UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Mecánica

DESARROLLO DE UN SOFTWARE LIBRE EDUCACIONAL PARA ANÁLISIS DE VIGAS CONTINUAS

Autor:

Cuadrado Matas, Bruno

Tutor:

Del Caño Sánchez, Juan Carlos

Departamento de Construcciones Arquitectónicas, Ingeniería del Terreno y Mecánica de Medios Continuos y Teoría de estructuras

Valladolid, junio 2018.

Resumen

El presente trabajo de fin de grado se plantea como el desarrollo de un software para el cálculo y trazado de los diagramas de esfuerzos cortantes, momentos flectores, giros y desplazamientos de vigas rectas, tanto isostáticas como hiperestáticas en lenguaje de programación Python.

Por otro lado, el trabajo de fin de grado no ha consistido solamente en el desarrollo del software, sino que también ha existido un trabajo previo a su concepción consistiendo en el aprendizaje y entendimiento de un lenguaje que era desconocido.

Además, se ha desarrollado con la mentalidad de software libre y con un enfoque académico, pensando en su posible utilización como herramienta complementaria en cualquier asignatura relacionada con la resistencia de materiales, y con un énfasis particular en que se adapte bien a la "manera de hacer" que viene siendo común.

Etiquetas: Python – Software – Vigas – Diagramas – Código libre

Abstract

The present end-of-degree project it has been proposed like the development of a software for the calculate and drawn of diagrams of shear loads, bending momentum, gyrations and displacements of horizontal beams, both isostatic and hyperstatic in programing language Python.

On the other hand, end-of-degree project has not consisted only in the development of a software, but also has been a previous work of learning and compression of the language that it was unknown.

In addition, the software has developed with a mentality of open source and academic approach, thinking in a possible use like complementary tool in any subject related with material resistance and with a particular emphasis in the good way.

Tags: Python – Software – Beams – Diagrams – Open source

Contenido

I. Introducción

El presente trabajo de fin de grado se enmarca en la oferta realizada por el profesor D. Juan Carlos del Caño, profesor titular en el departamento de Construcciones Arquitectónicas, Ingeniería del Terreno y Mecánica de Medios Continuos y Teoría de estructuras de la Universidad de Valladolid.

Dicho trabajo se plantea como el desarrollo de un software para el cálculo y trazado de los diagramas de esfuerzos cortantes, momentos flectores, giros y desplazamientos de vigas rectas, tanto isostáticas como hiperestáticas en lenguaje de programación Python.

II. Objetivos

Tal como me han sido comunicados por parte de mi director en el presente TFG, los objetivos del mismo tienen una doble vertiente. Por una parte en cuanto a mi propia formación, que se resumen en:

1) Aprender un lenguaje de programación moderno y de amplia implantación (Python) así como adquirir algunas competencias asociadas, como son la programación orientada a objetos, la creación de interfaces gráficas, y la organización de un proyecto de software de cierta magnitud.

2) Manejar herramientas abiertas y libres, aprendiendo a apreciar su valor tanto en el aspecto práctico (ausencia de problemas en cuanto a licencias etc), como en el aspecto ético (concepto de trabajo colaborativo abierto y su contribución a la sociedad).

Por otra parte, se plantea un objetivo en cuanto al resultado del trabajo en sí mismo:

3) Que el software a realizar resulte adecuado para servir como complemento en el estudio de la asignatura Resistencia de Materiales que se imparte en segundo curso de los grados de ingenierías de la rama industrial que se imparten en esta escuela.

Por lo tanto, el trabajo de fin de grado no ha consistido solamente en el desarrollo del software, sino que también ha existido un trabajo previo a su concepción consistiendo en el aprendizaje y entendimiento de un lenguaje que me era desconocido. Los conocimientos previos que poseía para su desarrollo eran más bien básicos tales como los de control de flujo (condicionales, bucles…), declaración de variables y funciones. Todos estos conocimientos adquiridos en lenguaje C.

En concordancia con los objetivos planteados, este software, además, se ha desarrollado con la mentalidad de software libre y con un enfoque académico, pensando en su posible utilización como herramienta complementaria en cualquier asignatura relacionada con la resistencia de materiales, y con un énfasis particular en que se adapte bien a la "manera de hacer" que viene siendo común en la asignatura de referencia.

Por este último motivo, se ha tratado de desarrollar un software pensado en los usuarios finales, los estudiantes, quienes deberían ser capaces de utilizarlo sin invertir mucho tiempo en aprender a usarlo y sin miedo a que la aplicación devuelva respuestas extrañas. Todo ello a pesar de que el usuario típico sólo poseerá conocimientos muy básicos sobre la materia.

Incluso en aplicaciones no demasiado complejas como la presente, el conseguir que un programa sea robusto frente los errores de manejo de usuarios inexpertos siempre es un reto para el desarrollador, y conlleva un gran volumen de trabajo en cuanto a análisis de posibilidades y en cuanto a implementación. Este aspecto ha ocupado efectivamente un porcentaje inesperadamente alto del tiempo de desarrollo.

III. Método de cálculo

Barajando los métodos conocidos para la resolución de los posibles casos (integración directa, principio de fuerzas virtuales, principio de desplazamientos virtuales…) el método elegido ha sido el método directo de rigidez (MDR).

Esta elección es debida a la mayor facilidad de automatizar las distintas posibilidades de continuidad y compatibilidad que se pueden dar en los ejercicios. A continuación se explica de una forma somera el MDR para el caso de dos dimensiones.

Método Directo de Rigidez

El MDR relaciona los desplazamientos de unos puntos elegidos en la estructura con las fuerzas externas necesarias para originar dichos desplazamientos a través de una matriz llamada matriz de rigidez como vemos a continuación.

$$\bar{F} = \bar{\bar{K}} \cdot \bar{u} \quad (ec.1)$$

Donde:

F - es la matriz de fuerzas externas.

K - es la matriz de rigidez.

u - son los desplazamientos de los puntos elegidos.

La matriz de rigidez es la encargada de caracterizar los elementos en los que hemos dividido nuestra estructura, siendo esta k_l una matriz de rigidez local. Agrupando de la forma adecuada todas las k_l obtendremos la matriz de rigidez global o matriz de rigidez de la estructura k_g.

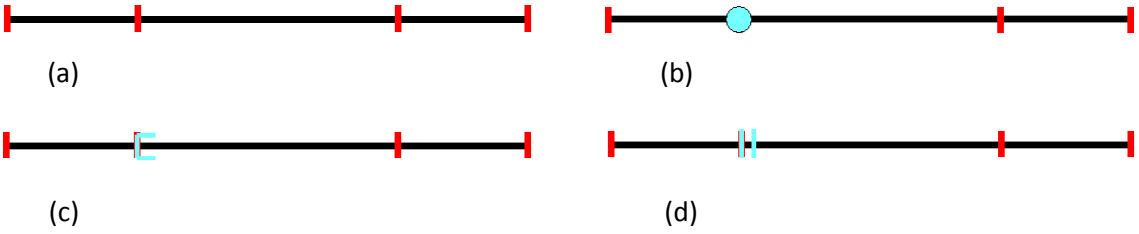
La k_l del elemento depende de las condiciones de compatibilidad entre elementos que no son las condiciones de contorno externas al sistema. Así tenemos 4 tipos de elementos con 4 k_l diferentes. Todas ellas las podemos ver a continuación con sus matrices (cada elemento se representa entre segmentos rojos):

(a) Biempotrada: en este caso no tenemos ningún tipo de discontinuidad entre los elementos.

(b) Giro-empotrada: en este caso tenemos que el elemento tiene una discontinuidad de tipo rótula con el elemento que tiene a su izquierda.

(c) X-empotrada: en este caso tenemos que nuestro elemento permite movimiento relativo entre las barras en el eje X.

(d) Y-empotrada: de forma similar al caso anterior en este caso tenemos que nuestro elemento permite movimiento relativo entre las barras en el eje Y.



$$(a)\begin{pmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{pmatrix}$$

$$(b)\begin{pmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & \frac{3EI}{L^3} & 0 & 0 & -\frac{3EI}{L^3} & \frac{3EI}{L^2} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & -\frac{3EI}{L^3} & 0 & 0 & \frac{3EI}{L^3} & -\frac{3EI}{L^2} \\ 0 & \frac{3EI}{L^2} & 0 & 0 & -\frac{3EI}{L^2} & \frac{3EI}{L} \end{pmatrix}$$

$$(c)\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{12EI}{L^3} & \frac{6EI}{L^2} & 0 & -\frac{12EI}{L^3} & \frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{4EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{2EI}{L} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{12EI}{L^3} & -\frac{6EI}{L^2} & 0 & \frac{12EI}{L^3} & -\frac{6EI}{L^2} \\ 0 & \frac{6EI}{L^2} & \frac{2EI}{L} & 0 & -\frac{6EI}{L^2} & \frac{4EI}{L} \end{pmatrix}$$

$$(d)\begin{pmatrix} \frac{EA}{L} & 0 & 0 & -\frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{EI}{L} & 0 & 0 & -\frac{EI}{L} \\ -\frac{EA}{L} & 0 & 0 & \frac{EA}{L} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{EI}{L} & 0 & 0 & \frac{EI}{L} \end{pmatrix}$$

La estructura de estas matrices es:

$$\begin{pmatrix} \bar{\bar{k}}_{aa} & \bar{\bar{k}}_{ab} \\ \bar{\bar{k}}_{ba} & \bar{\bar{k}}_{bb} \end{pmatrix}$$

Donde los subíndices a y b indican los nudos en los extremos de los elementos, y la estructura de la matriz es:

$$\begin{pmatrix} \bar{\bar{k}}_{aa} & \cdots & \bar{\bar{k}}_{an} \\ \vdots & \ddots & \vdots \\ \bar{\bar{k}}_{na} & \cdots & \bar{\bar{k}}_{nn} \end{pmatrix}$$

Y para montar esta matriz se aplica un algoritmo de acoplamiento donde se tiene en cuenta las uniones de los elementos.

Pero antes de montar la matriz de rigidez global tenemos que ver en función a qué sistema de coordenadas ya que, de forma local, o lo que es lo mismo, para cada elemento, puede que nos interese tomar sistemas distintos para simplificar determinados cálculos. Pero para unir todos los elementos debemos referenciarlo todo a un sistema de coordenadas común, global. Para ello se elige un sistema que, observando la estructura entera, nos pueda ayudar con los cálculos. Para los sistemas locales pondremos el subíndice l y para los globales el subíndice g.

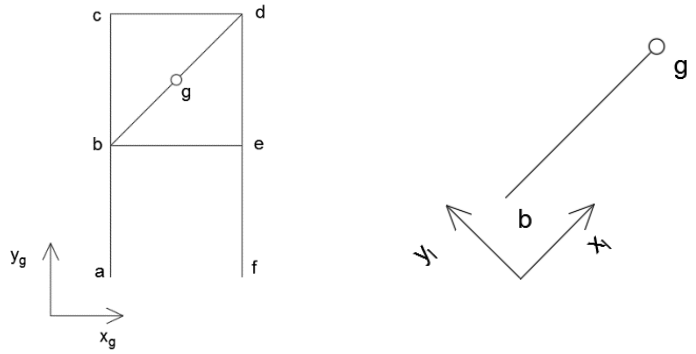


Figura 1

En la figura 1 podemos ver esta diferencia de sistemas de coordenadas. Observamos cómo el sistema de referencia de la barra b-g tiene orientado el eje x en la misma dirección que ella y el eje y de forma perpendicular. Esto nos obliga a tener que orientar todas nuestras matrices de rigidez locales respecto un eje que, por lo general, será el eje global. Esto lo hacemos con una matriz de cambio de base, llamada matriz L. Esta matriz L nos ayuda a orientar un extremo del elemento. Para orientar el elemento entero tenemos la matriz T, que es una matriz de 6x6, que nos permite orientar al elemento completo. Estas matrices son las que vemos en las ec. 2 y ec. 3 respectivamente:

$$\bar{\bar{L}} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (ec.2)$$

$$\bar{\bar{T}} = \begin{pmatrix} \bar{\bar{L}} & 0 \\ 0 & \bar{\bar{L}} \end{pmatrix} \quad (ec.3)$$

De esta forma, la relación entre los desplazamientos de los puntos, o nudos, elegidos y las fuerzas externas que se necesitan para provocar dichos desplazamientos queda:

$$\bar{F}_l = \bar{\bar{k}}_l \cdot \bar{u}_l \quad (ec.4)$$

$$\bar{F}_l = \bar{\bar{T}} \cdot \bar{F}_g \quad (ec.5)$$

$$\bar{u}_l = \bar{\bar{T}} \cdot \bar{u}_g \quad (ec.6)$$

$$\bar{\bar{T}}^{-1} \cdot \bar{\bar{T}} \cdot \bar{\bar{F}}_g = \bar{\bar{T}}^{-1} \cdot \bar{\bar{k}}_l \cdot \bar{\bar{T}} \cdot \bar{\bar{u}}_g \quad (ec.7)$$

$$\bar{\bar{F}}_g = \bar{\bar{T}}^T \cdot \bar{\bar{k}}_l \cdot \bar{\bar{T}} \cdot \bar{\bar{u}}_g \quad (ec.8)$$

$$\bar{F}_g = \bar{\bar{K}}_g \cdot \bar{u}_g \quad (ec.9)$$

$$\bar{\bar{K}}_g = \bar{\bar{T}}^T \cdot \bar{\bar{k}}_l \cdot \bar{\bar{T}} \quad (ec.10)$$

De acuerdo con este desarrollo, vemos claramente cómo obtener la matriz K_g a partir de la k_l y cómo plantear el sistema en coordenadas globales a resolver.

Hasta aquí hemos visto la diferencia entre coordenadas locales y globales, la forma de obtener la matriz de rigidez de la estructura y el sistema de ecuaciones a resolver. Ahora vamos a ver qué hacemos con los desplazamientos y por último, veremos la manipulación de las fuerzas para poder formar el sistema de ecuaciones de la estructura.

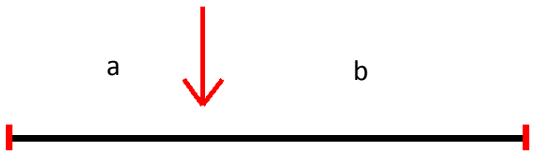
Para los desplazamientos usamos una matriz columna donde se colocan los desplazamientos de cada nudo. Dependiendo si es conocido o no se pondrá el valor del desplazamiento o la incógnita para su cálculo. Hay que tener presente que cumpliendo con las hipótesis de cálculo de elasticidad y resistencia de materiales, si tenemos el valor del desplazamiento, no sabemos el valor de la acción en dicho nudo y viceversa. La estructura de la matriz columna será:

$$\bar{\bar{u}} = \begin{pmatrix} \bar{u}_a \\ \vdots \\ \bar{u}_n \end{pmatrix}$$

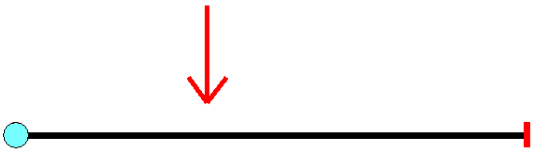
Donde cada $\bar{u}_x$ es el vector de desplazamientos del nudo x.

