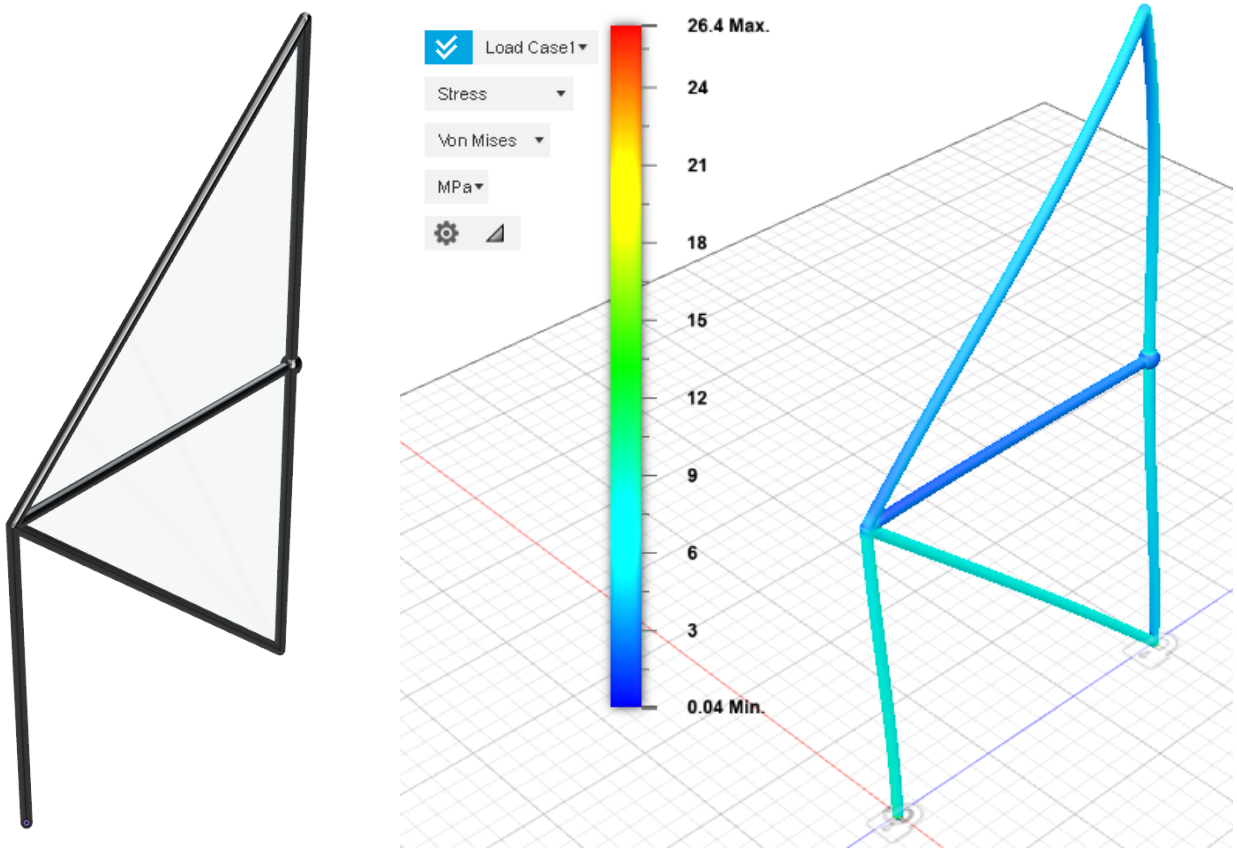

Figura 5: Geometría y esfuerzos de la armadura en Autodesk Fusion 360

6 Conclusiones

1. El esfuerzo máximo es distinto al calculado debido a que la fuerza se aplica de forma puntual y no de forma distribuida sobre un área; tan bien, sabemos que la reacción en el apoyo no se distribuye de forma regular sobre el área.

