XV Congreso Iberoamericano de Ingeniería Mecánica
XV Congresso Ibero-americano de Engenharia Mecânica
Madrid, España, 22-24 de noviembre de 2022

OpenBeam: Herramienta offline y online para análisis estático de estructuras

José Luis Blanco-Claraco[1], Javier López-Martínez[1], Francisco Javier Garrido-Jiménez[1], Pedro Gómez-Calvache[1], Vicente Martín-Rodríguez[1], José Manuel García-Manrique-Ocaña[2]

[1]Área de Ingeniería Mecánica, Dept. Ingeniería, Universidad de Almería, España.
Email: {jlblanco,javier.lopez,fjgarri,pgcalvac,vmartin}@ual.es
[2] Dept. de Ingeniería Civil, de Materiales y Fabricación, Universidad de Málaga, España. Email: josegmo@uma.es

Resumen

Los métodos matriciales para cálculo estático de estructuras representan uno de los más precisos y eficientes paradigmas para abordar el análisis de las estructuras más empleadas típicamente en la construcción. Con el presente trabajo se pretende llenar un nicho en el software de código abierto del ámbito de la mecánica computacional en relación a dichos métodos matriciales, aportando una librería de programación C++ y un conjunto de herramientas asociadas que permitan abordar de manera fácil el análisis estructural. Este nuevo software, denominado OpenBeam, presenta un diseño que enfatiza la orientación docente, entre otras características, mediante la fácil parametrización de las estructuras, la presentación de gráficas de diagramas y la creación de animaciones de las deformadas. Además, se ofrece una versión interactiva del software como herramienta online de acceso libre para uso en cualquier dispositivo de sobremesa o móvil sin necesidad de instalaciones, ya que se ejecuta en el propio navegador web.

Palabras clave: cálculo matricial; elementos finitos; software de innovación educativa; aplicaciones web.

Abstract

The direct stiffness matrix method for static calculation of structures represents one of the most precise and efficient paradigms to address the analysis of the structures most typically used in construction. The present work intends to fill a niche in open-source software in the field of computational mechanics in relation to said matrix methods, providing a C++ programming library and a set of associated tools that allow an easy approach to structural analysis. This new software project, named OpenBeam, presents a design that emphasizes didactic applications with, among other features: an easy parameterization of structures, the presentation of diagram graphs, and the creation of animations of the deformed structures. In addition, an interactive version of the software is offered as a freely accessible online tool for use on any desktop or mobile device without the need for installations since it runs directly on the web browser.

Keywords: Direct stiffness method; finite element methods; educational innovation software; web applications.

1. Introducción

Los métodos matriciales son ampliamente conocidos y empleados en la actualidad para el cálculo estático de estructuras típicamente empleadas en la construcción [1-2] y, mediante el método de elementos finitos, de piezas arbitrarias en dos o tres dimensiones [7]. Con el presente trabajo se pretende llenar un nicho en el software de código abierto del ámbito de la mecánica computacional en relación a dichos métodos matriciales, aportando una librería de programación C++ y un conjunto de herramientas asociadas que permitan abordar de manera fácil el análisis estructural. El software está dotado de un diseño que enfatiza la orientación docente, entre otras características, mediante la fácil parametrización de las estructuras, la presentación de gráficas de diagramas y la creación de animaciones de las deformadas.

La disponibilidad de herramientas versátiles, de código abierto y acceso universal gratuito para cálculo matricial de estructuras tendría múltiples aplicaciones. En primer lugar, como bloque reutilizable (librería de código) en la creación de herramientas de análisis estático de estructuras para proyectos de ingeniería. En el ámbito educativo, permitiría la automatización de

tareas como la creación de gráficas y animaciones de diferentes estructuras, incluyendo la generación parametrizada de estructuras y el cálculo automático de las soluciones estáticas (reacciones, deformadas y esfuerzos), facilitando por tanto la creación de material docente, incluyendo ejercicios y exámenes aleatorizados. Llevar todo esto a cabo de manera online, desde un navegador web, permitiría rebajar al máximo el umbral de acceso a la herramienta.

Existen en la actualidad numerosos software para cálculo estructural mediante el método matricial y/o por el método de elementos finitos, pero la mayoría son software privativo, sin posibilidad de acceso al código fuente. El presente trabajo presenta un software (OpenBeam) de código abierto que, además, en su forma offline es compatible con todos los sistemas operativos mayoritarios (Windows, Mac y GNU/Linux), y que en su forma online funciona desde cualquier ordenador o dispositivo móvil sin necesidad de instalación. Otra característica reseñable de OpenBeam es su capacidad para definir coordenadas nodales no concordantes con los ejes de coordenadas globales [10] para los nodos que se deseen, lo que permite definir deslizaderas en planos con inclinación arbitraria.