La matriz de fuerzas es una matriz columna como la matriz de desplazamientos. Se compone de la suma de la matriz de fuerzas estáticas, $\bar{\bar{F}}_{est}$, que son las fuerzas externas aplicadas en los nudos y, que al igual que en la matriz de desplazamientos, dependerá de si conocemos las fuerzas o, si por el contrario, no las sabemos y debemos poner las incógnitas para calcularlas, y la matriz de fuerzas equivalentes, $\bar{\bar{F}}_{eq}$. Esta última matriz se refiere a las acciones que serían necesarias aplicar en los extremos del elemento para que al aislarlo del resto de la estructura no sufriera cambios en sus esfuerzos internos. Estas fuerzas dependen de las condiciones de compatibilidad entre los elementos adyacentes y no de las condiciones de contorno de la estructura de forma análoga a las $\bar{\bar{k}}_l$. Además, también dependen de la acción a la que está sometido el elemento o suma de acciones, ya que se aplica el principio de superposición. A continuación se explicarán cada posible situación:

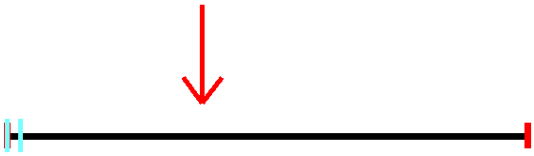
(a) Fuerza vertical puntual:



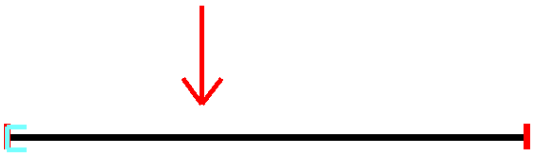
(a.1)



(a.2)



(a.3)



(a.4)

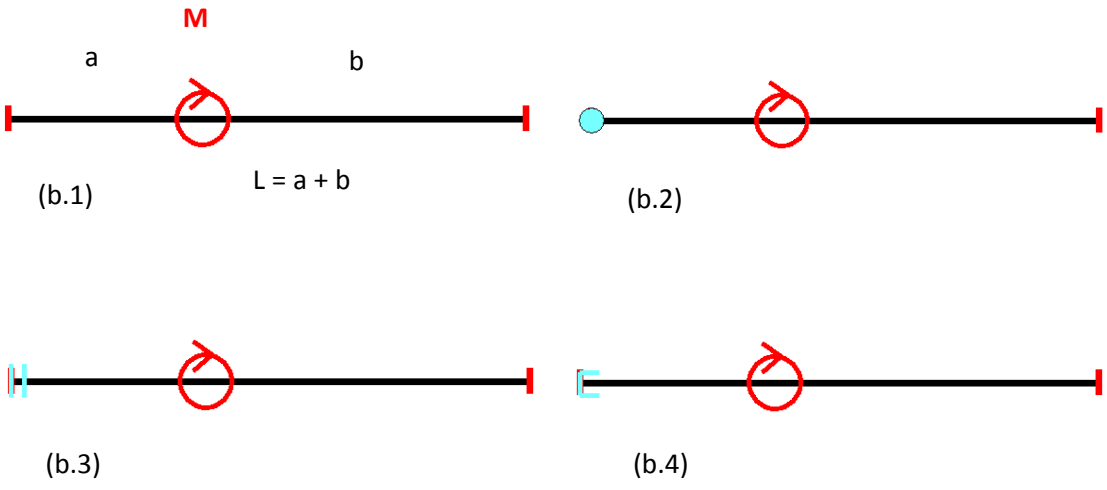
$$(a.1)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ \frac{Fb^2}{L^3}(L+2a) \\ \frac{Fab^2}{L^2} \\ 0 \\ \frac{Fa^2}{L^3}(L+2b) \\ -\frac{Fba^2}{L^2} \end{pmatrix}$$

$$(a.2)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ \frac{Fb^2}{L^3}(L+2a) - \frac{Fb^2}{L^3}(L+2a)\frac{3}{2L} \\ 0 \\ 0 \\ \frac{Fa^2}{L^3}(L+2b) + \frac{Fb^2}{L^3}(L+2a)\frac{3}{2L} \\ -\frac{Fba^2}{L^2} - \frac{Fab^2}{L^2}\frac{1}{2} \end{pmatrix}$$

$$(a.3)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ 0 \\ \frac{Fab^2}{L^2} - \frac{Fb^2}{L^3}(L+2a)\frac{L}{2} \\ 0 \\ \frac{Fa^2}{L^3}(L+2b) + \frac{Fb^2}{L^3}(L+2a) \\ -\frac{Fba^2}{L^2} - \frac{Fb^2}{L^3}(L+2a)\frac{L}{2} \end{pmatrix}$$

$$(a.4)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ -\frac{Fb^2}{L^3}(L+2a) \\ -\frac{Fab^2}{L^2} \\ 0 \\ \frac{Fa^2}{L^3}(L+2b) \\ -\frac{Fba^2}{L^2} \end{pmatrix}$$

(b) Momento puntual:



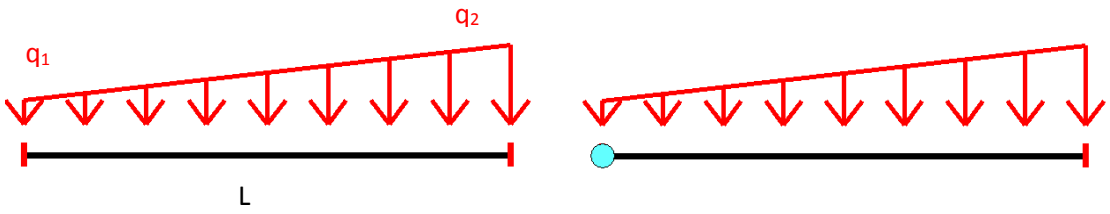
$$(b.1)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ -\frac{6Mab}{L^3} \\ -\frac{Mb}{L}(2-3\frac{b}{L}) \\ 0 \\ \frac{6Mab}{L^3} \\ -\frac{Ma}{L}(2-3\frac{a}{L}) \end{pmatrix}$$

$$(b.2)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ -\frac{6Mab}{L^3} - \frac{Mb}{L}(2-3\frac{b}{L})\frac{3}{2L} \\ 0 \\ 0 \\ \frac{6Mab}{L^3} + \frac{Mb}{L}(2-3\frac{b}{L})\frac{3}{2L} \\ -\frac{Ma}{L}\left(2-3\frac{a}{L}\right) - \frac{Mb}{L}(2-3\frac{b}{L})\frac{1}{2} \end{pmatrix}$$

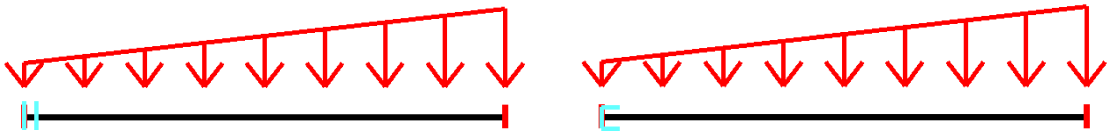
$$(b.3)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ 0 \\ -\frac{Mb}{L}\left(2-3\frac{b}{L}\right) - \frac{6Mab}{L^3}\frac{L}{2} \\ 0 \\ \frac{6Mab}{L^3} + \frac{6Mab}{L^3} \\ -\frac{Ma}{L}\left(2-3\frac{a}{L}\right) - \frac{6Mab}{L^3}\frac{L}{2} \end{pmatrix}$$

$$(b.4)\quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ -\frac{6Mab}{L^3} \\ -\frac{Mb}{L}(2-3\frac{b}{L}) \\ 0 \\ \frac{6Mab}{L^3} \\ -\frac{Ma}{L}(2-3\frac{a}{L}) \end{pmatrix}$$

(c) Fuerza vertical distribuida:



(c.1) (c.2)



(c.3) (c.4)

Por motivos de legibilidad, para las siguientes matrices utilizaremos otra notación:

$$R_i = \frac{L}{6}(2q_1 + q_2) + \frac{M_i - M_j}{L}$$

$$R_j = \frac{L}{6}(q_1 + 2q_2) - \frac{M_i - M_j}{L}$$

$$M_i = \left(\frac{L^2}{60}(3q_1 + 2q_2)\right)$$

$$M_j = \left(\frac{L^2}{60}(2q_1 + 3q_2)\right)$$

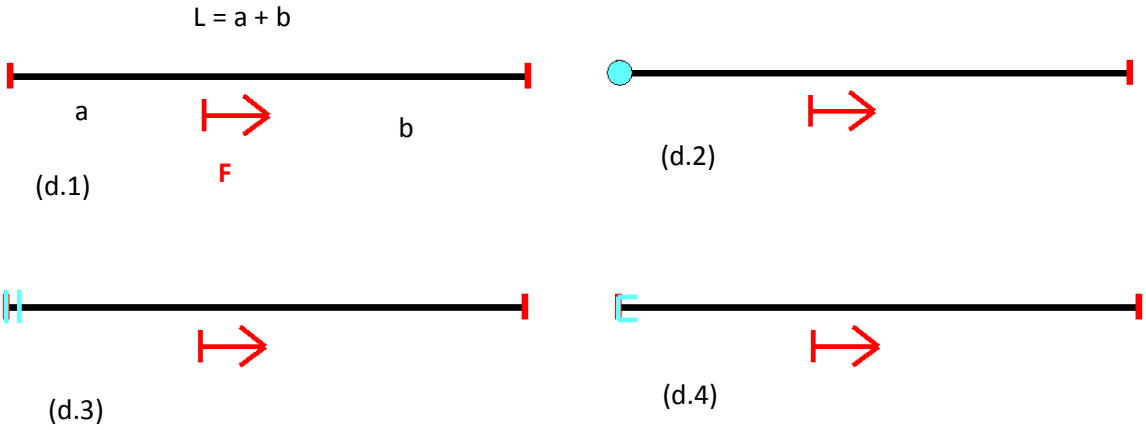
$$(c.1) \quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ R_i \\ M_i \\ 0 \\ R_j \\ -M_j \end{pmatrix}$$

$$(c.2) \quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ R_i - \frac{3M_i}{2L} \\ 0 \\ 0 \\ R_i + \frac{3M_i}{2L} \\ -M_j - \frac{M_i}{2} \end{pmatrix}$$

$$(c.3) \quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ 0 \\ M_i - \frac{R_i L}{2} \\ 0 \\ R_i + R_j \\ -M_j - \frac{R_i L}{2} \end{pmatrix}$$

$$(c.4) \quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ R_i \\ M_i \\ 0 \\ R_j \\ -M_j \end{pmatrix}$$

(d) Fuerza horizontal puntual:



$$(d.1)(d.2)(d.3) \quad \bar{\bar{F}}_{emp} = \begin{pmatrix} -\frac{b}{L}F \\ 0 \\ 0 \\ -\frac{a}{L}F \\ 0 \\ 0 \end{pmatrix}$$

$$(d.3) \quad \bar{\bar{F}}_{emp} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ -F \\ 0 \\ 0 \end{pmatrix}$$

Seguramente nos hemos percatado que las matrices expuestas para cada situación no son las $\bar{\bar{F}}_{equ}$ como se había prometido, sino que son $\bar{\bar{F}}_{emp}$. El paso de las segundas a las primeras es muy fácil y en nuestro caso más aún. Simplemente debemos hacer lo siguiente:

$$\bar{\bar{F}}_{equ} = -\bar{\bar{T}}^t \bar{\bar{F}}_{emp}$$

En nuestros casos, siempre vamos a tener elementos horizontales, por lo que nos queda:

$$\bar{\bar{F}}_{equ} = -\bar{\bar{F}}_{emp}$$

Solo nos queda saber cómo actuar cuando se den 2 condiciones más. 1 – Si tenemos un muelle en el nudo y 2 – si el nudo esta girado respecto el sistema de coordenadas global.

En la figura 2 vemos la representación de un muelle en el programa.

Figura 2

Los muelles generan una matriz como la siguiente:

$$\bar{\bar{k}}_{muelle} = \begin{pmatrix} k_x & 0 & 0 \\ 0 & k_y & 0 \\ 0 & 0 & k_\theta \end{pmatrix}$$

El punto donde esté el muelle siempre será un nudo. No importa si se encuentra en el extremo de un elemento o en su interior ya que si nos ocurre esto, automáticamente, en ese punto aparecerá un nudo. Esta explicación se debe a que, en la matriz de rigidez de la estructura, se sumará esta matriz del muelle en la zona de la diagonal principal que pertenezca a dicho nudo:

$$\begin{pmatrix} \bar{\bar{k}}_{aa} & \bar{\bar{k}}_{ab} & \bar{\bar{k}}_{ac} \\ \bar{\bar{k}}_{ba} & \bar{\bar{k}}_{bb} & \bar{\bar{k}}_{bc} \\ \bar{\bar{k}}_{ca} & \bar{\bar{k}}_{cb} & \bar{\bar{k}}_{cc} + \bar{\bar{k}}_{muelle} \end{pmatrix}$$

Por último, en el caso de que un apoyo de nuestra estructura este girado, debemos multiplicar a todos los elementos implicados del sistema por la matriz de giro $\bar{\bar{L}}$. De forma genérica tenemos:

$$\begin{pmatrix} \bar{\bar{F}}_{na} \\ \vdots \\ \bar{\bar{F}}_{nj} \\ \vdots \\ \bar{\bar{F}}_{nz} \end{pmatrix} \begin{pmatrix} \bar{\bar{L}}_a \bar{\bar{F}}_{equ,a} \\ \vdots \\ \bar{\bar{L}}_j \bar{\bar{F}}_{equ,j} \\ \vdots \\ \bar{\bar{L}}_z \bar{\bar{F}}_{equ,z} \end{pmatrix} \begin{pmatrix} \bar{\bar{L}}_a \bar{\bar{k}}_{aa} \bar{\bar{L}}_a^T & \cdots & \bar{\bar{L}}_a \bar{\bar{k}}_{aj} \bar{\bar{L}}_j^T & \cdots & \bar{\bar{L}}_a \bar{\bar{k}}_{az} \bar{\bar{L}}_z^T \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \bar{\bar{L}}_j \bar{\bar{k}}_{ja} \bar{\bar{L}}_a^T & \cdots & \bar{\bar{L}}_j \bar{\bar{k}}_{jj} \bar{\bar{L}}_j^T & \cdots & \bar{\bar{L}}_j \bar{\bar{k}}_{jz} \bar{\bar{L}}_z^T \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \bar{\bar{L}}_z \bar{\bar{k}}_{za} \bar{\bar{L}}_a^T & \cdots & \bar{\bar{L}}_z \bar{\bar{k}}_{zj} \bar{\bar{L}}_j^T & \cdots & \bar{\bar{L}}_z \bar{\bar{k}}_{zz} \bar{\bar{L}}_z^T \end{pmatrix} \begin{pmatrix} \bar{\bar{u}}_{nodal,a} \\ \vdots \\ \bar{\bar{u}}_{nodal,j} \\ \vdots \\ \bar{\bar{u}}_{nodal,z} \end{pmatrix}$$

Pues con esto daremos por explicado el MDR. Repetimos que ha sido una explicación muy rápida y somera, pero que nos ayuda a futura explicación de nuestro software. De aquí en adelante se explicará la estructura de archivos que sigue nuestro programa, la estructura interna de cada archivo y su funcionamiento.

IV. Estructura del software

El software se compone de 14 archivos de texto divididos en familias dependiendo de la función que desempeñan:

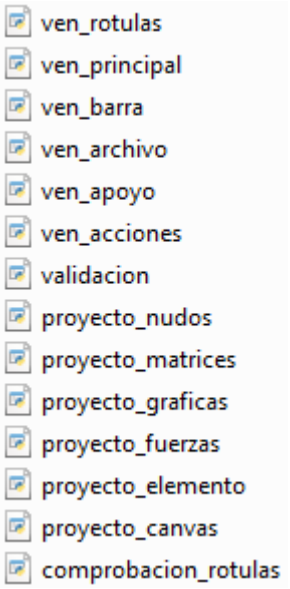


Figura 3

De todos estos archivos, el que inicia el programa es el llamado ven_principal. Este es el archivo principal donde se encuentra la función __main__ y desde el que se crea la ventana principal del programa. Desde este, se llama a todos los archivos tipo ven_xxx que son los que inician las ventanas que permiten introducir los datos de los elementos del problema (más adelante les veremos un poco más en detalle).

Dentro de estos, llamamos al archivo de validación para la comprobación del tipo de datos introducidos y asegurarse de que son números.

Una vez comprobado que los datos introducidos son números, y antes de iniciar el cálculo, se hace un repaso por todas las discontinuidades para ver si existe alguna acción en el mismo punto creando una situación "imposible". Una posible situación de este tipo es si tenemos en el mismo punto una discontinuidad tipo rótula y un momento aplicado. En estos casos aparecerá un menú en el que nos permitirá elegir a qué lado de la discontinuidad se aplicará la acción.

Una vez tenemos los datos verificados estamos preparados para montar el sistema de ecuaciones para resolver el problema. Desde el archivo de comprobacion_rotulas iremos llamando a las funciones de proyecto_matrices para ir formando las matrices $\bar{\bar{F}}_{emp}, \bar{\bar{F}}_{equ}, \bar{\bar{u}}_l, \bar{\bar{k}}_l, \bar{\bar{k}}_{est}$ y demás matrices auxiliares.

Una vez montado el sistema, localizaremos las incógnitas y se intentará resolver el sistema. En el caso de ser un mecanismo lo introducido, no se podrá

resolver el sistema por ser incompatible y saldrá en pantalla un mensaje indicando que no puede resolverse el sistema.

En el caso que estén bien introducidos tanto los datos del problema, como las condiciones de contorno y compatibilidad, se resolverá el sistema y aparecerán en pantalla las gráficas de esfuerzos cortantes, momentos flectores, giros y desplazamientos de la estructura.

La ventana de los resultados es interactuable y se podrá navegar pos las gráficas para encontrar los valores que interesen e incluso guardarlas como imagen.

En la figura 4 podemos ver un esquema de la manera en la que se llaman los ficheros unos a otros.

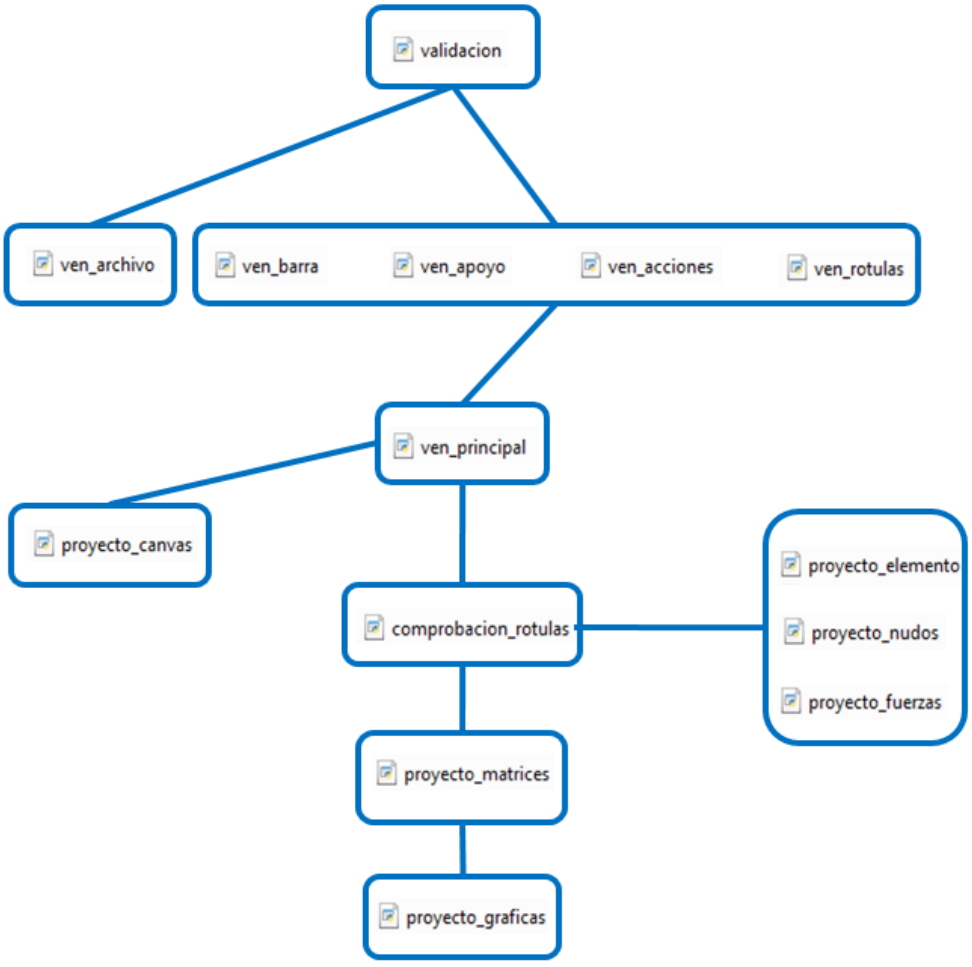


Figura 4

A continuación se explicará la estructura interna de cada tipo de ficheros y se verán los elementos de los que está compuesto cada uno.

