Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

Dinámica de barras: Caracterización experimental, formulación computacional en Python y calibrado aplicado a una viga biapoyada

Autor:
D. Álvaro Pascual García

Tutor:
D. Antolín Lorenzana Ibán

Valladolid, septiembre, 2018

Resumen

Realizar pruebas de carga en una estructura para obtener su respuesta y características dinámicas no siempre es una tarea sencilla ya que muchas veces la realización de test en el terreno puede ser demasiado compleja. Es en este campo donde la modelización de estructuras juega un papel fundamental, permitiéndonos obtener las frecuencias propias, respuestas modales y respuestas en frecuencia para unas condiciones determinadas. Esta caracterización es especialmente útil cuando hablamos de puentes o pasarelas, ya que nos permite anticipar su comportamiento y estudiar la viabilidad del proyecto, ahorrando tiempo y dinero.

El objetivo de este TFM es mostrar cómo se realiza el proceso de creación y validación de un modelo cuyo comportamiento puede extrapolarse al de una pasarela, partiendo desde cero de un programa en *Python*.

Puesto que los temas tratados en este trabajo no se ven en profundidad en el grado ni en el máster de industriales se ha elegido una viga biapoyada, la cual consta de una geometría sencilla y facilita la comprensión del principio del *model updating*.

Al final del documento se incluyen los resultados obtenidos y conclusiones, junto con una serie de propuestas y ejes de ampliación que pueden resultar interesantes para otro trabajo futuro.

Key words: Modelización de estructuras, frecuencias propias, respuesta modal, respuesta en frecuencia, Python.

Abstract

Testing a structure in order to obtain information about its dynamic properties it is not an easy task. Most empiric test are difficult to carry out because of its complexity. It is in this field where computer models play an essential role, making possible to obtain mode frequencies, modal responses and frequency responses for given conditions.

Nowadays, there is a wide range of softwares that make these simulations possible. They are becoming easier to use when it comes to their interface and they give good solutions for very complex structures. The principal aim of this post degree project, according to its didactic characteristics, is to explain to the user, with a python code, how those finite element programs work and which mathematics calculations have done to achieve the results.

As the subject of this memory was not deeply covered during the degree or the master, a simply supported beam has been chosen as the structure to model.

The creative nature of programming makes possible to any next student to edit, complete or profit of this work. Some improving axis are proposed with this purpose.

Key words: Computer model, mode frequencies, modal response, frequency response, python.

Índice

1.Introducción.

1.1 Motivación

La tendencia actual de construcción se orienta hacia el diseño de estructuras cada vez más impresionantes y, por lo tanto, complejas. Esto representa un reto no solo para los diseñadores, sino también para los ingenieros que realizan los estudios de límite de servicio, y eligen los materiales para que la estructura sea funcional. Esto ocurre en gran medida en pasarelas y puentes, cuya disposición es similar a la que se trata en este TFM.

A fin de estudiar la viabilidad de este tipo de proyectos, es de vital importancia comprobar antes, de una forma efectiva, que la estructura cumple con los requisitos y responde satisfactoriamente a las cargas a las que se le va a someter. Con esta idea en mente se desarrollan modelos y sistemas de cargas que tratan de parecerse lo más posible a cómo sería la estructura una vez construida.

Recurrir a estos métodos es muy beneficioso ya que nos permite ahorrar tiempo y dinero, aparte de descartar aquellas propuestas que puedan tener un comportamiento dudoso frente a las cargas. Sin embargo, no hay que pensar que el desarrollo de estos modelos es por ordenador es barato. De hecho puede ser bastante costoso en términos de tiempo y capital humano si hablamos de modelos muy complejos.

El objetivo de este TFM es mostrar cómo se realizaría este proceso de creación y validación de un modelo para una viga biapoyada, que puede extrapolarse a una pasarela, partiendo desde cero con un programa en *Python*.

Primeramente se caracteriza la viga del laboratorio para obtener sus características dinámicas. Posteriormente se crea el modelo en *Python*, discretizando la viga en numerosos elementos más pequeños. Este es el principio de la simulación por elementos finitos. A continuación se ensamblan estos elementos y finalmente se verifica y valida para correlar los resultados de la simulación con el comportamiento real.

Para llevar a cabo esta fase de verificación y validación del modelo, lo sometemos a un "*Finite element model updating*", cuyo proceso consta de cuatro etapas:

- Elegir el dominio en el que se van a presentar los datos: Tiempo, frecuencia, modal o tiempo-frecuencia.
- Determinar qué partes del modelo inicial tienen mayor tendencia de no estar bien modelados correctamente.
- La tercera tarea es formular una función que expresa la diferencia entre el valor real obtenido experimentalmente, y el que obtenemos mediante el modelo.
- Por último, se implementa método de optimización, identificando parámetros que minimizan esta función.

Finalmente debemos obtener un modelo que representa fiablemente el comportamiento de la viga, de forma que podemos editarlo a nuestra conveniencia para comprobar cómo le afectan otros sistemas de cargas.

1.2 Objetivo

El objetivo principal de este Trabajo de Fin de Máster consiste en el modelado de un simulador interactivo, que nos muestra la respuesta en vibración de una viga sometida a una cierta carga, de la cual podemos cambiar algunos parámetros para estudiar su influencia.

De entre todos los softwares disponibles para la modelización de la viga, he elegido utilizar un freeware llamado *Python*. El motivo principal de utilizar este lenguaje es que es de acceso libre, y por lo tanto gratuito para el usuario. Además, se trata de un lenguaje de alto nivel, con una gran cantidad de librerías a nuestra disposición, así como tutoriales en internet, lo cual facilita bastante la tarea.

La viga que modelo en *Python* tiene una sección rectangular constante. Las respuestas estáticas en base a las cuales se compara el modelo son obtenidas mediante varios ensayos, golpeando la viga en diferentes puntos, y recuperando la aceleración de los nudos con acelerómetros.

El presente trabajo, a parte de su cometido principal, tiene un gran valor formativo, pudiendo servir de herramienta a futuros estudiantes para comprender cómo se realiza el cálculo de elementos finitos y permitiendo futuras ampliaciones

1.3 Organización del documento

La memoria de este trabajo se puede dividir en cuatro partes. En la primera se explica cómo se lleva a cabo la obtención de los datos en el laboratorio y se describen los dispositivos con los cuales se ha llevado a cabo la caracterización de la viga. Los resultados obtenidos son de vital importancia, pues en base a ellos se realizará el *model updating* en *Python*.

En la segunda parte se centra en explicar el modelo matemático que se sigue para aproximar el comportamiento de la viga. Se definen las matrices de masa y rigidez que corresponden a cada elemento, así como el número de elementos en los que dividimos la viga.

A continuación se explica el código de *Python*. El objetivo es que el lector entienda qué operaciones se hacen en cada línea de programación, facilitando la tarea de edición en caso de que fuera necesario.

Finalmente se muestran los resultados obtenidos por el programa y se comparan con los del laboratorio, para así verificar que el modelo se corresponde con la estructura del laboratorio. Para evaluar la similitud entre el modelo y la estructura real, comparamos sendas frecuencias propias, desplazamientos nodales causados por un impulso y respuestas en frecuencia.

2. Fundamentos teóricos.

2.1 *Python.*

Se trata de un lenguaje de programación de alto nivel. Una de las características que lo distinguen de otros lenguajes es el hecho de que es un software libre, ya que está administrado por *Python Software Foundation*. Esto implica que la distribución y uso del código es libre, incluso para uso comercial. Estamos hablando, por lo tanto, de una herramienta de fácil acceso, y que funciona tanto en Windows como en Mac.

Otra de las ventajas de que sea un lenguaje abierto, es que se retroalimenta de sus usuarios, que publican mejoras, nuevas librerías, y hacen muy sencilla la programación, ya que hay multitud de foros a los que podemos recurrir en caso de no saber interpretar un algoritmo, o si estamos atascados en algún problema. Páginas como "www.stackoverflow.com" son muy útiles a la hora de programar.

Otro motivo por el que se elige este código es que el lenguaje está preparado para que sea de fácil comprensión, omitiendo los " ; " al final de la línea como en C++, y sustituyendo varios símbolos por palabras, de forma que es más fácil de leer y entender para el usuario.[1]

2.2 *Finite element model updating.*

El procedimiento "finite element *model updating*" o "*model updating*" simplemente, se basa en el proceso por el cual un modelo teórico inicial construido para analizar la respuesta de una estructura puede ser corregido utilizando datos medidos sobre la estructura real y, actualmente, es una de las aplicaciones más exigentes y demandadas. El término inglés "finite element *model updating*" no tiene una traducción directa al español, pero puede ser interpretado como "calibración del modelo de elementos finitos".

Existen varios métodos los cuales se pueden clasificar en cuatro tipos:

- Directos
- Iterativos
- Minimización de los residuos modales
- Minimización de los residuos de respuesta

En la construcción de un modelo de elementos finitos suele ser habitual hacer simplificaciones, tanto en las condiciones de contorno como en la interacción con otros elementos, las cuales inducen errores en los datos obtenidos. Aun así, no todos los errores están relacionados con el modelo de elementos finitos, sino que en la toma de datos experimental también se producen irregularidades. Es necesario saber que en el proceso de obtención de datos experimentales se añaden diversos equipos a la estructura, bien sea para excitar o para medir, los cuales afectan a la dinámica de la estructura mediante sus masas y sus rigideces.

Por ejemplo, los acelerómetros fomentan ciertos errores en la medida con su uso, de los cuales se comentan a continuación los más comunes:

- Errores en las frecuencias naturales debidos a la masa de un acelerómetro son especialmente comunes en el análisis modal.
- El movimiento transversal y la flexión de la base de los acelerómetros y el ruido del cable del acelerómetro son fuentes comunes de errores.
- Los transductores piezoeléctricos tienden a carecer de linealidad a bajas frecuencias y pueden ser sensibles a la temperatura y a campos magnéticos y acústicos.

Los sistemas electrónicos, como los controladores de señales, generalmente introducen bajos niveles de ruido de los instrumentos y los problemas se encuentran a menudo a la frecuencia de la red eléctrica (50 Hz). Al mismo tiempo, las dificultades prácticas de las pruebas experimentales generalmente dan lugar a datos de mediciones imprecisos.

El procesamiento posterior también introduce errores adicionales, especialmente con los métodos de ajuste de curvas populares que requieren una intervención manual del usuario [2].
Debido a estas inexactitudes y a las simplificaciones comentadas, el comportamiento del modelo de elementos finitos no se corresponde con el de la estructura real. Por ello es necesario un calibrado.

El propósito de *Model Updating* consiste en la modificación de los parámetros que afectan a la masa, a la rigidez y al amortiguamiento del modelo de elementos finitos, con la intención de que la respuesta del modelo virtual se ajuste a la de la estructural real. Si el propósito del modelo es predecir la respuesta ante cargas no probadas o modificaciones en la estructura es importante que el calibrado del modelo sea lo más perfecto posible [2].

2.3 Análisis modal experimental.

Un análisis modal experimental, EMA en sus siglas en inglés, es el proceso por el cual se describe una estructura o sistema mecánico mediante sus propiedades dinámicas a partir de la respuesta frente a una excitación conocida. Su objetivo es determinar las frecuencias naturales, el amortiguamiento y caracterizar los modos de vibración en el rango de frecuencias de interés. Estos parámetros también pueden ser obtenidos con la técnica del OMA (Análisis Modal Operacional), basado en la medida de la respuesta de una estructura usando solo el ambiente y las fuerzas de servicio que actúen sobre la misma, sin embargo, el EMA proporciona además los modos de vibración escalados o, alternativamente, las masas modales. Todas las estructuras poseen frecuencias naturales y modos de vibración, que dependen básicamente de la masa, de la rigidez de la estructura y de sus condiciones de contorno [3].

La principal herramienta del análisis modal experimental es la función de respuesta en frecuencia (FRF), que es una curva característica de la estructura asociada a sus propiedades modales. Relaciona la frecuencia de excitación en un determinado punto con la relación de amplitudes respuesta/excitación en régimen estacionario en el mismo u otro punto de la estructura [4]. Si la respuesta recogida se corresponde con datos de desplazamiento, la función de respuesta en frecuencia se denomina receptancia, por otro lado, si se toman datos de velocidad, se llama movilidad, y finalmente, si los datos recogidos son de aceleraciones, la FRF se denomina acelerancia [5].

Se pueden definir tres estimadores de la función de respuesta en frecuencia, en función de en qué señal, si en la de entrada o en la de salida, consideren el ruido.

- Estimador H1: es la función de respuesta en frecuencia convencional. Se determina usando el espectro cruzado entre la entrada y la salida, y el auto-espectro de entrada, siendo x la entrada e y la salida [6]. Considera el ruido solo en la señal de entrada.

$$H_1 = \frac{S_{xy}}{S_{xx}}$$

- Estimador H_2: se obtiene normalizando el auto-espectro de salida por el espectro cruzado entre la salida y la entrada [6]. Tiene en cuenta el ruido en la señal de salida.

$$H_2 = \frac{S_{yy}}{S_{yx}}$$

- Estimador H_v: puesto que H_1 es un estimador de límite inferior y H_2 es un estimador de límite superior, se puede calcular la media o la media geométrica de H_1 y H_2 para obtener otra estimación de la FRF, el llamado estimador H_v o H_3 [6].

$$H_v = \sqrt{H_1 * H_2}$$

Este estimador tiene en cuenta el ruido en la medida de la fuerza (entrada) y en el auto-espectro de respuesta (salida). Para el caso de que la señal correspondiente al ruido sea pequeña en comparación con las señales de entrada y salida, Hv debería proporcionar una gráfica similar a la auténtica FRF del sistema [6]. En el caso de este trabajo, la tarjeta de adquisición de datos con la que se trabaja cuenta con este último estimador para obtener las funciones de respuesta en frecuencia.

Los estimadores H_1 y H_2 deben dar el mismo resultado, por lo que un indicador de la calidad del análisis se puede definir como la relación de estos dos estimadores, es la llamada coherencia (γ). Es un coeficiente normalizado de correlación entre la fuerza medida y las señales de respuesta evaluadas en cada frecuencia. Este parámetro varía su valor entre cero y la unidad.

$$\gamma^2 = \frac{H_1}{H_2}$$

Si la función de coherencia es menor que la unidad, tienen lugar una o más de las causas siguientes:

- Ruido extraño presente en la medición.
- Los errores de resolución.
- El sistema que relaciona la entrada con la salida no es lineal.

Dado que la acelerancia es una variable compleja, hay varias maneras de representarla en una gráfica en función de la frecuencia de vibración [7]:

- Representación módulo (magnitude)
- Representación parte real e imaginaria.
- Diagrama de Nyquist

La representación módulo es la elegida para este trabajo, pues cuando se ven las gráficas en escala log dB se aprecia más fácilmente la diferencia de desplazamiento entre altas y bajas frecuencias [7].

3. Test realizado en el laboratorio.

3.1 Estructura de análisis.

La estructura en base a la cual se ha redactado el programa en *Python* se trata de una viga de sección rectangular. El material de la estructura es aluminio, y además cuenta con varios acelerómetros en algunos de sus puntos que el modelo matemático que utilizaríamos para una viga continua.

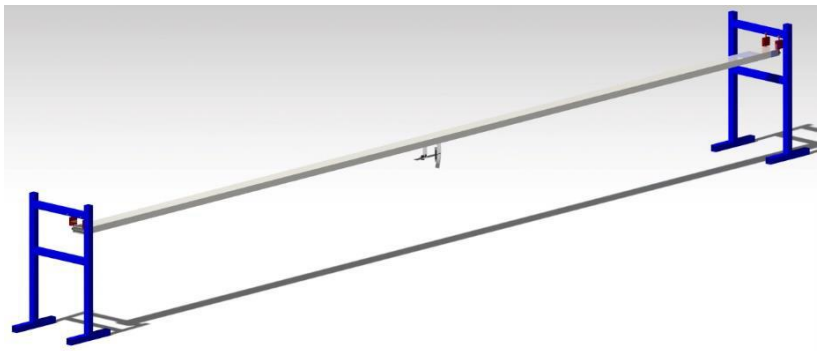


Ilustración 1. Modelo 3D de la Viga

Como vemos en la ilustración 1, el montaje de la estructura consta de la viga, dos soportes laterales que la sostienen por sus extremos y cuatro células de carga tipo S. La disposición de la viga tal y como se acaba de describir y el hecho de que el apoyo en las células de carga es suficientemente rígido, implica que el modelo que voy a utilizar se trata de una viga biapoyada.

La viga tiene una longitud de seis metros con un perfil de tipo 100*45-40*1.5 (mm) y las siguientes características teóricas:

- Área (A): 4.110 cm^2
- Momento de inercia (I_{xx}): 12.389 cm^4
- Producto de inercia (I_{xy}): 0 cm^4
- Momento de inercia (I_{yy}): 51.926 cm^4
- Módulo de alabeo (I_w): 25.4 cm^6
- Momento de torsión (J): 32.057 cm^4

En la siguiente ilustración podemos ver la sección de la viga:

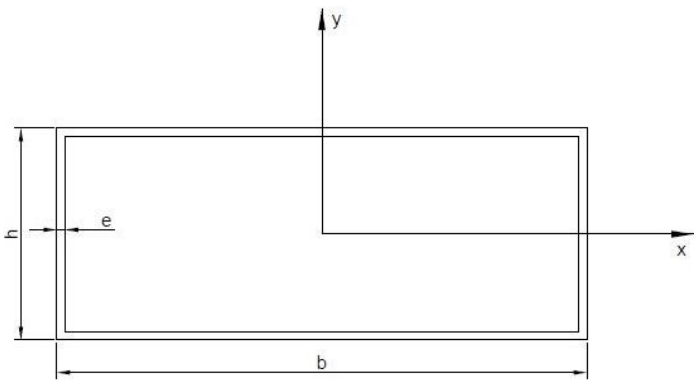


Ilustración 2. Sección viga rectangular

El aluminio utilizado presenta un acabado lacado en blanco y las siguientes propiedades teóricas:

- Módulo elástico, E: 70GPa
- Coeficiente de Poisson, υ: 0.33
- Límite elástico, σ_e: 80 MPa
- Densidad, ρ: 2700 kg/m3

Comprobaremos en el código que estas propiedades, junto con otras que caracterizan la viga, están situadas en la cabecera del programa facilitando su modificación en caso de requerir cualquier cambio.

En la siguiente imagen podemos ver la sección de la viga con el detalle de las dos células de carga que la sustentan apoyándose en sus dos superficies laterales. El perno que cruza la sección de lado a lado refuerza la hipótesis de que no hay torsión durante el ensayo. Además, durante el posicionamiento de los acelerómetros colocamos los dos situados en el centro, uno a cada lado de la viga, asegurando esta afirmación.

Ilustración 3. Sección de apoyo de la estructura

3.2 Elementos para la toma de datos.

Para la toma de datos disponemos de unos sensores, un ordenador con el software Dewesoft, y una tarjeta de adquisición de datos que hace de intermediario entre los dos.

3.2.1 Células de carga

Se trata de sensores de fuerza resistivos, basados en Puentes de Wheatstone, constituido por cuatro resistencias y un galvanómetro [8] que regulan la intensidad que se envía a la adquisición de datos. Su sensibilidad es de 2mV/V y su rango de entrada es de 10 Kg, lo cual es más que suficiente en nuestro caso. Podemos ver las células de carga a continuación:

Ilustración 4. Células de carga

Con estas células de carga obtendremos las reacciones en los apoyos. Sustentan la barra desde los dos extremos de forma que consideramos que la viga está biapoyada.

3.2.2 Acelerómetro

Son los sensores que se encargan de recopilar toda la información de desplazamientos de los puntos. Miden aceleraciones en un solo eje, por lo que se colocan sobre la viga con ayuda de un imán que se fija a una rosca que posee el propio acelerómetro. De esta manera medirá la aceleración en el eje vertical del punto de la pasarela sobre el que está colocado.

Ilustración 5. Acelerómetro

Se trata de un sensor de tipo piezoeléctrico.

3.2.3 Tarjeta de adquisición de datos

Para llevar la información proporcionada por todos estos sensores a un ordenador donde poder registrarla y procesarla se dispone de una tarjeta de adquisición de datos SIRIUS HD-STG.

