Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Laboratorio de Biomecánica

Reporte de Practica 1:

DESCRIPCIÓN Y USO DEL CÓDIGO DE OPTIMIZACIÓN TOPOLÓGICA DE 99 LÍNEAS EN MATLAB.

Docente: Dra. Yadira Moren Vera

Día clase: martes

Hora: N3

Alumno	Matricula	Carrera
Sergio Jared Moreno Rodríguez	1880525	IMTC
Melanie Sofia Sánchez Barbosa	1902265	IMTC
Irving Raúl Garza Escobar	1910127	IMTC
Alejandro Gaytan Ayala	1991902	IMTC
Hernan Abif Castillo Mota	1992201	IMTC
Luis Gerardo Castillo Troncoso	1992317	IMTC

Número de equipo: 4

Brigada 202

Fecha: martes 6 de septiembre de 2022

Semestre agosto-diciembre 2022

Objetivo

El Estudiante conocerá cada una de las secciones que mejoran el código de optimización topológica, como se debe de crear el archivo (-m) en MATLAB y como se ejecuta el análisis.

1.- Introducción

En esta actividad de la información obtenida de nuestra propuesta, se conoce qué son los métodos de análisis de elemento finito, las ecuaciones que son utilizadas, para que nos pueden servir ya que este método es uno de los más utilizados en la física y la ingeniería para la aproximación de soluciones de ecuaciones diferenciales entre más fina sea una malla más exacta será nuestro resultado.

Aunque puede usarse en cálculos a mano, es cuando se utilizan computadoras cuando permite la resolución de problemas sobre geometrías complicadas, de esta manera se trabaja con el código establecido en el programa MATLAB con las fórmulas que son empleadas en un código para hacernos la soluciones más acertadas y rápidas y de esta manera tener una noción de estos tipos de análisis de elementos finitos con programas que son los más utilizados en una ingeniería.

2.- Desarrollo

Esta parte comprende lo relacionado la propuesta de análisis de manera educativa que se llegó, asimismo se establece la debida investigación sobre el estado de la programación y sobre el análisis de elementos finitos en la actualidad, para dar con el procedimiento y el desarrollo que son usados.

Propuesta de análisis

La propuesta a la que se llega es la del análisis de una viga en el cual se llama “A 99 line topology optimization code written in Matlab” de O. Sigmund, cuyo propósito es mostrar la optimización en el campo de la topología.

1) **Nombre y definición de la programación y forma de la pieza**

Programación

A 99 line topology optimization code written in Matlab es una programación la cual está dividida en un programa principal. en el cual 12 lineas estan de criterio de optimización que se basa, 16 líneas son para filtración de la textura o malla original de la pieza a optimizar, 35 líneas son para la el algoritmo de análisis de elemento finito, y por ultimo 49 líneas son destinadas a la solución de este problema de optimización.

Método de elementos finitos

El método de elementos finitos es una técnica numérica para resolver problemas que se pueden describir por ecuaciones diferenciales parciales o que pueden ser formulados por medio de una minimización de un funcional (cálculo variacional). La mayoría de los métodos de optimización emplean este método para el análisis topológico.

La premisa básica es que una región de solución que se describe con funciones continuas puede ser modelada reemplazándola con un arreglo de elementos discretos, pudiendo incluso variar las condiciones de los elementos individualmente o en grupos de acuerdo con las ecuaciones constitutivas que se empleen en el problema.

Optimización topológica

El método de optimización topológica ha ganado una gran popularidad en la academia y la industria, y actualmente se aplica al diseño de estructuras automotrices y de aeroplanos, así como en materiales, mecanismos y diseño de sistemas microelectromecánicos (MEMS). Es una técnica dentro del campo de análisis mecánico y uno de los objetivos consiste en minimizar el peso, manteniendo la funcionalidad mecánica.