Se ofrece una librería que en su forma nativa está escrita en lenguaje C++17, y hace uso de la conocida librería Eigen3 [3] para la realización de todos los cálculos relacionados con matrices densas y dispersas, así como la resolución eficiente de sistemas lineales. La documentación y las herramientas online están accesibles en la web del proyecto [4] y en su repositorio en GitHub [1].

El resto de este artículo se organiza como sigue. La Sección 2 expone los materiales y métodos, la Sección 3 presenta los resultados del trabajo, y finalmente las conclusiones se resumen en la Sección 4.

2. Metodología

En esta sección se exponen las herramientas, teóricas o tecnológicas, empleadas en el desarrollo de OpenBeam.

2.1. Método matricial de la rigidez

El método matricial es un método que no está limitado por el tipo de estructura como sí pueden estarlo otros métodos de cálculo. Éste tiene la finalidad de organizar toda la información de una estructura en forma de matrices. Además, se pueden resolver un mayor número de incógnitas en comparación con métodos de resolución clásicos de la Resistencia de Materiales y con el beneficio de poder automatizarlo. Para poder aplicarlo a una estructura continua es necesario realizar una modelización de ésta por medio de un conjunto discreto y finito de variables. Existe cierta libertad por parte del ingeniero a la hora de escoger: (i) la manera en que los cuerpos continuos se dividen en una serie discreta de *elementos finitos*, y (ii) cuántos grados de libertad (gdl) tendrá cada uno de dichos elementos en los puntos de unión (*nudos*) con el resto de los elementos a los que está físicamente unido. El número total de grados de libertad independientes existentes en todos los nudos determinará el tamaño de los vectores y matrices globales del problema y, por tanto, el coste computacional de su resolución. Cabe reseñar que el empleo del método de la rigidez está motivado por la mayor facilidad de automatización, al permitir definir una librería de elementos predefinidos con matrices de rigideces con expresiones conocidas [1,7,10].

2.2. Cargas distribuidas

El estudio del problema discretizado tiene dos implicaciones: (i) se obtendrán resultados solamente en los grados de libertad definidos en el vector de estado, y (ii) únicamente se podrán definir cargas, de tipo puntuales (concentradas), en los grados de libertad definidos en los nudos. Por lo tanto, las cargas distribuidas que existan deben convertirse en equivalentes concentradas, para lo cual se emplean los métodos bien conocidos en cálculo matricial [10] usando las ecuaciones de Resistencia de Materiales [5, 8-9]. Respecto a las leyes de esfuerzos (axiles, cortantes, flectores y torsores), hay que reseñar que a las leyes de esfuerzos resultantes del cálculo matricial debidos a las cargas concentradas en los nudos (i.e. axiles y cortantes constantes y flectores lineales), hay que superponer los diagramas de esfuerzos originados propiamente por la carga distribuida en el interior del elemento. En nuestro trabajo, en lugar de implementar las ecuaciones exactas de Resistencia de Materiales para los diagramas dentro de cada elemento, se ha optado por realizar un mallado de las estructuras en elementos suficientemente pequeños, de manera que, calculando de manera rigurosa los esfuerzos en los extremos de cada elemento, se obtengan diagramas de esfuerzos lineales a trozos, que se acercarán todo lo que se desee a los reales aumentando el número de elementos. Dicho acercamiento al problema está respaldado por el principio de Saint-Venant [8] ya que cada elemento finito, tras el mallado, sufrirá en sus nodos extremos las mismas solicitaciones que su equivalente de la estructura real continua. Idéntico razonamiento se aplica al cálculo de la deformada de la estructura, que gracias al mallado no requiere de la definición de ecuaciones específicas adicionales a las soluciones obtenidas del cálculo matricial.

[1]URL: https://github.com/open-beam/openbeam

En concreto, se han implementado las siguientes cargas distribuidas, incluyendo métodos para particionarlas a la hora de mallar la estructura:

- Uniforme: carga de densidad constante a lo largo del elemento finito.
- Trapezoidal: con densidades de carga diferentes en los dos extremos del elemento finito. Este tipo incluye a la carga triangular como caso particular.
- Variación de temperatura: provoca fuerzas axiales equivalentes en los extremos del elemento.
- Puntual no nodal: se pueden definir cargas de fuerzas o momentos puntuales en puntos intermedios de un elemento.

