Optimización de parámetros en algoritmo detector de daños estructurales

Agostina Sigulin

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Escuela de Ingeniería Mecánica

Rosario, Argentina

2023

Optimización de parámetros en algortimo detector de daños estructurales

Agostina Sigulin

Proyecto Final de Ingeniería Mecánica presentado como requisito parcialpara optar al título de:

Ingeniera Mecánica

Director:

Título y nombre del directores:

Ing. Pedro Sismondi – Ing. Guillermo Rodríguez – Ing. Juan Sismondi

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Escuela de Ingeniería Mecánica

Rosario, Argentina

2023

Agradecimientos

Lograr llegar al final de la carrera no hubiera sido posible sin el apoyo incondicional de mi familia. Es a ellos a quienes profundamente agradezco de estar donde estoy. Mamá, papá, hermano, simplemente gracias.

También cabe mencionar que durante todos estos años me acompañaron también grandes personas, mis amigos y amigas de la facultad, con los que compartí las alegrías y los llantos, los días interminables de estudios y que me brindaron unas fuerzas increíbles. A ellos y ellas les digo gracias. Sin ustedes este camino hubiera no hubiera sido tan placentero.

No quiero dejar de lado en este agradecimiento a quien considero mi familia de corazón, quienes me hospedaron durante ocho meses en Francia mientras realizaba el trabajo de este proyecto. Siempre los voy a tener presentes.

Y para terminar quiero dar gracias a la Universidad Nacional de Rosario por dar la oportunidad de el acceso a estudios superiores a todos aquellos y aquellas que lo deseen de manera pública y gratuita.

Resumen

El algoritmo de *PSO - Particle Swram Optimization,* o en español optimización por enjambre de partículas, es un algoritmo complejo que se utiliza para minimizar una función objetiva en particular. En este proyecto se utilizó para minimizar la discrepancia que hay entre los datos de vibración identificados por un ensayo modal y un cálculo a partir de un modelo analítico con la finalidad de encontrar algún tipo de daño en una viga empotrada.

Para lograr distinguir si hay algún cambio en la rigidez de la estructura se debe seleccionar correctamente los valores de los parámetros del algoritmo PSO. Como las posibles combinaciones de valores son extensas, llegando a existir hasta 20.000 configuraciones posibles, se estudió la aplicación de dos metodologías para testear el modelo cumputacional PSO. Por un lado, el método F-Race, que se base en probar las configuraciones en diversos escenarios de daño en un tiempo delimitado, comparando entre ellas su rendimiento. Por otro lado, se utilizó el método *DOE – Design of Experiment* el cual consiste en generar un gráfico que aporte información de la relación de cuasa y efecto de varias el valor de cada parámetro del modelo PSO.

Luego de desarrollar los algoritmos en MATLAB, realziar varias pruebas y analizar sus resultados se halló una configuración que suponía ser la óptima. Pero la misma se tuvo que descartar por el hecho de no lograr en todos los casos de escenarios de daños posibles encontrar siempre al menos una solución. Por ello se concluyó que el enfoque de la investigación debía cambiar, no se debe enfocar en encontrar una configuración con el mejor rendimiento posbile, en cambio, se desea que la misma logre, sin importar el tipo de daño, encontrar al menos una solución.

Palabras clave: detección de daño, vibración, rigidez, función objetiva, *PSO, F-Race, DOE.*

Abstract

The particle swarm optimization problem (PSO) is an algorithm that is used to minimize a particular objective function. In this project it was used to minimize the discrepancy between the vibration data identified by modal testing and the computed one from the analytical model in order to find any damage in a cantilever beam.

To distinguish if there is any change in the rigidity of the structure, the values of the parameters of the PSO algorithm must be selected correctly. As all the possible combinations of values are vast, having at least 20.000 configurations, two different methodologies were performed in order to test the computational model PSO. In one hand, the F-Race method that it is based on testing all the configurations in various damage scenarios in a limited amount of time, comparing between them their performance. In the other hand, the DOE – Design of Experiment method was used which consist of creating a graphic that provides information on the relation cause-effect of the possible values of the parameters of the PSO model.

After writing all the algorithms in MATLAB, doing several tests and analyzing the results one configurations was found as the promising one, but immediately it has to be discarded because the combination was not able to find at least one solution in any damage scenario. It was therefore concluded that the focus of the investigation must be change, the idea of searching the configuration that has a high-performance is not the objective anymore, instead the configuration must find, no matter the type of damage scenario, at least one solution.

Keywords: damage detection, rigidity, vibration, objective function, PSO, F-Race, DOE.

Contenido

Introducción

La detección temprana de daño en diferentes tipos de estructuras relacionadas con ingeniería mecánica, civil o aeroespacial aporta enormes beneficios económicos y de seguridad. Encontrar el problema antes del fallo permite poner fuera de servicio la estructura, dando tiempo a reparar y/o reemplazar el componente previo a que ocurra un accidente grave.

El deterioro por el uso, el envejecimiento o por algún acontencimiento ocasional específico lleva a que no sea fácil e intiutivo predecir una falla y su ubiación. Se puede conocer cuánto tiempo de vida útil posee un componente antes de llegar al fallo, su ciclo de vida, pero esto no es más que una estimación. Saber a ciencia cierta el momento exacto en el que se debe reemplazar una pieza es una tarea compleja. La manera correcta de evaluar el estado de un elemnto que se encuentra en funcionamiento es a través de ensayos no destructivos (END), ya que estos no alteran el servicio posterior, las propiedades físico, químicas y/o mecánicas durante su aplicación. Los END poseen cinco parámetros característicos: aplicación de energía, interacción de la energía con el material, detección de los cambios de la energía (emitida y la resustante de la interacción del material), procesamiento de datos y, por último, la interpretación de los resultados.

Cuando se habla de detección de daños o de discontinuidad en una estructura hay diversas técnicas no destructivas que se pueden aplicar. Las mismas se pueden dividir en dos grandes grupos: por métodos de detección de daño local y por métodos de detección de daño global. El primer grupo requiere a priori conocer la zona donde se localiza el daño (por ejemplo: zonas de concentración de tensiones como entallas, puntos angulosos, aristas, etc.) y la estructura debe ser, en la mayoría de los casos, accesible para su inspección. De entre todos estos método podemos nombrar algunos: el método ultrasónico, campo magnético, radiografía, líquidos penetrantes, etc. Por otro lado, los métodos de detección de daño global hacen referencia a aquellos métodos que inspeccionan grandes estructuras donde no se tiene información previa de la zona de daño. En esta categoría se encuentran los métodos que se basan en el cambio en la vibración para encontrar una discontinuidad. Hay ciertas modificaciones que se dan en las propiedades y parámetros de la estructura si la misma está dañada, la rigidez, por

ejemplo, decrece. La disminución del módulo eslástico o módulo de Young en uno o varios elementos de una estructura se puede utilizar como indicador de falla.

Una manera de obtener información de los datos vibratorios de una estructura es a través del uso de dos elementos: un martillo que aplique la perturbación y acelerómetros que recolecten los datos de la respuesta (**Figura 1**).

Figura 1: Elementos para ensayo de vibración. Martillo (force input) y acelerómetro (accelerometer response)

Hay dos técnicas diferentes para realizar un análisis modal: método SISO (single input, single output), una entrada y una salida, o MIMO (multiple input, multiple output) múltiple entrada y múltiple salida. Dependiendo las condiciones del componente a estudiar se pueden elegir entre estas técnicas. Por ejemplo, si se esta estudiando una estructura de largas dimensiones realizar un estudio SISO no tendría mucho sentido, gran parte de la información se perdería. En otros casos, usar el método MIMO puede afectar el peso de los acelerometos en los datos finales.

Aplicando, según las condiciones del elemento a estudiar, cualquiera de los dos métodos como resultado se obtienen las fracuencias naturales. A continuación se explicará un ensayo vibratorio relizado en Laboratorio *LAMÉ* en el *INSA* con la finalidad de comprender un poco más en detalle el proceso de extracción de datos vibratorios y su complejidad.

Considerando como objeto de estudio una viga empotrada (dimensiones viga ver Capítulo 1), se decide realizar el método SISO con la variente de posicionar el martillo de excitación en un punto fijo y de cambiar la posición del acelerómetro en 5 puntos diferentes. El cambiar de ubicación del receptor permite obtener más información y de mejor calidad.

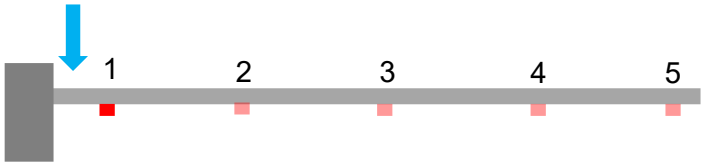


Figura 2: Representación esquemática de una viga empotrada y la ubicación del martillo de fuerza (flecha en azul) y las diferentes posiciones que toma el acelerómetro (cuadrado en rojo)

En la **Figura 3** podemos obsevar los datos vibratorios obtenidos luego de realizar un estudio SISO a una viga empotrada, cada color corresponde a la información obtenida por cada una de las 5 posiciones diferentes en que fue colocado el acelerómetro. Los picos que se observan corresponden a las frecuencias naturales. Los primeros tres modos de vibración lograron ser captados con mayor precisión, realizando un promedio entre las respuestas se obtuvieron los siguientes valores: modo nro.1 18,5Hz; modo nro.2 110Hz y modo nro. 3 316Hz. Obtener más de tres modos de vibración resultó imposible ya que la fuerza de excitación se encontraba en un punto fijo en uno de los extremos de la viga, impidiendo una buena excitación a lo largo de toda la estructura.

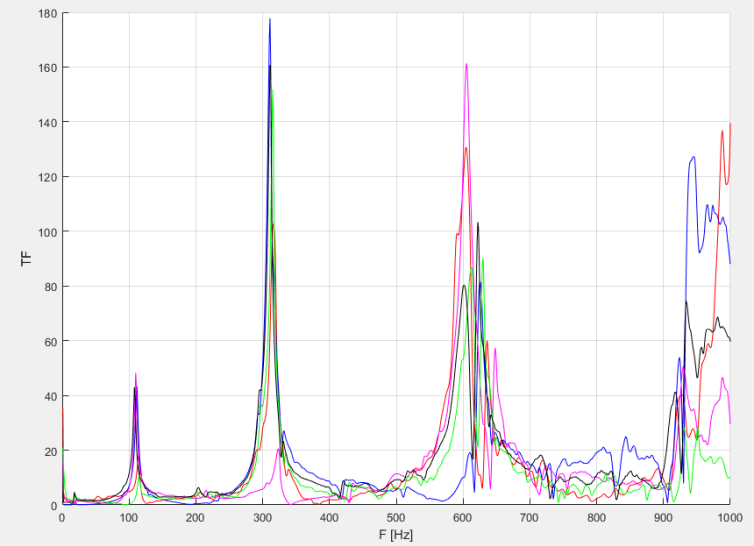


Figura 3: Función de Transferencia.

Para corroborar estos resultados, se realizó el cálculo de los tres primeros modos de la viga mediante un análisis numérico y se obtuvieron los siguientes resultados: modo nro.1 17,7Hz; modo nro.2 111,3Hz y modo nro.3 312Hz.

Las diferencias encontradas pueden deberse a diversos motivos, uno de ellos podría ser que el movimiento del acelerómetro puede cambiar la distribución de masa de esta manera alterando las frecuencias naturales de la viga. Por otro lado, la lectura modal podría estar afectada por la ubicación en la que se encuentra el acelerómetro, cuasando que cierta información del modo de vibración se pierda. Por lo expresado con anterioridad, queda en evidencia que la lectura de la frecuencia natural de una estructura es una tarea para nada fácil.

Una buena lectura de datos vibratorios es sumamente importante ya que la misma que puede aportar infomación de que es lo que está ocurriendo con el módulo de Young y, por consiguiente, con el estado de una estructura.

A lo largo de este proyecto, se trabajará con el método de aproximación inversa para la detección de daños mediante datos de vibración. La finalidad del método es minimizar una función objetiva que representa la discrepancia que hay entre los datos de vibración identificados por el ensayo modal (en un futuro se utilizaran los resultados de un ensayo vibratorio) y el calculado a partir de un modelo analítico. Lograr esto no es nada fácil, no es un problema que se pueda resolver con algoritmos de optimización convencionales.

La optimización por enjambre de partículas (o PSO por sus siglas en inglés "*particle swarm optimization*") es el algoritmo usado para minimizar la función objetiva. Se ha comprobado que el PSO tiene ventaja sobre cualquier otro tipo de algoritmo, tiene términos simples y son sólo dos ecuaciones. Son pocos los parámetros a ajustar pero es muy importante hacerlo correctamente.

El objetivo principal de este proyecto es encontrar la mejor combinación de parámetros para el algortimo de PSO. Se continuaron previas investigaciones realizadas en el laboratorio LaMé (*Laboratoire de Mécanique Grabiel Lamé*, Blois, Francia) en la universidad INSA (*Institut National des Sciences Appliquées – Centre Val de Loire*) utilizando dos algoritmos diferentes de optimización F-Race y DOE (por sus siglas en inglés *"desing of experiment"*).

Capítulo 1: Formulación del problema

En esta investigación se utiliza como estructura de estudio una viga en voladizo con amortiguación despreciable. El desplazamiento de la viga puede ser modelizado por la siguiente ecuación de Euler-Bernoulli:

$$\frac{\partial^2}{\partial x^2}\left[EI(x)\frac{\partial^2 \omega(x,t)}{\partial x^2}\right] + m(x)\frac{\partial^2 \omega(x,t)}{\partial t^2} = f(x,t) \tag{1}$$

donde $\mathrm{EI(x)}$ es la rigidez a la flexión, $\mathrm{m(x)}$ es la masa por unidad de viga y $\omega(\mathrm{x},\mathrm{t})$ representa el desplazamiento transversal. La viga se discretiza en D elementos usando el método de elementos finitos (MEF) y sus parámetros materiales son el módulo de Young $E = 2.0 \times 10^{11}\ Pa$ y su densidad $\rho = 7850\ kg/m^3$. Las dimensiones de la viga son definidas por la longitud total $l = 1\,m$ y su área tranversal, con un ancho de $w = 2.49 \times 10^{-2}\ m$ y una profundidad de $dp = 5.3 \times 10^{-3}\ m$.

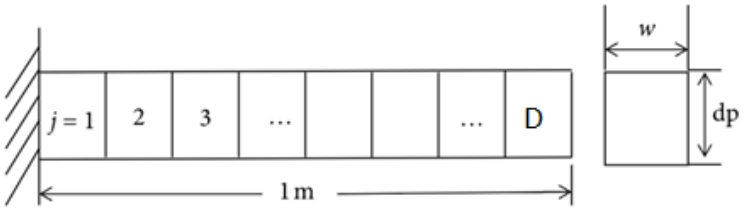


Figura 1-1: Modelo Euler-Bernoulli de la viga.

La ecuación dinámica de la estructura con n grados de libertad es:

$$M.\ddot{X}(t) + k.X(t) = F(t) \tag{2}$$

donde t es tiempo, $X(t)$ es el desplazamiento, $\ddot{X}(t)$ es la aceleración y $F(t)$ es la fuerza externa. Asumiendo que la fuerza externa es nula (libre vibración), la ecuación de eigenvalores asociada con la ecuación (2) es:

$$(K - \lambda_s M)\Phi_s = 0 \tag{3}$$

donde Φ_s representa el s^n orden de la forma de vibración, λ_s representa el s^n eigenvalor, el cuadrado de la frecuencia natural. M y K son las matrices globales de rigidez y masa:

$$K = \sum_{j=1}^{D} k_j \tag{4}$$

$$M = \sum_{j=1}^{D} m_j \tag{5}$$

Estas matrices se obtienen convirtiendo el sistema de coordenadas local al sistema de coordenadas global. Para una viga uniforme la matriz de masa y rigidez esta dada por cada elemento de la viga de la siguiente manera:

$$m_j = \frac{\rho.A.l}{420}\begin{bmatrix} 156 & 22l & 54 & -13l \\ 22l & 4l^2 & 13l & -3l^2 \\ 54 & 13l & 156 & -22l \\ -13l & -3l^2 & -22l & 4l^2 \end{bmatrix} \tag{6}$$

$$k_j = \frac{EI}{l^3}\begin{bmatrix} 12 & 6l & -12 & 6l \\ 6l & 4l^2 & -6l & 2l^2 \\ -12 & -6l & 12 & -6l \\ 6l & 2l^2 & -6l & 4l^2 \end{bmatrix} \tag{7}$$

donde ρ es la densidad de la viga, A y l son el área y longitud de la viga, E el módulo de Young e I es la inercia.

Ahora, si en algún elemento de la viga hay un daño presente, la ecuación de eigenvalores se representa de la siguiente manera:

$$(K_d - \lambda_{\mathrm{ds}} M)\Phi_{\mathrm{ds}} = 0 \tag{8}$$

donde K_d es la matriz de rigidez modificada, λ_{ds} es la nueva frecuencia natural y Φ_{ds} los nuevos vectores del modo de vibración luego del daño. Resolviendo las siguientes dos ecuaciones de eigenvalores Podemos obtenes la frecuencia natural antes y después del daño.

$$\lambda_{\mathrm{hs}} = \omega^2{}_{hs} \rightarrow f_{hs} = \frac{\omega_{hs}}{2\pi}\ [Hz] \tag{9}$$

$$\lambda_{\mathrm{ds}} = \omega^2{}_{ds} \rightarrow f_{ds} = \frac{\omega_{hs}}{2\pi}\ [Hz] \tag{10}$$

Asumiendo que la rigidez de cada elemento de la estructura decrece uniformemente luego del daño, el factor de reducción es:

$$\alpha_j = \frac{E - E_j}{E} \tag{11}$$

donde E y E_j son el módulo de Young del elemento j^n antes y después del daño. El factor α_j puede tomar valores entre 0 y 1. Si $\alpha_{j^n} = 0$ el elemento j^n de la estructura está intacto, pero si $\alpha_{j^n} = 1$ el elemento j^n está completamente dañado. La nueva matriz global de rigidez se representaría como:

$$K_d = \sum_{j=1}^{D} (1 - \alpha_j) k_j \tag{12}$$

La ecuación (11) forma la base del problema inverso. El objetivo es buscar un vector de daño específico α^* el cual su respuesta dinámica sea consistente con la frecuencia natural medida (referencia). En este estudio, para predecir el vector de daño α^* será utilizado un algoritmo de optimización por enjambre de partículas (PSO por sus siglas en inglés).

