BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE INGENIERÍA

ALGORITMO GENÉTICO PARA LA OPTIMIZACIÓN DE LAS DIMENSIONES EN COLUMNAS

T E S I S

QUE PARA OBTENER EL GRADO DE:

LICENCIATURA EN INGENIERÍA CIVIL

P R E S E N T A:

ALEXIS ERASTO MONTES DE OCA HERNÁNDEZ

DIRECTORA DE TESIS

MTRA. ANA ELENA POSADA SÁNCHEZ

CODIRECTOR DE TESIS

MTRO. EDGAR IRAM VILLAGRÁN ARROYO

Puebla, Pue. Agosto de 2024

AGRADECIMIENTOS

Quiero agradecer a todas esas personas que han sumado en mi vida, tanto como mis amigos, compañeros, maestros y cada una de las personas de las que he aprendido alguna cosa durante mi vida.

Pero en especial quiero agradecer a los mi pilares de mi vida que han hecho lo que soy hoy por hoy, que son mis padres, que sin su apoyo no hubiera podido llegar a estudiar una licenciatura, que sin sus consejos no sería una persona recta y de buenos valores, que sin sus correcciones no podría enmendar mis errores y que sin su cariño no me sentiría feliz y pleno, mi más sincera gratitud por su aliento y su comprensión durante esta investigación; quiero agradecer a mi hermana que siempre ha estado para mí y en la que tengo un amiga en la quien confiar, gracias por sus consejos y por sus regaños. Y quiero agradecer a esa persona especial en mi corazón que siempre ha estado conmigo en las buenas y malas, la que me ha acompañado a lo largo de la universidad y me dio ánimos cuando más lo necesitaba, la que me apoyó cuando sentía que no podía más en la universidad, esa que en días grises hacía que no me sintiera triste, al contrario, me hacía olvidar de mis penas. Gracias por su amor incondicional.

Quiero expresar mi gratitud a la Facultad de Ingeniería de la Benemérita Universidad Autónoma de Puebla por proporcionar los recursos y el ambiente propicio para llevar a cabo esta investigación.

Por último, quiero expresar mi profunda gratitud a mi asesor de tesis la maestra Ana Elena Posada Sánchez y a mi coasesor de tesis el maestro Edgar Iram Villagrán Arroyo por brindarme su orientación experta, su paciencia, su tiempo y su apoyo inquebrantable a lo largo de este viaje académico, gracias por estar semana a semana pendientes de esta investigación, gracias por atender mis dudas siempre que las requería.

Finalmente quiero dedicar este trabajo a mi familia y mis seres queridos, cuya inspiración y ejemplo me han impulsado a alcanzar mis metas académicas y profesionales.

Contenido

INTRODUCCIÓN

En este trabajo de tesis se muestra a los algoritmos genéticos como una herramienta útil en la búsqueda de soluciones y alternativas para obtener las dimensiones de columnas en marcos, tales que cumplan con los desplazamientos laterales por sismo, en el área de Ingeniería Civil. Algunos de los casos mostrados son ejemplos bibliográficos de otros países e incluso se presenta un caso que se realizó en nuestro país para fortalecer el estudio sustentado con esta investigación y darle una mayor credibilidad.

Nuestra hipótesis nos dice que los algoritmos genéticos pueden ser empleados para le optimización de las dimensiones de las columnas, tal que satisfagan los desplazamientos laterales por sismo estipulados en las normas de mayor uso en México.

El objetivo del presente trabajo es desarrollar un algoritmo para obtener las dimensiones óptimas de las columnas para marcos con ciertas características en cuanto a materiales, dimensiones, restricciones, etc. No es una tarea fácil, sobre todo si se intenta por métodos de prueba y error, es por eso que en esta investigación la optimización máxima se logra gracias a un lenguaje de programación (Python) que utiliza algoritmos genéticos y evalúa su diseño con base en las Normas Técnicas Complementarias para Diseño por Sismo (Puebla, 2017) para así, mejorar el rendimiento en tiempo en el diseño de marcos.

En el capítulo uno se describe cómo definimos los coeficientes y valores de algunas variables, dado que también incluimos datos de las normas manejadas en el algoritmo y se incluye, de igual forma, algunas fórmulas de las que utilizamos, así como también se presentan las descripciones de los tipos de terreno.

Para entender mejor qué son los algoritmos genéticos y cómo funcionan, el segundo capítulo de esta investigación explica de manera general en qué consisten los algoritmos genéticos y cómo su funcionamiento se asemeja a la naturaleza, de ahí el nombre de esta herramienta de búsqueda; así como sus características y sus operadores, en los cuales el algoritmo genético se basa para desarrollar su proceso de

búsqueda de soluciones, dando un excelente rendimiento para su aplicación. Se mencionan y explican la aplicación de algoritmos genéticos que utilizamos en nuestra investigación.

En el capítulo tres se describe todo el proceso que se llevó a cabo para la realización del algoritmo genético, partiendo desde las fórmulas que se emplearon hasta los códigos utilizados para el funcionamiento del algoritmo, así como la optimización de éste.

En el capítulo cuatro se define el problema, como también se menciona los operadores y parte de las funciones empleadas en el algoritmo para su funcionalidad. De igual forma, se menciona a detalle el manejo de las restricciones.

El capítulo cinco contiene la definición geométrica de los tres marcos usados en esta investigación, incluyendo sus dimensiones, es decir, alturas y bases, y su número de niveles, así como su número de crujías.

En el capítulo seis se menciona cómo se aplicó el algoritmo y lo que se buscaba con respecto a su función, mientras que en el capítulo siete se analizan los resultados (fuerzas, distorsiones máximas, desplazamientos máximos, etc.) del algoritmo aplicado a los marcos ya optimizados.

En el capítulo ocho y último de la investigación se muestra una comparación de los resultados obtenidos con el algoritmo genético, con los marcos calculados con un programa para diseño estructural y así corroborar que se está realizando un buen cálculo.

Con la finalidad de comprobar la hipótesis planteada, en las conclusiones se destaca que, con base en el algoritmo ejecutado, se pudieron obtener resultados positivos acerca de la optimización de columnas, así como también se presentan los datos que se obtuvieron como resultado de la presente investigación.

CAPÍTULO 1: DISPOSICIONES REGLAMENTARIAS DE LOS DESPLAZAMIENTOS LATERALES DEBIDOS A SISMO

La disposición reglamentaria de los desplazamientos laterales debidos a sismos varía según el país y sus códigos de construcción. Por lo que se tomaron en cuenta las del Código Reglamentario del Municipio de Puebla (Versión: Periódico Oficial del día 10 de abril de 2017). Estos códigos establecen las normas y requisitos para el diseño sísmico de estructuras que se tomaron en cuenta para esta investigación. Algunos puntos generales que suelen incluirse son los siguientes:

1.1. Normatividad

La normatividad en ingeniería civil se refiere al conjunto de reglas, estándares y regulaciones que rigen el diseño, construcción, mantenimiento y uso de infraestructuras civiles, como edificios, puentes, carreteras, presas y sistemas de agua y alcantarillado. Estas normas son esenciales para garantizar la seguridad, la funcionalidad y la durabilidad de las estructuras y para proteger el bienestar público. Aquí hay algunos aspectos clave relacionados con la normatividad en ingeniería civil:

1.1.1. Códigos de construcción

Los códigos de construcción en Puebla, México, se basan generalmente en las normativas nacionales y en las regulaciones específicas establecidas por las autoridades locales. Algunos de los documentos y códigos de referencia que pueden ser aplicables en Puebla son el Código Reglamentario Municipal, Reglamento de Construcción, Reglamento de Protección Civil, etc.

1.1.2. Normas técnicas

Las normas técnicas detallan especificaciones técnicas para materiales, métodos de construcción y prácticas de diseño. Por ejemplo, en el campo de la ingeniería civil, se utilizan normas como Normas Técnicas Complementarias del Reglamento de Construcción de la Ciudad de México para el Diseño y Construcción de las Estructuras de Concreto (NTCS, 2017), que para el estado de Puebla son empleadas de igual manera.

1.2. Zonificación Sísmica

Los códigos establecen zonas sísmicas que indican la probabilidad de actividad sísmica en diferentes regiones. En la ciudad de Puebla existe el Mapa de Zonificación que se presenta en la *Figura 1.2-1*

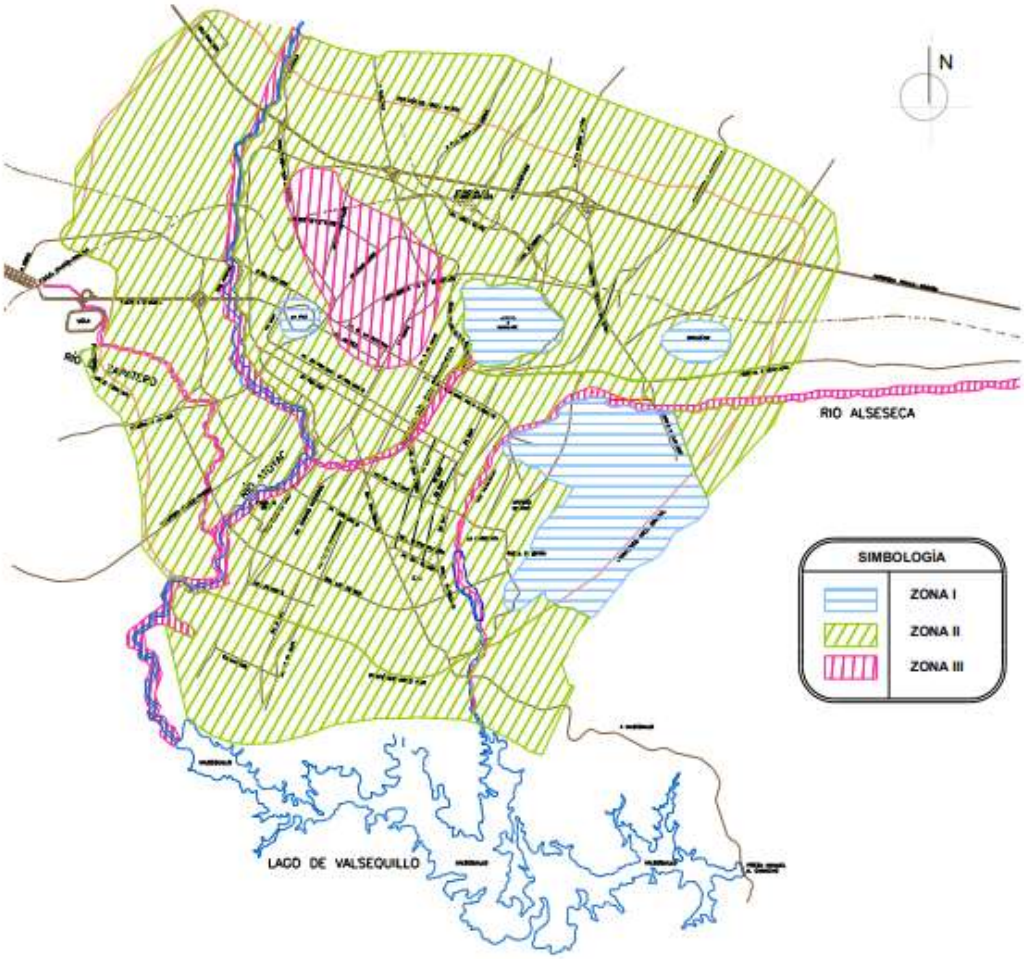


Figura 1.2-1: Mapa de zonificación sísmica de la ciudad de Puebla (Puebla, 2017)

1.3. Clasificación de las construcciones

Atendiendo a la seguridad estructural aconsejable para la estructura, las construcciones se clasifican según su destino como se indica a continuación. (NTCS, 2017)

1.3.1. Grupo A

Se requiere una estructura altamente segura. Edificios cuyo fallo estructural provocaría una gran pérdida de vidas o daños económicos o culturales especialmente graves, o que supongan un peligro importante debido a la presencia de sustancias tóxicas o inflamables, así como los edificios que deban funcionar después de un terremoto. Este puede ser el caso de puentes importantes, sistemas de suministro de agua potable, subestaciones eléctricas, centrales telefónicas, estaciones de bomberos, archivos y registros públicos, monumentos, museos, hospitales, escuelas, estadios, templos, centros de transporte, salas de espectáculos y hoteles con zonas de conferencias, lugares que albergan gran número de personas, gasolineras, almacenes de materiales inflamables o tóxicos y almacenamiento de equipos especialmente caros. El grupo A incluye muros de contención, almacenes ordinarios y muros con una altura superior a 2,5 m.

1.3.2. Grupo B

Se requiere una estructura de nivel de seguridad medio. No pertenecen los edificios cuyo fallo estructural causaría daños moderados o pondría en peligro a otros edificios de este grupo o del grupo A, tales como naves industriales, edificios comerciales, edificios públicos destinados a vivienda u oficinas, salas de espectáculos, hoteles, almacenes y edificios urbanos o industriales.

1.4. Clasificación de los tipos de terrenos

Hay tres tipos de terrenos en el código reglamentario (NTCS Puebla 2017) de los cuales todos tienen sus respectivos valores de a_o, c, T_a, T_b y r que se muestran en la *Tabla 1*.

Los valores aquí especificados para las ordenadas espectrales de diseño son aplicables a construcciones del grupo B.

Tabla 1 Parámetros de los espectros de diseño para estructuras del grupo B

Parámetros de los espectros de diseño para estructuras del grupo B					
Tipo de terreno	a_0	**c**	**Ta (s)**	**Tb (s)**	**r**
I	0.05	0.18	0.15	0.60	1/2
II	0.09	0.32	0.20	1.50	2/3
III	0.11	0.40	0.50	2.50	1

Para los valores del grupo A se tuvieron que multiplicar los valores de la *Tabla 1* por 1.5 dando como resultado los valores que se muestran en la *Tabla 2*.

Tabla 2 Parámetros de los espectros de diseño para estructuras del grupo A

Parámetros de los espectros de diseño para estructuras del grupo A					
Tipo de terreno	a_o	**c**	**Ta (s)**	**Tb (s)**	**r**
I	0.08	0.27	0.23	0.90	3/4
II	0.14	0.48	0.30	2.25	1
III	0.17	0.60	0.75	3.75	1 1/2

1.4.1. Tipo I

Roca o suelos de consistencia de muy firme a dura para limos y arcillas o de compacidad densa a muy densa para materiales granulares. Suelos de origen eólico volcánico.

1.4.2. Tipo II

Suelos de consistencia de media a firme para limos y arcillas o depósitos arenosos de compacidad media, o bien, capas intercaladas de estos materiales.

1.4.3. Tipo III

Suelos de consistencia de muy blanda a blanda para limos y arcillas o depósitos arenosos de compacidad suelta a muy suelta. Depósitos lacustres y aluviales.

1.5. Espectro de Diseño

Se define un espectro de diseño de aceleración que describe cómo las fuerzas sísmicas varían con la frecuencia para diferentes periodos de vibración. En este caso utilizamos las fórmulas brindadas en el código reglamentario (NTCS Puebla 2017) para calcular la ordenada del espectro de aceleraciones, expresada como fracción de la aceleración de la gravedad, que está dada por las siguientes expresiones:

$$a = a_0 + (c - a_0)\frac{T}{T_a}, \qquad \text{si } T < T_a$$

$$a = c; \qquad \text{si } T_a \leq T \leq T_b$$

$$a = c\left(\frac{T_b}{T}\right)^r; \qquad \text{si } T > T_b$$

Donde:

a_0: Coeficiente de aceleración del terreno.

c: Coeficiente sísmico.

T: Periodo natural de interés.

T_a: Límite inferior de la meseta espectral.

T_b: Límite superior de la meseta espectral.

r: Exponente que depende del tipo de terreno.

1.6. Coeficientes Sísmicos

Se utilizaron coeficientes de respuesta sísmica para calcular las fuerzas sísmicas actuantes en los marcos.

1.7. Factor de comportamiento sísmico

Existen diferentes valores del factor de comportamiento sísmico (Q) para los cuales se deben cumplir ciertas características. Sus valores pueden ser los siguientes

- Q = 1
- Q = 2
- Q = 3
- Q = 4

Deben cumplir algunas características para poder utilizar tal valor.

1.7.1. "En el caso Q = 4

Se usará Q = 4 cuando se cumplan los requisitos siguientes:

a) La resistencia en todos los entrepisos es suministrada exclusivamente por marcos no contra venteados de acero o concreto reforzado, o bien por marcos contra venteados o con muros de concreto reforzado en los que en cada entrepiso los marcos son capaces de resistir, sin contar muros ni contravientos, cuando menos 50 por ciento de la fuerza sísmica actuante.

b) Si hay muros ligados adecuadamente en todo su perímetro a los marcos estructurales o a castillos y dalas ligados a los marcos, éstos se deben tener en cuenta en el análisis de la estructura, pero su contribución a la capacidad ante fuerzas laterales sólo se tomará en cuenta si estos muros son de piezas macizas, y los marcos, sean o no contra venteados, y los muros de concreto reforzado son capaces de resistir al menos 80 por ciento de las fuerzas laterales totales sin la contribución de los muros de mampostería.

c) El mínimo cociente de la capacidad resistente de un entrepiso entre la acción de diseño no difiere en más de 35 por ciento del promedio de dichos cocientes para todos los entrepisos. Para verificar el cumplimiento de este requisito, se calculará la capacidad resistente de cada entrepiso teniendo en cuenta todos los elementos que puedan contribuir a la resistencia, en particular los muros ligados a la estructura en la forma especificada en el requisito b).

d) Los marcos y muros de concreto reforzado cumplen con los requisitos que fijan para marcos y muros dúctiles las NTCC.

e) Los marcos rígidos de acero satisfacen los requisitos para marcos dúctiles que fijan las NTCA.

1.7.2. En el caso de que Q = 3:

Se usará Q = 3 cuando se satisfacen las condiciones 1.6.1.b, 1.6.1.d y 1.6.1.e y en cualquier entrepiso dejan de satisfacerse las condiciones 1.6.1a o 1.6.1c especificadas para el caso 1.6.1, pero la resistencia en todos los entrepisos es suministrada por columnas de acero o de concreto reforzado con losas planas, por marcos rígidos de acero, por marcos de concreto reforzado, por muros de este material, por combinaciones de éstos y marcos o por diafragmas de madera contrachapada. Las estructuras con losas planas deberán además satisfacer los requisitos que sobre el particular marcan las NTCC.

1.7.3. En caso de Q = 2

Se usará Q = 2 cuando la resistencia a fuerzas laterales es suministrada por losas planas con columnas de acero o de concreto reforzado, por marcos de acero o de concreto reforzado, contra venteados o no, o por muros o columnas de concreto reforzado, que no cumplen en algún entrepiso lo especificado para los casos 1.6.1 y 1.6.2, o por muros de mampostería de piezas macizas confinados por castillos, dalas, columnas o trabes de concreto reforzado o de acero, que satisfacen los requisitos de las NTCM, o diafragmas construidos con duelas inclinadas o por sistemas de muros formados por duelas de madera horizontales o verticales combinados con elementos diagonales de madera maciza. También se usará Q = 2 cuando la resistencia es suministrada por elementos de concreto prefabricado o presforzado, con las excepciones que sobre el particular marcan las NTCC.

1.7.4. En el caso de Q = 1

Se usará Q = 1 en estructuras cuya resistencia a fuerzas laterales es suministrada al menos parcialmente por elementos o materiales diferentes de los arriba señalados, a menos que se haga un estudio que demuestre a satisfacción de la

Dirección que puede emplearse un valor más alto que el que aquí se especifica. En todos los casos se empleará para toda la estructura en la dirección de análisis el valor mínimo de Q que corresponde a los diversos entrepisos de la estructura en dicha dirección. El factor Q puede diferir en las dos direcciones ortogonales en que se analiza la estructura, según sean las propiedades de ésta en dichas direcciones." (Avilés López, Corona Carlos, González Espinoza, & Lira, 2017)

1.8. Análisis Sísmico

La norma permite realizar análisis estáticos o dinámicos para evaluar el comportamiento de las estructuras bajo carga sísmica.

1.8.1. Análisis Sísmico Dinámico

El análisis sísmico dinámico es un proceso utilizado en ingeniería estructural para evaluar cómo una estructura, como marcos, un edificio o un puente, responderá a las fuerzas generadas por un sismo.

Para realizar el análisis sísmico dinámico se implementaron las expresiones de Wilbur para el cálculo de la rigidez de entrepiso, mismas que servirán para calcular los desplazamientos laterales.

Las expresiones o fórmulas de Wilbur se aplican a marcos regulares formados por piezas de momentos de inercia constante en los que las deformaciones axiales son despreciables y las columnas tienen puntos de inflexión.

1.8.1.1. Fórmulas de Wilbur

A continuación, se presentan las fórmulas de Wilbur, en las que cada uno de los términos corresponde a:

h_o = es la altura del nivel de arriba

h_m= es la altura del nivel de abajo

h_n = es la altura del nivel a calcular "D_i"

K_t y K_c equivalen a las rigideces de trabes y columnas, mientras que D equivale al coeficiente de Wilbur.

Para el primer entrepiso (empotramiento en la cimentación).

$$K_1 = \frac{48E}{D_1 h_1}$$

$$D_1 = \frac{4h_1}{\sum K_{c1}} + \frac{(h_1 + h_2)}{\sum K_{t1} + \frac{\sum K_{c1}}{12}}$$

Para el primer entrepiso (articulación en la cimentación).

$$K_1 = \frac{24E}{D_1 h_1}$$

$$D_1 = \frac{h_1}{\sum K_{c1}} + \frac{(2h_1 + h_2)}{\sum K_{t1}}$$

Para el segundo entrepiso (empotramiento en la cimentación).

$$K_2 = \frac{48E}{D_2 h_2}$$

$$D_2 = \frac{4h_2}{\sum K_{c2}} + \frac{(h_1 + h_2)}{\sum K_{t1} + \frac{\sum K_{c1}}{12}} + \frac{(h_2 + h_3)}{\sum K_{t2}}$$

Para el segundo entrepiso (articulación en la cimentación).

$$K_2 = \frac{48E}{D_2 h_2}$$

$$D_2 = \frac{4h_2}{\sum K_{c2}} + \frac{(2h_1 + h_2)}{\sum K_{t1}} + \frac{(h_2 + h_3)}{\sum K_{t2}}$$

Para entrepisos intermedios.

$$K_n = \frac{48E}{D_n h_n}$$

$$D_n = \frac{4h_n}{\sum K_{cn}} + \frac{(h_m + h_n)}{\sum K_{tm}} + \frac{(h_n + h_o)}{\sum K_{tn}}$$

Para piso de azotea.

$$D_R = \frac{4h_n}{\sum K_{cn}} + \frac{(h_m + h_n)}{\sum K_{tm}} + \frac{h_n}{\sum K_{tn}}$$

Aunque las fórmulas de Wilbur (Castillo, 2020) son aproximadas, se consideran lo suficientemente precisas para el fin que se persiguió en esta investigación.

1.9. Revisión de estados límite

Los códigos especifican desplazamientos laterales máximos admisibles para estructuras, lo que influye en el diseño de sistemas de refuerzo y amortiguamiento.

Para este proyecto utilizamos la revisión de estados límite para los desplazamientos horizontales del marco, su límite debe de ser 0.006 si los muros están ligados y, si no es así, su valor será de 0.012 veces el valor de la deformación por nivel.

Algunos de los otros estados límite de servicio o de falla que existen son los siguientes:

- Rotura de vidrios: En fachadas, tanto interiores como exteriores, la colocación de los vidrios en los marcos, o la liga de éstos con la estructura, serán tales que las deformaciones de ésta no afecten a los vidrios. Para ello, se verificará que alrededor de cada tablero de vidrio o cada marco exista una holgura no menor que el desplazamiento relativo entre los extremos del tablero o marco, calculado a partir de la deformación por cortante de entrepiso y dividido entre *1+H/B*, donde *B* es la base del tablero o marco y *H* su altura.
- Choques contra estructuras adyacentes: Toda construcción deberá separarse de sus linderos con los predios vecinos una distancia no menor de 5 cm ni menor que el desplazamiento horizontal del nivel de que se trate. El desplazamiento horizontal se obtendrá como el calculado pero aumentado en 0.001, 0.003 o 0.005 de la altura de dicho nivel sobre el desplante para los terrenos tipo I, II o III, respectivamente. Si se emplea el método simplificado de análisis sísmico, la separación mencionada no será, en ningún nivel, menor de 5

cm ni menor de la altura del nivel sobre el desplante multiplicada por 0.007, 0.009 o 0.011 para los terrenos tipo I, II o III, respectivamente. La separación entre cuerpos de una misma estructura o entre estructuras adyacentes será cuando menos igual a la suma de las que de acuerdo con las especificaciones precedentes corresponden a cada una.

- Falla de cimentación: Al revisar con respecto al estado límite de falla de la cimentación se tendrá en cuenta la fuerza de inercia horizontal que obra en el volumen de suelo que se halla bajo los cimientos y que potencialmente se desplazaría al fallar el suelo en cortante, estando dicho volumen sujeto a una aceleración horizontal igual a a_0 veces la aceleración de la gravedad, donde a_0 es el coeficiente de aceleración del terreno que se consigna en la *Tabla 1*.

CAPÍTULO 2: ALGORITMO GENÉTICO

2.1. Definición

Una de las definiciones de algoritmos genéticos la propone (Darwin, 1859) como:

"Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos. A lo largo de las generaciones, las poblaciones evolucionan en la naturaleza acorde con los principios de la selección natural y la supervivencia de los más fuertes".

2.2. Extensiones y modificaciones

Estas son algunas extensiones y modificaciones del algoritmo genético simple de las cuales se utilizaron en la investigación:

2.2.1. Población

Esta se divide en dos:

- Tamaño de la población: en 1992 Alander se basó en evidencia empírica cuando sugirió que un tamaño de población comprendida entre ℓ y 2ℓ es suficiente para atacar con éxito los problemas que él consideró, así acabando con la consecuencia que el estudio teórico de Goldberg (1989) trajo consigo, ya que tuvo como conclusión que el tamaño óptimo de la población para ristras (una ristra es una serie larga de cosas, iguales o análogas, que están o se mencionan una tras otra) de longitud ℓ, con codificación binaria, crece exponencialmente con el tamaño de la ristra.
- Población Inicial: Habitualmente la población inicial se escoge generando ristras al azar, pudiendo contener cada gen uno de los posibles valores del alfabeto con probabilidad uniforme. En los pocos trabajos que existen sobre este aspecto, se constata que esta inicialización no aleatoria de la población inicial, puede acelerar la convergencia del algoritmo genético.

2.2.2. Función Objetivo

La función objetivo en algoritmos genéticos es una parte fundamental del proceso de optimización. Esta función define el problema que se está tratando de resolver y proporciona una medida de qué tan bueno es un conjunto de soluciones candidatas.

La función objetivo debe ser definida por el usuario y tiene las siguientes características:

2.2.2.1. Evaluación

Toma un conjunto de soluciones candidatas (llamadas cromosomas o individuos) como entrada y devuelve un valor numérico que representa la calidad de esas soluciones. Cuanto mayor sea el valor o menor, dependiendo del planteamiento, mejor será la solución.

2.2.2.2. Optimización

El objetivo del algoritmo genético es encontrar la mejor solución posible para el problema dado. Por lo tanto, la función objetivo debe ser formulada de tal manera que se busque maximizar (o minimizar) su valor. Por ejemplo, si estás buscando la mejor combinación de parámetros para un modelo de "*machine learning"* (es la ciencia del desarrollo de algoritmos y modelos estadísticos que los sistemas informáticos utilizan para realizar tareas sin instrucciones explícitas, en lugar de depender de patrones y razonamientos), la función objetivo podría ser la precisión del modelo, y se querrá maximizarla.

2.2.2.3. Dominio

La función objetivo debe estar definida en un dominio específico que sea relevante para el problema. Por ejemplo, si se está optimizando el diseño de una pieza mecánica, la función objetivo podría estar relacionada con las características físicas de la pieza.

2.2.3. Selección

La función de selección es proporcional a la función objetivo, en la cual cada valor tiene una probabilidad de ser seleccionado como máximo/mínimo.

La evaluación se encarga de decodificar los genes del cromosoma, para convertirlos en los parámetros del problema, para después encontrar una solución a partir de ellos. Dentro de la evaluación se califica la solución en función de lo cercano que esté a las condiciones requeridas, a esto se le llama *"fitness"* (en inglés). El *"fitness"* determina siempre los cromosomas que se van a reproducir, y aquellos que se van a eliminar.