V. Estructura de los archivos

La estructura interna de cada archivo viene determinada por la función que desempeña. De esta forma, tenemos una familia de archivos en especial que tienen la misma estructura, que es la de la creación de las ventanas del programa.

El resto de archivos tienen particularidades propias ya que algunos solo son auxiliares para interacción con la persona que esté trabajando con él, como son el archivo de validación de datos o de dibujo de dichos datos, y otros están enfocados a la resolución del problema puramente y la muestra de resultados.

Ventana principal

El archivo ven_principal, como se ha dicho antes, es el archivo que inicia el programa. A través de él se lanza la función main() del programa que llama a la primera ventana que se ve al iniciarlo (figura 5).

```
def main():


    vent_principal()

    return 0

if __name__ == '__main__':
    main()
```

Figura 5

En el encabezado se importan todas las librerías y archivos a los que realizará llamadas (figura 6).

```
from tkinter import *
from tkinter import ttk
from ven_barra import *
from ven_apoyo import *
from ven_acciones import *
from ven_rotulas import *
from proyecto_nudos import *
from proyecto_elemento impor
from proyecto_matrices impor
from proyecto_fuerzas import
from comprobacion_rotulas im
from proyecto_canvas import
from validacion import *
from ven_archivo import *
```

Figura 6

La función main() llama a un objeto llamado vent_principal en el cual se definen una serie de listas que se utilizarán para almacenar tanto los datos introducidos por el usuario, como los datos manipulados para que se puedan realizar los cálculos además de otros objetos destinados a dibujar los elementos introducidos y relativos al guardado del trabajo realizado (figura 7).

```
class vent_principal:

    def __init__(self):
        #####Inicialización de las listas de datos princples
        self.lista_barras = []
        self.lista_apoyos = []
        self.lista_acciones = []
        self.lista_acciones_aux = []
        self.lista_rotulas = []
        self.lista_n = lista_nudos()
        self.lista_e = lista_elementos()
        self.matriz = matriz(self.lista_e,self.lista_n,self.lista_acciones_aux)
        self.filename = [""]
        self.dibujo = Ven_Grafica()
```

Figura 7

A continuación se declaran las variables necesarias para la creación de los elementos necesarios para la ventana gráfica donde trabajará el usuario. También hay una función bucle, la cual capta continuamente el tamaño de la ventana para redimensionar la zona de dibujo de los elementos (figura 8).

```
self.raiz = Tk()
self.raiz.title("Ventana Principal")
self.raiz.geometry("700x300+300+250")
self.raiz.minsize(700,300)

self.canvas = Canvas(self.raiz)
self.canvas.pack(fill=BOTH,expand=True)

def motion(event):
    x, y = event.width, event.height
    print('{}, {}'.format(x, y))
    self.dibujo.escala(x,y)
    self.dibujo.dibuja_linea(self.canvas,self.dibujo)

self.canvas.bind('<Configure>', motion)

self.frame_fondo = ttk.Frame(self.raiz)
self.frame_fondo.pack()
```

Figura 8

A partir de aquí se crea el menú principal del programa. Gracias a él se podrá tanto introducir los datos del programa como abrir, guardar o iniciar un proyecto nuevo (figura 9).

```
self.menu_desplegable.add_cascade(label="Archivo", menu=self.menu_archivo)

#Fin menu archivo

########## Inicio menu barras ##########

self.menu_barra= Menu(self.menu_desplegable, tearoff=0)
self.menu_barra.add_command(label="Nueva", command=self.nueva_barra)
self.menu_barra.add_command(label="Modificar", command=self.modificar_barra)
self.menu_barra.add_command(label="Eliminar", command=self.eliminar_barra)

self.menu_desplegable.add_cascade(label="Barras", menu=self.menu_barra)

#Fin menu barras

########## Inicio menu apoyos ##########

self.menu_apoyo=Menu(self.menu_desplegable, tearoff=0)
self.menu_apoyo.add_command(label="Nuevo", command=self.nuevo_apoyo)
self.menu_apoyo.add_command(label="Modificar", command=self.modificar_apoyo)
self.menu_apoyo.add_command(label="Eliminar", command=self.eliminar_apoyo)

self.menu_desplegable.add_cascade(label="Apoyos", menu=self.menu_apoyo)

#Fin menu apoyos

########## Inicio menu acciones ##########

self.menu_acciones = Menu(self.menu_desplegable, tearoff=0)
self.menu_acciones.add_command(label="Nueva", command=self.nueva_fuerza)
self.menu_acciones.add_command(label="Modificar", command=self.modificar_fuerza)
self.menu_acciones.add_command(label="Eliminar", command=self.eliminar_fuerza)

self.menu_desplegable.add_cascade(label="Acciones", menu=self.menu_acciones)

#Fin menu acciones

########## Inicio menu discontinuidades ##########

self.menu_rotulas = Menu(self.menu_desplegable, tearoff=0)
self.menu_rotulas.add_command(label="Nueva", command=self.nueva_rotula)
self.menu_rotulas.add_command(label="Modificar", command=self.modificar_rotula)
self.menu_rotulas.add_command(label="Eliminar", command=self.eliminar_rotula)

self.menu_desplegable.add_cascade(label="Discont.", menu=self.menu_rotulas)

#Fin menu rotulas

########## Inicio calcular ##########

self.menu_calcular = Menu(self.menu_desplegable, tearoff=0)
self.menu_calcular.add_command(label="Calcular", command=self.calcular)

self.menu_desplegable.add_cascade(label="Calcular", menu=self.menu_calcular)

#Fin menu calcular

self.raiz.config(menu=self.menu_desplegable)
```

Figura 9

Desde este menú se llaman a las funciones de los demás archivos de ventana para los casos de introducción de nuevos elementos, modificarlos o

eliminarlos. También para las opciones de guardado, apertura, nuevo proyecto y cálculo del problema (figura 10, figura 11 y figura 12).

```
def nueva_barra(self):
    ven_nue_barra(self.raiz, self.lista_barras,self.canvas,self.dibujo)
def modificar_barra(self):
    ven_mod_barra(self.raiz, self.lista_barras,self.canvas,self.dibujo)
def eliminar_barra(self):
    ven_eli_barra(self.raiz, self.lista_barras,self.canvas,self.dibujo)
def nuevo_apoyo(self):
    ven_nue_apoyo(self.raiz, self.lista_apoyos,self.canvas,self.dibujo)
def modificar_apoyo(self):
    ven_mod_apoyo(self.raiz, self.lista_apoyos,self.canvas,self.dibujo)
def eliminar_apoyo(self):
    ven_eli_apoyo(self.raiz, self.lista_apoyos,self.canvas,self.dibujo)
def nueva_fuerza(self):
    ven_nue_fuerza(self.raiz, self.lista_acciones,self.lista_rotulas,self.canvas,self.dibujo)
def modificar_fuerza(self):
    ven_mod_fuerza(self.raiz, self.lista_acciones,self.lista_rotulas,self.canvas,self.dibujo)
def eliminar_fuerza(self):
    ven_eli_fuerza(self.raiz, self.lista_acciones,self.canvas,self.dibujo)
def nueva_rotula(self):
    ven_nue_rotula(self.raiz, self.lista_rotulas,self.canvas,self.dibujo)
def modificar_rotula(self):
    ven_mod_rotula(self.raiz, self.lista_rotulas,self.canvas,self.dibujo)
def eliminar_rotula(self):
    ven_eli_rotula(self.raiz, self.lista_rotulas,self.canvas,self.dibujo)
```

Figura 10

```
    def nuevo (self):

        if len(self.lista_barras)>0 or len(self.lista_apoyos)>0 or len(self.lista_acciones)>0 or len(self.lista_rotulas)>0:

            proyecto_nuevo(self.raiz, self.filename, self.canvas, self.dibujo, self.lista_barras, self.lista_apoyos,
                           self.lista_acciones, self.lista_rotulas, flag=False)

    def abrir (self):

        if len(self.lista_barras)>0 or len(self.lista_apoyos)>0 or len(self.lista_acciones)>0 or len(self.lista_rotulas)>0:

            proyecto_nuevo(self.raiz, self.filename, self.canvas, self.dibujo, self.lista_barras, self.lista_apoyos,
                           self.lista_acciones, self.lista_rotulas, flag=True)

        else:

            self.filename[0] = abrir_archivo(self.filename, self.canvas, self.dibujo, self.lista_barras, self.lista_apoyos,
                                             self.lista_acciones, self.lista_rotulas)

    def guardar_as (self):

        self.filename[0] = guardar_como(self.filename, self.lista_barras, self.lista_apoyos, self.lista_acciones, self.lista_rotulas)

    def guardar_solo (self):

        guardar(self.filename, self.lista_barras, self.lista_apoyos, self.lista_acciones, self.lista_rotulas)
```

Figura 11

```
def calcular (self):

    longitud = 0
    rango = True

    if len(self.lista_barras)>0:

        for i in self.lista_barras:

            longitud = longitud + float(i.lon)

        for j in self.lista_apoyos:

            if float(j.posx)<0 or float(j.posx)>longitud:

                rango = False

        for k in self.lista_acciones:

            if float(k.datos[1]) > longitud or float(k.datos[1]) < 0 or float(k.datos[2]) > longitud or float(k.datos[2]) < 0:

                rango = False

        for l in self.lista_rotulas:

            if float(l.posicion) >= longitud or float(l.posicion) <= 0:

                rango = False


        if rango==True:

            self.lista_n.anadir_nudo (self.lista_barras, self.lista_apoyos, self.lista_acciones, self.lista_rotulas)

            self.lista_e.anadir_elementos(self.lista_barras, self.lista_n)

            anadir_fuerzas(self.lista_n, self.lista_e,self.lista_acciones,self.lista_acciones_aux)

            comp_rotulas(self.raiz,self.lista_rotulas,self.lista_acciones_aux,self.matriz,self.lista_e,self.lista_n)

        else:

            fuera_rango(self.raiz)

    else:

        ven_no_barras(self.raiz)
```

Figura 12

Ventanas elementos

En este caso, con elementos no se está haciendo referencia únicamente a la definición que hace el MDR, sino que se tomará como elemento también a los apoyos, discontinuidades y acciones. Al fin y al cabo son las unidades necesarias para el planteamiento del problema.

De esta forma, en este apartado tenemos una familia de archivos que todos ellos siguen la misma estructura interna por lo que solo se analizará uno de ellos.

El elegido es el archivo llamado ven_barra. La diferencia entre unos y otros solo radica en la cantidad de campos a rellenar y en la forma de rellenarlos (campo de texto, check button…).

Como en todos los archivos, lo primero son las librerías que utilizará más adelante (figura 13).

```
from tkinter import *
from tkinter import ttk
import ven_principal
from proyecto_canvas import *
from validacion import *
```

Figura 13

Lo que sigue es la declaración del objeto que se utilizará para almacenar los datos del usuario además del nombre del elemento introducido. Este nombre se lo da el programa automáticamente.

En el caso actual, el objeto se llama Barra y tiene como atributos nombre, longitud, área, módulo elástico e inercia (figura 14).

```
class Barra:

    ####Clase para definir la introducción de datos de las barras
    def __init__(self, nombre, lon, area, E, inercia):
        self.nombre = nombre
        self.lon = lon
        self.area = area
        self.E = E
        self.inercia = inercia
```

Figura 14

A partir de aquí se definen los objetos ventana que aparecerán para la introducción de los elementos. Primero se define la ventana del nuevo elemento, después la de su modificación y por último la de su eliminación. Como estas ventanas dependen de la principal, debemos llevarnos las variables que necesitaremos como son las del campo de dibujo (canvas), la lista para almacenar los datos (lista) y el propio dibujo de los elementos (dibujo) (figura 14).

```
class ven_nue_barra:

    def __init__(self,raiz,lista,canvas,dibujo):


        self.canvas = canvas
        self.lista = lista
        self.dibujo = dibujo
        self.nueva_barra = Toplevel(raiz)
        self.nueva_barra.title("Nueva Barra")
        self.nueva_barra.geometry("300x150+550+200")
        self.nueva_barra.grab_set()

        self.frame_nueva_barra = ttk.Frame(self.nueva_barra)
        self.frame_nueva_barra.pack()
```

Figura 14

A su vez, excepto en la opción de eliminar elemento, primero se definen las etiquetas para dar nombre a los campos de datos. Después los propios campos de datos y finalmente los botones de aceptación o cancelación de la introducción de datos (figura 15).

```
########## Etiquetas ##########

self.nombre_label = ttk.Label(self.frame_nueva_barra, text="Nombre")
self.nombre_label.grid(column=0, row=0)

if len(self.lista)==0:
    self.m=1

else:
    if len(self.lista[-1].nombre)<7:
        self.m=int(self.lista[-1].nombre[-1])+1

    else:
        self.m=int(self.lista[-1].nombre[-2]+self.lista[-1].nombre[-1])+1

self.nombre_etiqueta = ttk.Label(self.frame_nueva_barra, text="Barra "+str(self.m))
self.nombre_etiqueta.grid(column=1, row=0)

self.longitud_label = ttk.Label(self.frame_nueva_barra, text="Longitud")
self.longitud_label.grid(column=0, row=1)

self.inercia_label = ttk.Label(self.frame_nueva_barra, text="Inercia")
self.inercia_label.grid(column=0, row=2)

self.mod_elas_label = ttk.Label(self.frame_nueva_barra, text="Módulo Elástico")
self.mod_elas_label.grid(column=0, row=3)

self.area_label = ttk.Label(self.frame_nueva_barra, text="Área")
self.area_label.grid(column=0, row=4)

#Fin etiquetas

########## Introduccion de variables ##########

self.lon = StringVar()
self.inercia = StringVar()
self.mod_elas = StringVar()
self.area = StringVar()

self.longitud_entry = ttk.Entry(self.frame_nueva_barra,text=self.lon)
self.longitud_entry.grid(column=1, row=1)
self.longitud_entry.insert(0,8)

self.inercia_entry = ttk.Entry(self.frame_nueva_barra, text=self.inercia)
self.inercia_entry.grid(column=1, row=2)
self.inercia_entry.insert(0,0.0002)

self.mod_elas_entry = ttk.Entry(self.frame_nueva_barra, text=self.mod_elas)
self.mod_elas_entry.grid(column=1, row=3)
self.mod_elas_entry.insert(0,2.1e11)

self.area_entry = ttk.Entry(self.frame_nueva_barra, text=self.area)
self.area_entry.grid(column=1, row=4)
self.area_entry.insert(0,0.006)

#Fin introduccion de variables

########## Botones ##########

self.aceptar_button = ttk.Button(self.frame_nueva_barra,
                                 text="Aceptar", command=self.anadir)
self.aceptar_button.grid(column=0, row=5)

self.cancelar_button= ttk.Button(self.frame_nueva_barra,
                                 text="Salir", command=self.nueva_barra.destroy)
self.cancelar_button.grid(column=1, row=5)

#Fin botones
```

Figura 15

Cuando se aceptan los datos, primero se verifican para comprobar que el tipo de datos introducido es el correcto, y luego se almacena en la lista destinada a ello (figura 16).

```
def anadir(self):
    #Falta validación de datos
    ######################################################
    nombre = 'Barra ' + str(self.m)
    lon = self.lon.get()
    inercia = self.inercia.get()
    e = self.mod_elas.get()
    area = self.area.get()

    aux_barra=[]

    aux_barra.append(Validar(lon))
    aux_barra.append(Validar(inercia))
    aux_barra.append(Validar(e))
    aux_barra.append(Validar(area))

    if False in aux_barra:
        aux=False
        ven_no_num(self.nueva_barra)

    else:
        aux=True


    if aux==True:
        barra = Barra(nombre, lon, area, e, inercia)

        ########## Añade barra ##########
        self.lista.append(barra)

        self.m +=1
        self.nombre_etiqueta.configure(text='Barra ' + str(self.m))
        self.longitud_entry.delete(0,END)
        self.inercia_entry.delete(0,END)
        self.area_entry.delete(0,END)

        y = self.canvas.winfo_height()
        x = self.canvas.winfo_width()
        self.dibujo.lista_barras_g = self.lista
        self.dibujo.escala(x,y)
        self.dibujo.dibuja_linea(self.canvas,self.dibujo)
```

