UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

METODOLOGÍA PARA LA OPTIMIZACIÓN DEL NÚMERO Y DISTRIBUCIÓN DE SENSORES PARA EL MONITOREO DE UNA VIGA UTILIZANDO ALGORITMOS GENÉTICOS

MEMORIA PARA OPTAR AL TITULO DE INGENIERO CIVIL MECANICO

ALAN OSVALDO ALFARO ARAYA

PROFESORA GUÍA:
VIVIANA MERUANE NARANJO

MIEMBROS DE LA COMISIÓN:
JUAN CRISTÓBAL ZAGAL MONTEALEGRE
ALEJANDRO ORTIZ BERNARDÍN

Este trabajo ha sido parcialmente financiado por proyecto Fondecyt

SANTIAGO DE CHILE
NOVIEMBRE 2012

Resumen ejecutivo

La presencia de una grieta en una estructura, no solo varía las propiedades mecánicas del material, sino que también influye sobre sus características dinámicas. Es por esa razón que el monitoreo de las vibraciones de una estructura es una técnica muy utilizada en ingeniería y permite la detección temprana del daño.

Generalmente, la decisión del posicionamiento de sensores de monitoreo, pasa por el juicio y la experiencia del ingeniero a cargo; sin embargo, un error puede provocar que el daño no se detecte.

Se han desarrollado variados métodos de optimización de posición de sensores que monitorean el comportamiento dinámico de la estructura. Sin embargo, hasta hoy no existe un procedimiento estándar que optimice la posición y el número de sensores a utilizar, enfocado en la identificación del daño.

Se desea desarrollar una metodología para optimizar la posición y número de sensores de monitoreo dinámico en una estructura tipo viga, utilizando algoritmos genéticos paralelos. En particular, se desea encontrar la configuración óptima de sensores cuando se monitorean las frecuencias de anti-resonancia de la estructura.

El algoritmo genético es un método de optimización basado en la teoría de la evolución de Darwin, el que ha cobrado una alta popularidad en todo el mundo durante los últimos años, debido a su robustez y a su independencia de la función objetivo. Es un algoritmo matemático que transforma un conjunto de objetos matemáticos usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto. Dentro de estas operaciones genéticas destacan la mutación y la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija denominado cromosoma. Cada gen dentro del cromosoma representa una variable a optimizar. La aptitud de cada cromosoma se mide evaluando en la función objetivo.

Para desarrollar esta metodología, primeramente se encontraron las frecuencias de anti-resonancia de la estructura y luego se determinó su sensibilidad al daño. Seguidamente, se define una función objetivo que relacione la información al daño y la ubicación de los sensores. Esta función es optimizada a través de la programación de algoritmos genéticos paralelos. Finalmente, se realiza una verificación experimental de la metodología creada, utilizando un algoritmo de detección de daño disponible y datos experimentales.

Para la programación en elementos finitos y algoritmos genéticos, se utiliza el software Matlab y su extensión GAOT. Para la verificación experimental, se trabajará en el laboratorio de sólidos Mecesup ubicado en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

Este trabajo es parte del proyecto Fondecyt de iniciación Nº 11110446 desarrollado por la Dra. en Ingeniería Viviana Meruane, Profesora Guía de este Trabajo de Título.

Agradecimientos

Cuantas palabras hacen falta para agradecerlo todo...
pues faltan las ya inventadas y las que no existen
las que carecen de sentido y las escritas al revés
un lenguaje extraterrestre incluso para agradecerlo todo.

Como agradecer la guía y las alegrías,
los abrazos y las sonrisas, los hombros y la paciencia
las mejores amistades, las resacas más terribles
como agradecerlo todo cuando faltan las palabras

Hacerlo con la vida, con la memoria no breve
el recuerdo infinito de todos aquellos
que formaron parte de este vaivén
que no son más que estos ocho años
llenos de todo, faltos de nada
un camino vermiforme alargado por el tiempo
atesorado por almas eternas para quienes no existen palabras
para agradecerlo todo

Para ellos mi vida y más que mi vida si pudiera
para cada uno, cada parte de mi
para agradecerlo todo o al menos parte de
para todos, todo
para nadie, nada

Indice

1. Introducción

1.1. Motivación

El monitoreo de las vibraciones de una estructura es una técnica ampliamente utilizada para la detección temprana de daño. El principio básico es que el daño (grieta) modifica las propiedades mecánicas de la estructura, lo que a su vez modifica sus características dinámicas. Mediante un monitoreo continuo de las vibraciones es posible detectar, localizar y cuantificar la presencia de daño estructural.

En general, la ubicación y número de sensores se definen en base al juicio y la experiencia de los ingenieros a cargo. Sin embargo, una distribución incorrecta o un número insuficiente de sensores pueden hacer que ciertos daños no sean detectables.

Este trabajo busca definir una metodología para optimizar la posición y el número de sensores, tomando de cuenta el uso de instrumentación limitada y su efecto en la predicción del daño estructural. En particular, se desea encontrar la configuración óptima de sensores cuando se monitorean las frecuencias de anti-resonancia de la estructura. La optimización se llevará a cabo por medio de Algoritmos Genéticos, estos son un método de búsqueda global basados en la teoría de la evolución de Darwin. Los resultados serán validados por medio de un algoritmo de detección de daño disponible utilizando datos simulados y experimentales.

La figura 1 ilustra un ejemplo de monitoreo de vibraciones de una estructura.

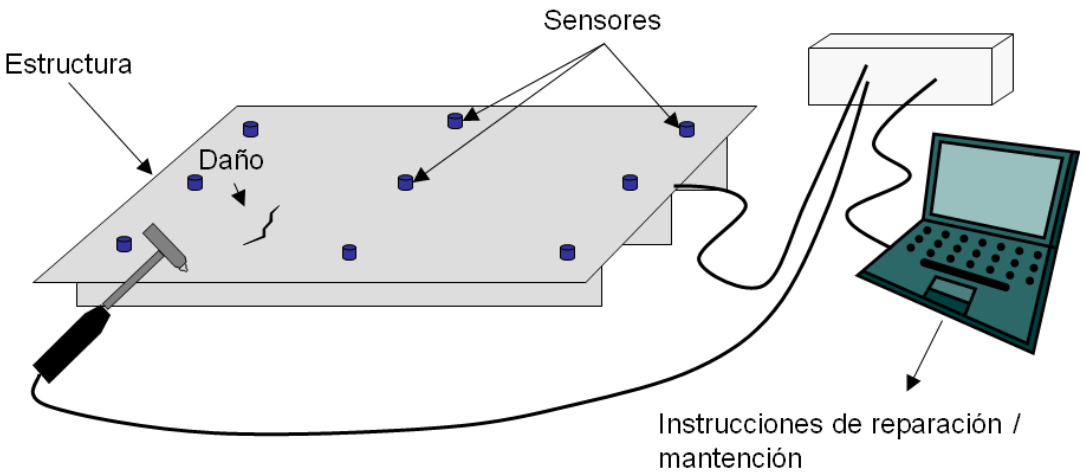

Figura 1: Ilustración del desarrollo del trabajo a realizar

1.2. Antecedentes Generales

El monitoreo de una estructura es una técnica ampliamente utilizada en Ingeniería para detectar tempranamente algún tipo de daño. El posicionamiento óptimo de los sensores de monitoreo es un asunto crítico en la creación e implementación de un sistema de monitoreo efectivo. Es por eso que existe un constante e intensivo desarrollo de innovadores sistemas de sensores. Sin embargo, aún existe una gran incertidumbre respecto al número de sensores y su posicionamiento para obtener una cantidad información adecuada que permitan describir el comportamiento estructural e identificar el daño de manera temprana.

Numerosas técnicas se han desarrollado para resolver el problema de la posición óptima de sensores, desde métodos intuitivos basados en el juicio y experiencia del ingeniero, hasta el desarrollo de métodos sistemáticos de optimización.

La caracterización del comportamiento dinámico de una estructura real es posible solo si existe una cantidad mínima de información disponible, lo que implica que una cantidad mínima de sensores de monitoreo deben ser posicionados en la estructura bajo una estricta evaluación. Ergo, los sensores deben ser cuidadosamente ubicados, con el objeto de obtener la información adecuada que permita identificar el comportamiento estructural.

Toda estructura posee frecuencias de resonancia, características globales de ella, es decir, no presentan variación independiente del punto de excitación y de medición, y además posee frecuencias de anti-resonancia las cuales son una característica local de la estructura, es decir, al contrario de las frecuencias resonantes, estas si varían al cambiar e punto de excitación y/o medición. Dado esto, las variaciones dinámicas que provoca algún tipo de daño se verán reflejadas en estas frecuencias, lo que permitirá detectar tal daño de manera temprana.

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar una metodología para ubicar y minimizar el número de sensores de monitoreo en una estructura tipo viga utilizando las frecuencias de anti-resonancia de la misma a través del método de algoritmos genéticos.

1.3.2. Objetivos específicos

- Encontrar las frecuencias de anti-resonancia en distintos puntos de una viga
- Determinar la sensibilidad de las anti-resonancias a daños en distintas posiciones de la viga
- Definir una función objetivo que relacione la información de daño, el número de sensores a utilizar y su ubicación.
- Implementar un método de optimización basado en algoritmos genéticos
- Validar los resultados con algoritmos de detección de daño disponibles y datos experimentales.

2. Antecedentes y Discusión bibliográfica

2.1. Vibraciones Mecánicas

El aumento permanente de las potencias en máquinas, junto con una disminución simultánea de gasto de materiales, y la alta exigencia de calidad y productividad industrial, hacen que el análisis dinámico de las vibraciones mecánicas en máquinas e instalaciones industriales sea cada vez más exacto. El fenómeno de las vibraciones mecánicas debe ser tenido en cuenta para el diseño, la producción y el empleo de maquinaria y equipos de automatización. Así lo exige un rápido desarrollo tecnológico que se ha hecho presente en todo el mundo.

Una vibración mecánica es el movimiento de una película o de un cuerpo que oscila alrededor de una posición de equilibrio. Al intervalo de tiempo necesario para que el sistema efectúe un ciclo completo de movimiento se le llama periodo de la vibración. El número de ciclos por unidad de tiempo define la frecuencia del movimiento y el desplazamiento máximo del sistema, desde su posición de equilibrio, se llama amplitud de la vibración.

Básicamente, las vibraciones se encuentran estrechamente relacionadas con tolerancias de mecanización, desajustes, movimientos relativos entre superficies, daños estructurales, etc. Estos fenómenos producen un desplazamiento del sistema desde su posición de equilibrio estable originando una vibración mecánica.

2.1.1. Modelo matemático, función de transferencia y FRF's

Consideremos un sistema de múltiples grados de libertad [1]. Para esto, se ilustra un ejemplo de dos grados de libertad en la Figura 2. La ecuación de movimiento de este sistema viene dada por:

$$m_1\ddot{x}_1 + (c_1 + c_2)\dot{x}_1 - c_2\dot{x}_2 + (k_1 + k_2)x_1 - k_2x_2 = f_1 \quad (2.1)$$

$$m_2\ddot{x}_2 + (c_2 + c_3)\dot{x}_2 - c_2\dot{x}_1 + (k_2 + k_3)x_2 - k_2x_1 = f_2 \quad (2.2)$$

En notación matricial:

$$M\{\ddot{x}\} + C\{\dot{x}\} + K\{x\} = \{f\} \quad (2.3)$$

Donde:

- M: Matriz de masa
- K: Matriz de rigidez
- C: Matriz de amortiguamiento
- {x}: Vector de respuesta
- $\{\dot{x}\}$: Vector velocidad
- $\{\ddot{x}\}$: Vector aceleración
- {f}: Vector de fuerzas

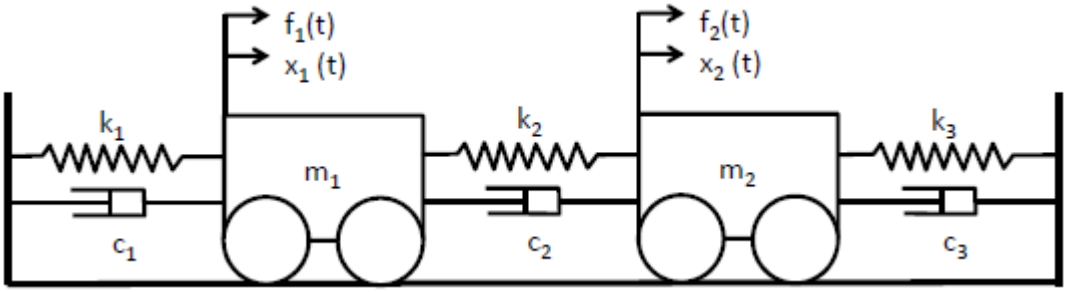

Figura 2: Ejemplo de un sistema de dos grados de libertad

Esta ecuación también describe el comportamiento de sistemas con más grados de libertad. La dimensión de las matrices aumenta con el número de grados de libertad. Transformando esta ecuación de movimiento en el dominio de Laplace, asumiendo que las velocidades y desplazamientos iniciales son cero, se obtiene:

$$(Mp^2 + Cp + K)X(p) = F(p) \tag{2.4}$$

Lo que se puede escribir como:

$$Z(p)X(p) = F(p) \tag{2.5}$$

$Z(p)$es la matriz de rigidez dinámica.

De la ecuación (2.6) se obtiene la función de transferencia $H(p)$:

$$H(p) = Z(p)^{-1} = \frac{X(p)}{F(p)} \tag{2.6}$$

En otras palabras, la función de transferencia es la relación entre la respuesta del sistema y la excitación aplicado a este.

Si evaluamos la ecuación (2.7) en el demonio de las frecuencias $s = j\omega$, se obtiene la función de respuesta en frecuencia (FRF):

$$H(\omega) = \frac{Respuesta\ (\omega)}{Excitacion\ (\omega)} \tag{2.7}$$

Las figuras 3, 4 y 5 muestran las tres FRF para el sistema de la figura 2. Los “peaks” en las curvas, son las frecuencias de resonancia, que tiene el mismo valor para todas las FRF, ya que son propiedades globales de la estructura. Por otro lado, las frecuencias donde las FRF tienen “caídas”, se denominan anti-resonancias, los cuales varían para cada FRF, ya que son propiedades locales de la estructura.

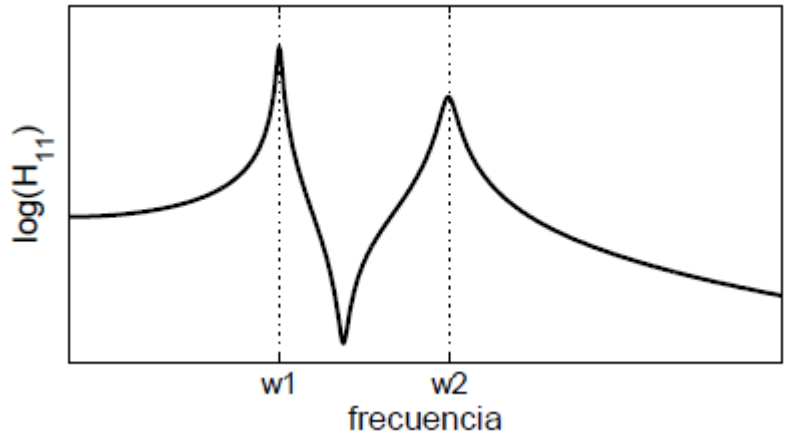

Figura 3: Funcion de respuesta en frecuencia, excitación en 1, respuesta en 1

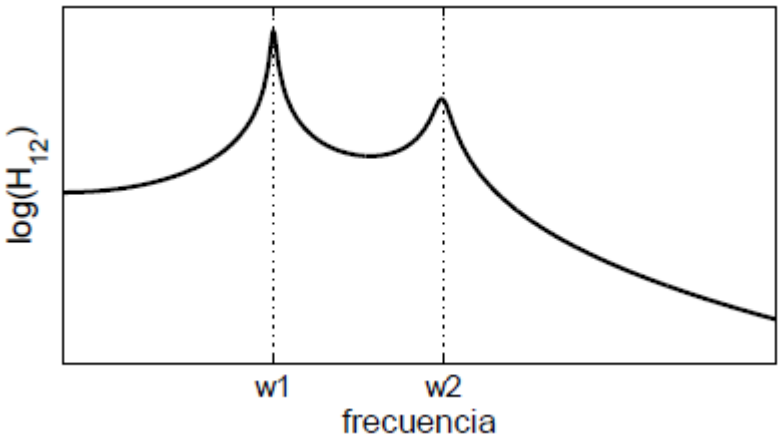

Figura 4: Funcion de respuesta en frecuencia, exitación en 1, respuesta en 2 o excitación en 2, respuesta en 1

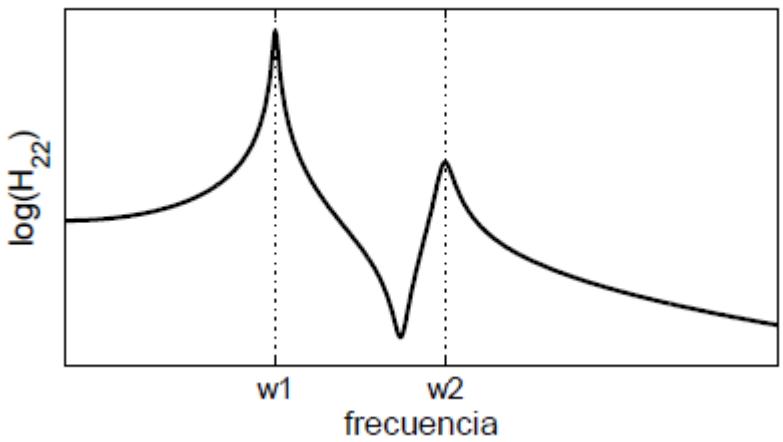

Figura 5: Funcion de respuesta en frecuencia, excitación en 2, respuesta en 2

2.1.2. Funciones de Respuesta en Frecuencia Experimentales

La ecuación 2.7, define la función de respuesta en frecuencia como una relación entre la señal de salida y la señal de entrada en el dominio de frecuencias. Si definiéramos F(ω) al espectro en frecuencia de la señal de entrada y X(ω) al espectro en frecuencia de la señal de salida entonces H(ω) quedaría como:

$$H(\omega) = \frac{X(\omega)}{F(\omega)} \tag{2.8}$$

Al evaluar la función de respuesta en frecuencia con la ecuación 2.8, se corre el riesgo que F(ω) sea cero para algunos términos. En la práctica existen formas alternativas de calcular H(ω), utilizando las potencias espectrales:

$$H_1(\omega) = \frac{X(\omega)}{F(\omega)}\frac{F^*(\omega)}{F^*(\omega)} = \frac{G_{XF}}{G_{FF}} \tag{2.9}$$

$$H_2(\omega) = \frac{X(\omega)}{F(\omega)}\frac{X^*(\omega)}{X^*(\omega)} = \frac{G_{XX}}{G_{FX}} \tag{2.10}$$

El principal motivo para estimar las FRF´s con las ecuaciones anteriores, es la disminución del ruido no correlacionado en las señales de entrada y salida al promediar.

En la práctica, la función de respuesta en frecuencia es estimada con valores promedio de las potencias espectrales,

$$\hat{G}_{FF} = \frac{1}{N_a}\sum_{n=1}^{N_a}(G_{FF})_n \tag{2.11}$$

$$\hat{G}_{XX} = \frac{1}{N_a}\sum_{n=1}^{N_a}(G_{XX})_n \tag{2.12}$$

$$\hat{G}_{FX} = \frac{1}{N_a}\sum_{n=1}^{N_a}(G_{FX})_n \tag{2.13}$$

$$\hat{G}_{XF} = \frac{1}{N_a}\sum_{n=1}^{N_a}(G_{XF})_n \tag{2.14}$$

Donde N_a es el número promedios que se repite el ensayo, lo que entrega una aproximación de mínimos cuadrados de H(ω).