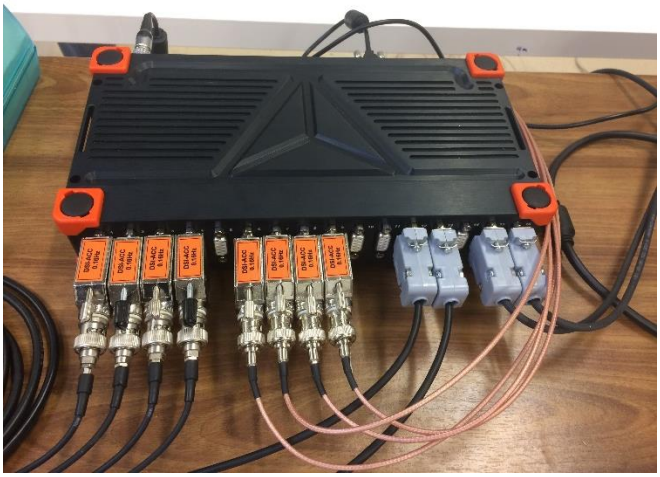


Ilustración 6. Tarjeta de adquisición de datos

Este aparato dispone de diferentes entradas y salidas, siendo las más importantes:

- 16 entradas tipo D-USB para la conexión de sensores.
- Salida tipo USB para conexión con el PC.
- Salida de alimentación para sensores (utilizada para el láser).
- Entrada de alimentación de corriente.

3.2.4 Software de adquisición de datos

El Software que nos permite trabajar con los resultados de los acelerómetros se llama Dewesoft. Configurado convenientemente podemos obtener las aceleraciones y respuestas en frecuencia de cada nodo. En la ilustración inferior se puede ver su apariencia.

Ilustración 7. Apariencia del software "Dewesoft" tras un ensayo

En la parte superior derecha se puede ver la respuesta en frecuencia del nudo 7, correspondiente a L=5m. El programa nos permite exportar los resultados a una hoja Excel, facilitando su análisis y la tarea de compararlos con las respuestas del programa.

3.3 Técnica de muestreo de datos.

El movimiento de la viga se registra mediante acelerómetros, los cuales generan una señal que, mediante cables, llega al registrador. En el registrador la señal es acondicionada para reducir el ruido. Posteriormente, se procesa la señal, que en este caso se realiza en Dewesoft.

Para la toma de datos utilizamos 7 acelerómetros piezoeléctricos y un martillo de impactos, con el que se da un golpe seco en cada punto dela barra. Para el procesamiento de datos utilizo un registrador de señales "SIRIUS" y un PC con el software Dewesoft instalado. Para el ensayo, situamos estos sensores en los puntos de la viga que tienen más amplitud en los primeros modos de vibración.

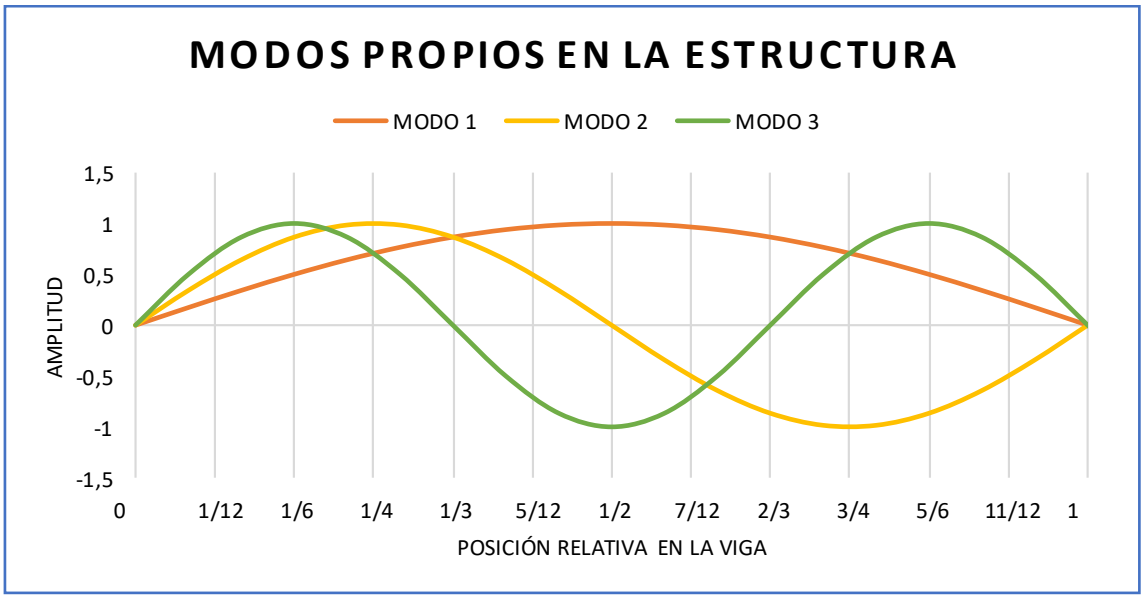


Ilustración 8. Tres primeros modos de vibración para una viga biapoyada

Conforme a los máximos que vemos en la ilustración, colocamos los acelerómetros en los tramos:

Modo	Posición relativa	L (m)
1	$^1/_2$	3 m
2	$^1/_4$, $^3/_4$	1,5 m y 4,5 m
3	$^1/_6$, $^1/_2$, $^5/_6$	1 m, 3 m y 5 m

Estos son los primeros modos, y los que más nos interesan para el análisis, ya que, como vamos a demostrar posteriormente, tienen una influencia muy dominante sobre el resto. De hecho, los siguientes modos son casi indetectables ya que están demasiado amortiguados.

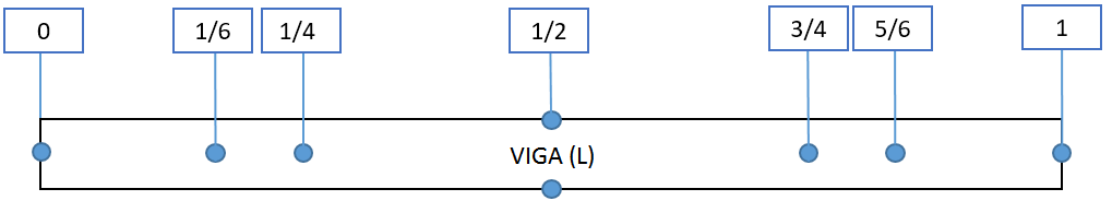


Ilustración 9. Esquema posicionamiento de los acelerómetros

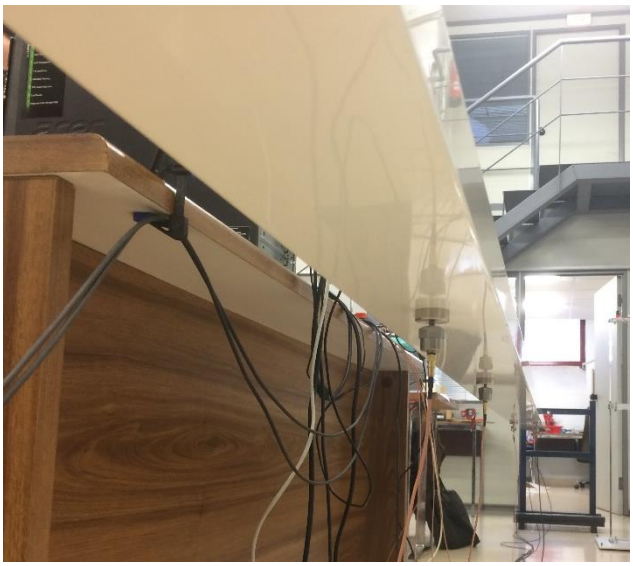


Ilustración 10. Posicionamiento de os acelerómetros.

3.4 Resultados.

Se realizan tres ensayos, golpeando cada vez un punto diferente. En el primer ensayo se golpea con el martillo en el medio (L=3m), en el segundo en L=4.5m y el tercero en L=5m. La idea es golpear en los puntos donde cada nodo se manifiesta más. Por ejemplo, al golpear en el medio, observaremos una gran amplitud en la frecuencia correspondiente al modo 1, ya que es justo donde está su máximo.

3.4.1 Frecuencias propias

Como el ruido que se produce en la medición es aleatorio, realizamos tres mediciones para cada punto, de modo que el programa filtra los gráficos, eliminando por completo este error.

Impacto en L=3m

Cuando realizamos en golpeo en el centro de la barra, la amplitud de la frecuencia correspondiente al primer modo es la máxima. En la imagen inferior se puede ver la FRF obtenida:

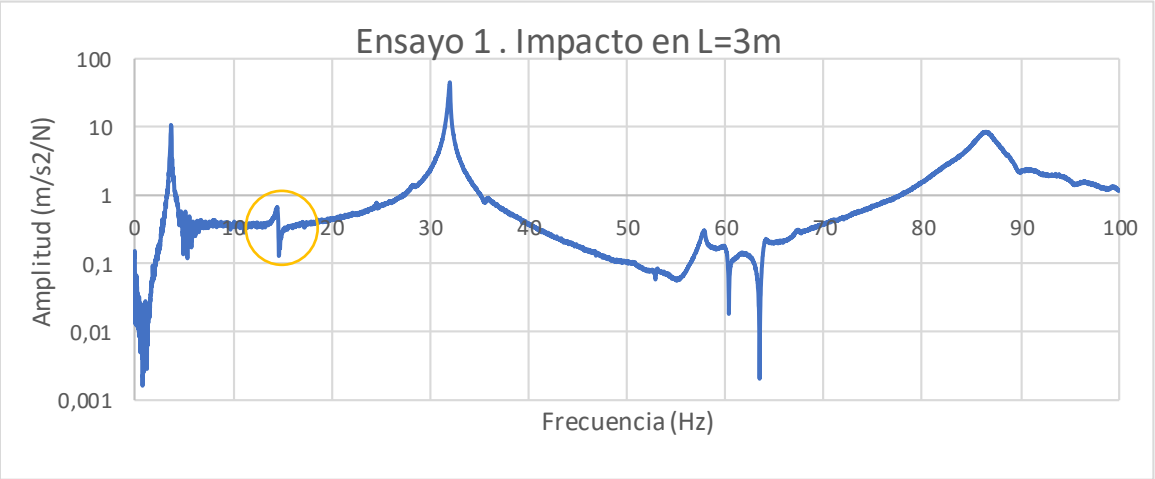


Ilustración 11. Ensayo 1, impacto en L=3m

El detalle en naranja es para resaltar la frecuencia a la cuál debería manifestarse el modo 2. Sin embargo éste está apagado, ya que el golpe ocurre justo en un punto en el que la amplitud de la forma modal es nulo.

Las frecuencias obtenidas son las siguientes:

Modo	Frecuencia (Hz)
1	3,703 Hz
2	-
3	32,031 Hz

Impacto en L=4.5m

En este ensayo se aprecian perfectamente los tres primeros modos propios. Sobre 87 Hz se puede ver un pico más alejado. Este no es de interés para el estudio ya que su influencia es mínima respecto a los tres modos principales.

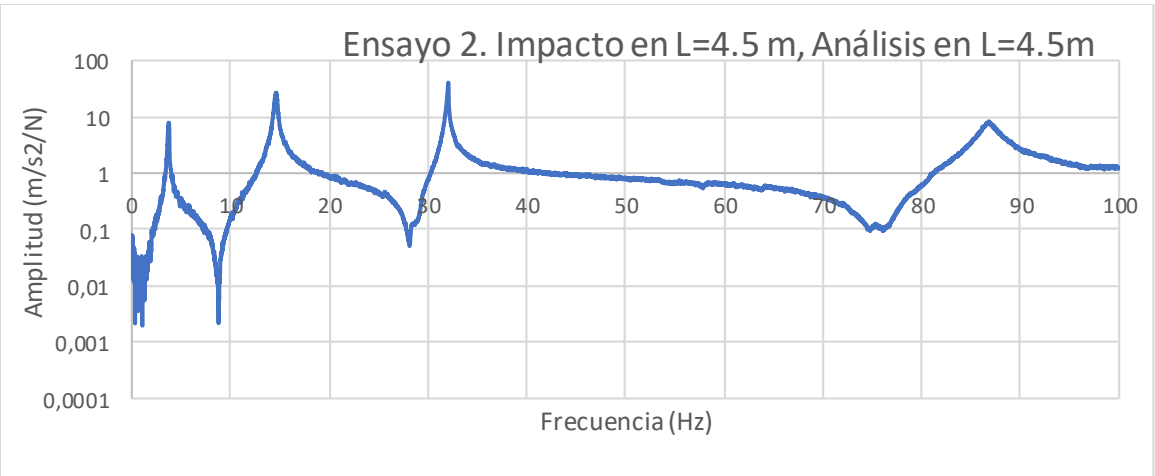


Ilustración 12. Ensayo 2. impacto en L=4.5m

Las frecuencias obtenidas son las siguientes:

Modo	Frecuencia (Hz)
1	3,719 Hz
2	14,594 Hz
3	32,047 Hz

Impacto en L=5m

En el tercer ensayo también obtenemos los picos para los tres primeros modos.

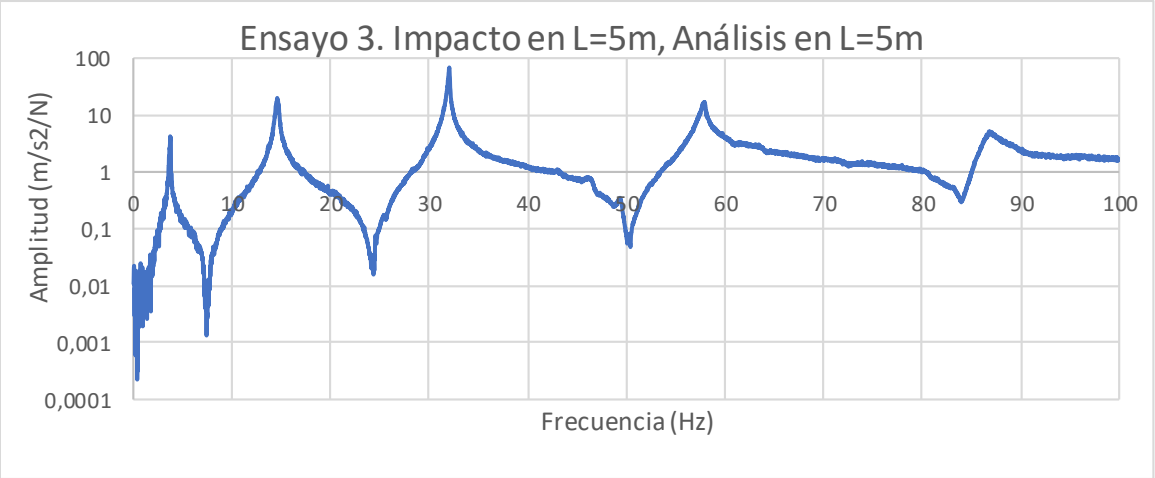


Ilustración 13. Ensayo 3, impacto en L=5m

Las frecuencias obtenidas son las siguientes:

Modo	Frecuencia (Hz)
1	3,734 Hz
2	14,594 Hz
3	32,016 Hz

Tras hacer la media de las frecuencias obtenidas se establece que las frecuencias en base a las cuales se ajustará el modelo computacional son las siguientes:

Modo	$\text{Frecuencia}_{\text{media}}$ (Hz)
1	3,71 Hz
2	14,59 Hz
3	32,03 Hz

3.4.2 Modelos FRF.

Gracias a los tres ensayos realizados se obtienen también las FRF de la viga cuando la golpeamos en 3, 4.5 y 6 metros. Como el programa nos permite exportar estas gráficas a Excel, las podremos superponer posteriormente, para así compararla con el modelo de *Python*.

3.4.3 Función impacto.

El martillo de impacto utilizado tiene un sensor piezoeléctrico. Está calibrado de forma que el programa recibe los Newtons de fuerza.

Ilustración 14. Roving hammer

El rango de frecuencias excitadas cambia en función del material de impacto. Si utilizamos un material más blando la duración del impacto es mayor. Es por este motivo que todos los ensayos se llevan a cabo con la misma cabecilla de martillo.

En la siguiente imagen se puede ver la función impacto del ensayo 1. Vemos que el intervalo de tiempo es muy pequeño, y que el valor máximo que alcanza roza los 20 N.

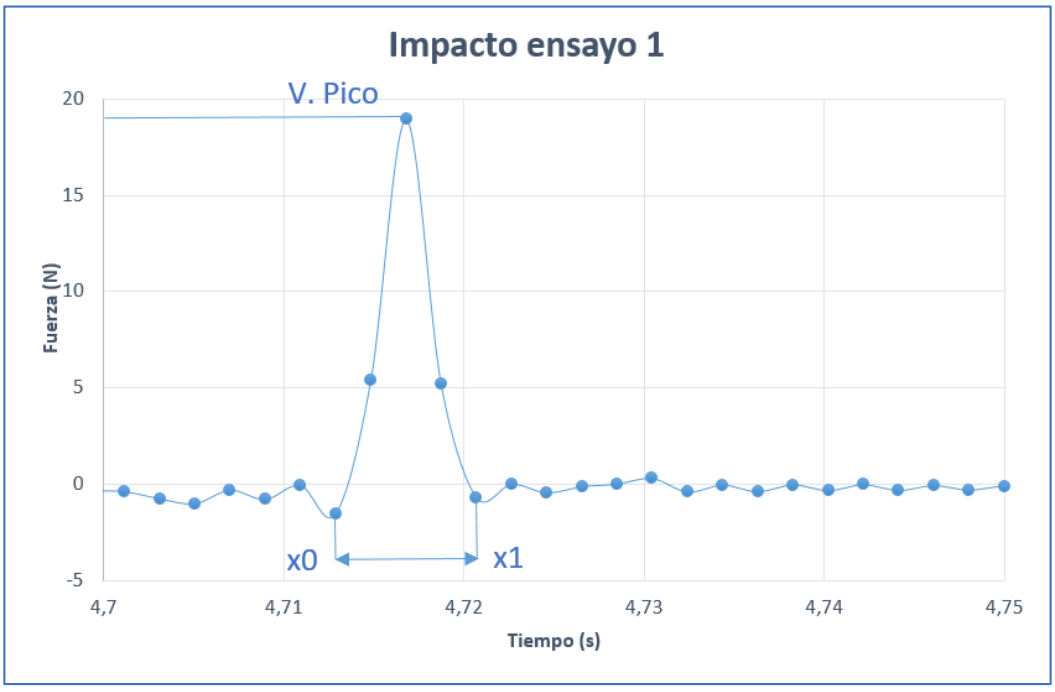


Ilustración 15. Plot fuerza del impacto

Para facilitar la implementación analítica del problema, aproximo la curva dada por el acelerómetro del martillo, a una función conocida. En este caso la función que más se parece es una función senoidal. Para definirla, necesito calcular los siguientes parámetros:

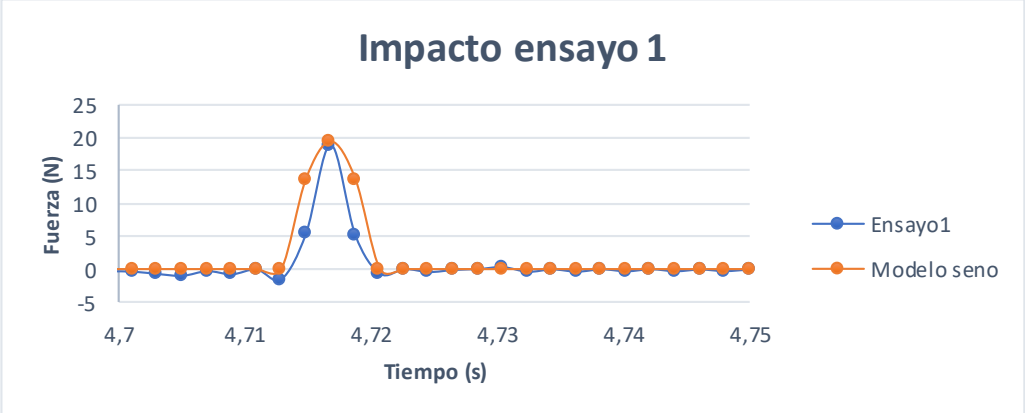
- Semiperiodo: Lo calculo como la diferencia de tiempos desde que empieza a subir la curva, hasta que vuelve a bajar.
- Valor pico: Calculado como el valor máximo de la función.