2.3. Diagramas de esfuerzos

Para un elemento finito plano lineal tendremos seis elementos en su vector de fuerzas generalizadas, como se muestra en la Fig. 1(a). En el caso más general de un elemento espacial o tridimensional, cada nodo definirá tres fuerzas y tres momentos. El signo de dichas fuerzas debe interpretarse a la hora de generar diagramas de esfuerzos de acuerdo con un criterio determinado para reflejar los esfuerzos que sufre cada sección del material. En concreto, en este trabajo se ha empleado el criterio mostrado en Fig. 1(b).

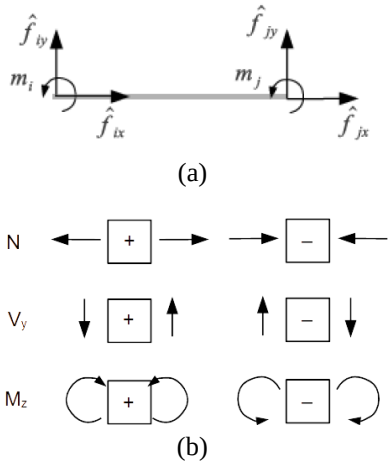


Fig. 1: (a) Elemento finito plano lineal y sus seis fuerzas definidas en los grados de libertad de sus dos nodos. (b) Criterio de signos empleado en el software para un elemento finito cuyo eje +X vaya de izquierda a derecha, para los esfuerzos axil (N), cortante (V_y) y flector (M_z).

2.4. Mallado

El mallado ("*meshing*") es el paso en el que un sólido continuo se divide en multitud de elementos finitos [7]. En nuestro trabajo solamente ha sido necesario implementar el mallado básico de elementos lineales (barras o vigas) en elementos más pequeños, también lineales, por lo que la conectividad entre elementos tras el mallado es trivial al ser puramente lineal. Únicamente hay dos aspectos reseñables:

- Un tipo de elemento (ver sección 3.2 y Tabla 1) al ser mallado se puede convertir en varios elementos de distintos tipos según los grados de libertad definidos en sus extremos (p.ej. al mallar una barra biarticulada se obtendrá un elemento articulado-rígido, varios elementos rígidos-rígidos, y finalmente un elemento rígido-articulado), y
- Las cargas distribuidas a lo largo de una viga o barra también deben "mallarse" para repartirse en los elementos finitos, que son los que finalmente son calculados.

Ambos procesos se realizan de manera automática en el software sin solicitar al usuario más parámetros que la longitud máxima deseada de los elementos tras el mallado ("resolución" del mallado), con lo que se puede tener control sobre el compromiso entre exactitud y nivel de detalle en las deformadas y leyes de esfuerzos obtenidos, por un lado, y el tiempo de cómputo requerido, por otro.

2.5. Lenguaje de programación C++

Como lenguaje de programación principal del proyecto se ha escogido C++ en su versión C++17 [11] debido a sus ventajas: portabilidad, eficiencia, orientación a objetos, y por ser uno de los lenguajes con acceso a una gran cantidad de librerías, como han sido las empleadas para los cálculos matemáticos o los gráficos.

2.6. Librería Eigen3

Eigen, en su versión 3, es una de las librerías C++ más ampliamente usadas en múltiples ramas de la ingeniería para representar y manipular matrices, vectores y tensores [3]. En nuestro trabajo se han empleado sus dos modos de almacenar matrices: densas y dispersas ("*sparse*"). Las matrices y vectores densos son aquellos cuyos elementos se almacenan, todos sin excepción, de manera consecutiva en una zona de memoria. En este modo se han representado las submatrices de rigidez de los elementos finitos, así como los vectores de coordenadas generalizadas. Las matrices dispersas, en cambio, se caracterizan por ser almacenadas en la memoria en formatos más sofisticados, especialmente optimizados para el caso en que un alto porcentaje de los elementos son ceros. Éste es el caso de las matrices de rigidez de estructuras de tamaño muy grande, especialmente tras realizar el mallado, ya que éste incrementa la cantidad de elementos. A nivel de cálculo matricial, el que las matrices sean dispersas se debe a la conectividad de los elementos [7, 10], ya que cada uno sólo está típicamente conectado a unos pocos elementos limítrofes.