Para cuantifiacar la discrepancia que hay entre la frecuencia de referencia y la frecuencia predecida por el algoritmo de PSO se utilizará una función fitness $S(\alpha^*)$, tipo particular de función objetiva.

$$\begin{gathered} \arg\min S(\alpha^*) \\ where\ \alpha^* = (\alpha_1^*, \alpha_2^*, \dots, \alpha_D^*) \\ subject\ to\ \alpha_j^*\ \epsilon\ [0,1]\ (1 \le j \le D) \end{gathered} \tag{13}$$

Esta función fitness evalúa que tan cerca se encuentra una solución dada de la solución óptima del problema, si $\alpha = \alpha^*$ la función fitness llegó a su valor mínimo.

Capítulo 2: Algoritmo de PSO

El algortimo de optimización a través de enjambre de particulas o más conocido como algoritmo PSO, por sus siglas en inglés, es un modelo computacional basado en la inteligencia de enjambre. Está inspirado en el comportamiento de búsqueda de alimentos en grupo de los pájaros y es usado para resolver problemas de optimización.

A cada pájaro de la bandada se lo denomina partícula, cada población esta compuesta por una cierta cantidad de partículas, y estas particulas represenrtan una solución potencial.

Cada particula tiene su propia velocidad y posición y ellas “vuelan” un esapacio de búsqueda de D dimensiones. El sistema es inicializado con un set random de potenciales soluciones y cada partícula evoluciona hacia una mejor a través de las iteraciones. En cada iteración las particulas actualizan su velocidad y posición trackeando la mejor posición personal $(pbest)$ y la mejor posición global $(gbest)$.

Supongamos que hay un enjambre en la iteración t y la velocidad v y la posición x de la partícula i de la dimensión j puede ser expresada como:

$$v_{ij}(t+1) = \omega . v_{ij}(t) + c_1 . r_{1ij}\left(pbest_{ij}(t) - x_{ij}(t)\right) + c_2 . r_{2ij}\left(gbest_j(t) - x_{ij}(t)\right) \quad (14)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t) \quad (15)$$

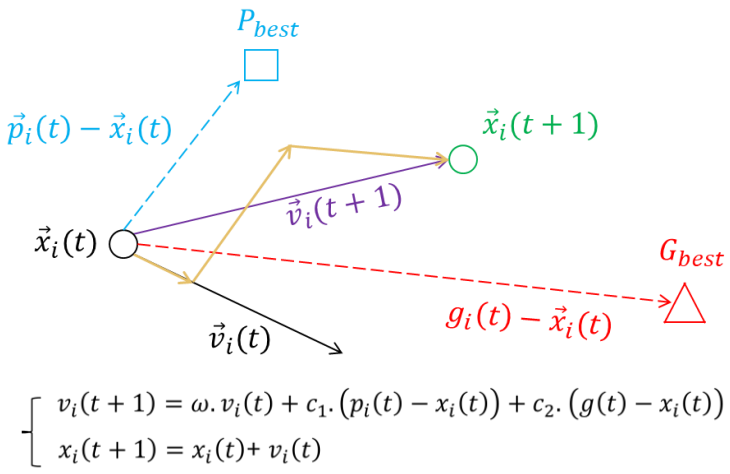
donde ω es la masa inercial que refleja el impacto de la velocidad actual de la particula en la siguiente iteración; r_{1ij} y r_{2ij} son números ramdom uniformemente distribuidos entre [0,1]; c_1 es el coeficiente de aceleración (parámetro social) que controla la tendencia de la partícula hacia su mejor localización personal y c_2 es el coeficiente de aceleración (parámetro cognitivo) que ajusta la tendencia del acercamiento de la particula hacia su mejor localización global.

Durante el proceso de búsqueda, para evitar una búsqueda inválida, la posición y la velocidad están limitadas por ciertos intervalos [Xmin; Xmax] y [Vmin; Vmax].

Las partículas tienen su propia memoria lo que permite que mantengan la mejor posición y el último movimiento que cada una llegó, también poseen memoria colectiva de la mejor posición encontrada entre todas ellas.

Para realizar el movimiento, cada partícula sigue tres tendencias como las que se pueden observar en los tres términos de la ecuación (14): la primera es el desplazamiento que realizó con anterioridad, la segunda es el camino hacia la mejor posición encontrada anteriormente y finalmente el camino que sigue hacia la mejor posición encontrada por todo el enjambre. Diferentes coeficientes son aplicados a cada tendencia con el fin de obtener una nueva y óptima posición de las partículas.

En la **Figura 2-1-1** se ve representada, en una sola dimensión para un mejor entendimiento, cómo trabaja el algortimo PSO:



$$\begin{cases} v_i(t+1) = \omega . v_i(t) + c_1 . \left(p_i(t) - x_i(t)\right) + c_2 . \left(g(t) - x_i(t)\right) \\ x_i(t+1) = x_i(t) + v_i(t) \end{cases}$$

Figura 2-1: Representación simple del movimiento de una partícula

El movimiento de las particulas hacia una nueva posición es repetido hasta que se llega al valor máximo de iteraciones permitidas o hasta que se encuentra la solución.

Los principales pasos del algortimo PSO se encuentran listados a continuación:

1- Inicializar la posición y la velocidad de las particulas.
2- Calcular el valor de la función fitness $S(x_i)$ (i=1,2, ⋯, N).
3- Para cada partícula x_i, comparar el valor de la función fitness $S(x_i)$ con la mejor posición local $S(pbest_i)$ que la misma haya experimentado. Si $S(x_i) < S(pbest_i)$, actualizar x_i como la actual mejor posisión personal.

4- Para cada partícula x_i, comprar su mejor valor fitness personal $S(pbest_i)$ con el mejor valor fitness global $S(gbest)$. Si $S(pbest_i) < S(gbest)$, actualizar $gbest$ como la actual mejor posición global.
5- Ajustar la velocidad y la posición de la población de acuerdo con las ecuaciones de velocidad y posición. Aplicar las condiciones de contorno.
6- Si el algoritmo llega al número máximo de iteraciones o al valor mínimo de la función fitness, el algoritmo se para y se exhiben los resultados, si no es así se vuelve al paso nro 2-. [4]

Para el caso de estudio de esta investigación, las partículas son vectores numéricos que representan el valor del defecto en cada elemento. La posición de las particulas representarían la distribución de esos defectos. En el Anexo A se encuentran las líneas de código del algoritmo que calculan lo explicado con anterioridad.

Como ya fue mencionado, para saber si se encontró la correcta distribución de defectos, se necesita hacer una comparación entre las frecuencias de referencia y el modelo de referencia obtenido con el modelo PSO-MEF a través del problema inverso utilizando la función fitness. Hay que realizar una aclaración para la comprensión del lector, cuando se habla de frecuencias de referencia en esta investigación hablamos de un vector de frecuencia generado por el modelo de MEF donde se elige la cantidad de elementos dañados y el nivel de daño. En un futuro, con el algortimo completamente desarrollado la frecuencia de referencia se tomará en campo.

No es una tarea sencilla asegurar la convergencia del enjambre hacia el mínimo valor de la función fitness. La **Figura 2-2** es una simple representación de tres inicializaciones de particulas diferentes y, dependiendo de este valor, pueden llegar a diversos resultados. Una inicialización podría llevar la partícula hasta un mínimo local (partícula en rojo) lo que da como resultado un falso positivo, otra puede salir fuera de las condiciones de contorno (partícula en violeta) y, el caso deseado, es que la correcta posición inicial logre hacer que la partícula llegue al mínimo real global de la función fitness (partícula en verde), la solución real.

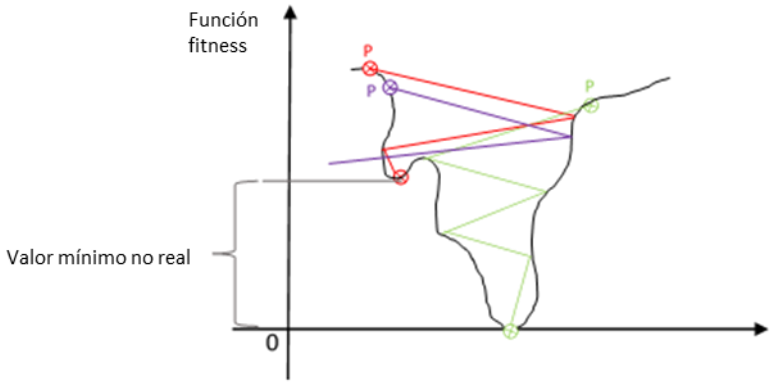


Figura 2-2: Representación simple del algoritmo PSO y la función fitness

Para evitar que los dos primeros ejemplos ocurran, la configuración de los parámetros del problema y de los coeficientes del algoritmo PSO deben ser óptimos.

A través del diagrama de flujo se puede comprender de manera más sencilla cómo funciona el algoritmo.

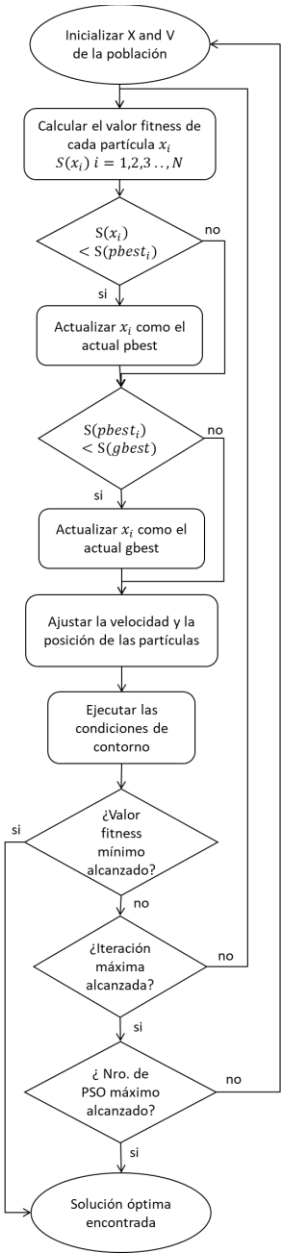


Figura 2-3: Diagrama de flujo algoritmo PSO

Capítulo 3: Parámetros que afectan la ejecución del algoritmo de detección de daños

La selcción de los parámetros del problema y del algoritmo PSO pueden afectar los resultados. Se pueden llegar a convergencias locales, falsos resultados, partículas que se van fuera de los límites del problema, altos costos de tiempo de computación, etc.

A continuación se detalla la lista de todos los parámetros a tener en cuenta a la hora de elegir sus valores:

Parámetros PSO:

- Función fitness
- Precisión
- Número de partículas
- Número de interaciones
- Número de repeticiones del PSO
- Coeficientes de aceleración c_1 y c_2
- Masa inercial
- Inicialización de X e Y
- Máximos y mínimos de X e Y

Parámetros del problema:

- Número de elementos, dimensión del espacio de búsqueda
- Número de daños
- Nivel de daños
- Número de frecuencias

En los siguientes párrafos se descibren cada uno de los parámetros y se fijarán valores a algunos de ellos.

Parámetros del algortimo PSO

Función fitness

Luego de realizarse una búsqueda en la literatura, siete tipo de funciones fitness basados en frecuencias fueron encontrados y testeados para seleccionar el que mejor funciona con el problema propuesto.

Extraido del artículo de Xiao-Li [2] se encontraron las siguientes ecuaciones:

$$S1 = \sum_{s=1}^{m} \left(\frac{(f_s^d)^2 - (f_{si}^p)^2}{(f_s^d)^2} \right)^2 \tag{16}$$

$$S2 = \sum_{s=1}^{m} \frac{(f_s^d - f_{si}^p)^2}{(f_s^d)^2} \tag{17}$$

$$S3 = \sqrt{\frac{1}{m} \sum_{s=1}^{m} \left(\frac{f_s^d}{f_{si}^p} - 1 \right)^2} \tag{18}$$

$$S4 = -\frac{1}{2} \left[\frac{|\Delta F^T . \delta F(X_i)|^2}{(\Delta F^T . \Delta F)(\delta F^T(X_i) . \delta F(X_i))} + \frac{1}{m} \sum_{s=1}^{m} \frac{\min(f_s^d, f_{si}^p)}{\max(f_s^d, f_{si}^p)} \right] \tag{19}$$

Donde m es el número de frecuencias. El vector $F_h = (f_1, f_2, \dots, f_m)^T$ representa la frecuncia de la estructura sana mientras que $F_d = ({f_1}^d, {f_2}^d, \dots, {f_m}^d)^T$ es la frecuencia con la estructura defectuosa calculada por MEF (frecuencia de referencia). El vector $F(X_i) = ({f_{1i}}^p, {f_{2i}}^p, \dots, {f_{mi}}^p)$ representa la frecuencia predecida por el algoritmo PSO-MEF usando la ubicación del vector de la partícula $X_i(1, 2, \dots, D)$. Finalmente, $\Delta F = (F_h - F(X_i))/F_h$ es el vector de variación de frecuencia de la estructura defectuosa relativo al vector de estructura sana calculado por MEF y, $\delta F(X_i) = F_h - F(X_i)/F_h)$ es, también, el vector de variación de frecuencia pero calculado por el algortimo PSO-MEF.

Otro artículo [11] pronone usar la función fitness basada en el mínimo de la norma euclediana de la variación de las frecuencias obtenidas ($\overline{f_j}$) y las frecuencias naturales ($\widehat{f_j}$).

$$S5 = min \sum_{j=1}^{q} \left\| \frac{\overline{f_J} - \widehat{f_J}}{\overline{f_J}} \right\|_2 \; ; \; j = 1, \dots, q \; (number \; of \; modes) \tag{20}$$

Esta otra función fitness (21) está basada en la diferencia de las frecuencias experimentales, de referencia, y analíticas. Donde n es el número total de modos, f es la frecuencia natural y los subíndices a y b representan si la misma es analítica o experimental respectivamente.

$$S6 = \sum_{i=1}^{n} \left[\frac{f_{a,i} - f_{e,i}}{f_{e,i}} \right]^2 \tag{21}$$

Y por último, tenemos la siguiente función fitness que se basa en la diferencia entre la frecuencia natural medida f_i^m y la calculada f_i^a:

$$S7 = \sum_{i}^{n} (f_i^m - f_i^a)^2 \; ; i = 1, .., n \; (orden \; modal) \tag{22}$$

Para evaluar la habilidad del algortimo PSO para encontrar la solución óptima se testeó el mismo con cada uno de las siete funciones fitness. Como indicador se utiliza la tasa de éxito (*succeess rate*) de solución entre un total de 10 ensayos. Se consideró que la viga daño en solamente dos elementos.

Tabla 3-1: Evaluación de la performance de las siete funciones fitness.

Fitness function	S1	S2	S3	S4	S5	S6	S7
Success Rate	0.06	0.11	0	0.47	0.05	0.11	0.1

De acuerdo a los resultados de la **Tabla 3-1**, la función S4 será seleccionada para ejecutar la detección de daño en una viga empotrada.

Número de partículas

Un gran enjambre representa una enorme número de particulas permite que gran parte del espacio de búsqueda esté cubierto en cada iteración y también permite una mayor

diversidad inicial. Sin embargo, esto puede llevar a un tiempo computacional alto. Este parámetro del PSO dependerá del problema que se desee resolver [8].

Número de iteraciones

Este párametro también dependerá del problema a resolver, pocas iteraciones puedo no dirigir a la solución óptima; y un gran número de iteraciones puede llevar a un tiempo computacional innecesario. El correcto número de iteraciones es cuando la solución es hallada. Teniendo como mínimo 200 iteraciones o más se puede asegurar la convergencia del algoritmo estándar del PSO [2].

Número de lanzamientos del PSO

El número de lanzamientos del PSO hace referencia a la cantidad de veces que el programa repetirá el algoritmo en un ensayo. Si su valor se aumenta, más resultados estarán disponibles para comprar pero requerirá de mayor tiempo computacional.

Masa inercial

Primer término de la ecuación de velocidad (14): $\omega . v_{ij}(t)$

El factor de inercia contrala parte de la velocidad. Si su valor es muy grande, las partículas estarán cerca de la solución pero jamás serían la solución. Sin embargo, si el valor es pequeño a la partícula le costará viajar a través de todo el espacio de búsqueda. Para resolver este problema, la estrategia de decrecimiento lineal del factor de inercia será utilizado. Cuando la particula se este acercando a la solución, la velocidad de la misma comenzará a descender y reducir sus posibilidades de irse lejos de la solución óptima [5].

La siguiente ecuación será usada para representar este fenómeno:

$$w = w_{max} + \frac{CurrIter}{MaxIter}(w_{max} - w_{min})$$

$$w_{max} = 0.9; w_{min} = 0.4 \quad (23)$$
$$CurrIter: iteración\ actual$$
$$MaxIter: máxima\ iteración$$

Coeficientes de aceleración c1 y c2

Segundo y tercer término de la ecuación de velocidad (14): $c_1.r_{1ij}\left(pbest_{ij}(t) - x_{ij}(t)\right) + c_2.r_{2ij}\left(gbest_j(t) - x_{ij}(t)\right)$

Los coeficientes de aceleración son aplicados a las tendencias de las particulas de moverse hacia la mejor posición encontrada por un partícula y hacia la mejor posición encontrada por el enjambre.

