DESARROLLO DE UN SOFTWARE LIBRE PARA EL MODELADO DE LA PROPAGACIÓN DE ONDAS ELASTICAS 2D POR EL METODO DE ELEMENTOS FINITOS

PETER ESCAMILLA MAHECHA

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
MAESTRÍA EN SOFTWARE LIBRE
BOGOTÁ, COLOMBIA
2018

DESARROLLO DE UN SOFTWARE LIBRE PARA EL MODELADO DE LA PROPAGACIÓN DE ONDAS ELASTICAS 2D POR EL METODO DE ELEMENTOS FINITOS

PETER ESCAMILLA MAHECHA

Tesis de Maestría

Director
Sebastián Roa Prada, PhD

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
MAESTRÍA EN SOFTWARE LIBRE
BUCARAMANGA, COLOMBIA
2018

NOTA DE ACEPTACIÓN

_____
_____
_____
_____
_____
_____

_____
Firma del presidente del jurado

_____
Firma del jurado

_____
Firma del jurado

Bucaramanga, 01 de septiembre de 2018

RESUMEN

Actualmente existe la necesidad de disminuir los costos en las actividades de prospección de yacimientos petrolíferos. Por ello se requiere de sistemas computacionales que analicen acertadamente los datos del subsuelo en donde posteriormente se van a realizar las perforaciones. Una manera de realizar esto es por medio del modelado por elementos finitos, el cual es más preciso que el modelado por diferencias finitas, que es el método numérico más usado en los softwares actuales.

Como resultado de este trabajo se obtuvo un programa que aplica el método de elementos finitos para la simulación de la propagación de ondas elásticas, que genera como salida los desplazamientos, velocidades y aceleraciones en instantes distintos de tiempo.

Además, este programa es escalable para futuros ajustes, como lo es la aplicación de diferentes condiciones de frontera, diversos tipos de elementos y diferentes tipos de malla. En la versión inicial, todos los elementos de la malla son del mismo material. En futuras versiones se podrá trabajar con mallas que incluyan diferentes materiales en su estructura.

Palabras clave: Modelado, Propagación de ondas elásticas, Sismografía, Elementos finitos, Software libre.

TABLA DE CONTENIDOS

LISTA DE TABLAS

TABLA DE ILUSTRACIONES

TABLA ANEXOS

INTRODUCCIÓN

Realizar perforaciones de pozos petrolíferos es sumamente costoso, especialmente por la alta incertidumbre sobre su presencia y cantidad en un lugar determinado, aumentando el riesgo en las actividades de perforación. Por este motivo, existe una gran cantidad de ayudas pre-exploratorias. Una de estas ayudas es el modelado de la propagación de ondas sísmicas, para que a partir de una toma sismográfica, se pueda determinar con mayor precisión lo que se encuentra bajo la superficie. En el caso particular bajo estudio, se pretende ubicar yacimientos de petróleo.

En éste libro se exponen los motivos que llevaron a usar el método de elementos finitos, junto con la manera en que se implementó éste algoritmo, además de mostrar la implementación realizada durante el desarrollo de este trabajo y las diversas opciones que existían inicialmente para su implementación entre las cuales estaba el lenguaje a usar, el repositorio de código en que se iba a almacenar así como la licencia en que se dejaría el proyecto para su posterior desarrollo.

Se tomó la decisión de dejar el desarrollo como software libre dado que el software utilizado actualmente en la industria realiza el modelado requerido pero es privativo y costoso, o en su defecto, no se permite su uso por empresas que van a hacer un aprovechamiento comercial del mismo.

Con este proyecto se busca acercar el conocimiento y la implementación del algoritmo a cualquier persona tanto para su uso como para realizar mejoras y/o personalizaciones sobre el mismo.

El desarrollo de este documento inicia con una presentación de los conceptos del software libre y las herramientas con las que se fundamentó el desarrollo del aplicativo objeto de esta tesis, luego se hace una presentación sobre el método utilizado, el proceso con el cual se llegó al desarrollo exitoso del aplicativo, luego se presentan los resultados obtenidos y al final del documento se presentan todas las ecuaciones usadas para el desarrollo del algoritmo.

1. MARCO TEÓRICO

Realizar una perforación para un pozo petrolífero es sumamente costoso. Se debe tener certeza de la existencia de petróleo, y de la cantidad de este, para evitar al máximo la posibilidad de pérdidas. Para esto existen diversos métodos para la prospección de yacimientos de petróleo. Hay prospecciones magnetométricas, fotogramétricas aéreas, gravimétricas, sísmicas, radiométricas y estratigráficas (Organización Internacional del Trabajo (OIT), 2001).