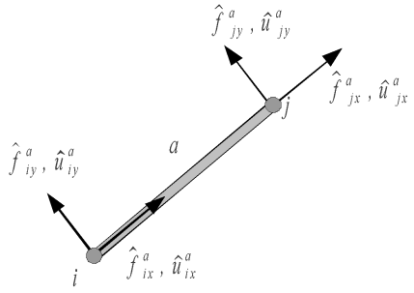
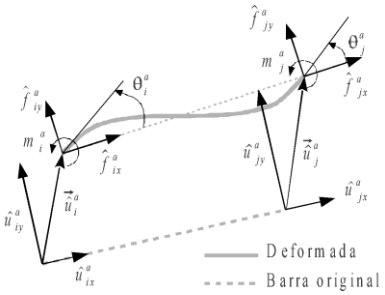
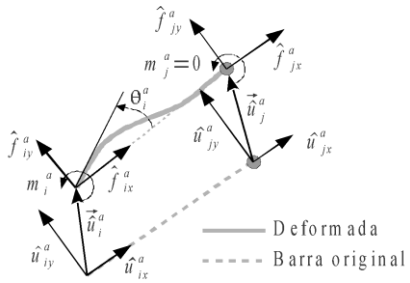
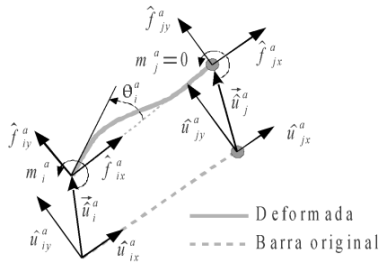
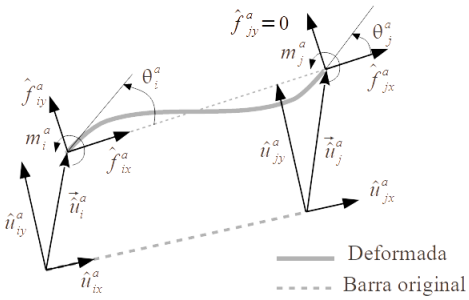
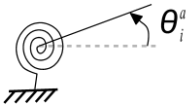
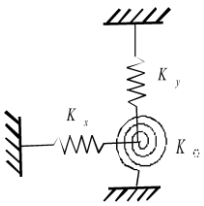
 Barra biarticulada (“BEAM2D_AA”)	 Barra biempotrada (“BEAM2D_RR”)
 Barra empotrada-articulada (“BEAM2D_RA”)	 Barra articulada- empotrada (“BEAM2D_AR”)
 Barra empotrada-deslizadera (“BEAM2D_RD”)	 Resorte lineal (“SPRING_1D”)
 Resorte de torsión (“SPRING_TORSION”)	 Triple resorte (“SPRING_DXDYRZ”)

Tabla 1: Resumen de los principales elementos finitos implementados en `libopenbeam` junto a su nombre corto a usar en el fichero de especificación de estructura. Ver discusión en la sección 3.2.

Una vez se tienen matrices representadas en forma dispersa, la resolución de sistemas lineales tipo canónico **Ax**=**b** (con **A** la matriz dispersa, **x** el vector de incógnitas y **b** los términos independientes) como los necesarios en el cálculo estático de estructuras requiere de algoritmos especialmente diseñados para conseguir explotar la dispersión de las matrices y resolverlos en tiempo típicamente casi lineal O(N) en lugar de cúbico O(N^3), con N el número de grados de libertad.

En OpenBeam se requiere resolver el sistema lineal $F_f = K_{ff}U_f$, donde U_f es el vector de desplazamientos desconocidos en los gdl libres, F_f el vector de las cargas (conocidas) en los gdl libres, y K_{ff} es la parte de la matriz de rigidez completa de la estructura para las columnas e índices correspondientes a los gdl libres. La matriz K_{ff}, así como la matriz de rigidez global del problema, se construyen en memoria siempre en forma de matrices dispersas por eficiencia de almacenamiento. En la librería se han implementado dos algoritmos para resolver dicho sistema lineal: (i) el algoritmo de descomposición de Cholesky (LL^T) para K_{ff} convertida en matriz densa, y (ii) Cholesky para el caso de matriz dispersa pura [3]. Por defecto, se emplea la versión para matrices densas, ya que a pesar de tener un orden computacional *asintóticamente* mayor que su versión dispersa, en la práctica la mayoría de las estructuras analizadas serán de tamaño lo suficientemente pequeño para que el método disperso no sea computacionalmente ventajoso.

2.7. Librería Cairo

La librería *Cairo*[2] es parte del proyecto GTK+ de interfaces gráficas ampliamente usadas en entornos GNU/Linux, y ofrece una API en C para generación de gráficos vectoriales. En este proyecto se ha usado para la generación de gráficos de estructuras en formato vectorial SVG en la herramienta de línea de comando `ob-solve` (presentada en la sección 3.4).

2.8. Librería mrpt-opengl

MRPT ("*Mobile Robot Programming Toolkit*") es un proyecto de software libre que ofrece librerías C++ con algoritmos y herramientas para la programación de robots móviles [12]. Su módulo `mrpt-opengl` ofrece una librería para la generación de gráficos tridimensionales de manera modular mediante ensamble y composición de primitivas visuales básicas (líneas, puntos, cilindros, etc.), por lo que ha sido escogida para generar y actualizar de manera dinámica las visualizaciones de estructuras en la versión on-line de la aplicación (ver sección 3.5).