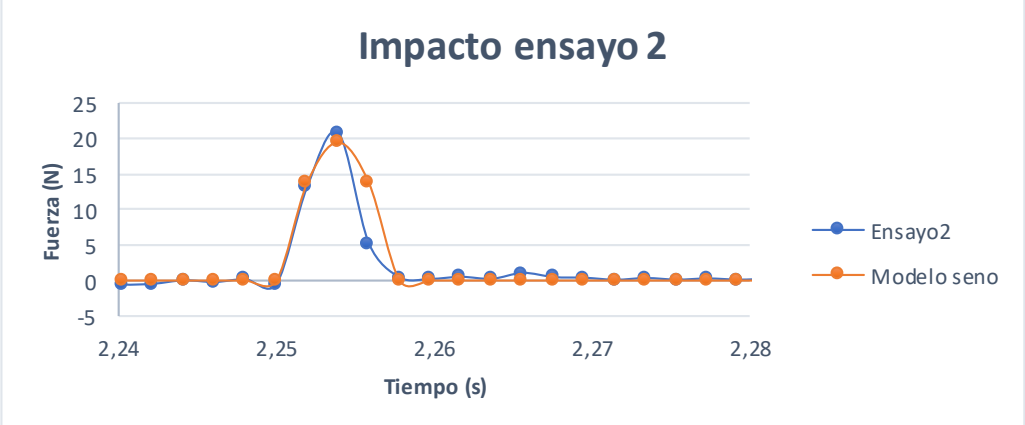
Calculo estos valores para cada ensayo, y a continuación hago la media, para minimizar el error:

	x0 (s)	x1 (s)	T (s)	Valor pico (N)
Ensayo1	4,7128	4,7207	0,0078	18,96
Ensayo2	2,2500	2,2578	0,0078	20,83
Ensayo3	2,2109	2,2187	0,0078	18,57
Media			0,0078	19,45

Ilustración 16. Valores medios definiendo la función impacto

La forma de esta función se puede ver en el apartado 5.3. En las siguientes imágenes se superponen la curva analítica con la experimental. Vemos que algunos puntos se van ligeramente del recorrido de la curva senoidal, aunque este error es asumible si tenemos en cuenta que estamos tratando con intervalos de tiempo de 2 milisegundos.





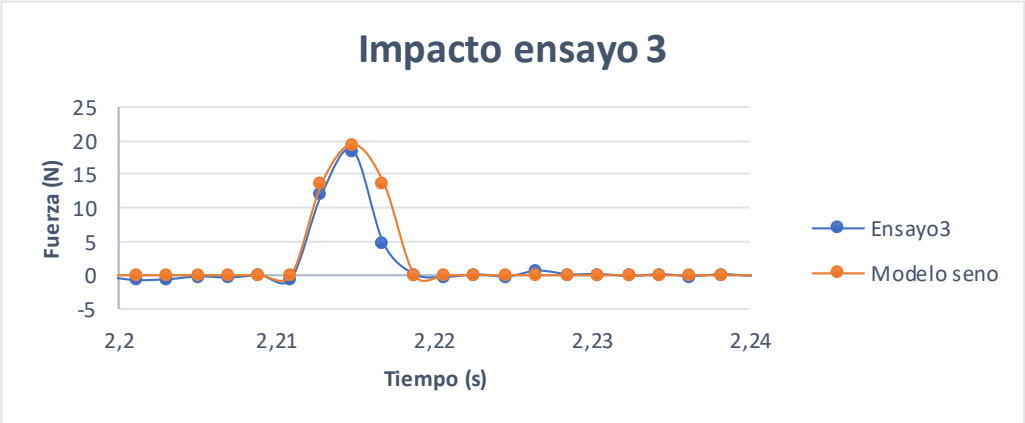


Ilustración 17. Comparación de la función impacto seno, con los datos prácticos.

4. Modelo matemático.

En este apartado se explica el modelo matemático elegido para la resolución de la viga en vibración. El objetivo es que el lector entienda cómo se divide la viga, cómo se calculan y de qué forma se ensamblan las matrices de masa y rigidez.

4.1 División de la viga.

En la siguiente imagen podemos ver una ilustración esquemática de cómo dividimos la viga principal. Para empezar, partimos la viga en 4 tramos, a los cuales denomino como "Barra", que tienen una longitud "lp" igual a:

$$l_P = \frac{L}{N_B}$$

Posteriormente dividimos cada barra en un cierto número de elementos (en el esquema mostrado abajo son 4), que tienen una longitud calculada como:

$$l_{Elem} = \frac{l_P}{N_{elem}}$$

De igual manera, si queremos saber el número de puntos, definido en el programa como "n", basta con multiplicar el número de elementos por el número de barras:

$$n = N_B * N_{Elem}$$

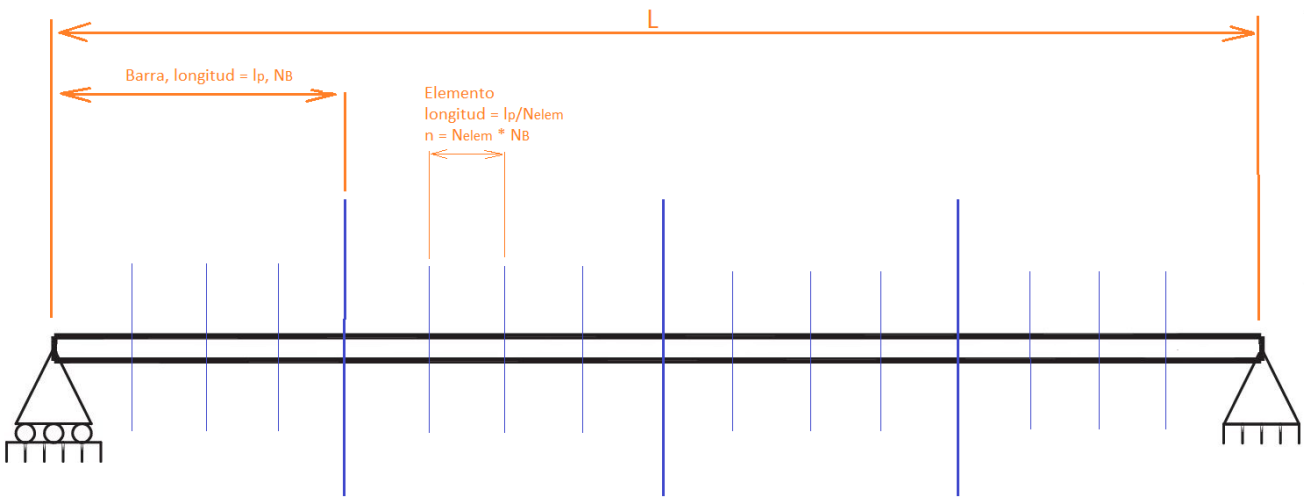


Ilustración 18. División de la viga en 4 barras, de 4 elementos cada una

Es importante destacar que cada barra tiene sus propiedades físicas, y que este es el principio del *model updating*. A fin de ajustar las curvas teóricas a las obtenidas en el ensayo, tendremos que cambiar algunas de las propiedades de las barras.

El número de elementos nos permite ajustar la precisión de los cálculos. Hay que tener en cuenta que esto implica un aumento del tiempo de procesamiento del programa, ya que el

tamaño de las matrices aumenta proporcionalmente como 2·(n+1). Es decir, si tenemos 14 elementos, como en la figura del ejemplo, resolvemos el problema de frecuencias con matrices de masa y rigidez de 30·30, sin embargo, si tenemos 28 elementos, las matrices serán de 58·58, lo cual implica una diferencia notable en tiempo de cálculo.

Tras varios ensayos, se demostró que 16 elementos es suficiente para tener una buena precisión, equilibrada con el tiempo de procesamiento.

4.2 Matrices de masa y rigidez.

Cada elemento consta de dos nodos, separados por una distancia l_{elem}. Para la solución solo consideramos desplazamientos y giros de los nodos. Es decir, consideramos el problema en dos dimensiones, asumiendo que la torsión es despreciable

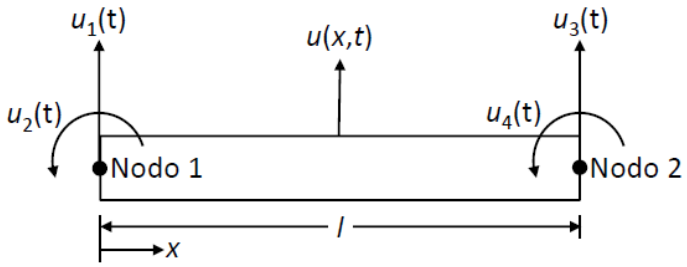


Ilustración 19. Malla de un elemento y dos nodos.

El desplazamiento estático (independiente del tiempo) de esta barra debe satisfacer la ecuación [9]:

$$EA\frac{\partial^2 u(x)}{\partial x^2} = 0$$

Tras resolver la ecuación aplicando las condiciones de contorno se determinan las funciones de forma. Posteriormente se halla la energía potencial y cinética a fin de conseguir la ecuación de movimiento y así hallar las matrices de masa y rigidez [10].

El modelo en elementos finitos para un elemento, considerando dos giros y desplazamiento vertical es el siguiente:

$$M_{elem} = \begin{bmatrix} 140 & 0 & 0 & 70 & 0 & 0 \\ 0 & 156 & 22l & 0 & 54 & -13l \\ 0 & 22l & 4l^2 & 0 & 13l & -3l^2 \\ 70 & 0 & 0 & 140 & 0 & 0 \\ 0 & 54 & 13l & 0 & 156 & -22l \\ 0 & -13l & -3l^2 & 0 & -22l & 4l^2 \end{bmatrix} * \frac{m}{420}$$

$$K_{elem} = \begin{bmatrix} l^2 & 0 & 0 & -l^2 & 0 & 0 \\ 0 & 12l & 6l & 0 & -12 & 6l \\ 0 & 6l & 4l^2 & 0 & -6l & 2l^2 \\ -l^2 & 0 & 0 & l^2 & 0 & 0 \\ 0 & -12 & -6l & 0 & 12 & -6l \\ 0 & 6l & 2l^2 & 0 & -6l & 4l^2 \end{bmatrix} * \frac{EA}{l^3}$$

Al considerar el problema bidimensional prescindimos de las filas y columnas de las matrices asociadas al giro en x de la viga. Estas son las filas y columnas 1 y 4. Las matrices de masa y rigidez resultantes son:

$$M_{elem} = \begin{bmatrix} 156 & 22l & 54 & -13l \\ 22l & 4l^2 & 13l & -3l^2 \\ 54 & 13l & 156 & -22l \\ -13l & -3l^2 & -22l & 4l^2 \end{bmatrix} * \frac{m}{420}$$

$$K_{elem} = \begin{bmatrix} 12l & 6l & -12 & 6l \\ 6l & 4l^2 & -6l & 2l^2 \\ -12 & -6l & 12 & -6l \\ 6l & 2l^2 & -6l & 4l^2 \end{bmatrix} * \frac{EA}{l^3}$$

4.3 Condiciones de contorno.

Para resolver este problema, necesitamos hacer uso de las condiciones de contorno, acotando la resolución. Se trata de una viga biapoyada, por lo tanto, sabemos que el desplazamiento en sus dos extremos debe ser nulo. Esto afecta a las matrices de masas y de rigidez, ya que tendremos dos filas y dos columnas menos en ambas. En concreto, nos tenemos que deshacer de aquellos elementos de la matriz, que hacen referencia al desplazamiento vertical de los nodos extremos de la viga.

Por ejemplo, en caso de tener una viga dividida en tres tramos, en la imagen inferior podemos comprobar que tenemos 4 nodos.

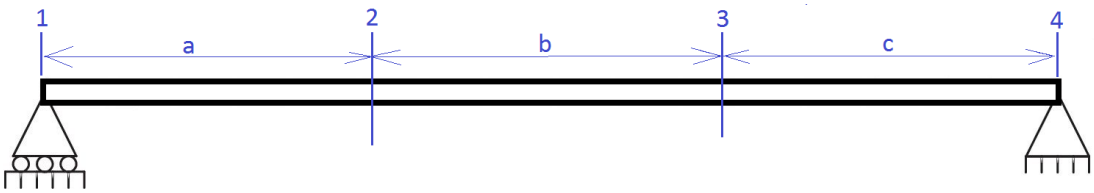


Ilustración 20. Viga dividida en tres tramos

La matriz de masas correspondiente para esta figura la podemos ver a continuación:

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33}+b_{11} & a_{34}+b_{12} & b_{13} & b_{14} & 0 & 0 \\ a_{41} & a_{42} & a_{43}+b_{21} & a_{44}+b_{22} & b_{23} & b_{24} & 0 & 0 \\ 0 & 0 & b_{31} & b_{32} & b_{33}+c_{11} & b_{34}+c_{12} & c_{13} & c_{14} \\ 0 & 0 & b_{41} & b_{42} & b_{43}+c_{21} & b_{44}+c_{22} & c_{23} & c_{24} \\ 0 & 0 & 0 & 0 & c_{31} & c_{32} & c_{33} & c_{34} \\ 0 & 0 & 0 & 0 & c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

Se trata de una matriz cuadrada de 2·(3+1) = 8 filas. En la siguiente expresión resalto las tres matrices correspondientes a cada tramo, en color azul, naranja y verde para los tramos a, b y c respectivamente.

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33}+b_{11} & a_{34}+b_{12} & b_{13} & b_{14} & 0 & 0 \\ a_{41} & a_{42} & a_{43}+b_{21} & a_{44}+b_{22} & b_{23} & b_{24} & 0 & 0 \\ 0 & 0 & b_{31} & b_{32} & b_{33}+c_{11} & b_{34}+c_{12} & c_{13} & c_{14} \\ 0 & 0 & b_{41} & b_{42} & b_{43}+c_{21} & b_{44}+c_{22} & c_{23} & c_{24} \\ 0 & 0 & 0 & 0 & c_{31} & c_{32} & c_{33} & c_{34} \\ 0 & 0 & 0 & 0 & c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

La primera y tercera fila de cada matriz elemento corresponde al desplazamiento del nodo. En este caso tenemos una matriz de masas de dimensiones 8*8, cuando solo hay 3 tramos y, por lo tanto, 6 incógnitas (tres rotaciones y tres desplazamientos). Sin embargo, sabemos que el desplazamiento del nodo 1 y el nodo 4 es nulo, dado que la viga está biapoyada. Para llevar a cabo esta afirmación en la práctica es necesario eliminar las filas y columnas 1 y 7, tal y como se ilustra en la siguiente imagen:

$$M = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33}+b_{11} & a_{34}+b_{12} & b_{13} & b_{14} & 0 & 0 \\ a_{41} & a_{42} & a_{43}+b_{21} & a_{44}+b_{22} & b_{23} & b_{24} & 0 & 0 \\ 0 & 0 & b_{31} & b_{32} & b_{33}+c_{11} & b_{34}+c_{12} & c_{13} & c_{14} \\ 0 & 0 & b_{41} & b_{42} & b_{43}+c_{21} & b_{44}+c_{22} & c_{23} & c_{24} \\ 0 & 0 & 0 & 0 & c_{31} & c_{32} & c_{33} & c_{34} \\ 0 & 0 & 0 & 0 & c_{41} & c_{42} & c_{43} & c_{44} \end{bmatrix}$$

La misma operación se realiza para la matriz de rigidez.

5. Aproximación matricial.

En este apartado se explica el fundamento teórico que nos ha permitido calcular los modos propios, formas modales, desplazamientos nodales y respuestas en frecuencia.

5.1 Frecuencias propias y modos propios.

Para hallar los modos y frecuencias propias asociados a la estructura, estudiamos el problema libre de oscilaciones. Considerando que no hay fuerzas externas, la ecuación que define el movimiento de la viga es [11]:

$$[M]\{\ddot{x}\} + [K]\{x\} = 0$$

Donde {x}={x(t)} es el vector desplazamiento. Cada elemento del vector contiene el desplazamiento de uno de los nodos. Encontramos la solución de la ecuación diferencial definiendo la variable como:

$$\{x(t)\} = \{X\}\big(a\cos(\omega t) + bsen(\omega t)\big)$$

Resultando la ecuación en el requerimiento:

$$([K] - [M]\omega^2)\{V_i\} = 0$$

Por lo tanto, el modelo modal tiene una única solución (modo de vibración) a una frecuencia natural, ω_0 , dada por $(k/m)^{\frac{1}{2}}$. Gracias a la simetría de las matrices [M] y [K] se puede demostrar que los modos propios el sistema son reales.

5.2 Función impacto. Aproximación teórica.

Para aproximar el impacto utilizo una función seno, ya que es la función que más se aproxima a la excitación real de la viga:

$$f(t) = \text{F0}\sin\left(\frac{\pi t}{T}\right) \qquad 0 \le t \le T$$
$$f(t) = 0 \qquad\qquad\qquad\qquad t > T$$

Donde F0 representa la amplitud del impacto en Newtons y T la duración del impacto en segundos. Esta excitación se produce en el nodo golpeado, y el resto vale 0, por lo que el vector impacto es inmediato:

$$\{f(t)\} = \begin{Bmatrix} 0 \\ \vdots \\ 0 \\ F0sin\left(\frac{\pi t}{T}\right) \\ 0 \\ \vdots \\ 0 \end{Bmatrix}$$

5.3 Respuesta nodal. Aproximación teórica.

Para obtener los desplazamientos de cada nodo de la estructura es necesario resolver la conocida segunda ley de Newton, que una vez discretizada en el dominio espacial resulta:

$$[M]\{\ddot{q}(t)\} + [C]\{\dot{q}(t)\} + [K]\{q(t)\} = \{f(t)\}$$

Donde {q(t)} representa el vector de desplazamientos nodales, y {f(t)} contiene las excitaciones externas que solicitan el sistema [12]. Las matrices [M], [C] y [K] son simétricas, de dimensiones 2(n+1).

En este apartado se explica cuál es el razonamiento teórico que se ha utilizado para llegar a la solución de q(t).

5.3.1 Diagonalización de las matrices.

Se define la matriz de transformación modal como [V]. Esta matriz contiene en todas sus columnas los modos propios, ordenados de forma creciente:

$$[V] = [\{\phi_1\}\{\phi_2\}\{\phi_3\}\dots\{\phi_n\}]$$

Gracias al principio de ortogonalidad de modos, los modos propios son linealmente independientes, y por lo tanto forman una base en el espacio de 2(n+1) dimensiones. En consecuencia, cualquier otro vector en este mismo espacio puede ser expresado como una combinación linear de estos 2(n+1) modos propios. Las matrices de masa y rigidez M y K se pueden diagonalizar mediante la siguiente operación:

$$\left[\widetilde{M}\right] = [V]^t[M][V] \qquad \left[\widetilde{K}\right] = [V]^t[K][V]$$

Estas matrices diagonalizadas se llaman matrices modales [13]. Ahora bien, con el fin de simplificar los cálculos, es posible normalizar la matriz [V], de forma que el resultado de la diagonalización de la matriz de masa es la matriz identidad, y la matriz de rigidez contiene en su diagonal las frecuencias propias.

$$\left[\widetilde{M}\right] = [V]^t[M][V] = [I] \qquad \left[\widetilde{K}\right] = [V]^t[K][V] = [\omega_r^2]$$

5.3.2 Matriz de amortiguamiento.

Para que la matriz de amortiguamiento sea diagonalizable es necesario que ésta sea simétrica. Por este motivo se utiliza un amortiguamiento de Rayleigh, que se basa en la combinación lineal de las matrices de masa y rigidez [14], y facilitará enormemente los cálculos más adelante:

$$[C] = \alpha[M] + \beta[K]$$

Donde el coeficiente alfa establece el coeficiente proporcional a la masa, y beta el proporcional a la rigidez. Si aplicamos la transformación de coordenadas modal, la matriz diagonal es de la forma:

$$[V]^T[C][V] = [\tilde{C}] = \alpha[I] + \beta[\omega^2]$$

La matriz de amortiguamiento modal se puede expresar también como:

$$[\tilde{C}] = 2[\xi\omega]$$

El coeficiente de amortiguamiento viscoso, c, para cualquier modo r, se calcula:

$$C_r = 2\xi_r\omega_r = \alpha + \beta\omega_r^2$$

Por lo tanto, el cociente de amortiguamiento viscoso, ξ_r, es:

$$\xi_r = \frac{\alpha}{2\omega_r} + \frac{\beta\omega_r}{2}$$

Los valores experimentales de este factor de amortiguamiento son, para los tres primeros modos:

- ξ_1 = 1,13 %
- ξ_2 = 0,58 %
- ξ_3 = 0,21 %

Como estos son los modos que más colaboran en la vibración de la estructura, ajusto los valores de alfa y gamma para obtener los valores más aproximados posibles. Los factores dependen de dos variables, luego por definición solo puedo ajustar dos de ellos. Resolviendo el sistema obtengo:

$$\alpha = 0.52350036$$
$$\beta = 7.94311582 \cdot 10^{-6}$$

Valores que dan una buena aproximación de los primeros factores de amortiguamiento:

$$\xi_1 = 1{,}13\ \%$$
$$\xi_2 = 0{,}32\ \%$$
$$\xi_3 = 0{,}21\ \%$$

Con estos valores de alfa y beta, podemos dibujar la gráfica en la que vemos cómo varía ξ_r en función de la frecuencia de vibración:

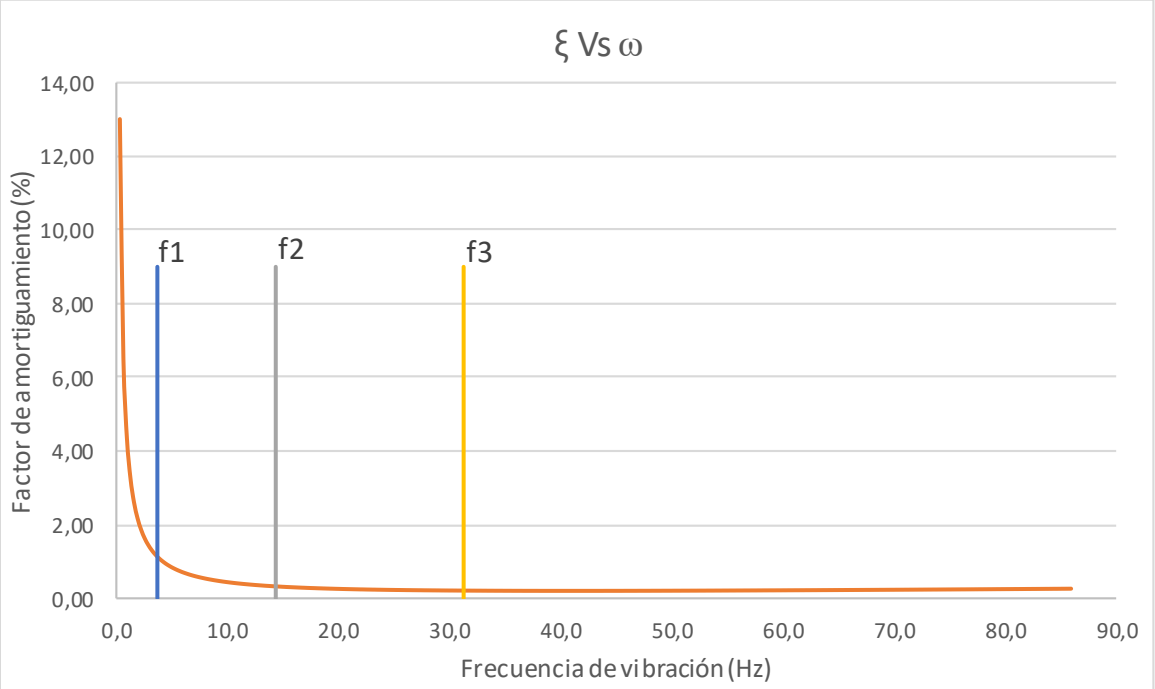


Ilustración 21. Factor de amortiguamiento en función de la frecuencia de vibración

Las líneas verticales marcadas en el gráfico indican la frecuencia a la cual se producen los tres primeros modos propios. Vemos que el factor de amortiguamiento es superior para el primer y segundo modo. Para frecuencias superiores al tercer modo, el valor del factor de amortiguamiento es muy similar, aproximándose al 0,21%.

5.3.3 Método de superposición modal.

Los modos propios forman una base a partir de la cual se puede expresar cualquier deformada de la estructura en cualquier instante de tiempo, q(t). De esta forma, q(t) no es más que una combinación lineal de todos los modos propios. A los coeficientes que multiplican a cada modo se les denomina grados de libertad modales o coordenadas generalizadas, $r_i(t)$.

De esta manera, se puede expresar la solución del problema mediante la expresión de la siguiente ecuación:

$$\{q(t)\} = r_1(t)\phi_1 + r_2(t)\phi_2 + \cdots + r_3(t)\phi_3 = [V]\{R(t)\}$$

Donde se emplea la matriz de modos y el vector R(t). La utilización de esta expresión se puede entender también como un cambio de variable que simplifica la resolución de la EDO. Con este cambio de variable obtenemos:

$$[M][V]\{\ddot{R}(t)\} + [C][V]\{\dot{R}(t)\} + [K][V]\{R(t)\} = \{f(t)\}$$

Continuando con este razonamiento, es posible pre-multiplicar a todo el sistema de ecuaciones por la matriz de modos transpuesta $[V]^t$, como se ilustra en la ecuación 5, diagonalizando las matrices físicas M, C y K y transformándolas en las correspondientes matrices modales. La forma de la ecuación es la siguiente:

$$[V]^t[M][V]\{\ddot{R}(t)\} + [V]^t[C][V]\{\dot{R}(t)\} + [V]^t[K][V]\{R(t)\} = [V]^t\{f(t)\}$$

$$[\tilde{M}]\{\ddot{R}(t)\} + [\tilde{C}]\{\dot{R}(t)\} + [\tilde{K}]\{R(t)\} = \tilde{F}(t)$$

Como hemos visto en los anteriores apartados, las matrices modales son diagonales, por lo que el sistema está desacoplado. Además, las matrices modales están escaladas a la masa, así que la forma de la ecuación es:

$$\{\ddot{R}(t)\} + [2\xi\,\omega^2]\{\dot{R}(t)\} + [\omega^2]\{R(t)\} = \tilde{F}(t)$$

Por lo tanto, se puede resolver el sistema hallando la solución de las 2(n+1) ecuaciones diferenciales. Cada una de estas ecuaciones se puede expresar como:

$$\ddot{r}_i + 2\xi_i\omega_i\dot{r}_i + \omega_i^2 r_i = \tilde{f}_i(t)$$

Una vez hallada la variable modal r, es posible volver a las variables físicas. Después se despeja la ecuación diferencial para la aceleración de los nodos. Este valor sirve para comparar los resultados del programa con el ensayo en el laboratorio.

$$\{q(t)\} = [V]\{R(t)\}$$

$$\{\dot{q}(t)\} = [V]\{\dot{R}(t)\}$$

$$\{\ddot{q}(t)\} = [M]^{-1}(\{f(t)\} - [C]\{\dot{q}(t)\} - [K]\{q(t)\})$$

5.4 Respuesta en frecuencia. FRF.

El tipo de amortiguamiento que se ha modelizado es uno de los más sencillos a la hora de analizar. Como se ha mencionado anteriormente, se trata de un amortiguamiento proporcional. La principal ventaja de utilizar este tipo de amortiguamiento es que las formas modales y las frecuencias naturales son idénticas a las del modelo no amortiguado.

Si volvemos a la ecuación general de desplazamiento, con la matriz de amortiguamiento viscoso obtenemos:

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{f\}$$

Se ha definido la matriz de amortiguamiento de forma que es proporcional a las matrices de masa y rigidez:

$$[C] = \alpha[M] + \beta[K]$$

Como ya se ha demostrado, esta matriz es simétrica y diagonalizable. Si realizamos un cambio de variable como el que se hizo para encontrar los modos propios del modelo sin amortiguamiento y premultiplicamos por la matriz de modos, obtenemos un sistema de r ecuaciones desacopladas[15]:

$$[m_r]\{\ddot{p}_r\} + [c_r]\{\dot{p}_r\} + [k_r]\{p_r\} = 0$$

A continuación, para hallar la respuesta en frecuencia necesitamos considerar el caso en el que la estructura es excitada sinusoidalmente por un conjunto de fuerzas a la misma frecuencia, ω, pero con amplitudes y fases individuales:

$$\{f(t)\} = \{F\}e^{i\omega t}$$

También asumimos que existe una solución en desplazamientos de la forma:

$$\{x(t)\} = \{X\}e^{i\omega t}$$

Donde {F} y {X} son Nx1 vectores independientes del tiempo y con amplitudes complejas. Realizando este cambio, la ecuación de movimiento queda:

$$(-\omega^2[M] + i\omega[C] + [K])\{X\}e^{i\omega t} = \{F\}e^{i\omega t}$$

Reorganizando la ecuación, para despejar las respuestas:

$$\{X\} = (-\omega^2[M] + i\omega[C] + [K])^{-1}\{F\}$$

Que puede ser escrito también como:

$$\{X\} = [\alpha(\omega)]\{F\}$$

Donde se define $\alpha(\omega)$ como la matriz NxN de receptancia FRF para el sistema, y constituye por lo tanto la respuesta en frecuencia. Este método para hallar la receptancia es demasiado costoso de implementar en el programa, ya que necesitaríamos calcular toda la matriz de receptancias para hallar la FRF de un único modo que nos interesa. Es decir, para calcular la FRF de un punto tendríamos que crear una lista de matrices NxN, de dimensión igual al rango de frecuencias que queremos analizar [15]

Es por este motivo que aprovechamos la posibilidad de diagonalizar las matrices, para obtener la expresión de la receptancia para un único modo. De la definición de receptancia, multiplicamos por la matriz de modos y su traspuesta a ambos lados de la ecuación:

$$[V]^T(-\omega^2[M] + i\omega[C] + [K])[V] = [V]^T[\alpha(\omega)]^{-1}[V]$$

Si la matriz de modos está reducida a la masa obtenemos:

$$-\omega^2[I] + i\omega[c_r] + [\omega_r^2] = [V]^T[\alpha(\omega)]^{-1}[V]$$

Despejando la receptancia:

$$[\alpha(\omega)] = [V][-\omega^2 + \omega_r^2 + i\omega c_r][V]^T$$

Donde $c_r = 2\xi_r\omega_r$, ya que el amortiguamiento es proporcional.

En esta última ecuación se ve claramente que la matriz de receptancia es simétrica, lo cual nos permite calcular individualmente cualquier FRF $\alpha_{jk}(\omega)$ como:

$$\alpha_{jk}(\omega) = \sum_{r=1}^{N}\left(\frac{[V]_{jr}[V]_{kr}}{\omega_r^2 - \omega^2 + i2\xi_r\omega_r\omega}\right)$$

La receptancia no es una magnitud que hayamos obtenido en el laboratorio, y por lo tanto no tenemos información experimental sobre ella. Para comparar las dos respuestas en frecuencia obtenidas, es necesario pasar la receptancia a acelerancia. Tras una doble derivación obtenemos[15]:

$$a_{jk}(\omega) = \frac{\{\ddot{X}(i\omega)\}}{\{F(i\omega)\}} = -\omega^2\alpha_{jk}(\omega)$$

Por supuesto, la FRF es una magnitud compleja, de modo que tenemos módulo y fase. Para comparar la magnitud con la del laboratorio nos vamos a centrar en la gráfica del módulo en función de la frecuencia de vibración.

6. Pseudocódigo computacional.

En este apartado se explicará, paso a paso, las líneas del programa. El objetivo es que el lector tenga una idea clara al final del capítulo, de las operaciones que se realizan en cada momento, a pesar de que no esté muy familiarizado con el código de programación *Python*.

Para mejorar la comprensión del código, cada línea está debidamente comentada. Recuerdo que en los anexos se puede encontrar el código completo.

6.1 Modelización de la viga.

El primer paso es definir todas las características de la viga.

```
17 # Modelización de la Viga-------------------------------------------------------
18
19 Nb=4            #Número de barras - No se debería poder cambiar a nivel usuario (comprobar que con 4 vale)
20 Nelem=4         #Número de elementos en los que dividimos cada barra
21 n=Nb*Nelem      #Número total de elementos
22 a=0.04          #Canto de la viga en metros
23 b=0.10          #Ancho de la viga (donde se apoya) en metros
24 e=0.0015        #Espesor de la barra en metros (Común para todos los elementos)
25 L=6             #Longitud total de la barra en metros
26
27
28 rho=2700             #Densidad de cada barra, en kg/m3 (aluminio)
29 lp=L/Nb              #Longitud de una barra
30 E=7.0e10             #Módulo de Young de cada barra, en Pa
```

Ilustración 22. Modelización de la viga

Como vemos en la imagen superior, se definen las variables dimensionales, como habíamos hablado, de una barra de 6m de longitud y un perfil de 40*100*1.5 mm.

También se define el mallado de la viga, estableciendo que se divide en cuatro barras, y estas a su vez en otras cuatro divisiones. Llevo a cabo estas divisiones por los siguientes motivos:

- Nb (Número de barras): Permite afinar los cálculos si lo aumentamos. Hay que tener en cuenta que un aumento en este número también produce un incremento en el tiempo de cálculo. Por lo tanto, hay que ser coherente y no hacer modificaciones cuando hemos obtenido suficientes cifras significativas. Tras muchos cálculos, he determinado que 4 barras son suficientes para obtener una precisión coherente. Este número no debe ser modificado a nivel usuario, porque su modificación implica realizar otros cambios adicionales, como introducir las características de cada nueva barra del *model updating*, y esta tarea puede no ser evidente para una persona que no está familiarizada con el código.

- Nelem (divisiones de cada barra): Sirve para obtener aún más precisión en los cálculos. Sin embargo, no es recomendable aumentarlo en exceso, ya que el tiempo de cálculo se incrementa mucho.

A continuación, se establecen en las líneas 28, 29 y 30, la densidad, módulo de Young, y longitud de cada barra. En la sección 3.1 se habla de estas variables.

6.2 Input impacto.

Durante estas líneas del programa, definimos los valores del impacto.

```
# Input Impacto-----------------------------------------------------------------

F0=19.45                            #Fuerza aplicada en Newtons
t0=15                               #Tiempo de análisis de la respuesta en segundos
T=0.0078125                         #Duración del impacto en segundos
intervalos=10000                     #Número de intervalos en el análisis del impacto
t=linspace(0,t0,intervalos)         #Vector temporal
h=t0/(intervalos-1)                 #Intervalo entre cada instante de tiempo en segundos

#Los nodos van desde 1 hasta n, pero los nodos 1 y n están apoyados, por lo que no tienen desplazamiento
#y el impacto no produce deformación.
p_a=11                               #Nudo que analizamos, entre 2 y n-1.
p_i=11                               #Posición del nudo donde aplico la fuerza, entre 2 y n-1

if(2<=p_i<=n+1 and 2<=p_a<=n+1)==False:
    print('\n ! Valor de "p" superior al número de puntos...\n')
    sys.exit()

elif (p_i==1 or p_i==1+n or p_a==1 or p_a==1+n):
    print('\n El modelo resuelto es una viga biapoyada. El desplazamiento en los nodos 1 y {} es nulo.\n'.format(n+1))
    sys.exit()
```

Ilustración 23. Input impacto.

La variable F0 representa la fuerza aplicada en Newtons, y T el tiempo de análisis de la respuesta en segundos. Podemos ver estos valores en la siguiente imagen, donde se representan de forma esquemática en la gráfica aproximada del impacto.

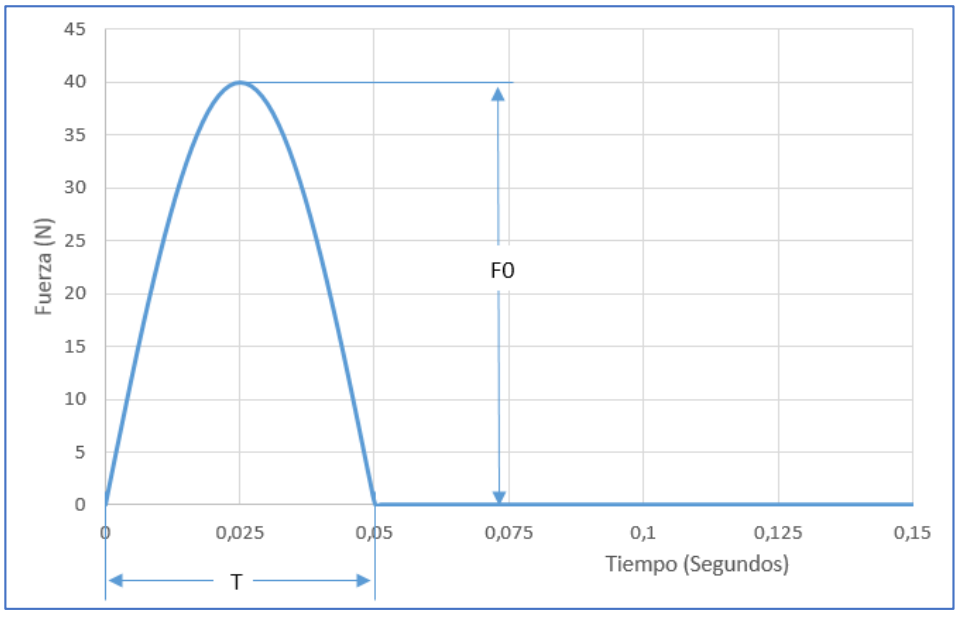


Ilustración 24. Aproximación impacto. Función seno.

La función "linspace(0,t0,intervalos)" nos permite separar un espacio de tiempo (0 → t0) en ciertos intervalos número de intervalos, definido por la variable "intervalos", que podemos cambiar a nuestro gusto para obtener mayor precisión.

La variable "h" representa el paso o duración en segundos de cada intervalo. Conocer este valor nos servirá posteriormente para resolver la ecuación de equilibrio de segundo orden.

Las variables p_a y p_i son especialmente importantes porque establecen cuál es el nudo que analizamos, y sobre qué nudo golpeamos con el martillo respectivamente. En la siguiente imagen vemos un esquema de cómo se distribuyen los puntos en la viga:

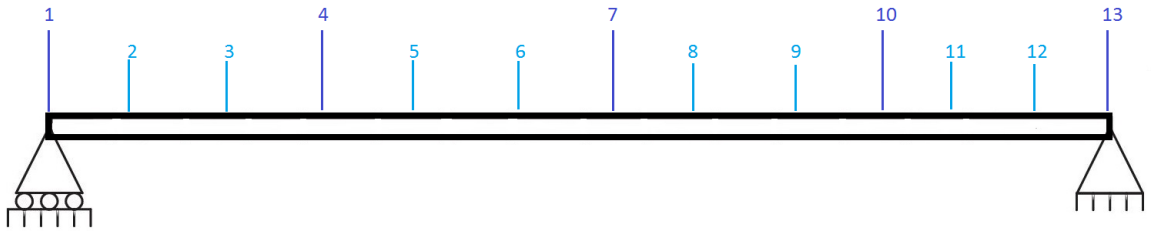


Ilustración 25. Distribución de nodos en la Viga

Como norma general, se ha establecido una división en cuatro barras, de tres elementos cada una. Esta elección se ha hecho al azar, pues nos permitirá más adelante seleccionar los nodos exactos en los que los primeros modos se manifiestan con mayor amplitud.

En la línea 48 se puede ver un aviso, que se muestra la consola en caso de seleccionar un punto de impacto superior al número de elementos de la barra. En caso de error, el mensaje salta en la pantalla, y automáticamente se cierra el sistema. En este caso el usuario debe modificar el punto a mano, cambiándolo directamente en el código. Un error común por parte del usuario puede ser introducir los puntos 1 ò 13, que carecen de interés en el análisis, ya que carecen de desplazamiento vertical.

6.3 Definición parámetros de las barras.

Como se explica en el apartado 2.2, el *model updating* se basa en cambiar los parámetros de definición de la estructura hasta que vemos que estos se adaptan a los resultados del laboratorio.

```
48 #Definición de cada una de las barras ----------------------------------------
49 #Mas adelante se pueden poner valores concretos para cada barra (model updating)
50 #Forma del vector: [Nelem,a,b,lp,E,rho]
51
52 Barras=([Nelem,a,b,lp,E,rho],
53         [Nelem,a,b,lp,E,rho],
54         [Nelem,a,b,lp,E,rho],
55         [Nelem,a,b,lp,E,rho])
```

Ilustración 26. Parámetros de las barras.

Las líneas de programación mostradas en la imagen superior muestran la matriz donde se encuentran las propiedades físicas e cada barra. Si queremos tener más divisiones en alguna de las barras podemos cambiar el número de elementos. Si queremos cambiar la anchura y el canto, podemos modificar los valores "a" y "b", así como "l_p" para la longitud. El módulo de Young y la densidad también son variables que podemos modificar. Es importante mencionar también que el espesor del metal no se puede cambiar directamente, ya que es un valor constante para todas las barras que nos permite calcular la masa considerada para cada barra como:

$$m_{Barra} = [abl_p - (a-e)(b-e)l_p]\rho$$

Durante las iteraciones, los valores de las variables de la matriz se modifican, a fin de ajustar los resultados. En caso de cambiar el número de barras en las que dividimos la viga, es necesario añadir una línea más en la matriz para que el programa no nos de error por incoherencia en las dimensiones.

