5° Práctica de Cálculo por Elementos Finitos - MC516

Josue Huaroto Villavicencio - 20174070I
Sección: E

7 de septiembre de 2020

1 Análisis analítico

De las ecuaciones de equlibrio de la estática:

$$\begin{aligned} R_1 &= -R_2 \\ R_2 \times 1000 &= -M \\ R_2 &= -200\,\mathrm{N} \\ R_1 &= 200\,\mathrm{N} \end{aligned}$$

Se halla los correspondientes diagramas de momento flector y de fuerza cortante.

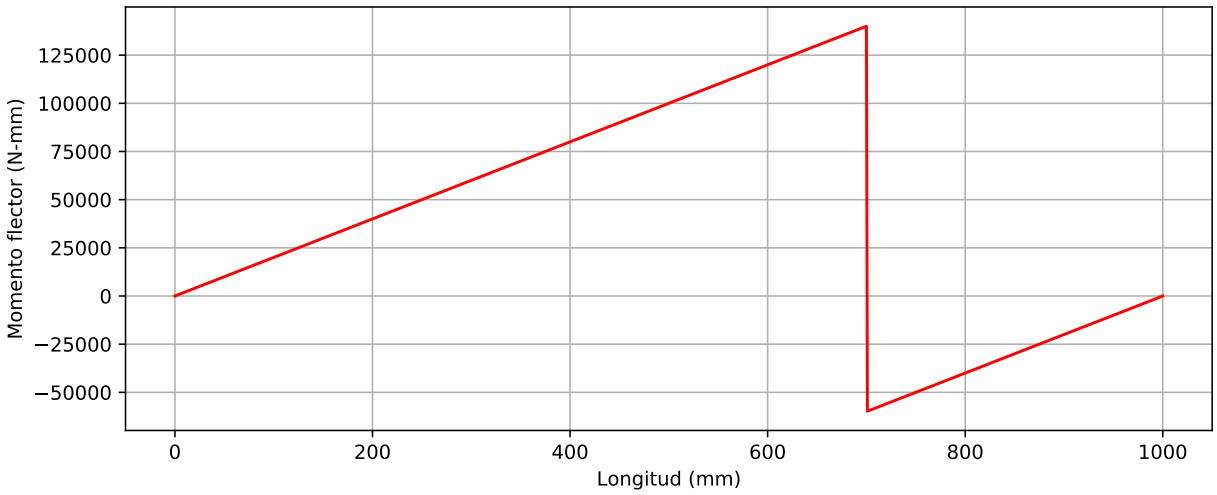


Figura 1: Diagrama de momento flector

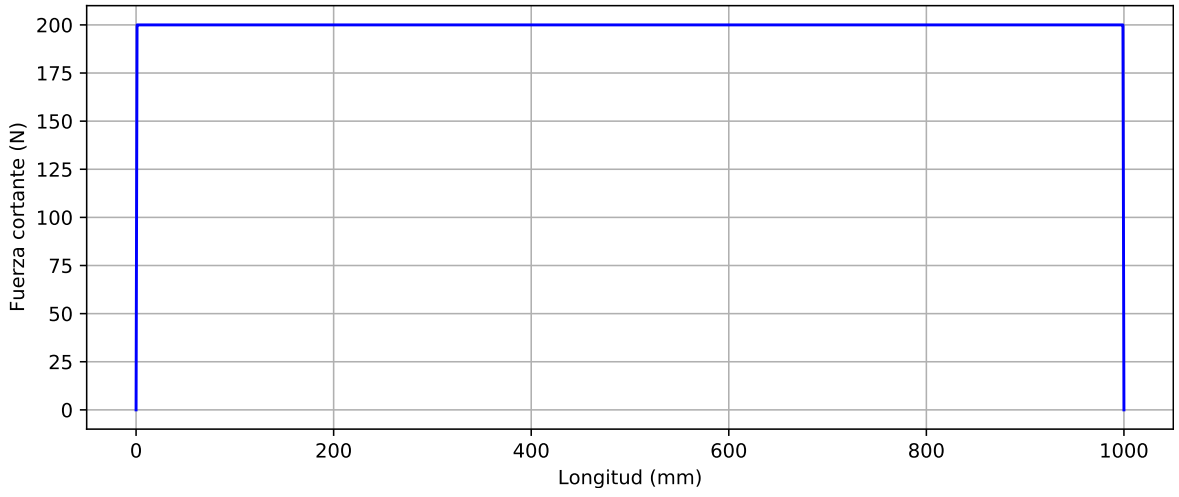


Figura 2: Diagrama de fuerza cortante

Luego, reescribiendo la función $M(x)$:

$$M(x) = R_1 < x - 0 >^1 - M < x - 700 >^0 + R_2 < x + 1000 >^1 \tag{1}$$

La ecuación diferencial correspondiente a la deformada de una viga:

$$EI\frac{\mathrm{d}y^2}{\mathrm{d}x^2} = M(x) \tag{2}$$

$$y(x) = \frac{1}{EI}\iint M(x)\mathrm{d}x\ \mathrm{d}x \tag{3}$$

$$y(x) = \frac{\frac{R_1 x^3}{6} + c_1 x + c_2}{EI} \quad | \quad x \leq 700 \tag{4}$$

$$y(x) = \frac{\frac{R_1 x^3}{6} - \frac{M x^2}{2} + c_3 x + c_4}{EI} \quad | \quad x \geq 700 \tag{5}$$

De las condiciones de frontera:

$$c_2 = 0 \tag{6}$$

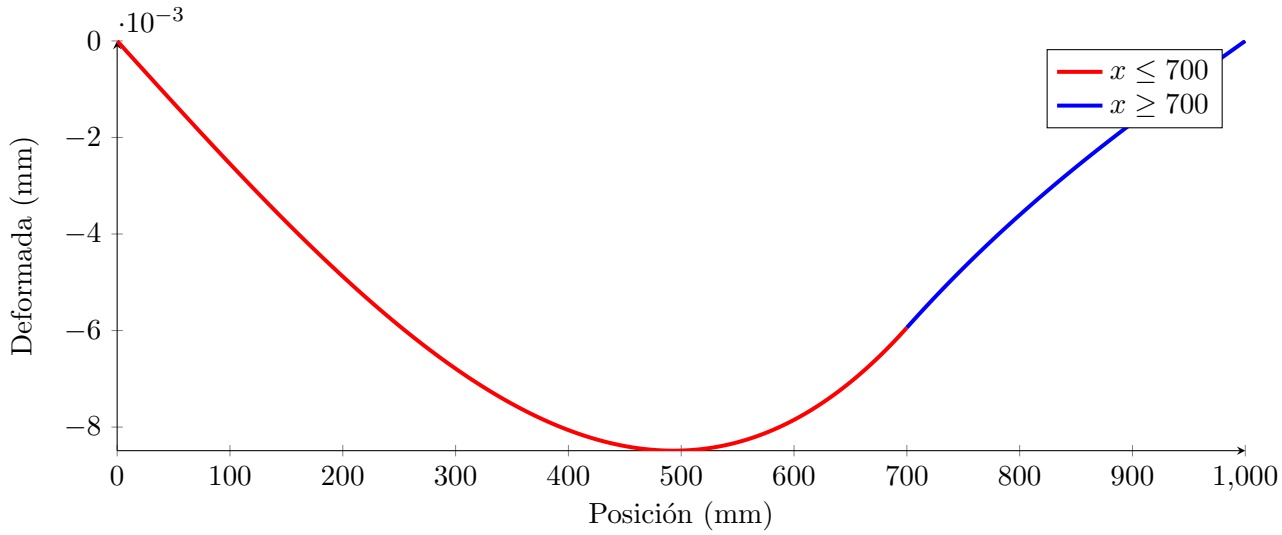
$$c_3 - 700M = c_1 \tag{7}$$

$$c_4 = -49 \times 10^9 \tag{8}$$

$$c_3 = 115666666.66667 \tag{9}$$

$$c_1 = -24333333.3333 \tag{10}$$

$$y(x) = \begin{cases} 3.536068 \cdot 10^{-11}x^3 - 2.58133 \cdot 10^{-5}x & x \leq 700 \\ 3.536068 \cdot 10^{-11}x^3 - 1.06082 \cdot 10^{-7}x^2 + 1.227 \cdot 10^{-4}x - 0.0519802 & x \geq 700 \end{cases} \tag{11}$$