2.9. Emscripten

Emscripten [13] es un proyecto, presentado en 2011 y en continuo desarrollo, que proporciona una versión modificada del compilador `clang` capaz de realizar compilaciones cruzadas desde varios lenguajes (incluyendo C++) a *JavaScript* y *WebAsm*, los dos lenguajes fundamentales para la creación y ejecución de aplicaciones web que se ejecutan dentro del mismo navegador. Compilando las librerías MRPT, Eigen3 y OpenBeam con Emscripten, se han podido desarrollar aplicaciones web en *Javascript* que hacen uso de todas las funciones de alto nivel expuestas en OpenBeam, como son analizar un fichero de definición de estructura, realizar el análisis estático de fuerzas y desplazamientos de la misma, o generar y actualizar su representación gráfica en un *canvas* WebGL de html5 [14], compatible con todos los navegadores web modernos en ordenador de escritorio y dispositivos móviles. *Emscripten* es, por tanto, la clave para la posibilidad de ejecución de nuestras aplicaciones web de manera distribuida directamente en el dispositivo del usuario, sin necesidad de ningún tipo de procesamiento en la infraestructura de los servidores, lo que permite un número de usuarios simultáneos ilimitado.

2.10. Lenguaje YAML

Entre los formatos de intercambio de datos más ampliamente usados en las últimas décadas se encuentra YAML [6]. El lenguaje en sí es totalmente genérico, por lo que se suele usar tanto para especificar parámetros y opciones en todo tipo de aplicaciones, como para almacenar bases de datos completas. Una de las ventajas fundamentales que tiene es que se almacena en ficheros de texto plano, fácilmente legibles e interpretables. Además, su naturaleza jerárquica permite anidar estructuras de datos de manera arbitrariamente compleja unas dentro de otras. En nuestro proyecto se usa YAML como formato para los ficheros de definición de problemas de cálculo de estructuras, según lo descrito en la sección 3.3.

3. Resultados

3.1. Librería C++ openbeam

La funcionalidad principal desarrollada en este trabajo está integrada en una librería C++, de la que hacen uso las aplicaciones en sí mismas, y que puede ser empleada por los usuarios interesados en crear sus propios proyectos que requieran de resolución de problemas de cálculo estáticos de estructuras.

[2] Disponible en https://www.cairographics.org/

A continuación, se resumen brevemente algunas de las clases C++ más importantes para entender la manera modular en que se ha creado el proyecto:

- `openbeam::CElement`: Es una clase virtual pura que define la interfaz que deben implementar obligatoriamente todos los elementos finitos. Su API obliga a definir, para cada elemento, su orden (a cuántos nodos se conecta), qué grados de libertad usa en cada uno de dichos nodos y las matrices de rigidez entre cada par de dichos nodos. Así mismo, permite cargar los parámetros que requiere cada elemento desde una sección en YAML, generar una visualización de sí mismo (tanto en formato SVG como en gráficos OpenGL), y mallar el elemento.
- `openbeam::CBaseElementBeam`: Hereda e implementa parcialmente `CElement` para todos los elementos de tipo barra o viga. Se deja para las clases hijas finales de cada tipo la definición de las matrices de rigidez, pero aquí ya se implementan cuestiones comunes como la representación gráfica, el mallado en elementos viga más pequeños y la definición de parámetros comunes como el módulo de elasticidad, etc.
- `openbeam::CFiniteElementProblem`: La clase principal de la librería, que almacena y permite definir la geometría, restricciones, elementos, y cargas de un problema dado. Aquí se implementan las funciones de cálculo estático de reacciones y desplazamientos, la interpretación de bloques YAML para cargar un problema desde un fichero, así como su representación visual.
- `openbeam::CStructureProblem`: Esta clase hereda de `CFiniteElementProblem` y añade dos funcionalidades adicionales: el mallado de las vigas/barras en elementos finitos más pequeños, y la definición de cargas distribuidas.
- `openbeam::CLoadOnBeam`: La clase base virtual pura que define la interfaz que deben implementar los distintos tipos de cargas no nodales. El principal punto por definir en las clases hijas es el cálculo de las tensiones o esfuerzos a sumar a los obtenidos del cálculo matricial (típicamente calculados según los principios y ecuaciones de Resistencia de Materiales), y cuyos valores en sentido opuesto (por el principio de acción y reacción) se transmiten como cargas nodales equivalentes a los nodos del elemento finito. Hay que recordar que el método matricial, por sí mismo, no es capaz de interpolar las deformadas ni los diagramas de esfuerzos, motivo por el cuál se realiza el mallado descrito en la sección 2.4.