La prospección sísmica se realiza a partir del envío de ondas sísmicas, cuya generación se hace por ejemplo con el uso de explosivos, y registrando posteriormente el retorno de estas ondas. Luego, a partir del análisis de estos datos, se generan imágenes tridimensionales de la estructura del subsuelo, esto pasando a través de un proceso asistido por software de computador que se conoce como inversión de datos sísmicos.

Este software que se utiliza para la generación del modelado de las ondas sismicas usa diversos métodos para generar una mejor calidad del modelado, uno de estos métodos es el de diferencias finitas y otro es el de elementos finitos, entre estos dos la diferencia principal es el coste de procesamiento que tienen, la calidad generada y la facilidad de agregar condiciones especiales, siendo el de elementos finitos el más costoso en tiempo pero igual el que genera con mayor precisión el estado final del modelado además de que permite mayor flexibilidad en el momento de agregar condiciones especiales como condiciones de frontera y simulado de materiales y formas complejas.

1.1. SOFTWARE LIBRE

Con respecto al software libre para los proyectos actuales existen una gran cantidad de herramientas que propician el trabajo en conjunto, y la automatización de una gran cantidad de tareas, que además garantizan la seguridad y facilitan operaciones como lo son pasos a producción entre versiones, actualización de textos para que se tengan en diversos idiomas, y

diversificación del proyecto cuando alguien quiere darle una nueva funcionalidad a algo ya existente.

El software libre también se presta para el mejoramiento continuo de las habilidades de quien colabora en este, y da un mayor reconocimiento a quienes lo utilizan, dada la facilidad que tienen estas personas de darse a conocer entre colegas.

1.1.1. Git. (Software Freedom Conservancy, Inc., 2018). Una de las herramientas más usadas hoy en día para los proyectos de software libre es Git, el cual es un sistema de gestión de versiones que permite que varias personas trabajen en un mismo código. Este sistema se encarga de mantener un historial de quien ha hecho que y en caso de ser necesario retornar a un punto anterior al proyecto. También permite la mezcla de cambios realizados por diversas personas en un mismo fragmento sin que se reemplacen entre ellos sino que quede combinado.

1.1.2. Github (GitHub, 2018). Git por sí mismo permite que diversas personas puedan trabajar en un proyecto pero para esto las personas deben conocerse y debe existir cierto grado de confianza y bastante interés para poder trabajar en el mismo proyecto. Entre ellos deben compartirse direcciones IP públicas y una manera de acceder a sus archivos para que se actualicen los cambios a medida que se van haciendo, o tener un servidor común al cual todos se integren para mantener los cambios actualizados. Esto resulta ser complicado y costoso. Por esto existe GitHub, una plataforma que ha facilitado enormemente la colaboración en proyectos libres, pues lo que deja de ser importante es las personas que desarrollan los proyectos, y lo importante son los proyectos en sí mismos, centraliza una gran cantidad de proyectos libres, permite que cualquiera pueda descargar estos código e incluso cualquiera puede enviar propuestas de cambios a cualquier proyecto, y finalmente los dueños de estos proyectos aprobarlos o rechazarlos, manteniendo así un control sobre el proyecto y al mismo tiempo una fácil colaboración con externos.

1.1.3. Licenciamiento software. Con respecto al licenciamiento del software hoy día hay una gran variedad de opciones para todas las necesidades pero se encuentra una principal división,

una es el software privativo donde se restringe el uso y este debe ser adquirido para usarlo e incluso se puede cobrar una suscripción, y el software libre donde este se puede modificar y usar libremente aunque en ocasiones se limita el uso comercial o su modificación si luego de esta no es liberado el código nuevamente.

1.1.3.1. Licenciamiento GNU GPL versión 3 (Free Software Foundation, Inc., 2007). El licenciamiento GNU GPL versión 3 busca que los desarrollos se mantengan libres y que puedan ser utilizados en cualquier fin incluso el uso comercial. Este licenciamiento es bastante popular pero tiene un aspecto negativo cuando se usa para proyectos comerciales y es que restringe sus derivaciones a que deban ser libres también.