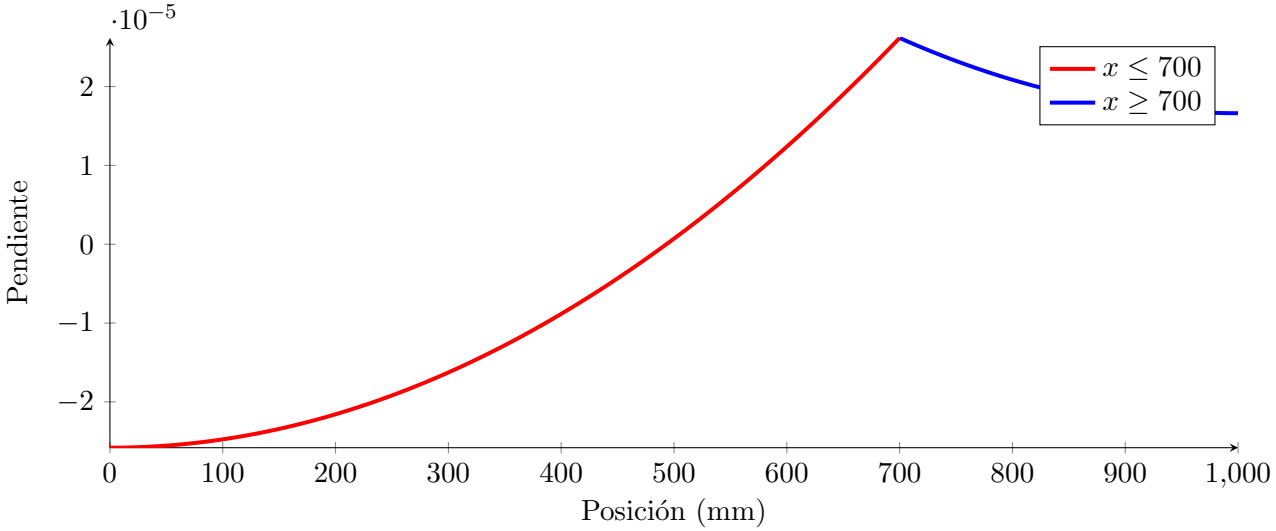
Derivando la ecuación (4), se obtiene el valor de la flecha máxima y su posición en la viga:

$$x_{\max} = 493.28828623$$
$$y_{\max} = 0.0084889329$$

Derivando la ecuación (11), se obtiene la gráfica de la pendiente de la deformada:

$$\theta(x) = \begin{cases} 10.60820368 \cdot 10^{-11}x^2 - 2.58132956 \cdot 10^{-5} & x \leq 700 \\ 10.60820368 \cdot 10^{-11}x^2 - 2.1216407 \cdot 10^{-7}x + 1.227 \cdot 10^{-4} & x \geq 700 \end{cases} \tag{12}$$

Siendo la pendiente en $L_1 = 700\text{mm}$, igual a 2.61669×10^{-5}.



2 Ejecución del código

Se agrega al solver principal una nueva función llamada DFBoundaryCondition que se utilizará para agregar una fuerza distribuida sobre un elemento.

Código 1: Carga distribuida sobre un elemento

```
def DFBoundaryCondition(nF,w,e):
    f = Elements[e][0]
    s = Elements[e][1]
    l = L[e]
    FBoundaryCondition(nF,w*l/2,2*f+0)
    FBoundaryCondition(nF,w*l*l/12,2*f+1)
    FBoundaryCondition(nF,w*l/2,2*s+0)
    FBoundaryCondition(nF,-w*l*l/12,2*s+1)
```

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 ElementStiffness(E, I, L):
    aux = np.zeros((4,4))
    s1 = np.array([12,6*L,-12,6*L])
    s2 = np.array([6*L,4*L*L,-6*L,2*L*L])
    for i in range(0,2):
        for j in range(0,4):
            aux[2*i][j] = s1[j]
            aux[2*i+1][j] = s2[j]
            if(i == 1):
                aux[2*i][j] *= -1
    aux[3][1] = 2*L*L
    aux[3][3] = 4*L*L
    aux = aux*E*I/(L**3)
    return aux

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]
```

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 = 1001
Nodes *= 2
NumberOfElement = 1000

E = 3e5 #MPA
K = []
I = 314.2222e4 #mm^4
M_A = 2e5 #N-mm
PosNodes = np.linspace(0,1000,NumberOfElement+1)
Elements = []
for i in range(0,NumberOfElement):
    Elements.append((i,i+1))
Elements = np.array(Elements)

L = []

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

L = np.array(L)

for i in range(0,NumberOfElement):
    aux = ElementStiffness(E,I,L[i])
    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)


UBoundaryCondition(U,0,2*0+0) #Nodo 0 en Y
UBoundaryCondition(U,0,2*NumberOfElement+0) #Nodo 1000 en Y

FBoundaryCondition(F,M_A,int(2*NumberOfElement*700/1000)+1) #Momento en el nodo 700

U,F=Solve(StiffnessMatrix,U,F)
print("Stiffness Matrix:\n",StiffnessMatrix,'\n')
print("Displacements:\n",U,'\n')
print("Forces:\n",F)
```