Existen varias técnicas de selección que se pueden utilizar en los algoritmos genéticos, para poder pasar a la generación siguiente, enseguida se presentan algunos citados por (Marczyk, 2004):

- Selección elitista: se hace una selección de los individuos más fuertes en cada generación. Aunque la mayoría de las veces no se realiza una selección puramente elitista, ya que se realiza una modificación para que los individuos seleccionados sean copiados a generaciones próximas si es que no existen individuos mejores de la nueva generación.
- Selección proporcional a la aptitud: aquí los individuos más aptos cuentan con mayor probabilidad de ser seleccionados, pero no la certeza.
- Selección por ruleta: esta técnica de selección es proporcional a la aptitud, en la que la probabilidad de que un individuo sea seleccionado es proporcional a la diferencia entre su aptitud y la de sus competidores.
- Selección escalada: al incrementarse la aptitud promedio de la población, la fuerza selectiva aumenta y, por lo tanto, la función de la aptitud se vuelve más rigurosa.
- Selección por torneo: se eligen aleatoriamente subgrupos de individuos de la población, y los miembros compiten entre ellos. Sólo se elige a un individuo de cada subgrupo.

- Selección por estado estacionario: los individuos selectos de la generación descendente vuelven al acervo genético preexistente, reemplazando a los miembros menos aptos de la siguiente generación.
- Selección por rango: a cada individuo de la población se le asigna un rango en función de su aptitud y, por lo tanto, la selección se basa en este rango.
- Selección generacional: la descendencia de los individuos seleccionados se convierte en la nueva generación. En este tipo de selección no se conservan individuos entre las generaciones.
- Selección jerárquica: los individuos pasan por varias rondas de selección, donde los primeros niveles de selección son más rápidos y menos discriminatorios, y a medida que van avanzando, éstos se vuelven más rigurosos en la evaluación. La gran ventaja de esta selección es la reducción del tiempo de cálculo, al ser una evaluación más rápida y menos selectiva para eliminar a aquellos individuos que no son tan aptos.

Comúnmente se utiliza la combinación de alguno de estos métodos, aunque hay algunos que son mutuamente exclusivos.

2.2.4. Cruce

Existen varios tipos de operadores de cruce en algoritmos genéticos, cada uno con su enfoque y aplicabilidad en diferentes problemas. Aquí se mencionan algunos de los tipos más comunes.

2.2.4.1. Cruce de un solo punto

En este tipo de cruce, un punto de corte se elige aleatoriamente en los cromosomas de los padres, y los segmentos anteriores y posteriores a ese punto se intercambian para crear dos descendientes.

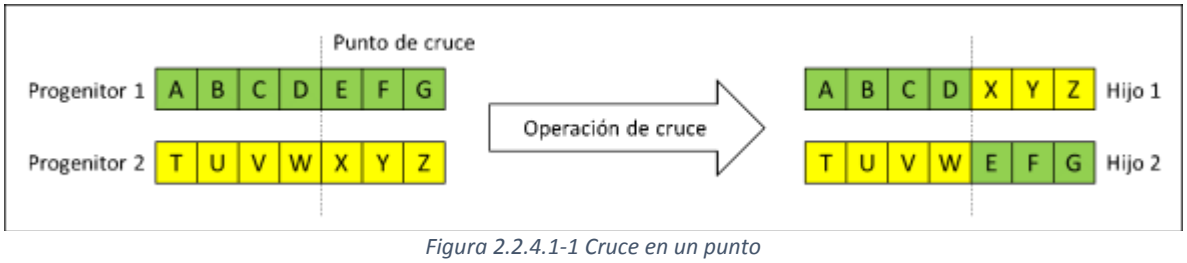


Figura 2.2.4.1-1 Cruce en un punto

2.2.4.2. Cruce de dos puntos

Similar al cruce de un solo punto, pero con dos puntos de corte, lo que resulta en tres segmentos que se intercambian entre los padres para generar dos descendientes.

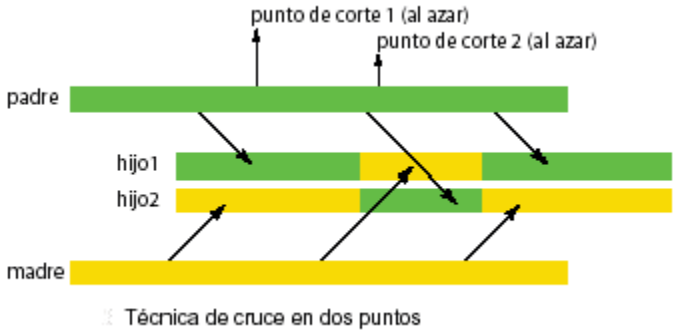


Figura 2.2.4.2-1 Cruce en dos puntos

2.2.4.3. Cruce uniforme

En este tipo de cruce, se eligen aleatoriamente genes individuales de los padres para formar los genes de los descendientes. Esto permite una mayor exploración del espacio de búsqueda y puede preservar características importantes de ambos padres.

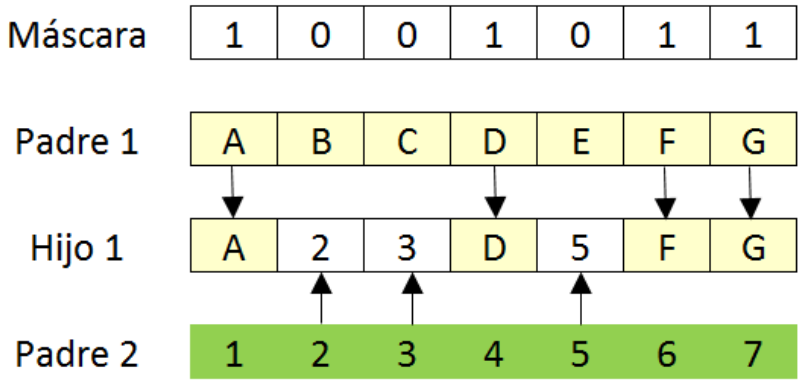


Figura 2.2.4.3-1 Cruce uniforme (Marcos, 2010)

Siendo el último cruce mencionado el que se utilizó para este algoritmo.

2.2.5. Mutación

"La mutación se considera un operador básico, que proporciona un pequeño elemento de aleatoriedad en el entorno de los individuos de la población. Si bien se admite que el operador de cruce es el responsable de efectuar la búsqueda a lo largo del espacio de posibles soluciones, también parece desprenderse de los experimentos efectuados por varios investigadores que el operador de mutación va ganando en importancia a medida que la población de individuos va convergiendo" (Davis, 1993).

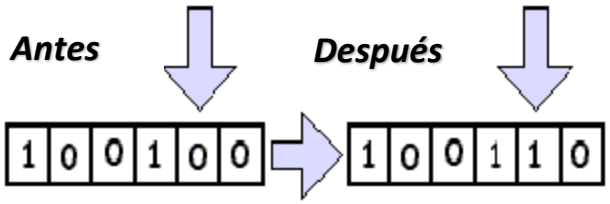


Figura 2.2.5-1 Mutación

2.2.6. Evaluación de Algoritmos Genéticos

Las tres medidas de evaluación que se tratarán en este apartado, fueron introducidas por (De Jong, 1975), y se conocen como:

- Evaluación on-line: mide el comportamiento medio de todas las ristras generadas hasta el tiempo T.
- Evaluación off-line: se refiere al comportamiento del algoritmo genético en su proceso de convergencia hacia el óptimo.
- Evaluación basada en el mejor: r trata de evaluar el algoritmo genético por medio del mejor valor de la función de evaluación encontrado en la evolución.

2.3. Algoritmos Genéticos en Estructuras

Los algoritmos genéticos ya han sido utilizados para resolver distintos problemas en el área de estructuras, como por ejemplo en estructuras metálicas, armaduras planas, en vigas, etc. Un ejemplo es la tesis presentada en 2015 por el Ingeniero Hernán David Salazar Valverde con el título de *"Optimización de vigas-trabe*

tipo I armadas mediante placas de acero, utilizando algoritmos genéticos" (Valverde Salazar, 2015) obteniendo la confirmación de su hipótesis logrando un proyecto exitoso. Para esta investigación, se propuso estructuras elaboradas y finalizadas para después realizar un seudocódigo para resolver el problema planteado. Estudios similares han sido realizados por diferentes investigadores, entre ellos "A*plicación de algoritmos genéticos en Ingeniería Civil"* (Jarquín Laguna, 2014) etc., aplicados en el área de Ingeniería Civil.

CAPÍTULO 3: FORMULACIÓN DEL ALGORITMO GENÉTICO

Para lograr el objetivo del trabajo se realizó la formulación del algoritmo a través de los diferentes aspectos que lo componen y que se describen a continuación.

3.1. Elementos de la Programación del Proyecto

Se utilizan tres marcos de distintas dimensiones (crujías, niveles, claros, etc.) para demostrar la eficacia de la optimización. Después, los datos de los tres distintos marcos se tomaron para introducirlos en la programación del algoritmo y continuar con la construcción correspondiente, haciendo el análisis estático y el análisis dinámico modal, las cuales especificaron pesos de entrepiso, rigidez, cálculo de las matrices de masas, las distorsiones por modo, entre otras. Específicamente, la programación nos indica:

- Cálculos y sumatoria de peso de entrepiso.
- Obtención de las inercias de trabe y las inercias de columnas.
- Obtención de las rigideces mediante la aplicación de las expresiones de Wilbur.
- Calculo de la cortante basal, las fuerzas y las cortantes por el Análisis dinámico.
- Obtención de las distorsiones y desplazamientos, así como los desplazamientos totales, por el método dinámico.
- Cálculo para obtener las masas.
- Elaboración de la matriz de masas, matriz de masas inversa, matriz de rigidez, matriz dinámica del sistema.
- Cálculo de los Eigenvectores y Eigenvalores gracias a la función *NumPY*.
- Obtención de los periodos y las frecuencias.
- Cálculo de las ordenadas espectrales de desplazamientos y de aceleraciones.
- Cálculos de los Parámetros modales
- Obtención de las distorsiones y desplazamientos, por el método de análisis modal.

3.2. El Algoritmo Genético

El algoritmo tiene las siguientes características:

- Una cadena: para tener los valores otorgados por la función *random.uniform* que nos dará valores al azar, siempre y cuando respete los parámetros establecidos.
- Selección por torneo: dos individuos con características similares son seleccionados aleatoriamente de la población y los mejores cinco individuos son pasados a la generación próxima. El proceso se repite hasta que se llega a la última generación otorgada por el usuario.
- "*Crossover*" parcial: este operador intercambia información contenida en dos individuos padres escogidos de la población para producir dos descendientes, los cuales reemplazaran a los individuos padres. En cada generación, el número de veces que el "*crossover*" es aplicado (N_x) es determinado por la probabilidad de "*crossover*" (P_x) y el tamaño de la población $(P_x) \cdot N_X = N \cdot P_X$
- Mutación: este operador selecciona aleatoriamente un individuo de la población de las cadenas, y después escoge dos elementos de la cadena de este individuo para intercambiar posiciones.

3.3. Parámetros del Algoritmo Genético

Se especifican los siguientes parámetros para los siguientes tres marcos:

- Tamaño de la población: 100 individuos.
- Número de generaciones: 50 generaciones.
- Probabilidad de *crossover*: 0.05
- Probabilidad de mutación: 0.07

3.4. Resumen del Procedimiento del Algoritmo Genético (Diagrama de flujo)

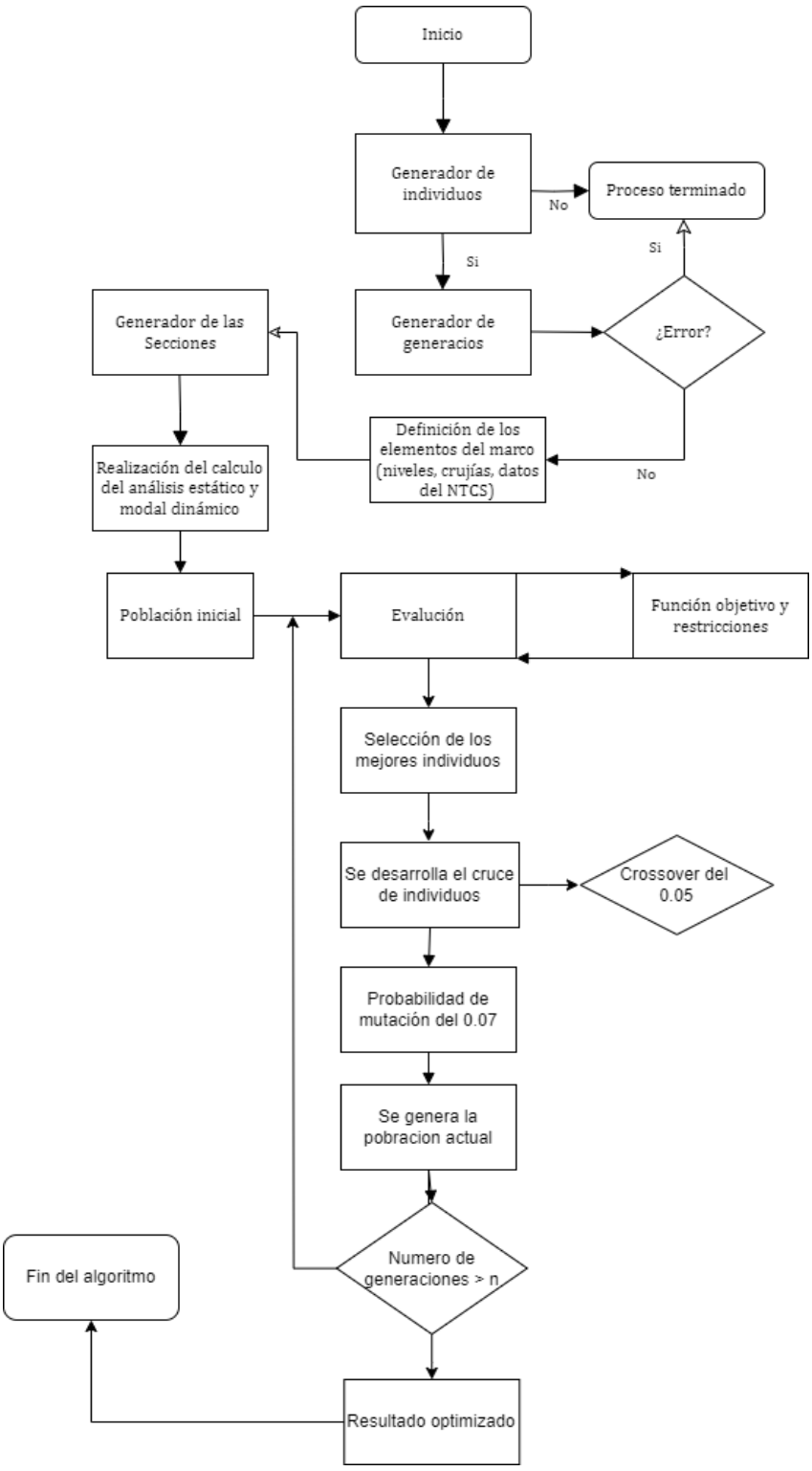


Figura 3.2-1 Digrama de flujo algoritmo genético

CAPÍTULO 4: SÍNTESIS DEL ALGORITMO GENÉTICO EN LA PROGRAMACIÓN DEL PROYECTO

4.1. Definición del problema

Los algoritmos genéticos son simulaciones de la selección natural que pueden resolver problemas de optimización, es decir, son considerados como una estrategia de búsqueda que imita la teoría de la evolución de la vida en la naturaleza. Ahora bien, si un algoritmo genético es una técnica de búsqueda iterativa inspirada en los principios de selección natural, es importante aclarar que los algoritmos genéticos no buscan modelar la evolución biológica, sino derivar estrategias de optimización. La idea de los algoritmos genéticos es optimizar (hallar el máximo o mínimo) una función objetivo utilizando los principios de selección natural sobre los parámetros de la función.

El problema es que las secciones de las columnas de los marcos no siempre están optimizadas, dado que esto causa una gran pérdida de tiempo en buscar una sección que se acerque al límite al desarrollar los marcos, sería complicado jugar con su valores y desarrollar todo un procedimiento para calcular valores (fuerzas, desplazamientos, distorsiones, etc.) marco por marco y que puede que, aun así, sus distorsiones no cumplan las normas en todos los niveles, por lo que al desarrollar un algoritmo genético por medio de Python, nos ayudó a reducir el tiempo, dando que por la codificación hace las iteraciones que se le pida y así busque la óptima, haciendo el diseño de marcos más eficaz.

4.2. Representación de datos

Para desarrollar el código utilizamos como principal herramienta un módulo que obtiene acceso al código en otro módulo por el proceso de importarlo, a este módulo lo llamamos con la fórmula más fácil y común de invocar con el código de "*import"* lo que este módulo contiene son paquetes que previamente instalamos como son "*pandas, math, NumPy, random, workbook, copy y openpyxl*". (Ver *figura 4.2-1*)

"Pandas" es una biblioteca de Python que proporciona estructuras de datos flexibles y herramientas de análisis de datos. Mientras que el módulo *"math"* es una biblioteca estándar de Python que proporciona funciones y constantes matemáticas

para realizar operaciones matemáticas más avanzadas que las básicas. Algunas de las funcionalidades incluyen operaciones algebraicas, trigonométricas, logarítmicas y estadísticas.

"NumPy" proporciona una amplia gama de funciones para trabajar con matrices y realizar operaciones matriciales eficientes. Mientras que *"random"* lo utilizamos para generar números pseudoaleatorios. Proporciona funciones para realizar operaciones relacionadas con la aleatoriedad, como la generación de números aleatorios, mezcla de secuencias y selección aleatoria de elementos.

```
1   import math
2   import random
3   import numpy as np
4   import pandas as pd
5   import xlwt
6   from xlwt import Workbook
7   import colorama
8   import openpyxl
9   import copy
10
11  from colorama import init, Fore
```

Figura 4.2-1 Módulos

También utilizamos entradas para números enteros (para ingresar datos como el número de niveles, de crujías, de sujetos, etc.) con la función "*input ( )"* que por sí sola siempre devuelve un objeto, en este caso los valores, en clase *"str (string)".* Entonces, para tomar entradas enteras, tenemos que convertir esas entradas en números enteros usando la función *"int ( )*" incorporada de Python. (Ver *figura 4.2-2*)

```
13  n_ind=int(input("Numero de individuos de la población: "))
14  n_gen=int(input("Número de generaciones: "))
```

Figura 4.2-2 Entrada int combinada con Input

Otro tipo de entrada que se utilizó en el código con la combinación de "*input ( )"* fue la de "*float ( )"*, que esta entrada nos permite obtener los valores no solo enteros, sino con números decimales. (Ver *figura 4.2-3*)

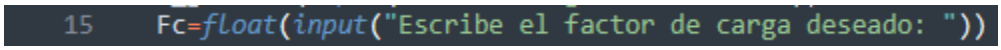


Figura 4.2-3 Entrada Float combinada con input

Utilizamos las sentencias condicionales que también se denominan sentencias de decisión. Se utilizan para ejecutar un bloque específico de código si las condiciones dadas son verdaderas o falsas. Python admite las condiciones lógicas matemáticas habituales:

- Es igual a: a == b
- No es igual a: a != b
- Menos que: a < b
- Menor o igual a: a <= b
- Mayor que: a > b
- Mayor o igual a: a >= b

Estas condiciones se pueden utilizar de diversas formas, siendo las más comunes las declaraciones "*if*" y los bucles.

Las "declaraciones "*if*" se escriben utilizando la palabra clave "*if*", que utilizamos en muchas partes del código para desarrollar opciones para los usuarios del algoritmo y para ver si el algoritmo cumple con las especificaciones requeridas.

A lo que, de igual forma, otra de las condicionales que utilizamos fue "*elif*" que se utiliza para verificar múltiples condiciones en orden. Si la condición de un "*if*" o "*elif*" es verdadera, el bloque de código dentro de esa condición se ejecutará y las condiciones restantes serán ignoradas.

Y, por último y no menos importante, la condición "*else*" se utiliza junto con "*if*" para ejecutar un bloque de código si la condición del "*if*"es falsa.

Estas condicionales son fundamentales para controlar el flujo de ejecución en programas Python y permiten que tu código tome decisiones basadas en ciertas condiciones. (Ver *figura 4.2-4*)

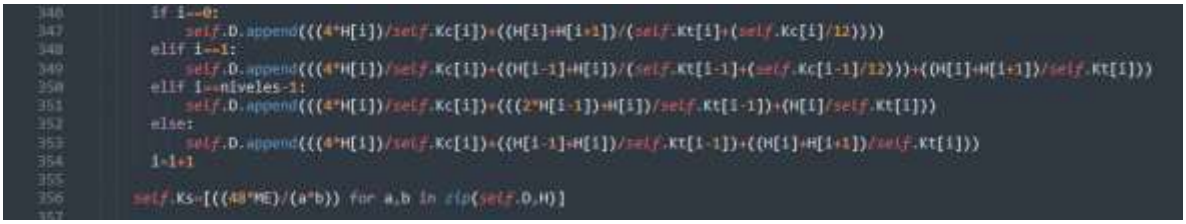


Figura 4.2-4 Condicionales (if, elif y else)

En programa Python nos permitió trabajar con el bucle *"while"* que es una estructura de control que permite ejecutar un bloque de código repetidamente mientras una condición específica sea verdadera.

El bucle *"while"* evalúa la condición antes de ejecutar el bloque de código. Si la condición es verdadera, el código dentro del bucle se ejecuta y luego la condición se vuelve a evaluar. Este proceso se repite hasta que la condición se vuelva falsa, momento en el que el bucle *"while"* se detiene y la ejecución del programa continúa con las instrucciones siguientes al bucle.

Es importante tener cuidado al utilizar bucles *"while"* para evitar entrar en un bucle infinito, donde la condición nunca se vuelve falsa y el bucle nunca termina. Para evitar bucles infinitos, nos aseguramos de que la condición dentro del bucle *"while"* eventualmente se vuelva falsa durante la ejecución del programa o, de igual forma, nos aseguramos de utilizar la declaración de ruptura. Con la sentencia *"break"* podemos detener el ciclo incluso si la condición *"while"* es verdadera. (Ver *figura 4.2-5*)

```
while True:
    print("Tipo de terreno")
    print("1. ", Za)
    print(Za1, "(Coeficiente Sismico)")
    print("2. ", Zb)
    print(Zb1, "(Coeficiente Sismico)")
    print("3. ", Zc)
    print(Zc1, "(Coeficiente Sismico)")
    print("4. SALIR")
    opc= int(input("Elije el TIPO DE TERRENO: "))
    if opc==1:
        print(Za)
        print("1.- %.2f" %Za1)
        cs=Za1
        a0=0.0005
        Ta=0.15
        Tb=0.60
        r=1/2
        if cs== Za1:
            print("1.1- ", Q1)
            minc=.20
            print("1.3- ", Q3)
            minc=.20
            print("1.4- ", Q4)
            minc=.25
            print("1.5- ", Q5)
            minc=.25
    elif opc==2: ...
    elif opc==3:
        print(Zc)
        print("3.- %.2f" %Zc1)
        cs= Zc1
        a0=0.011
        Ta=0.50
        Tb=2.5
        r=1
        if cs==Zc1:
            print("3.1- ", Q1)
            minc=.20
            print("3.3- ", Q3)
            minc=.20
            print("3.4- ", Q4)
            minc=.25
            print("3.5- ", Q5)
            minc=.25
    fq= float(input("Elije el Factor de comportamiento sismico (Q): "))
    if fq== Q1 or Q3 or Q4 and Q5:
        print(fq)
    else:
        print("opcion invalida")
    break
```

Figura 4.2-5 Bucle While

Se utilizó una declaración que es habitual en Python, *"return"* es una declaración que se utiliza dentro de una función para devolver un valor al llamado de la función. Cuando una función alcanza una declaración *"return"*, la ejecución de la función se detiene y el valor especificado después de *"return"* se devuelve como resultado de la función.

"return" es fundamental para que las funciones en Python sean capaces de proporcionar resultados que pueden ser utilizados en otras partes del programa. (Ver *figura 4.2-6*)

```
298         while 2 ** j <= x * 100:
299             j = j + 1
300             aux = j
301         self.num_bits=aux
302 
303         if x==0 or valor_variable==limites_inf:
304             valor_venbin=1
305         else:
306             valor_venbin = (round((((2 ** num_bits)-1) / x) * (abs(valor_variable - limites_inf))))
307 
308         self.bina = (bin(valor_venbin)[2:])
309         """print("bina   "+str(self.bina))
310         print("num bits self   " + str(self.num_bits))
311         print("num bits   " + str(num_bits))
312         print("X   " + str(x))
313         print("limite inf   " + str(limites_inf))
314         print("Valor R   " + str(valor_variable))
315         print("Valor F   " + str(valor_venbin))"""
316         auxx = ''
317         h = 0
318         if num_bits < len(self.bina):
319           self.bina=self.bina[num_bits:]
320         while num_bits != len(self.bina) + h:
321             auxx = auxx + '0'
322             h = h + 1
323         auxd = auxx + self.bina
324         self.bina = auxd
325 
326 
327         return (self.bina)
```

Figura 4.2-6 Declaración return

Otra de las declaraciones utilizadas es la denomina *"pass"*, es una declaración que indica que no hay que hacer nada. Se utiliza cuando la sintaxis requiere una declaración, pero no se necesita realizar ninguna acción. En otras palabras, *"pass"* es simplemente un marcador de posición que indica que no hay ninguna operación específica que debe realizarse en ese punto del código. (Ver *figura 4.2-7*)

```
185     opc= int(input("""¿Qué grupo es?:
186         1)A    2)B
187     """))
188     if opc==1:
189         cs= cs*1.5
190         a0=a0*1.5
191         Ta=Ta*1.5
192         Tb=Tb*1.5
193         r=r*1.5
194 
195     if opc==2:
196       pass
```

Figura 4.2-7 Declaración pass

Usamos una combinación de tres funciones como lo son "*round ( )*", "*append*" y "*uniform ( )*", para que el programa nos diera valores redondeados al azar de las secciones y se agregaran en una lista, respetando las fórmulas y las órdenes programadas para las secciones de las columnas y trabes.

La función "*round ( )*" nos permite redondear un número a la cantidad de decimales especificada o al número entero más cercano si no se proporciona ningún número de decimales, en este caso elegimos dos decimales para cada sección, dado que si no se especifica el número de decimales, este redondeara al número entero más cercanos.

Mientras que la función "*append ( )*" se utiliza para agregar un elemento a la lista; a lo que una lista es una estructura que puede contener elementos de diferentes tipos, en este caso siendo utilizado para que añadimos el número de lugares (niveles) que se tiene.

La función "*uniform ( )*" se utiliza para generar un número decimal aleatorio dentro de un rango especificado, por lo que se creó un límite inferior y un superior para que este fuera dado al azar. (Ver *figura 4.2-8*)

```
print("SECCIÓN ANCHO TRABE")
Ast=[]
i=0
while i<niveles:
  if i==0:
    limites_inf = Hst[i]/2.5
    limites_sup=Hst[i]/2.5
  else:
    limites_sup=Ast[i-1]

  Ast.append(round(random.uniform(limites_inf, limites_sup),2))
  i= i + 1
```

Figura 4.2-8 Funciones para la creación de las secciones.

En Python una de las palabras claves más utilizadas es "*def*", se utiliza para definir una función. Al crear una función, puedes usar "*def*" seguido del nombre de la función y la lista de parámetros entre paréntesis como se muestra en la *figura 4.2-9.*

```
293     def deci_a_bin (self, valor_variable, limites_inf, x,num_bits):
294       valor_venbin = 0
295       self.bina = ""
296       j = 0
297       aux = 1
298       while 2 ** j <= x * 100:
299           j = j + 1
300           aux = j
301       self.num_bits=aux
302
303       if x==0 or valor_variable==limites_inf:
304           valor_venbin=1
305       else:
306           valor_venbin = (round((((2 ** num_bits)-1) / x) * (abs(valor_variable - limites_inf))))
307
308       self.bina = (bin(valor_venbin)[2:])
309       """print("bina   "+str(self.bina))
310       print("num bits self    " + str(self.num_bits))
311       print("num bits    " + str(num_bits))
312       print("X    " + str(x))
313       print("limite inf    " + str(limites_inf))
314       print("Valor R    " + str(valor_variable))
315       print("Valor F    " + str(valor_venbin))"""
316       auxx = ''
317       h = 0
318       if num_bits < len(self.bina):
319         self.bina=self.bina[num_bits:]
320       while num_bits != len(self.bina) + h:
321           auxx = auxx + '0'
322           h = h + 1
323       auxd = auxx + self.bina
324       self.bina = auxd
325
326
327       return (self.bina)
```

Figura 4.2-9 Ejemplo de la aplicación de "def".