1.1.3.2. Licenciamiento MIT (Open Source Initiative and others, 2006). Es el licenciamiento más común en Github (Neil , 2013). Este licenciamiento busca que el código sea libre pero que sus derivaciones puedan ser cerradas. Con esto se le permite a los desarrolladores usar el código en cualquier tipo de proyecto, incluso privativo o más abierto como con un licenciamiento GNU.

1.2. LENGUAJES DE PROGRAMACIÓN (BULL, SMITH, POTTAGE, & FREEMAN, 2001).

1.2.1. C. Es el lenguaje base para el desarrollo de aplicaciones de alto desempeño, se tiene control bastante directo tanto sobre el manejo de la memoria como de los dispositivos a utilizar. Es el lenguaje en el que se desarrollan los sistemas operativos y muchas veces cuando se necesitan proyectos de uso científico se acude a este para el mejor uso posible de la máquina

1.2.1.1. Ventajas

- Alto desempeño
- Buena cantidad de librerías

1.2.1.2. Desventajas

- Dificultad del uso de los aplicativos por la instalación o el compilado de los mismos

- Falta de conocimiento en el lenguaje
- Lenguaje a bajo nivel lo que requiere mayor código para hacer algo

1.2.2. C++. Es la mejora de C en cuanto permite el uso de objetos en su código haciéndolo más organizado y escalable, manteniendo un alto desempeño, y por tanto es usado en numerosos proyectos científicos

1.2.2.1. Ventajas:

- Alto desempeño
- Buena cantidad de librerías
- Uno de los lenguajes más usados por la comunidad de desarrollo

1.2.2.2. Desventajas:

- La instalación suele ser complicada, requiriendo instalación previa de otras librerías

1.2.3. .Net. Es el avance de C++ creado por Microsoft. Los programas desarrollados en este *framework* se ejecutan en una máquina virtual. Además de tener objetos permite el uso de múltiples lenguajes para un mismo programa como lo son Visual Basic, C#, J#, C++ y ASP. Se usa más en proyectos no científicos como por ejemplo los corporativos, donde lo primordial es el desarrollo rápido.

1.2.3.1. Ventajas:

- Múltiples lenguajes de acuerdo a las necesidades de lo que se quiere que haga el aplicativo o el segmento de la aplicación

1.2.3.2. Desventajas:

- El performance es un poco más bajo que C++ para aplicaciones científicas (Pownuk, 2013).
- Los aplicativos funcionan principalmente en el Sistema Operativo Windows y puede ser complicado que corran en otros sistemas operativos, incluso pueden no correr de igual

forma en distintas máquinas Windows o no correr del todo dependiendo del soporte que tenga la máquina.

1.2.4. Java. Lenguaje abierto creado por Sun Microsystems orientado a objetos. Durante mucho tiempo fue la contraparte a C++, siendo prácticamente los únicos dos lenguajes para cualquier proyecto.

1.2.4.1. Ventajas:

- Corre sobre una máquina virtual lo que hace que funcione igual sin importar el sistema operativo
- Alto desempeño
- Una gran parte los desarrolladores a nivel mundial utiliza este lenguaje
- Inmensa cantidad de librerías, la gran mayoría libres
- Librerías libres para solución de sistemas de ecuaciones

1.2.4.2. Desventajas:

- Otros lenguajes pueden hacer más cosas en menos líneas de código, o estar más enfocados en el tema del proyecto el cual es solución de sistemas matriciales

1.2.5. Fortran. Es un lenguaje principalmente hecho para el cálculo numérico y la computación científica, esto lo hace ideal para resolver este tipo de problemas

1.2.5.1. Ventajas:

- Alto desempeño
- Funcionalidades directas de operaciones matriciales y solución de sistemas de ecuaciones

1.2.5.2. Desventajas:

- Pocos desarrolladores aún usan este lenguaje
- La instalación de estos programas puede ser complicada dependiendo del sistema operativo o del hardware de la máquina

1.2.6. Ruby. Lenguaje libre orientado a objetos, la arquitectura del código es MVC

1.2.6.1. Ventajas:

- Está más enfocado en la facilidad del desarrollo que en el comportamiento de la máquina lo que hace el código más entendible y con menos sorpresas

1.2.6.2. Desventajas:

- Un menor desempeño que C++, C o Java para este tipo de proyectos

1.2.7. Python. Lenguaje interpretado enfocado en ser fácilmente legible

1.2.7.1. Ventajas:

- Es multiparadigma, lo que permite una arquitectura orientada a objetos, imperativa o funcional
- Es interpretado lo que evita la compilación del programa y los cambios pueden ser implementados más rápida y fácilmente
- Una buena base de desarrolladores