3 Resultados del problema

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

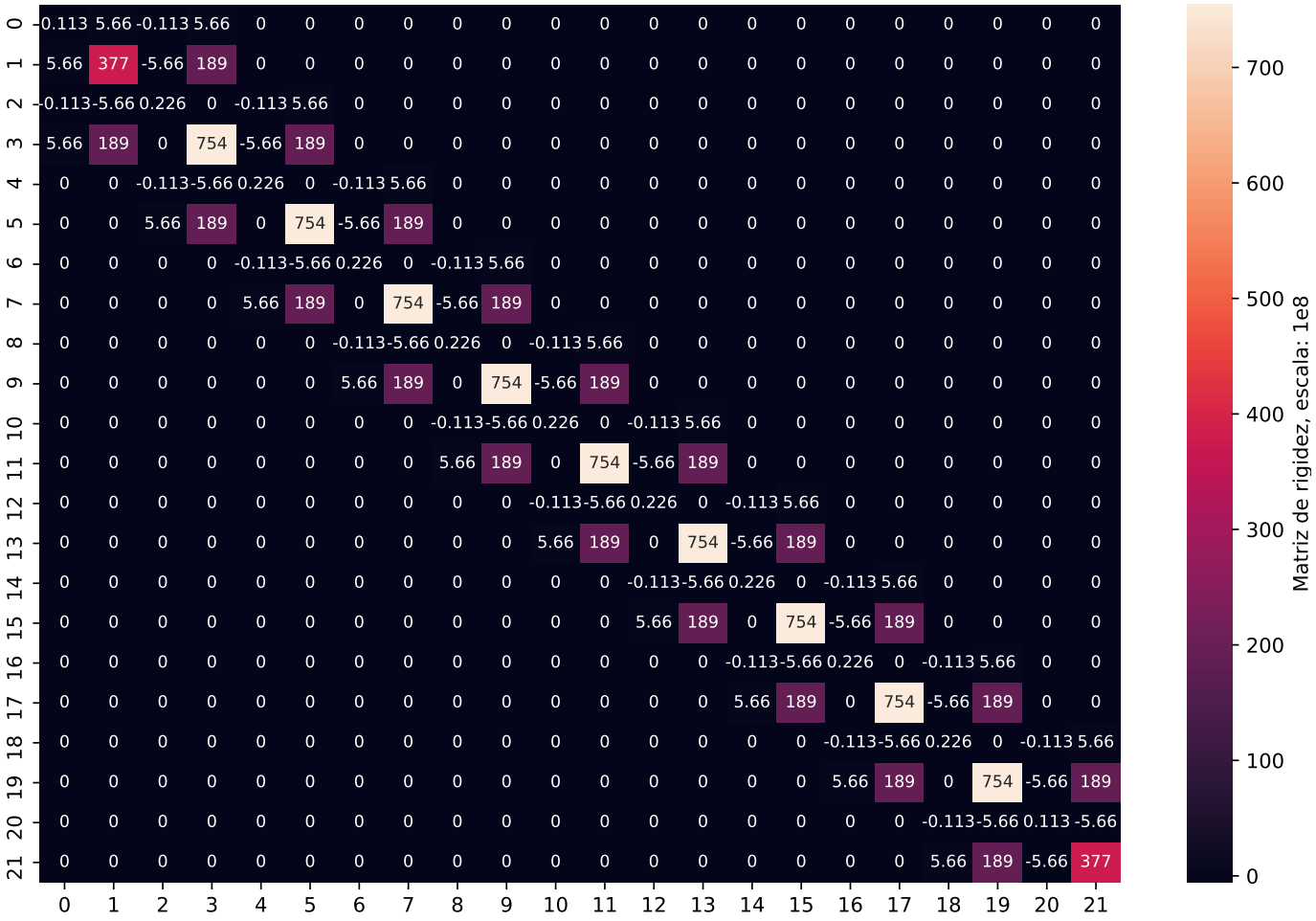


Figura 3: Matriz de rigidez

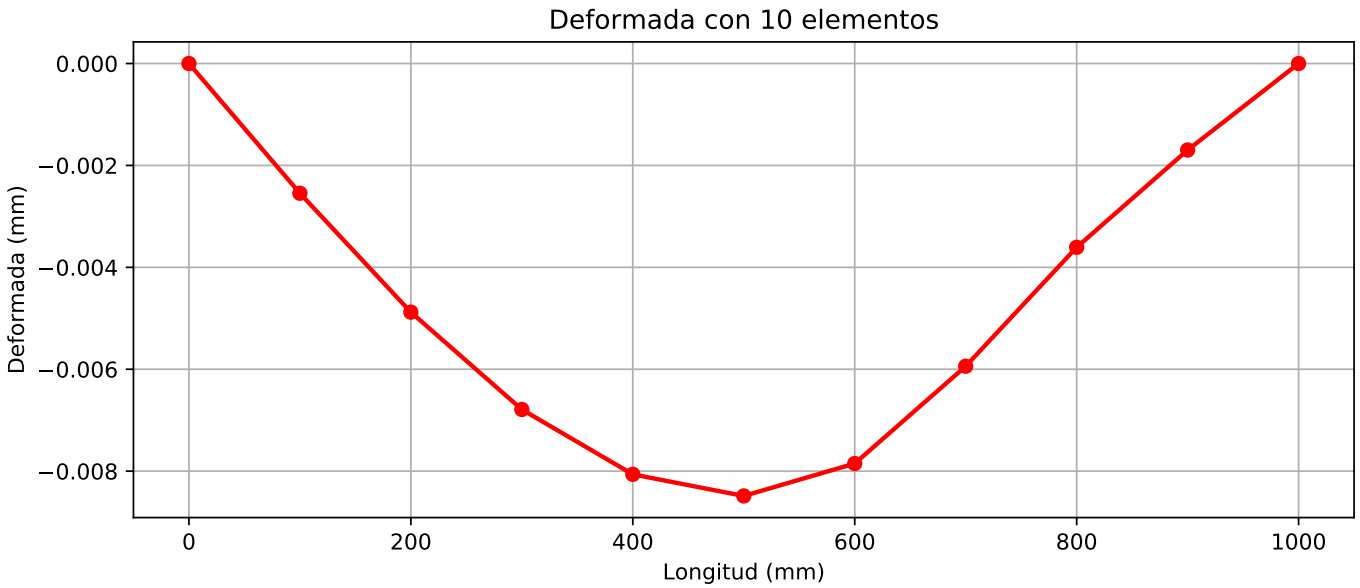


Figura 4: Deformada

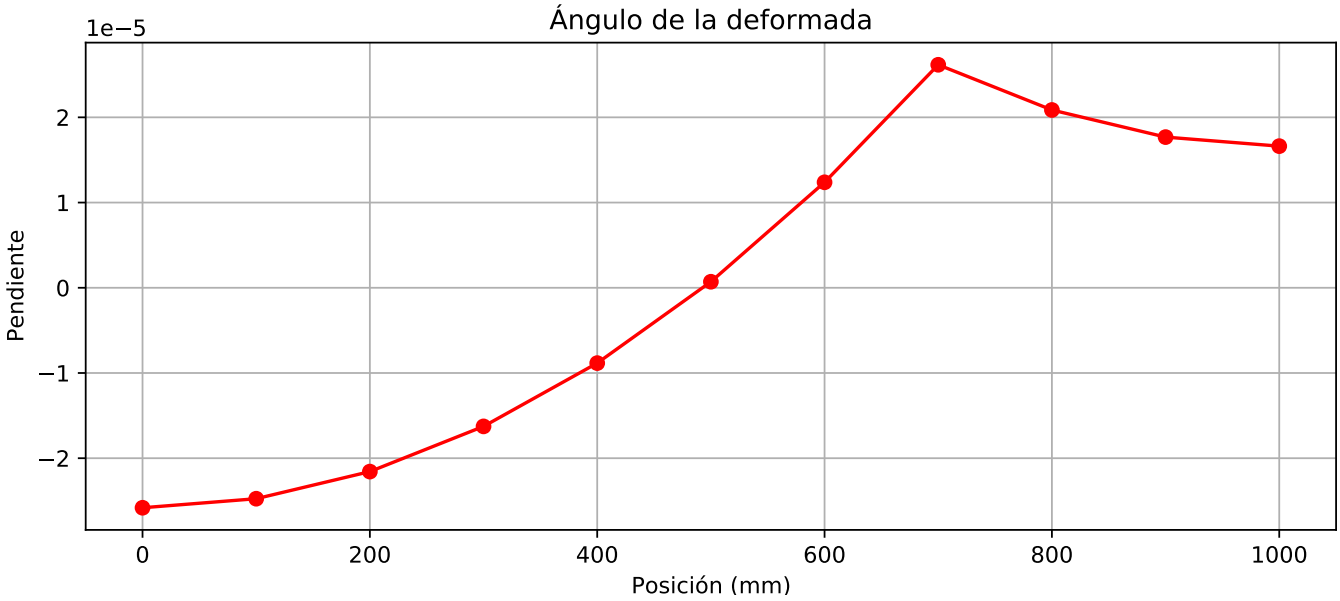


Figura 5: Pendiente

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

Resultados del análisis por elementos finitos				
Nodo	Desp. y (mm)	Desp. θ (rad)	Fuerza y (N)	Momento θ (N-mm)
0	0	-25.8133 $\times 10^{-6}$	200	0
1	-2.54597 $\times 10^{-3}$	-24.75248 $\times 10^{-6}$	-2.18279 $\times 10^{-11}$	0
2	-4.87977 $\times 10^{-3}$	-21.57001 $\times 10^{-6}$	7.09406 $\times 10^{-11}$	-4.65661 $\times 10^{-10}$
3	-6.78925 $\times 10^{-3}$	-16.26591 $\times 10^{-6}$	-1.18234 $\times 10^{-11}$	6.98492 $\times 10^{-10}$
4	-8.06223 $\times 10^{-3}$	-8.84017 $\times 10^{-6}$	-1.63709 $\times 10^{-11}$	0
5	-8.48656 $\times 10^{-3}$	0.70721 $\times 10^{-6}$	0	-3.63798 $\times 10^{-10}$
6	-7.85007 $\times 10^{-3}$	12.37624 $\times 10^{-6}$	-4.18368 $\times 10^{-11}$	4.65661 $\times 10^{-10}$
7	-5.94059 $\times 10^{-3}$	26.1669 $\times 10^{-6}$	1.90994 $\times 10^{-11}$	200000
8	-3.60679 $\times 10^{-3}$	20.8628 $\times 10^{-6}$	1.54614 $\times 10^{-11}$	0
9	-1.69731 $\times 10^{-3}$	17.68034 $\times 10^{-6}$	-0.1819 $\times 10^{-11}$	-1.16415 $\times 10^{-10}$
10	0	16.61952 $\times 10^{-6}$	-200	0