Los valores de c_1 y c_2 deben seguir ciertas condiciones. Podemos encontrar en gran parte de la literatura, como así también en investigaciones previas realizadas en el laboratorio *LaMé*, un criterio común $c_1 + c_2 < 4$. Muchos autores coinciden en usar $c_1 = c_2 = 2$, mientras que otros [18, 19] consideran un criterio diferente que será testeado en este preoyecto:

$$c_1 + c_2 < 2 \times (2 + 2w_{min}) \tag{24}$$

No obstante, hay otra estrategia que los coeficientes pueden seguir. Usando la ecuación (25) el enjambre se enfocará en la búsqueda en el área local de la particual en individual en las primeras iteraciones ($c_1.r_1 > c_2.r_2$) y convergerá hacia la mejor partícula global en la iteración final ($c_1.r_1 < c_2.r_2$) [5].

$$c_1 = c_{1,ini} + \left(c_{1,fin} - c_{1,ini}\right) \times \frac{CurrIter}{MaxIter}$$

$$c_2 = c_{2,ini} + \left(c_{2,fin} - c_{2,ini}\right) \times \frac{CurrIter}{MaxIter}$$

$$c_{1,ini} = 2.5;\ c_{1,fin} = 0.5;\ c_{2,ini} = 0.5;\ c_{2,fin} = 2.5 \tag{25}$$

$$CurrIter: iteración\ actual$$

$$MaxIter: máxima\ iteración$$

La siguiente imagen puede dar una mejor idea de cómo funciona esta estrategia:

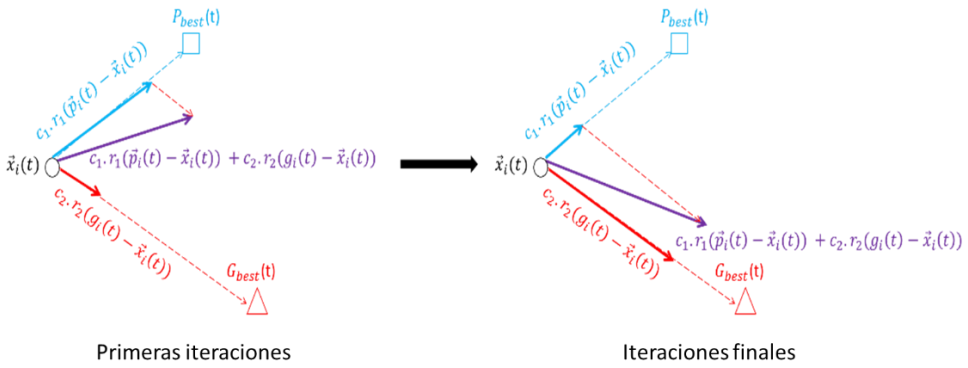


Figura 3-1: Estrategia de los coeficientes de aceleración.

Inicialización de la posición ,X, y la velocidad, V

Fue probado que la velocidad puede ser seteada en cero para prevenir una explosión del enjambre al inicio del algoritmo, mientras que la posición puede ser aleatorea [18].

Valores máximos y valores mínimos de la posición, X, y la velocidad, V

Los valores que pueden tomar las posiciones de las partículas se encuentran en el rango [0, 1]. Esto se debe a que la posición X de nuestro modelo es el vector de daño α.

Usualmente setear Vmáx=Xmáx es la mejor opción para los casos de detección de daño en la composición de una viga empotrada. Setear las condiciones de contorno de velocidad restringen la capacidad de exploración de la población y disminuyen la probabilidad de que la partícula se aleje del espacio de búsqueda [1].

Parámetros del problema

Número de elementos

El número de elementos está realcionado con MEF que discretiza la dimensión del espacio de búsqueda para el PSO. En este caso de estudio, la viga será divida en 30 elementos. A mayor número de elementos, mayor será la precisión del modelo pero requerirá mayor tiempo computacional para resolver el problema.

Cantidad y nivel de daño

Estos parámetros dependerán del ensayo a realizar. Se trabajará con una cantidad máxima de tres elementos dañados con una severidad entre el 10% y el 40%.

Cantidad de frecuencias y su precisión

El número de frecuencias y su precisión dependen en el número de sensores y su exactitud, lo cual estará limitado por los aspectos tecnológicos del material que se disponga en un ensayo real. Estos parámetros son muy importantes ya que aumentan la exactitud de los resultados y evitan la obtención de falsos positivos. Entre 5 y 9 frecuencias es un rango razonable para usar en el cálculo con una precisión de 4 dígitos luego de la coma decimal [9].

Capítulo 4: Algoritmo de optimización para la configuración de parámetros

Como fue mencionado con anterioridad, la performance de la parametrización del algoritmo PSO depende fuertemente de los valores de los parámetros. La selección correcta de ellos de por sí ya es un problema de optimización. Para encontrar la mejor configuración de parámetros para el PSO serán utilizados dos procesos diferentes y serán comparados. Por un lado, tenemos el “F-Race”y por el otro, “DOE” por sus siglas en inglés *Designo f Experiment”.*

F-Race

El F-Race es un algoritmo de optimización que encuentra, en una cantidad de tiempo limitado, una configuración de parámetros que ejecuta de la mejor manera posible un cierto problema de optimización, probando las diversas configuraciones en diferentes instancias o escenarios.

Para un mejor entendimiento, se puede comparar el procedimiento F-Race con la fuerza bruta. Este último es un método simple que prueba todas las configuraciones posibles en diversas instancias representando así todos los casos posibles existentes y observa cual de ellos da los mejores resultados. La fuerza bruta es un buen método para aplicar en casos donde el número de configuraciones posibles es bajo, si no es el caso, podría tomar una enorme cantidad de tiempo analizar y encontrar los mejores resultados de todos los candidatos posibles.

En la siguiente imagen, **Figura 4-1**, es posibles observar que tanto el método de fuerza bruta como el F-Race comienzan con la misma cantidad de configuraciones posibles para la primer instancia cubriendo incluso la misma superficie. Esto quiere decir que ambos enfoques, en un inicio, realizan la misma cantidad de experimentos. El algoritmo de F-Race decrece rápidamente la cantidad de configuraciones y, por lo tanto, el tiempo computacional que conlleva cada instancia. Se enfoca más en los candidatos más prometedores y permite evaluar las mejores configuraciones en más de una instancia, obteniendo resultados más confiables sobre su comportamiento. Por otro lado, como la

fuerza bruta realiza testeos de todas las configuraciones en todos las instancias, deriva en una pérdida de tiempo computacional en candidatos que no son buenos.

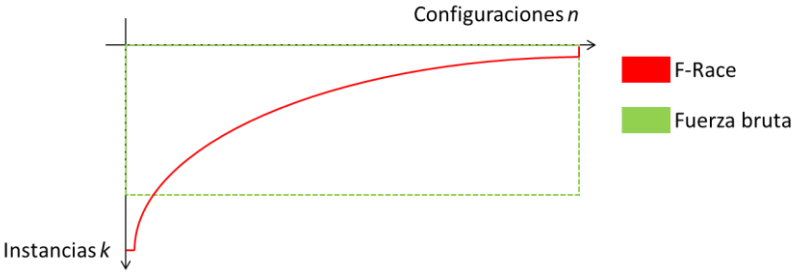


Figura 4-1: Representación visual de la cantidad de tiempo computacional necesario por el método de la fuerza bruta y por el F-Race

Funcionamiento del algoritmo F-Race para el problema de PSO

El algoritmo del F-Race evalua a un número finito de candidatos en diversas instancias. Los candidatos a evaluar son las diversas configuraciones posbiles de los parámetros del PSO. Las instancias hacen referencia a diferentes escenarios de daño que poseen una cierta cantidad de daño con sus correspondientes severidades. En cada instancia, todas las configuraciones restantes son evaluadas y el candidato más “pobre”, es decir, la configuración que sea menos eficiente, será eliminada tan pontro se tenga evidencia suficiente contra ella.

Para lograr lo antes mencionado, el costo de cada configuración es calculado. Este costo es la inversa de la cantidad de veces esa configuración encontró el resultado correcto. Una vez que los costos de todas las configuraciones es calculado, se clasificarán, rankeando de acuerdo al valor del costo. Mientras más grande sea el costo, menor será la calidad de la configuración, menos resultados correctos dará.

Un método estadístico, el test de Friedman, es usado para verificar una hipótesis sobre la variancia del rankeo de los costos de las configuraciones. En el caso de estudio la hipótesis a probar es:

$$H_o = las\ configuraciones\ son\ todas\ igual\ de\ eficientes$$

La siguiente expresión matemátima es usada en este test estadístico. El valor de k hace referencia al número de instancia, o escenario de daño; m es el número de

configuraciones que siguen en carrera, las que no fueron eliminados; R_{lj} representa rankeo o clasificación del costo de cada configuración ($1 \leq l \leq k;\ 1 \leq j \leq m$) y, por último, $R_j = \sum_{l=1}^{k} R_{lj}$ es la suma del rankeos para una misma configuración en todas las instancias que fue testeada.

$$T = \frac{(m-1)\sum_{j=1}^{m}\left(R_j - \frac{k(m+1)}{2}\right)^2}{\sum_{l=1}^{k}\sum_{j=1}^{m} {R_{lj}}^2 - \frac{km(m+1)^2}{4}} \tag{26}$$

Si $T \geq 1 - \frac{\alpha}{2}$, la hipótesis H_o es rechazada, esto quiere decir que al menos una configuración es más eficiente que las demás. Luego, todas las configuraciones deben ser comparadas para encontrar a la mejor, para ello se aplica el test de Conover post-hoc. Si la comparación por parejas, que se encuentra del lado izquierdo de la expresión (27) es mayor que la función de densidad de la probabildad t-student, $t_{1-\alpha/2}$, la configuración es eliminada. La misma no es óptima, por lo tanto no aparecerá en la próxima instancia.

$$\frac{\left|R_j - R_h\right|}{\sqrt{\frac{2k\left(1 - \frac{T}{k(m-1)}\right)\left(\sum_{l=1}^{k}\sum_{j=1}^{m} R_{ij}^2 - \frac{km(m+1)^2}{4}\right)}{(k-1)(m-1)}}} > t_{1-\frac{\alpha}{2}} \tag{27}$$

Otros escenarios que se pueden enfrentar son que en el test de Friedman $T \leq 1 - \frac{\alpha}{2}$ o que solamente quede una configuración, de esta manera el objetivo es logrado, de lo contrario todo debería recalcularse para una nueva instancia.

En el diagrama de flujo que se presenta en la **Figura 4-2** se encuentra resumido el procedimiento del algoritmo. En pocas palabras, el F-Race irá testeando todas las configuraciones posibles del algortimo PSO en diversos escenarios de daños, comparará la cantidad de veces que se obtuvieron buenos resultados, es decir, que se encontraron los daños en la viga y, mediante unos tests estadísticos, eliminará aquellas configuraciones que no logren cumplir con los objetivos.

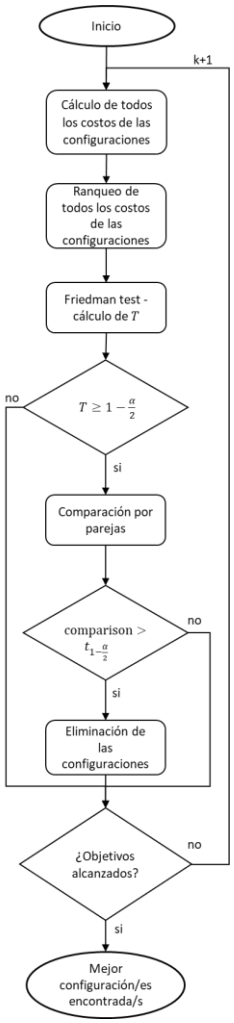


Figura 4-2: Diagrama de flujo del algoritmo F-Race.

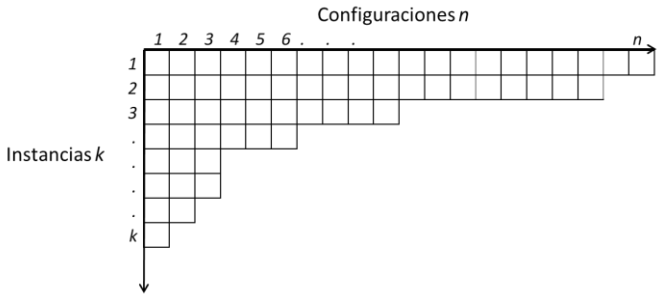


Figura 4-3: Representación gráfica del F-Race.

Este algoritmo puede testear miles y miles de configuraciones, por lo que requiere de mucho tiempo computacional, un lanzamiento del F-Race puede tomar hasta un mes para

completar el procedimiento y arrojar resultados. Por ello, para las pruebas a realizar se utilizará un server que ejecutará el código de MATLAB durante semanas.

Pruebas y resultados del F-Race

Como se comentó en el enunciado anterior, es necesario utilizar contar con un server que corra el programa en MATLAB durante semanas. Se realizó una mejora del código de programación del F-Race ya existente, la habilidad del mismo a recomenzar una prueba en caso de una caída del server. Anteriormente, si el server sufría algún problema y dejaba de funcionar toda la infomación obtenida hasta el momento era almacenada, pero el algoritmo no era capaz de recomenzar los cálculos, los datos del workspace de MATLAB eran eliminados. Toda esta codificación se puede observar en el Anexo B, en las líneas de programación del “F_racek2.mat” y principalmente en las funciones “restart_PSO.mat” y “parforsave2.mat”.

Una única problemática que queda a resolver es la de recomenzar la prueba automáticamente en caso de caída del sistema. Hoy en día, si esto ocurre, se debe volver a correr el programa de manera manual. Lamentablemente no se encontraron métodos disponibles para el programa MATLAB.

Tres lanzamientos del F-Race fueron ejecutados, dos de ellos siguen el criterio de los coeficientes de aceleración de la ecuación (24) y el último sigue el criterio de que $c_1 + c_2 < 4$. A continuación se describen los resultados obtenidos.

- Lanzamiento F-Race nro. 1

Como la cantidad de instancias que el programa ejecutará hasta encontrar la mejor configuració no es conocida, un set de 1000 instancias aleatoreas con un máximo de tres daños cuyas severidades varian entre 10 - 40% será creado (Anexo B – “Instance_generation1.mat”).

Se crea una matriz que contiene todas las posiciones iniciales de todas las particulas que se utilizará para obtener resultados que puedan ser comparables, todas las veces que se ejecute el programa del PSO contará con la misma población inicial (Anexo B – “InitXpar.mat”).

Antes de lanzar el programa, los parámetros del mismo deben ser seteados. Podemos dividirlos en dos grandes grupos, por un lado, los parámetros fijos y, por el otro, los parámetros variables.

Tabla 4-1: Valores de los parámetros del lanzamiento de F-Race nro. 1.

Parámetros fijos	
Parámetros PSO	
Cantidad de PSO	10
Función fitness	Ecuación 19
Presición	4 dígitos para la frecuencia; 2 dígitos para los valores de X y V
Parámetros del problema	
Cantidad de elementos	30
Cantidad de daños	3 máximo
Parámetros variables	
Parámetros PSO	
Cantidad de partículas	10; 15; 20; 25; 30; 50; 60; 100; 200; 300
Cantidad de iteraciones	10; 20; 50; 100; 200; 300
C1	0; 0.5; 1; 1.5; 2; 2.05; 2.5; 3; 3.5; 4
C2	4; 3.5; 3; 2.5; 2; 2.05; 1.5; 1; 0.5; 0
Parámetros del problema	
Número de frecuencias	5; 6; 7; 8

Una vez que todas las combinaciones posibles son creadas y, habiando aplicado el criterio de los coeficientes de aceleración de la ecuación (24), 19920 configuraciones fueron obtenidas para ejecutar el F-Race.

Al realizar esta prueba se encontraron algunos problemas en las versiones de los códigos de programación. La primera vez que se corrió el programa los resultados obtenidos no fueron los esperados, solamente en siete instancias se encontraron dos de las mejores configuraciones donde los parámetros obtenidos no concordarban con la literatura. El número de particulas para las mejores configuraciones fueron de 10 y 15 lo cual contradice lo que se expresó con anterioridad, que a mayor número de partíuclas mejores serían los resultados. Luego de investigar a fondo el programa se observó, en el scrip que trabaja el análisis modal resultado (Anexo A - “AimFcn_1.mat”), que cuando se obtenia la frecuencia a través de la ecuación de los eigenvalores, se obtenian número

imaginarios. Esto llevaba a problemas de cálculos por mezclar operaciones de matrices de número reales e imaginarios en MATLAB.

Con esto ya resuelto, se volvió a correr la prueba pero, nuevamente, se obtuvieron resultados sorpresivos. En tan solo dos instancias, el número de configuraciones bajó de 19920 a 9. Esta vez, el error se encontró scrip del algoritmo PSO (Anexo B – "PSO.mat"). Cuando la posición de la partícula era mayor o menor que la condición de contorno, tomaba valores random en vez de ser reemplazado por el valor máximo o mínimo permito.

Por último, se encontró que se venía usando dos funciones que MATLAB ofrece de manera alternada. El programa ofrece dos maneras de redondear un número *roundn* ó *round*, ambas pueden parecer lo mismo pero poseen una pequeña diferencia. Si, por ejemplo, se desea redondear el valor de 10.631 a 10.6 podemos hacerlo de las siguientes maneras *roundn(10.631, -2)* ó *round(10.631, 2)*, el orden de hacia que lado de la coma se redondea cambia de acuerdo a que funcion se utilice. En diversos scrip se venían usando simultáneamente ambas funciones pero ambas tenían como input *-2*, lo cual terminaban dando resultados erróneos.