Dado que las funciones de respuesta en frecuencia se obtienen de una aproximación de mínimos cuadrados, se puede definir un coeficiente de correlación. En este caso, la correlación se denomina **función de coherencia** y es una medida del error de mínimos cuadrados. Se define por:

$$\gamma^2 = \frac{|\hat{G}_{FX}|^2}{\hat{G}_{FF}\hat{G}_{XX}} = \frac{H_1(\omega)}{H_2(\omega)} \tag{2.15}$$

La coherencia varía entre 0 y 1. Un valor de 1, indica una relación perfectamente lineal entre las señales de entrada y salida por sobre todos los promedios. Una coherencia menor a uno, se puede deber a uno de los siguientes motivos:

- Ruido no correlacionado en las mediciones de f(t) y/o x(t)
- No-Linealidades del sistema en investigación
- Leakage en el análisis
- Desfase en las mediciones no compensado en el análisis.

2.2. Frecuencias de anti-resonancia

Los métodos de evaluación de daño tradicionales, usan la información modal obtenida de las frecuencias naturales y los modos normales de vibración. En general, las frecuencias naturales cuantifican y detectan la presencia de daño y los modos normales lo localizan. Sin embargo, los modos normales de vibración, son difíciles de identificar con exactitud. De hecho, solo se detectan con una precisión de a lo más un 10%, lo que puede provocar que el daño no se pueda localizar de manera exacta [2].

Debido a esto la función de respuesta en frecuencia (FRF), se ha convertido en una buena alternativa para el monitoreo de estructuras, en cuanto de ellas se puede extraer un muy buen detector de daño: las frecuencias de anti-resonancias.

Como se vio en la sección anterior, las frecuencias de anti-resonancias son las "caídas" que se observan en las curvas de las FRF y son una interesante alternativa para la detección de daños, pues se pueden detectar más fácilmente y con más precisión que los modos normales de vibración y aun así son capaces de proveer las misma cantidad de información. Además, son muy sensibles a pequeños cambios estructurales, lo que los convierte en muy buenos indicadores de daño.

Dilena y Morassi [3], estudiaron el problema de detección de daños en vigas usando frecuencias naturales y anti-resonancias y encontraron que el uso de estas últimas evitan el problema de encontrar más de una posible ubicación del daño, problema que aparecía cuando se usaban solo las frecuencias naturales. Sin embargo, también notaron que con el uso de las frecuencias de anti-resonancias, se amplificaba el ruido experimental y los errores del modelo.

A pesar de todas estas ventajas, el uso de las frecuencias de anti-resonancia, aún está en desarrollo y la aplicación de ella, en la detección de daño estructural, no ha sido completamente investigada.

2.2.1. Identificación experimental de las frecuencias de anti-resonancia

En la sección 2.1, se explica que es posible obtener las frecuencias características de un sistema de N grados de libertad, buscando los mínimos y máximos locales de las funciones de respuesta en frecuencia (FRF's) para detectar las frecuencias de anti-resonancia y de resonancia respectivamente. Cabe destacar, además, que estos máximos y mínimos están acompañados por un cambio abrupto en la fase de la FRF, en dirección de 180° para las frecuencias anti-resonantes y de -180° para las resonantes.

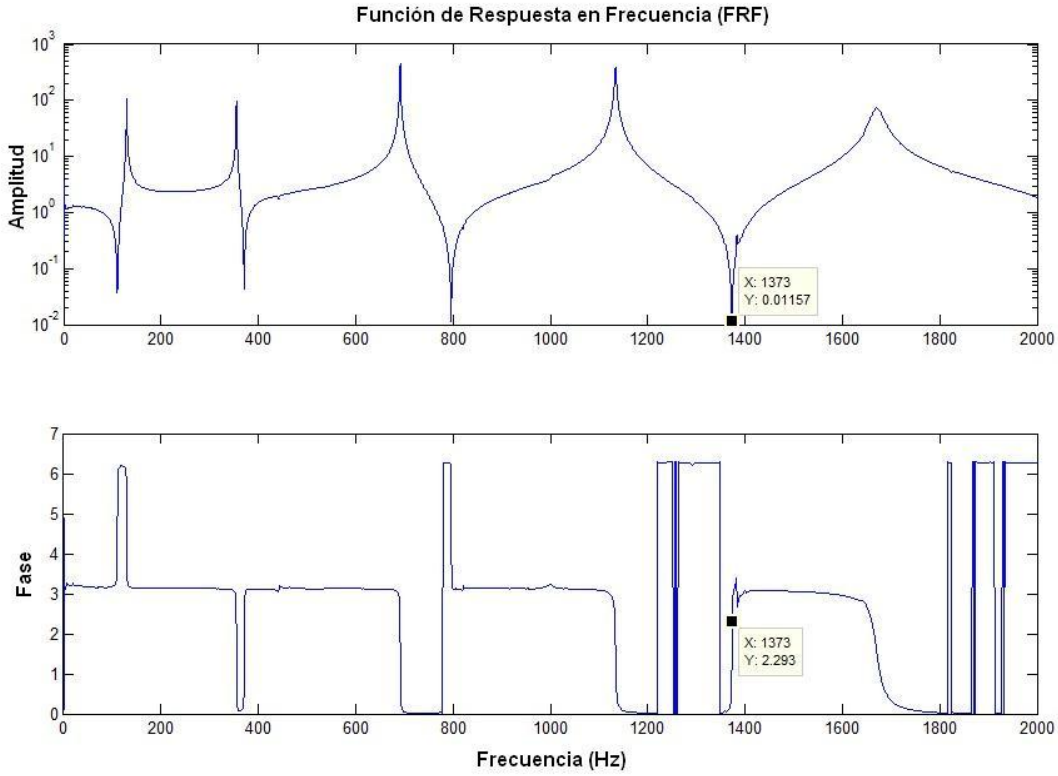

Figura 6: Función de Respuesta en Frecuencia (FRF) y fase correspondiente

La figura 6 muestra la detección de las frecuencias de anti-resonancia. Se puede observar que a los 1373 [Hz], no solo se encuentra un mínimo local, sino que también se produce un cambio de fase brusco. El eje vertical del gráfico de Fase, está en números reales, por lo que un cambio en 180° es equivalente a un cambio en π ó 3,14. Se observa que los 1373 [Hz] es una frecuencia de anti-resonancia, ya que no sólo es un mínimo local, sino que produce el cambio de fase correspondiente.

2.3. Método de elementos finitos

El método de los elementos finitos (MEF) [4], es un método numérico general para la aproximación de soluciones de ecuaciones diferenciales parciales muy utilizado en diversos problemas de ingeniería y física, el cual ha adquirido una gran importancia en la solución de problemas ingenieriles, físicos, etc., ya que permite resolver casos que hasta hace poco tiempo eran prácticamente imposibles de resolver por métodos matemáticos tradicionales. Esta circunstancia obligaba a realizar prototipos, ensayarlos e ir realizando mejoras de forma iterativa, lo que traía consigo un elevado coste tanto económico como en tiempo de desarrollo.

La idea general del método, es la división de un continuo en un conjunto de pequeños elementos interconectados por una serie de puntos llamados nodos. Las ecuaciones que rigen el comportamiento del continuo regirán también el del elemento. De esta forma se consigue pasar de un sistema continuo (infinitos grados de libertad), que es regido por una ecuación diferencial o un sistema de ecuaciones diferenciales, a un sistema con un número de grados de libertad finito cuyo comportamiento se modela

por un sistema de ecuaciones, lineales o no. Siendo un método aproximado, la precisión de los métodos de análisis de elementos finitos puede ser mejorada refinando la discretización en el modelo, usando más elementos y nodos.

Fundamentalmente, existen dos tipos de métodos de elementos finitos; el método de las fuerzas, donde se asumen las fuerzas y se calculan los desplazamientos y el método de los desplazamientos, donde se asumen los desplazamientos y se calculan las fuerzas. Este último es el más utilizado para análisis de vibraciones y será con el que se trabajará en esta memoria de título.

Una ventaja del MEF, frente al análisis experimental, y que será de gran ayuda para el desarrollo de este trabajo de título, es que en el primer caso, el análisis modal que se puede obtener es mucho más completo, ya que se pueden estimar los desplazamientos en cualquier punto de la estructura. Sin embargo, hay que tomar en consideración, que si bien en el MEF no existen errores y muchas veces las condiciones de borde son ideales, los resultados obtenidos por este medio no incluyen el amortiguamiento del sistema, es por esta razón que en la mayoría de los casos el modelo numérico debe ser mejorado iterativamente por medio de la comparación con el resultado del análisis experimental.

Para el caso de una viga, el MEF permite obtener la Matriz de masa, M, y la matriz de rigidez, K, a través del estudio de la energía cinética y potencial de la estructura, respectivamente [5]. Se tiene que:

$$M = \frac{\rho A l}{420}\begin{bmatrix} 156 & 22l & 54 & -13l \\ 22l & 4l^2 & 13l & -3l^2 \\ 54 & 13l & 156 & -22l \\ -13l & -3l^2 & -22l & 4l^2 \end{bmatrix} \tag{2.16}$$

$$K = \frac{EI}{l^3}\begin{bmatrix} 12 & 6l & -12 & 6l \\ 6l & 4l^2 & -6l & 2l^2 \\ -12 & -6l & 12 & -6l \\ 6l & 2l^2 & -6l & 4l^2 \end{bmatrix} \tag{2.17}$$

Donde,
ρ = densidad de la viga
l_e = largo del elemento finito
l = largo total de la viga
A = sección transversal
E = Módulo de Young
I = Momento de inercia

M y K son las matrices de masa y rigidez para cada elemento de viga, las cuales ensambladas definen el modelo dinámico de la barra y permiten obtener sus frecuencias naturales y modos de vibración como también las frecuencias de anti-resonancia, que son las que importan en este trabajo.

De la matriz $M^{-1}K$, eliminando la fila correspondiente al nodo donde se excita la viga y la columna del grado de libertad de la cual se quieren conocer las anti-resonancias, se obtienen los valores propios, ortogonales, los cuales contienen la información requerida.

2.3.1. Modelamiento del daño en elementos finitos

Para considerar analíticamente las influencias del daño en la estructura, generalmente causado por una grieta, se puede analizar como una reducción local de rigidez o mediante el análisis de un modelo en 2 o 3 dimensiones [6].

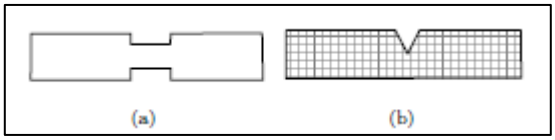

Figura 7: Esquema de a) reducción local de rigidez. b) Modelo de grieta.

La reducción de rigidez es la manera más fácil de modelar una estructura con daños, mediante la introducción del factor de reducción de rigidez (β). Este representa la proporción de reducción en comparación a la rigidez original, afectando la contribución de las rigideces locales en la matriz de rigidez total.

$$[K] = \sum(1 - \beta_i)[K_i] \tag{2.18}$$

Donde $[K_i]$ es la matriz local de rigidez y en un sistema de masa-resorte es la constante del i-esimo resorte. Por otro lado se pueden obtener mejores resultados al modelar las zona cercanas a la grieta (Figura 7b). Sin embargo, estos modelos son difíciles de aplicar en detección de daño, ya que requieren muchos grados de libertad y la malla debe ser modificada cada vez que se cambia la ubicación o el tamaño de la grieta.

2.4. Técnicas de optimización de posición de sensores

Numerosas técnicas se han desarrollado para resolver el problema de la posición óptima de sensores. En el trabajo desarrollado por M. Meo y G. Zumpano [7], se considera el problema de ubicar sensores en un puente colgante, con el fin de maximizar la información obtenida y así poder caracterizar el comportamiento dinámico de la estructura. Se investigan 6 técnicas de posicionamiento de sensores diferentes, tres basados en la maximización de la matriz de información de Fisher [8], una en las propiedades de los coeficientes de la matriz Covarianza y dos basados en aproximaciones energéticas. Como datos de medición, se consideran los modos normales de la estructura y se utilizan dos métodos de comparación. El primer criterio está basado en el error cuadrático medio entre el modelo del puente desarrollado con el método de elementos finitos y la interpolación cúbica de los modos normales medidos. El segundo criterio evalúa la información adquirida por los sensores según su intensidad y su habilidad para ignorar el ruido al que podrían estar expuestos manteniendo intacta la información relativa a las propiedades estructurales.

Meo y Zumpano, explican de manera muy clara las técnicas que hasta hoy día existen y se han desarrollado con el fin de resolver el problema de optimización de sensores. Principalmente, la ubicación de los sensores, en su trabajo, se basa en la adquisición de información dada por los modos normales de la estructura. El problema que presenta el alumno como trabajo de memoria, no utiliza el mismo criterio de obtención de información, si no que trabaja con las frecuencias de anti-resonancia de la estructura enfocada a maximizar la información sobre daño estructural.

2.5. Algoritmos genéticos

Durante siglos, para solucionar los diferentes problemas de optimización, se han desarrollado una infinidad de métodos iterativos, basados en cálculos de gradientes. Sin embargo, estos métodos suelen estar limitados a la vecindad de punto inicial de búsqueda, obteniendo solo un alcance local. Esto es porque seleccionan la dirección de exploración en función del gradiente, lo que además supone una serie de condiciones sobre la derivabilidad de la función objetivo, que en muchos casos pueden no darse.

Si bien es imposible negar la utilidad de estos métodos, tampoco se puede refutar el hecho de que son muy poco robustos, es decir, si se tiene una función característicamente oscilante y con gran cantidad de máximos locales, es casi evidente que estos métodos serán, más bien, ineficientes. Se necesita entonces un método que sea robusto independiente del tipo de función con la cual se trabajará.

Los algoritmos genéticos [9], son métodos de optimización adaptativos, basados en el proceso evolutivo de selección natural descrito por Darwin. Combinan la supervivencia de los organismos más aptos con el intercambio de información guiada probabilísticamente.

En la naturaleza, los individuos de una población compiten entre sí en la búsqueda de recursos, como comida, agua y refugio, para poder sobrevivir. Aquellos individuos que han tenido más éxito en alcanzar ese objetivo, tendrán más posibilidades de reproducirse y generar descendientes, por sobre aquellos que no pueden adaptarse de mejor forma al medio ambiente. Esto significa que los genes de los individuos mejor adaptados, se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes súper-individuos, cuya adaptación es mucho mayor que la de cualquiera de sus antepasados. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven.

Los Algoritmos Genéticos usan una analogía directa con el comportamiento natural. Trabajan con una población inicial de individuos, cada uno de los cuales representa una solución factible a un problema dado. Para el correcto funcionamiento del algoritmo solo es necesaria una medida de desempeño de los candidatos a solución. En la naturaleza, esto equivaldría al grado de efectividad de un organismo para competir por unos determinados recursos. Cuanto mayor sea la adaptación de un individuo al problema, mayor será la probabilidad de que el mismo sea seleccionado para reproducirse, cruzando su material genético con otro individuo seleccionado de

igual forma. Este cruce producirá nuevos individuos, descendientes de los anteriores, los cuales comparten algunas de las características de sus padres.

De esta manera se produce una nueva población de posibles soluciones, la cual remplaza a la anterior y verifica la interesante propiedad de que contiene una mayor proporción de buenas características en comparación con la población anterior. Así a lo largo de las generaciones las buenas características se propagan a través de la población. Favoreciendo el cruce de los individuos mejor adaptados, van siendo exploradas las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido bien diseñado, la población convergerá hacia una solución óptima del problema.

Los algoritmos genéticos presentan variadas ventajas y desventajas:

- Operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales
- Cuando se usan para problemas de optimización, resultan menos afectados por los máximos locales (falsas soluciones) que las técnicas tradicionales, sobretodo en espacios multimodales (de varios máximos)
- Funcionan de forma independiente de la función objetivo, mientras que los métodos tradicionales suelen requerir una gran cantidad de información extra, como las derivadas en el caso de las técnicas basadas en el gradiente.
- Usan operadores probabilísticos, en vez de los típicos operadores determinísticos de las otras técnicas.

Entre las desventajas, se pueden nombrar las siguientes:

- Pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen; tamaño de la población, número de generaciones, etc.
- Si los operadores y parámetros del algoritmo no son adecuados, este puede converger prematuramente a un máximo local.

Los algoritmos genéticos, al menos como fueron concebidos inicialmente, requieren que las posibles soluciones sean codificadas en cadenas de largo infinito y con un alfabeto también infinito (posibles valores para las posiciones de esta cadena). Esta representación, determina la estructuración del problema y define los parámetros y operadores genéticos que serán utilizados. Cada cromosoma, está representado por una secuencia de genes de cierto alfabeto. El alfabeto, puede ser de dígitos binarios (0 y 1), números reales y símbolos, entre otros.

El alfabeto binario, es preferido debido a su simplicidad, sin embargo, la codificación real es más eficiente en problemas donde el valor de las variables es continuo y es inherentemente más rápido ya que no es necesaria la decodificación de las variables antes de cada evaluación, como es el caso de las variables binarias.

Se han nombrado varias de las características principales que diferencian a los algoritmos genéticos de los métodos tradicionales. A continuación se presentan los tres operadores básicos que hacen posible este comportamiento [10]: selección, recombinación y mutación.

2.5.1. Selección

Durante la ejecución del algoritmo, los padres deben ser seleccionados para la reproducción, a continuación dichos padres se cruzarán generando dos hijos, sobre cada uno de los cuales actuará un operador de mutación. El resultado de la combinación de las anteriores funciones será un conjunto de individuos (posibles soluciones al problema), los cuales en la evolución del Algoritmo Genético formarán parte de la siguiente población.

En este proceso, las cadenas que representan a los individuos, son copiados en relación a su función de desempeño. Esto favorece a los individuos mejor adaptados, ya que a cada individuo se le asigna una probabilidad de ser seleccionado que es proporcional a su función de adaptación. Este operador es, claro está, una artificialidad de la selección natural, donde sobrevive el individuo más fuerte (de mejor desempeño).

La manera más común de utilizar este operador es a través del *método de la ruleta*. Este método consiste en generar una plataforma de selección en el que cada espacio (asociado a cada individuo) es proporcional a su desempeño. De esta forma, si se utiliza un generador de números aleatorios uniforme, la probabilidad de que un individuo sea elegido es:

$$p_i = \frac{f_i}{\sum_{i \in I} f_i} \tag{2.19}$$

Donde f_i es el desempeño del individuo i, e I es la población. Con este operador se determina que individuos son seleccionados para entrar al pozo de reproducción, donde trabajan el resto de los operadores genéticos. Este proceso se repite hasta que se completa la nueva generación de individuos.

Otro método de selección existente, es la *geométrica normalizada,* donde la probabilidad de seleccionar al mejor individuo (q) viene dado por:

$$p_i = q'(1-q)^{r-1} \tag{2.20}$$

Donde:

$$q' = \frac{q}{1-(1-q)^p} \tag{2.21}$$

r es el rango del individuo y *p* es el tamaño de la población.

Por último, existe el método de selección mediante *torneo.* Este método, escoge aleatoriamente *n* individuos de la población e inserta al mejor de ellos en una nueva población. M.A. Rao [11], muestra que este método presenta una mejor efectividad de selección, cuando el torneo se aplica entre dos individuos.

2.5.2. Recombinación

Luego de la selección, dos individuos combinan sus cromosomas con una probabilidad p_r. Este operador, genera dos descendencias producto del intercambio de información genética entre los individuos seleccionados.