Cuadro 1: Desplazamientos y reacciones en los nodos

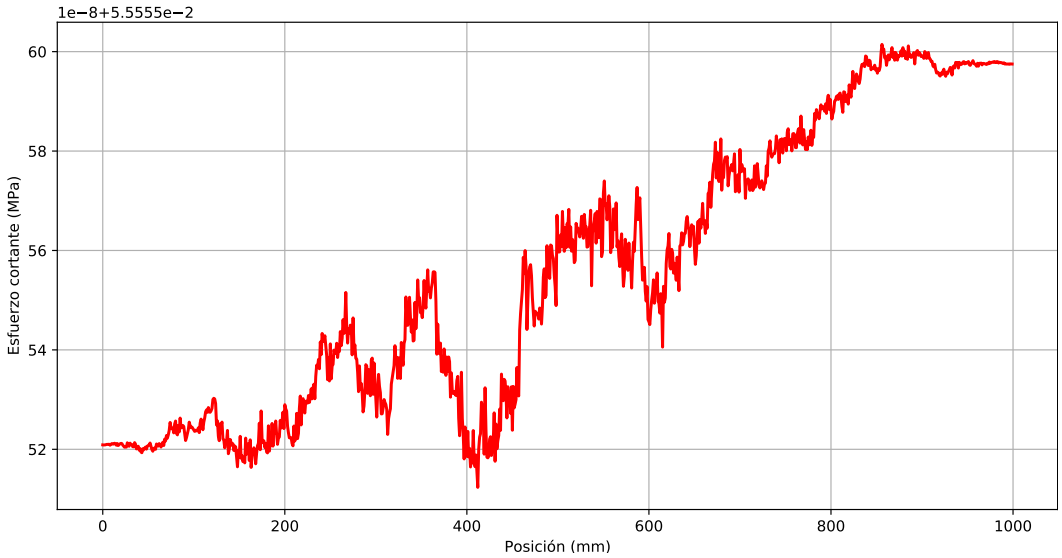


Figura 6: Error en el esfuerzo cortante

Debido a errores en la precisión del computador, se genera ruido en la solución, pero dichos errores son tan pequeños que son despreciables; eliminando el ruido de la gráfica, el esfuerzo cortante sería:

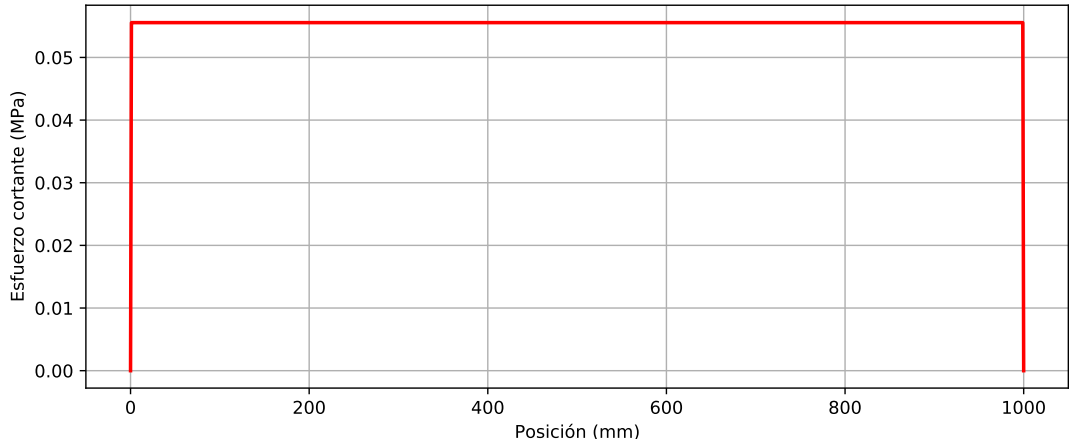


Figura 7: Esfuerzo cortante

Mientras que el esfuerzo máximo por flexión sería:

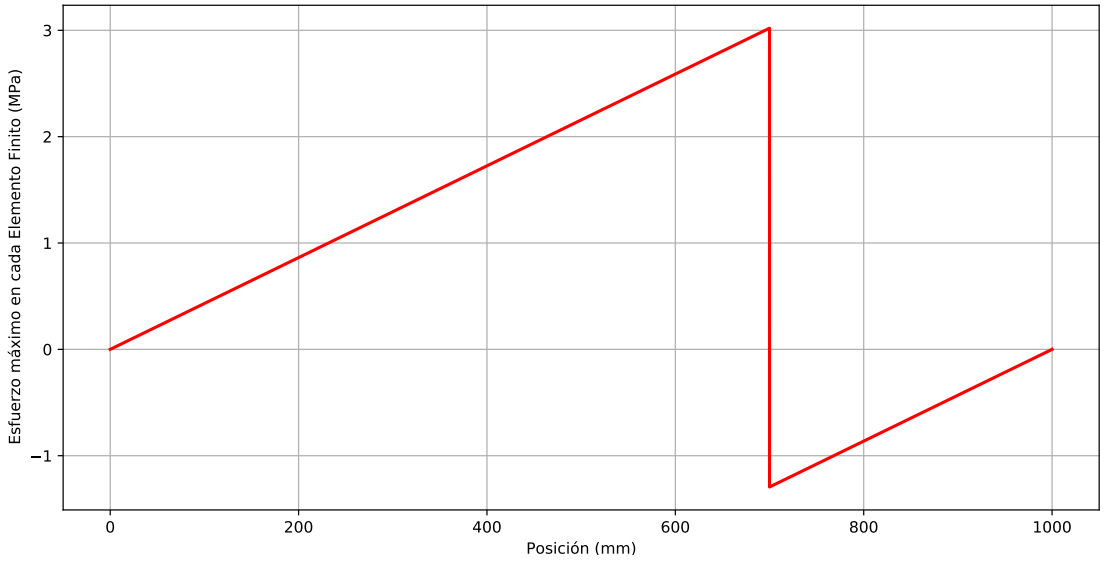


Figura 8: Esfuerzo por flexión

Luego, los esfuerzos en cada elemento:

Esfuerzos en los elementos					
Elemento	Nodo 1	Nodo 2	Longitud	Esfuerzo por flexión (MPa)	Esfuerzo cortante (MPa)
0	0	1	100 mm	0.4314	0.0555
1	1	2	100 mm	0.862801	0.0555
2	2	3	100 mm	1.294201	0.0555
3	3	4	100 mm	1.725601	0.0555
4	4	5	100 mm	2.157001	0.0555
5	5	6	100 mm	2.588402	0.0555
6	6	7	100 mm	3.019802	0.0555
7	7	8	100 mm	-1.294201	0.0555
8	8	9	100 mm	-0.862801	0.0555
9	9	10	100 mm	-0.4314	0.0555

Cuadro 2: Esfuerzos en la viga

4 Generalización para n elementos

Al incrementar la cantidad de elementos se obtiene mayor información sobre la pendiente y la deformada en más secciones de la viga.