6.4 Matriz de masas y rigidez de Bernouilli.

El siguiente paso es la construcción de las matrices de rigidez y masas tal y como se describió en el apartado 5 de este TFM.

```
59 #Matriz de masas y rigidez de Bernouilli----------------------------------------
60
61 M = zeros((2*(n+1),2*(n+1)))
62 K = zeros((2*(n+1),2*(n+1)))
63
64 #El bucle llama a una función que genera la matriz de masa y rigidez de ese
65 #elemento, y la coloca en su posición.
66 z=0 #Inicializo la variable z
67 for k in range(Nb):
68     Ks,Ms = Bernouilli(Barras[k],e)
69     index=k*(2*Nelem)
70
71     for z in range(Nelem):
72
73         K[index+2*z:index+2*z+4,index+2*z:index+2*z+4]+=Ks
74         M[index+2*z:index+2*z+4,index+2*z:index+2*z+4]+=Ms
```

Ilustración 27. Construcción de las matrices de masa y rigidez

El primer paso es crear las variables que contendrán los valores. "M" y "K" tienen este rol. La función "zeros()" construye una matriz cuadrada de dimensiones 2*(n+1), donde todos los términos son de la matriz valen 0. Esta función hace que la labor de construcción sea especialmente cómoda, ya que se editarán los valores que consideremos oportunos, mientras que los demás ya tendrán asignado el valor 0 por defecto.

La construcción de las matrices utiliza dos bucles:

- Primer bucle, que genera una matriz de masas y otra de rigidez, gracias a la función "Bernouilli()", comentada en el apartado 5. Estas matrices son las mismas para todos los elementos en los que se divide la barra, de modo que ya estamos listos para pasar al siguiente bucle.
- Segundo bucle, donde colocamos las matrices de masa y rigidez correspondientes a una misma barra. Las funciones "index+2*z: index+2*z+4" generan vectores de números desde el primer factor hasta el último. De esta forma colocamos las matrices una detrás de otra, adicionando cada factor, al correspondiente en la matriz final.

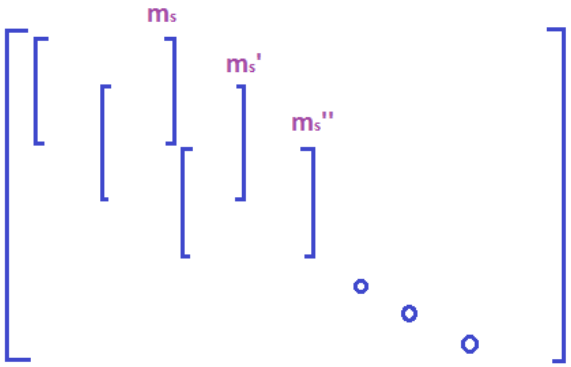


Ilustración 28. Esquema construcción de matrices con doble bucle

6.5 Adición de masas puntuales.

Como se especifica en la sección 3, donde se resume el ensayo en la viga real, los resultados prácticos se ven alterados en parte por la adición de los acelerómetros. Hay que tener en cuenta la posición en la que se sitúa cada receptor, a fin de conseguir unos resultados más representativos.

Para tener en cuenta el efecto de estas masas puntuales extra, sumamos su masa (~0,0577Kg) en cada uno de los puntos afectados.

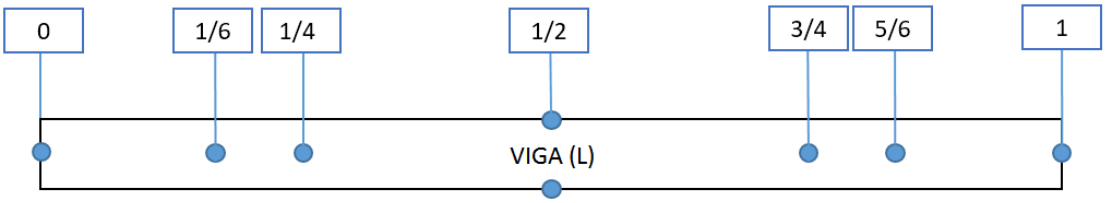


Ilustración 29. Posicionamiento de los acelerómetros en la viga de longitud "L"

Como vimos en la sección 5, los nodos afectados para 12 elementos son los que vemos en las líneas de programación de la imagen inferior:

```
106 #Añado las masas puntuales, para la correlación con el ensayo en el laboratorio
107 for i in [0,-2,12,12]: #Centro y extremos
108     M[i,i] += 0.0577
109 
110 for i in [6,18]: #Acelerómetros para el modo 2
111     M[i,i] += 0.0577
112 
113 for i in [4,20]: #Acelerómetros para el modo 3
114     M[i,i] += 0.0577
```

Ilustración 30. Posicionamiento masas puntuales

6.6 Condiciones de contorno.

La función delete(M, i,1) borra toda la fila "i" de números de la matriz M. Si en lugar de 1 hay un 0, se elimina la columna entera.

```
89 #Condiciones de contorno-----------------------------------------------------------
90 #Movimiento vertical y horizontal restringido en ambos extremos (Biapoyada)
91
92 for i in [0,-2]:
93     M=delete(M,i,1)
94     M=delete(M,i,0)
95     K=delete(K,i,1)
96     K=delete(K,i,0)
```

Ilustración 31. Condiciones de contorno

6.7 Frecuencias propias y modos propios.

Una vez tenemos las matrices de masa y rigidez con las que podemos trabajar, procedemos a resolver el sistema para hallar los modos propios y las frecuencias propias.

En las líneas 100 y 101 se halla la matriz inversa de M, y posteriormente se llama a la función "linalg.eig", que nos devuelve un vector (w2) que contiene las frecuencias propias al cuadrado ordenadas de mayor a menor, y una matriz (V) que contiene los modos propios ordenados en columnas de mayor a menor:

$$([K] - [M]\{\omega_2\})[V] = 0$$

```
98 #Hallar frecuencias y modos propios-----------------------------------------
99
100 invM = linalg.inv(M)            #Matriz inversa de la matriz de masas
101 w2,V = linalg.eig(invM @ K)     #Resolvemos el sistema,
102                                 #obteniendo las frecuencias propias al cuadrado, y los modos propios
103
104 #Ordenamos las frecuencias y vectores propios de menor a mayor-----------------
105
106 idx = w2.argsort()[0::]
107 w2 = w2[idx]
108 V = V[:,idx]                    #Las columnas son los modos propios de cada frecuencia ordenados.
109
110 for i in range(len(w2)):
111     w2[i]=linalg.norm(w2[i])    #Transformamos los números en reales para que el programa funcione correctamente
112                                 #Los valores imaginarios son 0, quiero eliminarlos para que no den problemas
113
114 frecuencia = list(1/(2*pi)*sqrt(w2)) #frecuencias propias en Hz
115
116 #Impresión de las tres primeras frecuencias propias
117
118 if len(frecuencia) >= 3:
119     message='f1 = {:.2f} Hz   f2 = {:.2f} Hz   f3 = {:.2f} Hz'.format(frecuencia[0],frecuencia[1],frecuencia[2])
120     print (message)
121 else:
122     message='f1 = {:.2f} Hz  f2 = {:.2f} Hz'.format(frecuencia[0],frecuencia[1])
123     print (message)
```

Ilustración 32. Frecuencias propias y modos propios

Para operar con este vector y la matriz, necesitamos que sus factores estén ordenados de menor a mayor. Por este motivo, el algoritmo de las líneas 106, 107 y 108 cambian las columnas de la matriz "V" de sitio, en función del puesto correspondiente del modo propio en el vector "w2".

En la línea 114 calculo el módulo de todos los componentes del vector w2. El motivo es que, muchas veces, estos tienen un residuo imaginario que viene de la función "linalg.ein". Siempre es igual a 0, por lo que no afecta para nada a los cálculos. Si embargo, varias operaciones posteriores pueden dar error, al no encontrarse con el formato de número que esperaban.

Finalmente, las frecuencias se imprimen por pantalla.

6.8 Representación de las formas modales

La matriz modal, nombrada "V" en el programa, contiene el valor de cada grado de libertad para cada modo:

$$[V] = \left[\begin{Bmatrix} \theta_{1,1} \\ \vdots \\ \theta_{2(n+1),1} \end{Bmatrix} \begin{Bmatrix} \Delta_{1,2} \\ \vdots \\ \Delta_{2(n+1),2} \end{Bmatrix} \begin{Bmatrix} \theta_{1,2} \\ \vdots \\ \theta_{2(n+1),2} \end{Bmatrix} \dots \begin{Bmatrix} \theta_{1,2(n+1)} \\ \vdots \\ \theta_{2(n+1),2(n+1)} \end{Bmatrix} \right]$$

Donde n es el número total de elementos en los que se divide la viga, el primer subíndice de cada factor es el modo de excitación y el segundo es el nodo afectado. Por ejemplo, el valor "$\Delta_{2,1}$" cuantifica cómo se ve afectado el nodo 1 por el modo 2 de vibración. Conociendo esta relación para cada punto, podemos dibujar la forma de los modos de vibración.

Es importante recordar que no hay columna para la traslación del nudo uno ni n+1, ya que el modelo es una viga biapoyada, y nos hemos deshecho previamente de estas columnas para trazar las matrices de masa y rigidez.

```
#Representación de los primeros modos de vibración-----------------------------

m1=array([0.]*(n+1))        #Contiene la amplitud del modo 1, para cada nodo de la viga
m2=array([0.]*(n+1))        #Contiene la amplitud del modo 2, para cada nodo de la viga
m3=array([0.]*(n+1))        #Contiene la amplitud del modo 3, para cada nodo de la viga
m4=array([0.]*(n+1))        #Contiene la amplitud del modo 4, para cada nodo de la viga

x=linspace(1,n+1,n+1)       #Posiciones en la viga, siendo los nodos 1 y n+1 los nodos apoyados

for i in range (1,n):       #Completa los vectores con la amplitud de los cuatro primeros
    m1[i]=V[i*2-1,0]        #modos, en todos los puntos de la viga.
    m2[i]=V[i*2-1,1]
    m3[i]=V[i*2-1,2]
    m4[i]=V[i*2-1,3]

t_1, c_1, k_1 = interpolate.splrep(x, m1, s=0, k=4) #splrep devuelve los nodos, los coeficientes
t_2, c_2, k_2 = interpolate.splrep(x, m2, s=0, k=4) #para calcular el B-Spline y el grado del spline.
t_3, c_3, k_3 = interpolate.splrep(x, m3, s=0, k=4)
t_4, c_4, k_4 = interpolate.splrep(x, m4, s=0, k=4)

xnew =linspace(min(x), max(x), 4*len(x)) #Puntos en los cuales se interpola el spline.

spline1 = interpolate.BSpline(t_1, c_1, k_1, extrapolate=False) #Crea una función que calcula el spline para
spline2 = interpolate.BSpline(t_2, c_2, k_2, extrapolate=False) #el conjunto de puntos xnew
spline3 = interpolate.BSpline(t_3, c_3, k_3, extrapolate=False)
spline4 = interpolate.BSpline(t_4, c_4, k_4, extrapolate=False)


plt.figure(1)   #Plot de los modos de vibración.
plt.plot(xnew,spline1(xnew),label='MODO 1')
plt.plot(xnew,spline2(xnew),label='MODO 2')
plt.plot(xnew,spline3(xnew),label='MODO 3')
plt.plot(xnew,spline4(xnew),label='MODO 4')
plt.title('Representación de los modos de vibración ')
plt.legend(loc='best')
plt.xlabel('Posición en la viga')
plt.ylabel('Amplitud')
```

Ilustración 33. Representación de los modos propios

En primer lugar se crean los vectores que contienen la traslación de todos los nodos, para cada modo de vibración. Como la división que se propone es de 12 elementos (n=12),

tenemos que tratar con 13 nodos. Con esta premisa se establece un vector x, que no es más que un vector que contiene números enteros equiespaciados, del 1 a n+1.

El bucle de la fila 142 recoge los valores adecuados de la matriz modal. Para cada modo recorre la matriz V en la fila, anotando el valor de las traslaciones. Este algoritmo introduce valores en el vector m desde la posición 1 hasta la n-1. La razón es que el valor de los nodos extremos es conocido e igual a 0. El resultado para el primer modo es el siguiente:

[0, -0.09162469, -0.17715782, -0.25097685, -0.30783553, -0.34359054, -0.35580382, -0.34359054, -0.30783553, -0.25097685, -0.17715782, -0.09162469, 0]

A continuación se utiliza la función BSpline, dentro de la librería interpolate, para crear la función "spline", que interpola en el rango de puntos deseado (xnew). El polinomio que se elige es de cuarto grado, y da unos resultados muy buenos en apariencia.

6.9 Frecuencias propias y modos propios.

Para hallar las matrices reducidas a la masa, es necesario en primer lugar hallar "D", el cual definimos como un vector de igual longitud que "Mdiag". Este vector contiene las constantes por las que tengo que dividir la matriz "V" de modos propios, para que a la hora de diagonalizar:

- La matriz de masas diagonalizada sea la matriz identidad
- La matriz de rigidez diagonalizada tenga en su diagonal los valores de las frecuencias propias al cuadrado.

```
125 #Matriz de masa y de rigidez diagonalizadas-----------------------------------
126
127 Mdiag=V.transpose() @ M @ V
128 D=[None]*len(Mdiag)
129
130 #Hallar las matrices reducidas a la masa-----------------------------------------
131
132 for i in range(len(Mdiag)):
133     D[i]=sqrt(Mdiag[i,i])
134     V[:,i]=V[:,i]/D[i]      #Divido cada modo propio por la raíz del valor en la diagonal
135                             #Ahora la matriz de masas modal es la matriz identidad.
136
137 Mdiag=V.transpose() @ M @ V #Resultando la matriz identidad
138 Kdiag=V.transpose() @ K @ V #En la diagonal contiene los valores de w2
139
```

Ilustración 34. Reducción a la masa

6.10 Matriz de amortiguamiento.

A continuación, se implementa la matriz de amortiguamiento, tal y como se describe en el punto 5. Para ello, es necesario introducir los valores a1, que equivale a alfa, y b1, que equivale a beta.

```
#Matriz de amortiguamiento-----------------------------------------------------

a1=0.52350036                       #Alfa, Valores calculados para obtener las amortiguaciones teóricas
b1=7.94311582E-06                   #Beta
C=a1*M+b1*K
Cdiag=V.transpose() @ C @ V         #La matriz es diagonal
Cdiag[1,1]=0.0058*2*sqrt(w2[1])

xi=[None]*len(w2)                   #Factores de amortiguamiento crítico para cada modo propio

for i in range(len(w2)):
    xi[i]=Cdiag[i,i]/(2*sqrt(w2[i])) #El valor de xi es simplemente xi= a1/(2*sqrt(w2[i]))+b1*sqrt(w2[i])/2
```

Ilustración 35. Matriz de amortiguamiento.

6.11 Resolución de la EDO.

En primer lugar se definen las columnas de la matriz que vamos a tratar. La enumeración de los nodos es la siguiente:

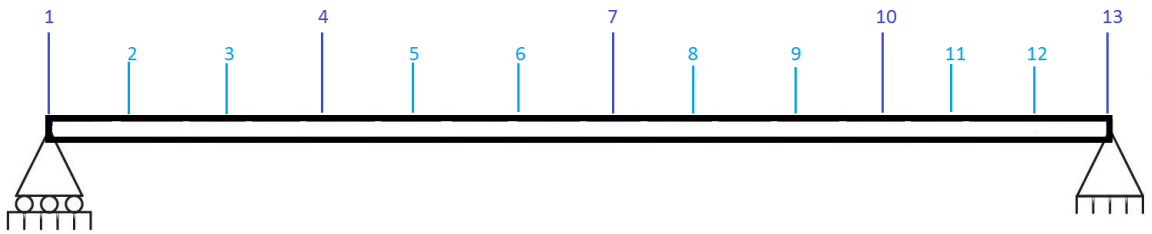


Ilustración 36. Enumeración de los nodos de la viga.

El nodo 1 y el 13 no nos interesan, dado que su desplazamiento vertical es nulo al estar apoyados. Cada desplazamiento vertical de un nodo corresponde a una fila en el vector desplazamiento:

$$\{q\} = \begin{Bmatrix} \theta_1(t) \\ y_2(t) \\ \theta_2(t) \\ y_3(t) \\ \theta_3(t) \end{Bmatrix}$$

Por ejemplo, si quiero obtener el desplazamiento del nodo 3, necesito acceder a la fila 3 del vector q, pero le desplazamiento del nodo 12 corresponde a la fila 21. Definimos por lo tanto la función que vemos en la línea 162, para definir cuál es la fila adecuada de q:

```
#Resolución numérica con Odeint()----------------------------------------------

p_a=2*p_a-3                         #Elijo la fila cuyos valores en la matriz corresponden
p_i=2*p_i-3                         #al desplazamiento vertical del nudo elegido.

f_matriz=zeros((len(w2),len(t)))
f_matriz[p_i,:]=Imp(t,T,F0)         #Función impacto en el punto "p_i", durante el tiempo de ensayo t

f_cte=[0]*len(w2)                   #Vector impacto.
f_cte[p_i]=F0                       #Se agrega el valor pico. Constante que multiplica el seno.
f_cte_r= V.transpose() @ f_cte      #Premultiplicación por la matriz de paso.
```

Ilustración 37. Definición de los vectores de impacto.

A continuación se define la matriz impacto (f_matriz), que contiene, en la fila "p_i", la función "Imp", la cual es la función medio seno de la que se habla en la sección 5,2:

$$[f_{matriz}] = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ f(t_1) & f(t_2) & \cdots & f(T) & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \end{bmatrix}$$

Esta matriz se usará más adelante para calcular la aceleración del nodo. Además se define el vector "f_cte", que contiene únicamente el valor pico de la función seno, en la fila correspondiente al nodo del impacto. Este vector se premultiplica por la matriz de modos traspuesta.

Seguidamente se aplica el método de resolución con la función "odeint", definiendo primero la función "Vibración", que baja un grado la ecuación diferencial, aplicando el cambio de variable:

$$x(t) = \dot{q}(t) \rightarrow \{y(t)\} = \begin{Bmatrix} q(t) \\ x(t) \end{Bmatrix}; \{\dot{y}(t)\} = \begin{Bmatrix} x(t) \\ \dot{x}(t) \end{Bmatrix}$$

Donde sabemos que:

$$\dot{x} = \ddot{q} = [M]^{-1}(\{f(t)\} - [C]\{\dot{q}(t)\} - [K]\{q(t)\})$$

Como se explica en el apartado 5.3.2.

```
def Vibracion(y,t,fr,T,cr,w2r):                #Función vibración para operar con odeint
    q,x=y                                      #x es una variable auxiliar que baja un grado la ecuación diferencial
    if t > T:                                  #Definición de la función impulso, tras el golpeo (T)
        t=0                                    #no hay más excitación.
    dydt=[x,fr*sin(pi*t/T)-cr*x-w2r*q]         #Definición de la función
    return dydt                                #Devuelve la derivada de la variable y

y0=[0.0,0.0]                        #Condiciones iniciales en desplazamiento y velocidad
r=zeros((len(w2),len(t)))           #Contiene los valores de la variable modal
r_vel=zeros((len(w2),len(t)))       #Contiene los valores de la derivada de primer orden de la variable modal

for i in range(len(w2)):            #Resolución del desplazamiento de todos los puntos
    cr=Cdiag[i,i]
    w2r=w2[i]
    fr=f_cte_r[i]

    sol=odeint(Vibracion,y0,t,args=(fr,T,cr, w2r))
    r[i,:]=sol[:,0]
    r_vel[i,:]=sol[:,1]

```

Ilustración 38. Resolución de la EDO con la función odeint()

La solución de la función odeint se almacena en dos matrices r, y r_vel: la primera contiene la solución de {q} para cada instante de tiempo, mientras que la segunda contiene los valores de la primera derivada.

Finalmente volvemos a las variables físicas.

```
#Volver a las variables físicas de desplazamientos-----------------------------
q=V@r                                   #Desplazamiento de cada nodo.
velocidad=V@r_vel                       #Velocidad de cada nodo.
aceleracion=invM@(f_matriz-C@velocidad-K@q)     #Aceleración de cada nodo. Solución de la EDO.

Excel_Vel_Acc(t,q,aceleracion,p_a)        #Crea un excel con los datos del desplazamiento y aceleración
```

Ilustración 39. Deshacer el cambio de variable para volver a las variables físicas

La línea 231 ejecuta la función "Excel_Vel_Acc", que podemos encontrar en los anexos. Su funcionamiento es muy sencillo: crea una hoja de Excel, graba en ella los nodos afectados y añade los valores de tiempo, velocidad y aceleración del nodo en tres columnas diferentes. El análisis de datos se facilita en gran medida cuando están directamente en el fichero Excel. La exportación de los datos de la consola copiando y arrastrando a la hoja de Excel es muy problemático, ya que da errores de formato. Esta función se encarga de hacer posible esta labor rápidamente.