Con todo ya resuelto, se ejecutó con éxito el primer lanzamiento del F-Race. Las dos mejores configuraciones encontradas se resumen en la **Tabla 4-2** y, en la **Tabla 4-3**, observamos el rendimiento del PSO con dichas configuraciones en diversos escenarios de daño.

Tabla 4-2: Mejores configuraciones del F-Race nro. 1.

Config	Nro. elementos	Nro. PSO	Precisión	V max	Nro. iteracines	Nro. frequencias	Nro. particulas	C1	C2
1	30	10	4	1	300	8	300	4	1
2	30	10	4	1	300	8	300	4	0.5

La **Tabla 4-3** nos demuestra que estas configuraciones tienen un excelente rendimiento ante casos de una viga con uno o dos elementos dañados. Al complejizar el problema, agregando un tercer daño y teniendo severidades menos homogéneas, al algoritmo le resulta más difícil encontrar la solución.

Tabla 4-3: Rendimiento de las dos mejores configuraciones del F-Race nro. 1 para distintos escenarios de daño.

Caso de daño		SR	
Nro. elemento	**Nivel de daño**	**Nro. elemento**	**Nivel de daño**
7	5%	1	1
2, 22	10%, 10%	1	1
5, 20, 22	40%, 20%, 10%	0	0.2
4, 13, 28	30%, 20%, 30%	0.5	0.8
13, 25, 30	10%, 10%, 20%	0.6	0.7

Se observa que los valores de los parámetros de las mejores configuraciones son los máximos permitidos presentados en la **Tabla 4-1**; número de iteraciones: 300; número de partíuclas: 300 y número de frecuencias: 8. Esto podría indicarnos que hay una posibilidad que valores más altos puedan ser incluso mejores. Es por ello que se propuso realizar otro lanzamiento del F-Race, pero esta vez con un set diferente de parámetros varibles.

- Lanzamiento F-Race nro. 2

En esta nueva prueba, no solo se cambiaran los valores de los parámetros variables sino que, además, se considera usar una nueva estrategia para los coeficientes de aceleración presentada en la escuación (25).

Tabla 4-4: Valores de los parámetros variables del lanzamiento de F-Race nro. 2.

	Parámetros variables
Parámetros PSO	
Cantidad de partículas	100; 200; 250; 300; 350; 400; 450
Cantidad de iteraciones	200; 250; 300; 350; 400; 450; 500
C1	0; 0.5; 1; 1.5; 2; 2.05; 2.5; 3; 3.5; 4
C2	4; 3.5; 3; 2.5; 2; 2.05; 1.5; 1; 0.5; 0
Parámetros del problema	
Número de frecuencias	5; 6; 7; 8; 9

En la prueba anterior, solamente con la información de las dos primeras instancias se procedía a eliminarse las primeras configuraciones pero en este lanzamiento, recién

luego de calcular las primeras tres intancias se permitirá descartar alguna configuración. El objetivo de esto es obtener más valores con los que se pueda comparar, aumentando así la confiabilidad de las configuraciones que fueron descartadas. En la siguientes tablas se exponen los resultados obtenidos.

Tabla 4-5: Mejores configuraciones del F-Race nro. 2.

Config	Nro. elementos	Nro. PSO	Precisión	V max	Nro. iteracines	Nro. frequencias	Nro. particulas	C1	C2
1	30	10	4	1	450	9	450	4	0.5
2	30	10	4	1	500	8	300	4	0.5

Tabla 4-6: Rendimiento de las dos mejores configuraciones del F-Race nro. 2 para distintos escenarios de daño.

Caso de daño		SR	
Nro. elemento	**Nivel de daño**	**Config 1**	**Config 2**
7	5%	1	1
2, 22	10%, 10%	1	1
5, 20, 22	40%, 20%, 10%	0.7	0.3
4, 13, 28	30%, 20%, 30%	0.9	0.7
13, 25, 30	10%, 10%, 20%	1	1

En este segundo lanzamiento del programa se observa una mejora en la tasa de éxito para los casos donde hay tres elementos dañados. Estos cambios se atribuyen a los nuevos valores de los parámetros de las configuraciones, especialmente al aumento de número de partículas y el número de iteraciones.

No es de extrañar que las mejores configuraciones tengan valores de frecuencias más altos, hay una mayor cantidad de valores de frecuencias de referencias con los cuales comparar con el modelo, obteniendo así mejores resultados. Es importante prestar atención al valor de este parámetro, en la práctica, no es tan fácil extraer de una estructura una cantidad eleveada de frecuencias, hay un límite tecnológico. Consecuentemente, consluimos que este valor es solamente teórico y no práctico.

- Lanzamiento F-Race nro. 3

En esta última prueba se consideró comprar las dos estrategias planteadas para los coeficientes de aceleración en un mismo lanzamiento con el fin de corroborar cuál se adapta mejor al problema. Por un lado, se considerará el criterio de la ecuación (25), donde los coeficientes son variables durante la ejecución del programa; por otro lado, se considerará el criterio $c_1 + c_2 < 4$ para valores fijos de c1 y c2.

Como se vio en la prueba anterior, a mayores número de partículas y número de iteraciones se tiende a obtener mejores resultados del PSO, por ello los valores de los parámetros variables fueron reducidos. Esto ayuda disminuir el tiempo computacional.

Tabla 4-7: valores de los parámetros variables del lanzamiento de F-Race nro. 3.

	Parámetros variables
Parámetros PSO	
Cantidad de partículas	350; 400; 450
Cantidad de iteraciones	400; 450; 500
C1	0; 0.5; 1; 1.5; 2; 2.05; 2.5; 3; 3.5; 4
C2	4; 3.5; 3; 2.5; 2; 2.05; 1.5; 1; 0.5; 0
C1 y C2	ecuación (25)
Parámetros del problema	
Número de frecuencias	8; 9

Las dos mejores configuraciones del PSO en este caso se resumen en la **Tabla 4-8**.

Tabla 4-8: Mejores configuraciones del F-Race nro. 3.

Config	**Nro. elementos**	**Nro. PSO**	**Precisión**	**V max**	**Nro. iteracines**	**Nro. frequencias**	**Nro. particulas**	**C1**	**C2**
1	30	10	4	1	450	9	450	3.5	0.5
2	30	10	4	1	500	8	400	3.5	0.5

De la **Tabla 4-9** notamos el rendimiento de estas configuraciones en este lanzamiento es muy similar al anterior. Se logró nuevamente un buen desempeño del PSO con los nuevos parámetros establecidos.

Tabla 4-9: Rendimiento de las dos mejores configuraciones del F-Race nro. 3 para distintos escenarios de daño

Caso de daño		SR	
Nro. elemento	**Nivel de daño**	**Config 1**	**Config 2**
7	5%	1	1
2, 22	10%, 10%	1	1
5, 20, 22	40%, 20%, 10%	0.7	0.8
4, 13, 28	30%, 20%, 30%	0.8	0.8
13, 25, 30	10%, 10%, 20%	1	0.9

Un estudio más profundo se realizó sobre las instancias y las configuraciones que se fueron eliminando. Aquellas configuraciones donde c1 y c2 utilizan la estrategia de la ecuación (25) son eliminadas en las últimas instancias. Esto podría demostrar que este enfoque es una opción pero, para sorpresa y en contra a lo que se expresa en la literatura, la mejor estrategia para los coeficientes de aceleración es cuando poseen valores fijos donde c1>c2 (ver **Tabla 4-8**). Este resultado nos lleva a pensar que el problema de la optimización de parámetros para el algoritmo *PSO* es un problema particular, por lo tanto las afirmaciones que se plasman en los artículos sobre el uso de valores varibles de los coeficientes de aceleración no aplican en este caso.

Finalmente, se concluye que, de acuerdo con el método *F-Race*, las dos posibles mejores combinaciones de los factores del algoritmos del *PSO* que aseguran que el programa detectará los daños en una viga empotrada son los resumidos en la **Tabla 4-8**.

En las próximas páginas se explicará otro método para optimizar los valores de los parámetros del algoritmo *PSO* y se compraran con los obtenidos con el método *F-Race*.

DOE

El diseño de experiemntos, o más conocido cómo *DOE - Desing of Experiments*, es una metodología que se enfoca en la compresión de los efectors de los valores de k factores en las y respuestas. Consiste en elegir combinaciones particulares de m_i modalidades (niveles), de los k_j factores dentro del domiño experimental a fin de obtener el efecto promedio de los factores. El objetivo de este médoto es el de minimizar la cantidad de emperimentos a realizar, a diferencia del *F-Race* que prueba todas las combinaciones

posibles. En otras palabras, el *DOE* permite organizar las pruebas de la manera más eficiente posible para obtener el máximo de información con la mínima cantidad de experimentos.

Existen varios tipos de *DOE*, se puede mencionar las talbas de Taguchi las cuales se usan para un pequeño grupo de factores (máximo 3 o 4), aunque sea práctico de usar el problema de este método es que puede llevar a una gran cantidad de experimentos.

Para probar este método se decidió utilizar los mismos parámetros del lanzamiento *F-Race* nro. 1 (**Tabla 4-1**). Se trabajaran entonces con 5 factores, 3 de ellos con 10 niveles y los otros 2 factores con 4 y 6 niveles. De todas las combinaciones posibles, se eliminan las que no cumplan con el criterio de los coeficientes de aceleración c1 y c2.

Al no poseer una matriz ortogonal con los valores de todos los posibles experimentos, se considera utilizar el diseño *D-Optimal*. En la siguiente sección se explayará cómo funciona este método.

D-Optimal

La idea principal de este diseño de experimento denominado *D-Optimal* es de crear un gráfico que aporte información sobre los efectors en la respuesta del modelo cuando el valor de los factores varia. Para lograr esto se debe reflejar la relación causa-efecto entre la variación de la respuesta y la variación de los factores. Un modelo aditivo sin acoplamiento [17] se utilizará para reflejar esta relación.

Como se mencionó con anterioridad, es necesario recudir la cantidad de experimentos a realizar y, para ello, se utiliza un algoritmo de intercambio. Este algoritmo se basa en crear un modelo de los experimentos y trabajar sobre ellos hasta lograr eliminar aquellos experimentos que consideran que no aportan información importante.

Se debe comenzar creando una matriz de experiencia ξ_N, la misma es una representación matemática del plan de experiencia. Cada fila es una combinación de los k_j factores. A esta matriz se le aplica una relación de codificación biyectiva, en este caso, consiste en recodificar cada columna de la matriz de experiencia en $m_i - 1$ columnas. En la **Figura 4-4** se observa un ejemplo de una matriz de experiencia ξ_3 con 3 factores cada uno con 3 niveles recodificada. Esta recodificación es sumamente sencilla de realizarla en Excel.

ξ_3	A	B	C
1	A2	B3	C1
2	A3	B3	C2
3	A1	B2	C3

⟷

	x(A2)	x(A3)	x(B2)	x(B3)	x(C2)	x(C3)
1	-1	-1	0	1	-1	-1
2	0	1	0	1	1	0
3	-1	-1	1	0	0	1

Figura 4-4: Ejemplo de codificación de una matriz de experiencia

Para esta modelización se tienen una cierta cantidad de valores desconocidos que se pueden estimar con la siguiente ecuación,

$$p = 1 + \sum_{i=1}^{k} (m_i - 1) \tag{28}$$

La matriz modelo X es una matriz $n \times p$ que depende del modelo con p coeficientes. El número de filas n son elegidos de acuerdo a la cantidad de experimentos. Para estimar los valores desconocidos del modelo es necesario que $n \geq p$. Una columna de de valores 1 se añade a la recodificación de la matriz ξ_N obteniendo fianlemnte la matriz modelo con la que se trabajará en el algoritmo de intercambio. Siguiendo con el ejemplo de la **Figura 4-4**, la matriz X queda formada de la siguiente manera,

$$(X) = \begin{bmatrix} 1 & -1 & -1 & 0 & 1 & -1 & -1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & -1 & -1 & 1 & 0 & 0 & 1 \end{bmatrix} \tag{29}$$

La primera columna de esta matriz representa el coeficiente constante a_0 de la función modelo que se presetan en la ecuación (22), es la respuesta de un polinomio de primer grado. Puede ser escrita como un producto escalar entre dos vectores, donde el vector a representa la estimación de los coeficientes del modelo y, el segundo vector, representa la función del modelo $f(x_0)$ donde x_0 es una combinación particular de los factores.

$$\hat{y}(x_0) = a.f(x_0) \rightarrow \hat{y}(x_0) = \begin{bmatrix} a_0 \\ a(A2) \\ a(A3) \\ a(B2) \\ a(B3) \\ a(C2) \\ a(C3) \end{bmatrix} . \begin{bmatrix} 1 \\ x(A2) \\ x(A3) \\ x(B2) \\ x(B3) \\ x(C2) \\ x(C3) \end{bmatrix} \tag{30}$$

La calidad de la matriz de experiencia se determina por la determinante de la matriz de información $X^t.X$, este valor depende solamente de la combinación de los niveles. Grandes valores de la determinante significan que mejor será la calidad de la estimación de los coeficientes del modelo (vector a). Si el resultado de $det\ (X^t.X)$ es cero será imposible calcular los coeficientes, por lo tanto una nueva matriz de experiencia deberá ser creada. La matriz de experiencia será óptima cuando la determinante de la matriz de información esté en su máximo valor. Para encontrar esta matriz óptima, el algoritmo de intercambio deberá ejecutarse.

Otro rescurso matemático que se utiliza a la hora de realizar el algoritmo de intercambio es la varianza estándar representada en la ecuación (31) donde $\mathrm{x_i}$ es el vector que describe un solo experimento.

$$dx(x_i) = f(x_i)^t.(X^t.X)^{-1} \tag{31}$$

Tomando los mismos parámetros y valores del F-Race nro. 1, 19920 son todas las combinaciones posibles de experimentos a realizar. De todas ellas se seleccionará, en un principio, 36 experimentos. A través del algoritmo de intercambio se escojerá la mejor matriz experimental ξ_{36} de la cual se extraerá la información para crear la gráfica de relación causa-efecto. A continuación se explica detalladamente el paso a paso del algoritmo de intercambio (Anexo C – “DOE.mat”).

1- Los experimentos que pertenecen a la matriz de partida ξ_{36} son seleccionados de manera aleatorea.
2- Para el resto de las combinaciones restantes se crea otra matriz de experiencia $\xi_{n-p=19883}$ y se calcula la variación estándar de todos los experimentos. El que posea el mayor valor de $dx(x_i)$ será retenido para entrar en la matriz de partida ξ_{36} convirtiendose ahora en ξ_{36+1}.
3- Para mantener $n = 36$ en la matriz de partida un experimento se debe eliminar. Este será aquel que, al ser eliminado, haga que la $det\ (X^t_{n=36}.X_{n=36})$ de la matriz decrezca lo menos posible.
4- El paso 2 y 3 se debe repetir las veces que sea necesario hasta que se provoque un incremento significativo entre la determinante de la matriz de partida y la determinante de la matriz luego de varios intercambios.
5- Finalmente se consigue la matriz del modelo óptima.

Para obtener resultados más fiables, se debe ejecutar el algortimo de intercambio con una matriz de partida aleatorea ξ_{36} unas 200 veces, guardando todas las matrices optimas obtenidas y seleccionando de entre todas ellas la que posea el valor más alto de la determinante de la matriz de información.

No es garantía que el número de experimentos necesarios en la matriz de experiencia sea de 36, por ello se ejecutará el algoritmo de intercambio a diversas matrices de experiencia con diferentes dimensiones de matriz de partida. La selección del correcto número de experimentos se logra a través de dos indicadores: la determinante de la matriz de momentos (32) y el criterio de G-eficiencia (33).

$$\log 10\left[det\left(\frac{1}{n}(X^t.X)\right)\right]^{1/p} \tag{32}$$

$$G-efficiency = \frac{p}{\max\{d(\xi_N, x_0)\}} \tag{33}$$

El algoritmo de intercambio se programó y ejecutó en MATLAB (Anexo C). Se realizaron en total 64 pruebas, comenzando con una matriz de experiencia de dimension $n = 36$ hasta $n = 100$. De las matrices optimas obtenidas en cada prueba se calcularon los indicadores mencionados con anterioridad.

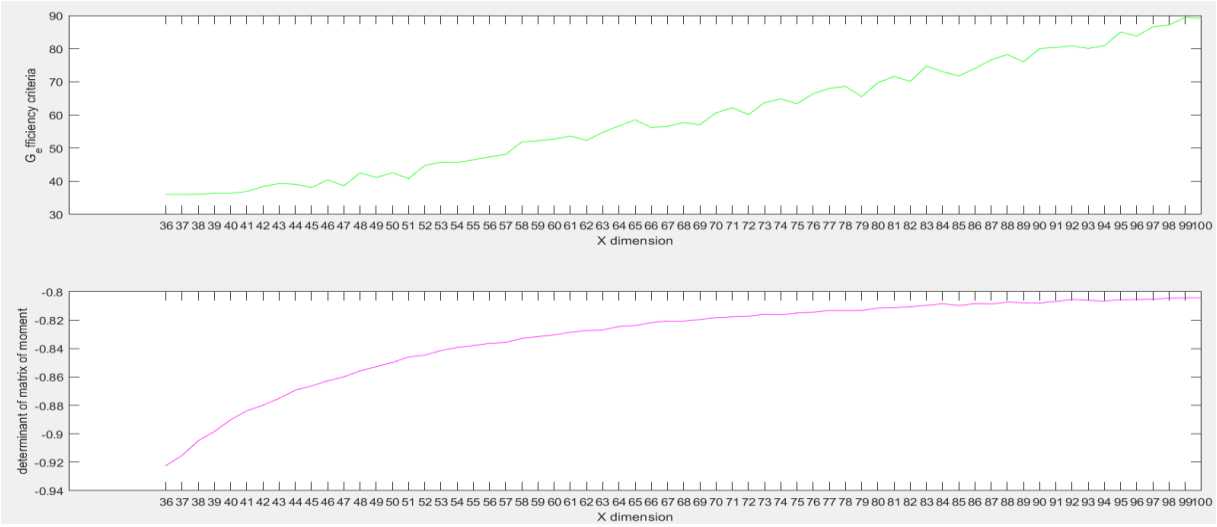


Figura 4-5: Incremento de los indicadores algebraicos de acuerdo a la dimensión de la matriz modelo n=[36:100]. G-eficiencia gráfico superior, determinante de la matriz de momentos gráfico inferior.