Recombinación Simple: Es la recombinación más básica. Este operador, selecciona un punto al azar, llamado "*punto de cruce*" dentro de la cadena de codificación de cada individuo. En este punto se realiza un corte, generando dos divisiones de cada cadena, las que son intercambiadas entre sí, para generar nuevos cromosomas completos (Figura 8).

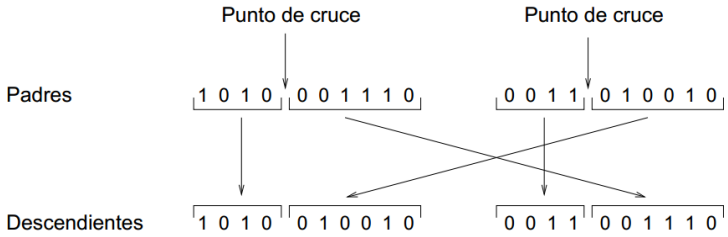

Figura 8: Operador de recombinación simple.

Recombinación de dos puntos: En este caso, se seleccionan dos puntos dentro de la cadena de codificación de los individuos para luego realizar el intercambio. (Figura 9)

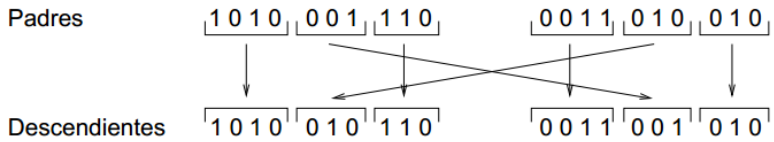

Figura 9: Operador de recombinación basado en dos puntos

Recombinación uniforme: Este operador, mira cada gen de los padres, y asigna al azar un gen de un padre a un hijo y el gen del otro padre al otro hijo. De esta manera, cada gen del cromosoma descendiente, tiene un 50% de posibilidad de pertenecer a cada uno de los padres. Para que esto ocurra, primero se genera una máscara al azar. Esta máscara, es un cromosoma con genes de valores 1 ó 0, del mismo largo que los padres. Dependiendo del valor de cada gen de la máscara, se define si el gen del padre se asignará a uno u otro hijo. (Figura 10)

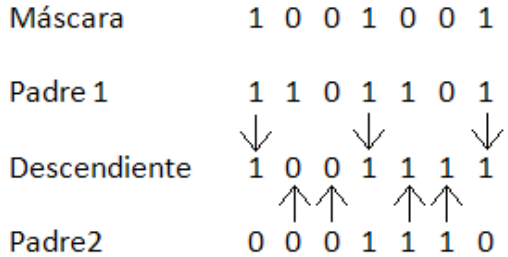

Figura 10: Recombinación uniforme.

En caso de no proceder la recombinación (no se logra vencer la probabilidad) la descendencia es igual a los individuos seleccionados. Generalmente se recomienda que la probabilidad de recombinación p_r oscile entre 0.6 y 0.8, dado que éste operador es el principal mecanismo de esparcimiento de información en los algoritmos genéticos.

2.5.3. Mutación

El operador de mutación se aplica a cada hijo de manera individual, y consiste en la alteración, de alguno de los genes componentes del individuo. Esta mutación se realiza con una probabilidad p_m generalmente pequeño. La Figura 11 muestra la mutación del quinto gen del cromosoma, el cual cambia su valor de cero a uno. Este tipo de mutación, en particular, se llama mutación binaria y es aquella se utiliza en el desarrollo de este trabajo de Título.

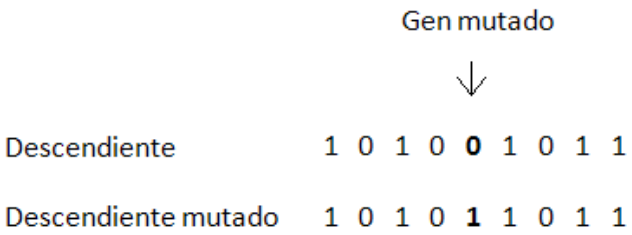

Figura 11: Mutación binaria.

Si bien puede, en principio, pensarse que el operador de cruce es más importante que el operador de mutación, ya que proporciona una exploración rápida del espacio de búsqueda, éste último asegura que ningún punto del espacio de búsqueda tenga probabilidad cero de ser examinado, y es de capital importancia para asegurar la convergencia de los Algoritmos Genéticos.

Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de las generaciones sucesivas de tal manera que la adaptación media extendida a todos los individuos de la población, así como la adaptación del mejor individuo se irán incrementando hacia el óptimo global. El concepto de convergencia está relacionado con la progresión hacia la uniformidad: un gen ha convergido cuando al menos el 95 % de los individuos de la población comparten el mismo valor para dicho gen. Se dice que la población converge cuando todos los genes han convergido. Se puede generalizar dicha definición al caso en que al menos un poco de los individuos de la población hayan convergido.

2.6. Algoritmos Genéticos Paralelos

Una de las aplicaciones de los algoritmos genéticos, son los algoritmos de optimización paralelos (figura 12), los cuales son programados a través del software Matlab. Este método de optimización es básicamente un algoritmo genético que consiste en dividir la población inicial en varias poblaciones, de manera que cada una encuentre una solución óptima. Si las soluciones encontradas no mejoran en cada población estas son nuevamente divididas y sometidas al algoritmo, si en cambio, se encuentran candidatos con óptimos que mejoren la solución, se comparan para ver si son iguales. Si lo son, el proceso termina y se encuentra el óptimo global, de no ser así,

pasan por un proceso en donde las poblaciones divididas migran a otras para luego repetir el proceso inicial.

Los algoritmos genéticos paralelos son particularmente fáciles de implementar y proporcionan un rendimiento numérico superior que los algoritmos genéticos simples. Además de ser más rápidos que los algoritmos genéticos secuenciales, conducen a mejores resultados. Muchos estudios muestran que con la implementación de algoritmos de optimización paralelos, el tiempo de ejecución puede reducirse por un factor mayor que el número de procesadores utilizado. Por lo tanto, muchos autores ejecutan algoritmos paralelos en un solo procesador y obtienen mejores resultados que con algoritmos genéticos secuenciales.

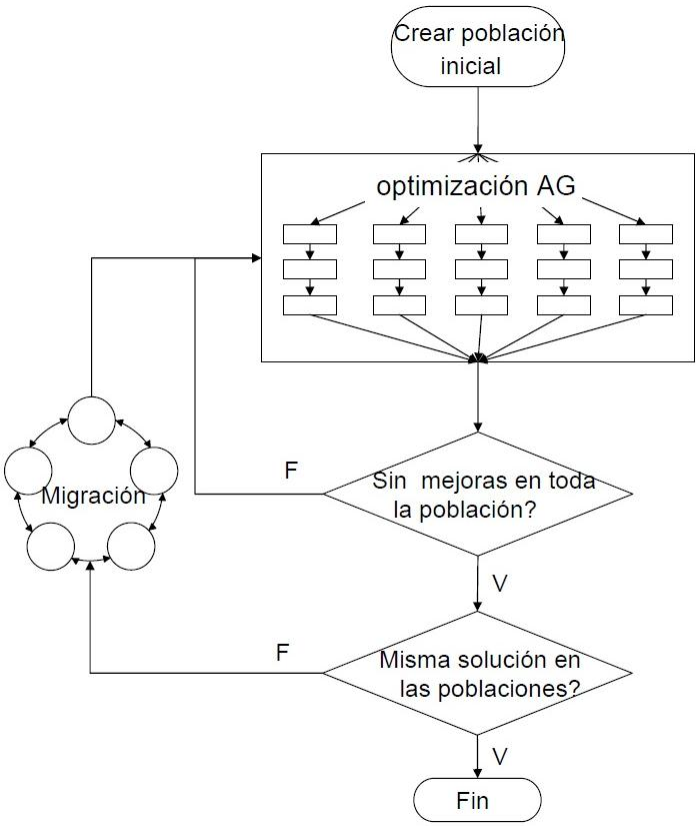

Figura 12: Idea básica del modus operandi de los Algoritmos Genéticos Paralelos

2.7. Modelo experimental

2.7.1. Procesamiento de señales

El procesamiento digital de señales es una herramienta muy importante en el análisis de sistemas. Dado que el objetivo del procesamiento de señales es extraer el máximo de información, es beneficioso estudiarlas en distintos dominios. Las señales medidas son funciones del tiempo y para estudiar su contenido en frecuencias, es más fácil examinarlas en el dominio de las frecuencias. La transformada (inversa) de Fourier [12] permite transformar de manera sencilla una señal en el tiempo a una señal en frecuencia y viceversa.

2.7.2. Ventanas

El uso de ventanas de tiempo no se puede evitar en el procesamiento digital de una señal. En general se buscan ventanas que reduzcan las discontinuidades en los extremos de la señal, dado que reducen el error por “lakeage” [13] al forzar a la señal a ser periódica. La selección de una ventana de tiempo, es siempre un compromiso entre una buena estimación de la amplitud y una buena resolución espectral.

En un test de impacto la señal de entrada es una señal tipo pulso y la respuesta es una combinación de sinusoides que disminuye en el tiempo (figura 13).

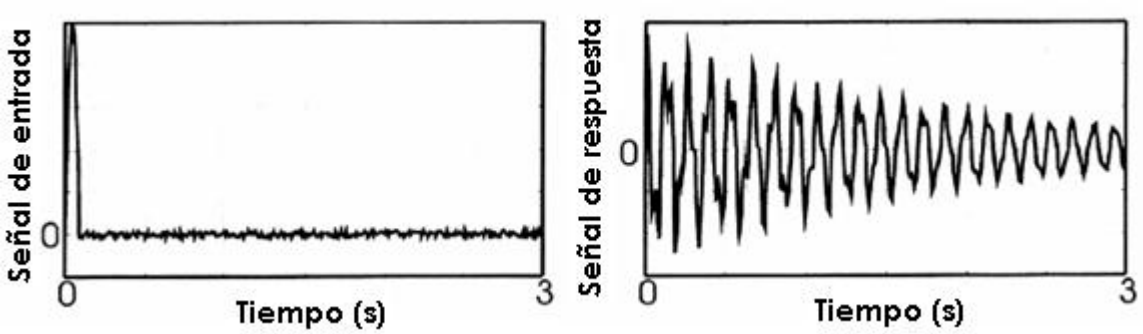

Figura 13: Señal de entrada y señal de salida

Dado que la fuerza es de corta duración, la señal llega a cero antes del final del bloque de tiempo y el resto de lo medido es básicamente ruido experimental no deseado. Para solucionar esto, se aplica una ventana exponencial durante el pulso, lo que permite que la señal descienda a cero justo después del pulso y permanece igual a cero durante el resto de bloque de tiempo.

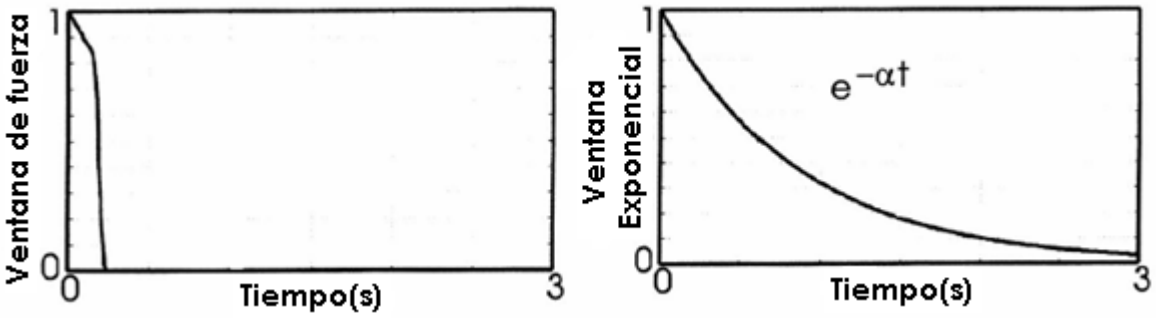

Figura 14: Ventanas de Fuerza y Exponencial para las señales de entrada y salida respectivamente

La combinación de una ventana de fuerza y una ventana exponencial, añade amortiguamiento al sistema. Las funciones de respuesta en frecuencia resultantes van a contener el efecto de este amortiguamiento extra.

3. Metodología

Se trabaja con el Software Matlab y su complemento GAOT para la programación en Elementos Finitos y Algoritmos genéticos, respectivamente.

Tal como menciona el título del trabajo a desarrollar por el alumno, el procedimiento a crear se desenvuelve en una viga. Es por eso que se genera un modelo, en elementos finitos, de la estructura, el cual permite encontrar frecuencias de anti-resonancia de la misma. Como se mencionó en la sección 2.2, las frecuencias de anti-resonancia son características locales de una estructura, es decir, varían para cada FRF y dependen de la ubicación de la fuerza aplicada, como también del punto de medición. Así, dependiendo de la discretización del modelo en EF, se tiene una cantidad definida de nodos, ergo, una cierta cantidad de frecuencias de anti-resonancia.

La discretización del modelo en EF, depende de su precisión para encontrar dichas frecuencias. Así, el número de elementos finitos del modelo, es el mínimo de elementos necesarios para definirlas de la mejor manera posible.

Seguidamente, una vez encontradas las mencionadas frecuencias, se debe determinar su sensibilidad al daño estructural. Este proceso también se realiza con el método de EF. La sensibilidad al daño, numéricamente, no es más que la derivada de la frecuencia de anti-resonancia con respecto al daño. El daño, se puede modelar, variando en un cierto porcentaje, la rigidez de algún elemento del modelo. Recreando esto en cada elemento, se puede determinar la variación de las frecuencias de anti-resonancia (su sensibilidad al daño) las cuales dependen de la ubicación del daño y del punto de medición. Se obtiene finalmente, un conjunto de matrices (una por cada punto de medición) de sensibilidad al daño de las frecuencias.

Todo esto permite, en el proceso de optimización, poder discriminar las frecuencias que entregan una mayor información respecto a la presencia de daño estructural y su habilidad para ignorar el ruido al que podrían estar expuestos manteniendo intacta la información relativa a las propiedades estructurales.

Una vez realizado lo anterior, se procede a definir una función objetivo que relacione la información de daño y la ubicación y número de los sensores. Para efectuar esto, se hace uso de los antecedentes recopilados, respecto a las técnicas de optimización de posición de sensores (sección 2.4). Dicha función será aquella a optimizar.

Para encontrar el óptimo de la función objetivo, se trabaja en Matlab y su extensión GAOT para la programación de algoritmos genéticos y así encontrar las posiciones óptimas de los sensores de monitoreo. Este método de optimización, es capaz de trabajar con las matrices creadas en el paso anterior y entregar un set de

puntos, los cuales corresponderán a las posiciones óptimas de los sensores para el monitoreo de la estructura.

Para poder verificar que los resultados obtenidos, una vez realizados los pasos anteriores, son correctos y que el procedimiento desarrollado cumple con las expectativas, se realiza una verificación experimental utilizando algoritmos de detección de daños disponibles y datos experimentales. Se compara la distribución de sensores entregados por la metodología desarrollada con una distribución, a definir, que generalmente se utiliza en la práctica. Para esto se trabaja en el laboratorio de Sólidos, ubicado en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile.

4. Procedimiento y Resultados

4.1. Modelo de la viga

El problema de estudio consiste en una viga libre que se puede mover en todas direcciones, pero solo se estudia sus desplazamientos verticales. Para el método de elementos finitos, la viga es discretizada en 20 secciones de igual tamaño, lo que da un total de 42 grados de libertad contando desplazamientos verticales y rotacionales.

Datos:

- $E = 2x10^{11}$ [Pa]
- L = 1 [m]
- $\rho = 7800$ [Kg/m^3]
- A= 0.01 x 0.025 [m^2]

La resolución de este problema se realiza programando una función en Matlab que crea las matrices de rigidez y de masa para cada uno de los elementos de la viga y luego las ensambla (Anexo 1). Cabe destacar que éste, es un modelo de una viga que no presenta daño alguno y será la base de comparación para definir la sensibilidad de las anti resonancias.

4.2. Modelo de la viga con presencia de daño

Para poder calcular la sensibilidad de cada una de las anti resonancias de la viga, es necesario, primero que todo, modelar el daño. Esto se realiza a través de los coeficientes de reducción de rigidez, descrito en la sección 2.3.

Para esto, se programa en Matlab, una función que aplica este coeficiente, en cada elemento de la viga, dejando los demás elementos intactos, es decir, primero se modela el daño solo en el primer elemento, luego solo en el segundo y así sucesivamente (Anexo 2). Finalmente, la función programada, entrega 20 vigas distintas, cada una con un daño en un elemento distinto. Para asegurar la detección de los daños en la viga, el daño modelado corresponde a una reducción de rigidez de un 10% (β=0.1).

4.3. Sensibilidad al daño

En cada uno de los casos anteriores (viga con y sin daño) se calculan las anti resonancias. El código numérico que realiza este cálculo se encuentra inmediatamente después de la creación de las vigas correspondientes. Como se describe en la sección 4.1., las frecuencias resonantes se obtienen como respuesta a una excitación dada, ergo, para obtener las frecuencias anti-resonantes se debe representar dicha aplicación de fuerza.

Para el caso de la viga sin daño, el programa entrega una matriz de 7x21 (7 anti resonancias por cada grado de libertad de movimiento vertical) y para el modelo con daño, la función entrega 20 matrices de 7x21, una por cada viga modelada con daño, todo esto para distintas ubicaciones de la aplicación de la fuerza.

Comparando cada una de las vigas dañadas con la viga intacta, se calcula la diferencia porcentual de cada anti resonancia (Anexo 3). La figura 15 muestra la sensibilidad de las frecuencias para dos casos: daño en el quinto y décimo quinto grado de libertad, con la fuerza aplicada en el décimo séptimo nodo. El eje X representa los 21 nodos de la estructura, el eje Y representa las 7 anti-resonancias para cada nodo y el eje Z equivale a la sensibilidad porcentual de las anti-resonancias.

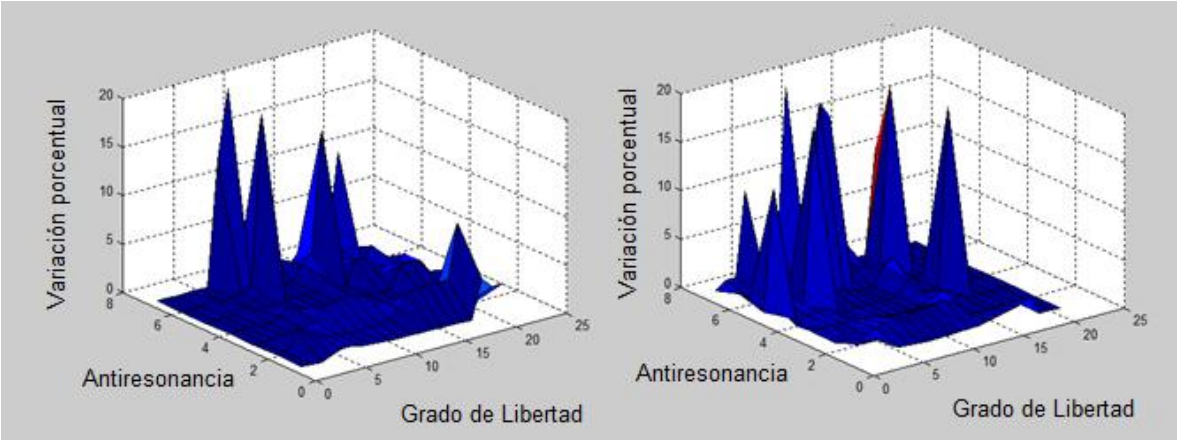

Figura 15: Gráficos de sensibilidad de anti-resonancias

4.4. Ajuste de Modelo

El ajuste de modelo es un método que permite ajustar los modelos analíticos con los modelos experimentales [14]. Un modelo analítico, si bien, puede estar bien formulado, puede distar del modelo experimental en ciertos parámetros, los que conllevan a la presencia de errores del modelo. Un ajuste de modelo, busca disminuir esas diferencias, con el objetivo de lograr que el método analítico se acerque lo más posible a la realidad.