Un listado exhaustivo de clases C++ y su API se encuentra disponible online en la web del proyecto.

3.2. Elementos finitos implementados

Aunque el software desarrollado permite elementos y estructuras genéricas en tres dimensiones, en la actualidad todos los elementos finitos implementados son de tipo eminentemente lineal (tipo barra o viga), o de tipo resorte de los diferentes tipos que pueden resultar útiles en problemas de estructuras planas. Esto es así por el carácter eminentemente didáctico con el que se ha enfocado el software hasta el momento, y porque se suele trabajar con estructuras planas en las asignaturas de Cálculo de Estructuras y Resistencia de Materiales.

La Tabla 1 resume los elementos finitos implementados y necesarios para definir estructuras planas. Como se ve, todos son binarios (conectan únicamente dos nodos) y cada uno hace uso de un número variable de grados de libertad en cada uno de sus dos extremos. Más detalles sobre los elementos finitos implementados, su lista de parámetros y sus matrices de rigidez, se pueden encontrar en la página correspondiente de la documentación on-line del proyecto[3].

3.3. Formato de definición de estructuras en YAML

Todos los detalles sobre la sintaxis y formato en que deben especificarse los problemas a resolver están disponibles en la documentación online del proyecto. A continuación, se discutirán brevemente las propiedades más importantes de dicho formato en base a los ejemplos mostrados en las Figuras 2 y 3.

Como se puede ver, tanto las dimensiones de la estructura como el valor de las cargas se han parametrizado a través de variables definidas en una primera sección (opcional) llamada "`parameters`". Hay que señalar que, tanto en esta sección como en el resto, cada vez que se necesita especificar un valor numérico se puede hacer uso de expresiones matemáticas (operaciones algebraicas, funciones trigonométricas, etc.) así como incluso de construcciones más complejas como "`if…then…else`" si el usuario lo requiere (p.ej. para definir que una carga sólo exista si una longitud cumple determinado criterio).

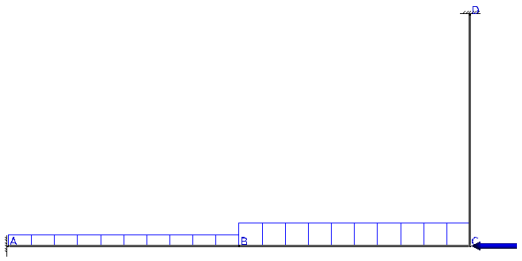
A continuación, y de manera opcional, el usuario puede definer en el bloque "`beam_sections`" las propiedades de los perfiles y materiales a emplear en la estructura.

La geometría del problema se define por medio de tres bloques: "`nodes`" con la cantidad y posición espacial de los nudos del problema, "`elements`" donde se definen qué barras o elementos existen entre qué nudos (se podrían definir elementos ternarios o de orden

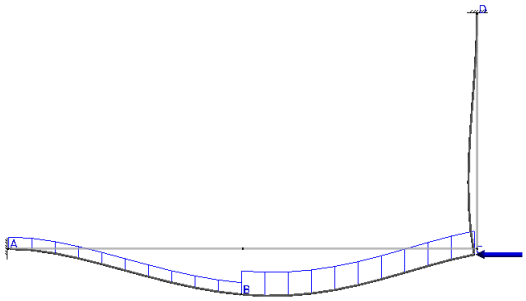
[3] Documentación sobre elementos finitos implementados: https://open-beam.github.io/openbeam/finite_elements.html

```yaml
# ------------------------------------------------
#  Parameters
# ------------------------------------------------
parameters:
  G: 9.81
  L: 1          # Dimensions of the problem
  P: G * 1000   # External load

# ------------------------------------------------
# beam sections
# ------------------------------------------------
beam_sections:
- name: IPE200
  E: 2.1e11     # Young module
  A: 28.5e-4    # Area
  Iz: 1940e-8   # Second moment of area in z

# ------------------------------------------------
#  Geometry: nodes, elements & constraints
# ------------------------------------------------
nodes:
- {id: 0, coords: [0   , 0], label: A}
- {id: 1, coords: [L   , 0], label: B}
- {id: 2, coords: [2*L , 0], label: C}
- {id: 3, coords: [2*L , L], label: D}

elements:
- {type: BEAM2D_RR, nodes: [0, 1], section: IPE200}
- {type: BEAM2D_RR, nodes: [1, 2], section: IPE200}
- {type: BEAM2D_RR, nodes: [2, 3], section: IPE200}

# Constraints
constraints:
- {node: 0, dof: DXDYRZ}
- {node: 3, dof: DXDYRZ}

# ------------------------------------------------
# Loads
# ------------------------------------------------
node_loads:
- {node: 2, dof: DX, value: -P}

element_loads:
- {element: 0, type: DISTRIB_UNIFORM, q: 1000*G, DX:
0, DY: -1, DZ: 0}
- {element: 1, type: DISTRIB_UNIFORM, q: 2000*G, DX:
0, DY: -1, DZ: 0}
```