El gráfico muestra que a medida que la dimensión de la matriz modelo aumenta, lo mismo lo hacen los valores de los indicadores algebraicos. Seleccionar la dimensión adecuada no es una tarea fácil y no existe una regla específica para hacerlo. De la **Figura 4-5** podemos tomar la dimensión $n = 95$, para este valor hay un gran incremento del valor de la determinante de la matriz de momento y el valor de la eficiencia es 85%. Se descarta seleccionar $n = 99$ a pesar de poseer la mayor eficiencia, el valor de la determinante no varía significativamente para esta dimensión y, además, se prefiere siempre trabajar con la menor dimensión posible con el fin de poseer una menor cantidad de experimentos a realizar.

El plan experimental óptimo fue encontrado y la matriz modelo a utilizar es $X_{95\times36}$. En la **Figura 4-6** se encuentra representado el dominio experimental y los 95 experimentos seleccionados.

Figura 4-6: Representación del dominio experimental junto a los 95 experimentos seleccionados (puntos negros).

Se puede observar que la distribución de los experimentos a realizar se distribuyen a lo largo de todo el dominio. Los experimentos no seleccionados que se encuentran dentro de la zona roja son las configuraciones que no satisfacen el criterio de los coeficientes de aceleración (24).

El próximo paso es obtener la relación entre los diferentes valores de los parámetros y su performance. Para ello, se debe encontrar el vector de los coeficientes a del modelo estimado.

$$Y = X_{95\times36} \,.\, a \tag{34}$$

$$a = (X^t.X)^{-1}.(X^t).(Y) \tag{35}$$

Como la matriz modelo optima $X_{95\times36}$ ya fue encontrada, resta calcular el vector Y. El mismo no es más que los resultados de los 95 experimentos, es decir, el resultado de ejecutar el algoritmo *PSO* con las 95 combinaciones de parámetros (Anexo C – "Y_PSO.mat").

Con el fin de obtener valores más fiables para comprar los resultados se eligen seis escenarios de daños extraidos del set de instancias de la prueba nro 1 del *F-Race* y se utiliza el mismo set de las posiciones iniciales de las partículas. Ahora solo resta calcular a.

Finalmente, con toda esta información es posible plotear el gráfico de los efectos de cada valor de los diferentes factores en el desempeño del *PSO*. El eje y de esta gráfica (**Figura 4-7**) representa la variación de la respuesta del modelo que, en este caso, la variación del soluciones del *PSO*. La forma correcta de leer la gráfica es mirando, por ejemplo, el nro de partículas, si incrementamos el valor de este parámetro de 100 a 200, el número de soluciones crece de 10 a 14.

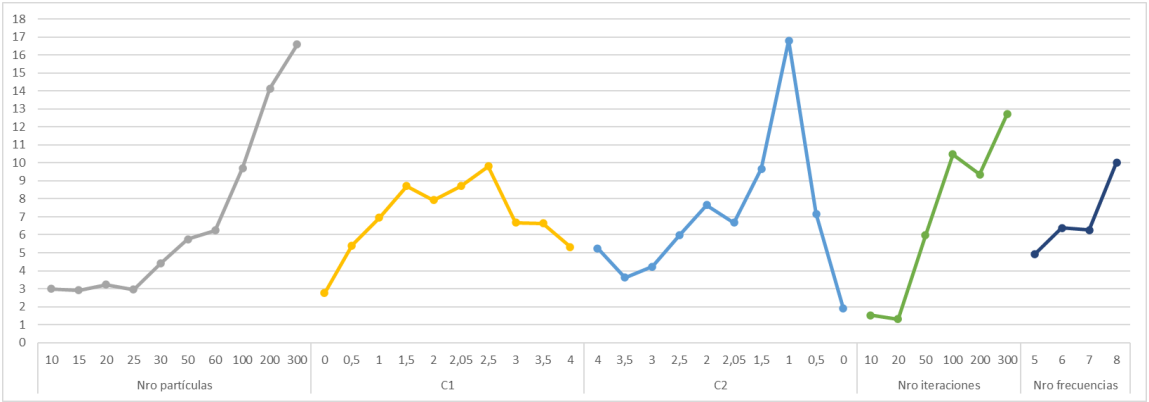


Figura 4-7: Efectos en la respuesta del modelo cuando el valor de los factores cambia.

Extrayendo los datos de la gráfica anterior, se concluye que la mejor configuración para el algortimo *PSO* es la siguiente:

Tabla 4-10: Mejor configuración del algoritmo *PSO* con el método *DOE.*

Nro de iteraciones	Nro de frecuencias	Nro de partículas	C1	C2
300	8	300	2.5	1

Tabla 4-11: Rendimiento de la mejor configuración del *DOE* en diferentes escenarios de daño.

Caso de daño		SR
Nro. elemento	Nivel de daño	
7	5%	1
2, 22	10%, 10%	1
5, 20, 22	40%, 20%, 10%	0
4, 13, 28	30%, 20%, 30%	0.6
13, 25, 30	10%, 10%, 20%	0.9

Al comparar el resultado de la mejor configuración de la prueba nro 1 del F-Race (**Tabla 4-2:** Mejores configuraciones del F-Race nro. 1. – Configuración 2) con el de la configuracion obtenida por el método *DOE* se observa que hay una discrepancia en los valores de los coeficientes c1 y c2. Esto lleva a una leve diferencia en el rendimiento de ambas configuraciones.

El no obtener el mismo resultados con ambos métodos radica en el hecho de que al aplicar el método *DOE* solamente 95 experimentos the todas las combinaciones posibles son testeados en solamente 6 instancias representando 0,47% del dominio experimental. Al ser este problema de optimización de parámetros un caso particular, no se tienen registros previos de la aplicación del método a un problema de optimización de parámetros *PSO*, por lo tanto, para mejorar su resultado podrían realizarse a futuro mejoras del programa *DOE*.

Una ventaja del *DOE* es que reduce la influencia de ciertos factores aleatoreos del algoritmo *PSO* en el resultado, ayudando así a estudiar con más detalle como los valores de los parámetros influyen realmente en su desepeño. Esto se logra gracias a que este método testea solo una pequña parte de todo el dominio experimental y con solo un modelo matemático predice la respuesta del sistema.

Capítulo 5: Conclusiones y recomendaciones

Conclusiones

Para realizar una correcta comparación de resultados y lograr comprender la influencia de cada uno de los parámetros, se tomaron las mejores configuraciones obtenidas a lo largo de este proyecto y se testearon en diversos escenarios de daños. Los mismos constan de situaciones donde la viga cuanta con un solo elemento dañado hasta tres, vairando la severidad de ellas y con diferentes distribuciones.

Al análisis se agragaron dos configuraciones más, por un lado, la combinación de parámetros que se presenta en el artículo de Xiao-Li (**Tabla 5-1**). Esta elección se debe a que el último estudio realizado por el *INSA* concluyó que esta misma configuración era la más óptima para el problema. Por otro lado, con el fin de corroborar que la ecuación (25), que propone una variación de los coeficientes de aceleración a través de las iteraciones, no aplica al problema particular de la localización de daño en una viga empotrada (**Tabla 5-2**).

Tabla 5-1: Configuración del artículo de Xiao-LI [2].

Nro. elementos	Nro. PSO	Precisión	V max	Nro. iteracines	Nro. frequencias	Nro. particulas	C1	C2
30	10	4	1	200	9	50	2	2

Tabla 5-2: Configuración nro 1 del *F-Race* nro 3 cambiando los valores constantes de c1 y c2 por la ecuación (25).

Nro. elementos	Nro. PSO	Precisión	V max	Nro. iteracines	Nro. frequencias	Nro. particulas	C1	C2
30	10	4	1	450	9	450	Ecuación (25)	

Se sabe que en los casos donde solamente hay uno o dos elementos dañados al algoritmo *PSO* tenie una mayor tasa de éxito. Esto se puede apreciar en la **Tabla 4-11**, en los tres primero casos todas las configuraciones, exceptuando la de Xiao-Li, poseen un rendimiento de más del 90%. Ahora bien, cuando el estado de la viga se encuentra más comprometido, con tres elementos dañados, el desempeño de las configuraciones

comienza a caer y, en algunos casos donde el nivel de daño varia ampliamente, la respuesta del sistema no es la deseable.

Los resultados de la configuración de la **Tabla 5-3** nos confirman que la ecuación (25) no es la adecuada para este problema en particular, se observan mejores resultados al imponer valores constantes en los coeficientes c1 y c2.

La pregunta principal de este proyecto sobre cual es la mejor combinación de parámetros que resuelvan el problema de encontrar el o los daños en una viga empotrada no es tan fácil de responder en base a los resultados de las pruebas realizadas. Si bien hay una combinación que aparenta ser la más adecuada, la configuración 1 de la prueba nro. 3 del *F-Race* que arroja en 4 casos un rendimiento del 100%, no se puede asegurar que sea la mejor solución. Si se observa el caso del último escenario de daño la configuración 1 de la prueba nro. 2 del *F-Race*, esta posee casi el doble de rendimiento. Cabe aclarar que ambas configuraciones poseen la misma combinación de parámetros exceptuando el valor de los coeficientes de aceleración.

No es tarea sencilla comparar los rendimientos de las configuraciones, dependiento del caso de daño este resultado varia de forma no predecible. Algo que se puede afirmar es que a mayores valores de números de partículas y número de frecuencias el rendimiento tiende a aumentar.

Tabla 5-3: Rendimiento de las diferentes configuraciones en diversos escenarios de daño.

Caso de daño		**SR**								
Nro. elemento	**Nivel de daño**	**F-Race nro. 1**		**F-Race nro. 2**		**F-Race nro. 3**		**DoE**	**Config de Xiao-Li**	**Config Tabla 14**
		Config 1	*Config 2*	*Config 1*	*Config 2*	*Config 1*	*Config 2*			
7	*5%*	1	1	1	1	1	1	1	0.7	1
2, 22	*10%, 10%*	1	1	1	1	1	1	1	0.6	0.9
25, 30	*10%, 20%*	1	1	1	0.9	1	1	0.9	0.1	0.7
5, 20, 22	*40%, 20%, 10%*	0	0.2	0.7	0.3	0.7	0.8	0	0	0.1
4, 13, 28	*30%, 20%, 30%*	0.5	0.8	0.9	0.7	0.8	0.8	0.6	0.3	0.6
13, 25, 30	*10%, 10%, 20%*	0.6	0.7	1	1	1	0.9	0.9	0.1	0.6
1, 11, 26	*30%, 10%, 40%*	0.2	0.3	0.9	0.4	0.5	0.6	0.1	0	0.6

Al estudiar estos resultados surge una nueva pregunta: ¿es que existe una configuración de parámetros de las ecuaciones del algoritmo del PSO que en cualquier escenario de daño tenga un rendimiento del 100%? Estudiando los resultados de la aparente mejor combinación obtenida con el *F-Race* nro. 3, en las 1007 instancias que tomó el programa en obtener un resultado, en ciertos escenarios de daño esta configuración no logró encontrar ninguna solución, su rendimiento fue nulo. Esto vuelve a poner en duda si realmente se encontró la mejor configuración, puede ser que existió en la prueba una configuración que siempre encuentró al menos una solución en todas las instancias que fue testeada, pero al no lograr la mayor cantidad de soluciones fue eliminada.

Con todo lo expuesto se concluye que se debe realizar un nuevo estudio con un enfoque diferente, buscar la combinación que para todos los escenarios de daño posibles siempre logre dar, aunque sea, una solución.

Recomendaciones

Como se expresó anteriormente, este proyecto no ha logrado encontrar la mejor combinación de parámetros del algoritmo PSO pero ayudó a encontrar una nueva línea de investigación para lograr dar con la respuesta correcta. Esta nueva prespectiva se deberá enfocar en encontrar la combinación de factores que logre siempre dar una solución, ya no se trata de obtener un rendimiendo del 100%. Una vez que se encuentre esta configuración se deberá enfocarse en aumentar su desempeño.

A la hora de seleccionar un método de optimización, ya sea el *F-Race* o el *DOE*, se recomienda seguir trabajando con el *F-Race* ya que es un algortimo con desarrollo previo y el más adecuado para aplicar la nueva estrategia. Por otro lado, se sugiere que se continue profundizando y estudiando la metodología *DOE* para futuras aplicaciones.

Bibliografía

[1] XIAO-LIN LI, ROGER SERRA, OLIVIER JULIEN. “Effects of the Particle Swarm Optimization parameters for structural dynamic monitoring of cantilever beam.“ Surveillance, Vishno and AVE conferences, INSA-Lyon, Université de Lyon, Jul 2019, Lyon, France. hal-02188562

[2] XIAO-LIN LI, ROGER SERRA, JULIEN OLIVIER. "Performance of Fitness Functions Based on Natural Frequencies in Defect Detection Using the Standard PSO-FEM Approach", *Shock and Vibration*, vol. 2021, Article ID 8863107, 9 pages, 2021. https://doi.org/10.1155/2021/8863107

[3] B. NANDA, D. MAITY, AND D. K. MAITI. “Vibration based structural damage detection technique using particle swarm optimization with incremental swarm size”. International Journal of Aeronautical and Space Sciences, 2012, vol. 13, no. 3, pp. 323–331,

[4] LI, XIAO-LIN, ROGER SERRA, JULIEN OLIVIER.. "An Investigation of Particle Swarm Optimization Topologies in Structural Damage Detection". 2021. Applied Sciences 11, no. 11: 5144. https://doi.org/10.3390/app11115144

[5] ZEPENG CHEN, LING YU. “A new structural damage detection strategy of hybrid PSO with Monte Carlo simulations and experimental verifications” Measurement, Volume 122, 2018, Pages 658-669, ISSN 0263-2241. https://doi.org/10.1016/j.measurement.2018.01.068

[6] MAURO BIRATTARI, THOMAS STÜTZLE, LUIS PAQUETE, AND KLAUS VARRENTRAPP. “A racing algorithm for configuring metaheuristics. In Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation (GECCO'02)”. Morgan Kaufmann Publishers Inc., 2002, San Francisco, CA, USA, 11–18.

[7] BIRATTARI, M., YUAN, Z., BALAPRAKASH, P., STÜTZLE, T. “F-Race and Iterated F-Race: An Overview.” In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (eds) Experimental Methods for the Analysis of Optimization Algorithms. Springer, Berlin, Heidelberg, 2010. https://doi.org/10.1007/978-3-642-02538-9_13

[8] YAN HE, WEI JIN MA AND JI PING ZHANG. “The Parameters Selection of PSO Algorithm influencing on performance of Fault Diagnosis” MATEC Web Conf., 63, 2016, 02019. https://doi.org/10.1051/matecconf/20166302019

[9] PELLOQUIN, L. Rapport de stage “Optimisation des paramètres de l’algorithme PSO“. 2020-2021

[10] JIA GUO, DEQING GUAN, JIANWEI ZHAO. "Structural Damage Identification Based on the Wavelet Transform and Improved Particle Swarm Optimization Algorithm", Advances in Civil Engineering, vol. 2020, ArticleID 8869810, 19 pages, 2020. https://doi.org/10.1155/2020/8869810

[11] Jiawei Xiang, Ming Liang, “A two-step approach to multi-damage detection for plate structures”, Engineering Fracture Mechanics, Volume 91, 2012, Pages 73-86, ISSN 0013-7944. https://doi.org/10.1016/j.engfracmech.2012.04.028.

[12] SHABBIR, FAISAL, MUHAMMAD I. KHAN, NAVEED AHMAD, MUHAMMAD F. TAHIR, NAEEM EJAZ, AND JAWAD HUSSAIN. "Structural Damage Detection with Different Objective Functions in Noisy Conditions Using an Evolutionary Algorithm". 2017. *Applied Sciences* 7, no. 12: 1245. https://doi.org/10.3390/app7121245

[13] ABDULHUSSEIN, R. T., & M., M. A. “Damage detection in composite plate based on vibration Measurements using Genetic Algorithm.” 2017. Al-Nahrain Journal for Engineering Sciences, 20(3), 709–718. Retrieved from https://www.nahje.com/index.php/main/article/view/274

[14] L.A. ROMÁN-RAMÍREZ, J. MARCO. “Design of experiments applied to lithium-ion batteries: A literature review.” Applied Energy, Volume 320, 2022, 119305, ISSN 0306-2619. https://doi.org/10.1016/j.apenergy.2022.119305

[15] TRIEFENBACH, FABIAN. “Design of Experiments: The D-Optimal Approach and Its Implementation As a Computer Algorithm”. 2008. Bachelor's Thesis in Information and Communication Technology, 15 ECTS

[16] P.F. DE AGUIAR, B. BOURGUIGNON, M.S. KHOTS, D.L. MASSART, R. PHAN-THAN-LUU. “D-optimal designs: Chemometrics and Intelligent Laboratory Systems”. Volume 30, Issue 2, 1995, Pages 199-210, ISSN 0169-7439. https://doi.org/10.1016/0169-7439(94)00076-X

[17] L. DELPLANQUE, F.LOUVET. “Les Plans d'Expériences : une approche pragmatique et illustrée“, 2005

[18] FEDERICO MARINI, BEATA WALCZAK. “Particle swarm optimization (PSO). A tutorial, Chemometrics and Intelligent Laboratory Systems.” Volume 149, Part B, 2015, Pages 153-165, ISSN 0169-7439. https://doi.org/10.1016/j.chemolab.2015.08.020.