Las frecuencias características de un sistema, dependen en particular de la densidad, módulo de Young y otras propiedades geométricas. Se define λ_j, (j entre 1 y el total de variables a ajustar) como los parámetros de ajuste del modelo y se utilizan para plantear un problema de optimización que minimice la diferencia existente entre el

modelo analítico y el experimental, en este caso, las frecuencias anti-resonantes del sistema.

$$\min \sum_n \sum_i \left\| \frac{\omega_{r,i,n}^{A}(\lambda_1,\lambda_2,\ldots,\lambda_j)^2}{\omega_{r,i,n}^{E}{}^2} - 1 \right\| ;\ li_j \leq \lambda j \leq ls_j \qquad (4.2)$$

Donde, $\omega_{r,i,n}$ es la i-ésima frecuencia de anti-resonancia del i-ésimo grado de libertad, los superíndices A y E, refieren a datos analíticos y experimentales y li, ls hacen referencia a los límites inferior y superior respectivamente.

4.5. Función objetivo

Se definen dos criterios importantes que indican si las anti resonancias son realmente sensibles al daño modelado y que definirán la posición de los sensores y además un tercer criterio que involucra el número de sensores a utilizar.

El primero, es que sea posible detectar la presencia de algún daño en la estructura. Para esto, es necesario que el daño mismo produzca algún cambio notorio entre las frecuencias de anti-resonancia de la viga sin daño y aquellas de la viga con daño. Este cambio notorio se fija en un 2%. Esto, ya que el ruido experimental se estima en ese valor, por lo que luego de eliminarlo de las FRF, solo se pueden detectar sensibilidades de mínimo un 2%, ergo, cambios menores en las frecuencias son imperceptibles. Es por eso que un daño solo será detectado si produce cambios en las anti-resonancias mayor al 2%. De esa manera se define la variable *N,* la cual adquiere el valor N=1, si hay al menos una anti resonancia con una sensibilidad mayor al 2%, para cada posible posición del daño, y N=0, si sucede lo contrario. Es fácil deducir que el mayor valor que puede adquirir esta variable es N=20.

El segundo criterio, es que además de detectar el daño, también sea posible localizarlo, esto es, que el método sea capaz de diferenciar si un daño se encuentra en uno u otro elemento de viga. Esto se logra solo si, para un daño con una ubicación específica, las frecuencias de anti-resonancia para distintos nodos presentan diferencias en sus valores. En caso contrario es imposible localizar la falla. Para esclarecer esto, se presenta el siguiente ejemplo. A continuación se muestran las frecuencias de anti-resonancia (menores a 3000 Hz) para los primeros tres nodos de una viga modelada a) sin daño, b) daño en el primer elemento y c) daño en el segundo elemento (β=0.1), con la fuerza de excitación aplicada en el primer grado de libertad.

$$\begin{bmatrix} 91.89 & 97.16 & 104.08 \\ 297.78 & 216.2 & 343.5 \\ 621.32 & 662.52 & 729.67 \\ 1062.6 & 1137.8 & 1268.8 \\ 1621.8 & 1743.9 & 1963 \\ 2299.1 & 2482.5 & 2810.3 \end{bmatrix} \quad \begin{bmatrix} 94.15 & 99.56 & 106.65 \\ 305.07 & 324.01 & 351.78 \\ 636.38 & 678.88 & 747.69 \\ 1088 & 1165.9 & 1301 \\ 1660.3 & 1787.1 & 2011.5 \\ 2352.4 & 2544 & 2878.6 \end{bmatrix} \quad \begin{bmatrix} 96.17 & 100.68 & 106.64 \\ 298.6 & 330.58 & 352 \\ 645.98 & 695.32 & 747.79 \\ 1154.6 & 1196.3 & 1300.3 \\ 1754.6 & 1806.5 & 2011.7 \\ 2325.3 & 2606.3 & 2879.3 \end{bmatrix}$$

a) Viga modelada sin daño b) Viga con daño en primer elemento c) Viga con daño en segundo elemento

Se puede notar, que los valores de una misma columna (anti-resonancias para un mismo grado de libertad), en las tres matrices, presentan una cierta variación entre ellas. Esto quiere decir, que la presencia de daño en la estructura, influye en las

frecuencias de anti-resonancia de la viga, propiedad que, como se explicó en la sección 2.2, las catapulta como un buen indicador de daño. Además, en esa misma columna, no todas las anti-resonancias presentan la misma variación (no exhiben la misma sensibilidad). Sin embargo, al observar las anti-resonancias para el tercer nodo, en los dos casos de la viga dañada, se puede notar que la variación es mínima. Esto significa, que las frecuencias del tercer nodo no son capaces de diferenciar si el daño se ubica en el primer o segundo elemento de viga.

$$\begin{bmatrix}104.08\\343.5\\729.67\\1268.8\\1963\\2810.3\end{bmatrix} \qquad \begin{bmatrix}106.65\\351.78\\747.69\\1301\\2011.5\\2878.6\end{bmatrix} \qquad \begin{bmatrix}106.64\\352\\747.79\\1300.3\\2011.7\\2879.3\end{bmatrix}$$

a) Anti-res. nodo 3 sin daño b) Anti-res. nodo 3 daño 1er elemento c) Anti-res. nodo 3 daño 2do elemento

Ahora bien, si se fija la atención en las anti-resonancias del segundo nodo para los tres casos, se observa que la variación en este caso es mejor que en el caso anterior, lo que ayuda a concluir que el segundo nodo es potencialmente una mejor ubicación de sensores que el nodo 3, ya que el primero si es capaz de localizar la ubicación del daño en desmedro del tercer nodo.

Analíticamente, esto se refleja en el producto punto entre los vectores anti-resonancia de cada nodo para cada posible ubicación del daño. A modo de ejemplo, si ubicados en el quinto nodo de la barra, se calcula el producto punto entre todos los vectores anti-resonancia de ese nodo para las 20 posibles ubicaciones del daño y se calcula su promedio y luego se realiza el mismo procedimiento pero para el octavo nodo, aquel nodo que presente el menor valor entre el promedio de los productos punto, será una mejor eventual ubicación del sensor, ya que dicho nodo presenta mayores variaciones en sus anti-resonancias para las distintas posibles ubicaciones del daño.

De esta manera, se define el valor *PP,* que representa el promedio entre los productos escalares de los vectores anti-resonancia, para cada uno de los 21 grados de libertad y puede adquirir valores que pertenecen al intervalo $]0,1[$.

Finalmente, para poder optimizar también el número de sensores a utilizar, se define un tercer valor *n*, que representa la cantidad de sensores que se necesitan para el óptimo monitoreo de la viga. Este valor, influye en la función objetivo a través de un factor pequeño pero negativo, de manera de minimizar el número de sensores a utilizar.

Precisados estos dos criterios de efectividad, son ellos los que definen la función objetivo que será aquella a optimizar por el Algoritmo Genético (Anexo 4). El valor de un individuo (posible solución) evaluada en esta función está dada por:

$$val = \alpha * {}^{N}/_{20} + \beta * (1 - PP) - \gamma * n \qquad (4.1)$$

Donde α=0.5, β=0.5 y γ=0.01. Los primeros dos factores fueron fijados en ese valor ya que mostraron otorgar un mejor resultado para la distribución de sensores en comparación con otras combinaciones y el tercer factor se fija en 0.01, con la intención de que el signo negativo no influya considerablemente en el valor adquirido por cada individuo de la población en los algoritmos genéticos paralelos, al evaluarse en la función objetivo.

4.6. Selección de parámetros y operadores del Algoritmo Genético

Tal como se explica en la sección 2.5., para trabajar con algoritmos genéticos, se debe primeramente definir una población inicial y los operadores de selección, recombinación y mutación.

Dada la naturaleza del problema, se decide codificar cada individuo con alfabeto binario y un largo de 21 genes. Cada gen representa cada grado de libertad donde podría ir ubicado un sensor. Luego, la cantidad y ubicación de los sensores en la viga quedará determinado por los genes que tengan el valor 1.

Si bien, es crucial seleccionar de manera adecuada los operadores del AG ya que esto afecta su efectividad y el tiempo de ejecución [15], el hecho de trabajar con algoritmos genéticos paralelos permite utilizar más de un operador de cada tipo en cada algoritmo. De esta manera, dado que se utilizaron cuatro (4) algoritmos en paralelo para la optimización de la función objetivo, se pueden utilizar distintos operadores en cada una de ellas. Es así que no se discrimina entre los distintos operadores de recombinación explicados en la sección 2.5.2., si no que se utilizan los tres en algoritmos distintos.

Algo parecido sucede con el operador de mutación, solo que en este caso no se utilizan distintos operadores de este tipo, si no que solo se utiliza el ilustrado en la sección 2.5.3., ya que es el único programado para trabajar con lenguaje binario.

Finalmente, solo resta escoger el operador de selección, la mejor combinación de parámetros de los operadores y el mejor tamaño de población. Para la selección, fueron testeados, tres métodos: (1) Geométrica normalizada, (2) Método de la Ruleta y (3) Torneo. Para el primer método, la probabilidad de que el mejor individuo sea seleccionado se fijo en 0.08 [16] y en el tercer método, el número de individuos en cada torneo se fijó en 2 (sección 2.5.1). La figura 16 muestra la convergencia promedio y la desviación estándar, para 15 iteraciones (5 poblaciones iniciales distintas para cada uno de los tres métodos de selección)

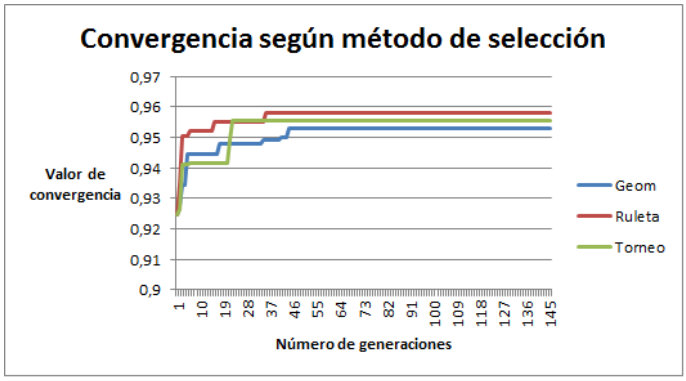

Figura 16: Convergencia del Algoritmo Genético para los distintos métodos de selección.

Una vez escogidos los tres operadores definitivos del AG, se procede a evaluar el método con distintas combinaciones de probabilidad de recombinación y mutación. La figura 17 muestra la combinación que permite alcanzar el máximo valor al evaluar en la función objetivo, esta es, una probabilidad de recombinación de 0.95 y de mutación de 0.04.

Finalmente, se realizan nuevas iteraciones del método, esta vez para seleccionar el tamaño de la población. La figura 17 ilustra que el mejor tamaño de población es de 30 individuos.

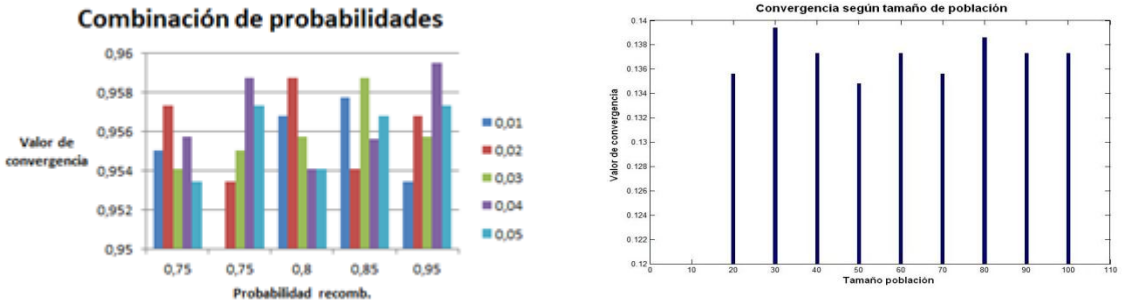

Figura 17: Convergencia del AG según distintas combinaciones de las probabilidades de mutación y recombinación y según tamaño de la población.

4.7. Resultado para la distribución de sensores

Con todos los parámetros y operadores ya definidos para los algoritmos genéticos paralelos, estos se comienzan a iterar para obtener la mejor distribución de los sensores en la viga, que permitirán obtener la cantidad de información adecuada para la detección de daño, la cual es la siguiente:

0 1 0 0 0 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 0 0

Esto quiere decir, que el número óptimo de sensores a utilizar es 7 los cuales irán distribuidos en los nodos 2, 9, 11, 13, 14, 15,19.

4.8 Detección de daño

4.8.1. Función Objetivo

La función objetivo recién explicada se utiliza para encontrar el número y posición óptima de los sensores y así obtener la mayor cantidad de información confiable respecto al comportamiento dinámico de la viga.

Una vez adquirida esta información, en pro de corroborar la eficacia del método programado para la detección de daño, es necesario verificar si ésta es correcta. Para ello, se utiliza un algoritmo de detección de daño ya disponible [17], que utiliza las frecuencias de anti-resonancias adquiridas de los datos experimentales.

Se define el error en las anti-resonancias, representado por la diferencia entre el cuadrado de la razón entre las anti-resonancias experimentales y las analíticas, para la viga dañada, y el cuadrado de la razón entre aquellas para una viga sin daño.

$$\varepsilon_{r,i,n}(\beta) = \left(\frac{\omega^{A}_{r,i,n}}{\omega^{E}_{r,i,n}}\right)^2 - \left(\frac{\omega^{A0}_{r,i,n}}{\omega^{EO}_{r,i,n}}\right)^2 \quad (4.2)$$

Donde $\omega_{r,i,n}$ es la i-ésima anti-resonancia de la n-ésima FRF. Los superíndices A y E, refieren a analítico y experimental respectivamente y el superíndice 0, es para los datos adquiridos de la viga sin daño.

La ecuación 4.2, no busca alcanzar una relación exacta entre los datos analíticos y experimentales, si no que busca alcanzar la misma correlación que el caso sin daño.

4.8.2. Algoritmo Genético

Viviana Meruane desarrolla un algoritmo de detección de daño, utilizando Algoritmos Genéticos Paralelos [18]. Dicho algoritmo, utiliza como método de selección, la geométrica normalizada. Para la recombinación, utiliza la recombinación aritmética, heurística, uniforme y la recombinación de mezcla (BLX crossover), cada una con una probabilidad de 0.8. En cuanto al método de mutación, en cada uno de los AGP utiliza dos tipos, la mutación uniforme y la mutación de borde, las dos con probabilidad de 0.02 y finalmente, una tamaño de población de 40 individuos.

4.9. Detección de daño con datos modelados

Para corroborar el modelo programado y asegurar que existe confiabilidad para la verificación del mismo con datos experimentales, es recomendable, primero que todo, probar la eficacia del modelo con datos modelados. En términos prácticos, esto quiere decir trabajar primero con datos analíticos a los cuales se les agrega un ruido “analítico” modelando datos experimentales. Se realiza esto modelando los datos para una viga con un daño y con dos daños. Los daños se programan en los mismos elementos que presentan las vigas reales con una reducción de rigidez análoga, esto es, una reducción una reducción del 30% para la viga con un solo daño y para la segunda viga, se modela una reducción de rigidez de un 30 y un 60% para el daño menor y mayor respectivamente.

Se obtienen datos modelados para tres distribuciones de sensores distintas, con el objetivo de realizar una comparación entre los resultados obtenidos. Se modela la distribución entregada por el modelo creado, una distribución uniforme y la peor distribución de sensores.

4.9.1. Distribución entregada por el modelo

Las figuras 18 y 19 muestran los resultados de la detección de daño para las dos vigas para la distribución entregada por el modelo detallado en la sección 4.7.

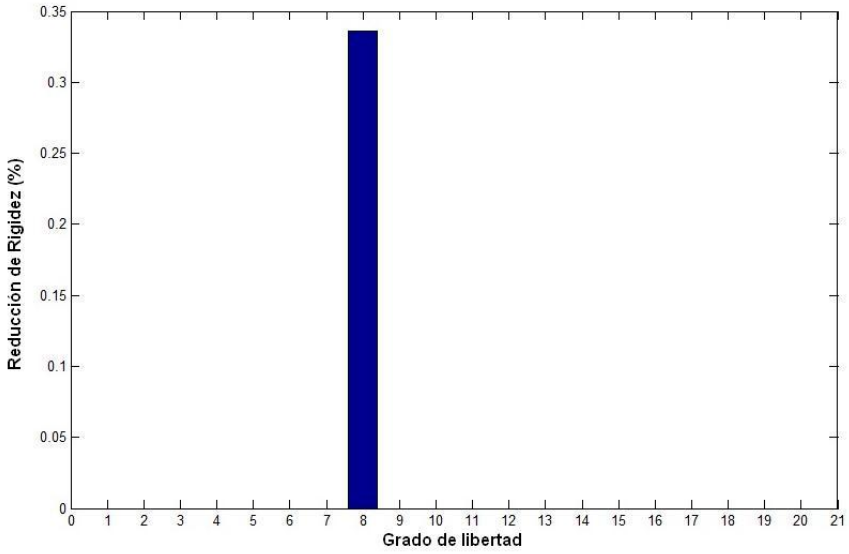

Figura 18: Detección de daño para viga con una sola falla y distribución de sensores entregada por el modelo

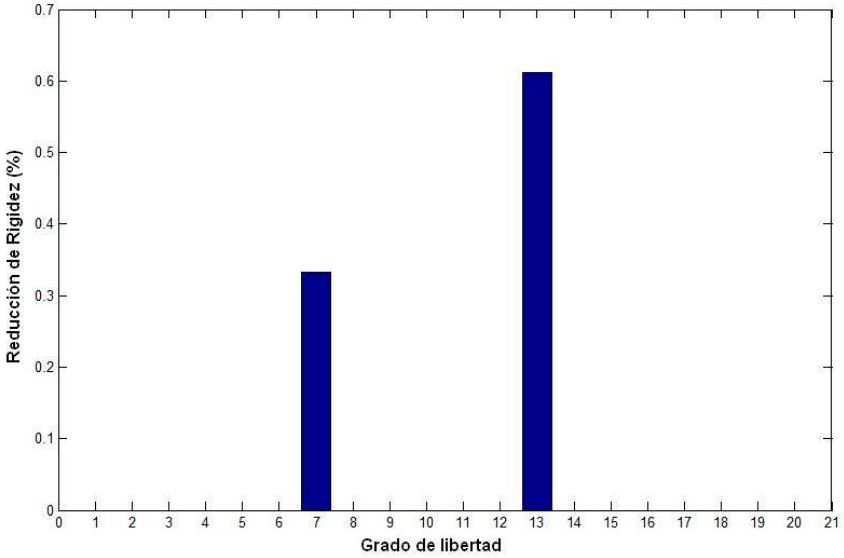

Figura 19: Detección de daño para viga con dos fallas y distribución de sensores entregada por el modelo

4.9.2. Distribución uniforme

Una distribución uniforme corresponde a una distribución equidistante de los sensores a lo largo de la viga, esto es, en los nodos 1, 4, 7, 10, 13, 16, 19. Las figuras 20 y 21 muestran los resultados obtenidos con esta distribución para la detección de daño.