Figura 16

De forma muy similar, el objeto ventana para modificar los datos tiene la misma estructura, salvo que para elegir el elemento que se desea modificar se ha insertado una lista para poder seleccionar el elemento a través de su nombre y modificará los datos introducidos en la lista de almacenamiento (figuras 17 y 18).

```
"""##########MODIFICAR BARRA#########"""

class ven_mod_barra:

    def __init__(self,raiz, lista,canvas,dibujo):


        self.canvas = canvas
        self.lista = lista
        self.dibujo = dibujo
        self.modificar_barra = Toplevel(raiz)
        self.modificar_barra.title("Modificar Barra")
        self.modificar_barra.geometry("300x150+550+200")
        self.modificar_barra.grab_set()

        self.frame_modificar_barra = ttk.Frame(self.modificar_barra)
        self.frame_modificar_barra.pack()

        ########## Etiquetas ##########

        self.nombre_label = ttk.Label(self.frame_modificar_barra, text="Nombre")
        self.nombre_label.grid(column=0, row=0)

            ########## Desplegable ##########

        self.aux = []

        for b in self.lista:
            self.aux.append(b.nombre)

        self.nombre_cbox = ttk.Combobox(self.frame_modificar_barra, value=self.aux)
        self.nombre_cbox.grid(column=1, row=0)
        self.nombre_cbox.bind('<<ComboboxSelected>>',self.seleccion)

            # Fin desplegable

        self.longitud_label = ttk.Label(self.frame_modificar_barra, text="Longitud")
        self.longitud_label.grid(column=0, row=1)

        self.inercia_label = ttk.Label(self.frame_modificar_barra, text="Inercia")
        self.inercia_label.grid(column=0, row=2)

        self.mod_elas_label = ttk.Label(self.frame_modificar_barra, text="Módulo Elástico")
        self.mod_elas_label.grid(column=0, row=3)

        self.area_label = ttk.Label(self.frame_modificar_barra, text="Área")
        self.area_label.grid(column=0, row=4)

        # Fin etiquetas

        ########## Introduccion de variables ##########
        self.lon = StringVar()
        self.inercia = StringVar()
        self.mod_elas = StringVar()
        self.area = StringVar()

        self.longitud_entry = ttk.Entry(self.frame_modificar_barra,text=self.lon)
        self.longitud_entry.grid(column=1, row=1)

        self.inercia_entry = ttk.Entry(self.frame_modificar_barra, text=self.inercia)
        self.inercia_entry.grid(column=1, row=2)

        self.mod_elas_entry = ttk.Entry(self.frame_modificar_barra, text=self.mod_elas)
        self.mod_elas_entry.grid(column=1, row=3)

        self.area_entry = ttk.Entry(self.frame_modificar_barra, text=self.area)
        self.area_entry.grid(column=1, row=4)

        # Fin introduccion de variables
```

Figura 17

```
    ########## Botones ##########

    self.modificar_button = ttk.Button(self.frame_modificar_barra,
                                       text="Modificar", command=self.modificar)
    self.modificar_button.grid(column=0, row=5)

    self.cancelar_button= ttk.Button(self.frame_modificar_barra,
                                     text="Salir", command=self.modificar_barra.destroy)
    self.cancelar_button.grid(column=1, row=5)

    # Fin botones

def seleccion(self,*ignorar):

    ########## Elección de la barra ##########

    i = StringVar()
    i = self.nombre_cbox.get()
    j = IntVar()
    self.j = self.aux.index(i)
    self.longitud_entry.delete(0,END)
    self.inercia_entry.delete(0,END)
    self.mod_elas_entry.delete(0,END)
    self.area_entry.delete(0,END)

    self.longitud_entry.insert(0,self.lista[self.j].lon)
    self.inercia_entry.insert(0,self.lista[self.j].inercia)
    self.mod_elas_entry.insert(0,self.lista[self.j].E)
    self.area_entry.insert(0,self.lista[self.j].area)

def modificar(self):
    #Validación de datos
    ######################################################
    ########## Modifica la barra ##########
    if len(self.lista)>0:
        aux_barra=[]
        aux_barra.append(Validar(self.longitud_entry.get()))
        aux_barra.append(Validar(self.inercia_entry.get()))
        aux_barra.append(Validar(self.mod_elas_entry.get()))
        aux_barra.append(Validar(self.area_entry.get()))

        if False in aux_barra:
            aux=False
            ven_no_num(self.modificar_barra)

        else:
            aux=True


        if aux==True:
            self.lista[self.j].lon = self.longitud_entry.get()
            self.lista[self.j].inercia = self.inercia_entry.get()
            self.lista[self.j].E = self.mod_elas_entry.get()
            self.lista[self.j].area = self.area_entry.get()

            y = self.canvas.winfo_height()
            x = self.canvas.winfo_width()
            self.dibujo.lista_barras_g = self.lista
            self.dibujo.escala(x,y)
            self.dibujo.dibuja_linea(self.canvas,self.dibujo)
```

Figura 18

Por último, tenemos la opción de borrar cualquier elemento. Esto se realiza seleccionando en una lista desplegable el elemento que deseemos eliminar y aceptando la eliminación. Esto hará que se elimine ese elemento de la lista de almacenamiento (figura 19 y 20).

```
"""##########ELIMINAR BARRA#########"""

class ven_eli_barra:

    ########## Desplegable elección de barra ##########

    def desplegable(self):

        self.aux = []

        for b in self.lista:
            self.aux.append(b.nombre)

        self.nombre_cbox = ttk.Combobox(self.frame_eliminar_barra, value=self.aux)
        self.nombre_cbox.grid(column=1, row=0)

    # Fin desplegable

    def __init__(self,raiz,lista,canvas,dibujo):


        self.canvas = canvas
        self.lista = lista
        self.dibujo = dibujo
        self.eliminar_barra = Toplevel(raiz)
        self.eliminar_barra.title("Eliminar Barra")
        self.eliminar_barra.geometry("300x50+550+200")
        self.eliminar_barra.grab_set()

        self.frame_eliminar_barra = ttk.Frame(self.eliminar_barra)
        self.frame_eliminar_barra.pack()

        ########## Introduccion de etiquetas ##########

        self.nombre_label = ttk.Label(self.frame_eliminar_barra, text="Nombre")
        self.nombre_label.grid(column=0, row=0)

        #Fin introduccion de etiquetas

        ########## Desplegable ##########

        self.desplegable()

        #Fin desplegable

        ########## Botones ##########

        self.eliminar_button = ttk.Button(self.frame_eliminar_barra,
                                          text="Eliminar", command=self.eliminar)
        self.eliminar_button.grid(column=0, row=2)

        self.cancelar_button= ttk.Button(self.frame_eliminar_barra,
                                         text="Salir", command=self.eliminar_barra.destroy)
        self.cancelar_button.grid(column=1, row=2)

        #Fin botones
```

Figura 19

```
def eliminar(self):

    if len(self.lista)>0:

        i = StringVar()
        i = self.nombre_cbox.get()
        self.j = self.aux.index(i)

        ########## Eliminación barra ##########
        del self.lista[self.j]
        del self.aux[self.j]

        self.desplegable()

        y = self.canvas.winfo_height()
        x = self.canvas.winfo_width()
        self.dibujo.lista_barras_g = self.lista
        self.dibujo.escala(x,y)
        self.dibujo.dibuja_linea(self.canvas,self.dibujo)
```

Figura 20

Vista la estructura de este archivo se entiende la estructura de los archivos ven_apoyo, ven_acciones y ven_rotulas ya que no existen diferencias significativas.

Archivo de validación

En este archivo, llamado validacion, encontraremos las validaciones que el programa hace para que no sea posible la introducción de datos no válidos.

Esto se consigue con una función que analizará el tipo de dato introducido y devolverá un valor True o False en función del resultado del análisis. Si el dato no es válido, se llamará a un objeto ventana con la finalidad de avisar al usuario de que el dato no es válido (figura 21)

```
def Validar(val):

    try:
        num = float(val)
        return True

    except ValueError:

        return False

class ven_no_num:
    def __init__(self,ventana):

        self.no_num = Toplevel(ventana)
        self.no_num.title("DATO INVÁLIDO")
        self.no_num.geometry("250x50+550+200")
        self.no_num.grab_set()

        self.frame_no_num = ttk.Frame(self.no_num)
        self.frame_no_num.pack()

        texto = "Algún dato introducido no es un número"

        aviso = ttk.Label(self.frame_no_num, text=texto)
        aviso.pack()
        aceptar = ttk.Button(self.frame_no_num, text="Aceptar", command=self.no_num.destroy)
        aceptar.pack()
```

Figura 111

Seguido a esto, tenemos los objetos ventana que se llama para cuando no existe ninguna barra en el problema planteado y para cuando existe alguna condición de contorno, acción… fuera del rango de esas barras(figura 22).

```
class ven_no_barras:
    def __init__(self,ventana):

        self.no_barras = Toplevel(ventana)
        self.no_barras.title("NO BARRAS")
        self.no_barras.geometry("250x50+550+200")
        self.no_barras.grab_set()

        self.frame_no_barras = ttk.Frame(self.no_barras)
        self.frame_no_barras.pack()

        texto = "Debe existir al menos 1 barra."

        aviso = ttk.Label(self.frame_no_barras, text=texto)
        aviso.pack()
        aceptar = ttk.Button(self.frame_no_barras, text="Aceptar", command=self.no_barras.des
        aceptar.pack()

class fuera_rango:
    def __init__(self,ventana):

        self.rango = Toplevel(ventana)
        self.rango.title("FUERA DE RANGO")
        self.rango.geometry("350x100+550+200")
        self.rango.grab_set()

        self.frame_rango = ttk.Frame(self.rango)
        self.frame_rango.pack()

        texto = "\nLas acciones, apoyos y discontinuidades deben estar dentro\n\
        de los límites de las barras (Las barras empiezan en 0).\n\
        (Las discontinuidades no pueden estar en los extremos)\n"

        aviso = ttk.Label(self.frame_rango, text=texto)
        aviso.pack()
        aceptar = ttk.Button(self.frame_rango, text="Aceptar", command=self.rango.destroy)
        aceptar.pack()
```

Figura 22

A continuación se encuentra el objeto ventana para el aviso de que el sistema de ecuaciones no se puede resolver. Puede ser debido a que no nos encontramos ante una estructura, si no ante un mecanismo, o a que las condiciones impuestas no sean coherentes (figura 23).

```
class matriz_singular:
    def __init__(self,ventana):

        self.singular = Toplevel(ventana)
        self.singular.title("MATRIZ SINGULAR")
        self.singular.geometry("350x80+550+200")
        self.singular.grab_set()

        self.frame_singular = ttk.Frame(self.singular)
        self.frame_singular.pack()

        texto = "MATRIZ SINGULAR (No se puede resolver el sistema).\n\
        (Puede deberse a que la viga es un mecanismo).\n"

        aviso = ttk.Label(self.frame_singular, text=texto)
        aviso.pack()
        aceptar = ttk.Button(self.frame_singular, text="Aceptar", command=self.singular.destroy)
        aceptar.pack()
```

Figura 12

Por último tenemos un objeto que es el encargado de crear, abrir y guardar un proyecto. Comprueba si el usuario ya guardó todo lo necesario antes de borrar los datos actuales (figura 24).

```
class proyecto_nuevo:
    def __init__(self,ventana,filename,canvas,dibujo,lista_barras,lista_apoyos,lista_acciones,lista_rotulas,flag):

        self.filename = filename
        self.ventana = ventana
        self.canvas = canvas
        self.dibujo = dibujo
        self.lista_barras = lista_barras
        self.lista_apoyos = lista_apoyos
        self.lista_acciones = lista_acciones
        self.lista_rotulas = lista_rotulas
        self.flag = flag

        self.rango = Toplevel(self.ventana)
        self.rango.title("Nuevo")
        self.rango.geometry("350x100+550+200")
        self.rango.grab_set()

        self.frame_rango = ttk.Frame(self.rango)
        self.frame_rango.pack()

        texto = "\n¿Has guardado todo lo que tenías que guardar?\n"

        aviso = ttk.Label(self.frame_rango, text=texto)
        aviso.pack()
        si = ttk.Button(self.frame_rango, text="Sí", command=self.afirmativo)
        si.pack()
        no = ttk.Button(self.frame_rango, text="No", command=self.rango.destroy)
        no.pack()

    def afirmativo(self):

        self.rango.destroy()

        if self.flag==False:

            borra_todo(self.canvas,self.dibujo,self.lista_barras, self.lista_apoyos,
                       self.lista_acciones, self.lista_rotulas)
            self.filename[0]=""

        elif self.flag==True:

            self.filename[0] = abrir_archivo(self.filename, self.canvas,
        self.dibujo, self.lista_barras, self.lista_apoyos, self.lista_acciones, self.lista_rotulas)
```

Figura 13

Ventana canvas

En el archivo proyecto_canvas tenemos un onjeto ventana que será donde se visualizarán todos los elementos introducidos para la ejecución del programa. Esta ventana canvas se inserta en el frame de la ventana principal.

La forma de operar es, ante un nuevo elemento, borra todo lo dibujado y redibuja todo de nuevo, pero antes de redibujarlo, primero comprueba los datos del tamaño de la ventana y realiza un escalado acorde con ello. Solo se mostrará la función de escalado y del dibujo de las barras debido a que todas las funciones para dibujar los elementos son repetitivas y no aportan nada nuevo (figuras 25 y 26).

```
def escala(self,x,y):

    self.px0 = x/2
    self.py0 = y/2
    self.lon = 0

    for i in self.lista_barras_g:

        self.lon = self.lon + float(i.lon)

    for i in self.lista_apoyos_g:

        if float(i.posx)>self.lon:

            self.lon = float(i.posx)

    for i in self.lista_acciones_g:

        if (i.datos[0]=="1" or i.datos[0]=="2" or i.datos[0]=="4")\
        and float(i.datos[1])>self.lon :

            self.lon = float(i.datos[1])

        elif i.datos[0]=="3" and float(i.datos[2])>self.lon:

            self.lon = float(i.datos[2])

    for i in self.lista_rotulas_g:

        if float(i.posicion)>self.lon:

            self.lon = float(i.posicion)

    if self.lon >0:

        self.esc_x = ((self.px0*2)-150)/self.lon
        self.px0 = (self.px0)-(self.lon*(self.esc_x/2))
        self.esc_y = self.esc_x

    if self.lon <=10:

        self.g1 = 7
        self.g2 = 7


    elif self.lon >=10 and self.lon <=20:

        self.g1 = 5
        self.g2 = 5


    elif self.lon >20:

        self.g1 = 4
        self.g2 = 4
```

Figura 14

```
def dibuja_linea(self,canvas,dibujo):

    ax = self.px0
    ay = self.py0-0.2*self.esc_y
    by = self.py0+0.2*self.esc_y

    self.dibujo = dibujo
    self.c_v_p = canvas

    print("linea canvas")
    print(len(self.lista_lineas))

    for i in self.lista_lineas:

        self.c_v_p.delete(i)

    del self.lista_lineas[0:len(self.lista_lineas)]

    for i in self.lista_barras_g:

        bx = ax + float(i.lon)*self.esc_x

        self.lista_lineas.append(self.c_v_p.create_line(ax,self.py0,bx,self.py0,fill='#000',width=self.g1))
        self.lista_lineas.append(self.c_v_p.create_line(ax,ay,ax,by,fill='#f00',width=self.g2))
        self.lista_lineas.append(self.c_v_p.create_line(bx,ay,bx,by,fill='#f00',width=self.g2))

        ax = ax + float(i.lon)*self.esc_x

    self.dibujo.dibuja_apoyo(self.c_v_p,self.dibujo)
    self.dibujo.dibuja_fuerza(self.c_v_p,self.dibujo)
    self.dibujo.dibuja_rotula(self.c_v_p,self.dibujo)
    self.dibujo.dibuja_despla(self.c_v_p,self.dibujo)
```

Figura 26

Se puede observar que desde la función dibuja_linea se llama al resto de funciones de dibujo. Esto se hace para evitar solapamientos en los dibujos y realizar el trazado siempre en el mismo orden.