Para obtener el valor absoluto de un número, es decir, su distancia respecto al cero en la recta numérica, sin considerar su dirección implementamos la función "*abs ( )*".

Esta función es útil cuando se necesita el valor positivo de un número, independientemente de su signo. (Ver *figura 4.2-10*)

```
293     def deci_a_bin (self, valor_variable, limites_inf, x,num_bits):
294       valor_venbin = 0
295       self.bina = ""
296       j = 0
297       aux = 1
298       while 2 ** j <= x * 100:
299           j = j + 1
300           aux = j
301       self.num_bits=aux
302
303       if x==0 or valor_variable==limites_inf:
304           valor_venbin=1
305       else:
306           valor_venbin = (round((((2 ** num_bits)-1) / x) * (abs(valor_variable - limites_inf))))
```

Figura 4.2-10 Función abs ( ).

Una de las funciones más importantes que se emplearon fue la función *"bin ( )"*. En Python se utiliza para convertir un número entero en su representación binaria. Esto nos sirvió al momento de hacer la cruza y la mutación de los individuos. (Ver *figura 4.2-11*)

```
308        self.bina = (bin(valor_venbin)[2:])
```

Figura 4.2-11 Función Bin ( ).

La función *"len ( )"* devuelve el número de elementos de un objeto. Cuando el objeto es una cadena, la función *"len ( )"* devuelve el número de caracteres de la cadena. (Ver *figura 4.2-12*)

```
318        if num_bits < len(self.bina):
319          self.bina=self.bina[num_bits:]
320        while num_bits != len(self.bina) + h:
321            auxx = auxx + '0'
322            h = h + 1
323        auxd = auxx + self.bina
324        self.bina = auxd
```

Figura 4.2-12 Función len ()

Usamos "*zip ( )*" función que toma iterables (pueden ser cero o más), los agrega en una tupla y los devuelve, en este caso lo utilizamos para calcular "*Sr*" (Desplazamiento Relativo/Drift), *"FiHi"* (Momento de volteo), etc. (Ver *figura 4.2-13*)

```
393        self.FiHi=[(a*b) for a,b in zip(self.F,self.Hi)]
394
395        self.Sr=[(a/b) for a,b in zip(self.V,self.Ks)]
```

Figura 4.2-13 Función zip ()

Tuvimos que crear una clase asignada como "Valores", su palabra clave es "*class*", esta clase es un modelo o prototipo definido por el usuario a partir del cual se crean objetos. Las clases proporcionan una manera de agrupar datos y funcionalidades. La creación de una nueva clase crea un objeto de un nuevo tipo, lo que permite crear nuevas instancias de ese tipo. Cada instancia de clase puede tener propiedades adjuntas

para mantener su estado. Las instancias de clase también pueden tener métodos (definidos por su clase) para modificar su estado.

La clase crea una estructura de datos que nosotros definidos, que contiene sus propios miembros de datos y función miembro, a los que se puede acceder y utilizar creando una instancia de esa clase. Una clase es como un modelo para un objeto.

Cuando se utilizan clases en Python, el término "*self*" se refiere a la instancia de la clase que se está utilizando actualmente. Es muy común utilizar "*self*" como primer parámetro en un método de instancia de una clase. Siempre que llamas a un método en un objeto creado a partir de una clase, el objeto se pasa automáticamente como primer argumento con el parámetro "*self*". Esto le permite modificar las propiedades del objeto y realizar tareas específicas para esa instancia específica.

En conjunto con "*def*" definimos la función *"__init__ ( )"* todas las clases tienen una función llamada así, que siempre se ejecuta cuando se inicia la clase, esto lo utilizamos para asignar valores a las propiedades del objeto u otras operaciones. (Ver *Figura 4.2-14*)

```
258 class valores:
259   def __init__(self):
260     print("La seccion del PERALTE DE LA COLUMNA sera definida aleatoriamente por el programa...")
261
262
263
264     print("SECCIÓN PERALTE COLUMNA")
265     self.Hsc=[]
266     self.num_bits2 = []
267     self.x2 = []
268     i=0
269     while i<niveles:
270       self.limites_inf2 = minc
271       if i==0:
272         self.limites_sup2=  (4*Ast[i])# no mayor 3veces el ancho de columna
273       else:
274         self.limites_sup2=self.Hsc[i-1]
275
276       self.x2.append(self.limites_sup2 - self.limites_inf2)
277       j = 0
278       aux =1
279       while 2**j <= self.x2[i]*100:
280         j = j + 1
281         aux = j
282       self.num_bits2.append(aux)
283       #print(self.num_bits2)
284
```

Figura 4.2-14 Class

Para la realización de los cálculos, utilizamos funciones matemáticas en la biblioteca de Python, como la función "*sum ( )*" que suma todos los elementos de una

lista (o cualquier iterable). Que a lo largo del código la utilizamos en distintas ocasiones como se puede ver un ejemplo en la *Figura 4.2-15.*

```
370     self.SWiHi=sum(self.WiHi)
```

Figura 4.2-15 Función sum ( )

Dado que los marcos son de "*n x m*" números de niveles y crujías tuvimos que utilizar matrices para la realización de operaciones, a lo que a nuestras distintas listas del código se empleó la función *"array ( )",* que va de la mano de la biblioteca "*NumPy"*, para la creación de matrices multidimensionales como se puede ver un ejemplo en la *Figura 4.2-16*.

```
428 ▼   def gen_Kmatrix_np(*Ks):
429         self.Ks = np.array(self.Ks)
430         shape = len(self.Ks)
431         self.K = np.zeros((shape, shape))
432         idx1, idx2 = np.arange(1, shape), np.arange(shape - 1)
433         # Llenamos diagonal principal
434         self.K[-1, -1] = self.Ks[-1]
435         self.K[idx2, idx2] = self.Ks[:-1] + self.Ks[1:]
436         # Llenamos diagonales paralelas superior e inferior
437         self.K[idx1, idx2] =self.K[idx2, idx1] = -self.Ks[1:]
438         return self.K
439
440     self.K=gen_Kmatrix_np(*self.Ks)
```

Figura 4.2-16 Arrays ( )

Para hacer que funcione un *"array ( )"* debe de ir acompañado de *"np"* (*np.array()*). Dentro de los "*arrays"* existen varias funciones que debimos de emplear para el cálculo de masas, eigenvalues o eigenvectors. Como por ejemplo la función *"matmul"* que utilizamos para realizar la multiplicación de matrices, o la función *"np.zeros"* que nos ayudó a crear una matriz llena de ceros para realizar las operaciones de la matriz identidad. Mientras que la función *"np.arange"* se utilizó para crear un "*np.array"* con valores espaciados de manera uniforme dentro de un rango especificado. Otra de las funciones que contiene y que sirvieron de mucho fue la función *"np.diag",* que se utilizó para crear las matrices dinámicas del sistema o la función *"np.transpose", que* se utilizó para realizar la transposición de una matriz. La transposición de una matriz implica intercambiar sus filas por columnas, esto lo hicimos para ordenar las matrices de los valores propios y sus vectores propios.

Usamos la función *"np.linalg.eig"* para calcular los valores propios y los vectores propios de una matriz cuadrada. Los valores propios (eigenvalues) son números que caracterizan la matriz, y los vectores propios (eigenvectors) son los vectores asociados a estos valores como se puede ver un ejemplo en la *Figura 4.2-17.*

```
442        self.A=np.matmul(self.MINV,self.K)
443
444
445        self.Eigenvalores, self.Eigenvectores=np.linalg.eig(self.A)
446
447        rank=np.argsort(self.Eigenvalores)
```

Figura 4.2-17 Funciones del Módulo NumPy

4.3. Función de aptitud

Se desarrolló una función objetivo, que permitiera optimizar el resultado lo máximo posible, respetando la fórmula asignada. Llamamos a todo el procedimiento realizado *"self.calculos ( )",* donde asignamos a una variable con el nombre de *"self.promedio=0",* para lo cual, en nuestra función utilizamos el bucle "*while*". (Ver *Figura 4.3-1*)

```
865    def funcion_objt(self):
866      self.calculos()
867      self.promedio=0
868      i=0
869      while i<niveles:
870        self.promedio=self.promedio + (self.RΣSmax[i]/Rel)/niveles
871        if max(self.RΣSmax)>Rel:
872          self.promedio=0
873        i=i+1
874      """self.promedio=self.distorsionmax/Rel/niveles
875      if max(self.RΣSmax)>Rel:
876        self.promedio=0"""
877
878      return (self.promedio)
879
```

Figura 4.3-1 Función objetivo

4.4. Selección de operadores genéticos

Para la selección de nuestros operadores genéticos, utilizamos el método de *"ruleta"*, que dice que la probabilidad de que un individuo sea seleccionado es proporcional a su *"fitness"* relativo o, de igual manera, utilizamos el método

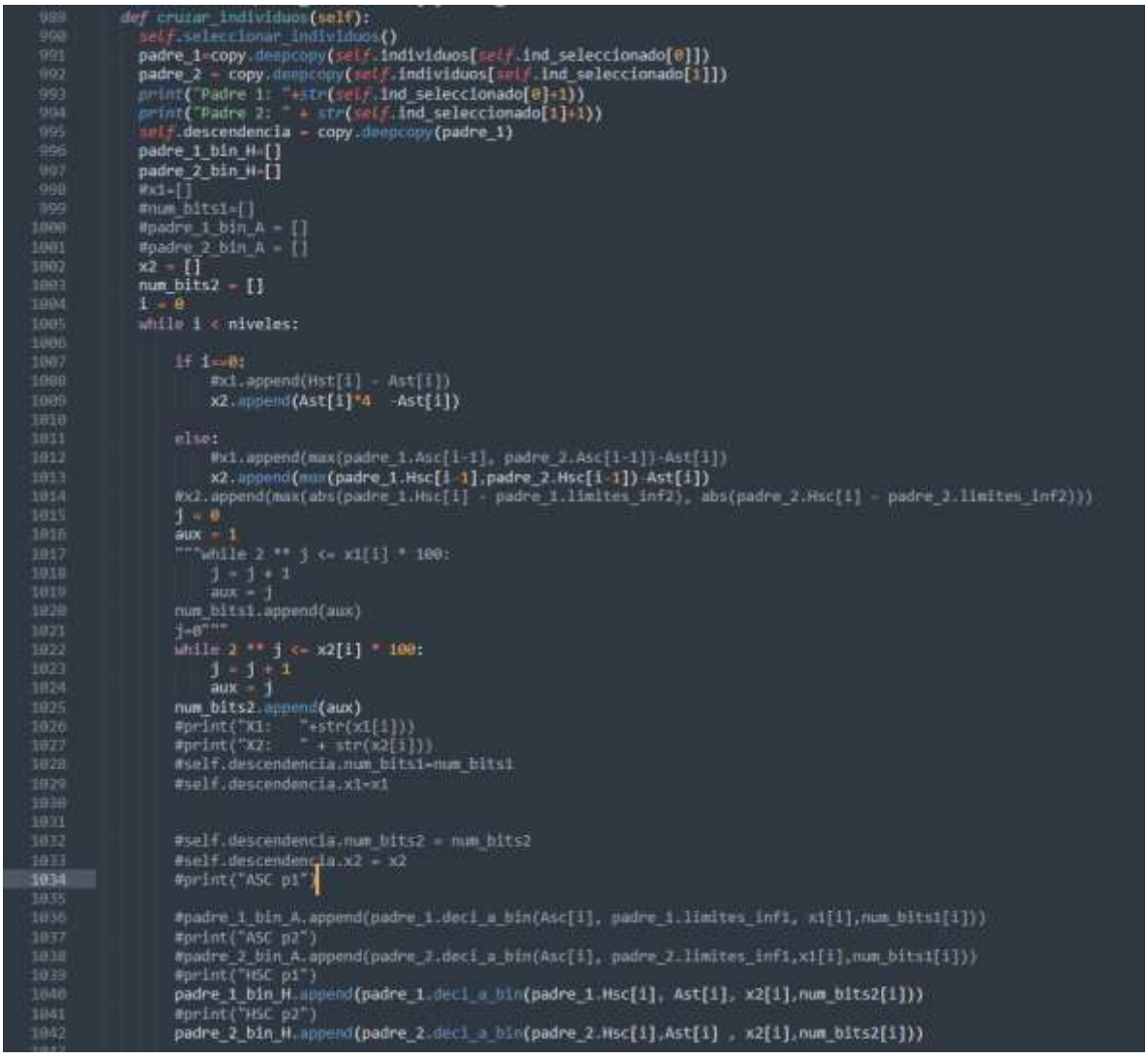
"tournament" que es como un torneo y que, generalmente, se refiere a una estructura de datos (secciones) para determinar el mejor y el peor. (Ver *Figura 4.4-1*)

```
def seleccionar_individuos(self):
    array_calificacion=np.repeat(None, self.n_ind)
    metodo_seleccion="ruleta"

    for i in np.arange(self.n_ind):
      array_calificacion[i]=copy.copy(self.individuos[i].funcion_objt())
    if metodo_seleccion == "ruleta":
      self.probabilidad_seleccion = array_calificacion / np.sum(array_calificacion)
      self.ind_seleccionado = np.random.choice(np.arange(self.n_ind), 2, True, list(self.probabilidad_seleccion))

    elif metodo_seleccion == "tournament":
      n=2
      self.ind_seleccionado = np.repeat(None, n)
      for i in np.arange(n):

            candidatos_a = np.random.choice(
              a=np.arange(self.n_ind),
              size=2,
              replace=false
            )
            candidatos_b = np.random.choice(
              a=np.arange(self.n_ind),
              size=2,
              replace=false
            )

            if abs(array_calificacion[candidatos_a[0]]-1 )<abs( array_calificacion[candidatos_a[1]]-1):
              ganador_a = candidatos_a[0]
            else:
              ganador_a = candidatos_a[1]

            if abs(array_calificacion[candidatos_b[0]]-1) < abs(array_calificacion[candidatos_b[1]]-1):
              ganador_b = candidatos_b[0]
            else:
              ganador_b = candidatos_b[1]


            if abs(array_calificacion[ganador_a]-1) < abs(array_calificacion[ganador_b]-1):
              ind_final = ganador_a
            else:
              ind_final = ganador_b
            self.ind_seleccionado[i] = ind_final
```

Figura 4.4-1 Selección de individuos

En Python, la cruza y la mutación son operaciones comunes utilizadas en algoritmos genéticos para optimizar soluciones. La cruza implica combinar la información genética de dos individuos para crear uno o más descendientes, mientras que la mutación implica realizar cambios aleatorios en la información genética de un individuo.

Para implementar la cruza, creamos una función que tome dos padres como entrada y produzca uno o más hijos combinando sus genes de alguna manera específica como se muestra en la F*igura 4.4-1.*



```
def cruzar_individuos(self):
    self.seleccionar_individuos()
    padre_1=copy.deepcopy(self.individuos[self.ind_seleccionado[0]])
    padre_2 = copy.deepcopy(self.individuos[self.ind_seleccionado[1]])
    print("Padre 1: "+str(self.ind_seleccionado[0]+1))
    print("Padre 2: " + str(self.ind_seleccionado[1]+1))
    self.descendencia = copy.deepcopy(padre_1)
    padre_1_bin_H=[]
    padre_2_bin_H=[]
    #x1=[]
    #num_bits1=[]
    #padre_1_bin_A = []
    #padre_2_bin_A = []
    x2 = []
    num_bits2 = []
    i = 0
    while i < niveles:

        if i==0:
            #x1.append(Hst[i] - Ast[i])
            x2.append(Ast[i]*4  -Ast[i])

        else:
            #x1.append(max(padre_1.Asc[i-1], padre_2.Asc[i-1])-Ast[i])
            x2.append(max(padre_1.Hsc[i-1],padre_2.Hsc[i-1])-Ast[i])
        #x2.append(max(abs(padre_1.Hsc[i] - padre_1.limites_inf2), abs(padre_2.Hsc[i] - padre_2.limites_inf2)))
        j = 0
        aux = 1
        """while 2 ** j <= x1[i] * 100:
            j = j + 1
            aux = j
        num_bits1.append(aux)
        j=0"""
        while 2 ** j <= x2[i] * 100:
            j = j + 1
            aux = j
        num_bits2.append(aux)
        #print("X1:   "+str(x1[i]))
        #print("X2:   " + str(x2[i]))
        #self.descendencia.num_bits1=num_bits1
        #self.descendencia.x1=x1


        #self.descendencia.num_bits2 = num_bits2
        #self.descendencia.x2 = x2
        #print("ASC p1")

        #padre_1_bin_A.append(padre_1.deci_a_bin(Asc[i], padre_1.limites_inf1, x1[i],num_bits1[i]))
        #print("ASC p2")
        #padre_2_bin_A.append(padre_2.deci_a_bin(Asc[i], padre_2.limites_inf1,x1[i],num_bits1[i]))
        #print("HSC p1")
        padre_1_bin_H.append(padre_1.deci_a_bin(padre_1.Hsc[i], Ast[i], x2[i],num_bits2[i]))
        #print("HSC p2")
        padre_2_bin_H.append(padre_2.deci_a_bin(padre_2.Hsc[i],Ast[i] , x2[i],num_bits2[i]))
```

Figura 4.4-2 Cruza

Para la mutación, creamos una función que tome un individuo como entrada y realice cambios aleatorios en sus genes. En la *Figura 4.4-2* se muestra la codificación.

En la *figura* 4.4-3, *"probabilidad_de_mutacion"* representa la probabilidad de que un gen sea mutado. La función "*mutante1*" se implementa de acuerdo con el problema que buscamos, en este caso a que solo una parte de los individuos muten con la función *np.radom.choice*.

Figura 4.4-3 Mutación

4.5. Manejo de restricciones

En este caso manejamos restricciones en el algoritmo genético: en la búsqueda de las secciones dadas por el código, no puede tener más de dos decimales y que la sección del nivel inferior sea mayor o igual a la del nivel superior, pero la sección de peralte de la columna mayor a la sección de la base de la trabe. Esto ocasiona que la búsqueda de una optimización al *100%* sea un poco complicada ya que estas restricciones impiden que el algoritmo logre optimizar al máximo.

De igual forma cuando se seleccione un individuo, éste debe cumplir con características específicas, como por ejemplo que todas las distorsiones de cada nivel sean menores o iguales a las de norma, si no es así, el código tuvo que castigarlos con una eficiencia de *0* haciendo que estos individuos no pasen a la siguiente generación.

También el algoritmo selecciona los mejores cinco sujetos de cada generación para que pasen a la siguiente sin ser modificados por la cruza o por la mutación.

CAPÍTULO 5: DEFINICIÓN GEOMÉTRICA Y CARGAS DE LOS MARCOS A OPTIMIZAR

Los valores que están establecidos para los tres ejemplos son; el módulo de elasticidad con un valor de $1739252713\ {}^{Kg}/_{m^2}$ y la gravedad con un valor ya conocido de $9.81\ {}^{m}/_{s^2}$. El módulo corresponde a $11000\sqrt{fc'}$, en donde $fc' = 250\ {}^{kg}/_{cm^2}$

5.1. Ejemplo 1: Marco de 5 niveles x 4 crujías

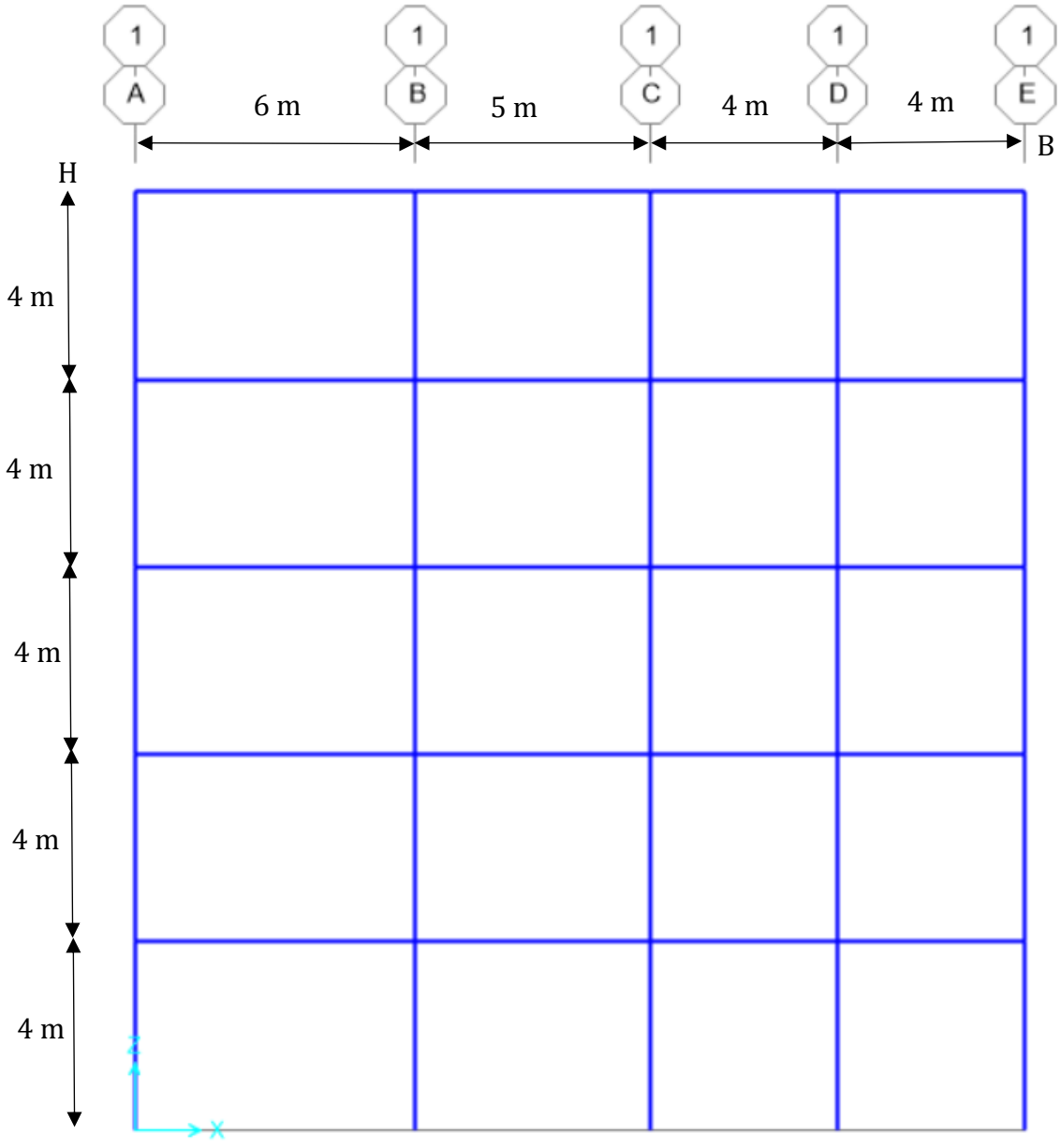


Figura 5.1-1 MARCO 5X4

El primer marco fue diseñado con las siguientes especificaciones para probarlo con el algoritmo, consta de cinco niveles con columnas y trabes de concreto, con una longitud de desplante de *19 m* y una altura de *16 m* como se muestra en la *Figura 5.1-1.* Mientras que los pesos de entrepiso por nivel tienen un valor de *52250 kg-m*, esta carga ya ha sido factorizada por *1.1* y su criterio de diseño se basa en las consideraciones sísmicas mencionadas en capítulos anteriores.

5.2. Ejemplo 2: Marco de 3 niveles x 7 crujías

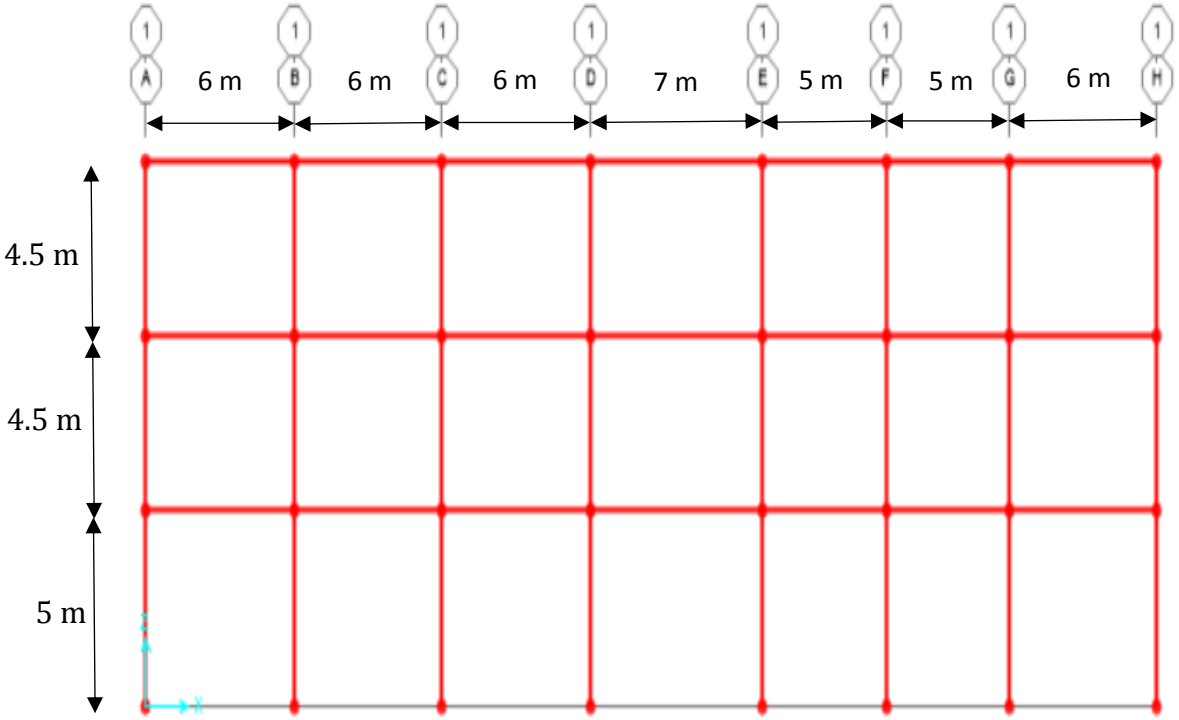


Figura 5.2-1 MARCO 3X7

El segundo marco fue diseñado con valores distintos, consta de tres niveles con columnas y trabes de concreto, teniendo una longitud de desplante de *41 m* y una altura de *14 m* tal y como se muestra en la *Figura 5.2-1*.

Los pesos de entrepiso, por nivel, tienen un valor de *112750 kg-m;* esta carga ya ha sido factorizada por *1.1*.

5.3. Ejemplo 3: Marco de 6 niveles x 4 crujías

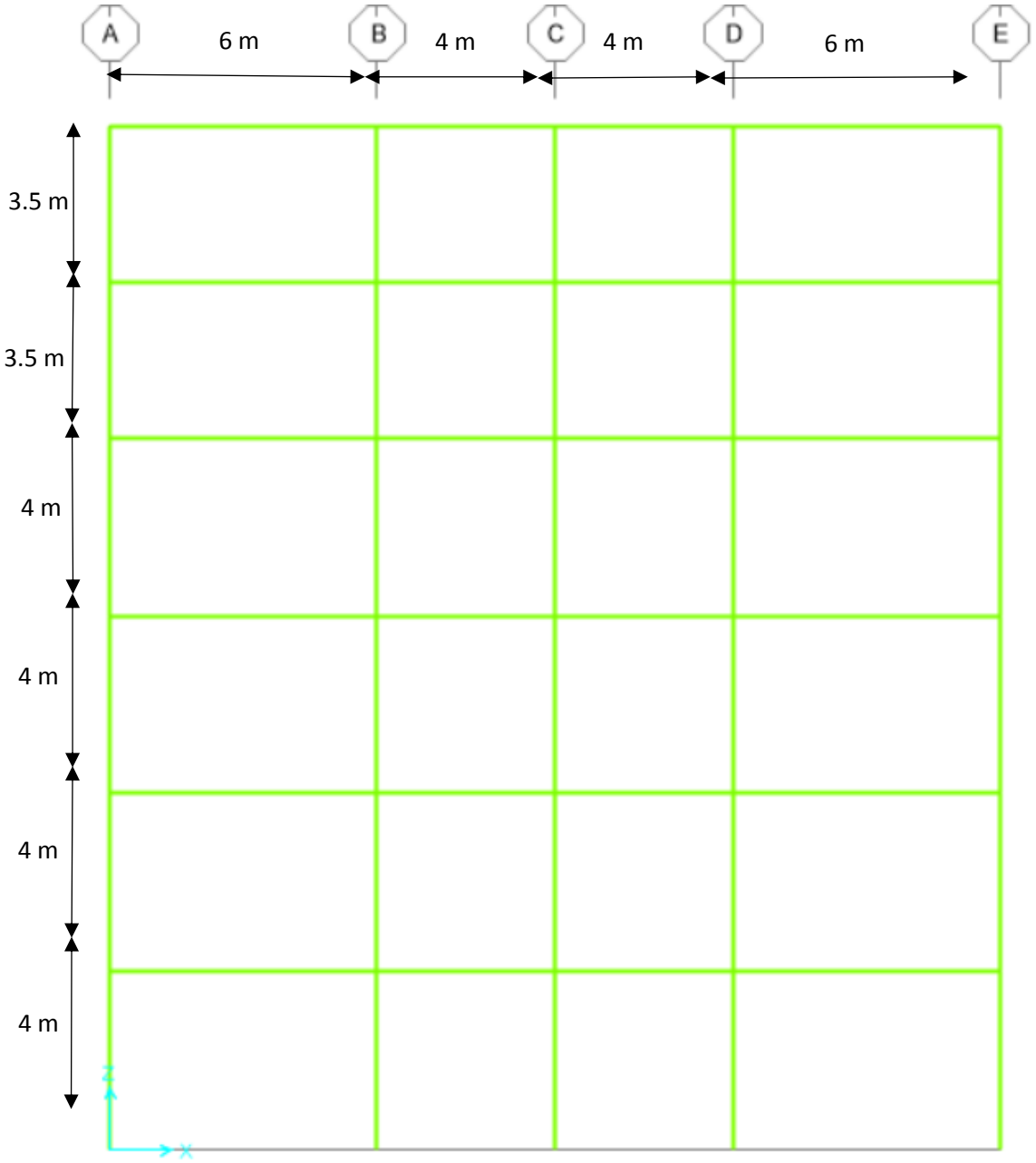


Figura 5.3-1 MARCO 6X4

El tercer marco fue diseñado con valores distintos, consta de seis niveles con columnas y trabes de concreto, teniendo una longitud de desplante de *20 m* y una altura de *23 m* tal y como se muestra en la F*igura 5.3-1*.