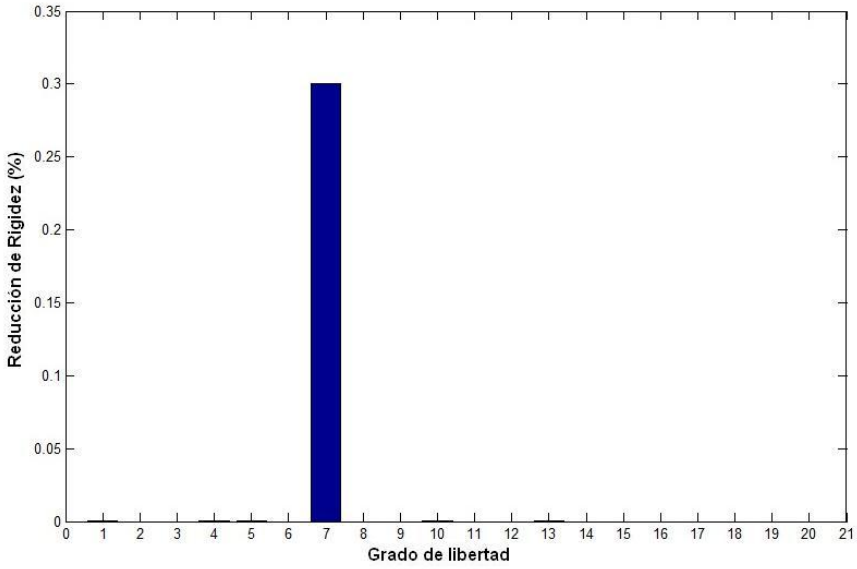

Figura 20: Detección de daño para viga con una sola falla y distribución de sensores uniforme

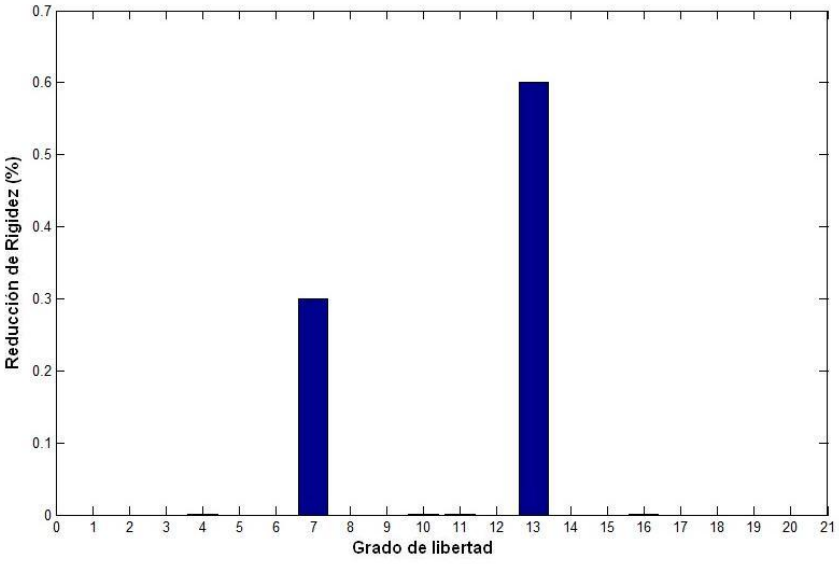

Figura 21: Detección de daño para viga con dos fallas y distribución de sensores uniforme

4.9.2. La peor distribución

Tomando en cuenta que la viga es excitada en uno de sus extremos, se considera que la peor distribución de sensores, es aquella que los ubica desde el otro extremo de la viga, esto es, los 7 sensores ubicados entre los nodos 15 y 21. Esto, ya que la menor cantidad de anti-resonancias detectadas se presentan en los últimos nodos de la barra.

Las figuras 22 y 23 ilustran los resultados obtenidos en este último caso.

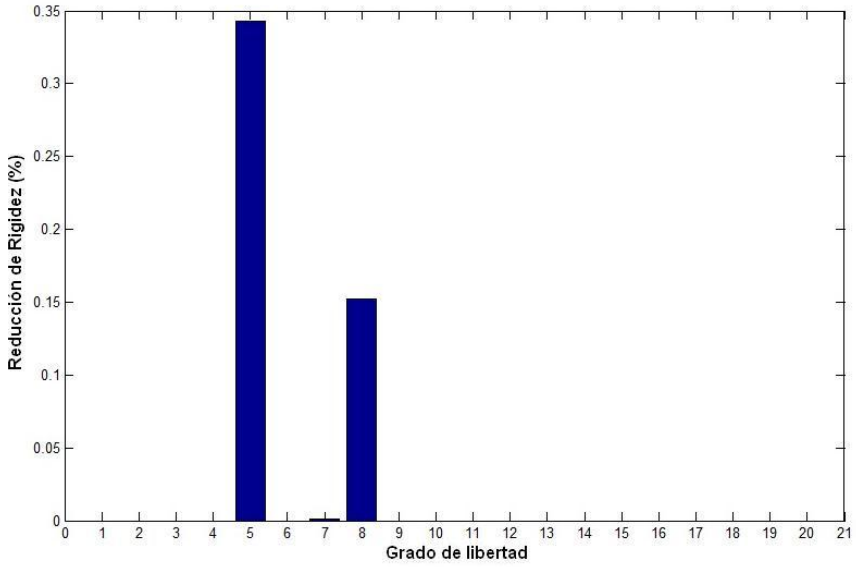

Figura 22: Detección de daño para viga con una sola falla y peor distribución de sensores

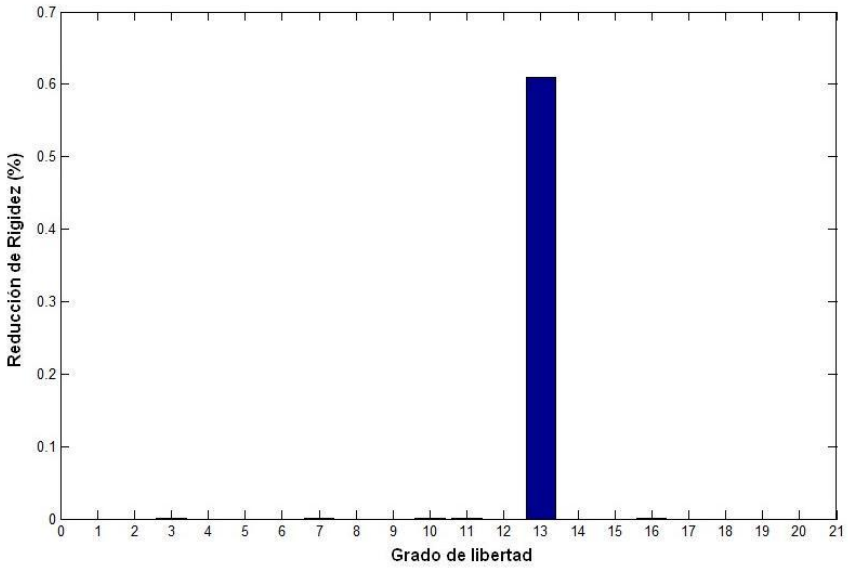

Figura 23: Detección de daño para viga con dos fallas y peor distribución de sensores

Considerando que, para la viga con un solo daño, este fue modelado en el elemento 7 y para el otro caso, las fallas se modelaron en los elementos de viga 7 y 13, se puede observar que la distribución uniforme de sensores es capaz de detectar con mayor exactitud el daño, particularmente, en el caso de la viga con un solo daño, donde la localización de la falla es exacta en comparación con la distribución propuesta por el modelo que localiza el daño en el octavo elemento. Esto puede indicar que la distribución uniforme es más eficaz que la distribución propuesta. Sin embargo, también se debe considerar en el análisis, la reducción de rigidez obtenida. La tabla 1 muestra los resultados adquiridos por ambas distribuciones (propuesto y uniforme) respecto a este parámetro.

Tabla 1: Comparación de la reducción de rigidez de la viga, para la distribución de sensores propuesta y uniforme

Distibucion	Reducción de rigidez	
	Viga con un daño	Viga con dos daños
Simulado	30%	30% y 60%
Del modelo	34%	34% y 64%
Uniforme	31%	31% y 62%

La tabla 1 indica que con la distribución de sensores propuesta por el modelo, se detecta una reducción de rigidez mayor que con la distribución uniforme. En la práctica la reducción de rigidez, es un factor de suma importancia ya que indica que tan grande es el daño localizado. Por otro lado, si bien la distribución propuesta no fue capaz de localizar el daño con total exactitud, se considera que su detección fue suficientemente precisa ya que dista muy poco de la real locación del daño. En la práctica, al momento de revisar la estructura en la búsqueda del daño, una pequeña diferencia entre la detección y la verdadera ubicación del daño, no impide realmente poder encontrar la falla y poder realizar las acciones pertinentes a ello.

Finalmente, la peor distribución de sensores arroja resultados que se alejan ampliamente de la realidad modelada. En el primer caso (figura 20), se detecta un daño falso y en el segundo caso (figura 21), la información dinámica adquirida con es suficiente para detectar los dos daños presentes en la viga.

De esta manera, se concluye que el modelo cumple con su objetivo y puede ser testeado con datos experimentales reales.

4.10. Obtención datos experimentales

4.10.1. Rango de frecuencias

Es importante definir el rango de frecuencias para poder trabajar con valores de anti-resonancias que sean capaces de detectar un daño con confianza.

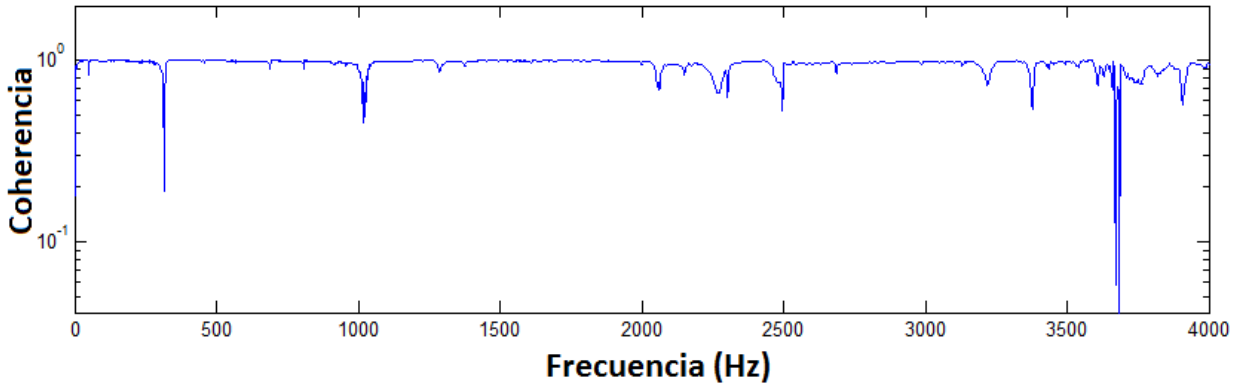

Figura 24: Función de Coherencia

La función de coherencia de la figura 24 muestra que para frecuencias menores a 2000 [Hz], la coherencia es casi perfectamente lineal y que para frecuencias mayores comienzan a detectarse problemas que pueden deberse a los motivos detallados en la sección 2.1.2. Por esta razón se trabaja con un rango de frecuencias entre 0 y 2000 [Hz].

4.10.2. Montaje experimental

Los datos experimentales se obtienen en el Laboratorio de Sólidos Mecesup ubicado en la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. El programa a utilizar es ECON, Avant Series. Para la obtención de datos, el programa debe setearse de la siguiente manera:

- Rango de frecuencias: 0 - 2000 (Hz)
- Número de mediciones promediadas: 5
- Tipo de Ventana para canal de excitación: fuerza
- Tipo de ventana para canal de respuesta: exponencial

Se disponen de 4 barras diferentes: una sin daño, la segunda con la presencia de un daño en el séptimo elemento. Una tercera barra presenta dos fallas, una más pequeña ubicada en el séptimo elemento y una mayor en el decimotercer elemento. Y por último, una cuarta barra también con dos daños, donde la más grande es aún mayor que el segundo daño de la tercera barra, y que están ubicados en los elementos 8 y 17. Cada viga posee idénticas propiedades mecánicas y características geométricas, las cuales se detallan a continuación:

- Largo barra (L): 1 m
- Ancho de la sección (b): 0,01 m
- Alto de la sección (t): 0,025m
- Módulo de Young (E): 2,1 e11
- Densidad (ρ): 7800
- Coeficiente de Poisson (μ): 0,3
- Número de elementos: 20
- Largo de los elementos (L_e): 0,05 m

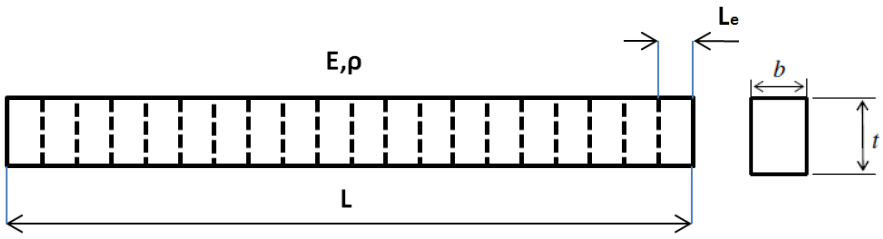

Figura 25: Ilustración de viga de trabajo

La figura 26, ilustra el tipo de falla con la que se trabaja. El valor l_c varía dependiendo de la falla y la viga. Para la viga con un solo daño l_c=0.009m. Para la primera viga con dos daños l_c=0.006 y 0.012m. Finalmente para la última viga, también con dos fallas, l_c=0.006m y 0.017m.

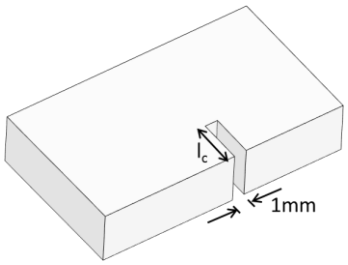

Figura 26: Daño presente en vigas de trabajo

La idea de este trabajo, es poder detectar el daño que se encuentra en una viga libre. Para representar esto se procede a colgar la barra en una estructura tipo caballete con tal de simular la condición de borde “libre-libre” como se muestra en la figura 27.

Figura 27: Viga colgada en estructura tipo caballete para representar la condición de borde “libre-libre”.

El proceso de la obtención de datos, comienza con la instalación del equipo de trabajo. La figura 28 ilustra un plano general de la instalación del equipo.

Figura 28: Ilustración de instalación de equipo de trabajo para la toma de datos

Si bien, según la sección 4.6.2, se necesitan 7 sensores para la obtención de la información dinámica de la viga, solo se disponen de 4 sensores por lo que por cada viga, se realizan dos series de obtención de datos.

Figura 29: Viga con los 4 sensores de monitoreo

El proceso consiste en golpear la viga en uno de sus extremos libres, en el cual se define el primer elemento de viga, con un martillo modal, formado por la punta del martillo (intercambiable por diferentes materiales), un sensor de fuerzas, una masa y el mango, logrando un golpe que se represente por un impulso.

Figura 30: Martillo modal

La tabla 2 detalla las características del martillo.

Tabla 2: Características del martillo modal utilizado para la toma de datos

Rango (V)	**10**
Tipo de transductor	Fuerza
Sensibilidad (mv/N)	11,2
Masa (Grs)	100
Dureza de punta	Silicona

El sensor que se utilizó para medir las respuestas de excitación en el análisis modal experimental fue el acelerómetro. El acelerómetro es el sensor más común utilizado para medir la señal de respuesta. Este mide la aceleración de un punto en la estructura donde la señal de salida viene en forma de voltaje. Esta es transformada por un acondicionador antes de ser procesada por otro hardware o software.

En el caso de este montaje se utilizó el tipo de acelerómetro más común, el acelerómetro piezoeléctrico. Estos sensores contienen un cristal piezoeléctrico en su interior, que produce una carga eléctrica al ser deformado. A continuación se señalan las características de los cuatro acelerómetros usados en el montaje experimental de las vigas.

Tabla 3: Características de los 4 acelerómetros utilizados

1er sensor	**Rango (V)**	10
	Tipo de transductor	Aceleración
	Sensibilidad (mv/G)	100.9
2do sensor	**Rango (V)**	10
	Tipo de transductor	Aceleración

	Sensibilidad (mv/G)	96
3er sensor	**Rango (V)**	10
	Tipo de transductor	Aceleración
	Sensibilidad (mv/G)	98.9
4to sensor	**Rango (V)**	10
	Tipo de transductor	Aceleración
	Sensibilidad (mv/G)	95.6

Una vez listo el montaje se procede a realizar la toma de datos. Se realizan 5 golpes y de ellos se obtiene información promediada que refiere a las FRF's de cada viga. Las figuras 31 y 32 ilustran las FRF's del segundo nodo para las primeras dos vigas dañadas. La curva azul representa los datos analíticos y la roja los datos experimentales. Para los datos analíticos, se simula un daño para cada viga de trabajo. Se considera que para la viga que presenta solo un daño, la reducción de rigidez en el elemento correspondiente (séptimo), es de un 30%. Lo mismo se modela para los daños menores de las otras dos vigas. La reducción de rigidez local que producen los daños mayores, en las dos últimas barras, se modelan en un 60% y 80%. Se puede notar que casi no hay diferencia entre las dos curvas, lo que indica que la información adquirida es confiable.

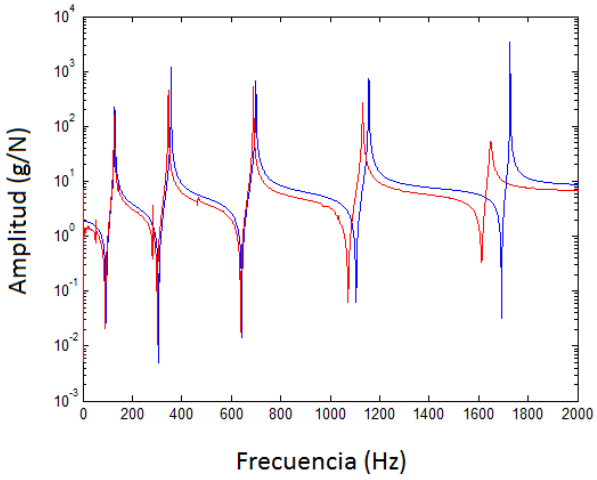

Figura 31: FRF nodo 2, viga con un daño.

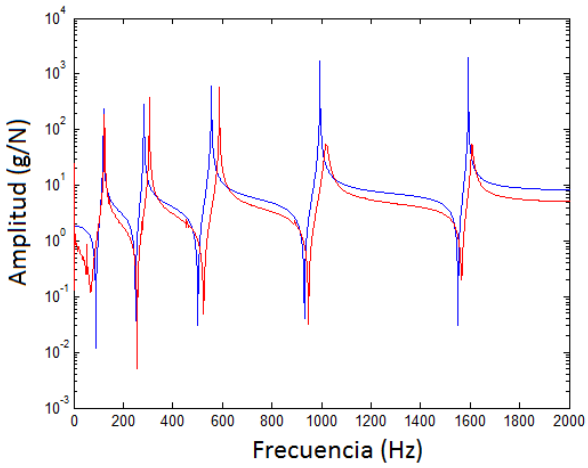

Figura 32: FRF nodo 2, viga con dos daños.

4.11. Resultados detección de daños

Con las FRF's de cada una de las tres vigas dañadas, para los 7 nodos, es posible identificar las anti-resonancias de cada una de ellas. En la sección 2.1., se menciona que las anti-resonancias, en una función de respuesta en frecuencia, son aquellos mínimos observables. Con estas anti-resonancias y el algoritmo de detección de daño mostrado en la sección 4.8., se obtienen las respectivas variaciones de rigidez en los elementos de viga correspondientes.