Comprobación de las discontinuidades

Con este archivo, llamado comprobación_rotulas, se comprueba la coincidencia de acciones en discontinuidades que pueden inducir condiciones erróneas e incoherentes en el problema. Un ejemplo de ello puede ser la aplicación de un momento en una rótula. Ésta acción debe especificarse en que elemento se aplica ya que existe la posibilidad de aplicarse a la izquierda, a la derecha, o causando la incoherencia, en la propia rótula. El objeto creado para este fin es el siguiente (figuras 27 y 28).

```
class comp_rotulas:

    def __init__(self,raiz, lista_r, lista_acc, matriz,lista_e,lista_n):

        self.lista_r = lista_r
        self.lista_acc = lista_acc
        self.matriz = matriz
        self.lista_e = lista_e
        self.lista_n = lista_n
        self.lista_coin = [] #Almacena los indices de las acciones de las coincidencias
        self.lista_obj_coin = []
        self.raiz = raiz

        for i in self.lista_acc:

            i.obser = ""


        class b:
            def __init__(self,frame,n,indice,lista_acc):

                self.frame_comp = frame
                n=n
                i=indice
                self.lista_acc = lista_acc

                self.var = StringVar()


                self.coin_label = ttk.Label(self.frame_comp,
                text=self.lista_acc[i].nombre).grid(column=0, row=n)

                self.coin_rbutton_i = ttk.Radiobutton(self.frame_comp,
                variable = self.var, text = "Izq",value=0).grid(column=1, row=n)

                self.coin_rbutton_d = ttk.Radiobutton(self.frame_comp,
                variable = self.var,text = "Der",value=1).grid(column=2, row=n)

            def okey(self,i):

                self.i = i

                self.lista_acc[self.i].obser = self.var.get()



        for i in self.lista_acc:

            for j in self.lista_r:

                if (i.datos[0] == "1" and j.tipo =="1" and i.datos[1]==j.posicion)\
                   or (i.datos[0] == "2" and j.tipo =="2" and i.datos[1]==j.posicion)\
                   or (i.datos[0] == "4" and j.tipo =="3" and i.datos[1]==j.posicion):

                    self.lista_coin.append(self.lista_acc.index(i))
```

Figura 27

```
        if (len(self.lista_coin))!=0:

            self.comprobacion = Toplevel(self.raiz)
            self.comprobacion.title("Coincidencias")
            self.comprobacion.geometry("300x150+550+200")
            self.comprobacion.grab_set()

            self.frame_comp = ttk.Frame(self.comprobacion)
            self.frame_comp.pack()

            self.expl_label = ttk.Label(self.frame_comp,
            text="Estas son las acciones que coinciden con alguna discontinuidad")

            self.expl_label.grid(column=0,row=0)

            n=1
            for i in self.lista_coin:

                self.lista_obj_coin.append(b(self.frame_comp,n,i,self.lista_acc))
                n+=1

            self.ok = ttk.Button(self.frame_comp,text="OK",
            command=self.oks).grid(column=0,row=8)

        else:
            self.sigue(self.raiz)


    def oks(self):
        m=0
        for i in self.lista_obj_coin:

            j = self.lista_coin[m]

            i.okey(j)
            m+=1

        self.comprobacion.destroy()

        self.sigue(self.raiz)

    def sigue (self,raiz):


        for i in self.lista_e.list_elementos:

            for j in self.lista_r:

                if i.nodIni[0]==j.posicion:

                    i.obser.append(j.tipo)
```

Figura 28

En este punto pasamos de los archivos usados para introducir los datos del problema, a los archivos que operan de forma interna. En primer lugar están los archivos que manipulan los datos introducidos por el usuario para adecuarlos a la estructura del sistema de ecuaciones. Después está el archivo encargado de toda la construcción matemática necesaria para la resolución del problema y, por último, está el archivo encargado de realizar las gráficas de los resultados de los esfuerzos.

Creación de nudos

El archivo proyecto_nudos crea los nudos que definen los elementos barra que formarán el problema que se resolverá realmente. Los nudos normalmente coinciden con los apoyos y las uniones de las distintas vigas introducidas por el usuario, pero además, este archivo crea nudos para que el archivo que genera los elementos matemáticos (matrices de rigidez, matrices equivalente…) pueda crearlos sin problemas.

Un ejemplo de creación de nudos es en los puntos de la viga donde empieza y acaba una fuerza distribuida. Si estos puntos coinciden con los extremos de la viga, no se crearán nudos nuevos, pero si alguno de estos puntos se encuentra en una zona intermedia, se creará un nudo.

Además, en los nudos se almacenan datos como si el apoyo de ese nudo está girado respecto del sistema de coordenadas, si ese punto tiene algún desplazamiento impuesto, las libertades que tiene y si existe un muelle en él.

En la figura 29 se ve el objeto Nudos con el que se trabajará.

```
class Nudos:
    ####Clase para la manipulación de los apoyos.
    def __init__(self, pos, giro,des,lib,a_e):

        self.pos = pos
        self.giro = giro
        self.des = des
        self.lib = lib
        self.a_e = a_e
        self.obser = ""
        self.nombre = ""
        self.mat_L =""
```

Figura 29

En la figura 30 se muestra una parte del objeto lista_nudos que será la lista donde se almacenarán los objetos Nudo. Básicamente, lo que realiza este objeto es comparar los datos de las vigas introducidas por el usuario con el resto de elementos (acciones, apoyos y discontinuidades).

```
class lista_nudos:

    def __init__(self):

        #Lista de objetos Nudos(pos,giro,des[x,y,o],lib[x,y,o],a_e[tipo,kx,ky,ko])
        self.list_nudos = []

    def anadir_nudo (self,lista_b, lista_a, lista_acc, lista_r):

        del self.list_nudos[0:len(self.list_nudos)]

        #Creacion de nudos a partir de las uniones de las barras, los apoyos introducidos.
        aux2 = float(0)
        giro = 0
        des = [0,0,0]
        lib = [1,1,1]
        a_e = [0,0,0,0]

        nudo = Nudos(aux2, giro,des,lib,a_e)
        self.list_nudos.append(nudo)
        auxLonBarra = 0

        #En cada unión de barras creamos un nudo si no hay un apoyo en ese lugar.
        for barra in lista_b:

            n = 0
            aux = float(barra.lon)

            auxLonBarra = auxLonBarra + aux
            aux2 = float(self.list_nudos[n].pos) + auxLonBarra
            giro = 0
            des = [0,0,0]
            lib = [1,1,1]
            a_e = [0,0,0,0]

            nudo = Nudos(aux2, giro,des,lib,a_e)
            self.list_nudos.append(nudo)
            n+=1

        lista_a_aux = lista_a[:]
        lista_a_aux.sort(key = lambda apoyo: apoyo.posx)
```

Figura 30

Manipulación de las distribuciones de fuerzas

En el archivo proyecto_fuerzas no se crea ningún objeto, sino que una función manipula la lista de las fuerzas introducidas por el usuario. Esta función solo modifica las distribuciones de fuerzas.

La manipulación consiste en dos situaciones, que pueden darse de forma separada o simultáneamente:

1. Ante distribuciones que cambian de signo.

2. Ante distribuciones que contengan nudos en alguna zona intermedia entre sus extremos.

Para la primera situación, se crean dos distribuciones. Una desde su valor inicial hasta 0 y la otra desde 0 hasta su valor inicial. Para ello calcula la posición del punto donde el valor de la distribución es 0.

Para la segunda situación, se trocea la distribución tantas veces como nudos haya entre sus extremos.

Esto se ve en las figuras 31 y 32.

```
def anadir_fuerzas (lista_n, lista_e, lista_acc, lista_aux):

    lista_n = lista_n
    lista_e = lista_e
    lista_acc = lista_acc
    lista_aux = lista_aux

    del lista_aux[0:len(lista_aux)]


    for i in lista_acc:

        if i.datos[0] == "1" or i.datos[0]=="2" or i.datos[0]=="4":

            lista_aux.append(i)

        elif i.datos[0] == "3":

            if float(i.datos[3])*float(i.datos[4]) <0:

                nombre = i.nombre + str(.2)
                tipo = i.datos[0]
                val_ini = 0
                pos_ini = float(i.datos[1])-(float(i.datos[3])/\
                ((float(i.datos[4])-float(i.datos[3]))/(float(i.datos[2])-float(i.datos[1]))))
                val_fin = i.datos[4]
                pos_fin = i.datos[2]

                datos = [tipo, pos_ini, pos_fin, val_ini, val_fin]
                fuerza = Accion(nombre, datos)
                lista_aux.append(fuerza)

                nombre = i.nombre +str(.1)
                tipo = i.datos[0]
                val_ini = i.datos[3]
                pos_fin = pos_ini
                pos_ini = i.datos[1]
                val_fin = 0

                datos = [tipo, pos_ini, pos_fin, val_ini, val_fin]
                fuerza = Accion(nombre, datos)
                lista_aux.append(fuerza)

            else:
                datos = [i.datos[0], i.datos[1], i.datos[2], i.datos[3], i.datos[4]]
                fuerza = Accion(i.nombre, datos)
                lista_aux.append(fuerza)
```

Figura 31

```
for k in lista_aux:

    if k.datos[0]=="3":

        for j in lista_n.list_nudos:

            if float(k.datos[1])< j.pos < float(k.datos[2]):

                if k.datos[3]==k.datos[4]:

                    nombre = k.nombre + str("a")
                    datos = [k.datos[0],j.pos,k.datos[2],k.datos[3],k.datos[4]]
                    accion = Accion(nombre,datos)
                    lista_aux.append(accion)


                    k.datos[2]=j.pos

                else:

                    m = float((float(k.datos[4])-float(k.datos[3]))/(float(k.datos[2])-float(k.datos[1])))

                    val_ini = m*(float(j.pos)-float(k.datos[1]))+float(k.datos[3])

                    nombre = k.nombre + str("a")
                    datos = [k.datos[0],j.pos,k.datos[2],val_ini,k.datos[4]]
                    accion = Accion(nombre,datos)
                    lista_aux.append(accion)

                    k.datos[2] = j.pos
                    k.datos[4] = val_ini

lista_aux.sort(key = lambda accion: float(accion.datos[1]))
```

Figura 32

Creación de los elementos

El archivo llamado proyecto_elemento toma la lista de vigas introducidas por el usuario y la lista de objetos Nudos obtenida anteriormente y se crea un objeto Elemento entre cada nudo consecutivo (figura 33).

Los objetos Elemento, al igual que los objetos Nudo, contienen toda la información de las características de cada elementos y además, cada elementos almacena sus propias matrices de rigidez y las matrices equivalentes, de cambio de base y las reacciones y desplazamientos de los extremos. La solución del sistema se almacenan en los propios elementos (figura 34).

```
class lista_elementos:

        #Lista de objetos Elemento(nombre, nodIni, nodFin, area, lon, inercia, E,femp[],femp_tot)
        def __init__(self):
                self.list_elementos = []


        def anadir_elementos(self, lista_b, lista_n):

                del self.list_elementos[0:len(self.list_elementos)]

                self.lista_b = lista_b
                self.lista_n = lista_n

                #Creación de un elemento en función de los nudos.
                #Entre dos apoyos creamos un elemento y le asignamos sus propiedades.
                #Los puntos de unión de dos barras también se consideran nudos.
                j = 0
                k = 1

                for i in self.lista_n.list_nudos:

                        if k < len(self.lista_n.list_nudos):

                                nombre = str(j)
                                j+=1
                                nodIni = [float(i.pos),i.giro,i.des,i.lib,i.a_e,i.obser]
                                nodFin = [float(self.lista_n.list_nudos[k].pos),(self.lista_n.list_nudos[k].giro),\
                                          (self.lista_n.list_nudos[k].des),(self.lista_n.list_nudos[k].lib),\
                                          (self.lista_n.list_nudos[k].a_e),(self.lista_n.list_nudos[k].obser)]
                                nudo_Ini = i.nombre
                                nudo_Fin = self.lista_n.list_nudos[k].nombre
                                lon = nodFin[0] - nodIni[0]
                                L_Ini = i.mat_L
                                L_Fin = self.lista_n.list_nudos[k].mat_L
                                k+=1
                                flag = True
                                m = 1
                                aux2=float(self.lista_b[0].lon)

                                for l in self.lista_b:

                                        if nodFin[0] <= aux2 and flag ==True:

                                                area = l.area
                                                inercia = l.inercia
                                                E = l.E

                                                flag = False

                                        else:
                                                aux2=aux2 + float(self.lista_b[m].lon)
                                                m+=1

                                elemento = Elemento(nombre,nudo_Ini, nudo_Fin, nodIni, nodFin, area, lon, inercia, E , L_Ini, L_Fin)
                                self.list_elementos.append(elemento)
```

Figura 33

```
class Elemento:

        def __init__ (self, nombre, nudo_Ini, nudo_Fin, nodIni, nodFin, area, lon, inercia, E, L_Ini, L_Fin):

                self.nombre = nombre
                self.nudo_Ini = nudo_Ini#Numero del nodo
                self.nudo_Fin = nudo_Fin#Numero del nodo
                self.nodIni = nodIni #pos,giro,des,lib,a_e,obser,mat_est* (obser: Para discontinuidades ("1": Rótula en la izq))
                self.nodFin = nodFin #pos,giro,des,lib,a_e,obser,mat_est* (obser: Para discontinuidades ("1": Rótula en la izq))
                self.area = area
                self.lon = lon
                self.inercia = inercia
                self.E = E
                self.femp_Ini = [] #Lista de matrices
                self.femp_tot_Ini = matrix([[0],[0],[0]])
                self.femp_Fin = [] #Lista de matrices
                self.femp_tot_Fin = matrix([[0],[0],[0]])
                self.fequ_Ini = ""
                self.fequ_Fin = ""
                self.L_Ini = L_Ini
                self.L_Fin = L_Fin
                self.mat_K = {}
                self.Fest_Ini = []
                self.Uest_Ini = []
                self.Fest_Fin = []
                self.Uest_Fin = []
                self.F_local =  matrix([[0],[0],[0],[0],[0],[0]])
                self.obser = []
                self.cortante = []
                self.flector = []
                self.giro = []
                self.deformada = []
                self.eje_x = []
```

Figura 34

Manipulaciones algebraicas

En este punto ya tenemos todos los datos introducidos, manipulados y organizados para que el programa pueda generar todos los elementos matemáticos para formar el sistema de ecuaciones. Para ello está el archivo proyecto_matrices.

Este archivo genera un objeto llamado matriz. Este objeto contiene todas las funciones con los métodos matemáticos necesarios.

Empieza con la formación de las matrices de empotramiento de cada objeto Elemento en función de la acción aplicada en él y de las discontinuidades en los extremos (solo se muestra una porción de la función) (figura 35).

```
def matriz_emp(self):

    for i in self.lista_e.list_elementos:

        for j in self.lista_f:

            if j.datos[0]=="1" and float(i.nodIni[0]) < float(j.datos[1]) < float(i.nodFin[0]):

                M = -float(j.datos[3])###############
                a = float(j.datos[1])-float(i.nodIni[0])
                b = float(i.nodFin[0]) - float(j.datos[1])
                L = float(i.lon)
                Mi = M*b*(2-3*(b/L))/L
                Mj = M*a*(2-3*(a/L))/L
                Ri = 6*M*a*b/power(L,3)
                Rj = 6*M*a*b/power(L,3)

                if i.nodIni[5]=="":

                    f_emp_Ini = matrix([[0.],\
                                        [-Ri],\
                                        [-Mi]])

                    f_emp_Fin = matrix([[0.],\
                                        [Rj],\
                                        [-Mj]])

                    i.femp_Ini.append(f_emp_Ini)
                    i.femp_Fin.append(f_emp_Fin)

                elif i.nodIni[5]=="1":

                    f_emp_Ini = matrix([[0.],\
                                        [-Ri-(3*Mi/(2*L))],\
                                        [0.]])

                    f_emp_Fin = matrix([[0.],\
                                        [Rj+(3*Mi/(2*L))],\
                                        [-Mj-(Mi/2)]])

                    i.femp_Ini.append(f_emp_Ini)
                    i.femp_Fin.append(f_emp_Fin)
```

Figura 35

Después está la función de cambio de base que se aplica en el caso de que algún nudo está girado (figura 36).

```
def matriz_L(self,giro):

    a = math.radians(giro)

    mat_L = matrix([[math.cos(a),math.sin(a),0],\
                    [-math.sin(a),math.cos(a),0],\
                    [0,0,1]])
```

Figura 36

Continua la función para formar las matrices equivalentes que dependen de las dos anteriores (figura 37).

```
def matriz_equ(self):
    for i in self.lista_e.list_elementos:
        if float(i.nodIni[1]) !=0:
            i.fequ_Ini = (i.L_Ini)*(-i.femp_tot_Ini)
        else:
            i.fequ_Ini = -i.femp_tot_Ini
        if float(i.nodFin[1]) !=0:
            i.fequ_Fin = (i.L_Fin)*(-i.femp_tot_Fin)
        else:
            i.fequ_Fin = -i.femp_tot_Fin
```

Figura 37

Seguido está la función que genera la matriz de rigidez de cada elemento. Al igual que la matriz de empotramiento, la matriz de rigidez depende de las acciones aplicadas en el elemento y las discontinuidades en los extremos (solo se muestra una pequeña porción de la función) (figura 38).