(a)



(b)



(c)

Fig. 2: (a) Ejemplo del lenguaje de definición de estructuras de OpenBeam en YAML. Representaciones visuales de dicha estructura en su estado (b) original sin deformar y (c) deformado.

superior), y finalmente "`constraints`" con la enumeración de qué grados de libertad de qué nudos tienen su movimiento impedido. Opcionalmente, en este último bloque se puede imponer un desplazamiento distinto de cero para analizar las tensiones y deformaciones provocadas.

Las solicitaciones externas vienen definidas por dos bloques: "`node_loads`" para cargas puntuales concentradas en nudos, y "`element_loads`" para cargas distribuidas (ver sección 2.2).

3.4. Aplicación de línea de comando ob-solve

Haciendo uso de la librería C++ se ha desarrollado un programa llamado "`ob-solve`" sin interfaz gráfica para su uso en modo línea de comando. El programa admite más de una treintena de argumentos y flags en su línea de comando lo que ofrece una gran flexibilidad a la hora de:

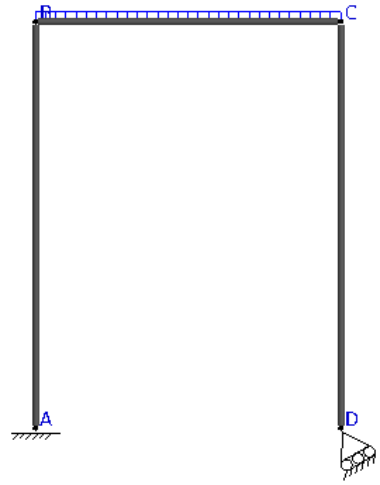
- Cargar una estructura de un fichero YAML.
- Mostrar los resultados del análisis estático por consola, o a un fichero HTML para su visualización desde un navegador web.
- Mostrar las matrices de rigidez de cada elemento, la global de la estructura completa, o las submatrices correspondientes a determinados tipos de gdl (libres y restringidos).
- Generar visualizaciones de la estructura en su estado original, deformado (con escala determinada automáticamente o dada por el usuario), o una animación. En todos los casos, las figuras se generan en formato vectorial (SVG), apto para ser empleado en publicaciones, informes, etc.
- Uso o no de mallado, y su resolución espacial.
- Cálculo de los esfuerzos a que está solicitada cada barra, y su representación en forma tabulada en texto, o como gráficas.

A modo de ejemplos de los resultados de este programa se proporciona una colección de problemas en repositorio on-line[4].

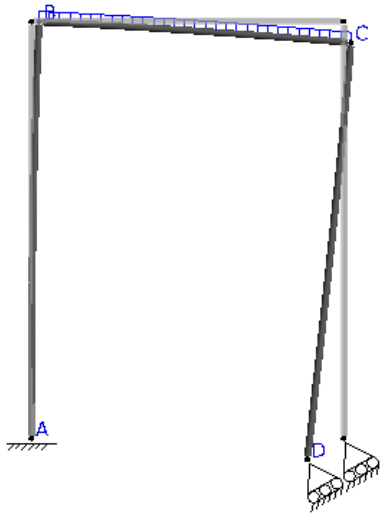
[4] URL: https://ingmec.ual.es/openbeam/fem/

```yaml
# ------------------------------------------------
#  Parameters
# ------------------------------------------------
parameters:
  G: 9.81
  L: 3.0
  H: 4.0

# ------------------------------------------------
# beam sections
# ------------------------------------------------
beam_sections:
- name: IPE200
  E: 2.1e11     # Young module
  A: 28.5e-4    # Area
  Iz: 1940e-8   # Second moment of area in z

# ------------------------------------------------
#  Geometry: nodes, elements & constraints
# ------------------------------------------------
nodes:
- {id: 0, coords: [0   , 0], label: A}
- {id: 1, coords: [0   , H], label: B}
- {id: 2, coords: [L   , H], label: C}
- {id: 3, coords: [L   , 0], label: D, rot_z: 30.0}

elements:
- {type: BEAM2D_RR, nodes: [0, 1], section: IPE200}
- {type: BEAM2D_RR, nodes: [1, 2], section: IPE200}
- {type: BEAM2D_RR, nodes: [2, 3], section: IPE200}

# Constraints
constraints:
- {node: 0, dof: DXDYRZ}
- {node: 3, dof: DY}

# ------------------------------------------------
# Loads
# ------------------------------------------------
element_loads:
- {element: 1, type: DISTRIB_UNIFORM, q: 1000*G, DX:
0, DY: -1, DZ: 0}
```

(a)



(b)



(c)

Fig. 3: (a) Otro ejemplo de estructura definida en el lenguaje YAML de OpenBeam, en este caso ilustrando el uso de coordenadas nodales no concordantes para el nudo "D". Nótese la definición del ángulo de rotación en la definición del nudo número 3, en la sección "*nodes*". Representaciones visuales de dicha estructura en su estado (b) original sin deformar y (c) deformado.