4.11.1. Viga con un daño

Anteriormente, se había mencionado que una de las vigas dañadas, solo presenta un daño, el cual está ubicado en el séptimo elemento de viga. Las anti-resonancias encontradas para esta viga como las de la figura 31, otorgan la información necesaria para la detección del daño, la cual se muestra en la Figura 33.

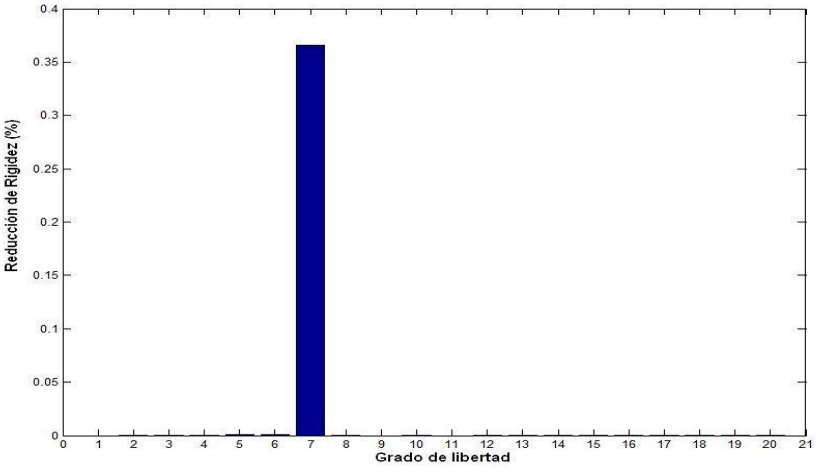

Figura 33: Detección de daño para viga con una falla.

La figura muestra que los datos experimentales en conjunto con el modelo creado por el alumno, indican una reducción de rigidez significativa en el séptimo

elemento de la viga (37%). Este resultado es aún mejor que el mostrado en la figura 18 (sección 4.9.), donde el daño detectado se ubicaba en el octavo elemento.

4.11.2. Viga con dos daños

Seguidamente, se comprueba el modelo creado, con la viga que presenta dos daños, de largo l_{c1}=0.006m y l_{c2}=0.012m. La figura 22 muestra alguna de las anti-resonancias de esta viga las cuales al aplicarlas al algoritmo de detección de daños, muestra el resultado ilustrado en la figura 34.

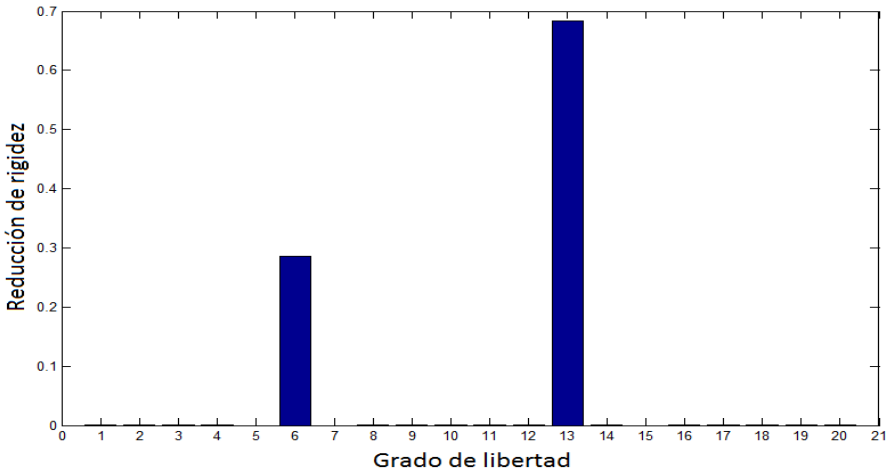

Figura 34: Detección daño segunda viga (l_{c1}=0.006m y l_{c2}=0.012m).

4.12. Comparación de resultados

Para asegurar que la distribución de sensores es la mejor, al igual como se procede en la sección 4.9, se compara la detección da daño ilustrada en la sección anterior, con las obtenidas con una distribución uniforme y con la peor distribución de sensores. Los resultados obtenidos se ilustran desde la figura 35 hasta la figura 38.

4.12.1. Viga con un daño

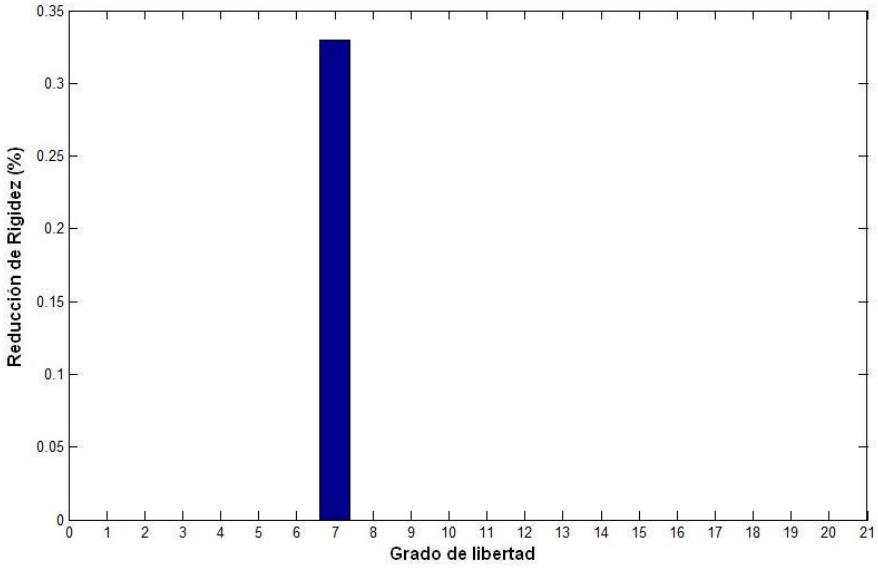

Figura 35: Detección experimental del daño para viga con una falla y distribución uniforme de sensores

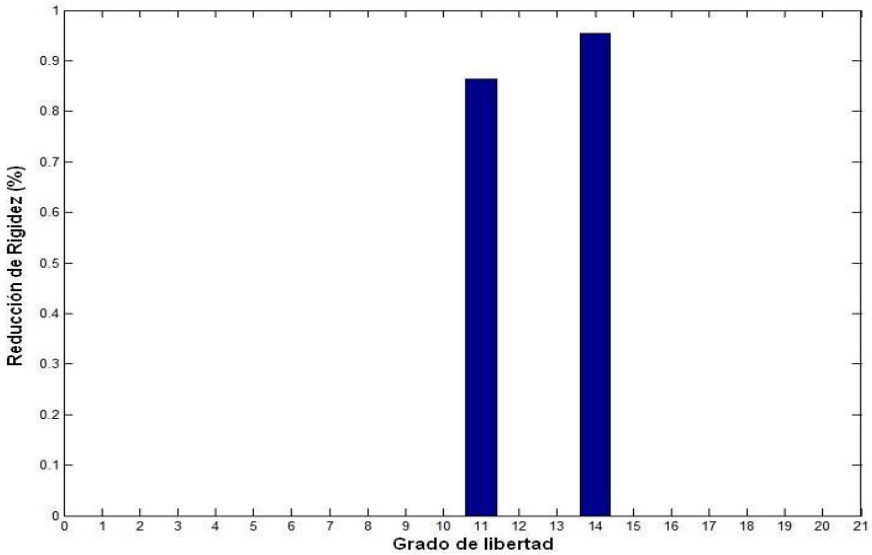

Figura 36: Detección experimental del daño para viga con una falla y la peor distribución de sensores

4.12.2. Viga con dos daños

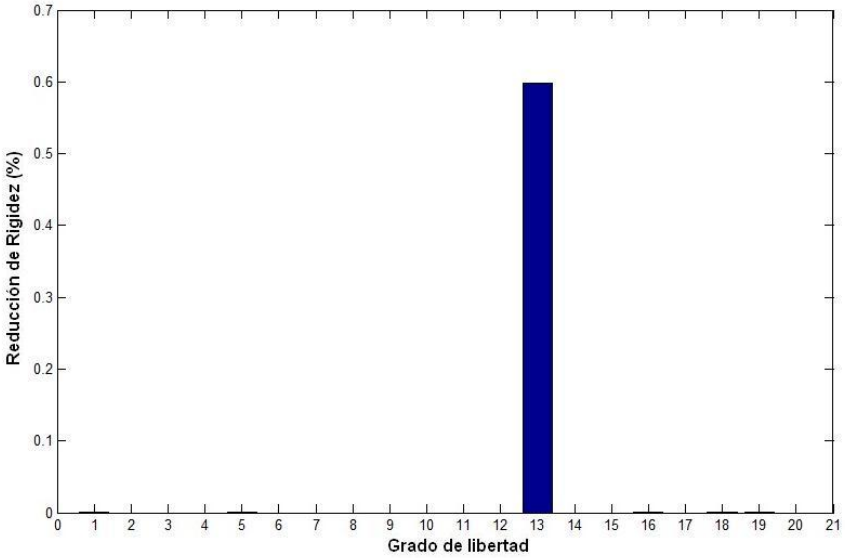

Figura 37: Detección experimental del daño para viga con dos fallas y distribución uniforme de sensores

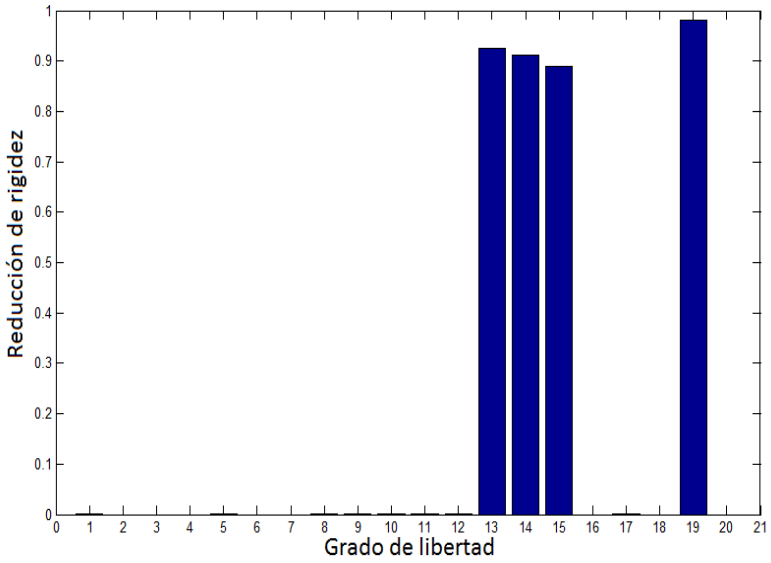

Figura 38: Detección experimental del daño para viga con dos fallas y la peor distribución de sensores

Al comparar los resultados obtenidos con las distintas distribuciones de sensores, se puede observar que en el caso de una viga con un solo daño, la distribución uniforme y la propuesta, son capaces de localizar con exactitud la falla. Sin embargo, tal como se comentó en la sección 4.9, la diferencia radica en la reducción de rigidez encontrada. El modelo propuesto indica que la viga presenta una reducción de rigidez, en el séptimo elemento, de un 38%, versus una reducción del 32% para el caso de la distribución equidistante.

En el caso de la viga que presenta dos fallas, el daño menor no fue detectado por la información dinámica adquirida por la distribución uniforme de sensores, como sí lo hizo la distribución propuesta por el modelo. Si bien, el daño menor de la viga, en este último caso, no fue localizado a la perfección, los resultados obtenidos se consideran aceptables, en cuanto presentan una mejor eficiencia del modelo en comparación con las demás distribuciones.

La peor distribución de sensores, muestra gran cantidad de falsos positivos en la detección de daño.

4.13. Una última prueba

Luego de la comparación realizada en la sección anterior, se decide probar el modelo con una última viga.

Existe un tercera barra que también presenta dos daños, pero en este caso estos se encuentran ubicados en los elementos 8 y 17 y tienen un largo de l_{c1}=0.006m y l_{c2}=0.017m respectivamente. Se comparan solo las distribuciones uniforme y la entregada por el modelo. En este caso, no se considera la peor distribución de sensores.

4.13.1. Distribución uniforme

Algunas de las anti-resonancias adquiridas de esta viga para esta distribución se ilustran en la figura 39. Al igual que en las figuras 31 y 32, la curva azul representa los datos analíticos y la roja los datos experimentales. El algoritmo de detección de daño arroja los resultados ilustrados en la figura 40.

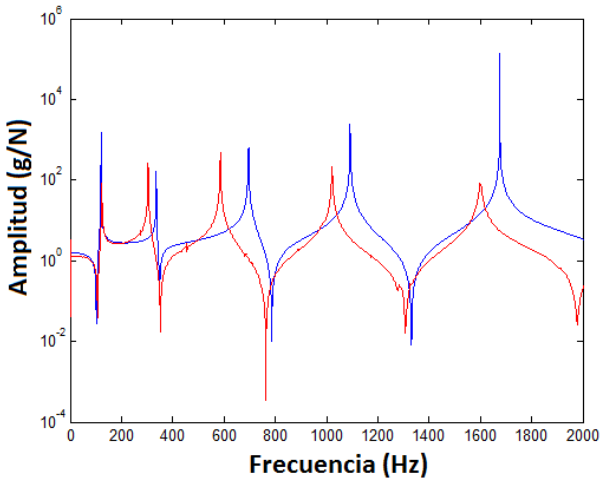

Figura 39: FRF nodo 4, última viga para la distribución uniforme

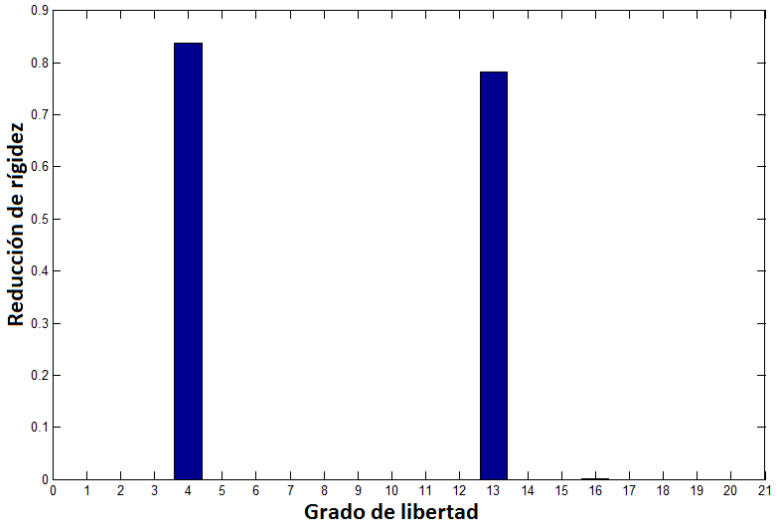

Figura 40: Detección daño última viga para la distribución uniforme

4.13.2. Distribución entregada por el modelo

La figura 41 ilustra la Función de Respuesta en Frecuencia para la última viga, monitoreando la estructura con la distribución de sensores entregada por el modelo. La curva azul representa los datos analíticos y la roja los datos experimentales. El algoritmo de detección de daño arroja los resultados ilustrados en la figura 42.

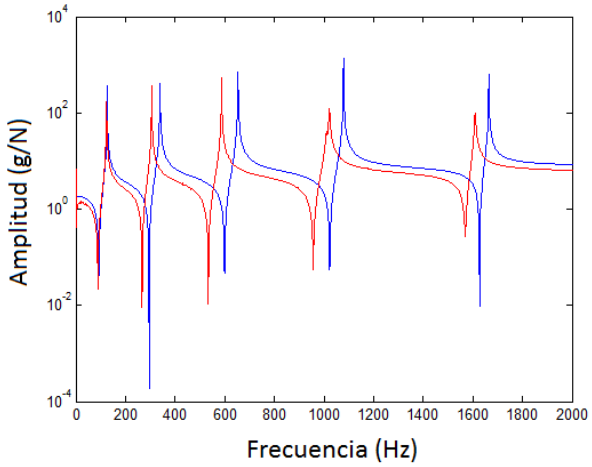

Figura 41: FRF nodo 2, última viga para la distribución entregada por el modelo

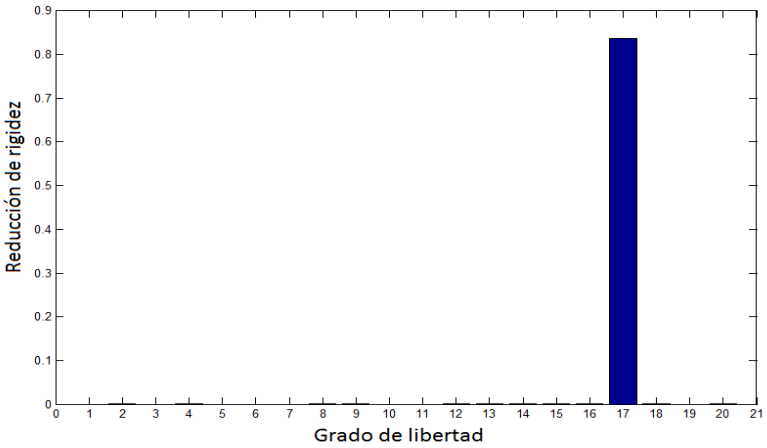

Figura 42: Detección daño última viga para la distribución entregada por el modelo

Se puede notar la diferencia en la detección de daños, cuando se utilizan la distribución uniforme y la entregada por el modelo. En el primero caso, se detectan dos daños falsos y nula detección de los daños reales.

Para el caso de la distribución entregada por el modelo, a diferencia de lo que sucedía para las dos primeras vigas, esta tercera búsqueda de falla no es totalmente exitosa. Solo se encuentra el daño mayor. Si se toma en cuenta que el daño menor es idéntico que en el caso anterior, es interesante ver que no se detecta en los dos casos. Es fácil deducir, entonces, que no es el largo del daño lo que realmente influye en la reducción de rigidez de la estructura, si no que es el cambio en el momento de inercia del elemento. Las características del daño (figura 26), influyen directamente en la sección del elemento de viga y por ende, en el factor cúbico del momento de inercia. Es por esta razón, que en esta última viga, la variación del momento de inercia, en el elemento donde se ubica el daño mayor, es demasiado grande en comparación con el efecto producido por el daño menor, lo que provoca que solo sea detectable el primero.

5. Discusiones

Una vez calculadas las anti resonancias de la viga en estudio y su sensibilidad al daño, el alumno procede a crear la función objetivo la cual será aquella a optimizar utilizando el método de Algoritmos Genéticos Paralelos. Para esto fue necesario definir los distintos parámetros del método de optimización.

Primeramente, se decide fijar el método de selección y mutación y utilizar distintos métodos de recombinación además de fijar la mejor combinación de parámetros de los operadores y el tamaño de la población. El método de selección fue escogido entre tres métodos distintos los cuales fueron comparados según la mejor convergencia lograda por cada uno de ellos. Lo mismo ocurre con la combinación de parámetros de los operadores y con el tamaño de la población. En el caso del método de mutación, se utiliza el único que se adecúa a este estudio en particular. En resumen, los operadores utilizados en los AGP son:

- Método de selección: Ruleta
- Operador de recombinación: Recombinación Simple, Recombinación de dos puntos y recombinación uniforme cada una con probabilidad 0.95 de que ocurra y 0.08 de escoger al individuo más apto.
- Operador de Mutación: Mutación Binaria, con probabilidad 0.04.
- Tamaño de población: 30 individuos.

La función objetivo, como se explica en la sección 4.5., se crea tomando en cuenta tres criterios. El primero, es la sensibilidad de las anti-resonancias, al comparar entre una viga sin daño y otra con daño. El segundo criterio se basa en la comparación de anti-resonancias de viga que poseen daños en distintos elementos y el tercer criterio tiene que ver con el número de sensores a utilizar en el monitoreo de la viga y es el que permite minimizar la cantidad de acelerómetros a utilizar.