Por lo general en un problema de optimización topológica, el objetivo principal es minimizar la cedencia C(x) teniendo en cuenta la distribución del material, la cual puede definirse como:

$$C(\mathbf{x}) = \boldsymbol{F}^{\boldsymbol{T}}\boldsymbol{u}$$

Esta energía de deformación es el producto de dos vectores y se asemeja al trabajo realizado por el vector de fuerza **F** a lo largo de los desplazamientos calculados **u**. Por lo tanto, la expresión dada es en realidad un potencial de trabajo similar a las formulaciones comunes para el equilibrio de la energía potencial en un sistema. De este modo el vector de fuerza **F** es igual al desplazamiento multiplicado por la matriz de rigidez estructural **K(x)**, donde K es la rigidez global y está en función de las variables de diseño:

$$\mathbf{K}(\mathbf{x})\,\mathbf{u} = \mathbf{F}$$

La transformación de la función objetivo puede ser escrita como en la siguiente ecuación. La energía de deformación aquí es una combinación lineal de

$$\min: \mathrm{C}(\mathrm{x}) = \sum_{\mathrm{e}=1}^{\mathrm{N}} \mathbf{u}_{\mathrm{e}}^{\mathrm{T}}\, \mathrm{K}_{\mathrm{e}}\,(\mathrm{x}_{\mathrm{e}})\mathbf{u}_{\mathrm{e}}$$

las energías de deformación de cada elemento formulado en el modelo de elementos finitos.

Puesto que es un valor normalizado, la variable de diseño sólo puede oscilar entre los valores 0 (sin efecto) y 1 (sólido). Para la prevención de posibles singularidades en las matrices del sistema, las densidades no están restringidas a cero, pero si por un límite inferior como se muestra en la siguiente ecuación:

$$0 < x_{min} \leq x \leq 1$$

Por último, debido a que en la optimización topológica existe una redistribución de material se debe tomar en cuenta la razón entre el volumen deseado **V(x)** y el volumen original **Vo**:

$$\frac{V(x)}{V_o} = f$$

Forma de la pieza

La forma de la pieza está dada por un acercamiento en el cual su desventaja está dada por la determinación óptima de su microestructura, resultando que es una estructura que no presenta una definida escala de longitud, sin embargo, este acercamiento es sigue siendo importante en el sentido que brinda información teórica del desempeño de las estructuras.

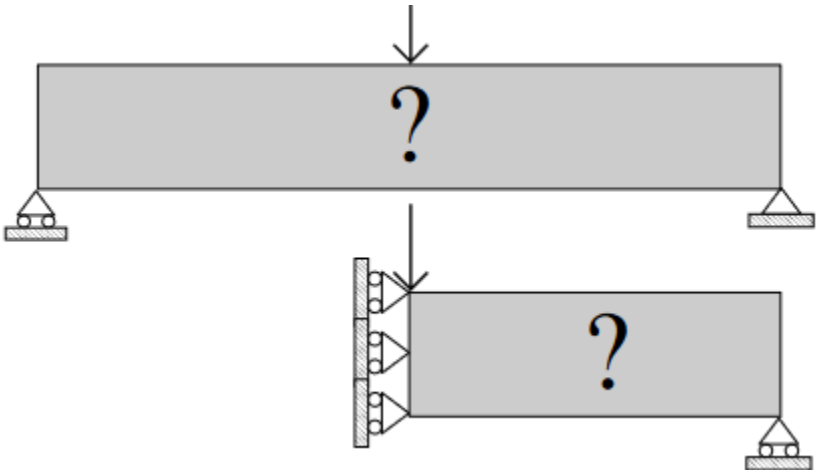

Figura 1. Forma de la pieza, Viga completa (Arriba), Viga seccionada equivalente (Abajo.)

El material de la pieza es asumido como una constante, en el cual cada elemento está diseñado en un dominio relativo en cuando se trata de densidades y propiedades del material, siendo en un principio sólido.

2) Estado del arte

Estado del arte de la programación

En el ámbito del desarrollo de software, el estado del arte (state of the art) típicamente ha estado vinculado con la evolución de los lenguajes de computación. Uno de los primeros grandes saltos en esa dirección fue el desarrollo del primer lenguaje de alto nivel, FORTRAN. En sus inicios, el desarrollo de software en ese lenguaje era considerado “programación automática” porque requería menos conocimientos técnicos que los lenguajes ensambladores. Los detractores (gurús que programaban en lenguaje ensamblador) solían justificar su rechazo mostrando que los programas en FORTRAN eran significativamente más ineficientes comparados con los que ellos escribían. Hoy no solo tenemos lenguajes de programación de alto nivel, de “cuarta” y de “quinta generación”, funcionales u orientados a objetos; también lenguajes de documentación (Latex, HTML, etc.), de especificación (bison, flex, etc.), y otros. Sin embargo, estos lenguajes hoy representan más bien lo que suele llamarse el “state of practice”: lo que incorporan las herramientas comerciales y que utiliza la industria.