3.5. Aplicación on-line web-ob-solve

Si la aplicación de línea de comandos ob-solve está orientada, principal pero no únicamente, a su uso para generación de gráficas de leyes de esfuerzos y deformadas, la versión on-line de la misma (ver [4]) es más interactiva y está pensada para ser usada por parte de estudiantes y profesores de la forma más intuitiva posible. Las opciones más importantes se pueden configurar directamente en la interfaz gráfica, se puede seleccionar cargar uno de varios ejemplos a modo de tutorial, y se ofrecen enlaces de ayuda que dan acceso directo al usuario a la documentación sobre el formato de ficheros para definición de estructuras en YAML, los tipos de elementos y cargas, etc.

4. Conclusiones

Con este trabajo se han conseguido varios objetivos: en primer lugar, la publicación de una nueva librería de código abierto (*libopen-beam*) que ofrece una interfaz de alto nivel para la definición de estructuras mecánicas, así como la resolución del problema de cálculo estático de las mismas. También se ofrecen dos herramientas listas para su uso sin necesidad de programar: (i) el programa de línea de comando con potencial tanto para los estudiantes como para preparación de material por parte de los docentes, y (ii) la página web con la herramienta on-line, lo que lo convierte, hasta donde saben los autores, en la primera aplicación capaz de correr en web y dispositivos móviles que permite definir y calcular estructuras

arbitrariamente complejas, de manera totalmente gratuita y con código abierto.

Como trabajos futuros, (1) se añadirán nuevos tipos de elementos que permitan calcular esfuerzos de tipo momento torsor, (2) se expondrá en la versión on-line el cálculo y representación de estructuras espaciales, y (3) se añadirá mayor número de ejemplos en la aplicación web.

5. Referencias

[1] Rubinstein, M. F. (1966). Matrix computer analysis of structures (Book on computer matrix analysis of structures). ENGLEWOOD CLIFFS, N. J., PRENTICE-HALL, INC., 1966. 402 P.

[2] Ghali, A., Neville, A. M., & Brown, T. G. (2017). Structural Analysis: A unified classical and matrix approach 6th edition. CRC Press.

[3] Gaël Guennebaud, Benoît Jacob, et al. (2010). Eigen v3, https://eigen.tuxfamily.org/, último acceso 2022/03/20.

[4] OpenBeam Homepage, https://open-beam.github.io/openbeam/, último acceso 2022/03/20.

[5] Blanco-Claraco, J. L., Garrido Jiménez, F. J. López Martínez, J., Jiménez Alonso, J. F., Hernández Díaz, A. M.: Resistencia de materiales: Resumen de teoría y problemas resueltos (Vol. 7). Editorial Universidad Almería (2016).

[6] Ben-Kiki, O., Evans, C., & Ingerson, B. (9 de 2009). YAML Ain't Markup Language (YAML) Version 1.2. Tech. rep., YAML.org. Obtenido de http://www.yaml.org/spec/1.2/spec.html

[7] Liu, G. R., & Quek, S. S. (2013). The finite element method: a practical course. Butterworth-Heinemann.

[8] Ortiz-Berrocal, L. Resistencia de materiales. McGraw-Hill, 2007.

[9] Vázquez, M. Resistencia de Materiales (4ª ed.). Noela, 2008.

[10] Blanco-Claraco, J.L., González, A. and García-Manrique-Ocaña, J.M. (2012). Análisis estático de estructuras por el método matricial. Servicio de Publicaciones e Intercambio Científico de la Universidad de Málaga.

[11] Smith, R. (2017). Working Draft, Standard for Programming Language C++ N4659. Google Inc, 03-21.

[12] Blanco-Claraco, J.L. et al. (2022), Mobile robot programming toolkit (MRPT), https://www.mrpt.org, último acceso 2022/09/11.

[13] Zakai, A. (2011). Emscripten: an LLVM-to-JavaScript compiler, Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion, pp. 301-312.

[14] Parisi, T. (2012). WebGL: up and running. O'Reilly Media, Inc.