[19] IOAN, CRISTIAN, TRELEA. "The particle swarm optimization algorithm: convergence analysis and parameter selection." Information Processing Letters, 85 (2003).:317-325. doi: 10.1016/S0020-0190(02)00447-7

[20] Apunte cátedra: Comportamiento y ensayos de materiales M20. “Introducción a los ensayos no destructivos”, 2020

Anexo A: algoritmo de PSO solo completo en MATLAB

"pso_alone_1.mat":

```
clear all
close all hidden
clc
rng('shuffle')

%% SETTING
%Parameters
nbElements  = 30; % Elements of the beam / space of research
NbFreq=9;   %Number of frequencies identified
precision=4;

nbMaxPSO    = 10;   % Number of PSO to execute
Maxiter     = 450; % Number of iterations
Npar        = 450;  % Number of particles
c1          = 3.5;   % Cognitive acceleration
c2          = 0.5;   % Social acceleration

%Boundary of the space problem
BoundaryMin = 0;
BoundaryMax = 1;
Boundary    = repmat([BoundaryMin BoundaryMax],nbElements,1);
% k=rand*nbElements;
Vmax = 1; %BoundaryMax. BoundaryMax*ones(Npar,nbElements); %((BoundaryMax-
BoundaryMin)/2)*k;

%Linearly decreasing inertia weight
Wmax        = 0.9;
Wmin        = 0.4;

% Pré-allocation
population    = zeros(nbMaxPSO,nbElements);
tempsdecalcul = zeros(nbMaxPSO,1);
fitGlobal     = zeros(nbMaxPSO,1);
itersol       = zeros(nbMaxPSO,1);

%% CASE
% level and damage location
instance = zeros(1,nbElements);
instance (1,7) =0.3;
instance (1,15) =0.1;
instance (1,18) =0.4;


% Reference  structure
[~,freqRef] = Reference_case_1(nbElements,NbFreq,precision,instance);
% % [~,freqRef, mode]
% freqRef is the frequency allowing to compare the unknown solution WRT the exact one
instance_undamage=zeros(1,nbElements);
[~,f_undamage] = Reference_case_1(nbElements,NbFreq,precision,instance_undamage);
%f_undamage is the frequency of a undamage structure use to have the fitness
% % [~,f_undamage,mode_und]

%%
for nbPSO = 1:nbMaxPSO
    tic
    %
    X           = zeros(Npar,nbElements);
    V           = zeros(Npar,nbElements);
    %
```

```
    Pbest       = zeros(Npar,nbElements);
    Gbest       = zeros(1,nbElements);
    Gbest_old   = 1E+12;
    %
    Fitness     = zeros(Npar,1);
    Fitness_old = zeros(Npar,1);
    Fitness_old(:,1) = 1E+10;

    %% INITIALIZATION
    for i=1:Npar
        %Inialization of V and X
        for j=1:nbElements
            Pbest(i,j) = round(rand,2);%*nbElements;      %*(30-0)mettre ici le nombre
d'element à la place de 30 ?
            V(i,j)     = 0; %BoundaryMin+round(rand,2)*(BoundaryMax-BoundaryMin);%0
%;BoundaryMin+rand*(BoundaryMax-BoundaryMin);      %*0.5*(30-0)Pourquoi 0.5 et 30 ?
            X(i,j)     = BoundaryMin+round(rand,2)*(BoundaryMax-BoundaryMin);
%roundn(rand,-2);%Npar*rand;      %*0.5*(30-0);
        end
        %Initial Gbest
        if
AimFcn_1(Pbest(i,:),nbElements,NbFreq,precision,f_undamage,freqRef)<Gbest_old
            Gbest     = Pbest(i,:);
            Gbest_old =
round(AimFcn_1(Gbest,nbElements,NbFreq,precision,f_undamage,freqRef),2);
        end
    end
       %% PSO optimization
    iter = zeros(1,Maxiter);
    it=0;
    for generation=1:Maxiter
        %Linearly decreasing inertia weight
        W=(Wmax-((Wmax-Wmin)*generation)/Maxiter);

        if c1&c2==12
            c1=2.5+(0.5-2.5)*(generation/Maxiter);
            c2=0.5+(2.5-0.5)*(generation/Maxiter);
         end
        %
        for i=1:Npar
            for j=1:nbElements

                % V calculation
                r1=rand;
                r2=rand;
                V(i,j)=W*V(i,j)+c1*r1*(Pbest(i,j)-X(i,j))+c2*r2*(Gbest(1,j)-X(i,j));
                %Test if V<Vmax
                if abs(V(i,j))>Vmax %Vmax(1,j)
                    V(i,j)=sign(V(i,j))*Vmax;
                end

                %Update the particle position
                X(i,j)=X(i,j)+V(i,j);

                %Test if X<Xmax
                if (X(i,j) < BoundaryMin)
                    X(i,j)=BoundaryMin;  %+cor*rand*(BoundaryMax-BoundaryMin);
                end
                if (X(i,j) > BoundaryMax )
                    X(i,j)=BoundaryMax;  %+cor*rand*(BoundaryMax-BoundaryMin);
                end
            end
            X(i,:)=roundn(X(i,:),-2); %the level of damage only takes values 0.05 and
0.20 aprox.

            %FITNESS FUNCTION CALCULATION

Fitness(i,1)=feval('AimFcn_1',X(i,:),nbElements,NbFreq,precision,f_undamage,freqRef);
%             Fitness(i,1) = real(Fitness(i,1));
            %Pbest - best position of the particle in the current iteration
```

```
            if Fitness(i,1)<Fitness_old(i,1)
                Pbest(i,:) = X(i,:);
                Fitness_old(i,1)=Fitness(i,1);
            end
        end
        %Selection of the best particle of the all the swarm
        [Gbest_new, Pos] = min(Fitness);
        if Gbest_new<Gbest_old
            Gbest_old=Gbest_new;
            Gbest=X(Pos,:);
        end
        %If the solution is found stop iterations
        if Fitness==-1 & it==0 %0 for S1, S2, S3, S5, S7 ; -1 for S4
           itersol(nbPSO)=generation;
           it=1;
            break
        end
    end

    % Best solution(POSITION) of each nbPSO is kept in vector population
    population(nbPSO,:)=Gbest;
    %Fitness value of the best solution of each nbPSO

fitGlobal(nbPSO)=AimFcn_1(population(nbPSO,:),nbElements,NbFreq,precision,f_undamage,fr
eqRef);
    %
    nbPSO
    %
    toc
    tempsdecalcul(nbPSO,1)=(toc);

end

%% RESULTS
[b,bi]=min(fitGlobal);
% b - error value
% bi - index of the best solution of the population
time=tempsdecalcul(bi,1);

%% GRAPHICS
subplot(1,2,1),plot(fitGlobal)
xlabel('nbPSO')
ylabel('Fitness Function of best sol')
xticks(0:15:100)
%Meilleure solution trouvée
sol=population(bi,:);
% AimFcn_1(sol);
subplot(1,2,2),bar(sol)
xlabel('nbElements')
ylabel('Damage vector')
xticks(0:1:30)

%% Solution exacte
% solExact=AimFcn(bExact)

%% NUMBER OF REAL SOLUTIONS FOUND IN EACH PSO
nb_sol = 0;
nb_false=0;
errTol = -0.999; %for S4: -0.999, for S1,S2, S3, S6, S7: 1E-4,
L_fitg = fitGlobal <= errTol;

for i = 1:nbMaxPSO
    if L_fitg(i) == 1
        comp =abs(population(i,:)-instance); %to have know if we achived a good result
this difference should be <=0.0001 (1E-4)
        if sum(comp <= 1E-2) == nbElements %to count as a solution the %30 elements
should have difference <=0.0001 (1E-4)
            nb_sol = nb_sol + 1;
        else
            nb_false=nb_false+1;
```

```
        end
    end
end

%% SHOWN VALUES
%time to reach that solution
time
%average iter
if nb_sol~=0
aviter=sum(itersol)/nb_sol
end
%total time
tottime=sum(tempsdecalcul)
%value of the solution (it has to be 0 or -1)
b
%number of solutions in total
nb_false
nb_sol


%% SAVE
%
save(filename,'b','nb_false','nb_sol','tottime','aviter','population','instance','fitGl
obal')
```

"Reference_case_1.mat":

```
function [x,freq] = Reference_case_1(nbElements,NbFreq,precision,instance)
% % [x,freq,Vect_prop_norm]
%% Returns the NATURAL FREQUENCY by applying FEM method on a reference cantilever beam
%% WRT given defect location and severity

%% REFERENCE CASE CONSIDERED
x = instance'; % Considered defected or not beam/instance

% Number of sensors
nbcapteur = NbFreq; % max(size(Fref)), Number of frequencies measured
nbre_deg_noeud = 2;

% Number of nodes
nb_noeuds = nbElements + 1; % 5 doigts et 4 sép

%% EULER-BERNOULLI BEAM CARACTERISTICS (RECTANGULAR SECTION)
Eyoung = 2E11; %Pa ou N/m2, STEEL
L_tot = 1; %m
rho = 7850; %kg/m3, density
h = 0.0053; %m
b = 0.0249; %m
%nu = 0.3;
S = b*h; %m2, surface
I = (b*h^3)/12; %m4, quadratic moment
L = L_tot/nbElements; %m, partition length

%% ELEMENTARY MATRICES
%E = zeros(nbElements,1);
E = Eyoung*(1-x)'; % Error taken into account, transposed into an array

% Elementary stiffness
Ke2 = (I/L^3)*[12   6*L    -12   6*L;
               6*L  4*L^2  -6*L  2*L^2;
               -12  -6*L   12    -6*L;
               6*L  2*L^2  -6*L  4*L^2];

% Elementary mass
Me = ((rho*S*L)/420)*[156   22*L   54     -13*L;
                      22*L  4*L^2  13*L   -3*L^2;
```

```
                      54    13*L   156    -22*L;
                      -13*L -3*L^2 -22*L  4*L^2];

sizeKe_totale = nbre_deg_noeud*nb_noeuds;

K_rigidite = zeros(sizeKe_totale,sizeKe_totale);
M_masse = zeros(sizeKe_totale,sizeKe_totale);

%% ASSEMBLY
for j = 1:1:nbElements
    Ke = E(j)*Ke2;
    for k = 1:1:4
      for n = 1:1:4
        K_rigidite(n+(j-1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud)...
        = K_rigidite(n+(j-1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud) + Ke(n,k);
        M_masse(n+(j-1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud)...
        = M_masse(n+(j-1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud) + Me(n,k);
      end
    end
end

%% FREE IMBED BEAM

K_rigidite = K_rigidite(3:end,3:end);
M_masse = M_masse(3:end,3:end);
%%
% K_rigidite(1:2,:)=[];
% K_rigidite(:,1:2)=[];
% M_masse(1:2,:)=[];
% M_masse(:,1:2)=[];
% %% MODAL ANALYSIS
%
% % Vect_prop = real(Vect_prop);
% Val_prop = real(Val_prop);
%
% % [Val_prop_Range,Indice] = sort(diag(Val_prop));% définition de l'ordre des indices
% [Val_prop_Range,~] = sort(diag(Val_prop));% définition de l'ordre des indices
%
%
% Puls_prop = Val_prop_Range.^(1/2) ;     % Natural frequency (rad/s)
% freqHz = Puls_prop/(2*pi) ;             % UNITS : Hertz
%
% freq = round(freqHz(1:nbcapteur,:),precision);
[Vect_prop,Val_prop] = eig(K_rigidite,M_masse);
Vect_prop=real(Vect_prop); %phi, mode shape
Val_prop=real(Val_prop); %lamnda (w²), frequency

[Val_prop_Range,Indice] = sort(diag(Val_prop)); % définition de l'ordre des indices
Vect_prop_range = Vect_prop(:,Indice); % Matrice ordonnée (Vecteurs propres)

Puls_prop = Val_prop_Range.^(1/2) ;% Pulsation propre (en rad/s)
freqHz = Puls_prop./(2*pi) ; % UNITS : Hertz
freqHzz=freqHz(1:nbcapteur,:);%nbcapteur,:);%

% Vect_prop_rang=Vect_prop_range(1:nbcapteur,1:nb_noeuds);
% %normalization
% Vect_prop_norm=Vect_prop_rang/norm(Vect_prop_rang);

freq=round(freqHzz,precision);
```

"AimFcn_1.mat":

```
function [err,freqHzz]=AimFcn_1(x,nbElements,NbFreq,precision,f_undamage,freqRef)

%% Case
Fref=freqRef;
```

```
%Nombre de DDL par noeud
nbre_deg_noeud=2;
%Nombre de noeuds
nb_noeuds=nbElements+1;
%Nombrer de capteur
nbcapteur=NbFreq;

%% EULER-BERNOULLI BEAM CARACTERISTICS (RECTANGULAR SECTION)
Eyoung=200e+009;%N/m2
L_tot = 1;%m
rho =  7850;%kg/m3
h  = 5.3e-003;%m
b  = 24.9e-003;%m
S   = b*h;%m
I  = (b*h^3)/12;%m4
L=L_tot/nbElements;%m

%% ELEMENTARY MATRICES
% Règle: - mettre -1 si pas d'endommagement
E=zeros(nbElements,1);
E=Eyoung*(1-x)';

% Elementary stiffness
Ke2=(I/L^3)*[
        12 6*L -12 6*L;
    6*L 4*L^2 -6*L 2*L^2;
    -12 -6*L 12 -6*L;
    6*L 2*L^2 -6*L 4*L^2];

% Elementary mass
Me=(rho*S*L/420)*[
        156 22*L 54 -13*L;
        22*L 4*L^2 13*L -3*L^2;
        54 13*L 156 -22*L;
        -13*L -3*L^2 -22*L 4*L^2];

sizeKe_totale=nbre_deg_noeud*nb_noeuds;
K_rigidite=zeros(sizeKe_totale,sizeKe_totale);
M_masse=zeros(sizeKe_totale,sizeKe_totale);

%% ASSEMBLY
for j = 1:1:nbElements
    Ke = E(j)*Ke2;
    for k = 1:1:4
      for n = 1:1:4
        K_rigidite(n+(j-1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud) = K_rigidite(n+(j-
1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud) + Ke(n,k);
        M_masse(n+(j-1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud) = M_masse(n+(j-
1)*nbre_deg_noeud,k+(j-1)*nbre_deg_noeud) + Me(n,k);
      end
    end
end
%
K_rigidite = K_rigidite(3:end,3:end);
M_masse = M_masse(3:end,3:end);

%% MODAL ANALYSIS
[~,Val_prop] = eig(K_rigidite,M_masse);

% Vect_prop=real(Vect_prop); %phi, mode shape
Val_prop=real(Val_prop); %lamnda (w²), frequency

[Val_prop_Range,~] = sort(diag(Val_prop)); % définition de l'ordre des indices
% Vect_prop_Range = Vect_prop(:,Indice); % Matrice ordonnée (Vecteurs propres)

Puls_prop = Val_prop_Range.^(1/2) ;% Pulsation propre (en rad/s)
Puls_prop = real(Puls_prop);
freqHz = Puls_prop./(2*pi) ; % UNITS : Hertz

freqHzz=freqHz(1:nbcapteur,:);%nbcapteur,:);%
```

```
freqHzz=round(freqHzz,precision);

%% FITNESS FUNCTIONS  % BE CAREFUL ** CHANGE THE VALUE OF FITNESS IN 'pso_alone_1' IF
REQUIRED (0 or -1)**
% %     S1 Xiao-Lin Li article - 0
% err = sum(((Fref.^2-freqHzz.^2)./(Fref.^2)).^2);
% %     S2  Xiao-Lin Li article
% err=sum(((Fref-freqHzz).^2).^2./(Fref).^2);

% %     S3  Xiao-Lin Li article - 0
% err = sqrt((1/nbcapteur)*sum((Fref./freqHzz)-1).^2);

% %     S4 Xiao-Lin Li article - -1
delta_f1=(f_undamage-Fref)./f_undamage;
delta_f2=(f_undamage-freqHzz)./f_undamage;
error4_1=(delta_f1'*delta_f2)^2/((delta_f1'*delta_f1)*(delta_f2'*delta_f2));
f_new=[freqHzz,Fref];
error4_2=(1/nbcapteur)*sum((min(f_new,[],2))./(max(f_new,[],2)));
err=(-1/2*(error4_1+error4_2));


% % S5 euclidean norm - 0
%  err=sum(norm((Fref-freqHzz)/norm(Fref)));

% % S6 - 0
% calc=(freqHzz-Fref)./Fref;
% err=sum(calc.^2);

% % S7 - 0
% err=sum((Fref-freqHzz).^2);
```