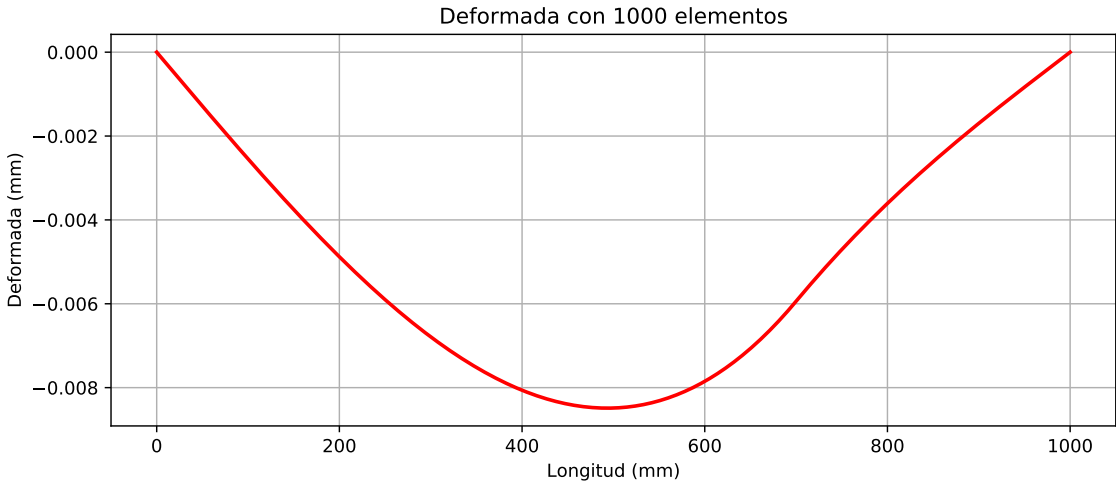


Figura 9: Deformada para 1000 elementos

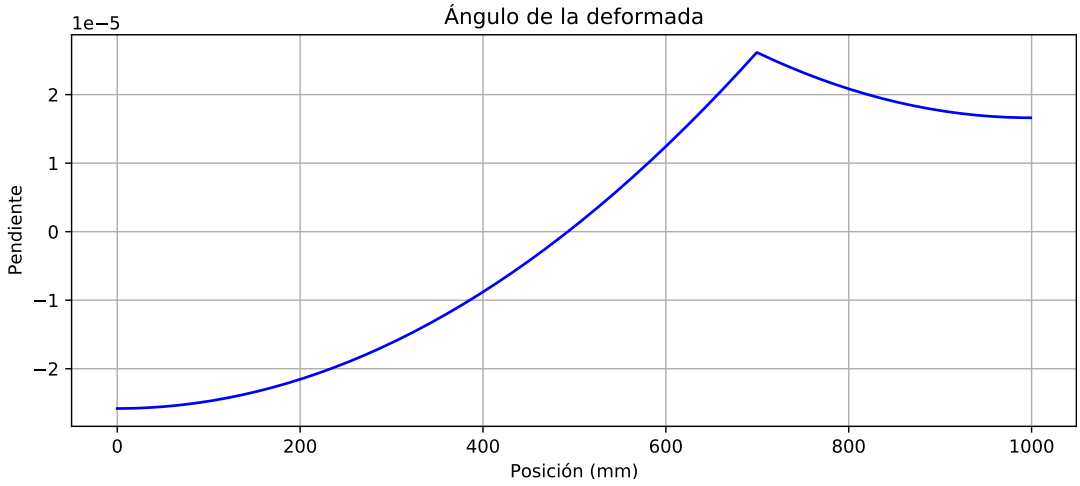


Figura 10: Pendiente para 1000 elementos

5 Verificación de resultados en Autodesk Fusion 360

Se modela la viga y sus apoyos en Fusion 360, para la verificación de resultados.

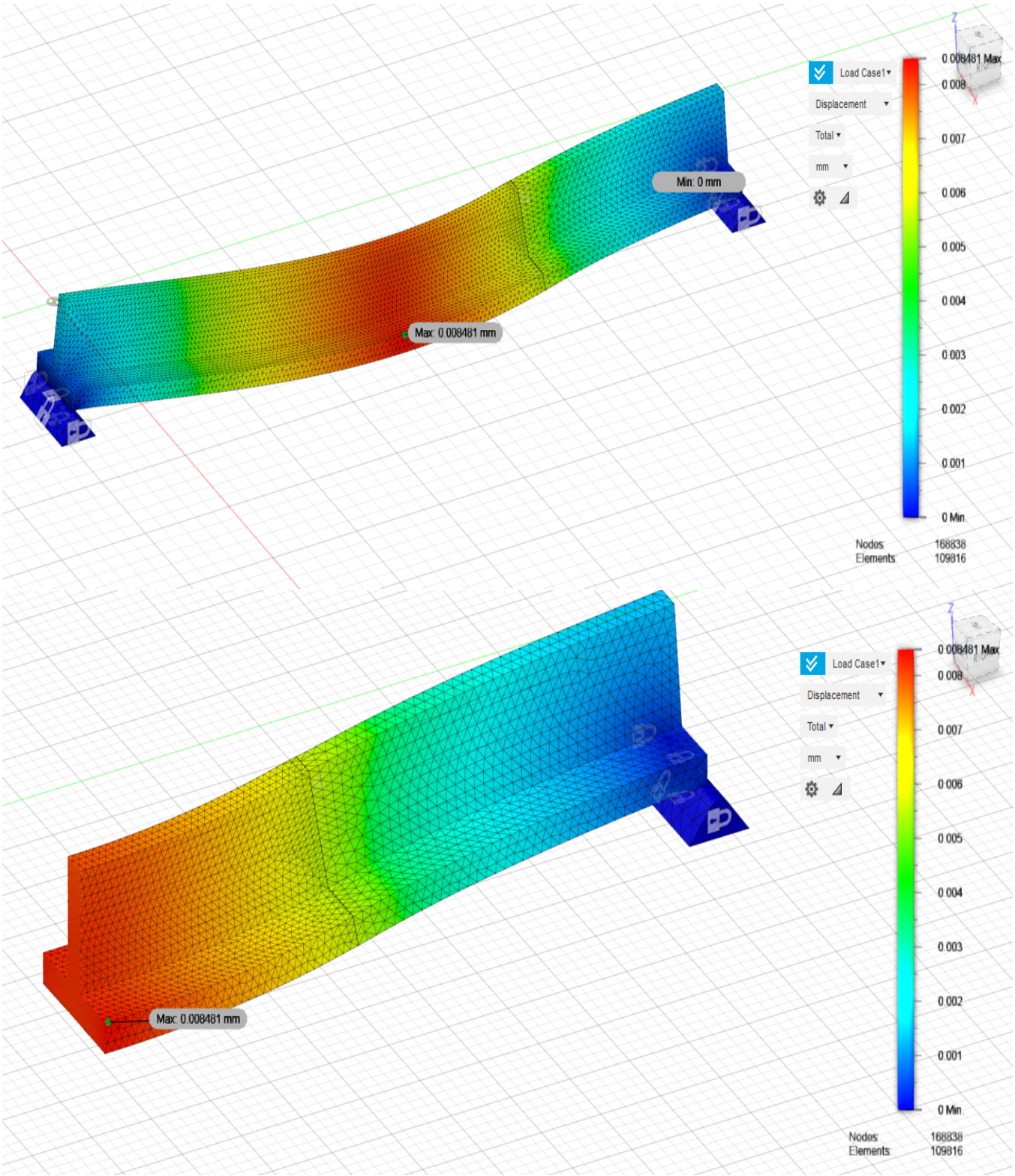


Figura 11: Deformación y flecha máxima de la viga a 493 mm

Los resultados son similares a los obtenidos con el código detallado en la sección 2. Tanto la flecha máxima como su posición en la viga coinciden con los resultados mostrados anteriormente.

6 Verificación de resultados en SolidsPy

Utilizando el software SolidsPy [1]; es posible modelar la viga con elementos cuadráticos de forma rápida y simple. Los resultados obtenidos se muestran a continuación:

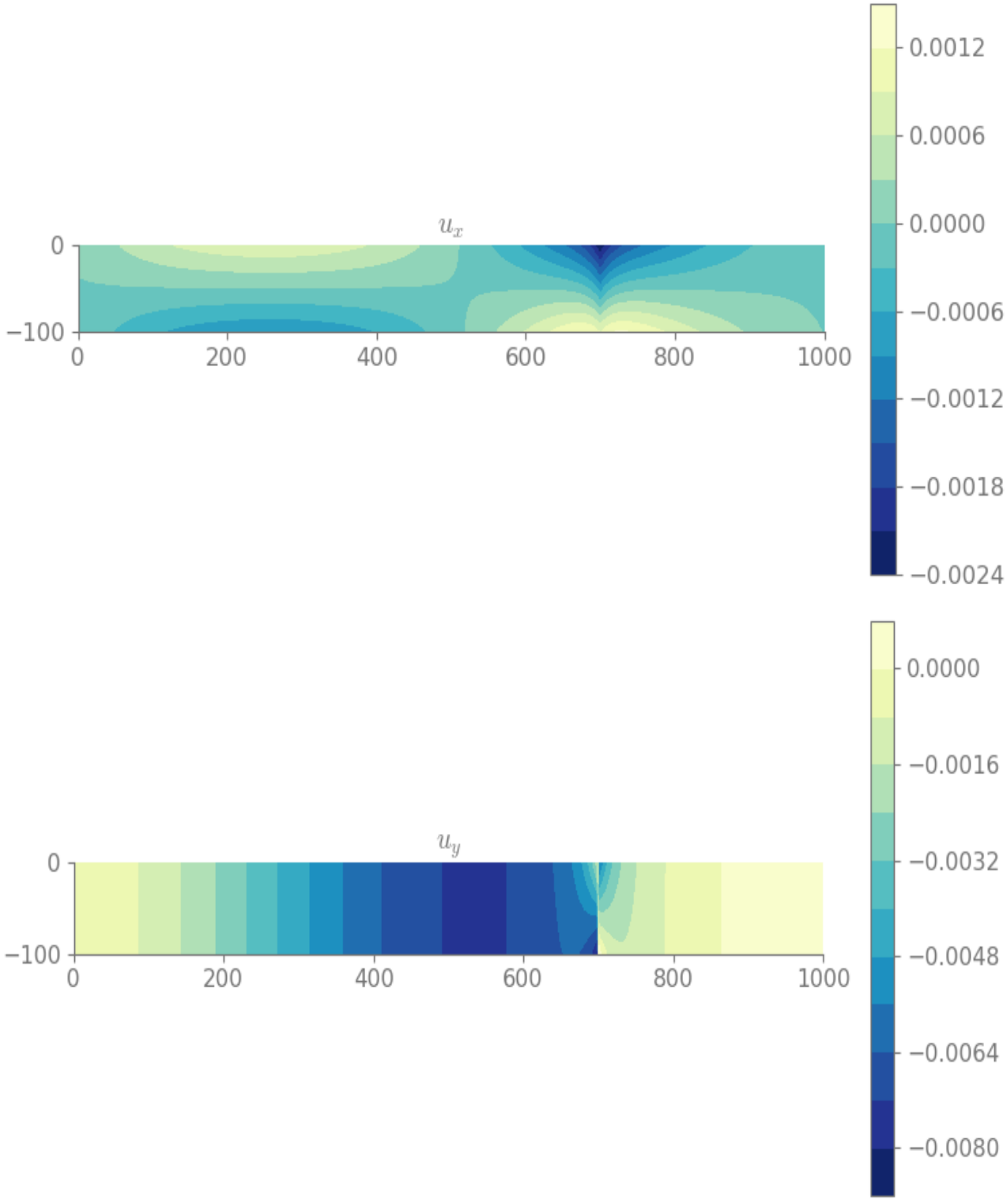


Figura 12: Desplazamientos horizontales y verticales

El desplazamiento horizontal es pequeño y despreciable en comparación con la vertical; mientras que la vertical tiene un valor máximo de 0.008 mm cerca a 500 mm como se demostró en los resultados anteriores y a 700 mm ocurre una variación más atenuada de la pendiente, como se puede apreciar en la gráfica obtenida.

Convergencia del método de elementos finitos

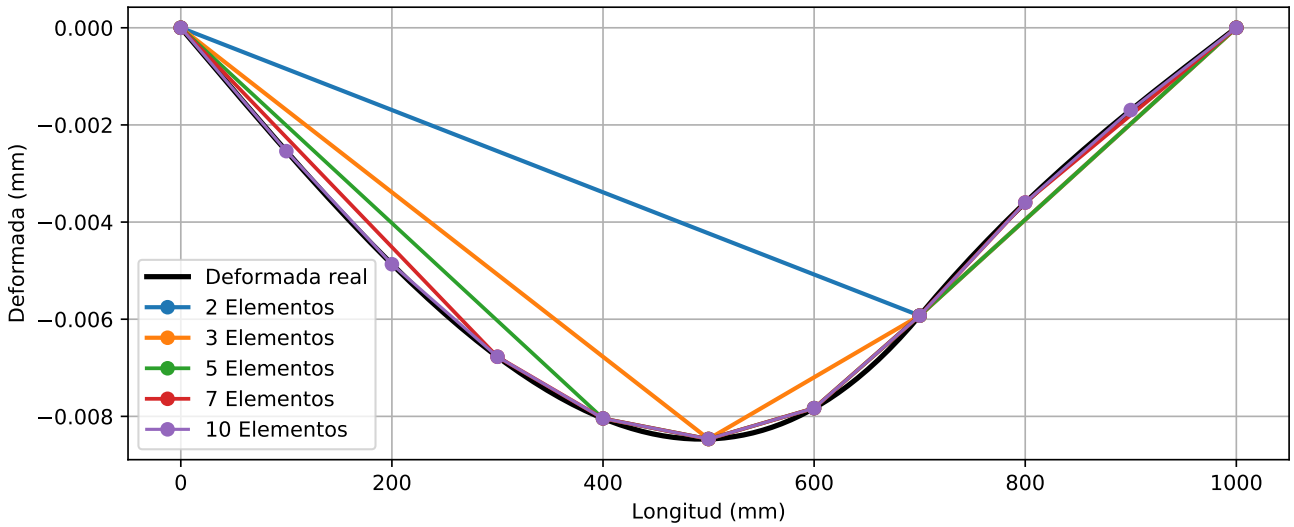


Figura 13: Convergencia de la deformada al aumentar los elementos

Optimización del mallado

Se observa que la flecha máxima se encuentra entre 400 y 500 mm. Para obtener una mayor precisión de la ubicación de la flecha puede aumentarse el mallado, pero esto crearía demasiados elementos, entonces solo se mejora el mallado entre una región más pequeña, mientras que en las otras secciones de la viga el mallado sigue como antes.

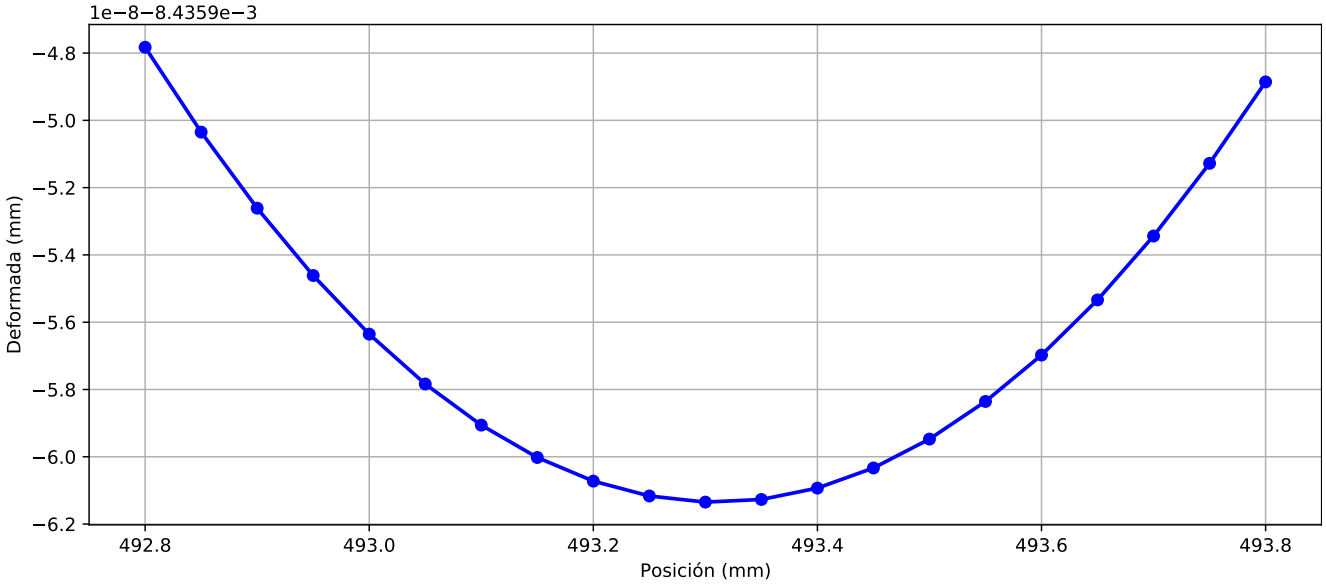


Figura 14: Refinamiento del mallado entre 490 y 500 mm

La flecha máxima está alrededor de 493.3 y tiene un valor de 0.0084889288 mm, similar al resultado obtenido de forma analítica. Con un error relativo igual a 0.000048125%.

Análisis por diferencias finitas

La viga también puede desarrollarse al seguir un procedimiento similar a como se hizo para el curso MC325, utilizando el código de [2] donde se puede usar el método de diferencias finitas para obtener una solución aproximada de la ecuación diferencial de la deformada.