Mientras que los pesos de entrepiso por nivel tienen un valor de *55000 kg-m* esta carga ya ha sido factorizada por *1.1*.

Elegimos tres modelos con algunas características diferentes para comprobar que el algoritmo está funcionando bien y que es eficiente.

CAPÍTULO 6: APLICACIÓN DEL ALGORITMO GENÉTICO

Para diseñar por sismo es necesario revisar el comportamiento del edificio en las dos direcciones ortogonales al plano (X y Y), aquí se realizó el análisis de manera independiente en cada dirección. Los parámetros del algoritmo genético son: 100 individuos por generación, probabilidad de *crossover* de *0.05* y probabilidad de mutación de *0.07*. Se eligió un tamaño de población par para la selección y reproducción por pares, y así pasar el individuo elite a la siguiente generación, es decir, el mejor individuo por generación.

Cabe mencionar, que cada generación consta de 100 individuos, pero diferentes registros. Los criterios de optimización son dos: el índice de distorsión modal de entrepiso y los desplazamientos máximos. El índice de distorsión modal de entrepiso indica qué tan cerca están las distorsiones de entrepiso de las distorsiones límite, pero no trataremos de llevarlas a un mínimo, sino tratar de mantenerlas en el límite; si reducimos en demasía las distorsiones, tendríamos como resultados un sobrecosto de la construcción. Para el cálculo del índice de distorsión de entrepiso, se evalúa un índice de distorsión de entrepiso local por nivel.

Las fórmulas para calcularlo son: $R\Sigma Smax[i] <= Rel;$ donde $R\Sigma Smax[i]$ es la raíz cuadra de la sumatoria de las distorsiones máximas de cada modo y Rel es igual al valor de los límites establecidos por el reglamento; que se indica como “PASA” o “NO PASA”, $Rel < R\Sigma Smax[i]$. Los límites establecidos por el reglamento; son de *0.006* o *0.012* dependiendo del usuario del algoritmo, por lo que este valor aparece en el denominador de las fórmulas anteriores. Si la distorsión va decreciendo se aleja del límite y su calificación también, por lo que una calificación de *0* es cuando nos encontramos en valor superior los límites *0.006* o *0.012*, y una calificación de *0* implica que la estructura es sumamente rígida. Buscando privilegiar a los individuos que presentan distorsiones dentro de los límites del reglamento, para seleccionar el individuo utilizamos una expresión $self.promedio = self.promedio + (self.R\Sigma Smax[i]/Rel)/niveles$.

CAPÍTULO 7: ANÁLISIS DE RESULTADOS

Se realizaron tres ejemplos, mostraron una gran diversidad en las secuencias de proyectos con valores de los beneficios muy cerrados entre unos y otros.

En resumen, los tres marcos tuvieron una buena eficiencia al buscar el mejor individuo, esto claro, con eficiencias no tan cercanas al *1*, sino que se quedan algo alejadas, ya que el algoritmo tiene restricciones ya mencionadas anteriormente que impiden acercarse más, tal y como dice la función objetivo.

7.1. Secciones

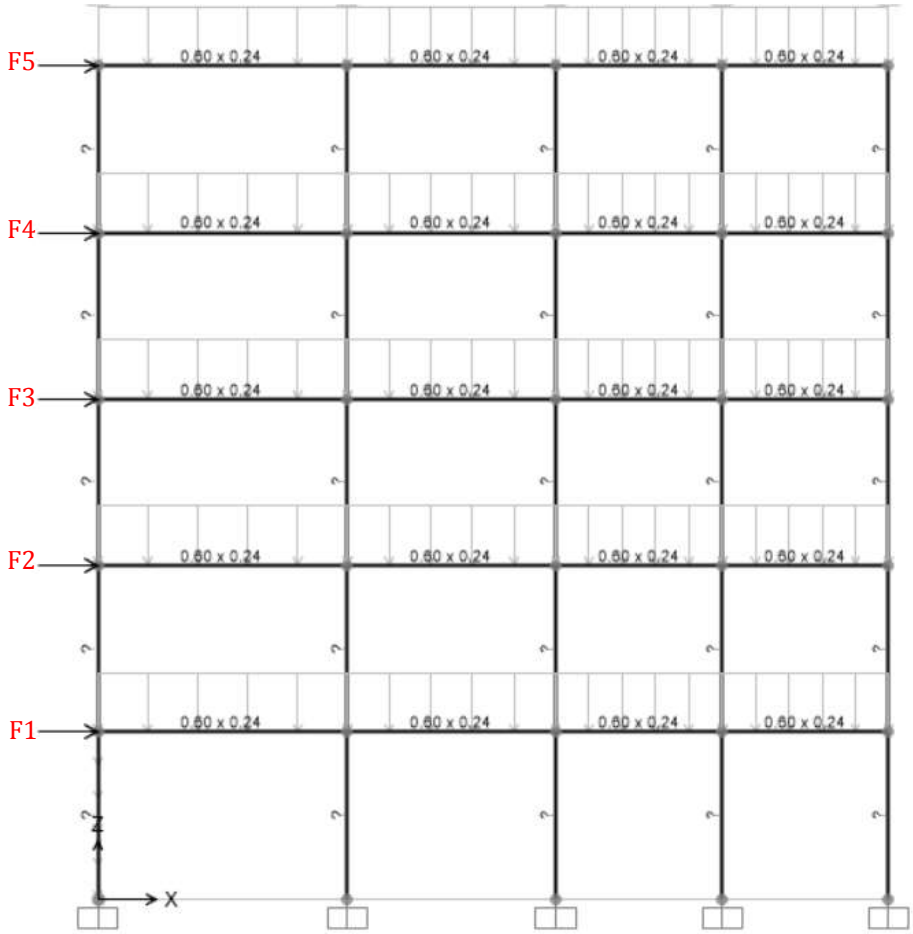


Figura 7.1-1 Marco 5x4: Secciones

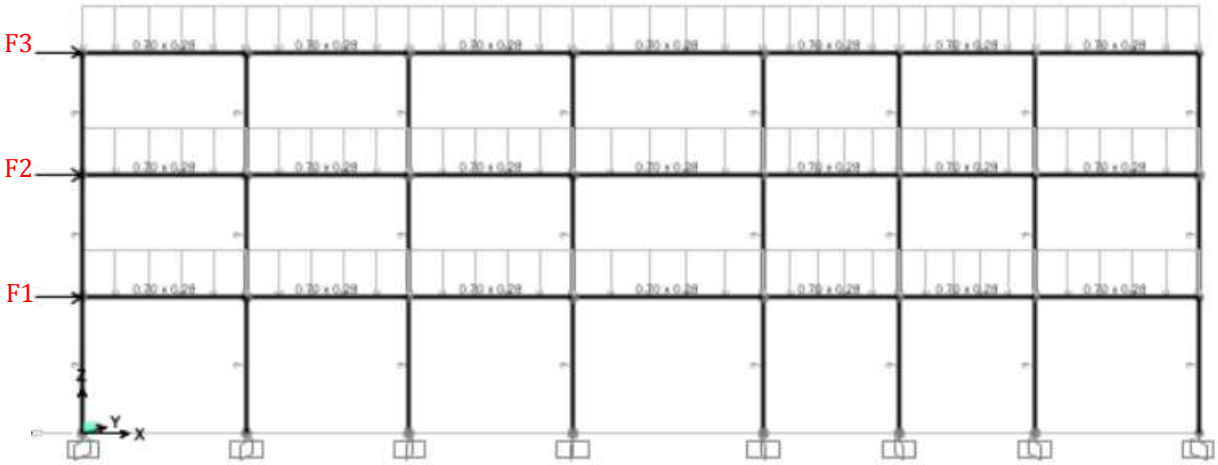


Figura 7.1-2 Marco 3x7: Secciones

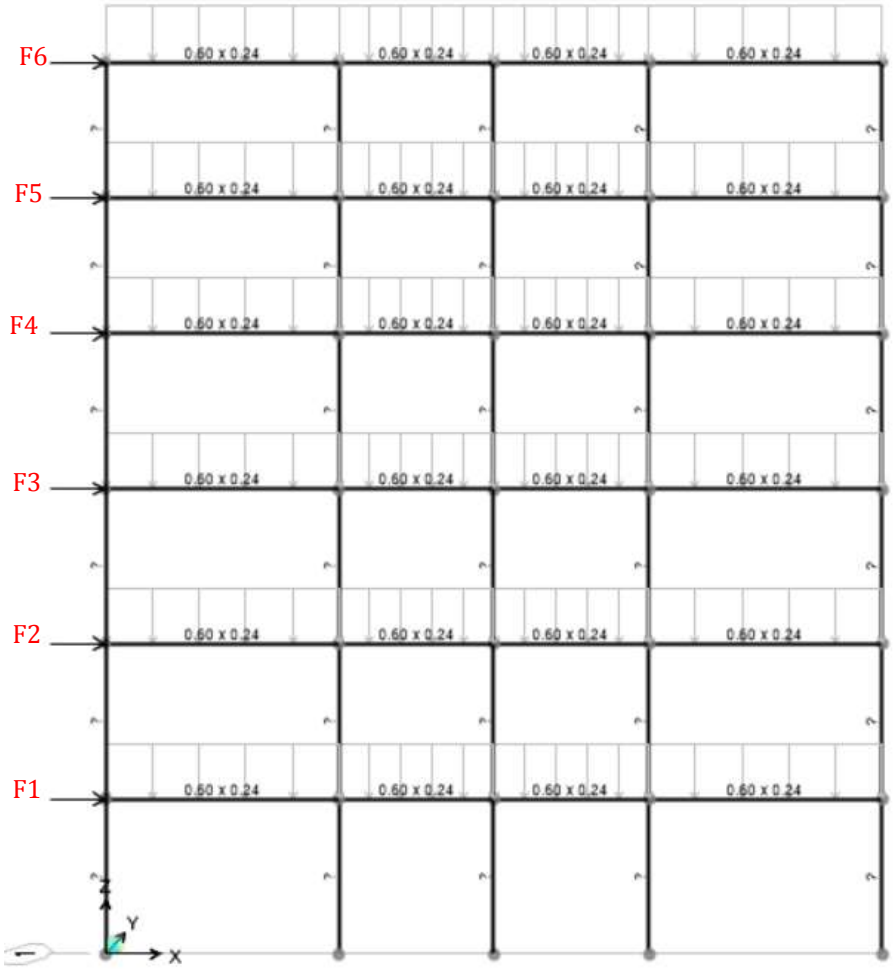


Figura 7.1-3 Marco 6x4: Secciones

En las *Figuras 7.1-1, 7.1-2 y 7.1-3* se puede observar las secciones de las trabes que fueron calculadas desde un inicio con la siguiente fórmula para el cálculo $PERALTE\ DE\ TRABE_i = \frac{CLARO\ MAYOR}{10}$ y para el $ANCHO\ DE\ TRABE_I = \frac{PERALTE\ DE\ TRABE_i}{2.5}$, así obteniendo las dimensiones para las trabes.

Para la obtención de las secciones de columna utilizamos fórmulas similares que tiene un límite inferior y uno superior, como se muestra en las siguientes fórmulas, el $ANCHO\ DE\ COLUMNA_I = ANCHO\ DE\ TRABE_i + 0.10$ y el límite inferior del $PERALTE\ DE\ COLUMNA_i = 0.20\ o\ 0.25$. Estos valores se define dependiendo el factor de comportamiento sísmico *(Q)* que se elija, para *Q = 1 o 2* se utiliza el *0.20* y para *Q = 3 o 4* se utiliza el *0.25*, esto lo pusimos como una restricción para controlar los valores de las secciones que arrojó el algoritmo, mientras que se propuso como límite superior para el $PERALTE\ DE\ COLUMNA_i = 4 * ANCHO\ DE\ TRABE_i$, así obteniendo nuestras secciones para los tres ejemplos.

7.2. Ejemplo 1: Marco 5x4

```
Numero de individuos de la población: 100
Número de generaciones: 50
Escribe el factor de carga deseado: 1.1
Crea tu propio marco, cuya gravedad(m/s^2) y modulo de elasticidad(kg/m2) son:
MODULO DE ELASTICIDAD (kg/m2): 1739252713
GRAVEDAD (m/s^2):  9.81
¿Los muros estan ligados?:
    1)SI    2)NO
2
¿Cúantos niveles tiene tu edificio? 5
¿Cúantos crujias tiene tu edificio? 4
NUMERO DE COLUMNAS DE ENTEREPISO POR NIVEL:  5
Escribe la ALTURA 1 en m: 4
Escribe la ALTURA 2 en m: 4
Escribe la ALTURA 3 en m: 4
Escribe la ALTURA 4 en m: 4
Escribe la ALTURA 5 en m: 4
ALTURA 1 (m): 4.0
ALTURA 2 (m): 4.0
ALTURA 3 (m): 4.0
ALTURA 4 (m): 4.0
ALTURA 5 (m): 4.0
Escribe la BASE 1 en m: 6
Escribe la BASE 2 en m: 5
Escribe la BASE 3 en m: 4
Escribe la BASE 4 en m: 4
BASE 1 (m): 6.0
BASE 2 (m): 5.0
BASE 3 (m): 4.0
BASE 4 (m): 4.0
```

Figura 7.2-1 Especificación MARCO 5X4

En la *Figura 7.2-1* se presentan los datos ya mencionados anteriormente, pero ahora dentro del algoritmo genético, así como sus dimensiones de las secciones, el número de individuos y su número de generaciones, como también nos indica el factor de carga empleado y que los muros no están ligados, dándonos un estado límite de distorsión de *0.012*.

```
¿Quieres utilizar los datos basadosen NORMAS TECNICAS COMPLEMENTARIAS PARA DISEÑO POR SISMO (PUEBLA)?

    1)SI    2)NO

-Selecciona una opcion: 1
Tipo de terreno
1.  I (Terreno firme)
0.18 (Coeficiente Sismico)
2.  II (Terreno intermedio)
0.32 (Coeficiente Sismico)
3.  III (Terreno blando)
0.4 (Coeficiente Sismico)
4. SALIR
Elije el TIPO DE TERRENO: 2
II (Terreno intermedio)
2.- 0.32
2.1-  1
2.3-  2
2.4-  3
2.5-  4
Elije el Factor de comportamiento sismico (Q): 2
2.0
```

Figura 7.2-2 Datos de la NTC en Python

En la *Figura 7.2-2* se puede apreciar que nos basamos en las Normas Técnicas Complementarias para Diseño por Sismo (Puebla, 2017). Tomando como el coeficiente sísmico *0.32* y un factor de comportamiento sísmico de *2*; en este ejemplo se optó por elegir el grupo B.

```
MEJOR INDIVIDUO
----------------
EFICIENCIA: 0.860614641576986
INDIVIDUO: 1
SECCIÓN PERALTE TRABE
NIVEL 1: 0.6
NIVEL 2: 0.6
NIVEL 3: 0.6
NIVEL 4: 0.6
NIVEL 5: 0.6
SECCIÓN ANCHO TRABE
NIVEL 1: 0.24
NIVEL 2: 0.24
NIVEL 3: 0.24
NIVEL 4: 0.24
NIVEL 5: 0.24
SECCIÓN ANCHO COLUMNA
NIVEL 1: 0.34
NIVEL 2: 0.34
NIVEL 3: 0.34
NIVEL 4: 0.34
NIVEL 5: 0.34
SECCIÓN PERALTE COLUMNA
NIVEL 1: 0.36
NIVEL 2: 0.36
NIVEL 3: 0.34
NIVEL 4: 0.29
NIVEL 5: 0.25
```

Figura 7.2-3 Secciones del MARCO 5X4

En la *Figura 7.2-3* se nos muestra el mejor individuo con la mejor eficiencia, también nos muestra las dimensiones que el algoritmo generó, cruzó y mutó para que este individuo fuera el mejor. En este caso, como se podrá ver, el algoritmo nos dio una eficiencia de *0.86061* dado que el algoritmo cumple con algunas restricciones.

7.2.1. Pesos de Entrepiso

```
PESO DE ENTREPISO 1 (Kg-m): 52250.00000000001
PESO DE ENTREPISO 2 (Kg-m): 52250.00000000001
PESO DE ENTREPISO 3 (Kg-m): 52250.00000000001
PESO DE ENTREPISO 4 (Kg-m): 52250.00000000001
PESO DE ENTREPISO 5 (Kg-m): 52250.00000000001
SUMATORIA DE PESO DE ENTREPISO [Kg]: 261250.00000000003
```

Figura 7.2.1-1 Carga Uniforme Sobre Trabe

Nuestros pesos de entre piso (carga uniforme sobre trabe) en este marco nos dieron como resultado de la operación de $((2500 \cdot \sum Base) \cdot factor\ de\ carga)$. Como se aprecia en la *Figura 7.1.1-1*, el valor del peso de entrepiso es igual en todos los niveles, teniendo como valor 52250 $kg \cdot m$ y un sumatoria total de 261250 kg.

7.2.2. Rigidez (Wilbur) (*Ks*)

Los datos que se presentan a continuación son las rigideces *(Ks)* por nivel, calculadas con las expresiones de Wilbur, arrojándonos los siguientes valores:

- *Ks 1:* 1777190.0673 $^{kg}/_{m}$
- *Ks 2:* 1503612.7490 $^{kg}/_{m}$
- *Ks 3:* 1323664.1411 $^{kg}/_{m}$
- *Ks 4:* 915524.5830 $^{kg}/_{m}$
- *Ks 5*: 628902.32063 $^{kg}/_{m}$

7.2.3. Fuerzas (*F*) (Análisis Estático)

Las fuerzas horizontales calculadas durante el proceso del algoritmo resultaron tener los siguientes valores por nivel:

- *F 1:* 3065 kg
- *F 2:* 6131 kg
- *F 3:* 9196 kg
- *F 4:* 12261 kg
- *F 5*: 15327 kg

7.2.4. Desplazamiento Total (*St*)

Los desplazamientos *(St)* por análisis estático, son el resultado del equilibrio entre las fuerzas aplicadas y las reacciones en la estructura, los valores arrojados por el mejor individuo son:

- *St 1:* 0.02587 m
- *St 2:* 0.05441 m
- *St 3:* 0.08220 m
- *St 4:* 0.11233 m
- *St 5:* 0.13670 m

7.2.5. Masas (*Ma*)

Las masas (Ma) fueron el resultado de la operación $\frac{Peso\ entrepiso}{gravedad}$ dándonos como resultado una masa de 5326 kg para todos los niveles.

7.2.6. Eigenvalores y Eigenvectores

Los valores corresponden a los eigenvalores, que dieron como resultado una matriz de 5x1, con los siguientes valores:

$$\begin{bmatrix} 924.5069 \\ 571.6895 \\ 326.3628 \\ 131.7956 \\ 20.8998 \end{bmatrix}$$

Mientras que el resultado del cálculo de los eigenvectores nos dio como resultado una matriz de *5x5* con los siguientes valores:

$$\begin{bmatrix} -0.6229 & -0.5725 & 0.4186 & -0.3047 & 0.1261 \\ 0.6808 & -0.0898 & 0.4295 & -0.5226 & 0.2660 \\ -0.3708 & 0.6651 & -0.1222 & -0.4930 & 0.4024 \\ 0.1032 & -0.4555 & -0.6878 & -0.0721 & 0.5508 \\ -0.0151 & 0.1185 & 0.3899 & 0.6210 & 0.6692 \end{bmatrix}$$

7.2.7. Periodos (*T*)

Los periodos *(T)* obtenidos mediante las operaciones realizadas con el algoritmo para el individuo con la mejor eficiencia de todas las generaciones son los siguientes:

- *T 1:* 1.37438 *s*
- *T 2:* 0.54730 *s*
- *T 3:* 0.34779 *s*
- *T 4:* 0.26278 *s*
- *T 5:* 0.20664 *s*

7.2.8. Distorsiones máximas (*Smax*)

Las distorsiones máximas obtenidos mediante las operaciones realizadas con el algoritmo fueron los siguientes valores todos pasando por la distorsión permisible:

- *Smax 1:* 0.00966
- *Smax 2:* 0.01063
- *Smax 3:* 0.01034
- *Smax 4:* 0.01141
- *Smax 5:* 0.009579

7.2.9. Gráficas

A continuación, en la *Figura 7.2.9-1* se muestra, por medio de cuatro gráficas, el comportamiento de la evolución que obtuvimos conforme avanzaron las generaciones, optimizando su eficiencia.

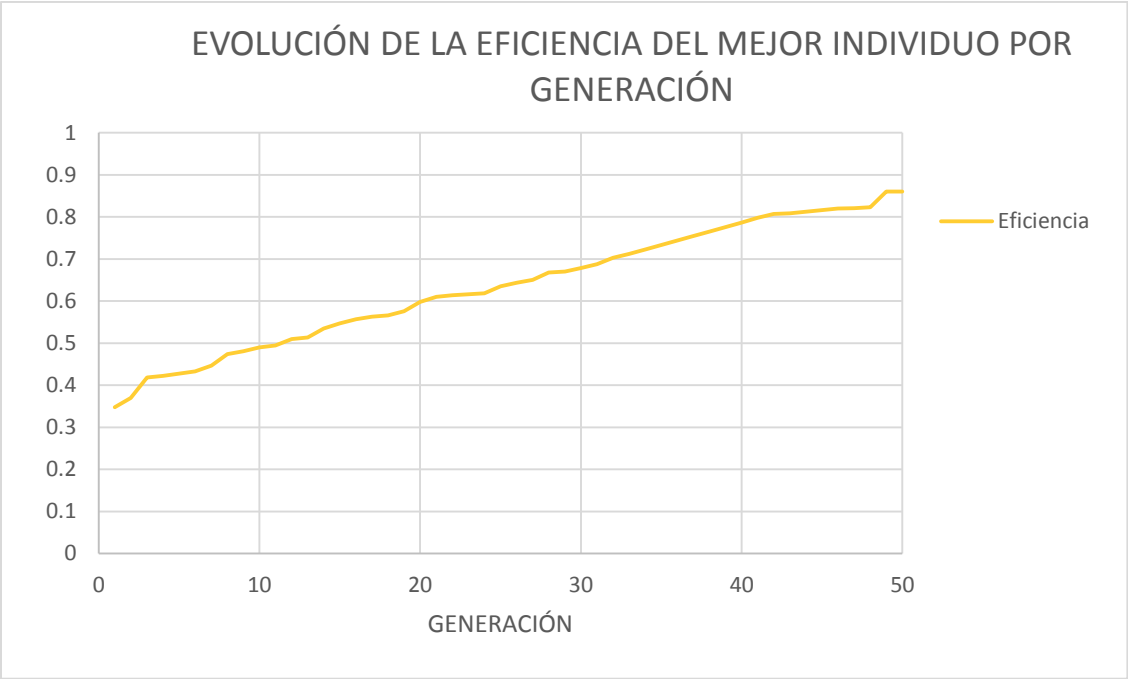


Figura 7.2.9-1 Eficiencia Marco 5x4

En la *Figura 7.2.9-2* se puede apreciar la evolución de las distorsiones máximas tomando en cuenta al mejor individuo de cada generación, se aprecia como varía en algunos puntos siendo más grandes o más pequeñas, haciendo que no se tenga una mayor eficiencia, mostrándonos que, para el mejor individuo con mejores distorsiones, sus distorsiones por cada nivel son más constantes.

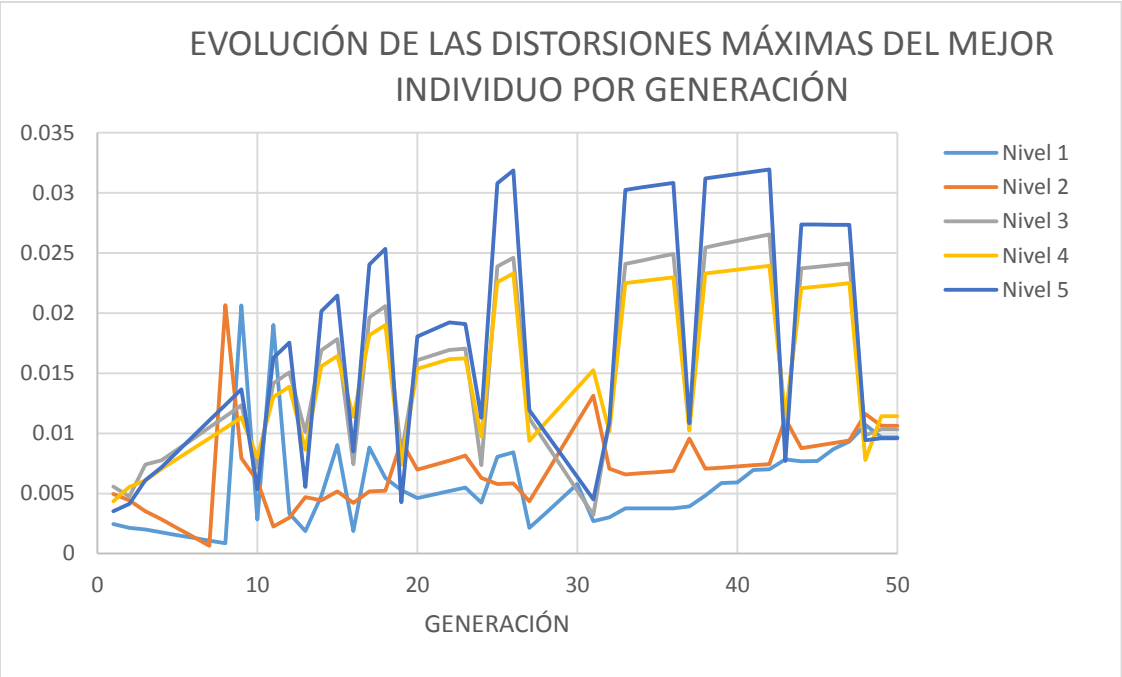


Figura 7.2.9-2 Distorsiones máximas del Marco 5x42

En la *Figura 7.2.9-3* se observa, de igual manera, las gráficas se disparan en algunos puntos y en otros se estabilizan siendo al final, en los últimos sujetos, cuando se logra una buena consistencia.