Definidos los parámetros de los AGP y la función objetivo, el modelo creado arroja como resultado que, para obtener una cantidad de información dinámica de la viga adecuada, para la detección de daño, se deben utilizar 7 sensores los cuales se distribuyen según lo detallado en la sección 4.7.

En la sección 3 se explica que la verificación experimental, se realiza con un algoritmo de detección de daño disponible. Aquel, también es un proceso de optimización que utiliza una función objetivo y algoritmos genéticos paralelos, desarrollados por Viviana Meruane y que se detallan en la sección 4.8 respectivamente.

Con esto, y utilizando tres vigas distintas, una con un daño y las otras dos con dos daños, se realiza la verificación experimental en el Laboratorio de Sólidos Mecesup, y los resultados obtenidos son los siguientes:

Para la viga que presentaba solo un daño, la distribución de sensores entregada por este trabajo y el algoritmo de detección de daño, fueron capaces de encontrar con exactitud la ubicación del daño. Lo mismo ocurre para una de las vigas que presentaban dos daños.

Para la tercera viga, que también presentaba dos daños, en el cual el daño mayor era aún más grande que el daño mayor de la segunda viga, se utilizaron la distribución uniforme y aquella entregada por el modelo para la detección del daño. Con la distribución uniforme no fue posible detectar las dos fallas si no que se detectaron dos daños falsos. En cambio, para la distribución entregada por el modelo de AG, solo se detectó una de las fallas: el mayor. Esto se explica, ya que la magnitud de un daño, influye directamente en el momento de inercia, en particular, en el factor cúbico del mismo. Es por eso, que mientras mayor sea el daño en una viga, su influencia es aún mayor en comparación con la preponderancia del daño menor.

Tomando en cuenta esto, y a pesar de que en esta última viga no se pudieron detectar ambos daños, se considera que la metodología creada por el alumno es una mejor alternativa a las otras distribuciones utilizadas, ya que es la que presenta los mejores resultados para las tres vigas en estudio.

6. Conclusiones

El objetivo general de este Trabajo de Título desarrollado por el alumno era crear una metodología para ubicar y minimizar el número de sensores de monitoreo en una estructura tipo viga utilizando las frecuencias de anti-resonancia de la misma a través del método de algoritmos genéticos. Si bien, lo ideal hubiera sido detectar todos los daños de todas las vigas con las cuales se trabajó, se considera que los resultados obtenidos son más que aceptables, en cuanto, presentan una muy buena alternativa al momento de monitorear la estructura en comparación con los métodos que se utilizan hoy en día presentes en la literatura existente. En particular, se ha logrado no solo crear una metodología de distribución de sensores, sino que también se minimiza el número de acelerómetros a utilizar, lo cual es un gran avance en el mundo de la ingeniería, ya que hasta el día de hoy, no existe en la literatura, trabajos desarrollados que hayan logrado aquello.

Finalmente, si bien se logra a cabalidad cumplir con los objetivos definidos, lo más importante de todo, a juicio modesto del alumno, es el aporte que este trabajo significa en la ingeniería, en particular, en el ámbito del monitoreo de las vibraciones mecánicas de estructuras y en su alto potencial en la seguridad de los ciudadanos del planeta.

Se espera que este trabajo, sea un punta pie inicial para un próspero desarrollo de la ingeniería actual.

7. Referencias

[1] Meruane V. ”Dinámica Estructural”, ME-706”, Semestre 2011-02. Introducción, Sistemas con dos grados de Libertad [En línea]
< http://viviana.meruane.com/publications.html > [Consulta: Abril 2012]

[2] V. Meruane, W. Heylen, "Structural damage assessment with antiresonances versus mode shapes using parallel genetic algorithms", Structural Control & Health Monitoring, Artículo en prensa, DOI:10.1002/stc.401, 2010.

[3] Dilena M, Morassi A. The use of antiresonances for crack detection in beams. Journal of Sound and Vibration 2004; 276(1–2):195–214.
DOI:10.1016/j.jsv.2003.07.021.

[4] Carnicero A. “Introducción al Método de Elemento Finitos” [En línea]
<http://www.profesores.frc.utn.edu.ar/industrial/sistemasinteligentes/FFlexible/Introduccion_al_MEF.pdf> [Consulta: Enero 2011]

[5] Meruane V. ”Vibraciones Mecánicas” ME4704”, Semestre 2012-01. Elementos Finitos, Elemento de viga [En línea]
< http://viviana.meruane.com/publications.html > [Consulta: Abril 2012]

[6] Meruane V. “Damage detection based upon experimental and numerical dynamic analysis techniques for mechanical structures”. Tesis (Doctor en Ciencia mención Ing. Mecánica). Lovaina, Bélgica. Katholieke Universiteit Leuven, Facultad de Ciencias Aplicadas, 2010. 14h.

[7] M. Meo, G. Zumpano. On the optimal sensor placement techniques for a bridge structure. International Journal of Engineering Structures 27 (2005) 1488-1497

[8] Meruane V. ”Dinámica Estructural”, ME-706”, Semestre 2011-02. Medición experimental, Selección de la ubicación de las respuestas [En línea]
< http://viviana.meruane.com/publications.html> [Consulta: Julio 2012]

[9] Alfaro E. Algoritmos genéticos [En línea]
<http://eddyalfaro.galeon.com/geneticos.html > [Consulta: Diciembre 2011]

[10] Meruane V. “Damage detection based upon experimental and numerical dynamic analysis techniques for mechanical structures”. Tesis (Doctor en Ciencia mención Ing. Mecánica). Lovaina, Bélgica. Katholieke Universiteit Leuven, Facultad de Ciencias Aplicadas, 2010. 38h.

[11] M.A. Rao, J. Srinivas, and B.S.N. Murthy. Damage detection in vibrating bodies using genetic algorithms. *Computers and Structures*, 82(11-12): 963–968, 2004.

[12] Meruane V. ”Dinámica Estructural”, ME-706”, Semestre 2011-02. Procesamiento de señales, La transformada de Fourier [En línea]
< http://viviana.meruane.com/publications.html > [Consulta: Septiembre 2012]

[13] Meruane V. ”Dinámica Estructural”, ME-706”, Semestre 2011-02. Procesamiento de señales, Errores y ventanas [En línea]
< http://viviana.meruane.com/publications.html > [Consulta: Septiembre 2012]

[14] Meruane V. ”Dinámica Estructural”, ME-706”, Semestre 2011-02. Correlación numérico-experimental y ajuste de modelos, Métodos iterativos de ajuste de modelo [En línea] < http://viviana.meruane.com/publications.html > [Consulta: Julio 2012]

[15] H.M. Elkamchouchi, M. Wagih, Genetic algorithm operators effect in optimizing the antenna array pattern synthesis, in: Proceedings of the 20th National Radio Science Conference, Cairo, Egypt, 2003.

[16] V. Meruane, W. Heylen, “An hybrid real genetic algorithm to detect structural damage using modal properties”. Mechanical Systems and Signal Processing, 25(2011):1559-1573. DOI: 10.1016/j.ymssp.2010.11.020

[17] Meruane V. “Damage detection based upon experimental and numerical dynamic analysis techniques for mechanical structures”. Tesis (Doctor en Ciencia mención Ing. Mecánica). Lovaina, Bélgica. Katholieke Universiteit Leuven, Facultad de Ciencias Aplicadas, 2010. 112h.

[18] Meruane V. “Damage detection based upon experimental and numerical dynamic analysis techniques for mechanical structures”. Tesis (Doctor en Ciencia mención Ing. Mecánica). Lovaina, Bélgica. Katholieke Universiteit Leuven, Facultad de Ciencias Aplicadas, 2010. 37h.

8. Anexos

Anexo A: Modelo de viga sin daño

```
function Wd=viga(f)
%Datos de la viga
rho=7800;
E=2.1e11;
le=.05;
A=.025*.01;
Iz=.025*(.010)^3/12;

%matriz de masa de un elemento
m=rho*A*le/420*[156     22*le   54     -13*le;
                22*le   4*le^2  13*le  -3*le^2;
                54      13*le   156    -22*le;
               -13*le  -3*le^2 -22*le   4*le^2];

%matriz de rigidez de un elemento
k=E*Iz/le^3*[12     6*le    -12    6*le;
             6*le   4*le^2  -6*le  2*le^2;
            -12    -6*le     12   -6*le;
             6*le   2*le^2  -6*le  4*le^2];
%Ensamble
K=zeros(42,42);
M=zeros(42,42);

for i=1:20
    K((1+2*(i-1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))=K((1+2*(i-
1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))+k;
    M((1+2*(i-1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))=M((1+2*(i-
1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))+m;
end

%Cálculo frecuencias de anti resonancias
all=1:42;

Wd=zeros(7,21);
g=2*f-1;
for i=1:2:42
fixed1=setdiff(all,[g]);
fixed2=setdiff(all,[i]);
Mr2=M(fixed1,fixed2);
Kr2=K(fixed1,fixed2);
W=eig(Kr2,Mr2);
w=sqrt((W))/2/pi;
[w]=sort(w);
w=w(imag(w)==0);
w=w(w<5000);
w=w(w>1);
if length(w)>=7
Wd(:,(i+1)/2)=w(1:7);
else
Wd(1:length(w),(i+1)/2)=w;
end
```

Anexo B: Modelo de viga con daño

```
function Wa=viga_alan(l,f)

u=zeros(20,1);
u(l)=0.2;%Factor de reducción de rigidez
rho=7800;
E=2.1e11;
le=.05;
A=.025*.01;
Iz=.025*(.010)^3/12;

%matriz de masa de un elemento
m=rho*A*le/420*[156     22*le   54     -13*le;
                22*le   4*le^2  13*le  -3*le^2;
                54      13*le   156    -22*le;
               -13*le  -3*le^2 -22*le   4*le^2];

%matriz de rigidez de un elemento
k=E*Iz/le^3*[12     6*le    -12    6*le;
             6*le   4*le^2  -6*le  2*le^2;
            -12    -6*le     12   -6*le;
             6*le   2*le^2  -6*le  4*le^2];
%Ensamble
K=zeros(42,42);
M=zeros(42,42);

for i=1:20
    K((1+2*(i-1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))=K((1+2*(i-
1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))+k*(1-u(i));
    M((1+2*(i-1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))=M((1+2*(i-
1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))+m;
end

%Cálculo frecuencias de anti resonancias
all=1:42;
Wa=zeros(7,21);
g=2*f-1;
for i=1:2:42
fixed1=setdiff(all,[g]);
fixed2=setdiff(all,[i]);
Mr2=M(fixed1,fixed2);
Kr2=K(fixed1,fixed2);
W=eig(Kr2,Mr2);
w=sqrt((W))/2/pi;
[w]=sort(w);
w=w(imag(w)==0);
w=w(w<5000);
w=w(w>1);
if length(w)>=7
Wa(:,(i+1)/2)=w(1:7);
else
Wa(1:length(w),(i+1)/2)=w;
end
end
```

Anexo C: Cálculo sensibilidad anti resonancias (diferencia porcentual)

```
function Wdif=diferencia(f)
close all;
clc;
Wd=viga(f);%Anti resonancias viga sin daño
for l=1:20
    Wa(7*l-6:7*l,:)=viga_alan(l,f); %Anti resonancias viga con daño(20  matrices)
    [Wd,Wa(7*l-6:7*l,:)]=parearv(Wd,Wa(7*l-6:7*l,:));%diferencia procentual
    Wdif(7*l-6:7*l,:)=(abs(Wa(7*l-6:7*l,:)-Wd))./Wd*100;
    for j=1:numel(Wdif)
        if isnan(Wdif(j))==1
            Wdif(j)=0;
        end
        if isinf(Wdif(j))==1
            Wdif(j)=0;
        end

    end
    Wmean(l)=mean(mean(Wdif));
    Wmean=Wmean';
end

end
```

Anexo D: Función objetivo distribución de sensores

```
function [u, val, N, PP]=objetivon(u,options)

n=sum(u);
k=1;
for i=1:21
    if u(i)==1;
        a(k)=i;
        k=k+1;
    end
end

%          f=1;
%          Wdif=diferencia(f);
%         save Wdif Wdif
         load Wdif

M=zeros(20,1);
for j=1:20
    for i=j*7-6:j*7
        for k=1:n
    if Wdif(i,k)>2
        M(j)=1;
    end
        end
    end
end

N=sum(M);


P=zeros(20,1);
prom=zeros(20,1);
% vector=[a b c d];
% for k=1:4
for j=1:20
    for i=1:20
        for k=1:numel(a)
            mmac(k)=mac(Wdif(j*7-6:j*7,a(k)),Wdif(i*7-6:i*7,a(k)));
        end
        P(i)=min(mmac);
    end
    prom(j)=sum(P)/20;
end

PP=sum(prom)/20;

alpha=0.01;

val=0.5*N/20 + 0.5*(1-PP)-n*alpha;
```

Anexo E: Selección probabilidad de recombinación y mutación para AGP

```
clear all;
close all;

ngenes=5;
lb=1;
ub=21;
bounds=ones(ngenes,2); bounds(:,1)=lb; bounds(:,2)=ub;

op1=1e-3;
op2=1;
op3=1;

tp=20;
pc=0.95;
pm=0.05;

ferror='objetivo1';

ng=400;
v0=1;
eps=0.001;

path(path,'C:\gaot')

initPop=randi([1 21],[tp 5]);
for i=1:tp
    [u val]=objetivo1(initPop(i,:),[]);
    initPop(i,6)=val;
end
for pc=0.7:0.05:0.95 %variacion de probabilidad de racombinación
    for pm=0.01:0.01:0.05 %variacion de probabilidad de mutación

        j=1;
        nx=tp*pc;
        nmu=round(tp*ngenes*pm);
[x endPop bPop traceInfo] = ga(bounds,ferror,[],initPop,[op1 op2
op3],'optMaxGenTerm',[ng v0
eps],'roulette',[],['uniXover'],[nx],['realMutation'],[nmu]);
j=j+1;

estructura(j,1).x=x;
estructura(j,1).traceInfo=traceInfo;
    end

end
save estructura estructura;
```

Anexo F: Selección tamaño de población para AGP

```
clear all;
close all;

ngenes=5;
lb=1;
ub=21;
bounds=ones(ngenes,2); bounds(:,1)=lb; bounds(:,2)=ub;

op1=1e-3;
op2=1;
op3=1;


tpv=[20:10:100]; %variación del tamaño de la población
pc=0.95;
pm=0.04;


ferror='objetivo1';

ng=100;
v0=1;
eps=0.001;

path(path,'C:\gaot')

for k=1:length(tpv)
tp=tpv(k);

for j=1:5
nx=tp*pc;
nmu=round(tp*ngenes*pm);

initPop=randi([1 21],[tp 5]);
for i=1:tp
    [u val]=objetivo1(initPop(i,:),[]);
    initPop(i,6)=val;
end

[x endPop bPop traceInfo] = ga(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'roulette',[0.08],['uniXover'],[nx],['realMutation'],[nmu]);

estructura(j,k).x=x;
estructura(j,k).traceInfo=traceInfo;
end
save estructura estructura;
end
```

Anexo G: Algoritmos Genéticos Paralelos para la distribución de sensores

Algoritmo paralelo 1

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%Parallel1, runs the GA algorithm for the first population

%%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
% load tp; %population size
tp=20;
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population
initPop=randi([0 1],[tp 21]); % poblacion inicial
for i=1:tp
    [u val]=objetivon(initPop(i,:),[]);
    initPop(i,22)=val;
end

save initPop initPop;
Trace1=[0 0 0 0 0];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=1;

it1=1;
save it1 it1 -v4;

ind=1;
b=1;

a=1;

while a>0.001
    %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end

%run the GA algorithm for population 1
[x endPop bPop traceInfo] = ga(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'roulette',[],['simpleXover'],[nx],['binaryMutation'],[nmu]);

save endPop endPop;

%increase iteration number
it1=it1+1;
save it1 it1 -v4;

%Load iteration from other populations and wait until they are the same as
%current
it2=0;
it3=0;
it4=0;
while (it1<it1)||(it3<it1)||(it4<it1)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
```

```
    end
end
pause(5);
end

%read best individual of other populations
test=0;
while test<5
    try load best1;
        test=test+1;
    catch
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);

%Save information from optimization algorithm
Trace1=[Trace1;traceInfo];
save Trace1 Trace1 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best2;
initPop(Jmin(2),:)=best4;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;


end

%update damage detection results
uw(ind,:)=best1(1:ngenes);
```

```
 %save damage detection results
save uw uw -v4;
```

Algoritmo paralelo 2

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%Parallel2, runs the GA algorithm for the second population

%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
% load tp; %population size
tp=40;
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population
initPop=randi([0 1],[tp 21]); % poblacion inicial
for i=1:tp
    [u val]=objetivon(initPop(i,:),[]);
    initPop(i,22)=val;
end

Trace2=[0 0 0 0 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
a=1;

it2=1;
save it2 it2 -v4;

ind=1;

a=1;

while a>0.001
  %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end

%run the GA algorithm for population 2
[x endPop bPop traceInfo] = ga2(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'roulette',[],['uniXover'],[nx],['binaryMutation'],[nmu]);

%increase iteration number
it2=it2+1;
save it2 it2 -v4;

%Load iteration from other populations and wait until they are the same as
%current
it1=0;
it3=0;
it4=0;
while (it1<it2)||(it3<it2)||(it4<it2)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
    end
end
pause(5);
end
```

```
%read best individual of other populations
test=0;
while test<5
    try load best1;
        test=test+1;
    catch
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);

%Save information from optimization algorithm
Trace2=[Trace2;traceInfo];
save Trace2 Trace2 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best1;
initPop(Jmin(2),:)=best3;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;


end
```

Algoritmo paralelo 3

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%Parallel3, runs the GA algorithm for the third population

%%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
% load tp; %population size
tp=60;
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population
initPop=randi([0 1],[tp 21]); % poblacion inicial
for i=1:tp
    [u val]=objetivon(initPop(i,:),[]);
    initPop(i,22)=val;
end

Trace3=[0 0 0 0 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=1;

it3=1;
save it3 it3 -v4;
```

```
ind=1;
a=1;

while a>0.001
    %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end

%run the GA algorithm for population 3
[x endPop bPop traceInfo] = ga3(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'roulette',[],['multipleXover'],[nx],['binaryMutation'],[nmu]);

%increase iteration number
it3=it3+1;
save it3 it3 -v4;

%Load iteration from other populations and wait until they are the same as
%current
it1=0;
it2=0;
it4=0;
while (it1<it3)||(it2<it3)||(it4<it3)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
    end
end
pause(5);
end

%read best individual of other populations
test=0;
while test<5
    try load best1;
        test=test+1;
    catch
```

```
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);

%Save information from optimization algorithm
Trace3=[Trace3;traceInfo];
save Trace3 Trace3 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best2;
initPop(Jmin(2),:)=best4;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;


end
```

Algoritmo paralelo 4

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%Parallel4, runs the GA algorithm for the fourth population

%%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
% load tp; %population size
tp=80;
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population
initPop=randi([0 1],[tp 21]); % poblacion inicial
for i=1:tp
    [u val]=objetivon(initPop(i,:),[]);
    initPop(i,22)=val;
end

Trace4=[0 0 0 0 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=1;

it4=1;
save it4 it4 -v4;
```

```
ind=1;

a=1;

while a>0.001
    %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end

%run the GA algorithm for population 4
[x endPop bPop traceInfo] = ga4(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'roulette',[],['uniXover'],[nx],['binaryMutation'],[nmu]);

%increase iteration number
it4=it4+1;
save it4 it4 -v4;

%Load iteration from other populations and wait until they are the same as
%current
it1=0;
it2=0;
it3=0;
while (it1<it4)||(it2<it4)||(it3<it4)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
    end
end
pause(5);
end

%read best individual of other populations
test=0;
while test<5
    try load best1;
        test=test+1;
```

```
    catch
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);

%Save information from optimization algorithm
Trace4=[Trace4;traceInfo];
save Trace4 Trace4 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best3;
initPop(Jmin(2),:)=best1;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;


end
```