Una de las vertientes del “state of the art” tiene que ver con lo que denominamos “métodos formales” (MF).

En términos generales, los MF son técnicas mediante las cuales:

1. Se escribe la especificación del sistema a desarrollar utilizando un lenguaje formal L1 (usualmente del paradigma declarativo), que luego es verificada con intervención humana y procesada mediante un compilador C1 para así

generar el código en otro lenguaje formal L2 que representa un diseño de alto nivel.
2. Este diseño escrito en el lenguaje L2 a su vez es verificado con intervención humana y procesado mediante un compilador C2 que genera el código en otro lenguaje formal L3 (usualmente del paradigma imperativo).
3. Este código en el lenguaje L3 se procesa mediante un compilador C3 para obtener finalmente el sistema ejecutable.

Las verificaciones mencionadas suelen ser demostraciones matemáticas de que el código actual Cn tiene ciertas propiedades.

En realidad, en el desarrollo de software siempre utilizamos un tipo de MF, pues finalmente se escribe el sistema en algún lenguaje Lx, que es procesado por un compilador Cx, el cual usualmente genera código ejecutable. Sin embargo, esta transformación suele realizarse solamente para la fase final de las arriba descritas (la generación del sistema ejecutable a partir del programa escrito en un lenguaje de programación, usualmente imperativo), no para todo el ciclo de desarrollo (requerimientos-diseño-programación). Los MF abordan el ciclo completo. Algunos de los métodos formales más conocidos son el “Método B” y la “Notación Z”.

Ejemplo de un método formal

Un compilador contiene 4 grandes bloques:

1. El analizador léxico, que se encarga de concatenar caracteres significativos para construir “palabras”.
2. El analizador sintáctico, que verifica que esas “palabras” formen “oraciones” con un orden que es permitido.
3. El procesador semántico, que checa que esas “oraciones” tengan significado.
4. El generador de código, que finalmente transforma el programa escrito en el lenguaje fuente en otro documento escrito en el lenguaje destino.

Probablemente también recuerden que el analizador sintáctico o “parser” suele definirse mediante una gramática, usualmente utilizando alguna nomenclatura especializada como los grafos de sintaxis (GS). Hay un tipo de gramáticas (las descendentes predictivas) para las cuales es relativamente fácil la escritura del parser que las procesen, pero tienen ciertas restricciones; en particular:

- la definición no puede tener recursión izquierda (el procesamiento se ciclaría indefinidamente),
- en una alteración de grafos, no puede ocurrir que un par de ellos comience con la misma cadena (siempre tomaría la primera alternativa).

Las reglas para construir un parser descendente predictivo son relativamente sencillas y pueden ser automatizadas para generarlos en

pseudocódigo; por razones de espacio, nos ahorraremos su definición aquí. Con esa automatización tendríamos un pequeño MF: 1. Escribiríamos la especificación del parser a desarrollar utilizando el lenguaje formal de los grafos de sintaxis, que sería verificada por un compilador de GS para detectar, entre otras cosas, recursión izquierda y/o alteraciones de GS no permitidas; en caso de haberlas, la especificación sería corregida con intervención humana. Una vez corregida y verificada, el compilador de GS podría generar el "diseño" del parser en pseudocódigo, como lenguaje intermedio. 2. Este "diseño" a su vez sería procesado mediante un compilador de pseudocódigo que generaría el programa P en algún lenguaje de alto nivel, a elección del usuario (v.gr. Java, LISP). 3. El programa P sería procesado por el compilador correspondiente y generaría el código ejecutable.

Utilizando el MF para desarrollar ese componente de un compilador, nuestra actividad se centra más en el diseño de una gramática que tenga características deseables, que en la escritura de un programa que la procese. Para lograrlo fue necesario conocer el lenguaje de especificación (los GS), y las características que deben tener las gramáticas (v.gr. no recursión izquierda). Ese es el enfoque usual en los MF.