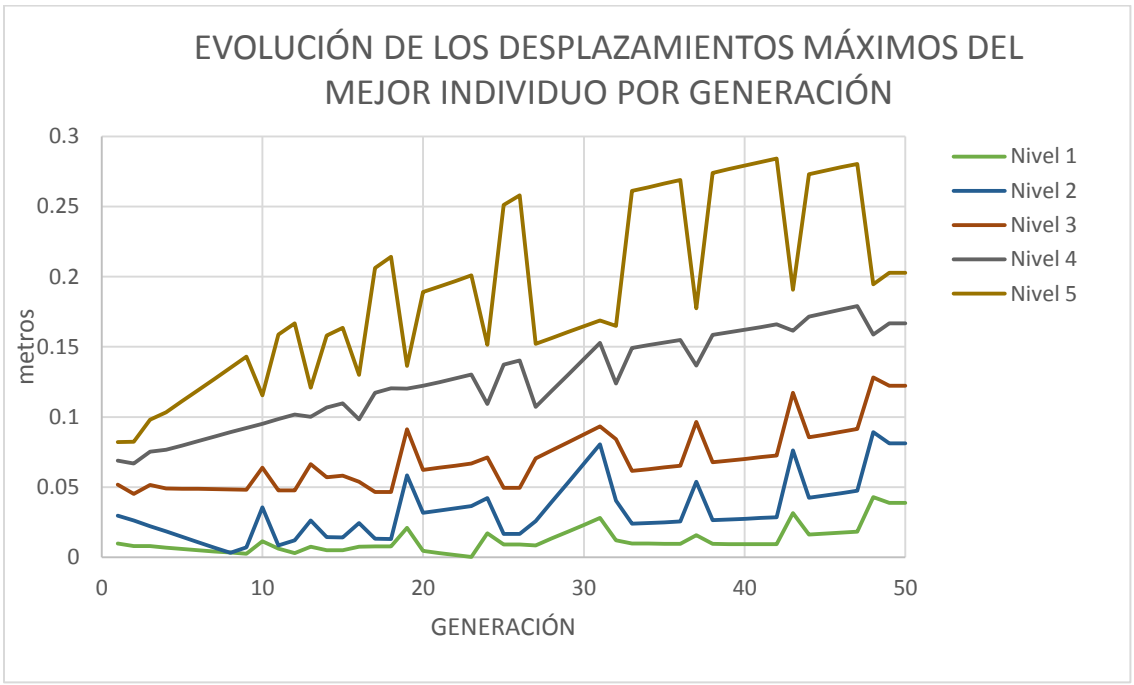


Figura 7.2.9-3 Desplazamientos máximos del Marco 5x4

En la *Figura 7.2.9-4* la gráfica se dispara en algunos puntos y en otros se estabilizan siendo al final, en los últimos sujetos cuando se logra un periodo adecuado.

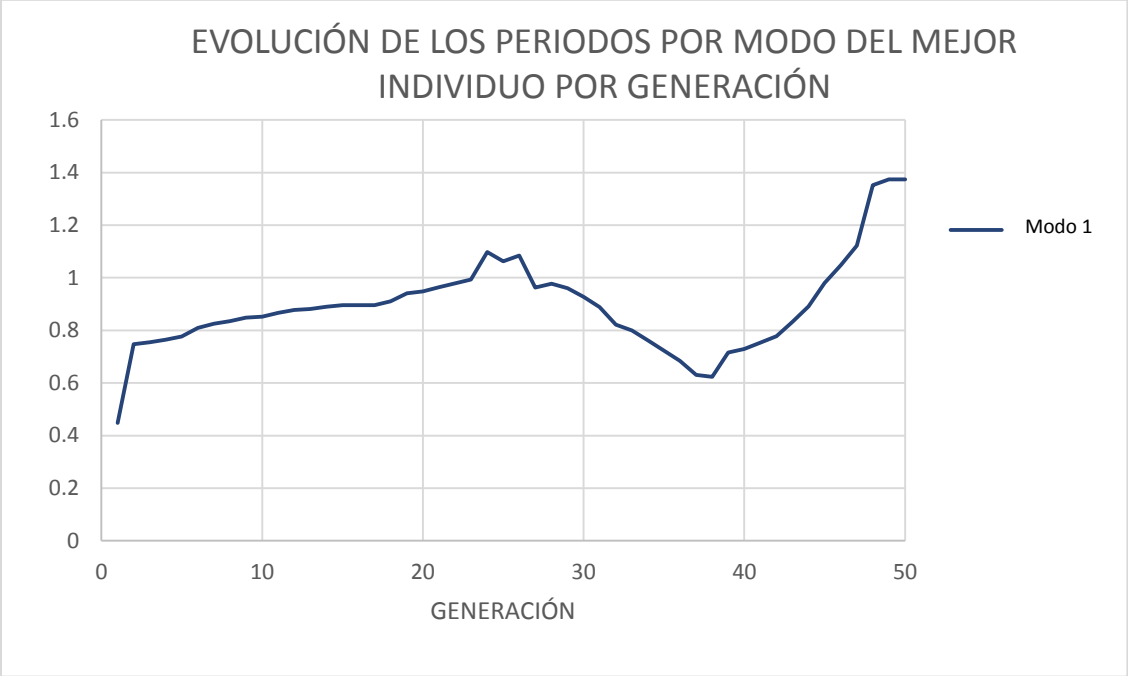


Figura 7.2.9-4 Periodos del Marco 5x4

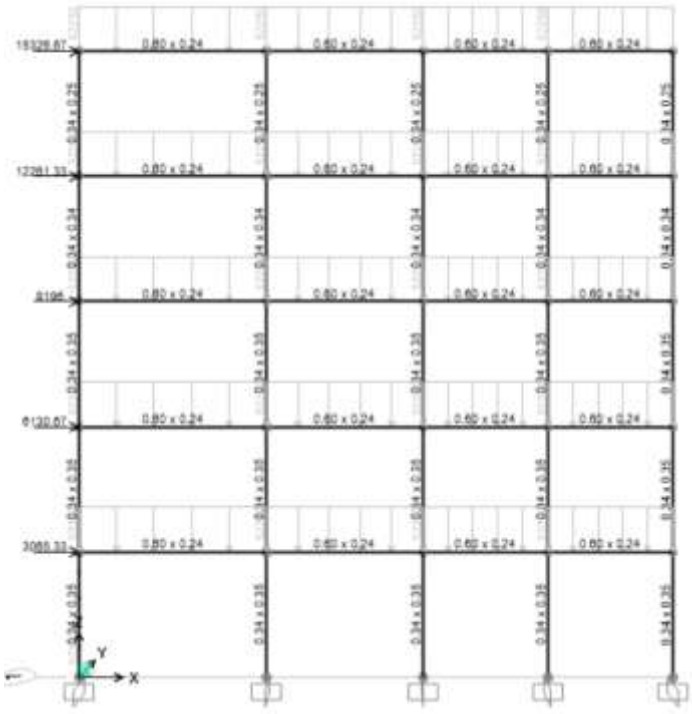


Figura 7.2-10 Marco 5x4 Mejor individuo

En la *Figura 7.2-10* nos muestra como se ve el individuo con mejor eficiencia ya con las secciones calculadas y sus fuerzas sísmicas.

7.3. Ejemplo 2: Marco 3x7

```
Numero de individuos de la población: 100
Número de generaciones: 50
Escribe el factor de carga deseado: 1.1
Crea tu propio marco, cuya gravedad(m/s^2) y modulo de elasticidad(kg/m2) son:
MODULO DE ELASTICIDAD (kg/m2): 1739252713
GRAVEDAD (m/s^2):  9.81
¿Los muros estan ligados?:
    1)SI    2)NO
1
¿Cúantos niveles tiene tu edificio? 3
¿Cúantos crujias tiene tu edificio? 7
NUMERO DE COLUMNAS DE ENTEREPISO POR NIVEL:  8
Escribe la ALTURA 1 en m: 5
Escribe la ALTURA 2 en m: 4.5
Escribe la ALTURA 3 en m: 4.5
ALTURA 1 (m): 5.0
ALTURA 2 (m): 4.5
ALTURA 3 (m): 4.5
Escribe la BASE 1 en m: 6
Escribe la BASE 2 en m: 6
Escribe la BASE 3 en m: 6
Escribe la BASE 4 en m: 7
Escribe la BASE 5 en m: 5
Escribe la BASE 6 en m: 5
Escribe la BASE 7 en m: 6
BASE 1 (m): 6.0
BASE 2 (m): 6.0
BASE 3 (m): 6.0
BASE 4 (m): 7.0
BASE 5 (m): 5.0
BASE 6 (m): 5.0
BASE 7 (m): 6.0
```

Figura 7.3-1 Especificación MARCO 5X4

En la *Figura 7.3-1* se muestran los datos mencionados, junto con sus dimensiones de las secciones, el número de individuos y sus generaciones y los factores de carga utilizados, y los muros sin unir entre sí muestran el estado límite de distorsión de *0.12*.

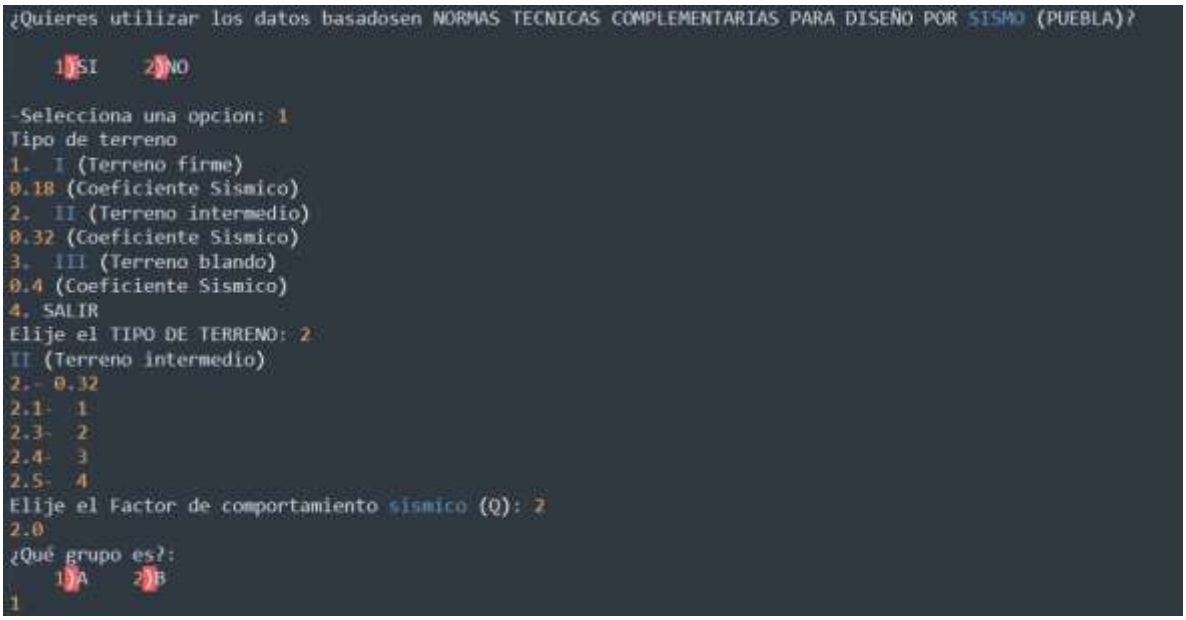


Figura 7.3-2 Datos de la NTC en Python

En la *Figura 7.3-2* se empleó un coeficiente sísmico de *0.32* y un factor de comportamiento sísmico de *2*; en este ejemplo se optó por elegir el grupo A.

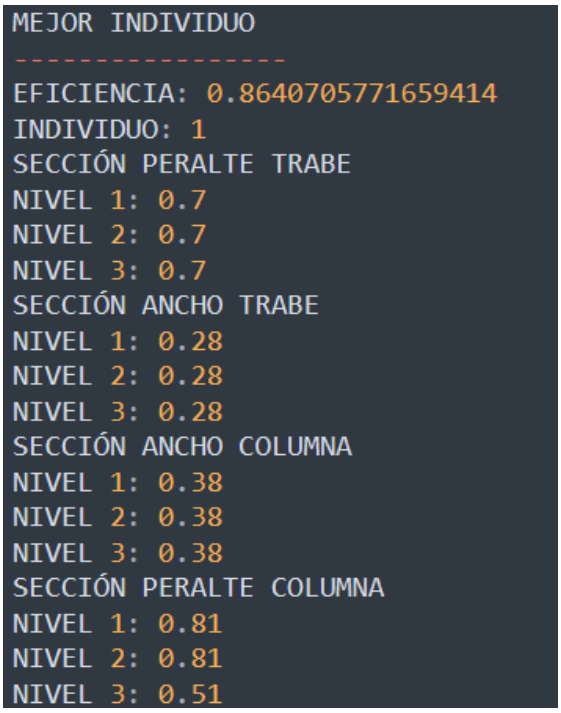


Figura 7.3-3 Secciones del MARCO 3X7

En la *Figura 7.3-3* nos muestra el mejor individuo con la mejor eficiencia, también nos muestra las dimensiones que el algoritmo generó, cruzó y mutó para que éste fuera el mejor. La mayor eficiencia fue de *0.86407*.

7.3.1. Pesos de Entrepiso

```
PESO DE ENTREPISO 1 (Kg-m): 112750.00000000001
PESO DE ENTREPISO 2 (Kg-m): 112750.00000000001
PESO DE ENTREPISO 3 (Kg-m): 112750.00000000001
SUMATORIA DE PESO DE ENTREPISO [Kg]: 338250.00000000006
```

Figura 7.3.1-1 Carga Uniforme Sobre Trabe

Nuestros pesos de entre piso (carga uniforme sobre trabe), como se aprecia en la *Figura 7.3.1-1*, el valor del peso de entrepiso es igual en todos los niveles teniendo como valor 112750 $kg \cdot m$ y un sumatoria total de 338250 kg.

7.3.2. Rigidez (Wilbur) (*Ks*)

Los siguientes datos son las rigideces *(Ks)* calculadas con las expresiones de Wilbur, arrojándonos los siguientes valores:

- *Ks 1:* 7375394.8257 $^{kg}/_{m}$
- *Ks 2:* 4243575.0259 $^{kg}/_{m}$
- *Ks 3:* 2685396.7612 $^{kg}/_{m}$

7.3.3. Fuerzas (*F*) (Análisis Estático)

Las fuerzas horizontales calculadas durante el proceso del algoritmo son los siguientes valores:

- *F 1:* 15666 kg
- *F 2:* 29766 kg
- *F 3:* 43865kg

7.3.4. Desplazamiento Total (*St*)

Los desplazamientos *(St)* por análisis estático son el resultado del equilibrio entre las fuerzas aplicadas y las reacciones en la estructura, los valores arrojados por el mejor individuo son:

- *St 1:* 0.0121 m
- *St 2:* 0.0294 m
- *St 3:* 0.0457 m

7.3.5. Masas (*Ma*)

Las masas (Ma) fueron el resultado de la operación $\frac{Peso\ entrepiso}{gravedad}$ dándonos como resultado una masa de 11493 kg para todos los niveles.

7.3.6. Eigenvalores y Eigenvectores

Los valores que se obtuvieron corresponden a los eigenvalores, que dieron como resultado una matriz de 3x1, con los siguientes valores:

$$\begin{bmatrix} 1243.3818 \\ 518.1309 \\ 85.9288 \end{bmatrix}$$

Mientras que el resultado del cálculo de los eigenvectores, nos dio como resultado una matriz de 3*x3* con los siguientes valores:

$$\begin{bmatrix} 0.8398 & 0.5010 & 0.2086 \\ -0.5287 & 0.6687 & 0.5226 \\ 0.1223 & -0.5492 & 0.8266 \end{bmatrix}$$

7.3.7. Periodos (*T*)

Los periodos *(T)* obtenidos mediante las operaciones realizadas con el algoritmo para el individuo con la mejor eficiencia de todas las generaciones son los siguientes:

- *T 1:* 0.67781 s
- *T 2:* 0.27603 s
- *T 3:* 0.17818 s

7.3.8. Distorsiones máximas (*Smax*)

Las distorsiones máximas obtenidos mediante las operaciones realizadas con el algoritmo fueron los siguientes valores todos pasando por la distorsión permisible:

- *Smax 1:* 0.00361
- *Smax 2:* 0.00597
- *Smax* 3: 0.00596

7.3.9. Gráficas

A continuación, se muestra por medio de cuatro gráficas el comportamiento de la evolución que obtuvimos conforme avanzaron las generaciones. En la *Figura 7.3.9-1* se observa cómo al paso de cada generación el marco fue optimizando su eficiencia.

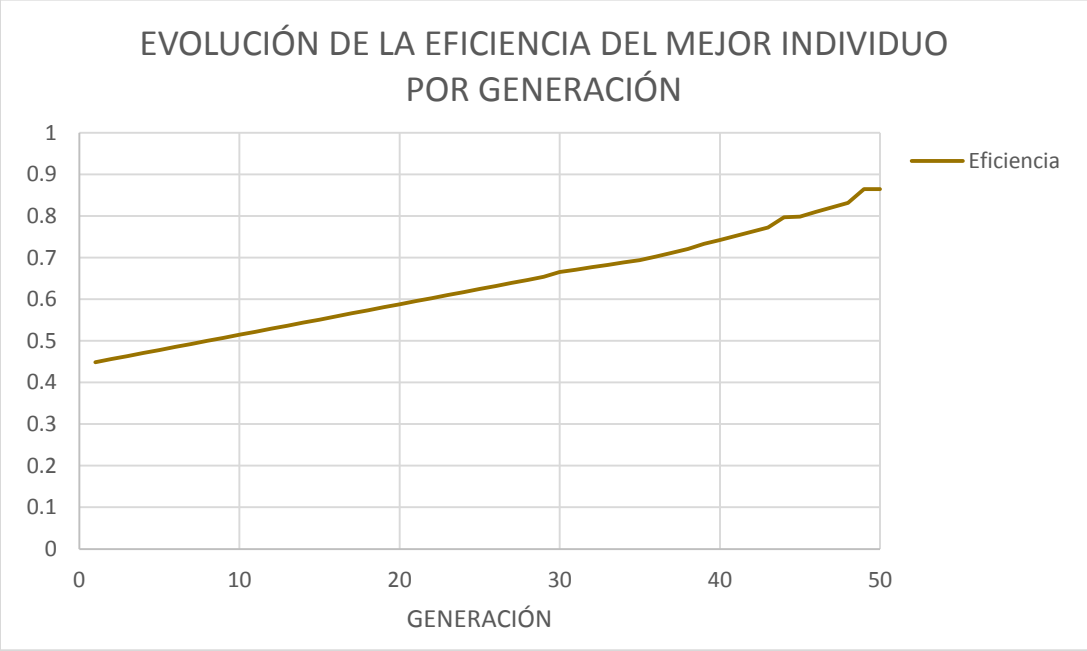


Figura 7.3.9-1 Eficiencia del Marco 3x7

En la *Figura 7.3.9-2* se muestra la evolución de las distorsiones máximas tomando en cuenta al mejor individuo de cada generación, se aprecia como varía en algunos puntos siendo más grandes o más pequeñas, haciendo que no se tenga una mayor eficiencia, mostrándonos que el mejor individuo con mejores distorsiones, sus distorsiones por cada nivel son más constantes.

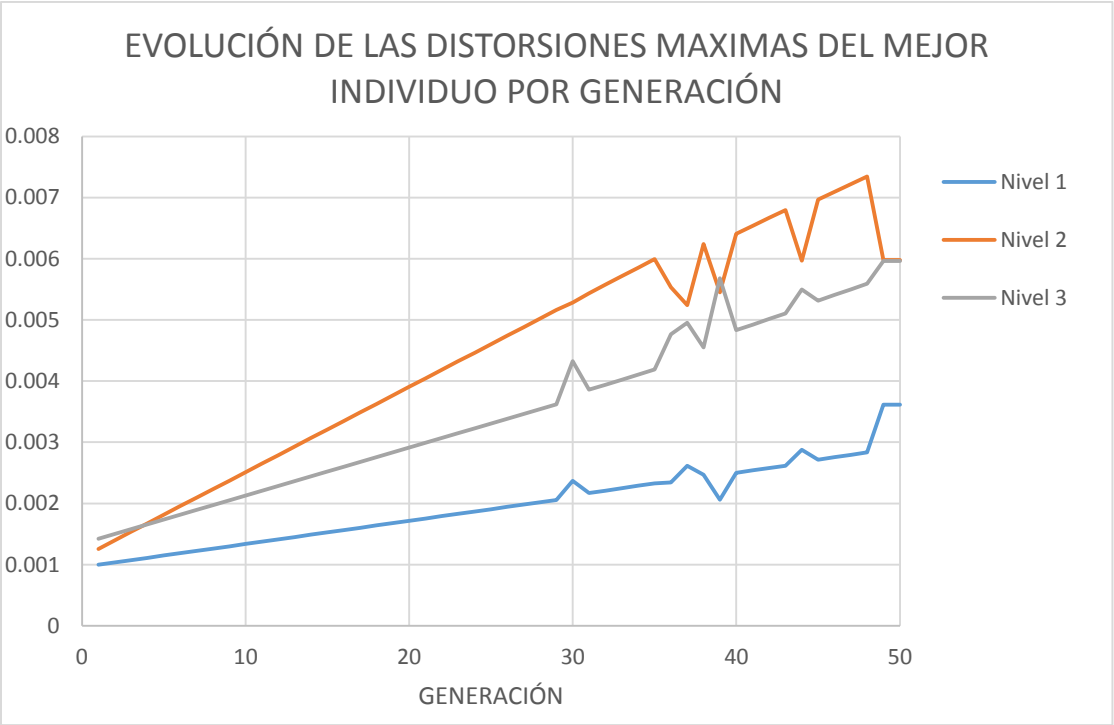


Figura 7.3.9-2 Distorsiones máximas del Marco 3x7

En la *Figura 7.3.9-3* se observa que al ser un marco con menor número de niveles la gráfica de los desplazamientos logra ser más constante desde un inicio, al llegar a la generación *30* tiene un incremento en sus desplazamientos en ciertos puntos.

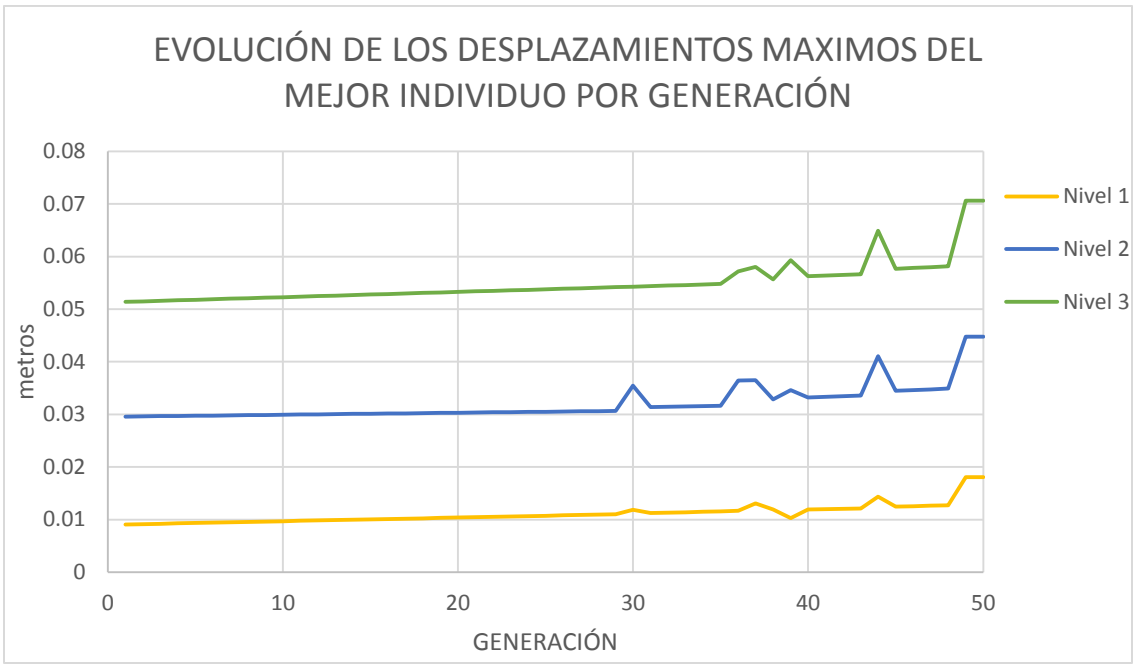


Figura 7.3.9-3 Desplazamientos máximos del Marco 3x7

En la *Figura 7.3.9-4* la gráfica, de igual modo, *al* tener un menor número de niveles los valores son más constantes.

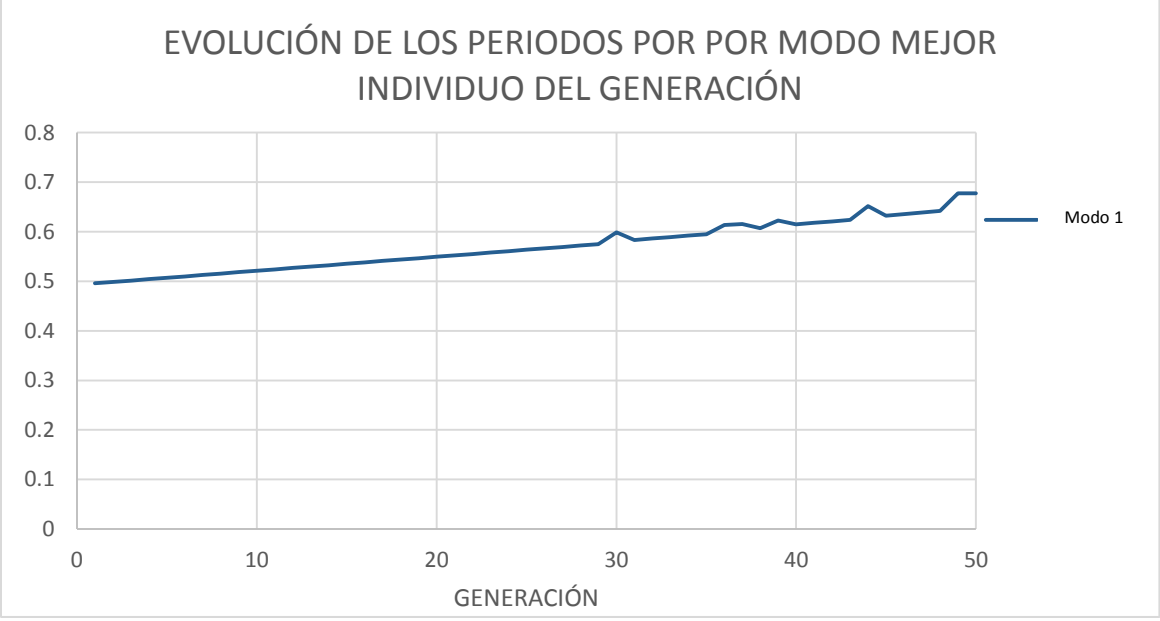


Figura 7.3.9-4 Periodos del Marco 3x7

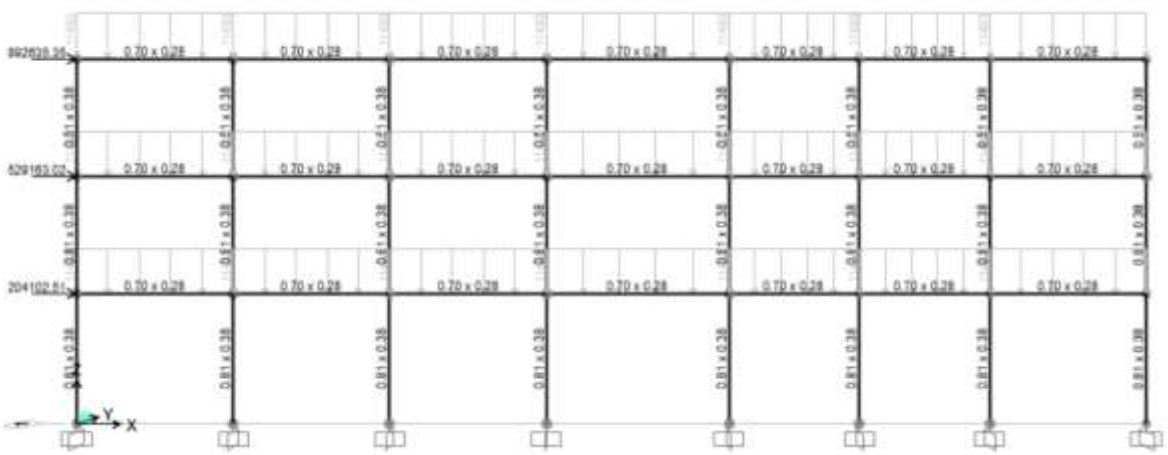


Figura 7.3-4 Marco 3x7 Mejor individuo

En la *Figura 7.3-4* nos muestra como se ve el individuo con mejor eficiencia, ya con las secciones calculadas y sus fuerzas sísmicas, de igual forma que se presentan en el marco anterior.