Anexo B: algoritmo del F-Race completo en MATLAB

"F_racek2.mat":

```
clear all
clc
close all
rng('shuffle')
%%                              AIM                                 %%
%    DETERMINE WHICH CONFIGURATION FITS THE PSO TO FIND THE    %
%%% NUMBER OF DAMAGES,THEIR POSITION AND THEIR SEVERITY ON A %%%
%%%%%%%%%%%%%%%%%%%%%%%%%% GIVEN BEAM %%%%%%%%%%%%%%%%%%%%%%%%%%
%% --- DATE AND TIME ---
d = datestr(clock,'dd/mm/YYYY_HH:MM:SS:FFF');
d = strrep(d,'/','-');
d(strfind(d,':')) = ['h' 'm' 's'];
%% --- LAUNCHING NUMBER & FOLDER ---
%create the folders with data
load('Lancement_num');
ex=1;
ex=exist ('Restart_Launches'); %if folder Restart_Launches exist -> ex=7
launch=a;
BasePath = pwd;
folder = fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',launch,d));
mkdir (folder)
folder1 = fullfile(BasePath,'Restart_Launches',sprintf('%d_Date&Launch',launch));
mkdir (folder1)
file = fullfile(folder1,'DATE&LAUNCH');
save (file,'d')

a = a + 1;
save('Lancement_num','a')

%% only if it is a launch after a crash -- load of necesary files
if ex==7; %if there is a crash in the MobeXtreme and we need all the data already save

A=a-2;
load(fullfile(BasePath,'Restart_Launches',sprintf('%d_Date&Launch',A),'DATE&LAUNCH'))
%load date and nb launch
load(fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',A,d),'Instances_set.ma
t')) %load the instances
file = fullfile(folder,'Instances_set');
save (file,'instances')
load(fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',A,d),'last.mat'))
%load k (last instance) and nber_config (amount of config left)

%load the last cost configuration and the set of configuratios
R=dir(fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',A,d),sprintf('Instanc
e_%d_nbConfig_%d',k,nber_config)))
s=size(R);
r=R(3:s(1));
r=struct2table(r);
s1=size(r);
r1=r(:,1);
r2=table2cell(r1);
r3= string(r2);
NB_rstart=zeros(1,s1(1));

for i=1:s1(1)
NB_rstart(i)=sscanf(r3(i),'last_configs_testPSO%d'); %takes the number out of the name
of the file
end
```

```
n=[1:nber_config];
N=setdiff(n,NB_rstart);%compare and gives the remaining configurations

k_ind=k;
end

%% if there is no crash, restart folder not used
if ex~=7; %Restart folder do not exist, no need to search for previous data saved

%% --- PARAMETERS ---
  % --  PSO  --
% Number of PSO - MAX: 1000
Nb_pso =[10];
% Number of iteration
Nb_iter = [400 450 500]; % Number of iteration in the PSO Loop
% Number of particles
Nb_par = [350 400 450];
% Coginitive & Social acceleration
C1 = [0 0.5 1 1.5 2 2.5 3 3.5 4];
C2 = Accelerations(C1);
% Maximum velocity of the particles
Vmax = 1;

    %-- PROBLEM PARAMETERS --
Nb_elem = 30; % Number of partition - MAX: 100
precision =4; % Precision shouldn't be high, wouldn't make any sense in real situations
Nb_default=3;
Nb_freq = [8,9];

    % --- F-RACE PARAMETERS ---
% number of intances
I = 1000;

%% Combination of parameters - CONFIGURATIONS
config_set_1 = Allcomb(Nb_elem,Nb_pso,Nb_iter,Nb_freq,precision,Nb_par,C1,C2,Vmax); %
Finite set of candidate configurations
[nber_config_1,~] = size(config_set_1);

% criteria for coef accel --> C1+C2<4 or C1+C2<2*(2+2W)
Wmin=0.4; % inertia min during PSO
for i= nber_config_1:-1:1
  if config_set_1(i,7)+config_set_1(i,8)>=4 %C1+C2<4 or C1+C2<2*(2+2W)
     config_set_1(i,:)=[];
  end
end
[nber_config_1,~] = size(config_set_1); % needed here because some config are discard
if C1+C2<2*(2+2W) is not true

% c1 and c2 equation configurations - add one more variable for the F-Race
c1=12;
c2=12;
config_set_2 = Allcomb(Nb_elem,Nb_pso,Nb_iter,Nb_freq,precision,Nb_par,c1,c2,Vmax);
[nber_config_2,~] = size(config_set_2);
config_set=[config_set_1;config_set_2];

[nber_config,nber_parameters] = size(config_set);
Configurations = {'Nb_elem', 'Nb_pso', 'Nb_iter', 'Nb_freq', 'precision', 'Nb_par',
'c1', 'c2', 'Vmax'};
for i = 1:nber_config
    for j = 1:nber_parameters
        Configurations{i+1,j} = config_set(i,j); %% matrix with tittle
    end
end

% Preallocation to promote fast computation
cost_config = zeros(1,nber_config);
Nb_solutions = zeros(1,nber_config);

% save initial config
file = fullfile(folder,'Initial_Configurations');
```

```
save (file,'Configurations')

%% --- DAMAGE CASE ---
%LEVEL AND POSITION OF DAMAGE
instances = Instance_generation1(I,Nb_elem,Nb_default);
%save cases of damage - one case for each I
file = fullfile(folder,'Instances_set');
save (file,'instances')
%
k_ind=1;
nr_discard=0;
end
rest=0;

%% F-RACE LOOP
alpha = 0.05; % 5% level
for k = k_ind:I %%if restart k_ind
    [nber_config,~] = size(config_set);

    %creating folder nbI and nbConfig that are being test
    new_f = fullfile(folder,sprintf('Instance_%d_nbConfig_%d',k,nber_config));
    mkdir (new_f)
    fileI = fullfile(folder,'last.mat');
    save(fileI,'k','nber_config','I','config_set','cost_config','Configurations')

    cprintf('key','Instance n');
    disp(num2str(k));
    cprintf('key','Number of configurations in the race: ');
    disp(num2str(nber_config));

    %%--Cost of all the configurations left on I=k--%%
     cost_config(k,1:nber_config) = NaN;     %initialization
     Nb_solutions(k,1:nber_config)= NaN;

     if ex==7 & rest~=1 % if it is a RE-START/CRASH
%       parfor
        for nb = 1:nber_config
            restart_PSO(config_set,instances,nb,N,k,new_f,I,nber_config); %only
calculate remainding configurations
        end
     else %if it is NOT CRASH
%       parfor
        for nb = 1:nber_config
            disp(num2str(nb));
            [cost_config(k,nb),Nb_solutions(k,nb)]=
PSO(config_set(nb,:),instances(k,:));

parforsave2(fullfile(new_f,sprintf('last_configs_testPSO%d',nb)),cost_config(k,nb),conf
ig_set(nb,:),k,nber_config,nb)
        end
     end

    if ex==7 & rest~=1 % RECONSTRUCTION OF cost_config and config_set ONLY AFTER A
CRASH
        cost_config_r = zeros(1,nber_config);
        for nro=NB_rstart

load(fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',A,d),sprintf('Instance
_%d_nbConfig_%d',k,nber_config),sprintf('last_configs_testPSO%d',nro)))
                cost_config_r(1,nro)=cost_config; %load in the matrix the config of the
folder of the RESTART_LAUNCHES
        end

        b=a-1;

load(fullfile(BasePath,'Restart_Launches',sprintf('%d_Date&Launch',b),'DATE&LAUNCH'))
        d1=d;

C=dir(fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',b,d1),sprintf('Instan
ce_%d_nbConfig_%d',k,nber_config)))
```

```
        s=size(C);
        C=C(3:s(1));
        c=struct2table(C);
        s1=size(c);
        c1=c(:,1);
        c2=table2cell(c1);
        c3= string(c2);
        NB_rstart2=zeros(1,s1(1));
        for i=1:s1(1)
        NB_rstart2(i)=sscanf(c3(i),'last_configs_testPSO%d');
        end
        for nro=NB_rstart2

load(fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',b,d),sprintf('Instance
_%d_nbConfig_%d',k,nber_config),sprintf('last_configs_testPSO%d',nro)))
                cost_config_r(1,nro)=cost_config; %%load in the matrix the config of
the running launch
        end

load(fullfile(BasePath,'Restart_Launches',sprintf('%d_Launch_%s',b,d),'last.mat'))
        cost_config(k,:)=cost_config_r;

    end
    rest=1;


    %%--Ranking each config in the race on the I=k--%%
    Blocks_rank = tiedrank(cost_config, length(size(cost_config)));
    if k ~= 1
            Sum_rank = sum(Blocks_rank(:,:)); % Sum of the ranks of each configuration
in the race over all instances
        else
            Sum_rank = Blocks_rank; % No summation needed
    end
   if k>3
    % TEST STATISTIC OF FRIEDMAN %
    num_sum = (Sum_rank - (k*(nber_config+1))/2).^2;
    num = (nber_config-1)*sum(num_sum);                                    % Numerator
    den = sum(Blocks_rank(:).^2) - (k*nber_config*(nber_config+1)^2)/4; % Denominator
    T = num/den;

    if T > (1-alpha)    %(1-alpha/2) %dof+2*sqrt(2*dof)*erfinv(2*(1-alpha)-1)
       % Best configuration (lowest expected rank)
       [Best_rank,best_conf] = min(Sum_rank);

        % POST-HOC TEST - CONOVER
            den_comp = sqrt(2*k*(1-(T/(k*(nber_config-1))))*den);   %den_comp =
sqrt(2*k*(1-(T/(k*(nber_config-1))))*den/((k-1)*(nber_config-1)))
            Best_Sum_rank = Best_rank*ones(1,length(Sum_rank));
            Comparison = abs(Best_Sum_rank - Sum_rank)*sqrt((k-1)*(nber_config-
1))/den_comp;
            tpdf = sqrt(2)*erfinv(2*(1-alpha/2)-1);  %% we have more than 30 datas,
Normal distribution used


         %%--Discard all the sub-optimal configurations and their cost from the set--%
            L = Comparison > tpdf;              % Logical: 1 if true, 0 if false
            %%%eliminate bad configurations%%%
            config_set(L==1,:) = [];
            V = [0 L];
            Configurations(V==1,:) = [];
            cost_config(:,L==1) = [];
            Nb_solutions(:,L==1)= [];
      end
    end
%%-- Number of configurations still in the race --%
    mem = nber_config;
    [nber_config,nber_parameters] = size(config_set); % nber_config is the number of
configurations in the race
    discard_config = mem - nber_config;
```

```
    cprintf([1,0.5,0],'Number of configurations discarded: ');
    disp(num2str(discard_config));
    cprintf([1,0.5,0],'\n');
    pause(1)

    for i = 1:nber_config
        for j = 1:nber_parameters
        Configurations{i+1,j} = config_set(i,j);
        end
    end

    if nber_config <= 2 % When the 2 best configurations survive it is finish
        break
    end

k_ind=1;
end

[nber_config,nber_parameters] = size(config_set);
Configurations = {'Nb_elem', 'Nb_pso', 'Nb_iter', 'Nb_freq', 'precision', 'Nb_par',
'c1', 'c2', 'Vmax'};
    for i = 1:1:nber_config
        for j = 1:nber_parameters
        Configurations{i+1,j} = config_set(i,j);
        end
    end

%% Delete folder for the restart - %% Final results
folder_end = fullfile(BasePath,'Launch_end',sprintf('%d_Launch_final_result',launch));
mkdir (folder_end)

copyfile Restart_Launches Launch_end
rmdir Restart_Launches s

if nber_config == 1
    cprintf('-[1,0,1]','Best configuration:\n\n');
    file = fullfile(folder_end,'Best_Configuration');
else
    cprintf('-[1,0,1]','Best configurations:\n\n');
    file = fullfile(folder_end,'Best_Configurations');
end
display(Configurations);

file_end = fullfile(folder_end,'Final_Results');
save (file_end,'Configurations','cost_config')
```

"Accelerations.mat":

```
function [c2] = Accelerations(c1)

c2 = 4*ones(1,length(c1)) - c1;

L = c1 == 2.05;
c2(L==1) = 2.05;
```

"Instance_generation1.mat":

```
function [instances] = Instance_generation1(I,Nb_elem,Nb_default)

instances = zeros(I,Nb_elem); % Structures with no defect
% def_nber = round(Nb_elem*rand(I,1)); % Number of defects on each instance

for i = 1:I
    % pour avoir instance avec moins de default voir aucun
```

```
    if randi(5)==1              % on priviligie le nombre maximum de dÃ©faut
        Nb_defaults=randi([0,Nb_default]);
    else
        Nb_defaults=Nb_default;
    end

    defaults = zeros(1,Nb_defaults);

    for j=1:Nb_defaults
        verif=0;
        while verif~=1      % on s'assure d'avoir le nombre de defaut voulu en
vÃ©rifiant qu'il n'y en a pas deja un
            def=randi([1 Nb_elem]);
            verif=1;
            for i2=1:j
                if def==defaults(i2)
                    verif=0;
                end
            end
        end
        defaults(j)=def;
        instances(i,def) = randi(4)/10; % defauts d'importance variable
    end
end
```

"PSO.mat":

```
function [b,Nb_solutions] = PSO(config,instance) % Configuration -
Nb_elem,Nb_pso,Nb_iter,Nb_freq,prec,Nb_par,C1,C2,w,Vmax
rng('shuffle')

a = 0;

load('InitXPos','position')
pos = position;

Configuration = {'Nb_elem', 'Nb_pso', 'Nb_iter', 'Nb_freq', 'precision', 'Nb_par',
'c1', 'c2', 'Vmax'};
for i = 1:length(config)
    Configuration{2,i} = config(i);
end

%% INITIALIZATION
% VARYING PARAMETERS
nbElements = config(1); % Swarm size / BEAM FEM

Maxiter = config(3); % Number of iterations
Npar = config(6); % Number of particles

NbFreq = config(4); % Number of frequencies identified (5 or 9)

c1 = config(7); % Cognitive Acceleration
c2 = config(8); % Social Acceleration

% FIXED PARAMETERS
precision = config(5); %
nbMaxPSO = config(2); % Number of PSO to execute
Wmax        = 0.9; % Inertial weight Max
Wmin        = 0.4; % Inertia weight Min

%% CASE - REFERENCE DEFECTED AND HEALTHY STRUCTURE
[~,freqRef] = Reference_case_1(nbElements,NbFreq,precision,instance); % freqRef is the
frequency allowing to compare the unknown solution WRT the exact one
instance_undamage=zeros(1,nbElements);
[~,f_undamage] = Reference_case_1(nbElements,NbFreq,precision,instance_undamage);
%f_undamage is the frequency of a undamage structure use to have the fitness

%% PROBLEM DIMENSION: boundary of the space problem
```

```
BoundaryMin = 0;
BoundaryMax = 1;
Bound_min = BoundaryMin*ones(Npar,nbElements);
Bound_max = BoundaryMax*ones(Npar,nbElements);

Vmax = config(9);
Vmax = Vmax*ones(Npar,nbElements); % Problem boundary

%% Preallocation des variables globales
population = zeros(nbMaxPSO,nbElements);
fitGlobal = zeros(nbMaxPSO,1);
% itersol       = zeros(nbMaxPSO,1);
% tempsdecalcul = zeros(nbMaxPSO,1);
%%
% matlabpool close;
% c = parcluster; matlabpool(c)

%% PSO LOOP
%parfor
for nbPSO = 1:nbMaxPSO
%   tic
    %
    X = zeros(Npar,nbElements);
    V = ones(Npar,nbElements);
    %
    %Inialization of V and X
    X = pos{nbPSO}(1:Npar,1:nbElements);
    V = 0;%round(rand(Npar,nbElements),2).*X;
    %
    Pbest = zeros(Npar,nbElements);
    Pbest = X; % Personal best

    % Preallocation
    Fitness = zeros(Npar,1);
    Fitness_old = 1E+10*ones(Npar,1);
    %
    Gbest = zeros(1,nbElements);
    Gbest_fit = 1E+12;
    %
    for i=1:Npar
        %Initial Gbest
        test_Gbest =
AimFcn_1(Pbest(i,:),nbElements,NbFreq,precision,f_undamage,freqRef);
        if test_Gbest<Gbest_fit
            Gbest     = Pbest(i,:);
            Gbest_fit = test_Gbest;
        end
    end
       %% PSO optimization
%     iter = zeros(1,Maxiter);
%     it=0;
    for generation=1:Maxiter

        %Linearly decreasing inertia weight
        r1 = round(rand(Npar,nbElements),2);
        r2 = round(rand(Npar,nbElements),2);

        Gbest = repmat(Gbest,Npar,1);

        W=(Wmax-((Wmax-Wmin)*generation)/Maxiter);
        W=round(W,2);
        if c1&c2==12
        c1=2.5+(0.5-2.5)*(generation/Maxiter);
        c2=0.5+(2.5-0.5)*(generation/Maxiter);
        end

        % V calculation
        V = W.*V + c1*r1.*(Pbest - X) + c2*r2.*(Gbest - X);
        V = round(V,2);
        %Test if V<Vmax
```

```
        L_V = abs(V) > Vmax;
        V(L_V==1) = sign(V(L_V==1)).*Vmax(L_V==1);

        % Update of the particles' position
        X = X + V;
        X = round(X,2);

        % Test if X is in the problem domain
        L1_X = X > Bound_max;
        X(L1_X == 1) = Bound_max(L1_X == 1);
        L2_X = X < Bound_min;
        X(L2_X == 1) = Bound_min(L2_X == 1);
        X = round(X,2);

  % FITNESS FUNCTION
        for i = 1:Npar
            Fitness(i,1) =
feval('AimFcn_1',X(i,:),nbElements,NbFreq,precision,f_undamage,freqRef);

            % Best position of the particle until the iteration
            if Fitness(i,1) < Fitness_old(i,1)
                Pbest(i,:) = X(i,:);
                Fitness_old(i,1) = Fitness(i,1);
            end

            if Fitness(i,1) == -1
                a = 1;
                break
            end
        end

        %Selection of the best particle of the all the swarm
        [Gbest_new, Pos] = min(Fitness);
        if Gbest_new < Gbest_fit
            Gbest_fit = Gbest_new;
            Gbest = X(Pos,:); % Particle (Pos) and : (every elements of the structure
for this considered particle)
        else
            Gbest = Gbest(1,:);
        end
        %If the solution is found stop iterations
        if a == 1
            a = 0;
            break
        end
    end
% Best solution(POSITION) of each nbPSO is kept in vector population
    population(nbPSO,:)=Gbest(1,:);
    %Fitness value of the best solution of each nbPSO

fitGlobal(nbPSO)=AimFcn_1(population(nbPSO,:),nbElements,NbFreq,precision,f_undamage,fr
eqRef);
end

%% NUMBER OF REAL SOLUTIONS FOUND WRT EACH PSO
nb_sol = 0;
errTol = -0.999; %fitness function S4 is used, if not 1E-4
L_fitgl = fitGlobal <= errTol;

% parfor, then sum all the number of solutions found by each worker (processor)
for i = 1:nbMaxPSO
    if L_fitgl(i) == 1
        comp =abs(population(i,:)-instance); %to have know if we achived a good result
this difference should be <=0.0001 (1E-4)
        if sum(comp <= 1E-2) == nbElements %to count as a solution the Nb_elements
should have difference <=0.0001 (1E-4)
            nb_sol = nb_sol + 1;
        end
    end
end
```

```
b = 1/nb_sol;
Nb_solutions=nb_sol;
```