Aplicación de los métodos formales en la industria El uso de los MF en el desarrollo de sistemas críticos no es poco común. En el artículo "Software's Chronic Crisis" pueden encontrarse algunos casos clásicos, entre ellos el uso del método formal "B" para desarrollar un sistema que solucionaba el siguiente problema del metro en París: La cantidad de usuarios de las líneas del metro se había incrementado considerablemente, de manera que era necesario construir más vías para que circularan más trenes, o bien automatizar la sincronización de las llegadas y salidas de las líneas a cada estación, de manera que se redujera el tiempo entre la partida de un convoy y la llegada de otro. Esta automatización es un buen ejemplo de un sistema crítico: si el software fallará al sincronizar llegadas y salidas, podría ocurrir que sobre la misma vía arribará un convoy en una dirección al tiempo que llegara otro en la dirección opuesta, ocasionando una colisión en la que podría morir mucha gente. Obviamente, ante esa posibilidad no es aceptable un sistema con digamos 99.99999% de confiabilidad (que puede fallar una vez en diez millones).

En lugar de ello se utilizó el MF B para "desarrollar software libre de defectos por construcción": se escribió una especificación formal que fue verificada matemáticamente, a partir de la cual se generó un diseño que fue igualmente verificado, para de ahí generar el código fuente del que se obtuvo luego el sistema ejecutable.

Los métodos formales y la prueba de software Este paradigma cambia tanto la actividad de la programación de sistemas, como la de la prueba de los mismos: el desarrollo tiene que ver menos con escribir código e incluye más tareas de verificación (detección de errores "de alto nivel"), que actualmente se incluyen en las actividades de pruebas.

En ese sentido, ambas actividades parecieran confluir. Sin embargo, aún en ese escenario, será necesario probar para revisar por un lado si el sistema desarrollado cumple con los requerimientos del usuario (aspecto que no puede validar el método formal), y por otro si no se cometieron errores durante las verificaciones. Además, hoy en día tenemos tecnologías que interactúan cada vez más con otras, y la integración de sistemas de sistemas es algo que también debe ser probado.

3) Procedimiento de la programación.

El problema de optimización está basado en un acercamiento de la ley de potencia, en donde un objeto es minimizado y está dada por la siguiente función.

$$\begin{aligned} \min_{\mathbf{x}}: \quad & c(\mathbf{x}) = \mathbf{U}^T\mathbf{K}\mathbf{U} = \sum_{e=1}^{N} (x_e)^p \, \mathbf{u}_e^T \, \mathbf{k}_0 \, \mathbf{u}_e \\ \text{subject to}: \quad & \frac{V(\mathbf{x})}{V_0} = f \\ : \quad & \mathbf{K}\mathbf{U} = \mathbf{F} \\ : \quad & \mathbf{0} < \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{1} \end{aligned}$$

Donde la U y la F, son vectores globales de desplazamiento, y K es la matriz de rigidez global, x es el vector de las variables de diseño, Xmin es un vector de densidades relativo, N = (nely, nely) que son el número de elementos del vector horizontal y vertical, p es la penalización de potencia, V es el volumen del material y f es una fracción del volumen.

A continuación, se muestra la implementación de un código compacto de MATLAB para optimización topológica para la minimización de una estructura sometida a cargas estáticas, dicho código fue desarrollado por O. Sigmund, y gran parte del análisis presentado proviene de su investigación.

4) Desarrollo de la programación en sus diferente vistas

Para este código el número total de líneas es 99, el cual se encuentra conformado por las siguientes secciones: creación del programa principal, optimizador basado en el criterio de optimización, filtro de mallado independiente, y código de elemento finito. En seguida se mostrará el código de cada una de las secciones y se dará una descripción de la tarea que desempeñan.