7.4. Ejemplo 3: Marco 6x4

```
Numero de individuos de la población: 100
Número de generaciones: 50
Escribe el factor de carga deseado: 1.1
Crea tu propio marco, cuya gravedad(m/s^2) y modulo de elasticidad(kg/m2) son:
MODULO DE ELASTICIDAD (kg/m2): 1739252713
GRAVEDAD (m/s^2):  9.81
¿Los muros estan ligados?:
    1)SI    2)NO
2
¿Cúantos niveles tiene tu edificio? 6
¿Cúantos crujias tiene tu edificio? 4
NUMERO DE COLUMNAS DE ENTEREPISO POR NIVEL:  5
Escribe la ALTURA 1 en m: 4
Escribe la ALTURA 2 en m: 4
Escribe la ALTURA 3 en m: 4
Escribe la ALTURA 4 en m: 4
Escribe la ALTURA 5 en m: 3.5
Escribe la ALTURA 6 en m: 3.5
ALTURA 1 (m): 4.0
ALTURA 2 (m): 4.0
ALTURA 3 (m): 4.0
ALTURA 4 (m): 4.0
ALTURA 5 (m): 3.5
ALTURA 6 (m): 3.5
Escribe la BASE 1 en m: 6
Escribe la BASE 2 en m: 4
Escribe la BASE 3 en m: 4
Escribe la BASE 4 en m: 6
BASE 1 (m): 6.0
BASE 2 (m): 4.0
BASE 3 (m): 4.0
BASE 4 (m): 6.0
```

Figura 7.4-1 Especificaciones Marco 6x4

En la *Figura 7.4-1* están las especificaciones ya mencionadas como sus dimensiones, número de individuos y número de generaciones, su factor de carga es *1.1* y sus muros no están ligados dándonos un estado límite de distorsión de *0.12*.

Figura 7.4-2 Datos de la NTC en Python

Como se puede observar en la *Figura 7.4-2* decidimos utilizar datos basados en las Normas Técnicas Complementarias para Diseño Sísmico (Puebla, 2017). Se utilizó el coeficiente sísmico *0.32* y el factor de comportamiento sísmico de *2*. También lo seleccionamos para que pertenezca al grupo B.

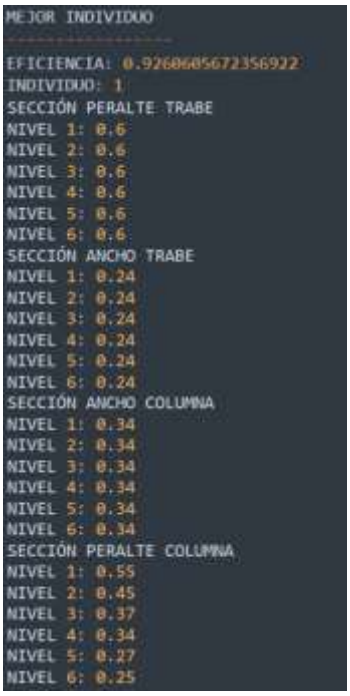


Figura 7.4-3 Secciones del MARCO 6X4

La *Figura 7.4-3* nos muestra el individuo óptimo con mayor eficiencia, y también nos muestra las dimensiones que generó, cruzó y mutó el algoritmo para que sea el mejor. En este caso, como puede ver, la eficiencia del algoritmo esta vez es 0.926060, dado que satisface algunas restricciones.

7.4.1. Pesos de Entrepiso

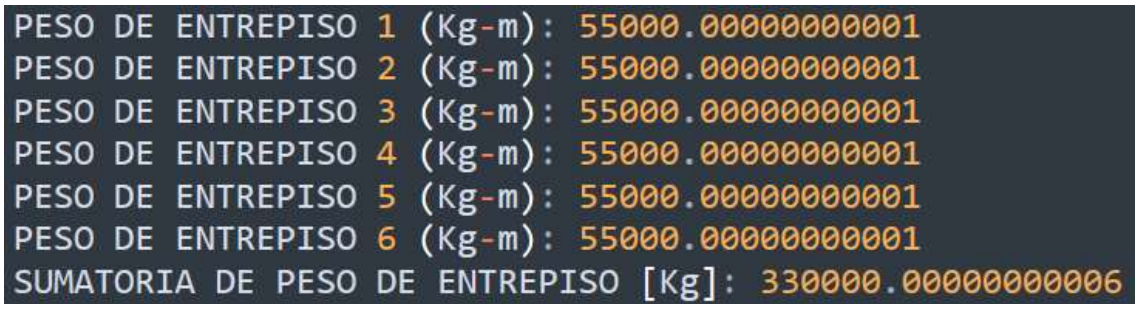


Figura 7.4.1-1 Carga Uniforme Sobre Trabe

Para nuestros pesos de entre piso (carga uniforme sobre trabe), como se aprecia en la *Figura 7.4.1-1*, el valor del peso de entrepiso es igual en todos los niveles, teniendo como valor 55000 $kg \cdot m$ y un sumatoria total de 330000 kg.

7.4.2. Rigidez (Wilbur) (*Ks*)

Las rigideces *(Ks)* calculadas con las expresiones de Wilbur en el algoritmo genético, nos arrojaron los siguientes valores:

- *Ks 1: 4468373.5379* $^{kg}/_{m}$
- *Ks 2:* 2284644.1294 $^{kg}/_{m}$
- *Ks 3:* 1561822.0673 $^{kg}/_{m}$
- *Ks 4:* 1320978.9501 $^{kg}/_{m}$
- *Ks 5:* 1104265.8713 $^{kg}/_{m}$
- *Ks 6:* 916514.0891 $^{kg}/_{m}$

7.4.3. Fuerzas (*F*) (Análisis Dinámico)

Las fuerzas horizontales calculadas por el análisis dinámico durante el proceso del algoritmo tienen como sus valores por nivel los siguientes:

- *F 1:* 2816 kg
- *F 2:* 5632 kg
- *F 3:* 8448 kg
- *F 4:* 11264 kg
- *F 5:* 13728 kg
- *F 6:* 16192 kg

7.4.4. Desplazamiento Total (*St*)

Los desplazamientos *(St)* por análisis estático son el resultado del equilibrio entre las fuerzas aplicadas y las reacciones en la estructura, los valores arrojados por el mejor individuo son:

- *St 1:* 0.01299 m
- *St 2:* 0.03718 m
- *St 3:* 0.06896 m
- *St 4:* 0.10014 m
- *St 5:* 0.12723 m

- *St 6:* 0.14490 m

7.4.5. Masas (*Ma*)

Las masas (Ma) fueron el resultado de la operación $\frac{Peso\ entrepiso}{gravedad}$, dándonos como resultado una masa de 5606 kg para todos los niveles.

7.4.6. Eigenvalores y Eigenvectores

Los eigenvalores calculados que dieron como resultado una matriz de 6x1, con los siguientes valores:

$$\begin{bmatrix} 1450.1822 \\ 828.6844 \\ 573.4228 \\ 345.8606 \\ 142.9335 \\ 20.1476 \end{bmatrix}$$

Mientras que el resultado del cálculo de los eigenvectores nos dio una matriz de 6*x6* con los siguientes valores:

$$\begin{bmatrix} -0.8445 & -0.3466 & 0.2871 & 0.2402 & -0.1537 & 0.0534 \\ 0.5091 & -0.3196 & 0.4446 & 0.5061 & -0.4005 & 0.1553 \\ -0.1612 & 0.67075 & -0.2401 & 0.2667 & -0.5560 & 0.2932 \\ 0.0387 & -0.5173 & 0.4653 & -0.4078 & -0.4025 & 0.4311 \\ -0.0071 & 0.2380 & 0.6200 & -0.4987 & 0.0731 & 0.5520 \\ 0.0009 & -0.0584 & -0.2472 & 0.4469 & 0.5824 & 0.6296 \end{bmatrix}$$

7.4.7. Periodos (*T*)

Los periodos *(T)* obtenidos mediante las operaciones realizadas con el algoritmo para el individuo con la mejor eficiencia de todas las generaciones son los siguientes:

- *T 1:* 1.39980 s
- *T 2:* 0.52554 s
- *T 3:* 0.33785 s
- *T 4:* 0.26238 s
- *T 5:* 0.21826 s
- *T 6:* 0.16499 s

7.4.8. Distorsiones máximas (*Smax*)

Las distorsiones máximas obtenidos mediante las operaciones realizadas con el algoritmo fueron los siguientes valores todos pasando por la distorsión permisible:

- *Smax 1:* 0.00448
- *Smax 2:* .00848
- *Smax 3:* 0.01139
- *Smax 4:* 0.01142
- *Smax 5:* 0.01168
- *Smax 6:* 0.00793

7.4.9. Gráficas

A continuación, en la *Figura 7.4.9-1* se muestra, por medio de cuatro gráficas, el comportamiento de la evolución que obtuvimos conforme avanzaron las generaciones, optimizando su eficiencia.

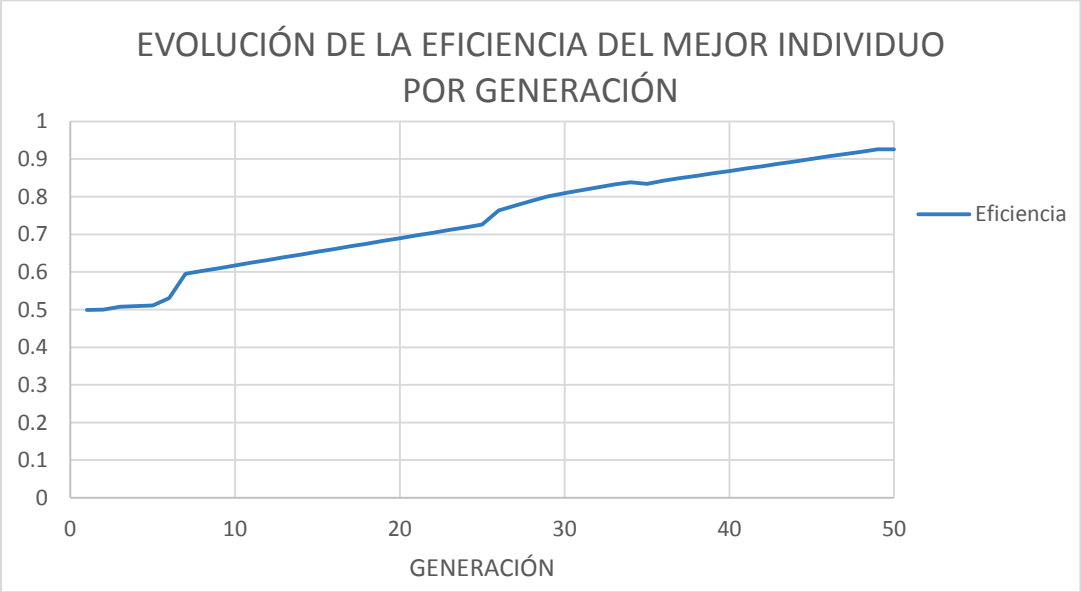


Figura 7.4.9-1 Eficiencia del Marco 6x4

En la *Figura 7.4.9-2* se muestra la evolución de las distorsiones máximas tomando en cuenta al mejor individuo de cada generación, se aprecia como varía en algunos puntos siendo más grandes o más pequeñas, haciendo que no se tenga unos

mejores valores, mostrándonos que para el mejor individuo con mejores distorsiones, sus distorsiones por cada nivel son más constantes.

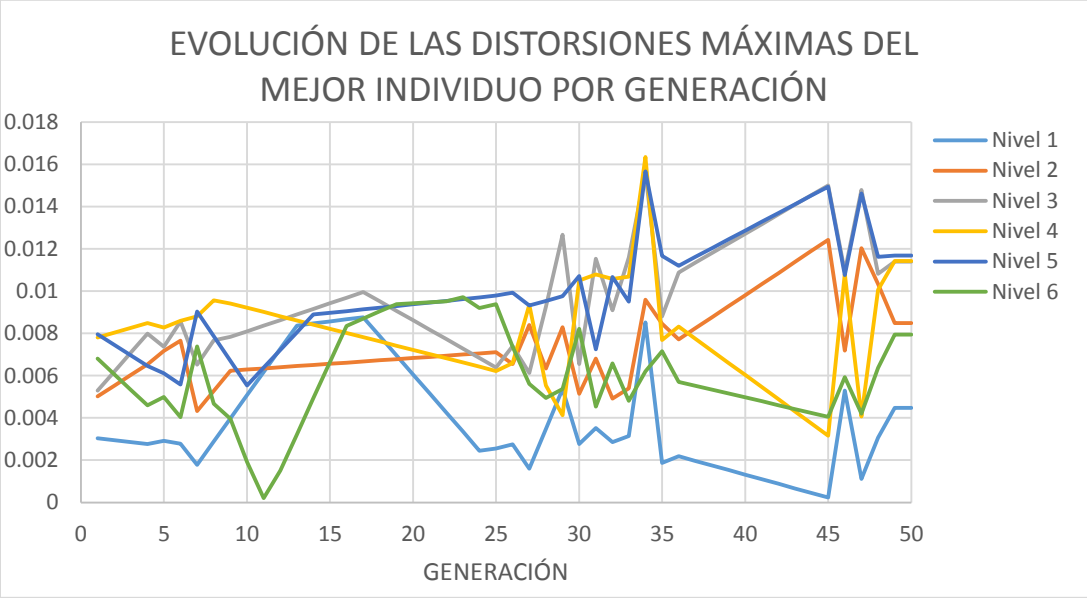


Figura 7.4.9-2 Distorsiones máximas del Marco 6x4

En la *Figura 7.4.9-3* se observa, de igual manera, la gráfica del primer ejemplo como la gráfica se dispara en algunos puntos y en otros se estabilizan siendo al final, en los últimos sujetos, cuando se logra una buena consistencia en sus valores.

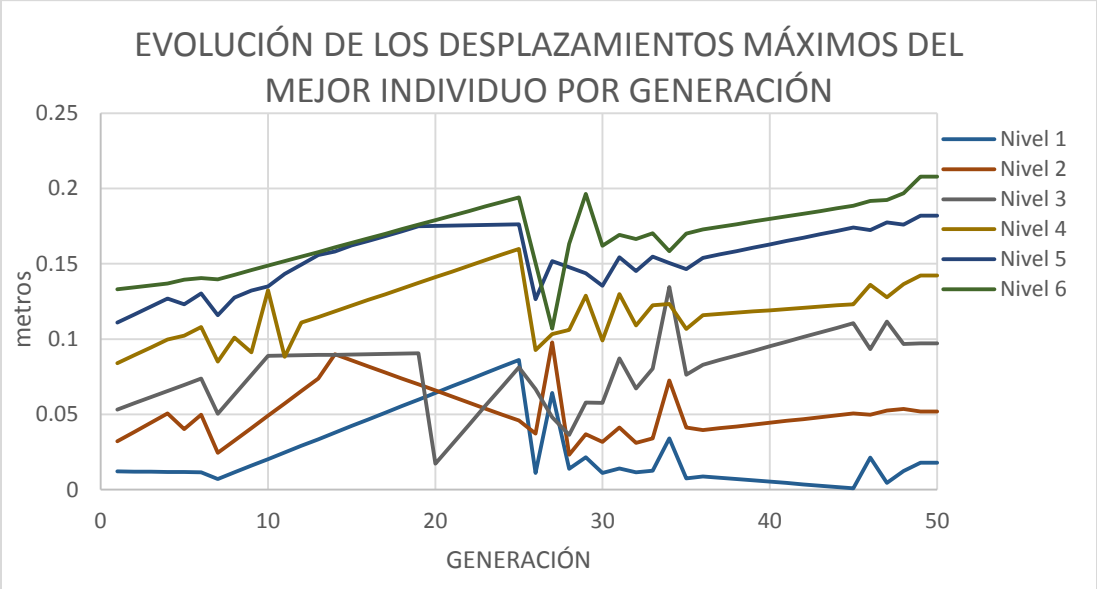


Figura 7.4.9-3 Desplazamientos máximos del Marco 6x4

En la *Figura 7.4.9-4* se muestra cómo se fue optimizando los periodos teniendo como valores una gráfica con una buena consistencia.

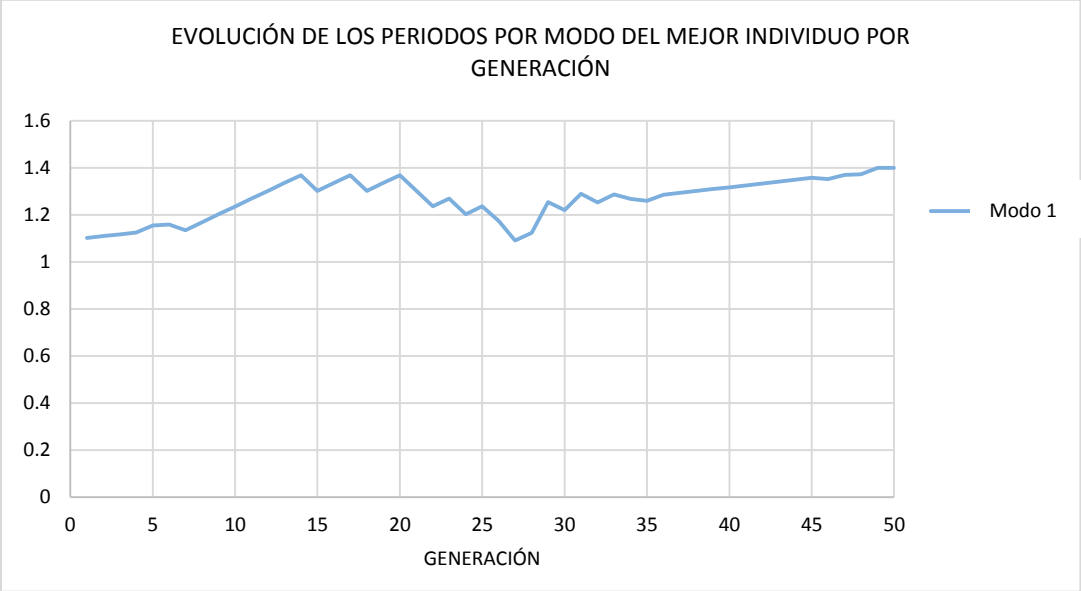


Figura 7.4.9-4 Periodos del Marco 6x4

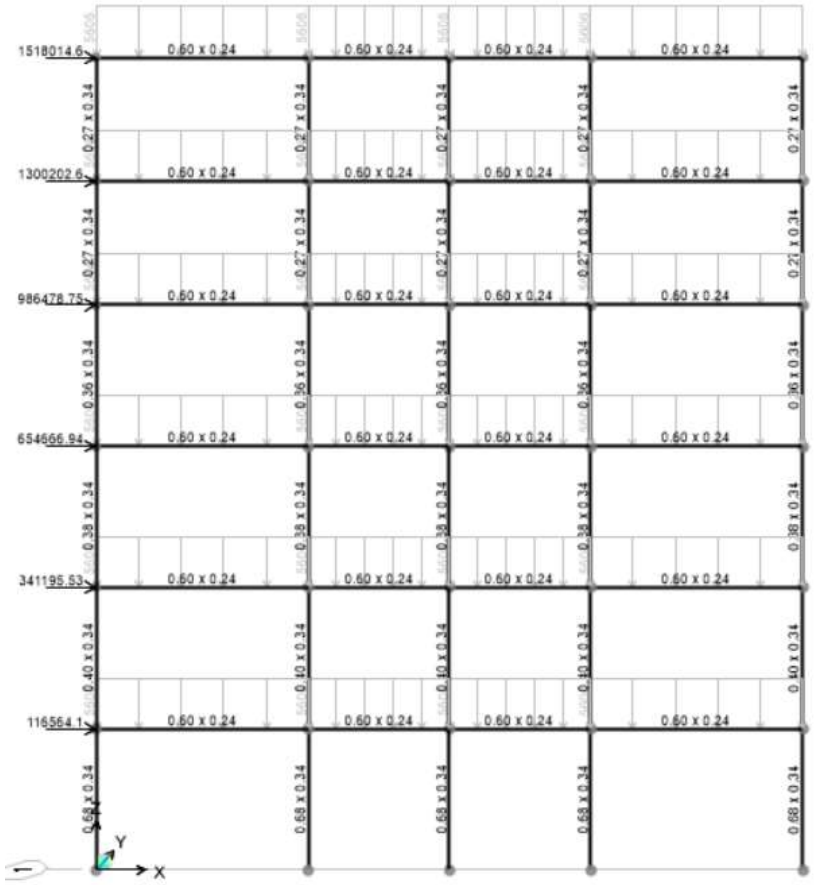


Figura 7.4-10 Marco 6x4 Mejor individuo

En la *Figura 7.4-10* se nos muestra, igual a los marcos anteriores, como se ve el individuo con mejor eficiencia ya con las secciones calculadas y sus fuerzas sísmicas.

CAPÍTULO 8: COMPARACIÓN DE RESULTADOS

8.1. Comparación del Marco 5x4

Tabla 3 Desplazamientos calculados con SAP3000 para el marco 5x4

TABLE: Joint Displacements

Joint	OutputCase	CaseType	UX
Text	Text	Text	m
1	MODAL	LinStatic	0
2	MODAL	LinStatic	0.038974
3	MODAL	LinStatic	0.081098
4	MODAL	LinStatic	0.122839
5	MODAL	LinStatic	0.166543
6	MODAL	LinStatic	0.202698

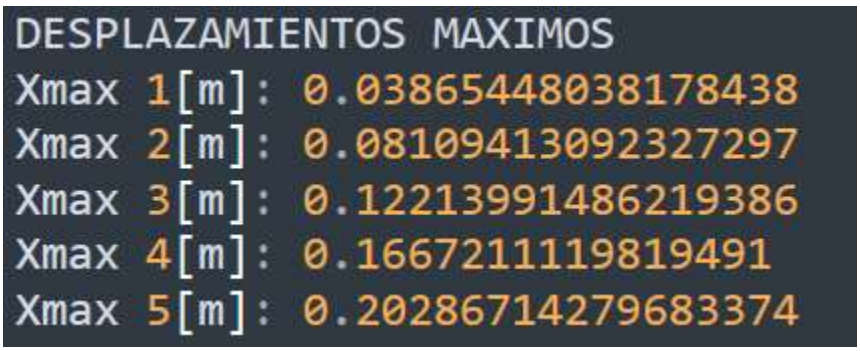


Figura 8.1-1 Desplazamientos Máximos del Marco 5x4 con el Algoritmo

La *Tabla 3* nos muestra los resultados obtenidos en los nodos que corresponden a los obtenidos con los desplazamientos máximos de la *Figura 8.1-1*. Como se puede apreciar los resultados de los desplazamientos son muy parecidos solo variando por algunas centésimas, no varía más de lo permitido que sería el *2%.*

Tabla 4 Fuerzas calculados con SAP3000 para el marco 5x4

TABLE: Element Joint Forces - Frames

Frame	Joint	OutputCase	CaseType	FX
2	2	SISMICA	LinStatic	124920.83
3	3	SISMICA	LinStatic	267724.09
4	4	SISMICA	LinStatic	416797.69
5	5	SISMICA	LinStatic	596747.40
6	6	SISMICA	LinStatic	761501.93

```
FUERZAS SISMICAS CON DESPLAZAMIENTOS
Fmax 1 [kg]: 124920.8416857881
Fmax 2 [kg]: 267724.1048267373
Fmax 3 [kg]: 416797.70596807596
Fmax 4 [kg]: 596747.4108768384
Fmax 5 [kg]: 761501.9464247333
```

Figura 8.1-2 Fuerzas sísmicas del Marco 5x4 con el Algoritmo

La *Tabla 4* nos muestra los resultados de las fuerzas obtenidas por el cálculo del programa SAP3000 que corresponden a los obtenidos con las fuerzas sísmicas del marco 5 x 4, como se aprecia en *Figura 8.1-2*, se observa que los resultados obtenidos son los mismos para ambos procedimientos.

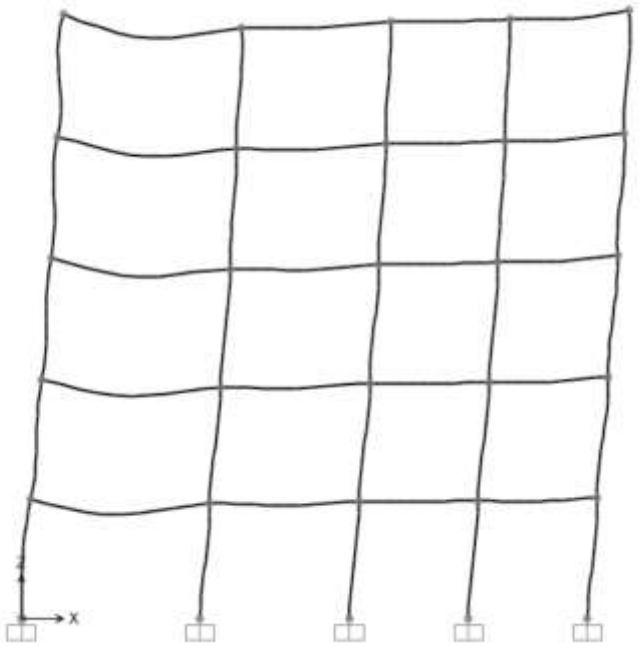


Figura 8.1-3 Imagen de cómo se desplazó el Marco 5x4 con el SAP3000

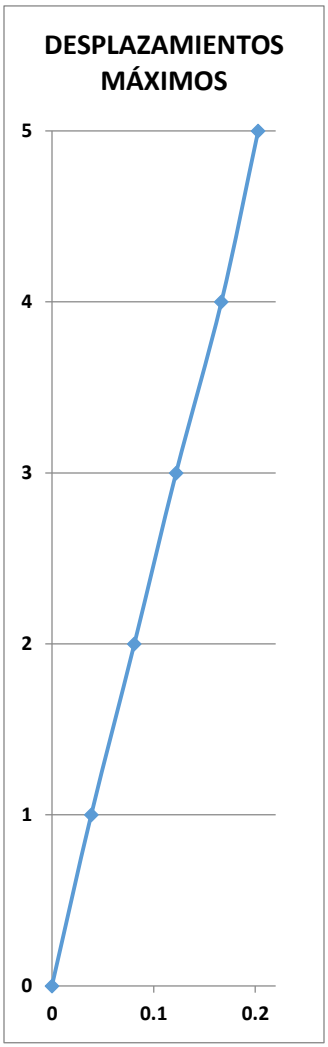


Figura 8.1-4 Como se desplaza el marco con el algoritmo mediante una gráfica

A partir de lo que se presenta en la *Figura 8.1-3,* se puede deducir que el marco para ambos sistemas (el algoritmo genético y el SAP3000) se asemeja suficiente con el de la *Figura 8.1-4.*

8.2. Comparación del Marco 3x7

Tabla 5 Desplazamientos calculados con SAP3000 para el marco 3x7

TABLE: Joint Displacements

Joint	OutputCase	CaseType	UX
7	MODAL	LinStatic	0.0179628
8	MODAL	LinStatic	0.0440926
9	MODAL	LinStatic	0.0706338

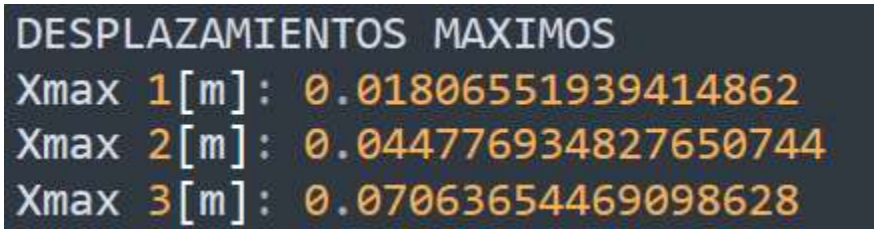


Figura 8.2-1 Desplazamientos Máximos del Marco 3x7 con el Algoritmo

La *Tabla 5* nos muestra los resultados obtenidos en los nodos que corresponderían a los obtenidos con los desplazamientos máximos de la *Figura 8.2-1*. Como se puede apreciar, los resultados de los desplazamientos son muy parecidos, solo variando por algunas centésimas.

Tabla 6 Fuerzas calculados con SAP3000 para el marco 3x7

TABLE: Element Joint Forces - Frames

Frame	Joint	OutputCase	CaseType	FX
7	2	SISMICA	LinStatic	204102.51
8	3	SISMICA	LinStatic	529163.02
9	4	SISMICA	LinStatic	892635.35

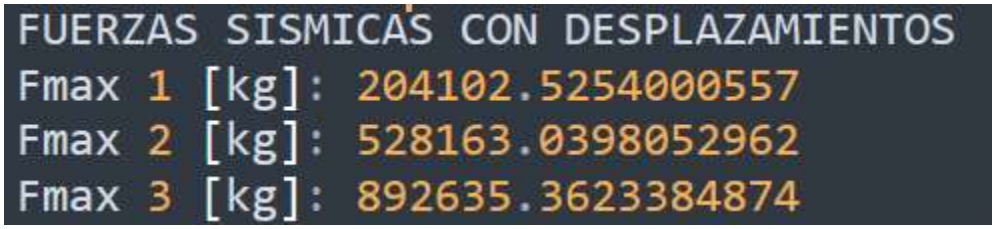


Figura 8.2-2 Fuerzas sísmicas del Marco 3x7 con el Algoritmo

La *Tabla 6* nos muestra los resultados de las fuerzas obtenidas por el cálculo del programa SAP3000, que corresponden a los obtenidos con las fuerzas sísmicas del marco 3 x 7, como se aprecia en *Figura 8.2-2.* Se observa que los resultados obtenidos son los mismos para ambos procedimientos.