Las matrices de rigidez se componen de 4 matrices 3x3 que se unen creando la matriz 6x6 del elemento completo.

```
def matriz_K_ele(self):

    self.matriz_muelle = {}
    for i in self.lista_e.list_elementos:

        L = float(i.lon)
        A = float(i.area)
        E = float(i.E)
        I = float(i.inercia)

        if i.nodIni[5]=="":

            mat_K1=matrix ([[E*A/L,0,0],\
                            [0,12*E*I/power(L,3),6*E*I/ power(L,2)],\
                            [0,6*E*I/power(L,2),4*E*I/L]])


            mat_K2=matrix ([[-E*A/L,0,0],\
                            [0,-12*E*I/ power(L,3),6*E*I/ power(L,2)],\
                            [0,-6*E*I/power(L,2),2*E*I/L]])


            mat_K3=matrix ([[-E*A/L,0,0],\
                            [0,-12*E*I/power(L,3),-6*E*I/power(L,2)],\
                            [0,6*E*I/power(L,2),2*E*I/L]])

            mat_K4=matrix ([[E*A/L,0,0],\
                            [0,12*E*I/power(L,3),-6*E*I/power(L,2)],\
                            [0,-6*E*I/power(L,2),4*E*I/L]])

            if i.nodIni[1]!="0":

                mat_L = matriz.matriz_L(self,float(i.nodIni[1]))

                mat_K1 = mat_L * mat_K1 * mat_L.transpose()
                mat_K2 = mat_K2 * mat_L.transpose()
                mat_K3 = mat_L * mat_K3

            if i.nodFin[1]!="0":

                mat_L = matriz.matriz_L(self,float(i.nodFin[1]))

                mat_K4 = mat_L * mat_K4 * mat_L.transpose()
                mat_K2 = mat_K2 * mat_L.transpose()
                mat_K3 = mat_L * mat_K3

            mat_K = {i.nudo_Ini+","+i.nudo_Ini:mat_K1,i.nudo_Ini+","+i.nudo_Fin:mat_K2,\
                     i.nudo_Fin+","+i.nudo_Ini:mat_K3,i.nudo_Fin+","+i.nudo_Fin:mat_K4}

            i.mat_K = mat_K
```

Figura 38

Y para finalizar el boque de los elementos matemáticos, está la función que crea las matrices estáticas, donde se encuentran las incógnitas de las fuerzas resultantes y los desplazamientos de los nodos (solo se muestra una pequeña porción de la función) (figura 39).

```
def matriz_est(self):

    for i in self.lista_e.list_elementos:

        if i.nodIni[3][0] == 1:
            Rxi = 0.
            Ui = str("U")

        else:
            Rxi = str("Rx")
            Ui = i.nodIni[2][0]
```

Figura 39

Sistema de ecuaciones

Ahora, en el mismo archivo que los elementos matemáticos, proyecto_matrices, sigue la función que forma el sistema de ecuaciones del problema. Aquí no se diferencia entre datos del problema e incógnitas. Se ve en la figura 40.

```
def sistema(self):

    for i in self.lista_e.list_elementos:

        print("elemento" + str(i.nombre))
        print("nudo ini " +str(i.nodIni[0])+" -- "+"nudo fin " +str(i.nodFin[0])+"\n")

    for j in self.lista_f:

        print(str(j.nombre))
        print("pos ini "+str(j.datos[1])+" -- "+"pos fin "+str(j.datos[2]))
        print("val ini "+str(j.datos[3])+" -- "+"val fin "+str(j.datos[4])+"\n")

    self.lista_FEst = []
    self.lista_UEst = []

    dim = (len(self.lista_e.list_elementos)+1)*3
    self.matriz_K = asmatrix(zeros((dim,dim)))
    self.matriz_Equi = asmatrix(zeros((dim,1)))
    p=0

    for i in self.lista_e.list_elementos:

        for clave in i.mat_K:

            indice = clave

            mat = i.mat_K[clave]

            coma = indice.index(",")
            sub1 =  (int(indice[:coma])-1)*3
            sub2 =  (int(indice[coma+1:])-1)*3

            for j in range(3):

                sub3=sub2

                for k in range(3):

                    self.matriz_K[sub1,sub3]=self.matriz_K[sub1,sub3]+mat[j,k]
                    sub3+=1

                sub1+=1

        for q in range(3):

            self.matriz_Equi[p,0]=self.matriz_Equi[p,0]+i.fequ_Ini[q,0]
            self.matriz_Equi[p+3,0]=self.matriz_Equi[p+3,0]+i.fequ_Fin[q,0]

            p+=1

        self.lista_FEst = self.lista_FEst+i.Fest_Ini
        self.lista_UEst = self.lista_UEst+i.Uest_Ini

    self.lista_FEst=self.lista_FEst+self.lista_e.list_elementos[-1].Fest_Fin
    self.lista_UEst=self.lista_UEst+self.lista_e.list_elementos[-1].Uest_Fin
```

Figura 40

Resolución

La solución obtenida con esta función son las reacciones en los apoyos de la estructura y los desplazamientos de los nudos. Los desplazamientos son los mismos que los de los objetos Elemento por separado, pero los esfuerzos que obtenemos aquí no.

La función trabaja de forma que primero busca las filas donde se encuentran las incógnitas de esfuerzos y forma un subsistema con solo dichas incógnitas. Entonces, se resuelve y con esos resultados, se sustituyen en las ecuaciones eliminadas anteriormente obteniendo los esfuerzos (solo se mostrará la parte de la función en la que se obtienen los desplazamientos) (figura 41 y 42).

```
def resolucion(self):

    incog = []
    aux=0
    m=0
    n=0
    mat_k = self.matriz_K
    dim1=[]
    dim2=[]

    for i in self.lista_FEst:

        if i=="Rx" or i=="Ry" or i=="Ro":

            incog.append(aux)

        else:
            dim1.append(aux)

        aux+=1

    dim = len(self.lista_FEst)-len(incog)

    mat_FEst=asmatrix(zeros((dim,1)))

    for i in range(len(self.lista_FEst)):

        if i not in incog:

            mat_FEst[m,0]= float(mat_FEst[m,0])+
            float(self.matriz_Equi[i,0])+float(self.lista_FEst[i])

            m+=1
    m=0

    for i in range(len(self.lista_UEst)):

        if self.lista_UEst[i]!="Do" and self.lista_UEst[i]!="0"
        and self.lista_UEst[i]!='U' and self.lista_UEst[i]!='V':

            dim2.append(i)

    aux_k_UEst=asmatrix(zeros((len(dim1),len(dim2))))
    aux_UEst=asmatrix(zeros((len(dim2),1)))

    for i in range(len(dim2)):

        aux_UEst[i,0] = float(self.lista_UEst[dim2[i]])

        for j in range(len(dim1)):

            aux_k_UEst[j,i]= mat_k[dim1[j],dim2[i]]

    aux_k=aux_k_UEst*aux_UEst

    mat_FEst = mat_FEst - aux_k
```

Figura 41

```
for i in incog:

    i=i-m
    mat_k = delete(mat_k,i,0)
    mat_k = delete(mat_k,i,1)
    m+=1

mat_Usolve=linalg.solve(mat_k,mat_FEst)

for i in range(len(self.lista_UEst)):

    if i not in incog:

        self.lista_UEst[i]=mat_Usolve[n,0]
        n+=1


m=0
n=0
p=0
dim = len(self.lista_UEst)

mat_UEst=asmatrix(zeros((dim,1)))

for i in range(len(self.lista_UEst)):

    mat_UEst[i,0]= float(mat_UEst[i,0])+float(self.lista_UEst[i])

mat_k = self.matriz_K


mat_Fsolve=mat_k*mat_UEst-self.matriz_Equi
```

Figura 42

Esfuerzos locales

Para finalizar, está la función que obtiene los esfuerzos cada elemento individualmente y en su sistema de referencia local (figura 43).

```
def locales(self):

    for i in self.lista_e.list_elementos:

        k_local = matrix([[0.,0.,0.,0.,0.,0.],\
                    [0.,0.,0.,0.,0.,0.],\
                    [0.,0.,0.,0.,0.,0.],\
                    [0.,0.,0.,0.,0.,0.],\
                    [0.,0.,0.,0.,0.,0.],\
                    [0.,0.,0.,0.,0.,0.]])

        u_aux = matrix([[0.],[0.],[0.],[0.],[0.],[0.]])

        F_emp_local = matrix([[0.],[0.],[0.],[0.],[0.],[0.]])

        for clave in i.mat_K:

            indice = clave

            mat = i.mat_K[clave]

            coma = indice.index(",")
            sub1 =  int(indice[:coma])
            sub2 =  int(indice[coma+1:])

            if sub1 == int(i.nudo_Ini) and sub2 == int(i.nudo_Ini):

                for j in range(3):

                    for k in range(3):

                        k_local[j,k] = mat[j,k]

            elif sub1 == int(i.nudo_Ini) and sub2 == int(i.nudo_Fin):

                for j in range(3):

                    for k in range(3):

                        k_local[j,k+3] = mat[j,k]

            elif sub1 == int(i.nudo_Fin) and sub2 == int(i.nudo_Ini):

                for j in range(3):

                    for k in range(3):

                        k_local[j+3,k] = mat[j,k]

            elif sub1 == int(i.nudo_Fin) and sub2 == int(i.nudo_Fin):

                for j in range(3):

                    for k in range(3):

                        k_local[j+3,k+3] = mat[j,k]

        for j in range(3):

            l=j+3
            u_aux[j,0] = i.Uest_Ini[j]
            u_aux[l,0] = i.Uest_Fin[j]
```

Figura 43

Estos valores, con los desplazamientos de los nudos, se almacenan en los objetos Elemento para, en el último archivo, dibujar las gráficas de los esfuerzos.

Gráficas

Con este último archivo, proyecto_graficas, se consiguen las gráficas de los elementos. Para ello se toman los valores de los desplazamientos y los esfuerzos del extremo izquierdo de las barras y le suma las ecuaciones de las acciones aplicadas.

Los resultados se van obteniendo por integración directa de las ecuaciones de las acciones.

Para los esfuerzos cortantes se integran las acciones, para los momentos cortantes se integran los esfuerzos cortantes. Para los giros se integran los momentos y para los desplazamientos los giros. Para estos dos últimos además hace falta dividir por la inercia de cada barra y su módulo elástico (se muestra solo la obtención de los esfuerzos cortantes) (figura 44).

```
def graficar (lista_e, lista_f):

    lista_e = lista_e
    lista_f = lista_f
    cortante_general = []
    flector_general = []
    giro_general = []
    deformada_general = []
    eje_x_general = []

    eje_aux = []
###### Cortante ######
    for j in lista_e.list_elementos:

        del j.cortante[0:len(j.cortante)]

        del j.eje_x[0:len(j.eje_x)]

        j.eje_x = linspace(float(j.nodIni[0]),float(j.nodFin[0]),num=1000)

        for l in range(len(j.eje_x)):

            j.cortante.append(0.)

        for i in lista_f:

            eje_aux = linspace(0,float(j.nodFin[0])-float(j.nodIni[0]),num=1000)


            if i.datos[0]=="3" and float(i.datos[1])>= float(j.nodIni[0])
            and float(i.datos[2]) <= float(j.nodFin[0]):

                a = float(i.datos[3])

                b = ((float(i.datos[4])-float(i.datos[3]))/(float(i.datos[2])-float(i.datos[1])))

                for l in range(len(eje_aux)):

                    j.cortante[l] = j.cortante[l] + a*eje_aux[l]+ b*0.5*power(eje_aux[l],2)

            elif i.datos[0]=="2" and float(i.datos[1])> float(j.nodIni[0])
            and float(i.datos[2]) < float(j.nodFin[0]):

                for k in range(len(eje_aux)):

                    if float(j.eje_x[k])>=float(i.datos[1]):

                        j.cortante[k] = j.cortante[k] + float(i.datos[3])


    for j in lista_e.list_elementos:

        for l in range(len(j.eje_x)):

            j.cortante[l] = -(j.cortante[l] - float(j.F_local[1,0]))
```

Figura 44

Con esto se ha hecho una rápida explicación de en qué consiste el software.

A partir de aquí se verá un ejemplo de funcionamiento del software y se verá con lo que trabajará el usuario a modo de guía de usuario.

VI. Función numpy.linalg.solve()

La resolución del sistema de ecuaciones es gracias a la función linalg.solve() de la librería NumPy. Esta función se basa en la colección de funciones de LAPACK (Linear Algebra Package) que está escrita en Fortran 90 desarrollándose desde 1987 por universidades y laboratorios de Europa y Estados Unidos apoyada económicamente por la Fundación Nacional de Ciencias de Estados Unidos.

A su vez, siguiendo el pensamiento de software libre, está incluida en una librería totalmente pública y con constantes actualizaciones.

Dado el largo tiempo de desarrollo de esta colección de funciones y las instituciones que han trabajado en ese desarrollo hace que las reglas utilizadas por esta función para la resolución de sistemas sean lo suficientemente robustas como para dar por buenos los resultados que arroja y queda justificada su utilización.

VII. Manual

En este apartado se mostrará el funcionamiento del software desde el punto de vista del usuario final. Desde la introducción de las variables del problema, hasta la visualización de las gráficas de los resultados pasando por la visualización del problema, modificación y eliminación de los datos introducidos.

Ventana principal

En la figura 45 se puede ver la ventana principal del programa. Desde ella se realizan las introducciones de datos, apertura, cierre y guardado de ficheros y se da la orden para comenzar el cálculo.

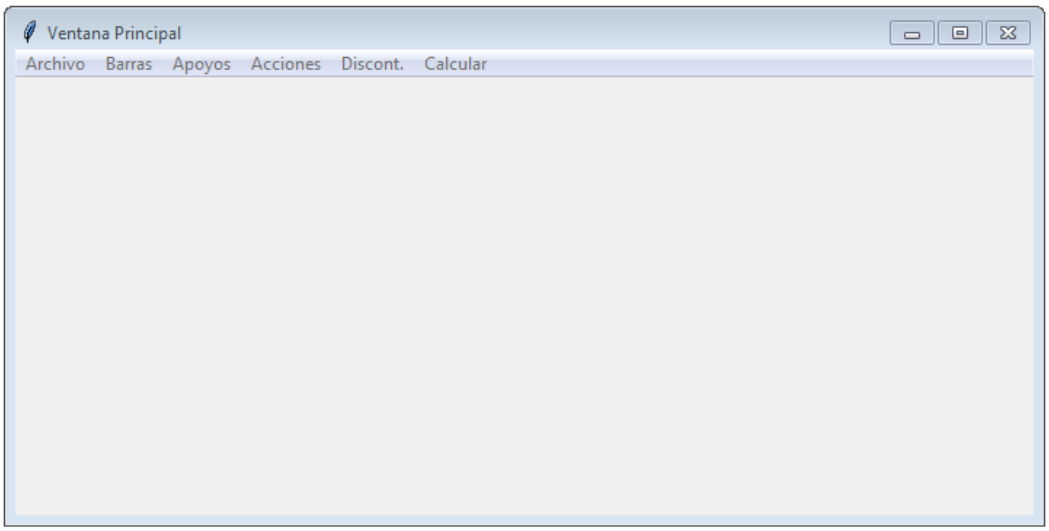


Figura 15

En esta ventana se pueden ver las pestañas de Archivo, Barras, Apoyos, Acciones, Discont. y Calcular. Además, en esta ventana se visualizarán los datos del problema. La estructura de las opciones de todas las opciones es la misma, siguiendo con el objetivo del software de ser un entorno amigable.

En la figura 46 se ve las opciones de la pestaña Archivo. Desde ella se puede crear un nuevo proyecto, abrir uno ya guardado y guardar el trabajo que se está realizando.

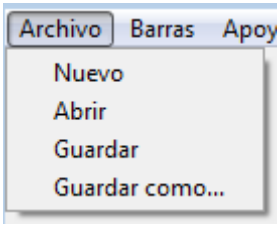


Figura 16

Barras

Ahora vamos a ver como se hace la introducción, modificación o eliminación de las barras.

En primer lugar, en la pestaña Barras se ve la opción de Nueva que nos lleva a la ventana de Nueva Barra (figura 47 y 48).

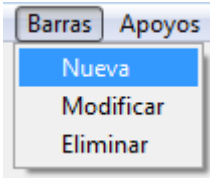


Figura 47

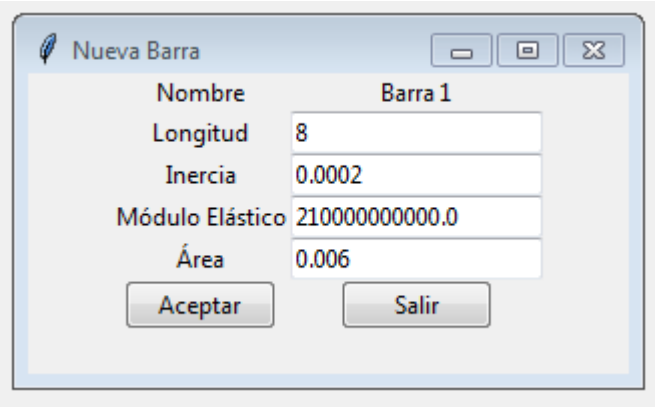


Figura 48

Los datos necesarios para las barras son su longitud, inercia, módulo elástico y su área. Al aceptar se ve así (figura 49).

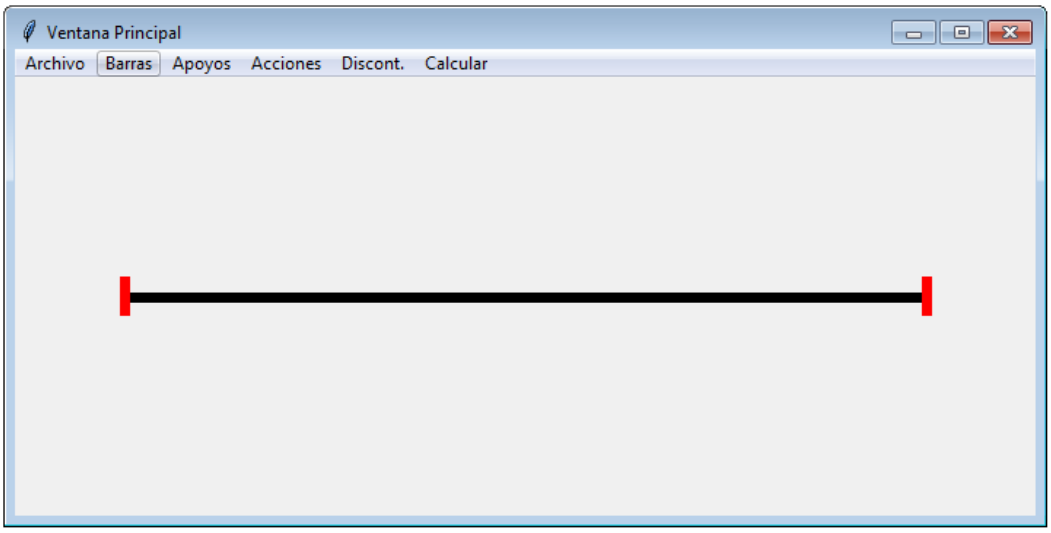


Figura 49

Al insertar más barras, la escala de la zona gráfica cambiará para una mejor visualización (figura 50).