Creación del programa principal (líneas 1-36)

El programa principal empieza con la distribución uniforme de material sobre el dominio determinado. Después de algunas inicializaciones comienza el

lazo con el que se llaman a las subrutinas de elemento finito, la cual regresa un vector U. También se llama una sola vez a la subrutina de la matriz de rigidez. Para posteriormente un nuevo lazo hacer un análisis de sensibilidad y función objetivo sobre cada uno de los elementos. Lo siguiente es un filtro de mallado independiente y aplicar el optimizador en base al criterio de optimización. Los resultados son impresos, junto con una representación gráfica de los resultados. La longitud del lazo principal está determinada por el cambio con respecto a los valores de diseño, si es menor al 1%, éste se detiene.

```
function top(nelx,nely,volfrac,penal,rmin)
% INITIALIZE
x(1:nely,1:nelx) = volfrac;
loop = 0;
change = 1.;
% START ITERATION
while change > 0.01
  loop = loop + 1;
  xold = x;
% FE-ANALYSIS
  [U]=FE(nelx,nely,x,penal);
% OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
  [KE] = lk;
  c = 0.;
  for ely = 1:nely
    for elx = 1:nelx
      n1 = (nely+1)*(elx-1)+ely;
      n2 = (nely+1)* elx   +ely;
      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
      dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue;

    end
  end
% FILTERING OF SENSITIVITIES
  [dc]   = check(nelx,nely,rmin,x,dc);
% DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
  [x]    = OC(nelx,nely,x,volfrac,dc);
% PRINT RESULTS
  change = max(max(abs(x-xold)));
  disp([' It.: ' sprintf('%4i',loop) ' Obj.: ' sprintf('%10.4f',c) ...
       ' Vol.: ' sprintf('%6.3f',sum(sum(x))/(nelx*nely)) ...
        ' ch.: ' sprintf('%6.3f',change )])
% PLOT DENSITIES
  colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
end
```

Optimizador basado en el criterio de optimización (líneas 37-48)

Las variables actualizadas son encontradas a través del optimizador. Sabiendo que el volumen del material es una función monótona decreciente de el multiplicador de Lagrange, el valor del multiplicador de Lagrange que satisface la restricción de volumen puede ser encontrado mediante algoritmos de bi-seccionamiento. El algoritmo de bi-seccionamiento es inicializado proponiendo un mínimo y un máximo, delimitados por el multiplicador de Lagrange. Los intervalos con los que dichos intervalos son delimitados se van cortando por la mitad repetidamente hasta que el valor es menor al criterio de convergencia.

```
%%%%%%%%% OPTIMALITY CRITERIA UPDATE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [xnew]=OC(nelx,nely,x,volfrac,dc)
l1 = 0; l2 = 100000; move = 0.2;
while (l2-l1 > 1e-4)
  lmid = 0.5*(l2+l1);
  xnew = max(0.001,max(x-move,min(1.,min(x+move,x.*sqrt(-dc./lmid)))));
  if sum(sum(xnew)) - volfrac*nelx*nely > 0;
    l1 = lmid;
  else
    l2 = lmid;
  end
end
```

Filtro de mallado independiente (líneas 49-64)

Nótese que no todos los elementos en el dominio del diseño son buscados para encontrar los elementos que se encuentran dentro del radio pero solo aquellos dentro de un cuadrado con longitud de lados dos veces el valor de alrededor del elemento considerado. Seleccionando menor a 1 cuando se manda a llamar la rutina, las sensibilidades filtradas serán igual a las sensibilidades originales, haciendo que el filtro quede inactivo.

```
%%%%%%%%% MESH-INDEPENDENCY FILTER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [dcn]=check(nelx,nely,rmin,x,dc)
dcn=zeros(nely,nelx);
for i = 1:nelx
  for j = 1:nely
    sum=0.0;
    for k = max(i-floor(rmin),1):min(i+floor(rmin),nelx)
      for l = max(j-floor(rmin),1):min(j+floor(rmin),nely)
        fac = rmin-sqrt((i-k)^2+(j-l)^2);
        sum = sum+max(0,fac);
        dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)*dc(l,k);
      end
    end
    dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
  end
end
```

Código de elemento finito (líneas 65-99)