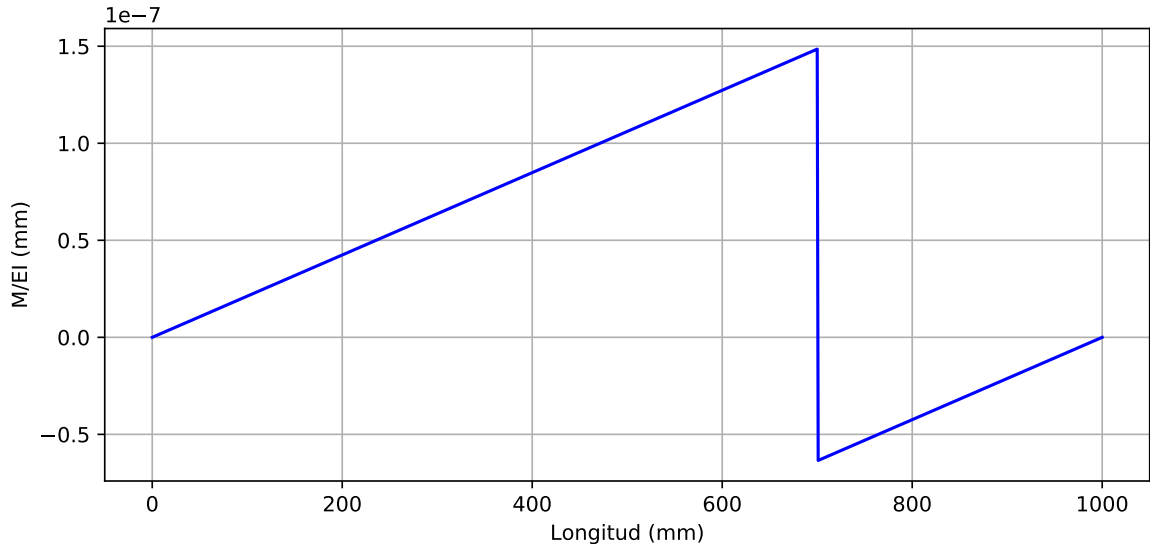


Figura 15: M/EI

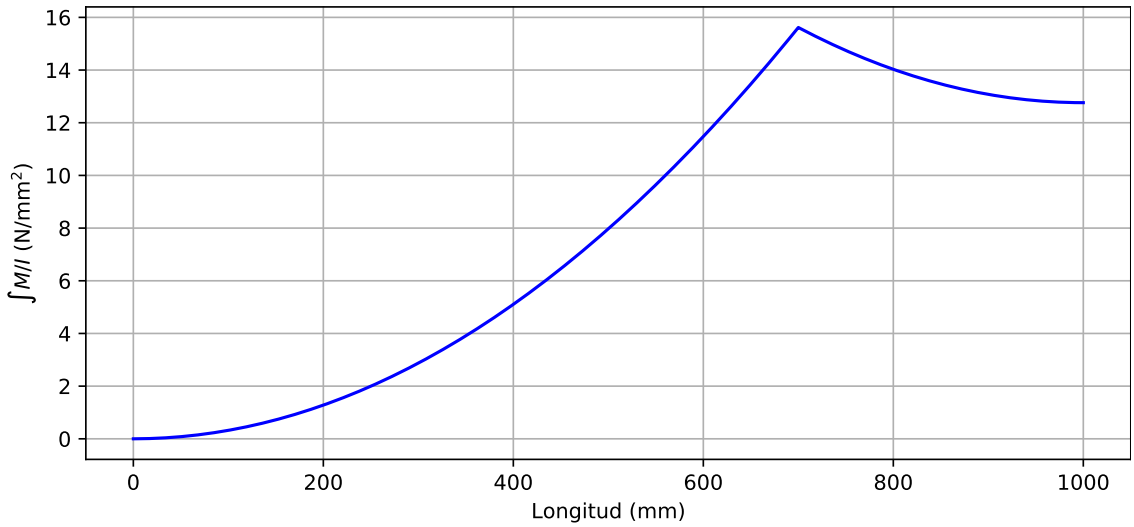


Figura 16: $\int M/I$

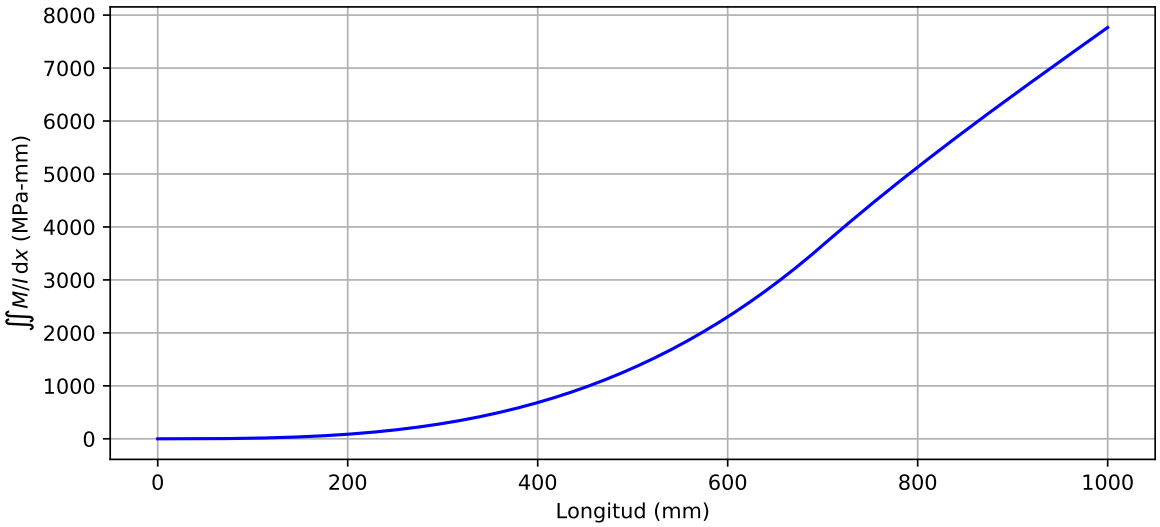


Figura 17: $\iint M/I$

Dado que la integral $\iint M/I$ contiene constantes, es necesario hallarlas y añadirlas a la gráfica final para obtener la deformada:

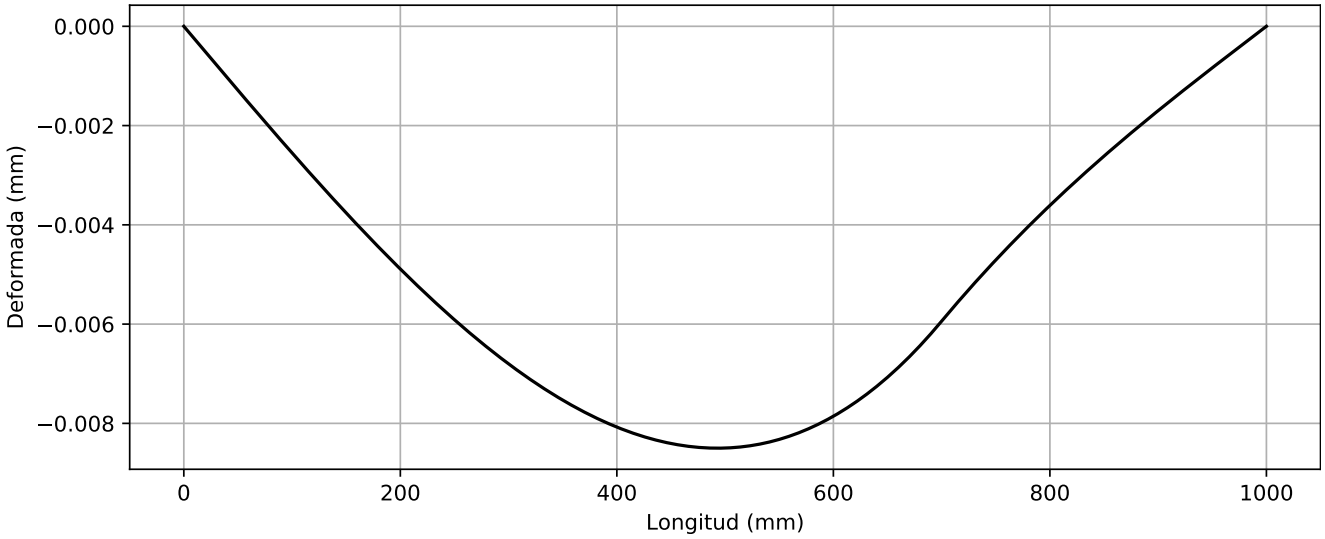


Figura 18: Deformada obtenida por diferencias finitas

Comparación entre diferencias finitas y elementos finitos

El método de diferencias finitas es la implementación de un clásico algoritmo DP, en el que se reutiliza estados anteriores para formar nuevos y así completar la solución completa. En este caso el algoritmo utiliza el estado anterior para así reconstruir la solución; es decir, es el método de Euler para la solución de una ecuación diferencial. Este método tiene una complejidad de tiempo $O(n)$ y la memoria puede optimizarse a $O(1)$, pero su convergencia a la solución es $O(n)$. Existen métodos más eficientes que utilizan aproximaciones más exactas para obtener una convergencia de $O(n^2)$. Mientras que el método de elementos finitos aproxima la región obteniendo una convergencia igual al grado de la solución analítica, es decir $O(n^3/6)$ pero con un uso de memoria $O(n^2)$ y un tiempo de ejecución de $O(n^3)$, con las optimizaciones que se realizan en el código fue posible reducir el tiempo de ejecución a $O(n^2 \log(n))$. Existen optimizaciones más poderosas para elementos finitos que reducen el uso de memoria excesivo usando una estructura de datos [3] que elimina regiones de la matriz que son iguales a 0, de esa forma es posible usar más elementos y segmentar la solución. Como tal, el método de elementos finitos presenta un mejor desarrollo para resolver problemas donde la convergencia obtenida puede ser muy alta, como es el caso de la viga. Esto puede observarse en la siguiente gráfica:

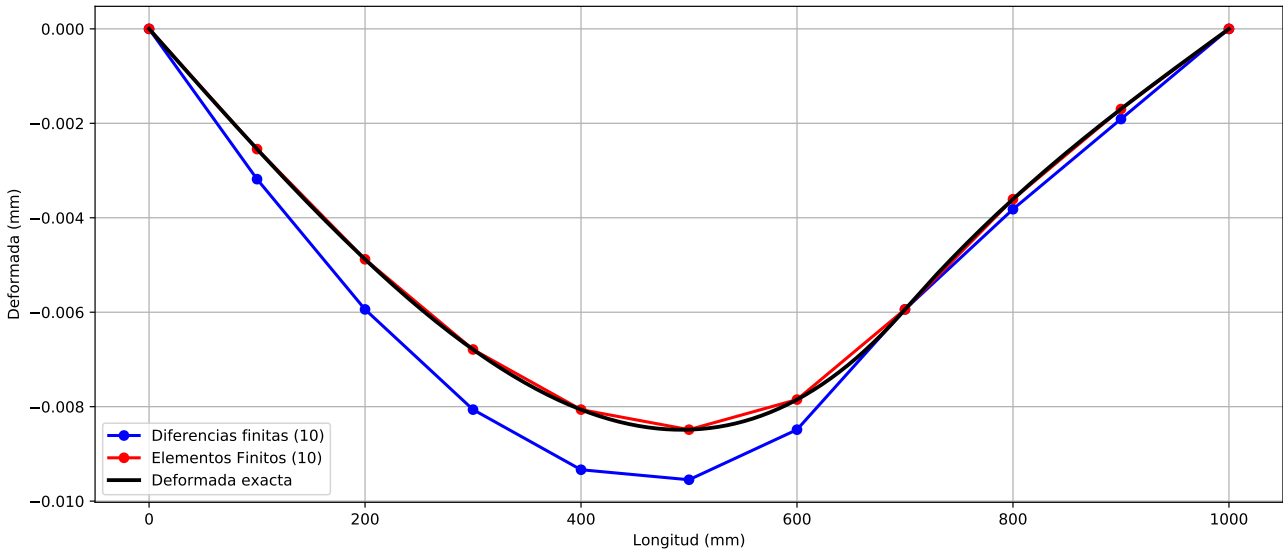


Figura 19: Comparación entre el método de diferencias finitas y elementos finitos

El método de elementos finitos con solo 10 elementos está más cerca a la deformada que el método de diferencias finitas con la misma cantidad; se observa que para obtener una convergencia similar sería

necesario usar alrededor de 1000 elementos en diferencias finitas, haciendo que el método de elementos finitos sea mucho más eficiente.

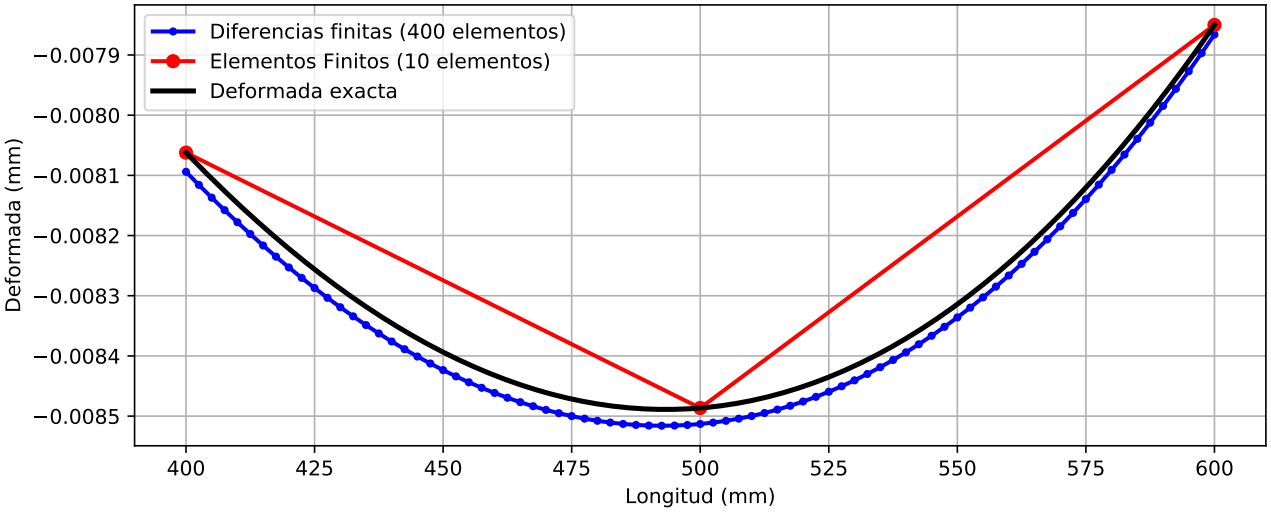


Figura 20: Comparación entre el método de diferencias finitas y elementos finitos

Elementos finitos y Diferencias finitas		
Características	**Diferencias finitas**	**Elementos finitos**
Tiempo de ejecución	$O(n)$	$O(n^2 \log(n))$
Uso de memoria	$O(1)$	$O(n^2)$
Convergencia	$O(n)$	$O(n^3)$

Cuadro 3: Comparación entre Elementos finitos y Diferencias finitas

Además, existen métodos numéricos [4] que pueden obtener una convergencia similar a elementos finitos con un tiempo de ejecución similar, dichos métodos son llamados multilinear-step y son usados como una generalización al método de Euler.

Conclusiones

1. Se logró verificar que el método de elementos finitos es capaz de dar resultados precisos aún con pocos elementos. Teniendo un error de menos del 0.05% con solo 10 elementos.

2. Tanto SolidsPy como Fusion 360, otorgan resultados similares aún cuando su mallado fue cuadrático y tetraédrico respectivamente. Sin embargo, se espera que el elemento viga tenga una mayor convergencia y rapidez en los cálculos.

3. Debido a que la longitud de la viga es relativamente grande, para obtener mayor precisión de la posición de la flecha máxima en la viga es necesario hacer un refinamiento de mallado, previa obtención de los resultados con menos elementos.

4. Logra demostrarse de forma efectiva que el método de elementos finitos tiene una convergencia superior al método de Euler de primer orden.

5. Se observa que, aunque el método de elementos finitos tiene una complejidad en ejecución muy alta, es posible optimizar el método con estructuras de datos más sofisticadas [3] reduciendo el tiempo de ejecución a la quinta parte que en una implementación común.

6. El tiempo de ejecución en un lenguaje de bajo nivel como C++, es alta debido a que el método del gradiente conjugado utilizó varios pasos hasta converger. Sin embargo, sigue siendo aún más rápido que Python o MATLAB.

Referencias

[1] J. Gómez and N. Guarín-Zapata, "Solidspy: 2d-finite element analysis with python," 2018. [Online]. Available: https://github.com/AppliedMechanics-EAFIT/SolidsPy

[2] J. Huaroto, "Análisis de una viga por diferencias finitas," 2019. [Online]. Available: https://github.com/HeNeos/Mechanical/blob/master/Beam_Deformation.py

[3] P. Kessler, "Sparse matrix library," 2020. [Online]. Available: https://github.com/uestla/Sparse-Matrix

[4] I. Gohberg and V. Olshevsky, "Fast algorithms with preprocessing for matrix-vector multiplication problems," *Journal of Complexity*, vol. 10, no. 4, pp. 411–427, 1994.