"restart_PSO.mat":

```
function restart_PSO(config_set,instances,nb,N,k,new_f,I,nber_config)
if ismember(nb,N)==1 %only calculate the PSO of the config if it is needed
    disp(num2str(nb));
    [cost_config(k,nb),Nb_solutions(k,nb)]= PSO(config_set(nb,:),instances(k,:));

parforsave2(fullfile(new_f,sprintf('last_configs_testPSO%d',nb)),cost_config(k,nb),conf
ig_set(nb,:),k,nber_config,nb)
end
```

"parforsave2.mat":

```
function parforsave2(file,cost_config,config_set,k,nber_config,nb)
   save (file,'cost_config','config_set','k','nber_config','nb')
      end
```

"Init_lancement.mat":

```
a = 1;

save('Lancement_num','a');
```

"InitXpar.mat":

```
clear;
close all;
clc;

%% INITIALIZES THE POSITION OF EACH PARTICLES IN THE SWARM %%
Npar = 500;
nbMaxPSO = 1000;
nbElements = 100;

BoundaryMin = 0;
BoundaryMax = 1;
BoundMin = BoundaryMin*ones(Npar,nbElements);
BoundMax = BoundaryMax*ones(Npar,nbElements);

% position = cell(Npar,nbMaxPSO);
position = cell(1,nbMaxPSO);

for s = 1:nbMaxPSO
    X = BoundMin + (BoundaryMax-BoundaryMin)*round(rand(Npar,nbElements),1);
    position{1,s} = X;
end

%% FILE SAVING
% InitXPos = position;

save('InitXPos','position')
```

Anexo C: algortimo DOE en MATLAB

“DOE1.mat”:

```
%% DOE %%
clear all
clc
rng('shuffle')
%% algorithm of exchange %%
a=1;
tic
while a<=200    %200 depart matrices
    %% load of experiments and matrices creation
load('X and config.mat')
load('Nb_init_matrix_depart1') %a=1; save('Nb_init_matrix_depart1','a')
BasePath = pwd;
ninit=36; %%CHANGE NRO % N=38,39,40, ...,55
folder = fullfile(BasePath,sprintf("X_opt_init_%d",ninit));
mkdir (folder)
% chose randomly nmodel experiments of the X matrix

p=36; % p=sum(mi-1)+1=36 ;mi is the numbers of levels for each factor [(10-1)+(10-
1)+(10-1)+(6-1)+(4-1)]+1
nexper=n-ninit;
exper=nexper;
det_miinf_b=0;
while det_miinf_b<=1
    r_xinit = randi([1 19920],1,ninit); %random initial  matrix
    r_xinit = sort(r_xinit);
    r_xexper=1:n;
    r_xexper(r_xinit)=[];

    Xinit=zeros(ninit,p);
    Xexper=zeros(nexper,p);
    i=1;
    for i=1:ninit
        Xinit(i,:)=X(r_xinit(1,i),:);
    end
    j=1;
    for j=1:nexper
        Xexper(j,:)=X(r_xexper(1,j),:);
    end

    %check if det=~0 for the matrices
    [~,det_meinf,inv_meinf]=information_matrix(Xexper);
    [~,det_miinf_b,~]=information_matrix(Xinit);
end

% while the exchange provoke a significative increase in the determinant
% det(X'.X) (difference between det_miinf_b before exchange and det_miinf after
exchange)

z=0;
% val_dx=1;
while z<=1.6e+4
    z=z+1;
        %% keep the config in Xeper with the dx maximun
    [nexper,~]=size(Xexper);
    [dx]=variation_dx(Xexper,inv_meinf);
    dxM(z,1)=max(dx);

    config_dxM=zeros(1,36);
    x=1;
    i=1;
    for i=1:nexper
```

```
            if dx(i)==dxM(z,1)
                maxim(x)=i; %keep all the confg with dx max
                x=x+1;
            end
    end
    [rows,~]=size(maxim);
    i_keep=maxim(randi(rows),:); %chose randomly one config with dx max
    xi_keep=Xexper(i_keep,:);
    Xexper(i_keep,:)=[]; %eliminate xi_keep

        %% Add xi_keep to Xmodel
    Xinit(ninit+1,:)=xi_keep(1,:); %nmodel+1
    [~,det_mi2inf,~]=information_matrix(Xinit); %ninit+1

        %% elimination of one configuration of Xmodel
    k=1;
    Xtest=Xinit;
    for k=1:ninit+1
            Xtest(k,:)=[];
            [~,det_miinf(k),~]=information_matrix(Xtest); %ninit
            Xtest=Xinit;
            dif_det(k)=det_mi2inf-det_miinf(k); %difference between det(X'.X)-
det(X'+1.X+1)
    end
    [~,ind]=min(dif_det);
    Xinit(ind,:)=[]; % suppression x that once eliminated decrease the value of
det(X+1'.X+1) the least
    det_miinf_ae(z)=det_miinf(ind);
    diff(z)=det_miinf_ae(z)-det_miinf_b; %diff between det(x'.x)_initial and
det(x'.x)_after.A.E
%     val_dx=dxM(z);
end
det_ae=det_miinf_ae(z);

f = fullfile(folder,sprintf("Init_%d_%d.mat",ninit,a));
save(f,'Xinit','Xexper','det_miinf_b','det_miinf_ae','diff','det_ae')%
save(f,'Xinit','Xexper','det_miinf_b','det_miinf_ae','diff','D','det_ae')
%
save(sprintf("Init_%d_all.mat",ninit),'Xinit','Xexper','det_miinf_b','det_miinf_ae','di
ff','D','dxM','det_ae')%
a=a+1;
save('Nb_init_matrix_depart1.mat','a')
end
toc

%% best Xinit of the 200 matrices of depart
BasePath=pwd;
folder = fullfile(BasePath,sprintf("X_opt_init_%d",ninit)); %%%%%
for i=1:200
    load(fullfile(folder,sprintf("Init_%d_%d.mat",ninit,i)))
    dett(i,1)=det_ae;
end
[~,index]=max(dett);
load(fullfile(folder,sprintf("Init_%d_%d.mat",ninit,index)))
best_X=Xinit;

folder2 = fullfile(BasePath,'Optimal matricies');
mkdir (folder2)
f2 = fullfile(folder2,sprintf("best_X_%d.mat",ninit));
save(f2,'best_X')
```

"information_matrix.mat":

```
function [matrix_inf,det_infor,inv_matrix_inf]=information_matrix(X)
tX=X.';
matrix_inf=tX*X;
```

```
det_infor=det(matrix_inf);
inv_matrix_inf=inv(matrix_inf);
end
```

"variation_dx.mat":

```
function [dx]=variation_dx(X,inv_matrix_inf)
[n,~]=size(X);

for i=1:n
    f=X(i,:);
    ft=f.';
    dx(i)=f*inv_matrix_inf*ft; %variation
end
end
```

"Y_PSO.mat":

```
clear all
clc
%% reconstruction matrix experience - re-coding
load('best_X_95.mat')
load('X and config.mat')
[nXinit,p]=size(best_X); %%load matrix
[nX,~]=size(X);
E_init=zeros(1,5);
i=1;
for xo=1:nXinit
    for i=1:nX
        l=best_X(xo,:)==X(i,:);
        if sum(l)==p
            ind_X0(xo)=i;
            E_init(xo,:)=config(ind_X0(1,xo),:);%(Nb_par,C1,C2,Nb_iter,Nb_freq)
        end
    end
end
%% Y %%

%'Nb_pso','Nb_elem', 'precision','Vmax'
Nb_pso=10;
Nb_elem=30;
precision=4;
Vmax=1;

[c,~]=size(E_init);
F=zeros(c,4);
F(:,1)=Nb_pso;
F(:,2)=Nb_elem;
F(:,3)=precision;
F(:,4)=Vmax;
config=zeros(c,9);
config(:,1:5)=E_init;
config(:,6:9)=F;

load('Instances_set.mat') %instances used in the F-Race
% r=randi([1:22],1,6);
r=[22    11    18     4    10    21];
Y=zeros(65,6);

for j=1:65
    for i=1:6
    [~,Y(j,i)] = PSO_DOE(config(j,:),instances(r(1,i),:));
    save('Y_PSO_results.mat','config','Y')
```

```
    end
end
```

"PSO_DOE.mat":

```
function [b,Nb_solutions] = PSO(config,instance) % Configuration -
Nb_elem,Nb_pso,Nb_iter,Nb_freq,prec,Nb_par,C1,C2,w,Vmax
rng('shuffle')

a = 0;

load('InitXPos','position')
pos = position;
%Nb_par,C1,C2,Nb_iter,Nb_freq
Configuration = {'Nb_par','c1', 'c2','Nb_iter', 'Nb_freq', 'Nb_pso','Nb_elem',
'precision','Vmax'};
for i = 1:length(config)
    Configuration{2,i} = config(i);
end

%% INITIALIZATION
% VARYING PARAMETERS
nbElements = config(7); % Swarm size / BEAM FEM

Maxiter = config(4); % Number of iterations
Npar = config(1); % Number of particles

NbFreq = config(5); % Number of frequencies identified (5 or 9)

c1 = config(2); % Cognitive Acceleration
c2 = config(3); % Social Acceleration

% FIXED PARAMETERS
precision = config(8); %
nbMaxPSO = config(6); % Number of PSO to execute
Wmax        = 0.9; % Inertial weight Max
Wmin        = 0.4; % Inertia weight Min

%% CASE - REFERENCE DEFECTED AND HEALTHY STRUCTURE
[~,freqRef] = Reference_case_1(nbElements,NbFreq,precision,instance); % freqRef is the
frequency allowing to compare the unknown solution WRT the exact one
instance_undamage=zeros(1,nbElements);
[~,f_undamage] = Reference_case_1(nbElements,NbFreq,precision,instance_undamage);
%f_undamage is the frequency of a undamage structure use to have the fitness

%% PROBLEM DIMENSION: boundary of the space problem
BoundaryMin = 0;
BoundaryMax = 1;
Bound_min = BoundaryMin*ones(Npar,nbElements);
Bound_max = BoundaryMax*ones(Npar,nbElements);

Vmax = config(9);
Vmax = Vmax*ones(Npar,nbElements); % Problem boundary

%% Preallocation des variables globales
population = zeros(nbMaxPSO,nbElements);
fitGlobal = zeros(nbMaxPSO,1);
% itersol       = zeros(nbMaxPSO,1);
% tempsdecalcul = zeros(nbMaxPSO,1);
%%
% matlabpool close;
% c = parcluster; matlabpool(c)

%% PSO LOOP
%parfor
for nbPSO = 1:nbMaxPSO
```

```
%   tic
    %
    X = zeros(Npar,nbElements);
    V = ones(Npar,nbElements);
    %
    %Inialization of V and X
%   X = pos{nbPSO}(1:Npar,1:nbElements);
%   X = pos{1}(1:Npar,1:nbElements);
    X = pos{nbPSO}(1:Npar,1:nbElements);
    V = round(rand(Npar,nbElements),2).*X; %rand(Npar,nbElements).*X; % Velocity (or
X)!!!!!! why $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
    %
    Pbest = zeros(Npar,nbElements);
    Pbest = X; % Personal best

    % Preallocation
    Fitness = zeros(Npar,1);
    Fitness_old = 1E+10*ones(Npar,1);
    %
    Gbest = zeros(1,nbElements);
    Gbest_fit = 1E+12;
    %
    for i=1:Npar
        %Initial Gbest
        test_Gbest =
AimFcn_1(Pbest(i,:),nbElements,NbFreq,precision,f_undamage,freqRef);
        if test_Gbest<Gbest_fit
            Gbest     = Pbest(i,:);
            Gbest_fit = test_Gbest;
        end
    end
       %% PSO optimization
%     iter = zeros(1,Maxiter);
%     it=0;
    for generation=1:Maxiter
        %Linearly decreasing inertia weight
        r1 = round(rand(Npar,nbElements),2);
        r2 = round(rand(Npar,nbElements),2);

        Gbest = repmat(Gbest,Npar,1);

        W=(Wmax-((Wmax-Wmin)*generation)/Maxiter);
        W=round(W,2);
%         c1=2.5+(0.5-2.5)*(generation/Maxiter);
%         c2=0.5+(2.5-0.5)*(generation/Maxiter);
%         %

    % V calculation

                V = W.*V + c1*r1.*(Pbest - X) + c2*r2.*(Gbest - X);
                V = round(V,2);
                %Test if V<Vmax
                L_V = abs(V) > Vmax;
                V(L_V==1) = sign(V(L_V==1)).*Vmax(L_V==1);

                % Update of the particles' position
                X = X + V;
                X = round(X,2);

                 % Test if X is in the problem domain
        L1_X = X > Bound_max;
        X(L1_X == 1) = Bound_max(L1_X == 1);
%         X(L1_X == 1) = roundn(rand(),-2);
        L2_X = X < Bound_min;
        X(L2_X == 1) = Bound_min(L2_X == 1);
%         X(L2_X == 1) = roundn(rand(),-2);
        X = round(X,2);
%

           % FITNESS FUNCTION
```

```
        for i = 1:Npar
%             disp(['PSO n°',num2str(nbPSO),' itération n°',num2str(generation),'
particule n°',num2str(i)]);
            Fitness(i,1) =
feval('AimFcn_1',X(i,:),nbElements,NbFreq,precision,f_undamage,freqRef);

%             Convergence(i,generation) = Fitness(i,1);

            % Best position of the particle until the iteration
            if Fitness(i,1) < Fitness_old(i,1)
                Pbest(i,:) = X(i,:);
                Fitness_old(i,1) = Fitness(i,1);
            end

            if Fitness(i,1) == -1
                a = 1;
                break
            end
        end

        %Selection of the best particle of the all the swarm
        [Gbest_new, Pos] = min(Fitness);
        if Gbest_new < Gbest_fit
            Gbest_fit = Gbest_new;
            Gbest = X(Pos,:); % Particle (Pos) and : (every elements of the structure
for this considered particle)
        else
            Gbest = Gbest(1,:);
        end
        %If the solution is found stop iterations
        if a == 1
            a = 0;
            break
        end
    end
% Best solution(POSITION) of each nbPSO is kept in vector population
    population(nbPSO,:)=Gbest(1,:);
    %Fitness value of the best solution of each nbPSO

fitGlobal(nbPSO)=AimFcn_1(population(nbPSO,:),nbElements,NbFreq,precision,f_undamage,fr
eqRef);
end

%% NUMBER OF REAL SOLUTIONS FOUND WRT EACH PSO
nb_sol = 0;
errTol = -0.999; %fitness function S4 is used, if not 1E-4
L_fitgl = fitGlobal <= errTol;

% parfor, then sum all the number of solutions found by each worker (processor)
for i = 1:nbMaxPSO
    if L_fitgl(i) == 1
        comp =abs(population(i,:)-instance); %to have know if we achived a good result
this difference should be <=0.0001 (1E-4)
        if sum(comp <= 1E-2) == nbElements %to count as a solution the Nb_elements
should have difference <=0.0001 (1E-4)
            nb_sol = nb_sol + 1;
        end
    end
end
b = 1/nb_sol;
Nb_solutions=nb_sol;
```

“Vector_A.mat”:

```
%% vector A
```

```
[matrix_inf,det_infor,inv_matrix_inf]=information_matrix(best_X); %load best_X_100
Y=[0.5
4.166666667
1.166666667
0
0
0
0
3.5
4
6.333333333
5
5.333333333
0
0
0
0
5
0
0
17.66666667
11.66666667
9.833333333
50.5
30.5
0
11.16666667
19.66666667
26.83333333
0
0
0
0
0
0
0.166666667
0.666666667
2.833333333
3.5
1.833333333
2.333333333
5.5
1.666666667
1
3.666666667
24.33333333
19.33333333
0
21.5
0
3
29.16666667
0
0
3.333333333
2.166666667
4.166666667
8.333333333
3.333333333
0
0
7.666666667
16.5
0
8.833333333
41
7.333333333
22
13.16666667
0
```

```
4.666666667
0
6.833333333
21.83333333
6.666666667
4.5
0
27.83333333
11.83333333
22.83333333
0.666666667
12.16666667
4.5
1.833333333
11.33333333
0
0
0.333333333
9.166666667
9.166666667
0
4.833333333
9.166666667
4.5
17
18.16666667
2
0
6.333333333
0
19.5
]; %put the values from the excel file - Y is the avarage solutions of the 6 instances
that are tested.
a=(inv_matrix_inf*best_X.')*Y;
```

"Graphic.mat":

```
[~,n]=size(diff);
diff1=diff;
dxM1=dxM;

x=1:n;

subplot(2,1,1); plot(x,dxM1,'g',x,dxM2,'b',x,dxM3,'m',x,dxM4,'r')
xlabel('nb of exchange - X75')
ylabel('how variance of Xexper change')
subplot(2,1,2);plot(x,diff1,'g',x,diff2,'b',x,diff3,'m',x,diff4,'r')
xlabel('nb of exchange - X75')
ylabel('difference - det(X_t.X) before and after exchange')
grid on
```