La matriz global de rigidez está formada por un lazo efectuado sobre todos los elementos. Las variables y denotan el número de los nodos superiores izquierdo y derecho y son usados para insertar el elemento de la matriz de rigidez en las posiciones correctas de la matriz global. Además, cada nodo tiene dos grados de libertad (horizontal y vertical), por lo que el comando indica que se está aplicando una unidad de fuerza vertical en la esquina superior izquierda. Los apoyos son implementados mediante la eliminación de los grados de libertad de las ecuaciones lineales. Los elementos de la matriz de rigidez son calculados en las últimas líneas, la matriz de 88 fue determinada analíticamente usando manipulación simbólica.

```
%%%%%%%%%% FE-ANALYSIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [U]=FE(nelx,nely,x,penal)
[KE] = lk;
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1));
F = sparse(2*(nely+1)*(nelx+1),1); U = zeros(2*(nely+1)*(nelx+1),1);
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx   +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end
% DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
F(2,1) = -1;
fixeddofs   = union([1:2:2*(nely+1)],[2*(nelx+1)*(nely+1)]);
alldofs     = [1:2*(nely+1)*(nelx+1)];
freedofs    = setdiff(alldofs,fixeddofs);
% SOLVING
U(freedofs,:) = K(freedofs,freedofs) \ F(freedofs,:);
U(fixeddofs,:)= 0;


%%%%%%%%%% ELEMENT STIFFNESS MATRIX %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [KE]=lk
E = 1.;
nu = 0.3;
k=[ 1/2-nu/6   1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
   -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
KE = E/(1-nu^2)*[ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
                  k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
                  k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
                  k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
                  k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
                  k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
                  k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
                  k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
```

En la siguiente imagen se muestran los resultados gráficos de la implementación del código con los parámetros iniciales de “top (60,20,0.5,3.0,1.5)”

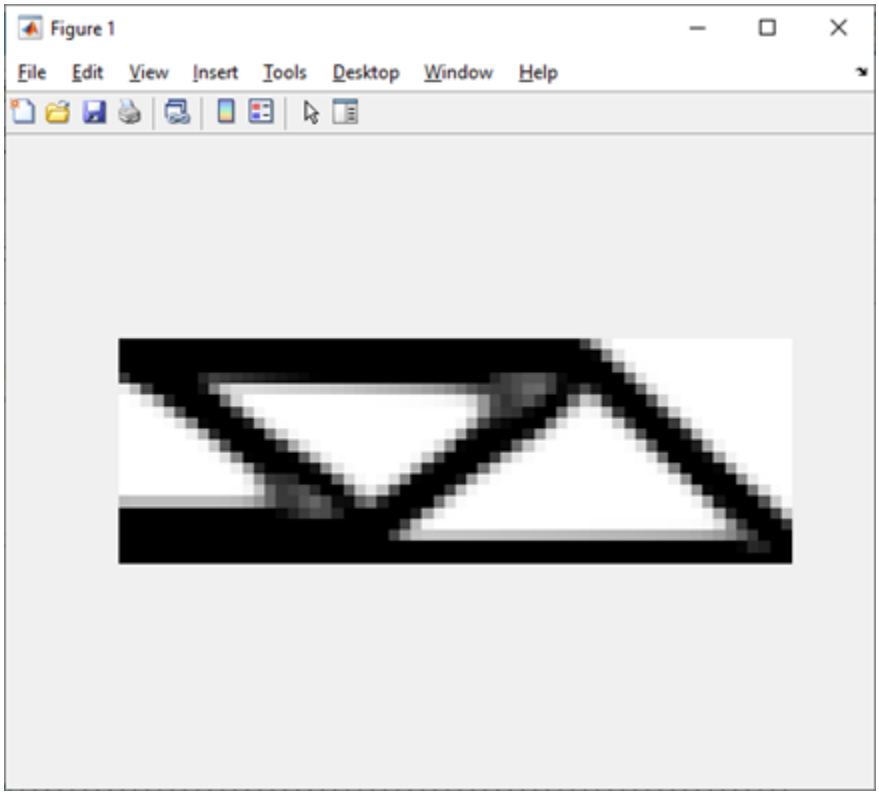

Figura 2. Viga seccionada equivalente optimizada.

3.- Conclusiones de cada autor

Sergio Jared Moreno Rodríguez – 1880525

En esta primera actividad iniciamos con el análisis de la optimización topológica la cual es una técnica en el análisis estructural para que sean más ligeras , consiste en modificar la geometría del dominio preservando su topología, es decir sin crear huecos o cavidades en su interior.