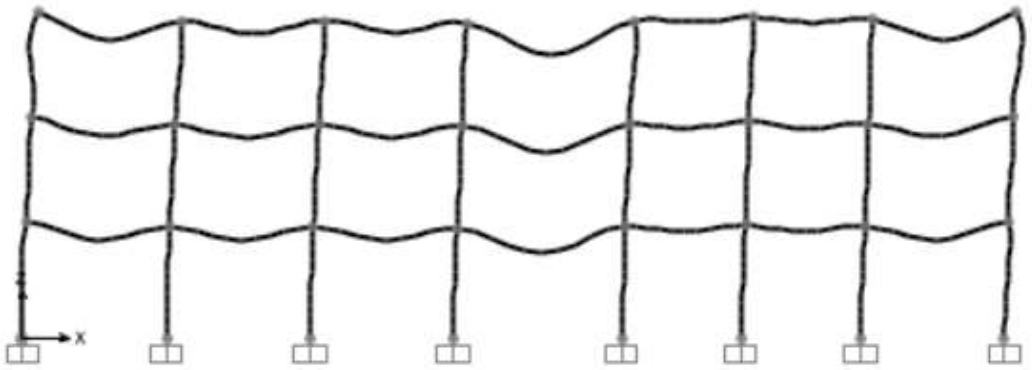


Figura 8.2-3 Imagen de cómo se desplazó el Marco 3x7 con el SAP3000

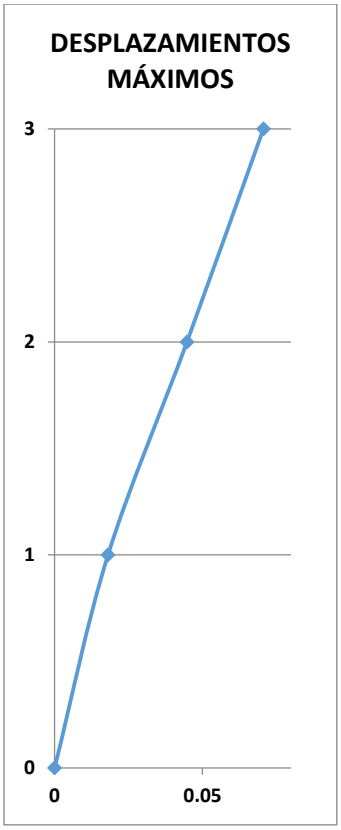


Figura 8.2-4 Como se desplaza el marco con el algoritmo mediante una gráfica

Analizando la *Figura 8.2-3* se puede deducir que el marco para ambos sistemas (el algoritmo genético y el SAP3000) se asemeja suficiente con el de la *Figura 8.2-4.*

8.3. Comparación del Marco 6x4

Tabla 7 Desplazamientos calculados con SAP3000 para el marco 6x4

TABLE: Joint Displacements

Joint	OutputCase	CaseType	UX
1	DEAD	LinStatic	0
2	DEAD	LinStatic	0.017184
3	DEAD	LinStatic	0.051410
4	DEAD	LinStatic	0.096240
5	DEAD	LinStatic	0.141813
6	DEAD	LinStatic	0.180997
7	DEAD	LinStatic	0.207699

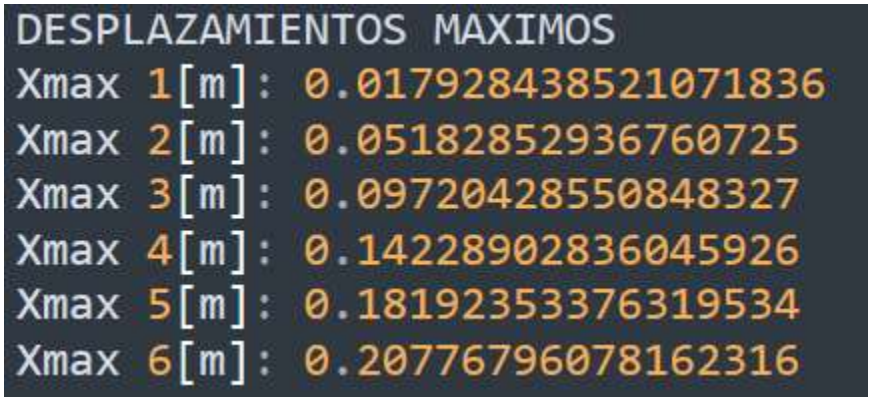


Figura 8.3-1 Desplazamientos Máximos del Marco 6x4 con el Algoritmo

La *Tabla 7* nos muestra los resultados obtenidos en los nodos que corresponderían a los obtenidos con los desplazamientos máximos de la *Figura 8.3-1*. Como se puede apreciar, los resultados de los desplazamientos son muy parecidos, solo variando por algunas centésimas.

Tabla 8 Fuerzas calculados con SAP3000 para el marco 6x4

TABLE: Element Joint Forces - Frames

Frame	Joint	OutputCase	CaseType	FX
60	37	DEAD	LinStatic	116564.1
61	38	DEAD	LinStatic	341195.53
62	39	DEAD	LinStatic	654666.94
63	40	DEAD	LinStatic	986478.75
64	41	DEAD	LinStatic	1300202.57
95	42	DEAD	LinStatic	1518014.59

```
FUERZAS SISMICAS CON DESPLAZAMIENTOS
Fmax 1 [kg]: 116564.11393748128
Fmax 2 [kg]: 341195.54825377476
Fmax 3 [kg]: 654666.9554415136
Fmax 4 [kg]: 986478.7654696291
Fmax 5 [kg]: 1300202.5884286768
Fmax 6 [kg]: 1518014.6062997733
```

Figura 8.3-2 Fuerzas sísmicas del Marco 6x4 con el Algoritmo

La *Tabla 8* nos muestra los resultados de las fuerzas obtenidas por el cálculo del programa SAP3000, que corresponden a los obtenidos con las fuerzas sísmicas del marco 6 x 4, como se aprecia en *Figura 8.3-2.* Se observa que los resultados obtenidos son los mismos para ambos procedimientos.

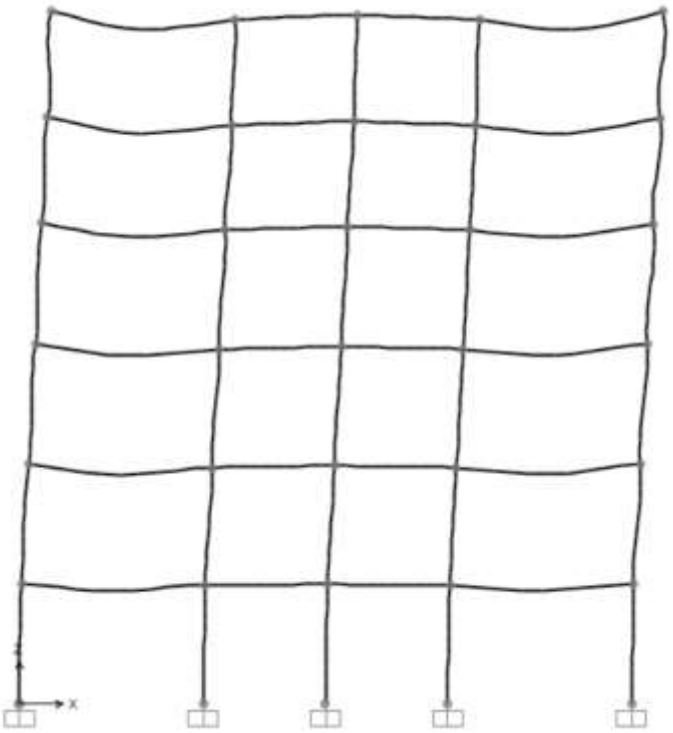


Figura 8.3-3 Imagen de cómo se desplazó el Marco 6x4 con el SAP30003

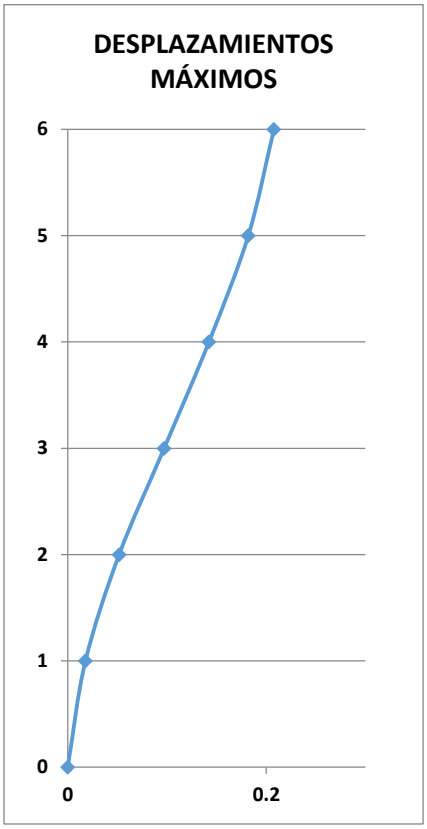


Figura 8.3-4 Como se desplaza el marco con el algoritmo mediante una gráfica

Al observar la *Figura 8.3-3* se puede deducir que el marco para ambos sistemas (el algoritmo genético y el SAP3000) se asemeja suficiente con el de la *Figura 8.3-4.*

CONCLUSIONES

La optimización de una función algorítmica implica encontrar el equilibrio perfecto entre eficiencia y precisión. Al perfeccionar los procesos internos, reducir la complejidad y minimizar el tiempo utilizado, se logra una ejecución más rápida y efectiva. La continua refinación de algoritmos, combinada con estrategias inteligentes de manejo de datos, conduce a un rendimiento óptimo, permitiendo que las funciones algorítmicas aborden con eficacia una variedad de desafíos en este caso en la ingeniería civil.

Utilizar un algoritmo genético como un procedimiento de búsqueda es muy eficiente, ya que de los tres ejemplos utilizados se logró la obtención de tres marcos optimizados en un corto tiempo y con la mejor eficiencia posible, sin embargo nuestro algoritmo genético puede mejorar aún más, haciendo que este algoritmo sirva como base para que en un futuro se pueda mejorar en varios aspectos, porque cada día que transcurre la tecnología va avanzando, haciendo que este proyecto no quede obsoleto.

La eficiencia del algoritmo genético se destacó en su capacidad para abordar el problema mediante la simulación de procesos evolutivos. Su habilidad para explorar amplios espacios de soluciones, adaptarse a cambios y converger hacia óptimos locales hizo que este algoritmo sea una herramienta versátil y efectiva en la búsqueda de la solución de la obtención de las dimensiones en las columnas, desde optimización hasta diseño, contribuyendo significativamente a la resolución eficiente del problema.

Dado los resultados mostrados, podemos decir que las expresiones de Wilbur son efectivas en todas las circunstancias y condiciones mostradas. Sin embargo, es importante recalcar que la ciencia y la tecnología están en constante evolución, por lo que incluso las fórmulas más sólidas pueden ser mejoradas o refinadas con el tiempo y nuevos descubrimientos.

Nuestra hipótesis planteada se cumple, dado que el proyecto nos proporcionó un respaldo sólido de la teoría propuesta. No obstante, las restricciones colocadas en nuestro algoritmo pueden cambiar cuando las cargas y las secciones de las trabes sean diferentes.

En esta investigación cuidamos la continuidad de las columnas, de tal manera que el peralte de nivel superior siempre fuera menor o igual que el del nivel inferior.

Analizando esta investigación podemos decir que el número de individuos asignado y el número de generaciones asignadas, pueden no ser quizás el máximo de la optimización. Si el marco es más grande necesitaría un mayor número de individuos o, tal vez, para llegar a su eficiencia total no podemos deducir si ya fue el 100% de la optimización porque quizás todavía podían pasar 3 o 5 generaciones más, lo que se desprenden de la tendencia en las curvas que no se estabilizaron totalmente.

BIBLIOGRAFÍA

Avilés López, J., Corona Carlos, G. A., González Espinoza, C., & Lira, J. E. (2017). *NORMAS TECNICAS COMPLEMENTARIAS PARA SISMO PUEBLA.* Puebla: Periodico Oficial del Estado de Puebla.

Castillo, M. (2020). *Analisis SIismico.* Puebla: Benemerita Universidad Autonoma de Puebla.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection.* London: Jonh Murray. Recuperado el 26 de 06 de 2023, de http://darwin-online.org.uk/content/frameset?itemID=F373&viewtype=text&pageseq=1

Davis, L. D. (1993). *Handbook of Genetic Algorithms*.* New York: an Nostrand Reinhold.

De Jong, K. A. (1975). *Analysis of the behavior of a class of genetic adaptive systems.* Michigan: University of Michigan.

Jarquín Laguna, R. (2014). *Aplicación de algoritmos genéticos en Ingeniería Civil.* Ciudad de Mexico: Universidad Nacional Autónoma de México .

Marczyk, A. (2004). *Algoritmos genéticos y computación evolutiva.* Computer Based Learning Unit, University of Leeds. Obtenido de Algoritmos genéticos y computación evolutiva.

NTCS. (15 de Diciembre de 2017). *Gaceta oficial de la ciudad de México*.

Puebla, M. d. (2017). NORMAS TÉCNICAS PARA DISEÑO POR SISMO. En M. d. Puebla, *COREMUN.* Puebla: Periodico oficial del estado.

Valverde Salazar, H. D. (2015). *Optimizacion de vigas-trade tipo I armadas mediante placas de acero, utilizando algoritmos geneticos.* Chimborazo: Riobamba: Universidad Nacional de Chimborazo.