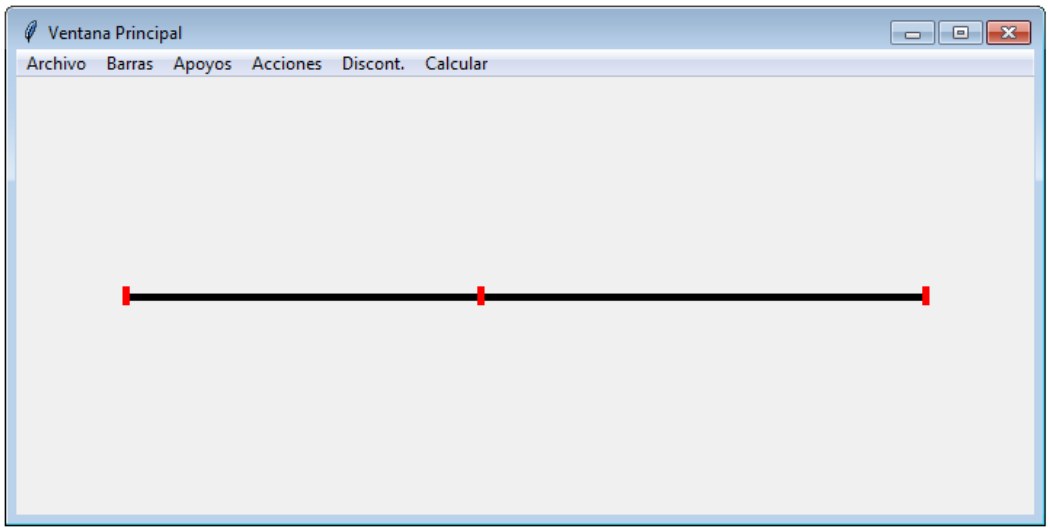


Figura 50

Si se quiere modificar cualquier barra simplemente se tiene que elegir la opción de Modificar en la pestaña de Barras (figura 51 y 52).

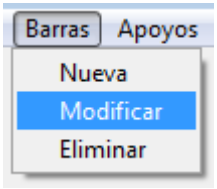


Figura 51

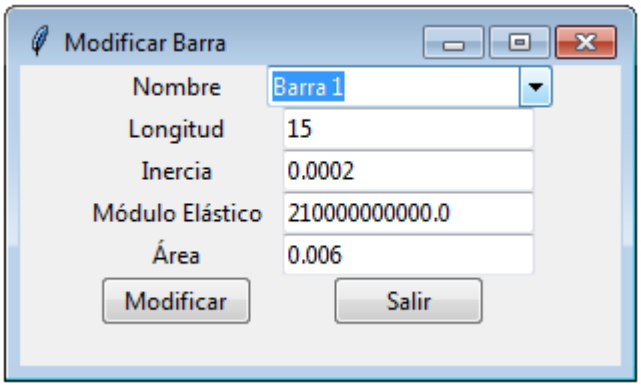


Figura 52

En el campo de nombre se selecciona la barra que se desea modificar y a continuación se cambian los datos y se modifica.

Para eliminar la barra se selecciona la opción Eliminar (figura 53 y 54), se selecciona la barra y se elimina.

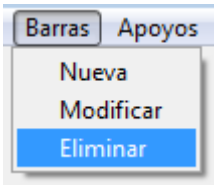


Figura 53

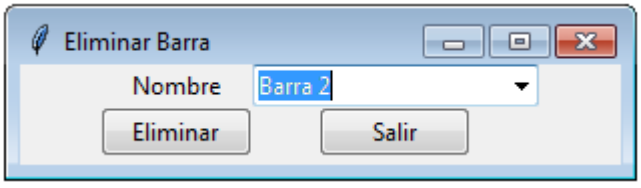


Figura 54

Apoyos

Después de las barras está la pestaña Apoyos que permite introducir las condiciones de contorno.

La ventana de Nuevo Apoyo permite introducir los valores de posición, giro del apoyo, desplazamiento en ejes x e y, giro del nudo en θ, libertades del apoyo en x, y y θ y si es elástico y de qué tipo (figura 55 y 56).

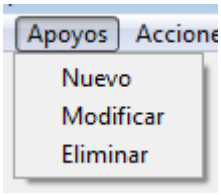


Figura 55

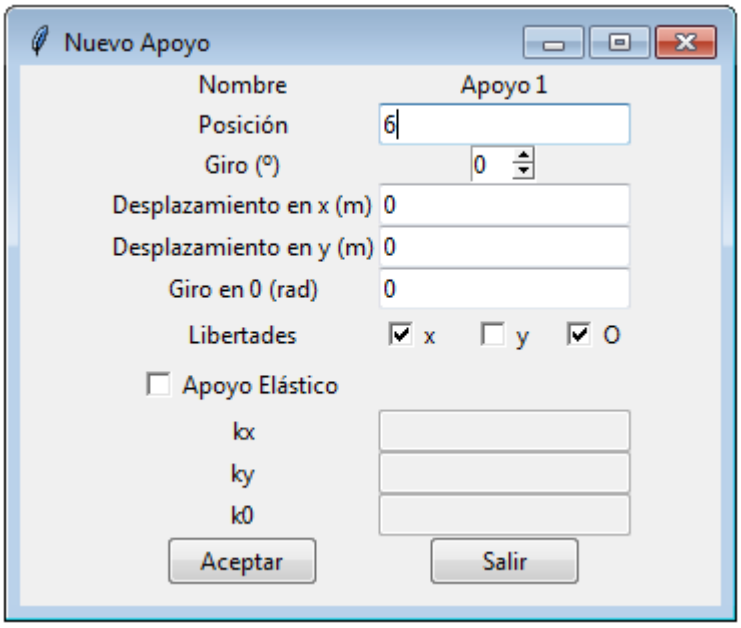


Figura 56

Y la vista del apoyo es con elementos de color verde (figura 57).

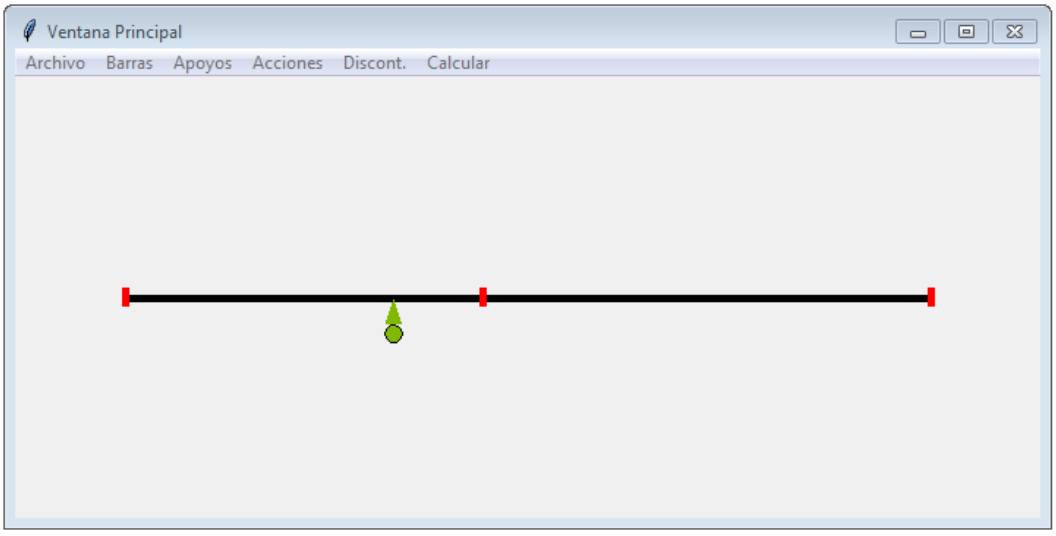


Figura 57

En el caso de que el apoyo tenga un desplazamiento impuesto, además de la representación del apoyo, aparecerá una flecha señalando el sentido de dicho desplazamiento de color azul oscuro (figura 58).

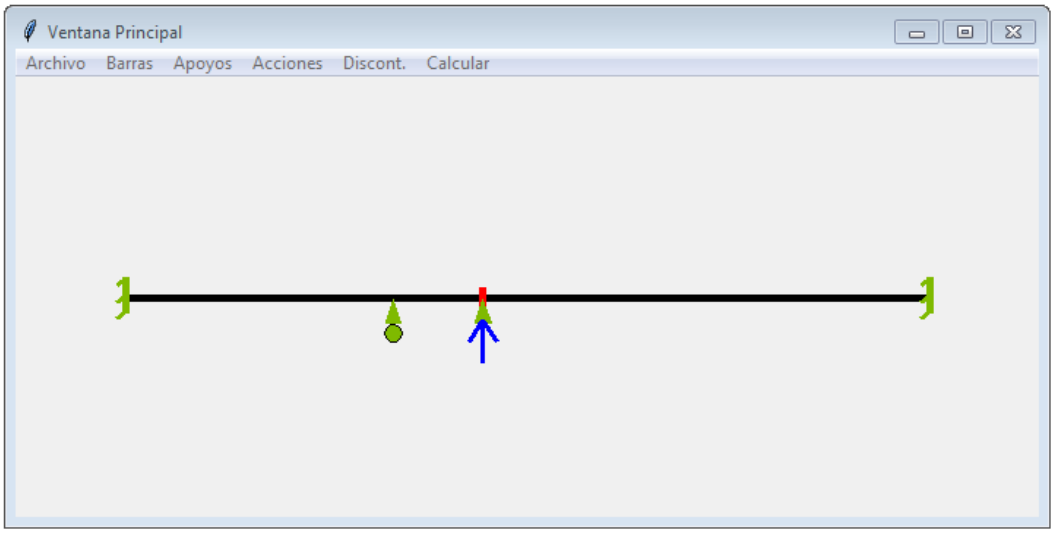


Figura 58

Para modificar el apoyo, simplemente se elige la opción Modificar en la pestaña Apoyos y aparecerá una ventana llamada Modificar Apoyo, que será igual que la de Nuevo Apoyo salvo que en el nombre aparecerá una lista desplegable con los apoyos ya introducidos (figura 59).

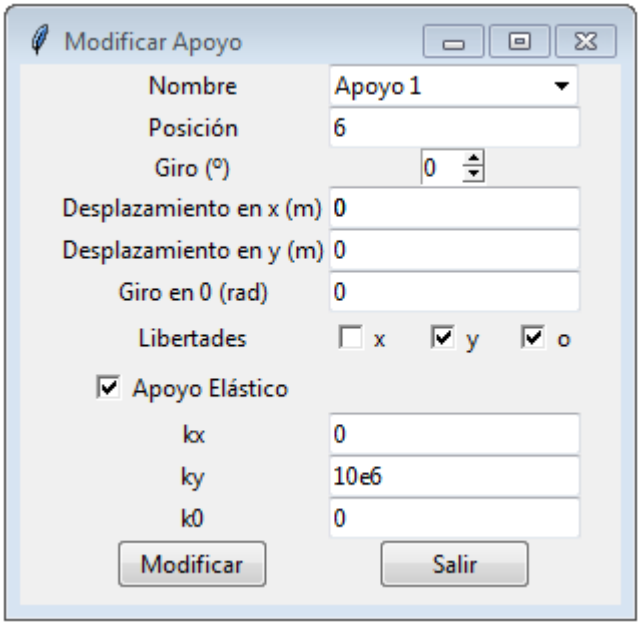


Figura 59

Para eliminar algún apoyo se elige la opción Eliminar y aparecerá la ventana Eliminar Apoyo (figura 60).

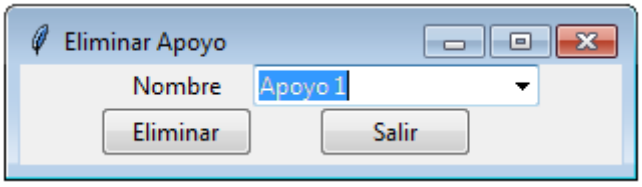


Figura 60

Acciones

La siguiente pestaña es la de Acciones, donde se podrán introducir fuerzas puntuales, linealmente distribuidas y momentos en función del problema a analizar (figura 61).

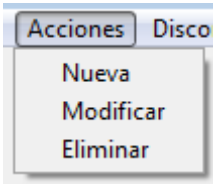


Figura 61

En la ventana Nueva Fuerza se pueden meter los datos de tipo de acción, posición y valor (figura 62).

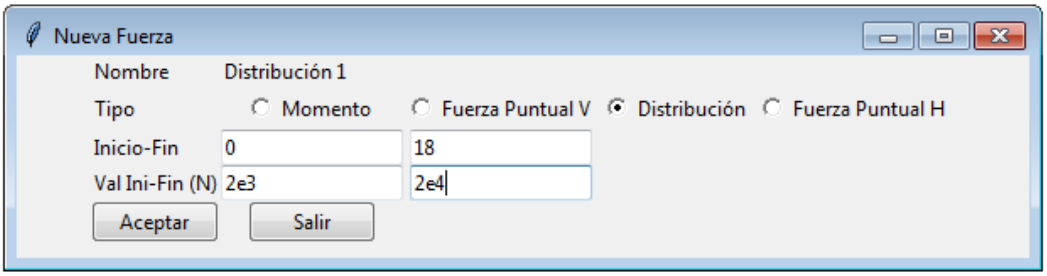


Figura 62

Y su visualización es mediante elementos de color rojo (figura 63).

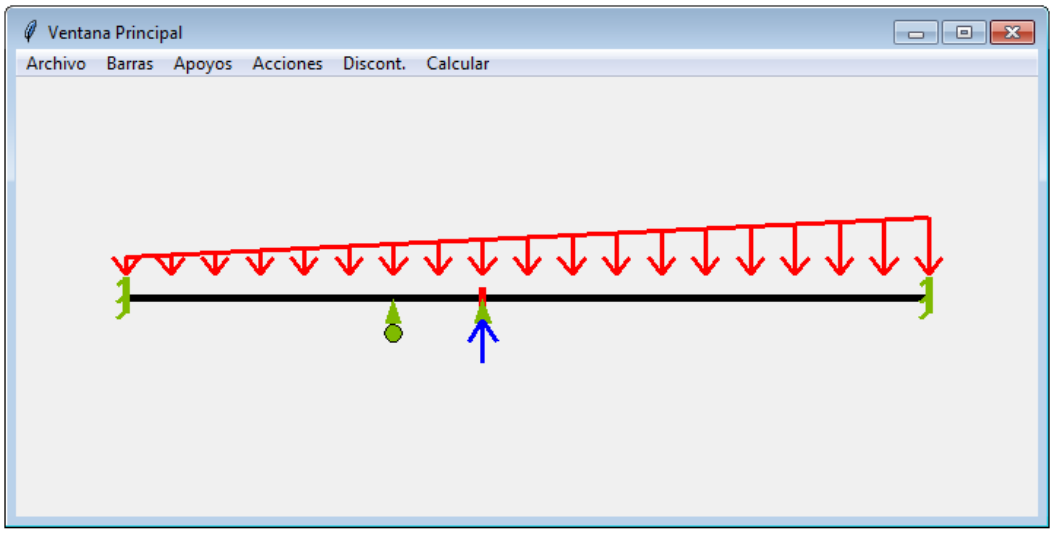


Figura 63

Para modificar las acciones, seleccionamos la opción de Modificar de la pestaña Acciones, la cual nos mostrará la siguiente ventana (figura 64).

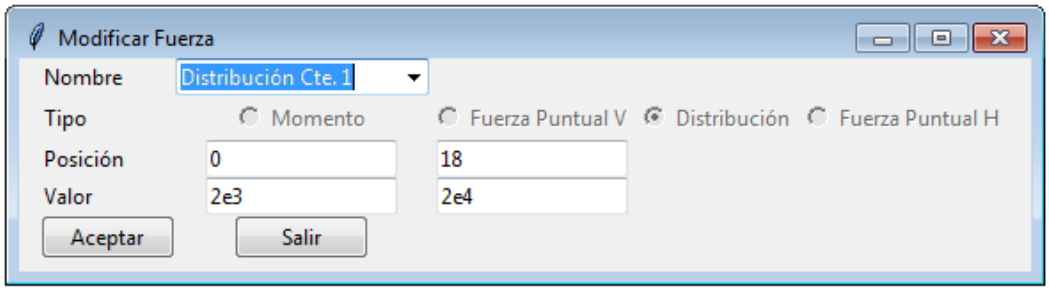


Figura 64

Como se puede ver, la opción del tipo de acción no se puede modificar. Solo se pueden modificar las opciones de posición y valor. Para cambiar el tipo de acción simplemente se elimina la acción deseada y se introduce otra nueva del tipo deseado. La ventana de eliminación de acciones es la siguiente (figura 65).

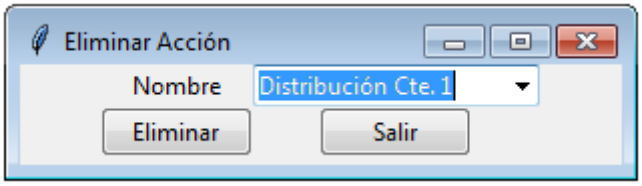


Figura 65

Discontinuidades

El último elemento para la formulación de los problemas que queda por introducir son las discontinuidades. Éstas modifican las condiciones de compatibilidad entre barras.

Como el resto de elementos, para introducir una nueva discontinuidad se debe seleccionar la opción de Nueva en la pestaña Discont. (figura 66).

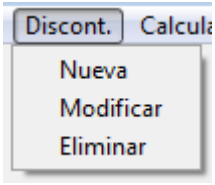


Figura 66

En la ventana Nueva Discontinuidad se puede elegir entre los tipos de discontinuidad y su posición (figura 67).