También vimos las ecuaciones de análisis de elemento finito las cuales nos sirven en este caso para tener un análisis más preciso una malla más fina y más exacta.

Para aterrizar este análisis se trabajó mediante un código en MATLAB.

Melanie Sofia Sánchez Barbosa – 1902265

En esta práctica pudimos aprender acerca de qué son y la utilidad de los algoritmos de optimización topológica, así como también se vio un ejemplo de esto mediante un código de Matlab de tan solo 99 líneas, el cual se encargaba de optimizar una estructura para ciertas cargas definidas. La optimización topológica es de gran importancia en el área de la biomecánica, por ejemplo en el desarrollo de prótesis, generalmente se busca que éstas sean muy resistentes, pero sin pesar demasiado, ya que puede complicarle al usuario el uso de dicha prótesis, este es un claro ejemplo en donde la optimización topológica tiene un gran campo pues se pueden analizar las estructuras a emplear en las prótesis para que éstas sean lo más ligeras posibles pero sin sacrificar su resistencia.

Irving Raúl Garza Escobar – 1910127

Se dio un acercamiento de como de un algoritmo matemático que conforma un método de análisis de elemento finito, en el cual se conoce como parte, en el cual es algo que sin un programa sería casi imposible o muy tardado de realizar, en el cual se da a conocer también qué parámetros toma para poderse realizar este tipo de optimización, en el cual, se toma este análisis se toma de una viga que está soportada y con la de una carga hacia un dirección estática, toma los diferentes valores como, la flexión que se le genera a si como puntos críticos o el esfuerzo de Von misses, para de esta manera cambiar su estructura de acuerdo a los valores de vectores que se le dio para la optimización en este caso, resultando en una armadura de la optimización que resultó de la viga.

En el cual es interesante de observar como un modelo de optimización se pone a prueba en un programa que no sabía que se podía hacer estos tipos de análisis.

Alejandro Gaytán Ayala – 1991902

Mediante la primera práctica me di cuenta de la gran importancia que tiene el método de elemento finito como técnica numérica con el fin de poder resolver problemas de optimización (empleando el análisis topológico).

Asimismo para el método de optimización, me sorprendió que este se aplique al diseño de estructuras automotrices y de aeroplanos en la industria(así como en materiales, mecanismos y diseño de sistemas electromecánicos)

Hernán Abif Castillo Mota – 1992201

Esta práctica de laboratorio tiene la función de presentar una implementación muy simple de un algoritmo de optimización topológica, en este caso el código se implementa utilizando solo 99 líneas de entrada de Matlab.

El método de elementos finitos está pensado para ser usado en computadoras y permite resolver ecuaciones diferenciales asociadas a un problema físico sobre geometrías complicadas, en este caso se implementó para una optimización topológica, para así aligerar la estructura propuesta y manteniendo las funcionalidades mecánicas.

Por último, como podemos observar todo se realizó en Matlab, por lo que es importante darse cuenta de la importancia y utilidad de distintos software, ya que nos facilitan mucho el trabajo y permiten realizar cálculos más precisos.

Luis Gerardo Castillo Troncoso – 1992317

Esta práctica me hizo investigar más acerca del método de los elementos finitos aplicado en un programa de computadora la idea es resolver un problema sobre una geometría a través de ecuaciones diferenciales, esto da como resultado poder hacer un análisis topológico, se realizó en matlab por el cual se estuvo programando en 4 partes importantes: lla creación del programa principal, optimizador basado en el criterio de optimización, filtro de mallado independiente, y código de elemento finito, este al final se crea el resultado esperado de manera gráfica una viga seccionada equivalente optimizada.

4.- Bibliografías

W. Gibbs; “Software’s Chronic Crisis”, Scientific American, September 1994. http://bit.ly/sg30r1

Ramirez F, García H, López F & De la garza F. Optimización estructural del codo de una prensa mecánica mediante análisis topológico. Universidad Autónoma de Nuevo León. Facultad de Ingeniería Mecánica y Eléctrica. (pp 3-5).

Sigmund, O. (2001). A 99 line topology optimization code written in Matlab. *Struct Multidisc Optim, 21*, 120-127. Obtenido de https://mecheng.iisc.ac.in/suresh/me256/99linecodePaper.pdf