Para terminar con este apartado, la consola muestra dos gráficas: Aceleración vs tiempo y desplazamiento vs tiempo. Todas las gráficas del programa tienen la misma forma, incluyendo nombre en ambos ejes y valor máximo para el eje x.

Destacar simplemente que en la representación gráfica de desplazamientos, se multiplican las traslaciones por 1000 para obtener un valor más sencillo de interpretar, en mm en lugar de metros.

```
201 #Plot del desplazamiento del punto de análisis---------------------------------
202
203 plt.figure(1)
204 plt.plot(t[1:],q[p_a,1:]*1000,label='Solución en desplazamientos')
205 plt.title('Desplazamiento del punto {}'.format(int((p_a+3)/2)))
206 plt.legend(loc='best')
207 plt.xlabel('Tiempo (s)')
208 plt.ylabel('Desplazamiento (mm)')
209 plt.xlim(xmax=t0)
210
211 plt.figure(2)
212 plt.plot(t[1:],aceleracion[p_a,1:],label='Solución aceleración')
213 plt.title('Aceleracion del punto {}'.format(int((p_a+3)/2)))
214 plt.legend(loc='best')
215 plt.xlabel('Tiempo (s)')
216 plt.ylabel('Aceleracion (m/s2)')
217 plt.xlim(xmax=t0)
```

Ilustración 40. Representación gráfica de la velocidad y la aceleración del nodo analizado.

6.12 Respuesta en frecuencia.

Las últimas líneas del programa se encargan de calcular la FRF de la estructura para un nodo concreto excitado por un impacto. En primer lugar se define el vector "Rec", en el cual se almacenan los valores de la receptancia para un cierto rango de frecuencias.

Posteriormente se define este rango de frecuencias, nombrado como "w_vector". La forma en que está definido nos permite aumentar o reducir el rango de frecuencias en función de las frecuencias naturales de vibración. En la imagen inferior el rango de frecuencia es de cero hasta w2[5], que corresponde al quinto modo propio.

El bucle de la línea 265 realiza la ecuación definida para la receptancia:

$$\alpha_{jk}(\omega) = \sum_{r=1}^{N} \left(\frac{[V]_{jr}[V]_{kr}}{\omega_r^2 - \omega^2 + i2\xi_r\omega_r\omega} \right)$$

La receptancia es calculada con un bucle encadenado. El primero introduce en la fórmula la frecuencia a la cual se calcula la receptancia y el segundo realiza el sumatorio, que tiene una longitud igual al número de elementos en los que hemos dividido la viga.

```
#Cálculo de la respuesta en frecuencias de la estructura-----------------------

Rec=zeros((len(t)),dtype=complex)           #Contiene la receptancia del nodo p_a, producida por una excitación
                                            #en el nodo p_i para un cierto rango de frecuencias
w_vector=linspace(0,sqrt(w2[5]),len(t))     #Define el rango de frecuencias en el que vamos a calcular la FRF

j=complex(0,1)                              #Defino la variable compleja

#Cálculo de la receptancia de cada nodo, para el rango de frecuencias w_vector.
for i in range(len(t)):
    for k in range(len(w2)):
        Rec[i]+=V[p_a,k]*V[p_i,k]/(-w_vector[i]**2+w2[k]+2*sqrt(w2[k])*xi[k]*w_vector[i]*j)

Accel=w_vector**2*Rec                       #Cálculo de la acelerancia de cada nodo a partir de la receptancia.
f_vector=w_vector/(2*pi)                    #Conversión de la frecuencia de rad/s a Hz.

"""Excel_FRF(f_vector,Accel,p_a,p_i)"""
```

Ilustración 41. Cálculo de la respuesta en frecuencia de la estructura

Finalmente se calcula la acelerancia a partir de la receptancia y se pasa la frecuencia de rad/s a Hz para que la gráfica sea más comprensible.

La función "Excel_FRF()" se utiliza para pasar las FRF a un documento Excel. Funciona de la misma forma que la función "Excel_Vel_Acc()". Es necesario tener esta aplicación si se quieren exportar los datos, ya que no se pueden copiar y pegar directamente de la consola a una hoja de Excel.

```
plt.figure(4)

plt.plot(f_vector[1:-5],abs(Accel[1:-5]),'b-')
plt.yscale('log')
plt.title('Respuesta en frecuencia del nudo {}\n con excitación armónica en el nudo {}'.format(int((p_a+3)/2),format(int((p_i+3)/2))))
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Amplitud (m/N)')
plt.legend(['Acelerancia - Módulo'],loc=0)

plt.figure(5)

plt.plot(f_vector[1:-5],angle(Accel[1:-5],deg=True),'b-')
plt.title('Respuesta en frecuencia del nudo {}\n con excitación armónica en el nudo {}'.format(int((p_a+3)/2),format(int((p_i+3)/2))))
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Amplitud (m/N)')
plt.legend(['Acelerancia - Fase'],loc=0)
```

Ilustración 42. Plot respuesta en frecuencia.

7. Ejemplos de verificación y aplicación.

A continuación se muestran los resultados de las variables que se han utilizado para verificar el *model updating*.

7.1 Frecuencias propias.

Para poder decir que el modelo de *Python* representa fielmente el comportamiento de la viga, es esencial ajustar los modos propios. Éstos nos dicen cómo responde la estructura a una excitación cualquiera. Las frecuencias que tienen más influencia en la respuesta de la viga son las correspondientes a los primeros modos. En esta viga me he centrado en los tres primeros.

En la siguiente tabla se puede ver una comparación de los resultados obtenidos con el programa, y los obtenidos mediante experimentación en el laboratorio.

Modo	Frecuencia laboratorio (Hz)	Frecuencia aproximada (Hz)	E.relativo (%)
1	3,71 Hz	3,70 Hz	0%
2	14,59 Hz	14,70 Hz	0,14%
3	32,03 Hz	32,66 Hz	2,05%

El error obtenido es bastante pequeño, e incluso nulo para el primer modo. Lo cual es bastante favorable si tenemos en cuenta que es el predominante. Conseguir unas aproximaciones tan buenas ha sido posible gracias al *model updating* que he llevado a cabo en la definición de la viga:

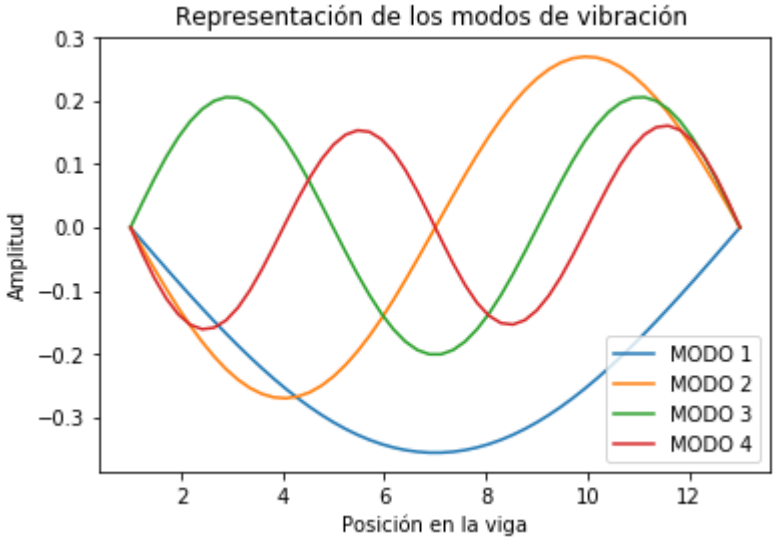
```
47 #Definición de cada una de las barras ----------------------------------------
48 #Model updating donde se pueden poner valores concretos para cada barra
49 #Forma del vector: [Nelem,a,b,lp,E,rho]
50
51 Barras=([Nelem,1.01*a,b,lp,E,0.94*rho],
52         [Nelem,a,b,lp,0.985*E,rho],
53         [Nelem,a,b,lp,0.985*E,rho],
54         [Nelem,1.01*a,b,lp,E,0.94*rho])
55
```

Ilustración 43. Model updating de la viga, con división en cuatro barras.

Los sucesivos cambios en las propiedades de las barras aproximan el valor de la frecuencia. El método utilizado ha sido de prueba y error, teniendo en cuenta solo dos condiciones:

- Es recomendable que el resultado sea una estructura de propiedades simétricas.
- Intentar no descompensar mucho alguna de las propiedades, alejándonos lo menos posible de su valor experimental.

Adicionalmente he calculado la representación de los modos propios. Dibujando la forma de los modos me aseguro de que éstos tienen la forma seno teórica. La matriz de modos tiene, por columnas, el valor de cada grado de libertad para cada modo. Utilizo solo los correspondientes a la translación, imponiendo que los extremos no pueden moverse. El resultado es la siguiente gráfica:



El eje vertical muestra la amplitud de vibración, mientras que el horizontal indica la posición en la viga, como ya se ha mencionado varias veces, del uno al trece. Vemos que las formas modales se parecen bastante a la representación senoidal teórica. Es necesario mencionar que la curva que conecta las amplitudes se ha conseguido con un spline de cuarto grado que suaviza la forma de del modo.

7.2 Aceleración nodal tras un impacto.

Una manera más de verificar la semejanza del modelo con la realidad es mediante la comparación de la aceleración nodal en la respuesta a un impacto. La aceleración experimental corresponde a la obtenida en los test del laboratorio, mientras que la curva de *Python* es calculada resolviendo la EDO:

$$[M]\{\ddot{x}\} + [C]\{\dot{x}\} + [K]\{x\} = \{f(t)\}$$

Y despejando la segunda derivada del vector desplazamiento {x}. Para obtener este vector ha sido necesario establecer el factor de amortiguamiento para cada modo, aproximándolo, como se menciona en el apartado 5.3.2, a su valor teórico.

Modo	ξ teórico (%)	ξ aproximado (%)	E.relativo (%)
1	1,13 %	1,13 %	0,00 %
2	0,58 %	0,32 %	44,6 %
3	0,21 %	0,21 %	0,00 %

La tabla superior muestra los valores obtenidos para los factores de amortiguamiento modal. La expresión del factor de amortiguamiento no nos permite ajustar más de dos grados de libertad, por ese motivo los factores de los modos uno y tres no tienen error relativo, mientras que el del modo dos se dista más del valor teórico. He elegido ajustarlo al primer y tercer modo por conveniencia, ya que en caso de ajustar el factor uno y dos, el tercero tiene un valor muy superior al teórico, y resulta en una peor aproximación.

Una vez definida la matriz de amortiguamiento gracias a este último parámetro, estamos en condiciones de resolver la ecuación diferencial para la aceleración:

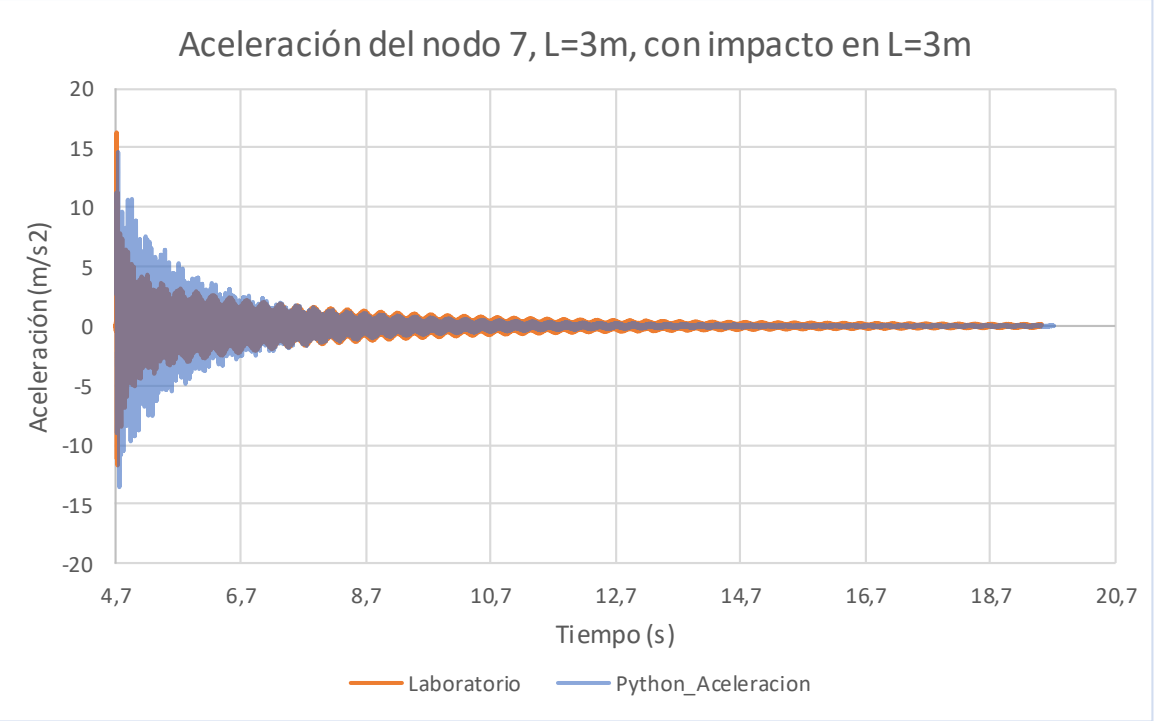


Ilustración 44. Aceleración del nodo 7 tras una excitación en el mismo nodo

La primera gráfica superior muestra la comparación entre la respuesta en aceleración del nodo 7 (es decir el nodo central) tras un impacto en el mismo nodo. A primera vista se puede comprobar que son bastante similares, la simulación de *Python* muestra una amplitud algo más elevada al inicio, pero vemos que siempre se mantiene en el mismo rango que la obtenida en el laboratorio. Además ambas tienen la misma forma, y los dos llegan al estado de reposo en tiempos muy similares.

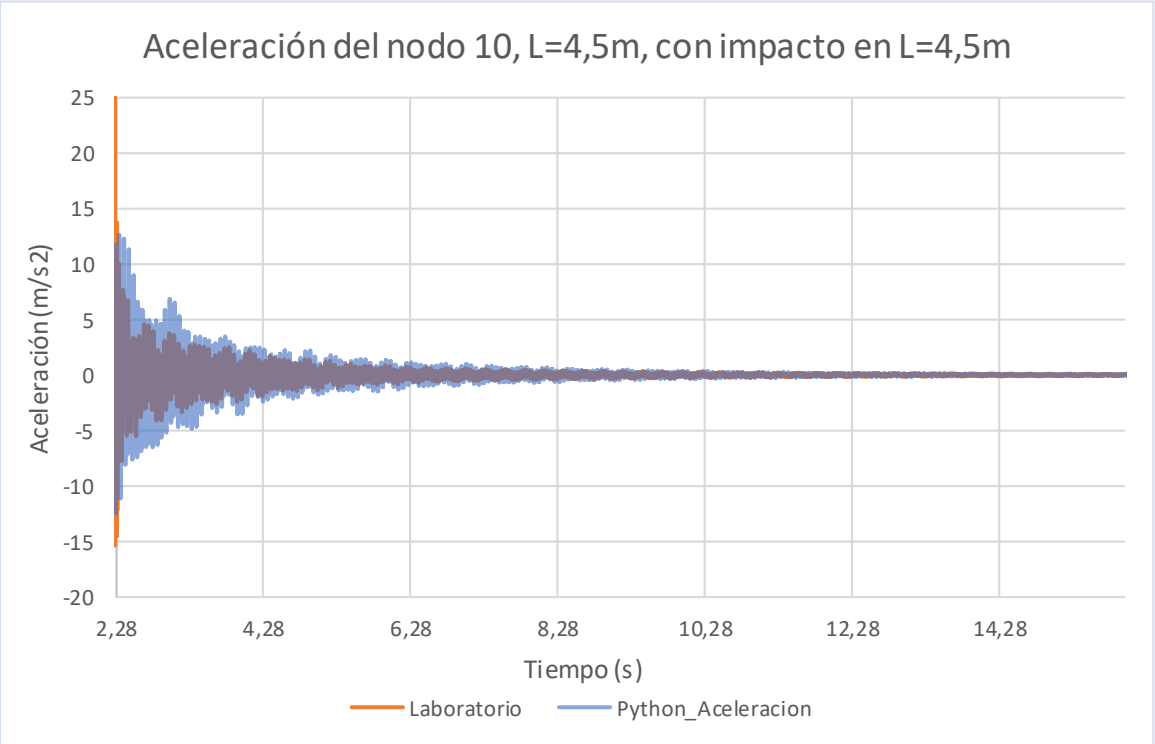


Ilustración 45. Aceleración del nodo 10 tras una excitación en el mismo nodo

El mismo razonamiento es aplicable para la respuesta nodal del nodo 10. Ambas curvas son muy similares, con la salvedad de que la simulación en *Python* muestra una amplitud mayor. Sin embargo, ambas son estables al mismo tiempo.

7.3 Respuesta nodal en frecuencia.

A continuación se muestran los las respuestas en frecuencia para tres nodos diferentes. Una vez más, se analiza la semejanza entre los valores obtenidos en el laboratorio, y los logrados a partir de la simulación en *Python*.

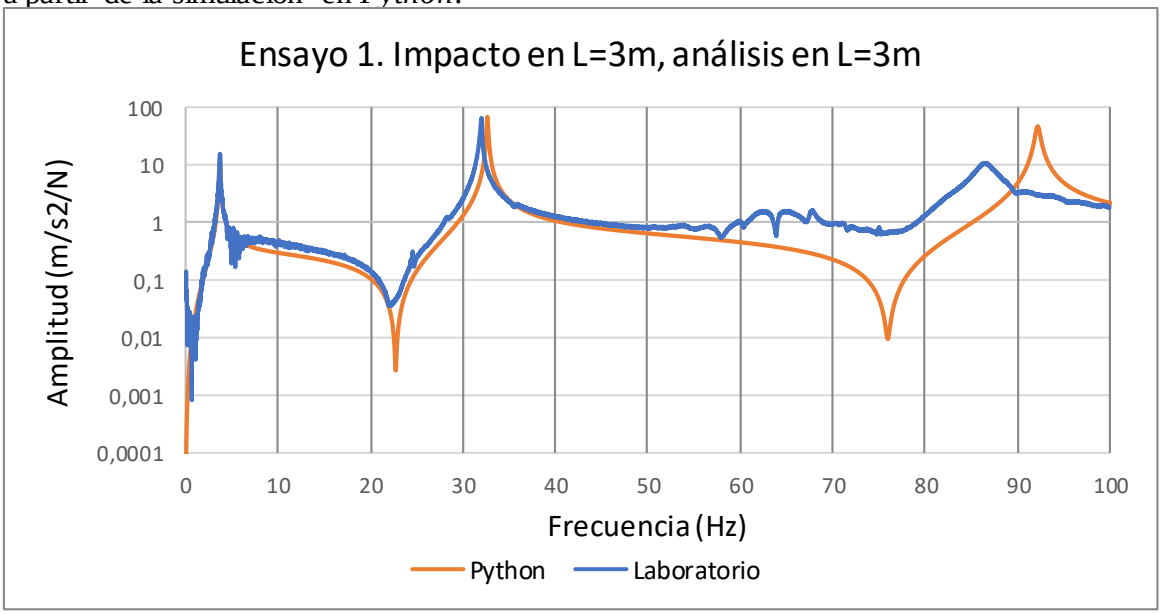


Ilustración 46. Respuesta en frecuencia del nodo 7 tras un impacto en el nodo 7

En la gráfica superior se puede ver que las curvas se superponen para el primer y tercer modo. Como ya se ha explicado previamente no vemos el pico correspondiente al segundo modo de vibración porque estamos golpeando en un nudo y por lo tanto su amplitud es cero. Sin embargo sendas curvas empiezan a separarse a partir de los 60 Hz. En este instante se aprecia un atraso del modelo en *Python* respecto al modelo experimental en el laboratorio, dando lugar a picos que aparecen más tarde de lo que deberían. Este error se puede deber a diversos factores:

- El análisis a frecuencias elevadas implica que las medidas de los acelerómetros tienen más error, de forma que al superar los 60 Hz puede que estemos en un rango en el que los sensores no son capaces de reflejar los datos fielmente.

- El modelo elegido para la simulación puede no ser una representación exacta para la estructura del laboratorio. Hemos supuesto una viga biapoyada para la simulación en *Python*, mientras que la estructura del laboratorio está sostenida por dos células, que no son perfectamente rígidas.