ANEXOS

1. Código

```
import math
import random
import numpy as np
import pandas as pd
import xlwt
from xlwt import Workbook
import colorama
import openpyxl
import copy
from colorama import init, Fore
colorama.init(autoreset=True)
n_ind=int(input("Numero de individuos de la población: "))
n_gen=int(input("Número de generaciones: "))
Fc=float(input("Escribe el factor de carga deseado: "))
print("Crea tu propio marco, cuya gravedad(m/s^2) y modulo de elasticidad(kg/m2) son: ")
ME= (11000*(math.sqrt(250)))*10000
G= 9.81
amor=.05
print("MODULO DE ELASTICIDAD (kg/m2): {:.0f}".format(ME))
print("GRAVEDAD (m/s^2): ", G)
opc= int(input("""¿Los muros estan ligados?:
   1)SI    2)NO
"""))
if opc==1:
  Rel=0.006
if opc==2:
  Rel=0.012
niveles=int(input("¿Cúantos niveles tiene tu edificio? "))
crujias=int(input("¿Cúantos crujias tiene tu edificio? "))
columnas=crujias+1
print("NUMERO DE COLUMNAS DE ENTEREPISO POR NIVEL: ", columnas)
H=[]
```

```
Wi=[]
i=0
while i<niveles:
   H.append( float(input("Escribe la ALTURA "+str(i+1)+" en m: ")))
   intentos=0
   while intentos <= 2:
      if H[i] <= 0:
         print("El valor escrito no es valido\nVuelve a intentarlo")
         H[i]=float(input("Escribe la ALTURA "+str(i+1)+" en m: "))
         intentos=intentos+1
      else:
         break
   i = i + 1
i=0
while i<niveles:
   print("ALTURA "+str(i+1)+" (m): "+str(H[i]))
   i=i+1
B=[]
i=0
while i<crujias:
   B.append(float(input("Escribe la BASE " + str(i + 1) + " en m: ")))
   intentos = 0
   while intentos <= 2:
      if B[i] <= 0:
         print("El valor escrito no es valido\nVuelve a intentarlo")
         B[i] = float(input("Escribe la BASE " + str(i + 1) + " en m: "))
         intentos = intentos + 1
      else:
         break
   i= i + 1
i=0
while i<crujias:
   print("BASE " + str(i + 1) + " (m): "+ str(B[i]))
   i=i+1
   #CARGA UNIFORME SOBRE TRABE
```

```
            Wit=2500*sum(B)
            Wit=Wit*Fc
            SB=sum(B)
            Wit=np.repeat(Wit,niveles)
            i=0
            while i<niveles:
               print("PESO DE ENTREPISO "+str(i+1)+" (Kg-m): "+str(Wit[i]))
               i=i+1
            Swit=sum(Wit)
            print("SUMATORIA DE PESO DE ENTREPISO [Kg]: "+str(Swit))
            print("¿Quieres utilizar los datos basadosen NORMAS TECNICAS COMPLEMENTARIAS PARA DISEÑO POR
SISMO (PUEBLA)?")
            print("""
               1)SI    2)NO
               """)
            eligio=int(input("-Selecciona una opcion: "))
            if eligio==2:
             Q1=1
             #Q2=1.5
             Q3=2
             Q4=3
             Q5=4
             cs=float(input("Escribe el coeficiente sismico deseado: "))
             Ta=float(input("Escribe la meseta espectral inferior deseado: "))
             Tb=float(input("Escribe la meseta espectral superior deseado: "))
             r=float(input("Escribe el Exponente que depende del tipo de terreno deseado: "))
             a0=float(input("Escribe el coeficiente de aceleracion del terreno deseado: "))
             print("3.1- ", Q1)
             print("3.3- ", Q3)
             print("3.4- ", Q4)
             print("3.5- ", Q5)
             fq= float(input("Elije el Factor de comportamiento sismico (Q): "))
             if fq== Q1  or Q3 or Q4 and Q5:
               print(fq)
             else:
```

```
    print("opcion invalida")
if eligio==1:
  Za="I (Terreno firme)"
  Za1=0.18
  Zb="II (Terreno intermedio)"
  Zb1=0.32
  Zc="III (Terreno blando)"
  Zc1=0.40
  Q1=1
  Q3=2
  Q4=3
  Q5=4
  while True:
    print("Tipo de terreno")
    print("1. ", Za)
    print(Za1, "(Coeficiente Sismico)")
    print("2. ", Zb)
    print(Zb1, "(Coeficiente Sismico)")
    print("3. ", Zc)
    print(Zc1, "(Coeficiente Sismico)")
    print("4. SALIR")
    opc= int(input("Elije el TIPO DE TERRENO: "))
    if opc==1:
      print(Za)
      print("1.- %.2f" %Za1)
      cs=Za1
      a0=0.0005
      Ta=0.15
      Tb=0.60
      r=1/2
      if cs== Za1:
        print("1.1- ", Q1)
        minc=.20
        print("1.3- ", Q3)
        minc=.20
```

```
        print("1.4- ", Q4)
        minc=.25
        print("1.5- ", Q5)
        minc=.25
    elif opc==2:
        print(Zb)
        print("2.- %.2f" %Zb1)
        cs= Zb1
        a0=0.09
        Ta=0.20
        Tb=1.5
        r=2/3
        if cs== Zb1:
            print("2.1- ", Q1)
            minc=.20
            print("2.3- ", Q3)
            minc=.20
            print("2.4- ", Q4)
            minc=.25
            print("2.5- ", Q5)
            minc=.25
    elif opc==3:
        print(Zc)
        print("3.- %.2f" %Zc1)
        cs= Zc1
        a0=0.011
        Ta=0.50
        Tb=2.5
        r=1
        if cs==Zc1:
            print("3.1- ", Q1)
            minc=.20
            print("3.3- ", Q3)
            minc=.20
            print("3.4- ", Q4)
```

```
        minc=.25
        print("3.5- ", Q5)
        minc=.25
    fq= float(input("Elije el Factor de comportamiento sismico (Q): "))
    if fq== Q1 or Q3 or Q4 and Q5:
      print(fq)
    else:
      print("opcion invalida")
    break
opc= int(input("""¿Qué grupo es?:
    1)A    2)B
"""))
if opc==1:
    cs= cs*1.5
    a0=a0*1.5
    Ta=Ta*1.5
    Tb=Tb*1.5
    r=r*1.5

if opc==2:
  pass
claromayor=max(B)
print("SECCIÓN PERALTE TRABE")
Hst=[]
i=0
while i<niveles:
  if i==0:
    limites_inf = claromayor/10
    limites_sup=claromayor/10
  else:
    limites_sup=Hst[i-1]
  Hst.append(round(random.uniform(limites_inf, limites_sup),2))
  i= i + 1
i=0
while i<niveles:
```

```
 print("NIVEL "+str(i+1)+": "+str(Hst[i]))
 i=i+1
print("SECCIÓN ANCHO TRABE")
Ast=[]
i=0
while i<niveles:
 if i==0:
  limites_inf = Hst[i]/2.5
  limites_sup=Hst[i]/2.5
 else:
  limites_sup=Ast[i-1]
 Ast.append(round(random.uniform(limites_inf, limites_sup),2))
 i= i + 1
i=0
while i<niveles:
 print("NIVEL "+str(i+1)+": "+str(Ast[i]))
 i=i+1
print("SECCIÓN ANCHO COLUMNA")
Asc=[]
#self.num_bits1 = []
#self.x1 = []
i=0
while i<niveles:
 limites_inf1 = Ast[i]+.10
 if i==0:
  limites_sup1= Ast[i]+.10
 else:
  limites_sup1=Asc[i-1]
 #self.x1.append(self.limites_sup1 - self.limites_inf1)
 #j = 0
 #aux =1
 #while 2**j <= self.x1[i]*100:
 #  j = j + 1
 #  aux = j
 #self.num_bits1.append(aux)
```

```
 #print(self.num_bits1)
 Asc.append(round(random.uniform(limites_inf1,limites_sup1),2))
 i= i + 1
#print(self.x1)
i=0
while i<niveles:
   print("NIVEL "+str(i+1)+": "+str(Asc[i]))
   i=i+1
class valores:
 def __init__(self):
   print("La seccion del PERALTE DE LA COLUMNA sera definida aleatoriamente por el programa...")
   print("SECCIÓN PERALTE COLUMNA")
   self.Hsc=[]
   self.num_bits2 = []
   self.x2 = []
   i=0
   while i<niveles:
    self.limites_inf2 = minc
    if i==0:
     self.limites_sup2=  (4*Ast[i])# no mayor 3veces el ancho de columna
    else:
     self.limites_sup2=self.Hsc[i-1]
    self.x2.append(self.limites_sup2 - self.limites_inf2)
    j = 0
    aux =1
    while 2**j <= self.x2[i]*100:
     j = j + 1
     aux = j
    self.num_bits2.append(aux)
    #print(self.num_bits2)
    self.Hsc.append(round(random.uniform(self.limites_inf2, self.limites_sup2),2))
    i= i + 1
   #print(self.x2)
   i=0
   while i<niveles:
```

```
    print("NIVEL "+str(i+1)+": "+str(self.Hsc[i]))
    i=i+1
  def deci_a_bin (self, valor_variable, limites_inf, x,num_bits):
    valor_venbin = 0
    self.bina = ""
    j = 0
    aux = 1
    while 2 ** j <= x * 100:
      j = j + 1
      aux = j
    self.num_bits=aux
    if x==0 or valor_variable==limites_inf:
      valor_venbin=1
    else:
      valor_venbin = (round((((2 ** num_bits)-1) / x) * (abs(valor_variable - limites_inf))))
    self.bina = (bin(valor_venbin)[2:])
    """print("bina   "+str(self.bina))
    print("num bits self   " + str(self.num_bits))
    print("num bits   " + str(num_bits))
    print("X   " + str(x))
    print("limite inf   " + str(limites_inf))
    print("Valor R   " + str(valor_variable))
    print("Valor F   " + str(valor_venbin))"""
    auxx = ''
    h = 0
    if num_bits < len(self.bina):
      self.bina=self.bina[num_bits:]
    while num_bits != len(self.bina) + h:
      auxx = auxx + '0'
      h = h + 1
    auxd = auxx + self.bina
    self.bina = auxd
    return (self.bina)
  def calculos(self):
    self.It=[((a*(b**3)))/12 for a,b in zip(Ast,Hst)]
```

```
            self.Ic=[((a*(b**3)))/12 for a,b in zip(Asc,self.Hsc)]
            self.Kc=[(a/b)*columnas for a,b in zip(self.Ic,H)]
            self.Kt=[(a/b) for a,b in zip(self.It,B)]
            self.Kt=sum(self.Kt)
            self.Kt=np.repeat(self.Kt, niveles)
            self.D=[]
            i=0
            while i<niveles:
             if i==0:
                self.D.append(((4*H[i])/self.Kc[i])+((H[i]+H[i+1])/(self.Kt[i]+(self.Kc[i]/12))))
             elif i==1:
                self.D.append(((4*H[i])/self.Kc[i])+((H[i-1]+H[i])/(self.Kt[i-1]+(self.Kc[i-
1]/12)))+((H[i]+H[i+1])/self.Kt[i]))
             elif i==niveles-1:
                self.D.append(((4*H[i])/self.Kc[i])+(((2*H[i-1])+H[i])/self.Kt[i-1])+(H[i]/self.Kt[i]))
             else:
                self.D.append(((4*H[i])/self.Kc[i])+((H[i-1]+H[i])/self.Kt[i-1])+((H[i]+H[i+1])/self.Kt[i]))
             i=i+1
            self.Ks=[((48*ME)/(a*b)) for a,b in zip(self.D,H)]
            self.Hi=[]
            i=0
            while i<niveles:
             j=0
             Hia=0
             while j<=i:
                Hia=Hia+H[j]
                j=j+1
             self.Hi.append(Hia)
             i=i+1
            self.WiHi=[(a*b) for a,b in zip(Wit,self.Hi)]
            self.SWiHi=sum(self.WiHi)
            self.Vb=(Swit*cs)/fq
            self.F=(self.WiHi/self.SWiHi)*self.Vb
            self.F=np.array(self.F)
            self.F=self.F*Fc
```

```
self.V=[]
i=0
while i<niveles:
 j=niveles
 Va=0
 while j>i:
    Va=Va+self.F[j-1]
    j=j-1
 self.V.append(Va)
 i=i+1
self.FiHi=[(a*b) for a,b in zip(self.F,self.Hi)]
self.Sr=[(a/b) for a,b in zip(self.V,self.Ks)]
self.St=[]
i=0
while i<niveles:
 j=0
 Sta=0
 while j<=i:
  Sta=Sta+self.Sr[j]
  j=j+1
 self.St.append(Sta)
 i=i+1
self.Idx=[(a/b) for a,b in zip(self.Sr,H)]
 self.Fidi=[(a*b) for a,b in zip(self.F,self.St)]
self.St2 = [n**2 for n in self.St]
self.Widi=[a*b for a,b in zip(Wit,self.St2)]
self.Ma=Wit/G
self.Ma=np.array(self.Ma)
self.M=np.diag(self.Ma)
self.M=np.array(self.M)
self.MINV=np.linalg.inv(self.M)
def gen_Kmatrix_np(*Ks):
 self.Ks = np.array(self.Ks)
 shape = len(self.Ks)
 self.K = np.zeros((shape, shape))
```

```
 idx1, idx2 = np.arange(1, shape), np.arange(shape - 1)
 # Llenamos diagonal principal
 self.K[-1, -1] = self.Ks[-1]
 self.K[idx2, idx2] = self.Ks[:-1] + self.Ks[1:]
 # Llenamos diagonales paralelas superior e inferior
 self.K[idx1, idx2] =self.K[idx2, idx1] = -self.Ks[1:]
 return self.K
self.K=gen_Kmatrix_np(*self.Ks)
self.A=np.matmul(self.MINV,self.K)
self.Eigenvalores, self.Eigenvectores=np.linalg.eig(self.A)
rank=np.argsort(self.Eigenvalores)
self.Eigenva=[]
self.Eigenve=[]
i=0
while i<niveles:
 j=0
 while j<niveles:
  if rank[j]==i:
   self.Eigenva.append(self.Eigenvalores[j])
   self.Eigenve.append(self.Eigenvectores[:,j])
  j=j+1
 i=i+1
self.Eigenve=np.transpose(self.Eigenve)
self.w2=sorted(self.Eigenva)
self.w=np.sqrt(self.w2)
self.T=(math.pi*2)/self.w
self.T2= self.T**2
self.f=1/self.T
self.Tc=2*Ta
self.r=2/3
self.Sa1=[]
i=0
while i<niveles:
 if self.T[i]<Ta:
  self.Sa1.append(a0+(cs-a0)*(self.T[i]/Ta))
```

```
 elif Ta<self.T[i]<Tb:
  self.Sa1.append(cs)
 elif self.T[i]>Tb:
  self.Sa1.append(cs*((Tb/self.T[i])**self.r))
 i=i+1
self.fq2=[]
i=0
while i<niveles:
 if Ta<=self.T[i]:
  self.fq2.append(fq)
 else:
  self.fq2.append(1+(((fq-1)/Ta))*self.T[i])
 i=i+1
self.fq1=[]
i=0
while i<niveles:
 if self.fq2[i]<1:
  self.fq1.append(1)
 else:
  self.fq1.append(self.fq2[i])
 i=i+1
self.Sa=[]
i=0
while i<niveles:
 self.Sa.append(((self.Sa1[i]*G)/(self.fq1[i]))*1)
 i=i+1
self.Sa=np.array(self.Sa)
self.Sd=self.Sa/self.w**2
self.FIT=np.transpose(self.Eigenve)
self.FITM=np.matmul(self.FIT,self.M)
self.FITK=np.matmul(self.FIT,self.K)
self.KA=np.matmul(self.FITK,self.Eigenve)
self.I=np.repeat(1,niveles)
self.I=np.array(self.I)
self.L=np.matmul(self.FITM,self.I)
```

```
self.MA=np.matmul(self.FITM,self.Eigenve)
self.DKA=np.diag(self.KA)
self.DKA=np.array(self.DKA)
self.DMA=np.diag(self.MA)
self.DMA=np.array(self.DMA)
self.FP=[(a/b) for a,b in zip(self.L,self.DMA)]
self.Wd=self.w*math.sqrt(1-(amor**2))
self.Wd=np.array(self.Wd)
self.Td=(math.pi*2)/self.Wd
self.Td=np.array(self.Td)
self.Fd=1/self.Td
self.Fd=np.array(self.Fd)
self.Ymax=self.FP*self.Eigenve*self.Sd
self.Ymax=np.transpose(self.Ymax)
self.Ymax2=self.Ymax**2
self.ΣYmax2=[]
for j in range(len(self.Ymax2[0])):
 self.suma_columna=0
 for i in range(len(self.Ymax2)):
  self.suma_columna += self.Ymax2[i][j]
 self.ΣYmax2.append(self.suma_columna)
self.RΣYmax=np.sqrt(self.ΣYmax2)
self.RΣYmax=self.RΣYmax*fq
self.Smaxt=[]
self.Ymaxt=np.transpose(self.Ymax)
i=0
while i<niveles:
 if i==0:
  self.Smaxt.append((self.Ymaxt[0,:]/H[i]))
 else:
  self.Smaxt.append(((self.Ymaxt[i]-self.Ymaxt[i-1])/H[i]))
 i=i+1
self.Smax=np.transpose(self.Smaxt)
self.Smax=np.array(self.Smax)
self.Smax2=self.Smax**2
```

```
  self.ΣSmax=[]
  for j in range(len(self.Smax2[0])):
   self.suma_columna1=0
   for i in range(len(self.Smax2)):
    self.suma_columna1 += self.Smax2[i][j]
   self.ΣSmax.append(self.suma_columna1)
  self.RΣSmax=np.sqrt(self.ΣSmax)
  self.RΣSmax=self.RΣSmax*fq
  self.Fmax=np.matmul(self.K, self.Ymax)
  self.Fmax2=self.Fmax**2
  self.ΣFmax=[]
  for j in range(len(self.Fmax2[0])):
   self.suma_columna1=0
   for i in range(len(self.Fmax2)):
    self.suma_columna1 += self.Fmax2[i][j]
   self.ΣFmax.append(self.suma_columna1)
  self.RΣFmax=np.sqrt(self.ΣFmax)
  self.RΣFmax=self.RΣFmax*fq
  i=0
  self.distorsionmax=self.RΣSmax[0]
  while i<niveles:
   #self.promedio=self.promedio + (self.RΣSmax[i]/Rel)/niveles
   if self.RΣSmax[i] > self.distorsionmax:
    self.distorsionmax=self.RΣSmax[i]
   i=i+1
  return (self.distorsionmax)
 def texto1(self):
  #print("DESPLAZAMIENTOS MAXIMOS")
  i=0
  while i<niveles:
   #print("Xmax "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.RΣYmax[i]))
   i=i+1
  #print("DISTORSIONES DE ENTREPISO")
  i=0
  while i<niveles:
```

```
    #print("Smax "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.RΣSmax[i]))
    i=i+1
  def texto2(self):
   print("SECCIÓN PERALTE TRABE")
   i=0
   while i<niveles:
    print("NIVEL "+str(i+1)+": "+str(Hst[i]))
    i=i+1
   print("SECCIÓN ANCHO TRABE")
   i=0
   while i<niveles:    #
    print("NIVEL "+str(i+1)+": "+str(Ast[i]))
    i=i+1
   print("SECCIÓN ANCHO COLUMNA")
   i=0
   while i<niveles:
    print("NIVEL "+str(i+1)+": "+str(Asc[i]))
    i=i+1
   print("SECCIÓN PERALTE COLUMNA")
   i=0
   while i<niveles:
    print("NIVEL "+str(i+1)+": "+str(self.Hsc[i]))
    i=i+1
  def texto3(self):
   i=0
   while i<niveles:
    print("INERCIA DE TRABE "+str(i+1)+" (m^4): "+str(self.It[i]))
    i=i+1
   i=0
   while i<niveles:
    print("INERCIA DE COLUMNA "+str(i+1)+" (m^4): "+str(self.Ic[i]))
    i=i+1
   i=0
   while i<niveles:
    print("FUERZAS "+str(i+1)+" (F)[Kg]: "+str(self.F[i]))
```

```
 i=i+1
print("DESPLAZAMIENTOS MAXIMOS")
i=0
while i<niveles:
 print("Xmax "+str(i+1)+"[m]: "+str(self.RΣYmax[i]))
 i=i+1
print("DISTORSIONES DE ENTREPISO")
i=0
while i<niveles:
 print("Smax "+str(i+1)+": "+str(self.RΣSmax[i]))
 i=i+1
print("FUERZAS SISMICAS CON DESPLAZAMIENTOS")
i=0
while i<niveles:
 print("Fmax "+str(i+1)+" [kg]: "+str(self.RΣFmax[i]))
 i=i+1
i=0
while i<niveles:
 print("RIGIDEZ DE LAS COLUMNAS "+str(i+1)+" (ΣKc)[m^3]: "+str(self.Kc[i]))
 i=i+1
i=0
while i<niveles:
 print("RIGIDEZ DE LAS TRABES "+str(i+1)+" (ΣKt)[m^3]: "+str(self.Kt[i]))
 i=i+1
i=0
while i<niveles:
 print("DESPLAZAMIENTOS " +str(i+1)+" (D)[m^-2]: "+str(self.D[i]))
 i=i+1
i=0
while i<niveles:
 print("RIGIDEZ AXIAL DEL EJE "+str(i+1)+" (Ks)[kg/m]: "+str(self.Ks[i]))
 i=i+1
i=0
while i<niveles:
 print("Hi " +str(i+1)+" (Hi)[m]: "+str(self.Hi[i]))
```

```
 i=i+1
i=0
while i<niveles:
 print("WiHi "+str(i+1)+" [m]: "+str(self.WiHi[i]))
 i=i+1
print("SUMATORIA DE WiHi[m]: "+str(self.SWiHi))
print("CORTANTE BASAL (Kg): %.5f" %self.Vb)
i=0
while i<niveles:
 print("CORTANTES " +str(i+1)+" (V)[Kg]: "+str(self.V[i]))
 i=i+1
i=0
while i<niveles:
 print("MOMENTO DE VOLTEO "+str(i+1)+" (FiHi)[Kg-m]: "+str(self.FiHi[i]))
 i=i+1
i=0
while i<niveles:
 print("DESPLAZAMIENTO RELATIVO/Drift "+str(i+1)+" (Sr)[m]: "+str(self.Sr[i]))
 i=i+1
i=0
while i<niveles:
 print("DESPLAZAMIENTO TOTAL " +str(i+1)+" (St)[m]: "+str(self.St[i]))
 i=i+1
i=0
while i<niveles:
 print("INDÉX "+str(i+1)+": "+str(self.Idx[i]))
 i=i+1
i=0
while i<niveles:
 print("Fidi "+str(i+1)+" [Kg-m]: "+str(self.Fidi[i]))
 i=i+1
i=0
while i<niveles:
 print("Widi^2 "+str(i+1)+" [Kg-m^2]: "+str(self.Widi[i]))
 i=i+1
```

```
i=0
while i<niveles:
 print("MASAS "+str(i+1)+" [Kg]: "+str(self.Ma[i]))
 i=i+1
print("MATRIZ DE MASAS")
print(self.M)
print("MATRIZ INVERSA DE MASAS")
print(self.MINV)
print("MATRIZ DE RIGIDEZ")
print(self.K)
print("MATRIZ DINAMICA DEL SISTEMA")
print(self.A)
print("EIGENVALORES - lamba")
print(self.Eigenvalores)
print("EIGENVECTORES - AA")
print( self.Eigenvectores)
print("EIGENVECTORES - AA ORDENADOS")
print(self.Eigenve)
print("w2")
print(self.w2)
print("w")
print(self.w)
print("PERIODOS - T")
print(self.T)
print("FRECUENCIAS - f")
print(self.f)
print("Coeficiente de aceleracion del terreno - a0")
print(a0)
print("Limite inferior de la meseta espectral - Ta")
print(Ta)
print("Limite superior de la meseta espectral - Tb")
print(Tb)
print("Periodo caracteristico - Tc")
print(self.Tc)
print("Exponente que depende del tipo de terreno - r")
```

```
print(self.r)
print("Coeficiente sismico - c")
print(cs)
print("Amortiguamiento - ξ")
print(amor)
i=0
while i<niveles:
 print("Q1 "+str(i+1)+": "+str(self.fq1[i]))
 i=i+1
i=0
while i<niveles:
 print("ORDENADAS ESPECTRALES ACELERACION " +str(i+1)+" (Sa): "+str(self.Sa1[i]))
 i=i+1
i=0
while i<niveles:
 print("ORDENADAS ESPECTRALES ACELERACION FACTORIZADAS " +str(i+1)+" (Sa): "+str(self.Sa[i]))
 i=i+1
i=0
while i<niveles:
 print("ORDENADAS ESPECTRALES DESPLAZAMIENTO " +str(i+1)+" (Sd): "+str(self.Sd[i]))
 i=i+1
print("FIT")
print(self.FIT)
print("FITM")
print(self.FITM)
print("FITK")
print(self.FITK)
print("KA")
print(self.KA)
print("L")
print(self.L)
print("DKA")
print(self.DKA)
print("DMA")
print(self.DMA)
```

```
            print("I")
            print(self.I)
            print("MA")
            print(self.MA)
            print("PARAMETROS MODALES")
            print("K*")
            i=0
            while i<niveles:
              print("RIGIDEZ " +str(i+1)+" (K*): "+str(self.DKA[i]))
              i=i+1
            print("M*")
            i=0
            while i<niveles:
              print("MASAS " +str(i+1)+" (M*): "+str(self.DMA[i]))
              i=i+1
            i=0
            while i<niveles:
              print("FACTOR DE PARTICIPACIÓN "+str(i+1)+" (FP)[Y*]: "+str(self.FP[i]))
              i=i+1
            print("Amortiguamiento-ζ(%)")
            print(amor)
            i=0
            while i<niveles:
              print("FRECUENCIA CIRCULAR AMORTIGUADA "+str(i+1)+" (MODO "+str(i+1)+") (Wd): "+str(self.Wd[i]))
              i=i+1
            i=0
            while i<niveles:
              print("PERIODO AMORTIGUADO "+str(i+1)+" (MODO "+str(i+1)+") (Td): "+str(self.Td[i]))
              i=i+1
            i=0
            while i<niveles:
              print("FRECUENCIA LINEAL AMORTIGUADA "+str(i+1)+" (MODO "+str(i+1)+") (Fd): "+str(self.Fd[i]))
              i=i+1
              print("DESPLAZAMIENTOS")
```

```
 print("ϕ[i]*Sd[i]*FP[i]")
i=0
while i<niveles:
 print("Ymax "+str(i+1)+" (MODO "+str(i+1)+") (Ymax): "+str(self.Ymax[i]))
 i=i+1
 print("Ymax^2")
i=0
while i<niveles:
 print("Ymax^2 "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.Ymax2[i]))
 i=i+1
i=0
while i<niveles:
 print("SUMATORIA DE Ymax^2 "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.ΣYmax2[i]))
 i=i+1
i=0
while i<niveles:
 print("DISTORSIONES DE ENTREPISO "+str(i+1)+" (MODO "+str(i+1)+")(δmax): "+str(self.Smax[i]))
 i=i+1
print("δmax^2")
i=0
while i<niveles:
 print("Smax^2 "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.Smax2[i]))
 i=i+1
i=0
while i<niveles:
  print("SUMATORIA DE Smax^2 "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.ΣSmax[i]))
  i=i+1
i=0
while i<niveles:
 print("Fmax "+str(i+1)+" (MODO "+str(i+1)+") (Ymax): "+str(self.Fmax[i]))
 i=i+1
 print("Fmax^2")
i=0
while i<niveles:
 print("Fmax^2 "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.Fmax2[i]))
```

```
      i=i+1
    i=0
    while i<niveles:
     print("SUMATORIA DE Fmax^2 "+str(i+1)+" (MODO "+str(i+1)+"): "+str(self.ΣFmax[i]))
     i=i+1
    i=0
    print("COMPARACION DE LAS DISTORSIONES")
    print("LIMITE: ",Rel)
    i=0
    while i<niveles:
     if self.RΣSmax[i]<=Rel:
      print(Fore.GREEN+"Pasa")
     elif Rel<self.RΣSmax[i]:
      print(Fore.RED + "No Pasa")
     i=i+1
  def funcion_objt(self):
    self.calculos()
    self.promedio=0
    i=0
    while i<niveles:
     self.promedio=self.promedio + (self.RΣSmax[i]/Rel)/niveles
     if max(self.RΣSmax)>Rel:
      self.promedio=0
     i=i+1
    """self.promedio=self.distorsionmax/Rel/niveles
    if max(self.RΣSmax)>Rel:
     self.promedio=0"""
    return (self.promedio)
class poblacion:
  def __init__(self):
    self.n_ind=int(n_ind)
    self.individuos=[]
    for i in range(n_ind):
     print("""
      INDIVIDUO """ +str(i+1)+ """
```

```
    """)
    individuo_i=valores()
    individuo_i.calculos()
    individuo_i.texto1()
    print(individuo_i.funcion_objt())
    self.individuos.append(individuo_i)
  def evaluacion_poblacion(self):
   self.num=self.n_ind
   self.mejor_individuo=copy.deepcopy(self.individuos[0])
   self.genera=0
   for i in range(0,self.num, 1):
    if abs(self.individuos[i].funcion_objt()-1)<= abs(self.mejor_individuo.funcion_objt()-1):
     self.genera=i+1
     self.mejor_individuo = copy.deepcopy(self.individuos[i])
   self.mejor_puntuacion=self.mejor_individuo.promedio
   self.mejor_valor_variables=self.mejor_individuo.Hsc
   print("MEJOR INDIVIDUO")
   print("------------------")
   print("EFICIENCIA: " + str(self.mejor_puntuacion))
   print("INDIVIDUO: " + str(self.genera))
   self.mejor_individuo.texto2()
   opc= int(input("""¿Quieres ver los resultados de todas las operaciones?:
   1)SI    2)NO
   """))
   if opc==1:
    self.mejor_individuo.texto3()
    if opc==2:
     pass
  def excel (self):
   my_wb = openpyxl.Workbook()
   my_sheet = my_wb.active
   f = 0
   while f < niveles:
     encabezado1 = my_sheet.cell(row= 1, column=1 + f)
     encabezado1.value ="Desplazamiento Total "+ str(f+1)
```

```
                encabezado2 = my_sheet.cell(row=1, column=1 + niveles + f)
                encabezado2.value = "Distorsiones de entrepiso Smax"+str(f+1)
                f = f + 1
            i=0
            while i <self.n_ind:
                f=0
                while f<niveles:
                    c1 = my_sheet.cell(row=i + 2, column=1+f)
                    c1.value = str(self.individuos[i].RΣYmax[f])
                    c2 = my_sheet.cell(row=i + 2, column=1+niveles+f)
                    c2.value = str(self.individuos[i].RΣSmax[f])
                    f=f+1
                i = i + 1
            my_wb.save(r"C:\Users\Alex\Desktop\PYTHON COLUMNAS.xlsx")
        def seleccionar_individuos(self):
            array_calificacion=np.repeat(None, self.n_ind)
            metodo_seleccion="ruleta"
            for i in np.arange(self.n_ind):
                array_calificacion[i]=copy.copy(self.individuos[i].funcion_objt())
            if metodo_seleccion == "ruleta":
                self.probabilidad_seleccion = array_calificacion / np.sum(array_calificacion)
                self.ind_seleccionado = np.random.choice(np.arange(self.n_ind), 2, True,
list(self.probabilidad_seleccion))
            elif metodo_seleccion == "tournament":
                n=2
                self.ind_seleccionado = np.repeat(None, n)
                for i in np.arange(n):
                    candidatos_a = np.random.choice(
                        a=np.arange(self.n_ind),
                        size=2,
                        replace=False
                    )
                    candidatos_b = np.random.choice(
                        a=np.arange(self.n_ind),
                        size=2,
```

```
            replace=False
          )
          if abs(array_calificacion[candidatos_a[0]]-1 )<abs( array_calificacion[candidatos_a[1]]-1):
            ganador_a = candidatos_a[0]
          else:
            ganador_a = candidatos_a[1]
          if abs(array_calificacion[candidatos_b[0]]-1) < abs(array_calificacion[candidatos_b[1]]-1):
            ganador_b = candidatos_b[0]
          else:
            ganador_b = candidatos_b[1]
          if abs(array_calificacion[ganador_a]-1) < abs(array_calificacion[ganador_b]-1):
            ind_final = ganador_a
          else:
            ind_final = ganador_b
          self.ind_seleccionado[i] = ind_final
    def cruzar_individuos(self):
      self.seleccionar_individuos()
      padre_1=copy.deepcopy(self.individuos[self.ind_seleccionado[0]])
      padre_2 = copy.deepcopy(self.individuos[self.ind_seleccionado[1]])
      print("Padre 1: "+str(self.ind_seleccionado[0]+1))
      print("Padre 2: " + str(self.ind_seleccionado[1]+1))
      self.descendencia = copy.deepcopy(padre_1)
      padre_1_bin_H=[]
      padre_2_bin_H=[]
      #x1=[]
      #num_bits1=[]
      #padre_1_bin_A = []
      #padre_2_bin_A = []
      x2 = []
      num_bits2 = []
      i = 0
      while i < niveles:
        if i==0:
          #x1.append(Hst[i] - Ast[i])
          x2.append(Ast[i]*4  -Ast[i])
```

```
            else:
                #x1.append(max(padre_1.Asc[i-1], padre_2.Asc[i-1])-Ast[i])
                x2.append(max(padre_1.Hsc[i-1],padre_2.Hsc[i-1])-Ast[i])
            #x2.append(max(abs(padre_1.Hsc[i] - padre_1.limites_inf2), abs(padre_2.Hsc[i] - padre_2.limites_inf2)))
            j = 0
            aux = 1
            """while 2 ** j <= x1[i] * 100:
                j = j + 1
                aux = j
            num_bits1.append(aux)
            j=0"""
            while 2 ** j <= x2[i] * 100:
                j = j + 1
                aux = j
            num_bits2.append(aux)
            #print("X1:   "+str(x1[i]))
            #print("X2:   " + str(x2[i]))
            #self.descendencia.num_bits1=num_bits1
            #self.descendencia.x1=x1



            #self.descendencia.num_bits2 = num_bits2
            #self.descendencia.x2 = x2
            #print("ASC p1")
            #padre_1_bin_A.append(padre_1.deci_a_bin(Asc[i], padre_1.limites_inf1, x1[i],num_bits1[i]))
            #print("ASC p2")
            #padre_2_bin_A.append(padre_2.deci_a_bin(Asc[i], padre_2.limites_inf1,x1[i],num_bits1[i]))
            #print("HSC p1")
            padre_1_bin_H.append(padre_1.deci_a_bin(padre_1.Hsc[i], Ast[i], x2[i],num_bits2[i]))
            #print("HSC p2")
            padre_2_bin_H.append(padre_2.deci_a_bin(padre_2.Hsc[i],Ast[i] , x2[i],num_bits2[i]))
            #print("Peralte de columna del padre 1, nivel  "+str(i+1)+" :  "+str(padre_1.Hsc[i])+"    numero en binario:  "+str(padre_1_bin_H[i]))
            #print("Peralte de columna del padre 2, nivel   "+str(i+1)+" :  " + str(padre_2.Hsc[i]) + "    numero en binario:  " + str( padre_2_bin_H[i]))
```

```
            i = i + 1
            #self.descendencia.Asc = np.repeat(None, niveles)
            self.descendencia.Hsc = np.repeat(None, niveles)
            i = 0
            texc = ''
            tt = ''
            probabilidad_de_mutacion=0.05
            mutante1 = np.random.choice([True, False], 1, True,[probabilidad_de_mutacion, 1 - probabilidad_de_mutacion])
            mn1 = int(random.randint(1, niveles) - 1)
            mn2 = int(random.randint(1, niveles) - 1)
            """while i < niveles:
              n = int(random.randint(1, num_bits1[i]) - 1)
              n2 = np.random.choice([1, 0], 1, True, [0.5, 0.5])
              if n2 == 1:
                tt = padre_2_bin_A[i][:n] + padre_1_bin_A[i][n:]
                if mutante1 == True and mn1 ==i:
                  posicion_mutada=random.randint(1,num_bits1[i])
                  x = tt[:posicion_mutada - 1]
                  inter = tt[posicion_mutada - 1:posicion_mutada]
                  y = tt[posicion_mutada:]
                  if inter == "0":
                    inter = "1"
                  else:
                    inter = "0"
                    tt= x + inter + y
                self.descendencia.Asc[i] = round((int(tt, 2) * x1[i] / ((2 ** num_bits1[i]) - 1) + self.descendencia.limites_inf1), 2)
              else:
                tt = padre_1_bin_A[i][:n] + padre_2_bin_A[i][n:]
                if mutante1 == True and mn1 ==i:
                  posicion_mutada=random.randint(1,num_bits1[i])
                  x = tt[:posicion_mutada - 1]
                  inter = tt[posicion_mutada - 1:posicion_mutada]
                  y = tt[posicion_mutada:]
                  if inter == "0":
```

```
                    inter = "1"
                   else:
                    inter = "0"
                    tt= x + inter + y
                  self.descendencia.Asc[i] = round((int(tt, 2) * x1[i] / ((2 ** num_bits1[i]) - 1) +
self.descendencia.limites_inf1), 2)
                 texc = texc + " ------" + str(n) + " " + str(n2) + " " + tt
                 if i != 0:
                   if self.descendencia.Asc[i] > self.descendencia.Asc[i - 1]:
                     self.descendencia.Asc[i] = self.descendencia.Asc[i - 1]
                 print("Nueva base del nivel " + str(i + 1) + " :  " + str(
                   self.descendencia.Asc[i]) + ",  numero binario:   " + tt)
                 i = i + 1"""
               i=0
               texc=''
               tt=''
               while i < niveles:
                n = int(random.randint(1, num_bits2[i]) - 1)
                n2 = np.random.choice([1, 0], 1, True, [0.5, 0.5])
                if n2 == 1:
                 tt = padre_2_bin_H[i][:n] + padre_1_bin_H[i][n:]
                 if mutante1 == True and mn2 ==i:
                  posicion_mutada=random.randint(1,num_bits2[i])
                  x = tt[:posicion_mutada - 1]
                  inter = tt[posicion_mutada - 1:posicion_mutada]
                  y = tt[posicion_mutada:]
                  if inter == "0":
                   inter = "1"
                  else:
                   inter = "0"
                   tt= x + inter + y
                 if round((int(tt,2)*x2[i]/((2**num_bits2[i])-1) + self.descendencia.limites_inf2),2)<Ast[i]:
                  self.descendencia.Hsc[i] = Ast[i]
                 elif round((int(tt,2)*x2[i]/((2**num_bits2[i])-1) + self.descendencia.limites_inf2),2)>Ast[i]*4:
                  self.descendencia.Hsc[i] =Ast[i]*4
```

```
                else:
                    self.descendencia.Hsc[i] = round((int(tt, 2) *x2[i] / ((2 **num_bits2[i]) - 1) + self.descendencia.limites_inf2),2)
                else:
                    tt = padre_1_bin_H[i][:n] + padre_2_bin_H[i][n:]
                    if mutante1 == True and mn2 ==i:
                        posicion_mutada=random.randint(1,num_bits2[i])
                        x = tt[:posicion_mutada - 1]
                        inter = tt[posicion_mutada - 1:posicion_mutada]
                        y = tt[posicion_mutada:]
                        if inter == "0":
                            inter = "1"
                        else:
                            inter = "0"
                            tt= x + inter + y
                    if round((int(tt,2)*x2[i]/((2**num_bits2[i])-1) + self.descendencia.limites_inf2),2)<Ast[i]:
                        self.descendencia.Hsc[i] = Ast[i]
                    elif round((int(tt,2)*x2[i]/((2**num_bits2[i])-1) + self.descendencia.limites_inf2),2)>Ast[i]*4:
                        self.descendencia.Hsc[i] =Ast[i]*4
                    else:
                        self.descendencia.Hsc[i] = round((int(tt, 2) * x2[i] / ((2 ** num_bits2[i]) - 1) + self.descendencia.limites_inf2),2)
                texc = texc + " ------" + str(n) + " " + str(n2) + " " + tt
                if self.descendencia.Hsc[i] < Ast[i]:
                    self.descendencia.Hsc[i] = Ast[i]
                if self.descendencia.Hsc[i] > Ast[i]*4:
                    self.descendencia.Hsc[i] = Ast[i]*4
                if i != 0:
                    if self.descendencia.Hsc[i] > self.descendencia.Hsc[i - 1]:
                        self.descendencia.Hsc[i] = self.descendencia.Hsc[i - 1]
                print("Nuevo peralte del nivel "+str(i+1)+" :  "+str(self.descendencia.Hsc[i])+",  numero binario:   "+tt)
                i = i + 1
            self.descendencia = copy.deepcopy(self.descendencia)
        def crear_generaciones(self):
            n_elitismo = 10
            n_ind_gen = self.n_ind - n_elitismo
```

```
self.ngene = int(n_gen)
probabilidad_de_mutacion = 0.07
self.historia_mejores_individuos = np.repeat(None, self.ngene)
h = 1
while h <= self.ngene:
  individuos_nuevos = []
  print("------------------")
  print("GENERACION: " + str(h))
  if n_elitismo > 0:
    array_fitness = np.repeat(None, self.n_ind)
    for i in np.arange(self.n_ind):
      array_fitness[i] = copy.deepcopy(self.individuos[i].funcion_objt())
    rank = np.flip(np.argsort(array_fitness))
    x = 0
    while x < n_elitismo:
      elite = copy.deepcopy(self.individuos[rank[x]])
      individuos_nuevos.append(elite)
      # self.historia_individuos.append(elite)
      x = x + 1
  i = 1
  while i <= n_ind_gen:
    print("""
          INDIVIDUO """ + str(i) + """
          """)
    self.cruzar_individuos()
    auxi2 = copy.deepcopy(self.descendencia)
    self.descendencia.calculos()
    self.descendencia.texto1()
    print(self.descendencia.funcion_objt())
    """ij = 0
    while ij < 6:
      mutara = np.random.choice([True, False], 1, True,
                [probabilidad_de_mutacion, 1 - probabilidad_de_mutacion])
      mutara2 = np.random.choice([True, False], 1, True,
                 [0.07, 1 - 0.07])
```

```
            if mutara == True:
                auxi = self.desc_valores_mutados
                auxi2.mutaaa = "SI"
                auxi2.valor_variables[ij] = auxi[ij]
            else:
                auxi2.mutaaa = "No"
            ij = ij + 1"""
        # print(auxi2.texto_de_indi()+self.tecx)
        individuos_nuevos.append(auxi2)
        # self.historia_individuos.append(auxi2)
        i = i + 1
      j=0
      while j < n_ind_gen:
        self.individuos[j] = individuos_nuevos[j]
        j=j+1
      h=h+1
Pobl=poblacion()
Pobl.crear_generaciones()
#Pobl.excel()
Pobl.evaluacion_poblacion()
```