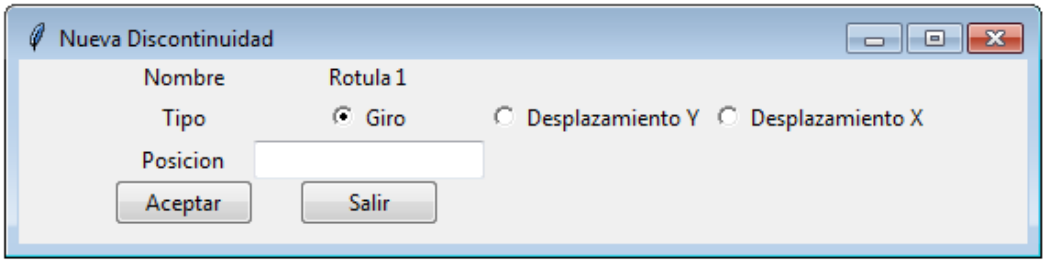


Figura 67

Su visualización se hace mediante elementos azules claro (figura 68).

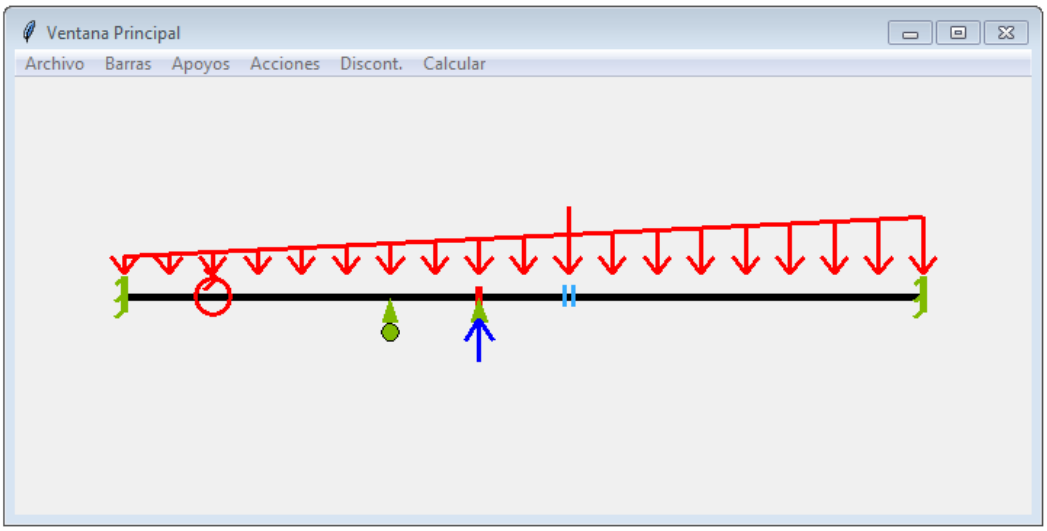


Figura 68

En cambio, al igual que lo que pasaba con las acciones, a la hora de modificar una discontinuidad, el único valor que se puede modificar es la posición, pero el tipo no (figura 69).

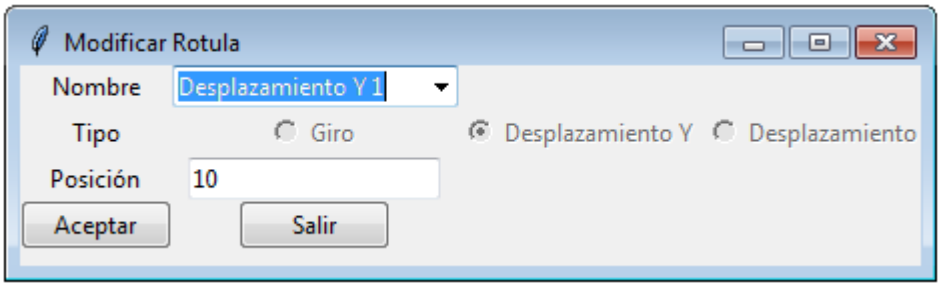


Figura 69

Y por último, para la eliminación de cualquier discontinuidad introducida, se debe elegir la opción de Eliminar en la pestaña Discont. (figura 70).

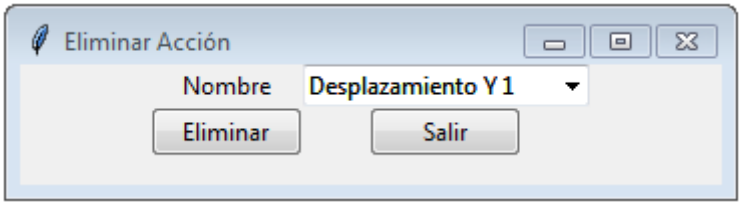


Figura 70

Cálculo

En este punto, ya están introducidos todos los datos del problema. Para resolverlo está la pestaña Calcular. Al elegir esta opción aparecerán las gráficas de los esfuerzos cortantes, momentos flectores, giros y desplazamientos de las barras.

Para la demostración, se han introducido los siguientes datos:

- Barras
  - Barra 1:

    Longitud: 8

    Inercia: 2e-4

    Módulo elástico: 2,1e11

    Área: 6e-3

- Barra 2:

    Longitud: 10

    Inercia: 5e-4

    Módulo elástico: 2,1e11

    Área: 9e-3

- Apoyos
    - Apoyo 1:

        Posición: 0

        Giro: 0

        Desplazamientos: Sin desplazamientos

        Libertades: Sin libertades

        Apoyo elástico: No
    - Apoyo 2:

        Posición: 6

        Giro: 0

        Desplazamientos: Sin desplazamientos

        Libertades: x - θ

        Apoyo elástico: No
    - Apoyo 3:

        Posición: 8

        Giro: 0

        Desplazamientos: y = 0,001

        Libertades: θ

        Apoyo elástico: No

- Apoyo 4:

  Posición: 18

  Giro: 0

  Desplazamientos: Sin desplazamientos

  Libertades: Sin libertades

  Apoyo elástico: No

- Acciones
  - Distribución 1:

    Tipo: Distribución

    Posición: 0 – 18

    Valor: 2e3 – 2e4
  - Momento 2:

    Tipo: Momento

    Posición: 2

    Valor: -5e5
  - Fuerza puntual V 3:

    Tipo: Fuerza puntual Vertical

    Posición: 10

    Valor: 8e5
- Discontinuidades
  - Desplazamiento Y 1

    Tipo: Desplazamiento Y

    Posición: 10

Y su visualización es (figura 71):

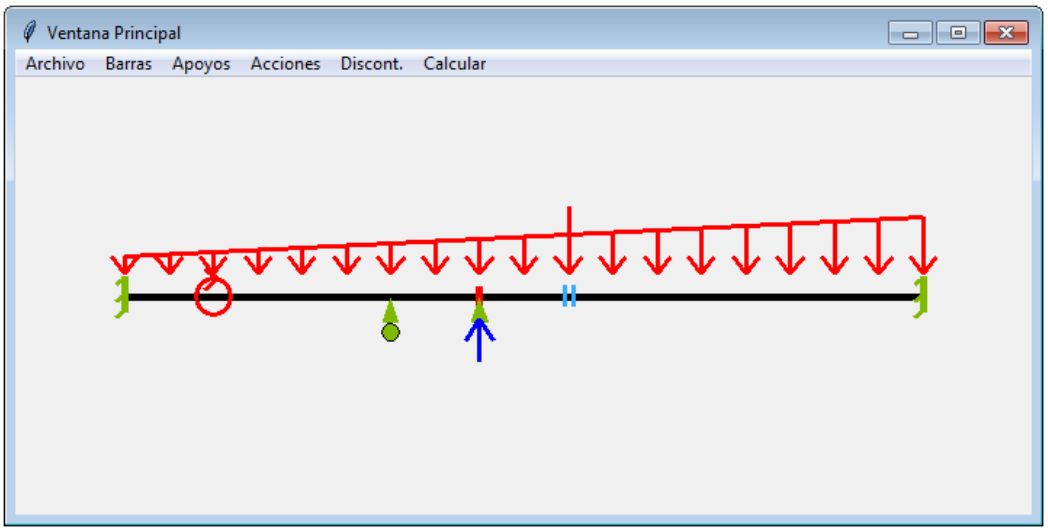


Figura 71

En el momento que se elija la opción Calcular aparecerá la siguiente ventana (figura 72).

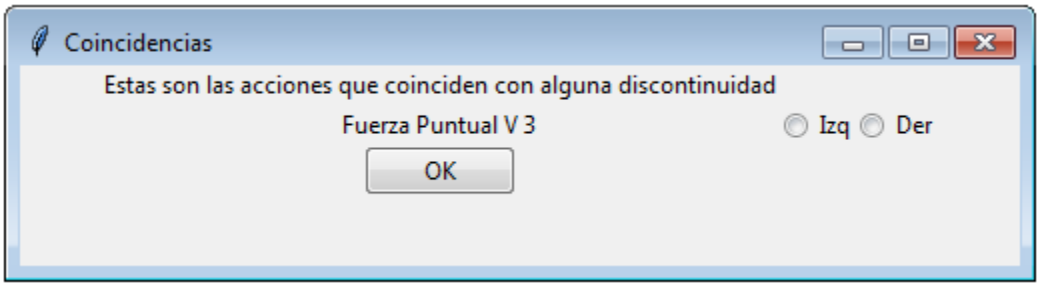


Figura 72

Se observa que en el punto 10 coinciden la Fuerza Puntual Vertical 3 con la discontinuidad Desplazamiento Y. Ocurriendo esto se necesita saber en qué lado de la discontinuidad se aplica la fuerza ya que el problema cambia sustancialmente si es aplicada a la derecha o a la izquierda. En este caso se aplicará a la izquierda. En este caso se selecciona Izq y Ok.

Al momento aparecerá una ventana mostrando los resultados (figura 73).

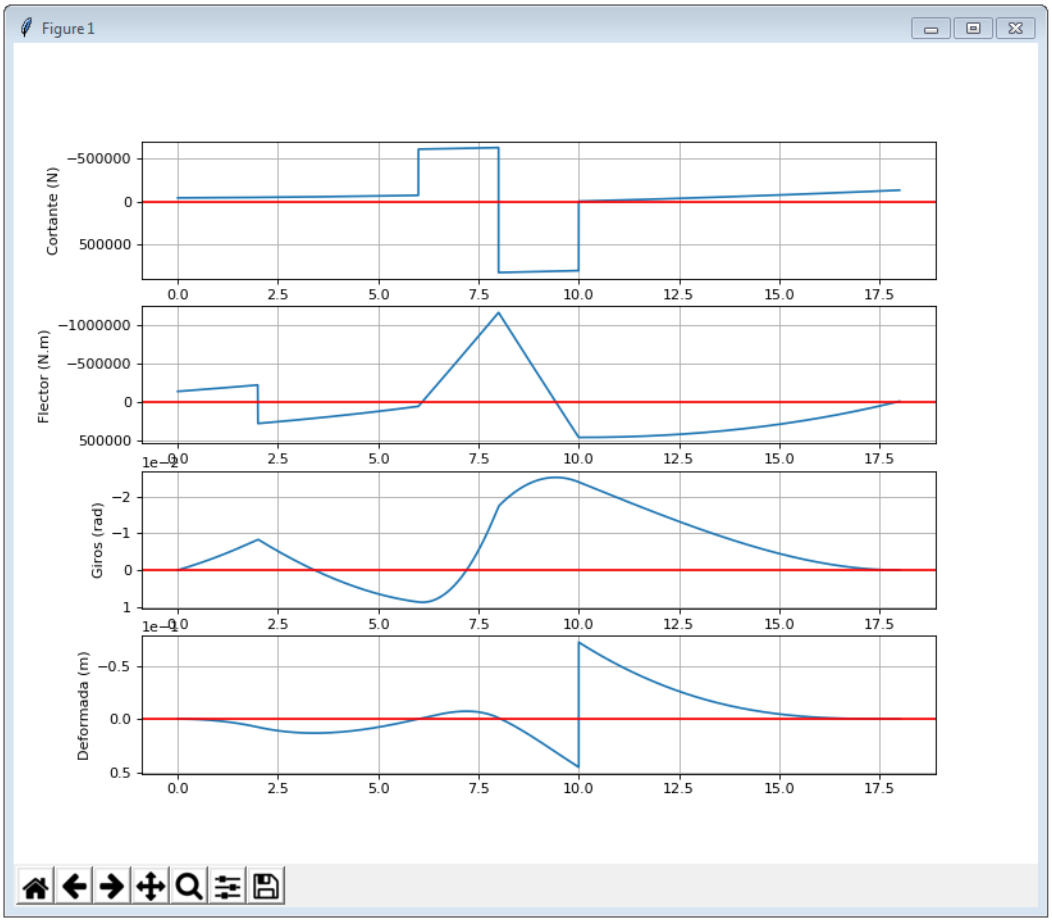


Figure 1
Cortante (N)
−500000
0
500000
Flector (N.m)
−1000000
−500000
0
500000
1e−2
Giros (rad)
−2
−1
0
1
1e−1
Deformada (m)
−0.5
0.0
0.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Figura 73

VIII. Conclusiones

A la vista del trabajo realizado, se piensa que han sido alcanzados todos los objetivos previstos.

1) El aprendizaje del lenguaje de programación Python con el que se ha logrado comprender conceptos nuevos como el de la programación orientada a objetos y con el que se ha desarrollado este software completamente.

2) Manejo de recursos con una ética de conocimiento abierto. Gracias a ello se han encontrado a disposición inmediata todos los elementos necesarios para el desarrollo del trabajo, incluyendo el lenguaje de programación, una variedad de IDEs para elegir (Integrated Development Environments, eventualmente se ha adoptado IDLE 3.6), y las librerías de utilidad que se han necesitado. Ello sin más coste que el de respetar los términos de licencias que están pensadas para compartir más que para restringir el uso.

3) El software en sí. Un software amigable que permite el cálculo de los esfuerzos internos de vigas rectas con la misma mentalidad y ética que el propio lenguaje. Se ha conseguido un software lo suficientemente robusto como para que su aprendizaje de uso no resulte difícil ni frustrante.

Licencia

Resumen de términos de licencia:

Este programa es Software Libre (Free Software). Como se le aplican los términos de la "GNU General Public License", en su versión 2 o bien (como usted prefiera) en una versión posterior. Básicamente, usted puede usar libremente el programa, realizar copias del mismo y distribuirlas libremente, estudiar su código para aprender cómo funciona, realizar modificaciones / mejoras del programa bajo las siguientes condiciones:

- Las modificaciones realizadas al programa deben hacerse públicas bajo una licencia como la presente (así el software puede mejorar con aportaciones realizadas sobre el trabajo de otros, como se hace en la ciencia).

- Las modificaciones y trabajos derivados en general deben incluir el reconocimiento al autor original (no puede decir que es usted quien ha escrito este programa). En este caso, debe mencionar al autor original como:

- Bruno Cuadrado Matas, estudiante del grado en Ingeniería Mecánica en la Escuela de Ingenierías Industriales de la Universidad de Valladolid (España).

Este programa se distribuye con la esperanza de que sea útil, pero SIN NINGUNA GARANTIA, ni siquiera la garantía de comerciabilidad o de adecuación para un propósito particular. Lea la GNU General Public License para más detalles.

IX. Webgrafía

- Alejandro Suárez Lamadrid y Antonio Suárez Jiménez. Python 3 para impacientes [Blog]. Recuperado de https://python-para-impacientes. blogspot.com/p/indice.html

- Bryan Oakley (1 de noviembre de 2010). How do I get a windows current size using Tkinter? [Entrada en blog]. Recuperado de https://stackoverflow.com/questions/4065783/how-do-i-get-a-windows-current-size-using-tkinter

- Python 3 [Blog]. Recuperado de http://acodigo.blogspot.com/p/ python.html

- John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team (8 de febrero de 2018). Customizing matplotlib [Entrada en blog]. Recuperado de https://matplotlib.org/2.1.2/tutorials/introductory/customizing.html#sphx-glr-tutorials-introductory-customizing-py

- José María Herrera Fernández, Luis Miguel Sánchez Brea (2013). 4.3. Matplotlib - Computación científica con Python para módulos de evaluación continua en asignaturas de ciencias aplicadas [Entrada en blog]. Recuperado de http://webs.ucm.es/info/aocg/python/ modulos_cientificos/matplotlib/index.html

- Nicolas Rougier, Mike Müller, Gaël Varoquaux. 1.4. Matplotlib: Gráficas usando pylab [Entrada de blog]. Recuperado de http://claudiovz.github.io/scipy-lecture-notes-ES/intro/matplotlib /matplotlib.html#valores-por-defecto

- Hektor Profe [Canal Youtube] (2 de octubre de 2016). Curso Maestro de Python 3 [Lista de reproducción]. Recuperado de https://www.youtube.com/watch?v=3gNOO_YlO8s

- JAOR SOFTWARE [Canal Youtube](22 de octubre de 2017). Curso Python [Lista de reproducción]. Recuperado de https://www.youtube.com/ user/jaorsoftware/playlists

- Jesús Conde [Canal Youtube](12 de enero de 2014). Curso Inicio práctico a la programación con Python 3.X [Lista de reproducción]. Recuperado de https://www.youtube.com/user/0utKast/playlists

- Nicolas P. Rougier. Matplotlib tutorial [Blog]. Recuperado de https://www.labri.fr/perso/nrougier/teaching/matplotlib/ matplotlib.html

Agradecimiento especial a Carmen Rada (Centro de formación Aulateknia)