A modo ilustrativo, se pueden ver las siguientes dos gráficas, donde se repite la misma dinámica que en el primer ejemplo, pero para dos ensayos distintos:

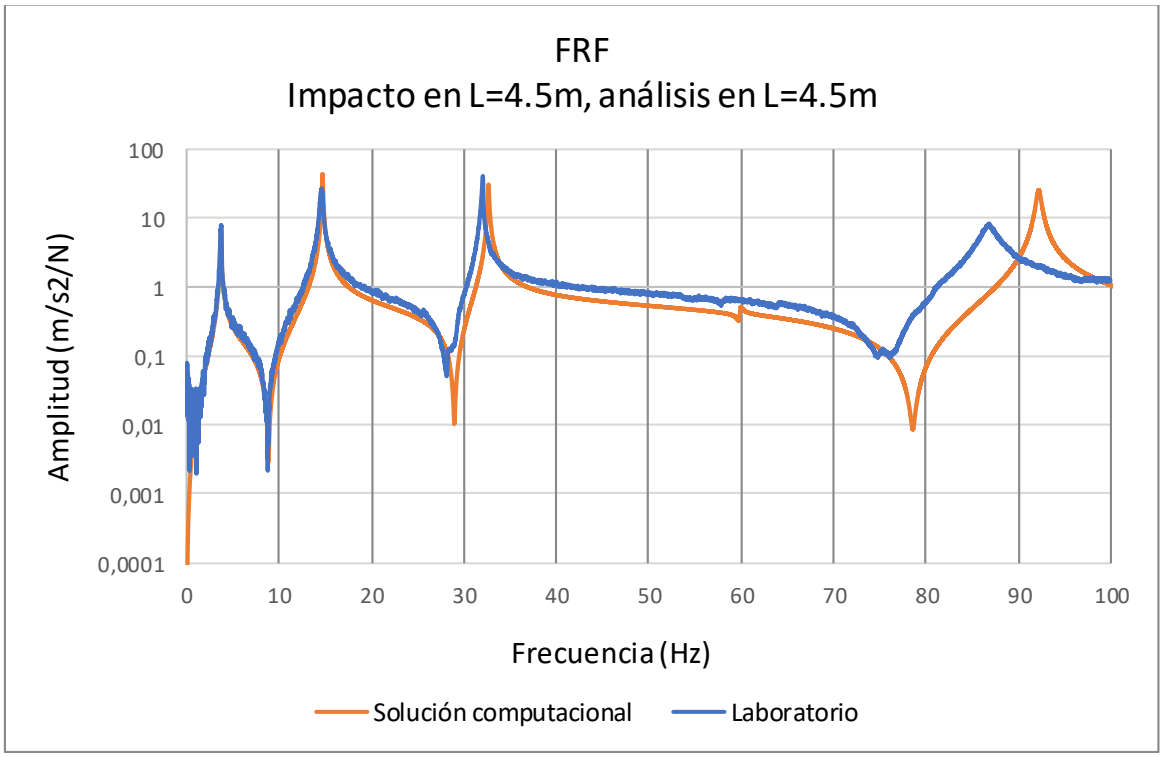


Ilustración 47. Respuesta en frecuencia del nodo L=4,5m tras un impacto en el nodo L=4,5m

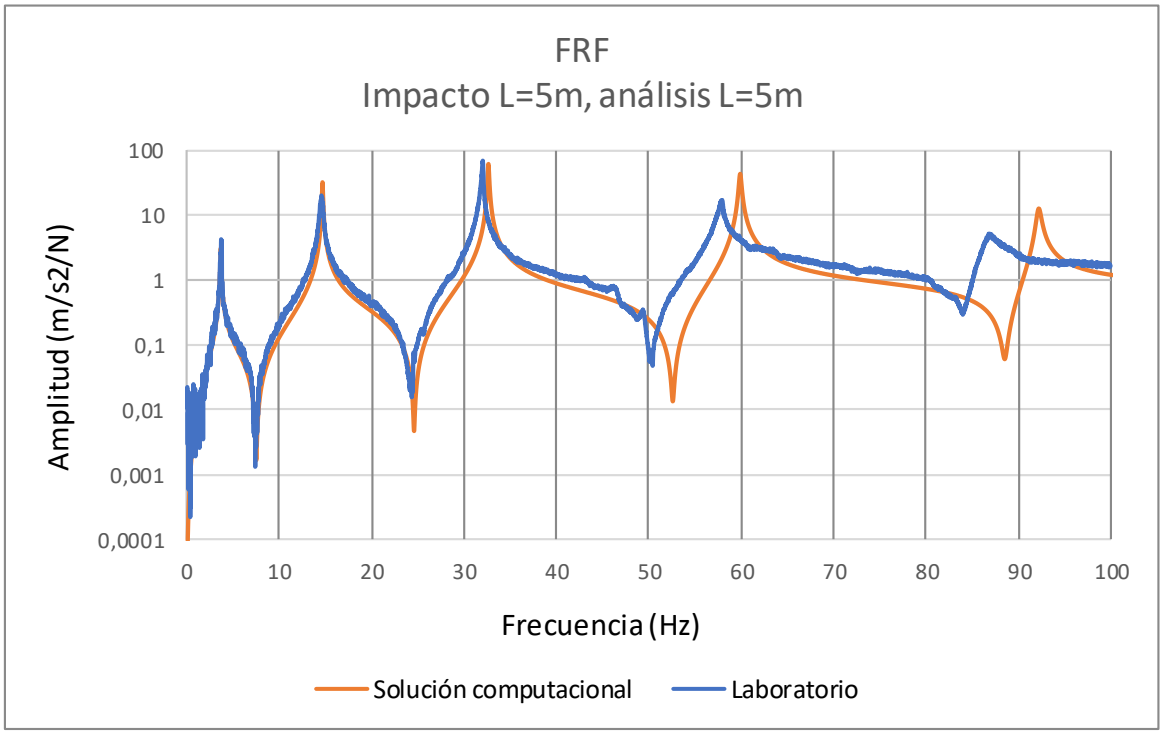


Ilustración 48. Respuesta en frecuencia del nodo L=5m tras un impacto en el nodo L=5m

7.4 Aplicación práctica. Comparación con el resultado teórico.

El objetivo de esta aplicación práctica, cuyo programa es nombrado como "aplicación1", es comprobar si la configuración del modelo sigue siendo válida para una viga más pesada. Para este ejemplo en concreto, se valora la viabilidad para una viga 2 kilogramos más pesada.

- Solución analítica:
  La expresión analítica de la frecuencia natural, para cada modo para una viga continua biapoyada es la siguiente:

$$f_i(Hz) = \frac{\lambda_i^2}{L^2}\left(\frac{EI}{w}\right)^{\frac{1}{2}} \quad , con\ \lambda_i = i\pi$$

  Donde L es la longitud de la viga, E el módulo de *Young*, w es la masa por unidad de longitud e I el momento de inercia.

  Dado que la masa cambia para ser 2kg mayor, también lo hace la masa por unidad de longitud. Ésta es la única propiedad que cambia dentro de la expresión anterior. Es importante destacar que también hay que añadir los 8 acelerómentros, de 57,7 gramos cada uno. Llamamos w' a la nueva masa por unidad de longitud, definida como:

$$w' = \frac{m + 2 + 8 \cdot 0{,}0577}{L} = \frac{\rho LA + 2{,}4616}{L}$$

- Ajuste de la viga en *Python*:
  Para ajustar el modelo en *Python* es necesario cambiar la variable ρ, de modo que la nueva viga tenga una masa $m' = m + 2$. La nueva densidad, definida en *Python* como "rho" (línea 45), se calcula:

$$\rho' = \frac{m+2}{AL}$$

  Las masas correspondientes a los acelerómetros se suman ya en el modelo, por lo que no es necesario añadirlos en la densidad.

```
39 #Edit del modelo, con los tornillos repartidos uniformemente
40
41 A=b*a-(a-2*e)*(b-2*e)              #Área transversal de la viga en m^2
42 Vol=L*A                           #Volumen total de la viga en m^3
43 Masa=rho*Vol                      #Masa total de la viga en kg
44 Masa_prima=Masa+2+0.0577*8        #Nueva masa de la viga añadiendo los 2kg uniformemente repartidos, y la masa
45                                   #correspondiente a los acelerómetros
46
47 rho_prima=Masa_prima/Vol          #Densidad aplicada a la solución analítica.
48
49 rho=(rho*Vol+2)/Vol                #Densidad aplicada al modelo de python. Posteriormente se añade la masa de
50                                   #los acelerómentros en las matrices de cada elemento.
```

Ilustración 49. Caracterización de una viga 2kg más pesada.

El funcionamiento del programa es exactamente igual que el desarrollado en el resto del documento, con la salvedad de que no se ejecutan las líneas correspondientes a la respuesta en frecuencia, ya que solo nos interesa el análisis formal de modos. Los resultados obtenidos en *Python*, comparados con la expresión analítica se muestran a continuación:

Modo	Frecuencia modelo(Hz)	Frecuencia analítica (Hz)	E. Relativo (%)
1	3.26 Hz	3.30 Hz	0.3 %
2	13.43 Hz	13.18 Hz	1.9 %
3	30.08 Hz	29.66 Hz	1.4 %

Los resultados muestran una correspondencia muy buena, vemos un desvío un poco mayor para el segundo modo, pero aún así no sobrepasa el 2% de error relativo. Este breve ensayo confirma que el diseño del modelo está bien correlado con el tipo de estructura, y por lo tanto nos sirve para hacernos una idea bastante buena de la respuesta de la viga.

8. Conclusiones y recomendaciones.

8.1 Conclusiones

Las principales conclusiones que se extraen del trabajo pueden resumirse en las siguientes:

- El modelo de elementos finitos es una buena aproximación al comportamiento real de la estructura. Incluso cuando la barra mide seis metros, la división en 12 elementos, y por lo tanto 13 nodos, nos permite obtener unos resultados bastante buenos. Se han lanzado simulaciones aumentando el número de elementos y las características dinámicas obtenidas son muy similares las de 12 elementos. El ahorro de tiempo es importante ya que se reduce considerablemente la cantidad de operaciones necesarias.

- A pesar de que las masas de los acelerómetros son bajas (0.057 kg), su inclusión en el modelo es necesaria para conseguir una buena aproximación. El hecho de que éstas se añadan directamente en la matriz de masas cambia ligeramente las frecuencias naturales de vibración, lo que hace posible su ajuste mediante el *model updating*.

- Se han conseguido unas aproximaciones muy realistas para las frecuencias de vibración natural y las formas modales. El estudio se ha centrado en ajustar los tres primeros modos, ya que son las formas modales que predominan en la deformada de la barra. Como consecuencia, los picos de la respuesta en frecuencia tienen también una correspondencia muy buena con los resultados del laboratorio.

- En altas frecuencias la FRF del modelo muestra una pequeña desviación que le hace separarse de las curvas del laboratorio. Es decir, los primeros cuatro modos tienen una buena correspondencia, pero a partir del quinto encontramos ligeros errores. Esto demuestra que la caracterización de la barra en el laboratorio no es una tarea sencilla, los resultados del laboratorio dependen en gran medida de la destreza del ingeniero, y por lo tanto es muy importante ser muy meticuloso.

8.2 Recomendaciones

Dado que este trabajo trata del desarrollo de un modelo en *Python*, tareas como la modificación de parámetros o la adición de funciones resultan muy sencillas. El programa es una herramienta didáctica muy potente, ya que podemos ver la respuesta de la estructura para un conjunto de parámetros antes de ver la respuesta en el laboratorio. Por lo tanto, gracias a estas múltiples posibilidades que ofrece *Python*, surgen distintas líneas futuras de desarrollo del trabajo, entre las que destacan las siguientes:

- Para comprobar que los resultados de la aplicación 1 son correctos sería conveniente realizar el ensayo de forma práctica. Es posible simular el aumento de masa de la viga añadiendo una tuerca de 0.01kg cada 30mm, de forma que obtenemos 2kg adicionales. Este caso no se corresponde exactamente a una carga uniforme, pero el error es mínimo.

- Si se modeliza en el programa la adición de una masa central en la viga, de por ejemplo 0.5 kg, podríamos ver cómo los modos y respuestas en frecuencia de la estructura cambian, constituyendo este un ejercicio muy interesante para el usuario. Este ensayo es relativamente simple de lanzar en el laboratorio, por lo que podríamos usar los resultados para compararlos con el modelo real y así comprobar si el modelo responde correctamente. La idea detrás de la adición de la masa es comprobar cómo afectaría a una estructura de este tipo, véase una pasarela, el tránsito de un coche.

- Dada la naturaleza de la estructura es posible añadir nuevas condiciones, complicando el modelo, que nos permitan representar el comportamiento de estructuras más complejas, como una pasarela o un puente. Es una tarea que puede resultar costosa, ya que implica hacer un *model updating* quizás para un mayor número de secciones, pero es definitivamente viable.

9. Referencias bibliográficas.

[1] Marco Butto "El gran libro de Python", Marcombo 2016.

[2] P. Directorate, F. Street, and E. Street, "Finite Element *Model Updating* Using FRF Measurement," vol. 252, pp. 717–727, 2002.

[3] J. E. Mottershead and M. I. Friswell, "*Model Updating* In Structural Dynamics: A Survey," *Journal of Sound and Vibration*, vol. 167, no. 2. pp. 347–375, 1993.

[4] L. M. Operacional, "5 Análisis modal," pp. 51–62, 2000.

[5] Á. Magdaleno, "Estudio de nuevos indicadores en el dominio de la frecuencia y del tiempo para la sintonización óptima de TMDs múltiples en estructuras esbeltas," Universidad de Valladolid, 2017.

[6] J. M. M. Silva and N. M. M. Maia, "Modal Analysis and Testing." p. 612, 1999.

[7] Marta Herráez, "Análisis modal experimental" tema 6. Universidad de Valladolid

[8] Julian Fernández Ferrer, "Iniciación a la Física (2ª edición)" pp.72-73,1987.

[9] Leonard Meirovitch "Elements of vibration analysis" McGraw-Hill,1986, pp.287-295

[10] Inman, D.J. "Engineering vibration" Prentice Hall International, 1994.

[11] D.J. Ewins, "Modal testing: theory practice and application (2ª edición)" pp. 25-64.

[12] Maia Silva, "Theoretical and experimental modal analysis Fundamentals of modal analysis." pp. 54-63, 1997.

[13] M. Cacho-Pérez, N. Frechilla y A. Lorenzana. "Estimación de las masas modales de una estructura en servicio mediante transformación del espacio de estados" 2017

[14] http://help.solidworks.com/2016/spanish/SolidWorks/cworks/c_rayleigh_damping.htm

[15] D.J. Ewins, "Modal testing: theory practice and application (2ª edición)" pp. 62-66.

10.Anexo

A continuación se adjunta el código de programación implementado, debidamente comentado para facilitar la comprensión del usuario. El orden es el siguiente:

- TFM_Viga
- TFM_Modelo
- TFM_Impacto
- Imprimir_Excel
- TFM_Aplicación1

```
"""TFM_Viga (Main)"""
#-----------------------------------------------------------------------------------------------------------------------

"""
Created on Wed Mar 28 12:57:48 2018

@author: Coque
"""
import sys
from numpy import *


#Definición de las diferentes librerías y funciones que voy a usar-------------

import matplotlib.pyplot as plt
from scipy.integrate import odeint
import scipy.interpolate as interpolate
from TFM_Modelo import Bernouilli
from TFM_Impacto import Imp
from Imprimir_excel import Excel_Vel_Acc,Excel_FRF

# Modelización de la Viga-------------------------------------------------------

Nb=4            #Número de barras - No se debería poder cambiar a nivel usuario
Nelem=3         #Número de elementos en los que dividimos cada barra
n=Nb*Nelem      #Número total de elementos
a=0.04          #Canto de la viga en metros
b=0.10          #Ancho de la viga (donde se apoya) en metros
e=0.0015        #Espesor de la barra en metros (Común para todos los elementos)
L=6             #Longitud total de la barra en metros


rho=2700            #Densidad de cada barra, en kg/m3 (aluminio)
lp=L/Nb             #Longitud de una barra
E=7.0e10            #Módulo de Young de cada barra, en Pa


# Input Impacto-----------------------------------------------------------------

F0=19.45                            #Fuerza aplicada en Newtons
```

```
t0=15                               #Tiempo de análisis de la respuesta en segundos
T=0.0078125                         #Duración del impacto en segundos
intervalos=10000                    #Número de intervalos en el análisis del impacto
t=linspace(0,t0,intervalos)         #Vector temporal
h=t0/(intervalos-1)                 #Intervalo entre cada instante de tiempo en segundos

#Los nodos van desde 1 hasta n, pero los nodos 1 y n están apoyados, por lo que no tienen desplazamiento
#y el impacto no produce deformación.
p_a=7                               #Nudo que analizamos, entre 2 y n-1.
p_i=7                               #Posición del nudo donde aplico la fuerza, entre 2 y n-1

#Aviso en caso de seleccionar algún nodo fuera del rango establecido
if(2<=p_i<=n+1 and 2<=p_a<=n+1)==False:
    print('\n ! Valor de "p" superior al número de puntos...\n')
    sys.exit()

#Aviso en caso de seleccionar alguno de lo snodos extremos (1 ò n+1)
elif (p_i==1 or p_i==1+n or p_a==1 or p_a==1+n):
    print('\n El modelo resuelto es una viga biapoyada. El desplazamiento en los nodos 1 y {} es nulo.\n'.format(n+1))
    sys.exit()

#Definición de cada una de las barras ----------------------------------------
#Mas adelante se pueden poner valores concretos para cada barra (model updating)
#Forma del vector: [Nelem,a,b,lp,E,rho]

Barras=([Nelem,1.01*a,b,lp,E,1.08*rho],
        [Nelem,a,b,lp,0.98*E,rho],
        [Nelem,a,b,lp,0.98*E,rho],
        [Nelem,1.01*a,b,lp,E,1.08*rho])

#Matriz de masas y rigidez de Bernouilli--------------------------------------

M = zeros((2*(n+1),2*(n+1)))
K = zeros((2*(n+1),2*(n+1)))

#El bucle llama a una función que genera la matriz de masa y rigidez de ese
#elemento, y la coloca en su posición.
z=0 #Inicializo la variable z
for k in range(Nb):
    Ks,Ms = Bernouilli(Barras[k],e)
```

```
    index=k*(2*Nelem)

    for z in range(Nelem):

        K[index+2*z:index+2*z+4,index+2*z:index+2*z+4]+=Ks
        M[index+2*z:index+2*z+4,index+2*z:index+2*z+4]+=Ms

#Añado las masas puntuales, para la correlación con el ensayo en el laboratorio
for i in [0,-2,12,12]: #Centro y extremos
    M[i,i] += 0.0577

for i in [6,18]: #Acelerómetros para el modo 2
    M[i,i] += 0.0577

for i in [4,20]: #Acelerómetros para el modo 3
    M[i,i] += 0.0577

#Condiciones de contorno-----------------------------------------------------
#Movimiento vertical y horizontal restringido en ambos extremos (Biapoyada):

for i in [0,-2]:
    M=delete(M,i,1) #Elimina la columna 0 y -2 de la matriz de masas
    M=delete(M,i,0) #Elimina la columna 0 y -2 de la matriz de rigidez
    K=delete(K,i,1) #Elimina la fila 0 y -2 de la matriz de masas
    K=delete(K,i,0) #Elimina la fila 0 y -2 de la matriz de masas

#Hallar frecuencias y modos propios-------------------------------------------

invM = linalg.inv(M)            #Matriz inversa de la matriz de masas
w2,V = linalg.eig(invM@K)       #Resolvemos el sistema, se obtienen frecuencias propias al cuadrado, y los modos propios

#Ordenamos las frecuencias y vectores propios de menor a mayor-----------------

idx = w2.argsort()[0::]
w2 = w2[idx]
V = V[:,idx]                    #Las columnas son los modos propios de cada frecuencia ordenados.

w2[i]=w2[i].real                #Nos quedamos solo con la parte real de las frecuencias
                                #los valores imaginarios son 0, se eliminan para que no den problemas
```

```
frecuencia = list(1/(2*pi)*sqrt(w2)) #Frecuencias propias en Hz

#Impresión de las tres primeras frecuencias propias----------------------------

if len(frecuencia) >= 3:
    message='f1 = {:.2f} Hz   f2 = {:.2f} Hz   f3 = {:.2f} Hz'.format(frecuencia[0],frecuencia[1],frecuencia[2])
    print (message)
else:
    message='f1 = {:.2f} Hz  f2 = {:.2f} Hz'.format(frecuencia[0],frecuencia[1])
    print (message)

#Representación de los primeros modos de vibración-----------------------------

m1=array([0.]*(n+1))        #Contiene la amplitud del modo 1, para cada nodo de la viga
m2=array([0.]*(n+1))        #Contiene la amplitud del modo 2, para cada nodo de la viga
m3=array([0.]*(n+1))        #Contiene la amplitud del modo 3, para cada nodo de la viga
m4=array([0.]*(n+1))        #Contiene la amplitud del modo 4, para cada nodo de la viga

x=linspace(1,n+1,n+1)       #Posiciones en la viga, siendo los nodos 1 y n+1 los nodos apoyados

for i in range (1,n):       #Completa los vectores con la amplitud de los cuatro primeros
    m1[i]=V[i*2-1,0]        #modos, en todos los puntos de la viga.
    m2[i]=V[i*2-1,1]
    m3[i]=V[i*2-1,2]
    m4[i]=V[i*2-1,3]

t_1, c_1, k_1 = interpolate.splrep(x, m1, s=0, k=4) #splrep devuelve los nodos, los coeficientes
t_2, c_2, k_2 = interpolate.splrep(x, m2, s=0, k=4) #para calcular el B-Spline y el grado del spline.
t_3, c_3, k_3 = interpolate.splrep(x, m3, s=0, k=4)
t_4, c_4, k_4 = interpolate.splrep(x, m4, s=0, k=4)

xnew =linspace(min(x), max(x), 4*len(x)) #Puntos en los cuales se interpola el spline.

spline1 = interpolate.BSpline(t_1, c_1, k_1, extrapolate=False) #Crea una función que calcula el spline para
spline2 = interpolate.BSpline(t_2, c_2, k_2, extrapolate=False) #el conjunto de puntos xnew
spline3 = interpolate.BSpline(t_3, c_3, k_3, extrapolate=False)
spline4 = interpolate.BSpline(t_4, c_4, k_4, extrapolate=False)


plt.figure(1)   #Plot de los modos de vibración.
```

```
plt.plot(xnew,spline1(xnew),label='MODO 1')
plt.plot(xnew,spline2(xnew),label='MODO 2')
plt.plot(xnew,spline3(xnew),label='MODO 3')
plt.plot(xnew,spline4(xnew),label='MODO 4')
plt.title('Representación de los modos de vibración ')
plt.legend(loc='best')
plt.xlabel('Posición en la viga')
plt.ylabel('Amplitud')


#Matriz de masa y de rigidez diagonalizadas-----------------------------------

Mdiag=V.T @ M @ V     #Diagonalización de la matriz de masa, multiplicando por la matriz modal
D=[None]*len(Mdiag)   #Contiene los módulos para reducir la matriz modal a la masa.

#Hallar las matrices reducidas a la masa--------------------------------------

for i in range(len(Mdiag)):
    D[i]=sqrt(Mdiag[i,i])   #Divido cada modo propio por la raíz del valor en la diagonal,
    V[:,i]=V[:,i]/D[i]      #para que la matriz de masas diagonalizada sea la matriz identidad.

Mdiag=V.T @ M @ V           #Resultando la matriz identidad
Kdiag=V.T @ K @ V           #En la diagonal contiene los valores de w2

#Matriz de amortiguamiento----------------------------------------------------

a1=0.52039956                          #Alfa, Valores calculados para obtener las amortiguaciones teóricas
b1=8.108347E-06                        #Beta
C=a1*M+b1*K                            #Definición física de la matriz de amortiguamiento
Cdiag=V.transpose() @ C @ V            #Diagonalización de la matriz de amortiguamiento.
xi=diagonal(Cdiag)/(2*sqrt(w2))        #Factor de amortiguamiento de cada modo

#Resolución numérica con Odeint()---------------------------------------------

p_a=2*p_a-3                            #Elijo la fila cuyos valores en la matriz corresponden
p_i=2*p_i-3                            #al desplazamiento vertical del nudo elegido.

f_matriz=zeros((len(w2),len(t)))
f_matriz[p_i,:]=Imp(t,T,F0)            #Función impacto en el punto "p_i", durante el tiempo de ensayo t
```

```
f_cte=[0]*len(w2)                       #Vector impacto.
f_cte[p_i]=F0                           #Se agrega el valor pico. Constante que multiplica el seno.
f_cte_r= V.T @ f_cte                    #Premultiplicación por la matriz de paso.


def Vibracion(y,t,fr,T,cr,w2r):                 #Función vibración para operar con odeint
    q,x=y                                       #x es una variable auxiliar que baja un grado la ecuación diferencial
    if t > T:                                   #Definición de la función impulso, tras el golpeo (T)
        t=0                                     #no hay más excitación.
    dydt=[x,fr*sin(pi*t/T)-cr*x-w2r*q]          #Definición de la función
    return dydt                                 #Devuelve la derivada de la variable y

y0=[0.0,0.0]                        #Condiciones iniciales en desplazamiento y velocidad
r=zeros((len(w2),len(t)))           #Contiene los valores de la variable modal
r_vel=zeros((len(w2),len(t)))       #Contiene los valores de la derivada de primer orden de la variable modal

for i in range(len(w2)):            #Resolución del desplazamiento de todos los puntos. Realiza la operación
    cr=Cdiag[i,i]                   #para cada nodo e instante de tiempo
    w2r=w2[i]
    fr=f_cte_r[i]

    sol=odeint(Vibracion,y0,t,args=(fr,T,cr, w2r))
    r[i,:]=sol[:,0]                 #La primera columna del vector "sol" corresponde a la posición del nodo.
    r_vel[i,:]=sol[:,1]             #La segunda columna del vector "sol" corresponde a la velocidad del nodo.


#Volver a las variables físicas de desplazamientos-----------------------------
q=V@r                                       #Desplazamiento de cada nodo.
velocidad=V@r_vel                           #Velocidad de cada nodo.
aceleracion=invM@(f_matriz-C@velocidad-K@q)  #Aceleración de cada nodo. Solución de la EDO.

"""Excel_Vel_Acc(t,q,aceleracion,p_a)"""        #Crea un excel con los datos del desplazamiento y aceleración del nodo

#Plot del desplazamiento del punto de análisis---------------------------------

plt.figure(2)
plt.plot(t,q[p_a,:]*1000,label='Solución en desplazamientos')
plt.title('Desplazamiento del punto {}'.format(int((p_a+3)/2)))
plt.legend(loc='best')
plt.xlabel('Tiempo (s)')
```

```
plt.ylabel('Desplazamiento (mm)')
plt.xlim(xmax=t0)

#Plot de la aceleración del punto de análisis----------------------------------

plt.figure(3)
plt.plot(t,aceleracion[p_a,:],label='Solución aceleración')
plt.title('Aceleracion del punto {}'.format(int((p_a+3)/2)))
plt.legend(loc='best')
plt.xlabel('Tiempo (s)')
plt.ylabel('Aceleracion (m/s2)')
plt.xlim(xmax=t0)

#Cálculo de la respuesta en frecuencias de la estructura----------------------

Rec=zeros((len(t)),dtype=complex)           #Contiene la receptancia del nodo p_a, producida por una excitación
                                            #en el nodo p_i para un cierto rango de frecuencias
w_vector=linspace(0,sqrt(w2[5]),len(t))     #Define el rango de frecuencias en el que vamos a calcular la FRF

j=complex(0,1)                              #Defino la variable compleja

#Cálculo de la receptancia de cada nodo, para el rango de frecuencias w_vector.
for i in range(len(t)):
    for k in range(len(w2)):
        Rec[i]+=V[p_a,k]*V[p_i,k]/(-w_vector[i]**2+w2[k]+2*sqrt(w2[k])*xi[k]*w_vector[i]*j)

Accel=w_vector**2*Rec                       #Cálculo de la acelerancia de cada nodo a partir de la receptancia.
f_vector=w_vector/(2*pi)                    #Conversión de la frecuencia de rad/s a Hz.

"""Excel_FRF(f_vector,Accel,p_a,p_i)"""     #Crea un excel con la FRF del nodo especificado

plt.figure(4) #Plot del módulo de la respuesta en frecuencia del nodo

plt.plot(f_vector[1:-5],abs(Accel[1:-5]),'b-')
plt.yscale('log')
plt.title('Respuesta en frecuencia del nudo {}\n con excitación armónica en el nudo {}'.format(int((p_a+3)/2),format(int((p_i+3)/2))))
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Amplitud (m/N)')
plt.legend(['Acelerancia - Módulo'],loc=0)
```

```
plt.figure(5) #Plot de la fase de la respuesta en frecuencia del nodo

plt.plot(f_vector[1:-5],angle(Accel[1:-5],deg=True),'b-')
plt.title('Respuesta en frecuencia del nudo {}\n con excitación armónica en el nudo {}'.format(int((p_a+3)/2),format(int((p_i+3)/2))))
plt.xlabel('Frecuencia (Hz)')
plt.ylabel('Amplitud (m/N)')
plt.legend(['Acelerancia - Fase'],loc=0)
```

```
"""TFM_Modelo"""
#-----------------------------------------------------------------------------------------------------------------------

"""
Created on Wed Mar 28 20:23:05 2018

@author: PascualA
"""

from numpy import array

#Define las matrices de masa y elástica de cada elemento.
def Bernouilli(Barras,e):
     Nelem=Barras[0]                              #Número de elementos en los que dividimos cada barra
     a=Barras[1]                                  #Canto de la barra en m
     b=Barras[2]                                  #Anchura de la barra en m
     lp=Barras[3]                                 #longitud de la barra en m
     E=Barras[4]                                  #Módulo de Young de la barra en Pa
     rho=Barras[5]                                #Densidad de la barra en Kg/m3

     l=lp/Nelem                                   #longitud de un elemento en m
     I=b*a**3/12-(b-2*e)*(a-2*e)**3/12            #Inercia de cada barra en m4
     m=rho*l*(b*a-(a-2*e)*(b-2*e))                #Masa de cada elemento en kg

     #Definición de la matriz de rigidez de cada elemento

     Ks = array([[12,6*l,-12,6*l],[6*l,4*l**2,-6*l,2*l**2],
              [-12,-6*l,12,-6*l],[6*l,2*l**2,-6*l,4*l**2]])*E*I/l**3

     #Definición de la matriz de masa de cada elemento

     Ms = array([[156,22*l,54,-13*l],[22*l,4*l**2,13*l,-3*l**2],
               [54,13*l,156,-22*l],[-13*l,-3*l**2,-22*l,4*l**2]])*m/420

     return Ks,Ms

 #La función devuelve dos matrices:
 #Ks, matriz de rigidez, 4*4 de cada elemento
 #Ms, matriz de masa, 4*4 de cada elemento
```

```
"""Función impacto"""
#-------------------------------------------------------------------------------------------------------------------

"""
Created on Sat May  5 19:42:11 2018

@author: PascualA
"""
import sys
from numpy import *


def Imp(t,T,F0):
    F=[None]*len(t)                     #Vector que contiene la fuerza del impacto para cada instante de tiempo
    for i in range(len(t)):             #Bucle para el intervalo de tiempo "t"
        if t[i] < T:                    #Si el instante está dentro del intervalo de tiempo de impacto "T"
            F[i] = F0*sin(pi*t[i]/T)    #la respuesta de la función es distinta de 0 y de forma sinusoidal
        else:
            F[i] = 0                    #De lo contrario el valor es 0.

    return F                            #La función devuelve el vector con los valores de la fuerza para cada instante.
```

```
"""Imprimir_excel"""
#-----------------------------------------------------------------------------------------------------------------------

"""
Created on Wed Jul 11 16:18:15 2018

@author: PascualA
"""
#Definición de las diferentes librerías y funciones que voy a usar-------------
import sys
from numpy import *
import matplotlib.pyplot as plt
from openpyxl import Workbook


def Excel_Vel_Acc(t,q,aceleracion,p_a):

    wb = Workbook()
    ruta = 'Aceleracion.xlsx'

    hoja = wb.active
    hoja.title = "Resultados nodo {}".format(int((p_a+1)/2))

    fila = 1                            #Fila donde empezamos
    col_tiempo = 1                      #Columna donde guardamos los tiempos
    col_posicion = 2                    #Columna donde guardamos el desplazamiento
    col_aceleracion = 3                 #Columna donde guardamos la aceleración

    hoja.cell(column=1, row=1, value='Tiempo (s)')
    hoja.cell(column=2, row=1, value='Desplazamiento (mm)')
    hoja.cell(column=3, row=1, value='Aceleración (m/s2)')

    for i in range(len(t)):
        fila=fila+1
        hoja.cell(column=col_tiempo, row=fila, value=t[i])
        hoja.cell(column=col_posicion, row=fila, value=q[p_a,i]*1000)
        hoja.cell(column=col_aceleracion, row=fila, value=aceleracion[p_a,i])

    wb.save(filename = ruta)
```

```
    return

def Excel_FRF(f,acelerancia,p_a,p_i):
    wb = Workbook()
    ruta = 'FRF_i{}_a{}.xlsx'.format(int((p_i+3)/2),int((p_a+3)/2))

    hoja = wb.active
    hoja.title = "Resultados nodo {}".format(int((p_a+3)/2))

    fila = 3                                #Fila donde empezamos
    col_frecuencia = 1                      #Columna donde guardamos las frecuencias (Hz)
    col_acelerancia = 2                     #Columna donde guardamos la acelerancia
    col_acelerancia_fase = 3                #Columna con la fase de la acelerancia

    hoja.cell(column=1, row=1, value='P.análisis: {}'.format(int((p_a+3)/2)))
    hoja.cell(column=2, row=1, value='P.impacto: {}'.format(int((p_i+3)/2)))
    hoja.cell(column=col_frecuencia, row=2, value='Frecuencia (Hz)')
    hoja.cell(column=col_acelerancia, row=2, value='Acelerancia (m/s2)')
    hoja.cell(column=col_acelerancia_fase, row=2, value='Fase Frecuencia (Grados)')

    Mod_acelerancia=abs(acelerancia)
    Angle_acelerancia=angle(acelerancia,deg=True)

    for i in range(len(f)):
        hoja.cell(column=col_frecuencia, row=fila, value=f[i])
        hoja.cell(column=col_acelerancia, row=fila, value=Mod_acelerancia[i])
        hoja.cell(column=col_acelerancia_fase, row=fila, value=Angle_acelerancia[i])
        fila=fila+1

    wb.save(filename = ruta)

    return
```

```
"""TFM_Aplicación1"""
#-----------------------------------------------------------------------------------------------------------------------

"""
Created on Wed Mar 28 12:57:48 2018

@author: Coque
"""
import sys
from numpy import *

"""Para comprobar que la simulación es correcta, suponemos el caso en el que
añadimos a la viga una tuerca de 0.01 kg cada 30mm.
Es decir un total de 2kg distribuidos uniformemente en la viga.Para simular
este supuesto aumentaremos la densidad de la viga del modelo.Posteriormente se
comparan los modos resultantes del modelo con los modos teóricos. """

#Definición de las diferentes librerías y funciones que voy a usar-------------

import matplotlib.pyplot as plt
from scipy.integrate import odeint
import scipy.interpolate as interpolate
from TFM_Modelo import Bernouilli
from TFM_Impacto import Imp
from Imprimir_excel import Excel_Vel_Acc,Excel_FRF

# Modelización de la Viga-----------------------------------------------------

Nb=4            #Número de barras - No se debería poder cambiar a nivel usuario
Nelem=3         #Número de elementos en los que dividimos cada barra
n=Nb*Nelem      #Número total de elementos
a=0.04          #Canto de la viga en metros
b=0.10          #Ancho de la viga (donde se apoya) en metros
e=0.0015        #Espesor de la barra en metros (Común para todos los elementos)
L=6             #Longitud total de la barra en metros


rho=2700            #Densidad de cada barra, en kg/m3 (aluminio)
lp=L/Nb             #Longitud de una barra
E=7.0e10            #Módulo de Young de cada barra, en Pa
```

```
#Edit del modelo, con los tornillos repartidos uniformemente

A=b*a-(a-2*e)*(b-2*e)                #Área transversal de la viga en m^2
Vol=L*A                              #Volumen total de la viga en m^3
Masa=rho*Vol                         #Masa total de la viga en kg
Masa_prima=Masa+2+0.0577*8           #Nueva masa de la viga añadiendo los 2kg
                                     #uniformemente repartidos, y la masa
                                     #correspondiente a los acelerómetros

rho_prima=Masa_prima/Vol             #Densidad aplicada a la solución analítica.

rho=(rho*Vol+2)/Vol                  #Densidad aplicada al modelo de python.
                                     #Posteriormente se añade la masa de
                                     #los acelerómentros en las matrices de cada elemento.

#Definición de cada una de las barras ----------------------------------------
#Model updating donde se pueden poner valores concretos para cada barra
#Forma del vector: [Nelem,a,b,lp,E,rho]

Barras=([Nelem,1.01*a,b,lp,E,1.08*rho],
        [Nelem,a,b,lp,0.98*E,rho],
        [Nelem,a,b,lp,0.98*E,rho],
        [Nelem,1.01*a,b,lp,E,1.08*rho])

#Matriz de masas y rigidez de Bernouilli--------------------------------------

M = zeros((2*(n+1),2*(n+1)))
K = zeros((2*(n+1),2*(n+1)))

#El bucle llama a una función que genera la matriz de masa y rigidez de ese
#elemento, y la coloca en su posición.
z=0 #Inicializo la variable z
for k in range(Nb):
    Ks,Ms = Bernouilli(Barras[k],e)
    index=k*(2*Nelem)

    for z in range(Nelem):

        K[index+2*z:index+2*z+4,index+2*z:index+2*z+4]+=Ks
```

```
        M[index+2*z:index+2*z+4,index+2*z:index+2*z+4]+=Ms

#Añado las masas puntuales, para la correlación con el ensayo en el laboratorio

for i in [0,-2,12,12]: #Centro y extremos
    M[i,i] += 0.0577

for i in [6,18]: #Acelerómetros para el modo 2
    M[i,i] += 0.0577

for i in [4,20]: #Acelerómetros para el modo 3
    M[i,i] += 0.0577


#Condiciones de contorno-----------------------------------------------------
#Movimiento vertical y horizontal restringido en ambos extremos (Biapoyada):

for i in [0,-2]:
    M=delete(M,i,1) #Elimina la columna 0 y -2 de la matriz de masas
    M=delete(M,i,0) #Elimina la columna 0 y -2 de la matriz de rigidez
    K=delete(K,i,1) #Elimina la fila 0 y -2 de la matriz de masas
    K=delete(K,i,0) #Elimina la fila 0 y -2 de la matriz de masas

#Hallar frecuencias y modos propios-------------------------------------------

invM = linalg.inv(M)          #Matriz inversa de la matriz de masas
w2,V = linalg.eig(invM@K)     #Resolvemos el sistema, obteniendo las frecuencias propias
                              #al cuadrado, y los modos propios

#Ordenamos las frecuencias y vectores propios de menor a mayor----------------

idx = w2.argsort()[0::]
w2 = w2[idx]
V = V[:,idx]                          #Las columnas son los modos propios de cada
                                      #frecuencia ordenados.

w2[i]=w2[i].real                      #Nos quedamos solo con la parte real de las frecuencias
                                      #los valores imaginarios son 0.

frecuencia = list(1/(2*pi)*sqrt(w2)) #Frecuencias propias en Hz
```

```
#Impresión de las tres primeras frecuencias propias----------------------------

message='Solucion simulada:\nf1 = {:.2f} Hz   f2 = {:.2f} Hz   f3 = {:.2f} Hz\n'.format(frecuencia[0],frecuencia[1],frecuencia[2])
print (message)

#Solución teórica -> Tabla 11-12.1-----------------------------------------------

frecuencia_t=[0]*5                              #Lista que contiene la frecuencia propia para los tres primeros modos propios.
I=b*a**3/12-(b-2*e)*(a-2*e)**3/12               #Inercia de la viga

#Calculamos la frecuencia natural de cada modo mediante la fórmula de la tabla 11-12.1
for i in [1,2,3,4,5]:
    frecuencia_t[i-1]=(i*pi)**2/(2*pi*L**2)*(E*I/(A*rho_prima))**(1/2)

#Impresión de las tres primeras frecuencias propias teóricas-------------------

message='Solucion teorica:\nf1 = {:.2f} Hz   f2 = {:.2f} Hz   f3 = {:.2f} Hz\n'.format(frecuencia_t[0],frecuencia_t[1],frecuencia_t[2])
print (message)
```