Función que relaciona los algoritmos paralelos

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%this function loads and defines all the necesary data to run the damage
%detection algorithm

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%GENETIC ALGORITHM%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
GApath0='C:\gaot'; %Path of Genetic Algorithms in current PC

save GApath0 GApath0 -v4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%Set problem parameters%%%%%%%%%%%%%%%%%%%%%%%%
ngenes=21; %number of genes (numero de parametros)
v0=1; %valor optimo de la funcion (maximo valor en la función error)
tp=30; %population size (tamaño de la población)

%limite de variables
lb=0; %variables lower limit
ub=1; %variable upper limit

save ngenes ngenes -v4; save v0 v0 -v4; save tp tp -v4; save lb lb -v4; save
ub ub -v4;
%%%%%%%%%%%%%%%%%Set GA parameters%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pc=0.95; %croosover probability
pm=0.04; %mutation probability
ng=20; %number of generations before stoping

save pc pc -v4; save pm pm -v4; save ng ng -v4;

%%%%%%%%%%%%%%%%%%%%%Error function%%%%%%%%%%%%%%%%
ferror='objetivon';

save ferror ferror -v4;


%run parallel algorithm
matlabpool open 4 %prepares matlab to run in 4 parallel procesors
j1 = batch('parallel1', 'matlabpool', 0);
j2 = batch('parallel2', 'matlabpool', 0);
j3 = batch('parallel3', 'matlabpool', 0);
j4 = batch('parallel4', 'matlabpool', 0);

matlabpool close
```

Anexo H: Función objetivo detección de daño

```
function [u, val] = error_antiresonances(u,options)

%-----------------------------------------------------------------
% PURPOSE
% Compute fitness value of an individual using resonaces and antiresonances
% residuals
%
% INPUT:
%    u = stiffness reduction factors
%    options = [] default
%
% OUTPUT:
%    u = stiffness reduction factors
%    val = fitness value
%-----------------------------------------------------------------

%Modelo en elementos finitos de una viga
nn=21; %numero de nodos
bl=1; %largo de la viga en metros

load kmv;
% kmv=zeros(1,4);
test=0;
while(test==0)
try
    %creacion de los nodos
    model.Node=[];
    for i=1:nn
        model.Node=[model.Node; i 0 0 0 (i-1)*bl/(nn-1) 0 0];
    end

    %creacion de los elementos
    dir=[0 0 1]; %normal a los elementos
    model.Elt=[Inf abs('beam1') 0];
    for i=1:(nn-1)
        model.Elt=[model.Elt; i i+1 i 1 dir];
    end

    %definicion del material
    model.pl=[];
    for i=1:(nn-1)
    ID=i; %id del material
    E=2.1e11*(1+kmv(1))*(1-u(i)); %modulo de Young (%%%%)
    nu=0.3; %coeficiente de Poisson
    rho=7800*(1+kmv(2)); %densidad
    model.pl=[model.pl; ID fe_mat('m_elastic','SI',1) E nu rho];
    end

    %definicion de las propiedades del elemento
    a1=0.01*(1+kmv(3));
    a2=0.025*(1+kmv(4));
    A=a1*a2; %m^2 area
    Iy=(a1*a2^3)/12; %m^4 inercia sección
    Iz=(a2*a1^3)/12; %m^4 inercia sección
    Ix=Iy+Iz;
    model.il=[1 fe_mat('p_beam','SI',1) Ix Iy Iz A];
```

```
    %grados de libertad fijos
%     model0=fe_case(model,'FixDof','MovimientoEnUnPlano',[.01 .02 .04 .06]);

    %construir matrices de rigidez y masa
    [m0,k0,mdof0] = fe_mk(model,'Options',[]);
    test=1;
catch
    test=0;
end
end
m0=full(m0);
k0=full(k0);
    N=length(m0);
    todos=1:1:N;
    delete=sort([1:6:N 2:6:N 4:6:N 6:6:N]);
    activos=setdiff(todos,delete);
    m0=m0(activos,activos);
    k0=k0(activos,activos);

excdof=[1 1 1 1 1 1 1]*2-1; %excitacion
mesdof=[2 9 11 13 14 15 19]*2-1; %respuesta

% datos experimentales
anti=[5 3   2   2   2   2   1
88.75   205.9   280.3   385.9   217.8   57.81   412.2
267.2   611.6   902.5   1599    1066    634.7   0
534.1   1504    0   0   0   0   0
956.9   0   0   0   0   0   0
1569    0   0   0   0   0   0]';

%Compute antiresonances residual
par2=[];
for k=1:length(anti(:,1))
%     for k=1:4
if anti(k,1)> 0
i=excdof(k); % Exitation degreess of freedom
j=mesdof(k); % Measured degrees of freedom
%pair numerical and experimental antiresonances
try [par]=parear(anti(k,[2:(anti(k,1)+1)]),k0,m0,i,j);
catch
    par=[1 1];
end
par2=[par2;par];
%compute residual
    end
end

% load par0;
Wa=sum(abs((par2(:,2).^2./par2(:,1).^2+1e-8)-par0(:,2).^2./par0(:,1).^2))


load Wa0;
ngenes=20;
Wf=0.05;
PHI=Wf*sum(u(1:ngenes)>=0.001)+Wf*sum(u(1:ngenes));
```

```
val=20-Wa/Wa0-PHI;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if options==1e5
    Wa0=Wa;
    save Wa0 Wa0;
end


clear all;
close all;

ngenes=5;
lb=1; %variables lower limit
ub=21; %variable upper limit
bounds=ones(ngenes,2); bounds(:,1)=lb; bounds(:,2)=ub;

op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=1; %1 diplay information at each generation, 0 doesnt diplay information

tp=20;
pc=0.95; %probabilidad recombinación
pm=0.05; %probabilidad mutacion


ferror='objetivo1'; %nombre de función objetivo

ng=100; %numero de generaciones sin mejoras antes de terminar
v0=1; %mejor valor de la función de objetivo
eps=0.001; %error aceptado

path(path,'C:\gaot')

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

for j=1:5
initPop=randi([1 21],[tp 5]); % poblacion inicial
for i=1:tp
    [u val]=objetivo1(initPop(i,:),[]);
    initPop(i,6)=val;
end

[x endPop bPop traceInfo] = ga(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'normGeomSelect',[0.08],['simpleXover'],[nx],['realMutation'],[nmu]); %AG
con selección geométrica normalizada

estructura(j,1).x=x;
estructura(j,1).traceInfo=traceInfo;
A=estructura(j,1).traceInfo;
plot(A(:,2))
hold on;
```

```
[x endPop bPop traceInfo] = ga(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'roulette',[],['simpleXover'],[nx],['realMutation'],[nmu]); %AG con
selección método de ruleta

estructura(j,2).x=x;
estructura(j,2).traceInfo=traceInfo;
B=estructura(j,2).traceInfo;
plot(B(:,2))
hold on;

[x endPop bPop traceInfo] = ga(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0
eps],'tournSelect',[2],['simpleXover'],[nx],['realMutation'],[nmu]); %AG con
selección torneo

estructura(j,3).x=x;
estructura(j,3).traceInfo=traceInfo;
C=estructura(j,3).traceInfo;
plot(C(:,2))

end
save estructura estructura;
```

Anexo I: Algoritmos Genéticos Paralelos para detección de daño

Algoritmo paralelo 1

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%Parallel1, runs the GA algorithm for the first population

%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
load tp; %population size
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population

initPop=lb + (ub-lb).*rand([tp 21]); % poblacion inicial
for i=1:tp
    [u val]=error_antiresonances(initPop(i,:),[]);
    initPop(i,ngenes+1)=val;
end
% initPop(1,:)=[zeros(1,ngenes) 19];

% initPop=initializega(tp,bounds,ferror,1);
% initPop(1,:)=[zeros(1,ngenes) (v0-2)];
save initPop initPop;
Trace1=[0 0 0 0 0];
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=1;

it1=1;
save it1 it1 -v4;

ind=1;
b=1;

a=1;

while a>0.001
    %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end

%run the GA algorithm for population 1
[x endPop bPop traceInfo] = ga(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0 eps],...
'normGeomSelect',[0.06],['arithXover'],[nx 0.5],['boundaryMutation
unifMutation'],[floor(nmu/2);ceil(nmu/2)],1);
save endPop endPop;

%increase iteration number
it1=it1+1;
save it1 it1 -v4;

%Load iteration from other populations and wait until they are the same as
%current
it2=0;
it3=0;
it4=0;
while (it2<it1)||(it3<it1)||(it4<it1)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
    end
```

```
end
pause(5);
end

%read best individual of other populations
test=0;
while test<5
    try load best1;
        test=test+1;
    catch
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);

%Save information from optimization algorithm
Trace1=[Trace1;traceInfo];
save Trace1 Trace1 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best2;
initPop(Jmin(2),:)=best4;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;


end

%update damage detection results
uw(ind,:)=best1(1:ngenes);
```

```
%save damage detection results
save uw uw -v4;
```

Algoritmo paralelo 2

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%Parallel2, runs the GA algorithm for the second population

%%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
load tp; %population size
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population
initPop=lb + (ub-lb).*rand([tp 21]); % poblacion inicial
for i=1:tp
    [u val]=error_antiresonances(initPop(i,:),[]);
    initPop(i,ngenes+1)=val;
end
% initPop(1,:)=[zeros(1,ngenes) 19];

Trace2=[0 0 0 0 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=1;
```

```
it2=1;
save it2 it2 -v4;

ind=1;

a=1;

while a>0.001
  %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end

%run the GA algorithm for population 2
[x endPop bPop traceInfo] = ga2(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0 eps],...
'normGeomSelect',[0.06],['heuristicXover'],[nx 10],['boundaryMutation
unifMutation'],[floor(nmu/2);ceil(nmu/2)],2);

%increase iteration number
it2=it2+1;
save it2 it2 -v4;

%Load iteration from other populations and wait until they are the same as
%current
it1=0;
it3=0;
it4=0;
while (it1<it2)||(it3<it2)||(it4<it2)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
    end
end
pause(5);
end

%read best individual of other populations
test=0;
```

```
while test<5
    try load best1;
        test=test+1;
    catch
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);

%Save information from optimization algorithm
Trace2=[Trace2;traceInfo];
save Trace2 Trace2 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best1;
initPop(Jmin(2),:)=best3;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;


end
```

Algoritmo paralelo 3

```
clear all;
close all;
warning off all;
more off
echo off
clc;
```

```
%Parallel3, runs the GA algorithm for the third population

%%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
load tp; %population size
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population
initPop=lb + (ub-lb).*rand([tp 21]); % poblacion inicial
for i=1:tp
    [u val]=error_antiresonances(initPop(i,:),[]);
    initPop(i,ngenes+1)=val;
end
% initPop(1,:)=[zeros(1,ngenes) 19];

Trace3=[0 0 0 0 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=1;

it3=1;
save it3 it3 -v4;

ind=1;
a=1;

while a>0.001
    %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end
```

```
%run the GA algorithm for population 3
[x endPop bPop traceInfo] = ga3(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0 eps],...
'normGeomSelect',[0.06],['uniXover'],[nx 0.5],['boundaryMutation
unifMutation'],[floor(nmu/2);ceil(nmu/2)],3);

%increase iteration number
it3=it3+1;
save it3 it3 -v4;

%Load iteration from other populations and wait until they are the same as
%current
it1=0;
it2=0;
it4=0;
while (it1<it3)||(it2<it3)||(it4<it3)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
    end
end
pause(5);
end

%read best individual of other populations
test=0;
while test<5
    try load best1;
        test=test+1;
    catch
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
```

```
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);

%Save information from optimization algorithm
Trace3=[Trace3;traceInfo];
save Trace3 Trace3 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best2;
initPop(Jmin(2),:)=best4;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;


end
```

Algoritmo paralelo 4

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%Parallel4, runs the GA algorithm for the fourth population

%%%%%%%%%%%%%%Include path of Genetic Algorithm%%%%%%%%%%%%%%
load GApath0;
path(path,GApath0);

%%%%%%%%%%%%%Load data%%%%%%%%%%%%%%%%%%%%%%
load ngenes; %number of genes
load v0; %optimum value objective function
load tp; %population size
load pc; %crossover probability
load pm; %mutation probability
load ng; %number of generations before stoping
```

```
load ferror; %error function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load lb;
load ub;
bounds=ones(ngenes,2);
bounds(:,1)=lb; bounds(:,2)=ub;
op1=1e-3; %precision
op2=1; %1 real-coded, 0 binary
op3=0; %1 diplay information at each generation, 0 doesnt diplay information
eps=0.001; %acepted error

nx=tp*pc; %number of xover at each generation
nmu=round(tp*ngenes*pm); %number of mutations at each generation

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initial population
initPop=lb + (ub-lb).*rand([tp 21]); % poblacion inicial
for i=1:tp
    [u val]=error_antiresonances(initPop(i,:),[]);
    initPop(i,ngenes+1)=val;
end
% initPop(1,:)=[zeros(1,ngenes) 19];

Trace4=[0 0 0 0 0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%FIRST
WF=0%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a=1;

it4=1;
save it4 it4 -v4;

ind=1;

a=1;

while a>0.001
    %evaluate fitness of entire population
for i=1:tp
    eval(['[ initPop(i,:) initPop(i,ngenes+1)]=' ferror
'(initPop(i,:),[]);']);
end

%run the GA algorithm for population 4
[x endPop bPop traceInfo] = ga4(bounds,ferror,[],initPop,[op1 op2
op3],'noimpTerm',[ng v0 eps],...
'normGeomSelect',[0.06],['BLXXover'],[nx 0.5],['boundaryMutation
unifMutation'],[floor(nmu/2);ceil(nmu/2)],4);

%increase iteration number
it4=it4+1;
save it4 it4 -v4;
```

```
%Load iteration from other populations and wait until they are the same as
%current
it1=0;
it2=0;
it3=0;
while (it1<it4)||(it2<it4)||(it3<it4)
test=0;
while test<5
    try load it1;
        test=test+1;
    catch
        pause(1);
    end
    try load it2;
        test=test+1;
    catch
        pause(1);
    end
    try load it3;
        test=test+1;
    catch
        pause(1);
    end
    try load it4;
        test=test+1;
    catch
        pause(1);
    end
end
pause(5);
end

%read best individual of other populations
test=0;
while test<5
    try load best1;
        test=test+1;
    catch
        pause(1);
    end
    try load best2;
        test=test+1;
    catch
        pause(1);
    end
    try load best3;
        test=test+1;
    catch
        pause(1);
    end
    try load best4;
        test=test+1;
    catch
        pause(1);
    end
end

%Compare the best individuals of others population with best
%individual of first population
```

```
a=norm(best1(ngenes+1)-best2(ngenes+1))+norm(best1(ngenes+1)-
best3(ngenes+1))+norm(best1(ngenes+1)-best4(ngenes+1));
pause(15);
% a=1e-7;
%Save information from optimization algorithm
Trace4=[Trace4;traceInfo];
save Trace4 Trace4 -v4;

%Find worst individual in final population
[Imin,Jmin]=sort(endPop(:,(ngenes+1)));

%Set new population equal to current population
initPop=endPop;
%Include migrating individuals in new population
initPop(Jmin(1),:)=best3;
initPop(Jmin(2),:)=best1;
%Include best individual of current population in new population (elitism
strategy)
initPop(Jmin(3),:)=x;
```

Función que relaciona los algoritmos paralelos

```
clear all;
close all;
warning off all;
more off
echo off
clc;

%this function loads and defines all the necesary data to run the damage
%detection algorithm

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%GENETIC ALGORITHM%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
GApath0='C:\gaot'; %Path of Genetic Algorithms in current PC

save GApath0 GApath0 -v4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%Set problem parameters%%%%%%%%%%%%%%%%%%%%%%%%
ngenes=20; %number of genes (numero de parametros)
v0=20; %valor optimo de la funcion (maximo valor en la función error)
tp=40; %population size (tamaño de la población)

%limite de variables
lb=0; %variables lower limit
ub=0.99; %variable upper limit

save ngenes ngenes -v4; save v0 v0 -v4; save tp tp -v4; save lb lb -v4; save
ub ub -v4;
%%%%%%%%%%%%%%%%Set GA parameters%%%%%%%%%%%%%%%%%%%%%%%%%%%%
pc=0.8; %croosover probability
pm=0.02; %mutation probability
ng=20; %number of generations before stoping

save pc pc -v4; save pm pm -v4; save ng ng -v4;
```

```
%%%%%%%%%%%%%%%%%%%%Error function%%%%%%%%%%%%%%%%
ferror='error_antiresonances';

save ferror ferror -v4;

[u, val] = error_antiresonances2(zeros(20,1),[1e5]);

%run parallel algorithm
matlabpool open 4 %prepares matlab to run in 4 parallel procesors
j1 = batch('parallel1', 'matlabpool', 0);
j2 = batch('parallel2', 'matlabpool', 0);
j3 = batch('parallel3', 'matlabpool', 0);
j4 = batch('parallel4', 'matlabpool', 0);

matlabpool close
```

Anexo J: Obtención FRF's

```
clear all;
close all;

%function [Wr,Wa]=vigal(Beta)
% function Wd=viga(f)
%Datos
load km
rho=(1+km(2))*7800;
E=(1+km(1))*2.1e11;
a1=0.01*(1+km(3));
a2=0.025*(1+km(4));
A=a1*a2;%-.036*.036;
Iz=a1*(a2)^3/12;%-.036*(.036)^3/12;
le=.05;

u=zeros(20,1);
u(8)=0.2; %Ubicación del (los) daño(s) en la viga
u(17)=0.9;

%matriz de masa de un elemento
m=rho*A*le/420*[156     22*le   54     -13*le;
                22*le   4*le^2  13*le  -3*le^2;
                54      13*le   156    -22*le;
               -13*le  -3*le^2 -22*le   4*le^2];

%matriz de rigidez de un elemento
k=E*Iz/le^3*[12     6*le    -12    6*le;
             6*le   4*le^2  -6*le  2*le^2;
            -12    -6*le     12   -6*le;
             6*le   2*le^2  -6*le  4*le^2];
%Ensamble
K=zeros(42,42);
M=zeros(42,42);

for i=1:20
    K((1+2*(i-1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))=K((1+2*(i-
1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))+k*(1-u(i));
    M((1+2*(i-1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))=M((1+2*(i-
1)):(2*(i+1)),(1+2*(i-1)):(2*(i+1)))+m;
end

f=1;
omega=1:1:3000;

for i=1:3000
    FRF(:,:,i)=(omega(i)*2*pi)^2*inv(K-(omega(i)*2*pi)^2*M);
end
% load FRF;

i=f*2-1;
j=[2 9 11 13]*2-1; %Nodos donde se ubican los sensores


for k=1:4
frf(k,:)=FRF(i,j(k),:);
```

```
end



[UffDataSets, Info, errmsg] =
readuff('C:\Users\mem2\Desktop\Viga\uff\medicionesf\daño2\f1-max\2-9-11-
13\H.uff'); %Ubicación del (los) archivos de los datos experimentales


for i=1:4
FRFe(i,:)=UffDataSets{1,i}.measData;
end
w=UffDataSets{1,1}.x;

for i=1:4
    figure
    semilogy(omega,abs(frf(i,:)))
    hold on
    semilogy(w,abs(FRFe(i,:)),'r')
    xlim([0 2000])
end
```