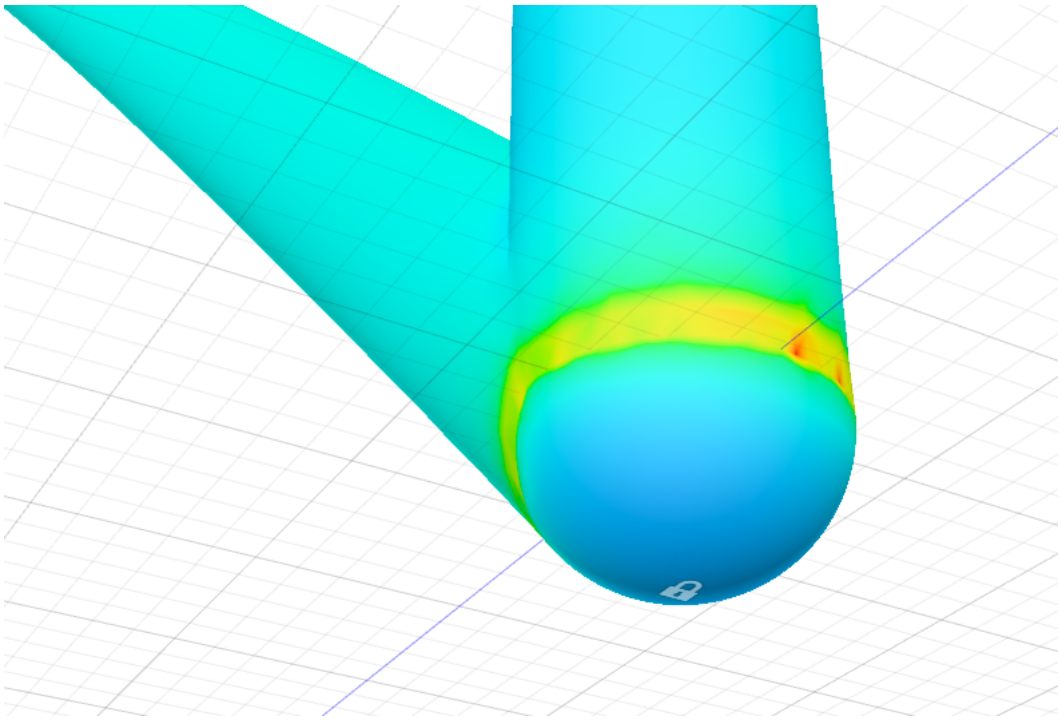

Figura 6: Esfuerzo máximo sobre un apoyo (nodo 1)

2. La implementación del código para una armadura no difiere mucho del hecho para barras, debido a la versatilidad y flexibilidad con la que se desarrolló el código inicial.

3. Los resultados brindados por la simulación en Fusion 360, son muy similares a los obtenidos por el código, siendo los elementos de mayor esfuerzo el 0 y el 1; mientras que el de menor esfuerzo es el elemento 3.

4. Si bien, es posible incrementar la cantidad de elementos en la armadura, no tiene un gran efecto sobre los resultados y es suficiente tratar a cada elemento de la armadura como uno solo.

5. El tiempo esperado por solución es de a lo mucho 3 segundos para 10000 elementos en C++.

6. La implementación del código en MATLAB es más simple y corta pero demasiado lenta; tardando varios minutos para ejecutar 1000 elementos.

7. Debido a limitaciones de memoria no es posible usar más elementos, siendo el límite de 20000 elementos; usando 3.2 GB de RAM.

Lenguaje/Cantidad de elementos	Tiempo de ejecución (s)
C++/5000	2.1
Python/5000	30.34
MATLAB/5000	5854.79

Referencias

[1] Optimized methods in FEM:
https://www.sciencedirect.com/topics/engineering/gauss-seidel-method

[2] Sparse Matrix:
https://en.wikipedia.org/wiki/Sparse_matrix

[3] Sparse Matrix Library:
https://github.com/uestla/Sparse-Matrix

[4] Mailman algorithm:
http://www.cs.yale.edu/homes/el327/papers/matrixVectorApp.pdf

[5] Fast Algorithms with Preprocessing for Matrix-Vector Multiplication Problems:
https://www.sciencedirect.com/science/article/pii/S0885064X84710211