1.2.7.2. Desventajas:

- Un menor desempeño que C++, C o Java para este tipo de proyectos

1.3. PROGRAMAS PARA MODELADO DE PROPAGACIÓN DE ONDAS SÍSMICAS POR EL METODO DE ELEMENTOS FINITOS

Actualmente hay una gran cantidad de software que utiliza métodos numéricos para la solución de problemas complejos, tal es el caso del software Ecoelast2D de la empresa colombiana Numérica, el cual usa diferencias finitas (Hortua Bayona & Martinez Abaunza, 2006).
Aunque la gran mayoría de software es privativo, existen algunos casos de software libre (Galeano, Mantilla, Duque, & Mejía, 2007) que implementan el método de elementos finitos aunque de una manera bastante genérica y no al nivel de detalle que se requiere para la prospección de pozos petrolíferos. Se tiene por ejemplo el caso de OpenFEM (SDTools, 2009) que procesa datos utilizando métodos finitos y es libre, pero debe ser usado dentro de Matlab, el cual es privativo y genera costos de licenciamiento y limitaciones en su uso.

Estos softwares por lo general son realizados en Matlab por su capacidad de procesamiento de matrices, aunque cuando es para grandes empresas, y necesita una interfaz gráfica más amable, suele utilizarse la programación en C++ por su mayor flexibilidad y alto rendimiento. Tal es el caso de PETSc-FEM un software realizado en Argentina y su posterior interfaz gráfica siendo esta última sí desarrollada en Java, aunque esta separación de interfaz y núcleo puede hacer que con futuras versiones del núcleo se vuelva incompatible la interfaz.

Hay algunos ejemplos de software muy completos y bastante desarrollados de categoría de software libre como lo es Elmer el cual fue creado en 1995 en cooperación de varias universidades danesas, y su código fue liberado en 2005.

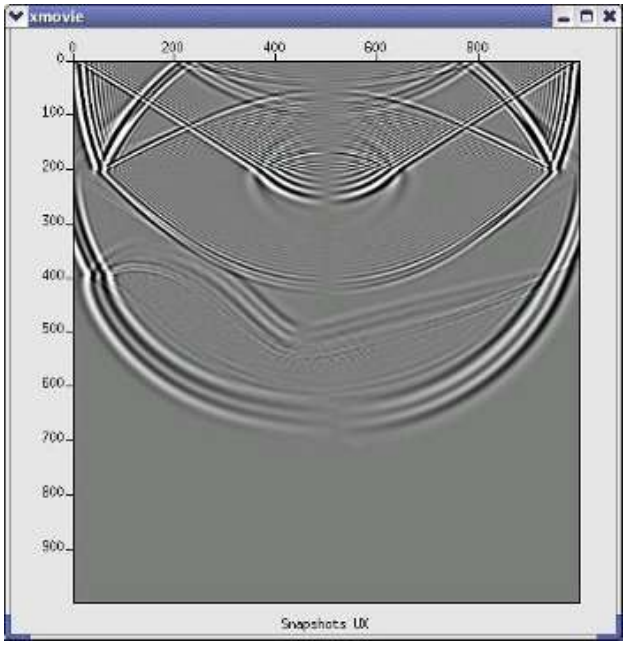
1.3.1. Elmer.

Como se mencionó es un software libre iniciado por una universidad finlandesa (Lyly, Ruokolainen, & Jarvinen, 1999) y actualmente desarrollado en conjunto por varias universidades, empresas, investigadores independientes y personas interesadas en el tema de desarrollo de software para modelos numéricos.

Este software se encuentra desarrollado principalmente en Fortran, inicialmente por varias universidades finlandesas y luego se presentó como código abierto por más universidades, corporaciones y personas individuales.

1.3.2. EcoElast2D. Es un software propiedad de Numerica Ltda., este software realiza la simulación de la propagación de ondas elásticas a partir del método de diferencias finitas. Actualmente se puede conseguir comercialmente por medio de la página http://www.numerica.com.co/. En la Ilustración 1 se aprecia una muestra de los resultados que arroja el software Ecoelast 2D. Su código es cerrado pero es un buen ejemplo de software desarrollado localmente con el mismo propósito del proyecto de esta tesis.

Ilustración 1. ecoElast2D



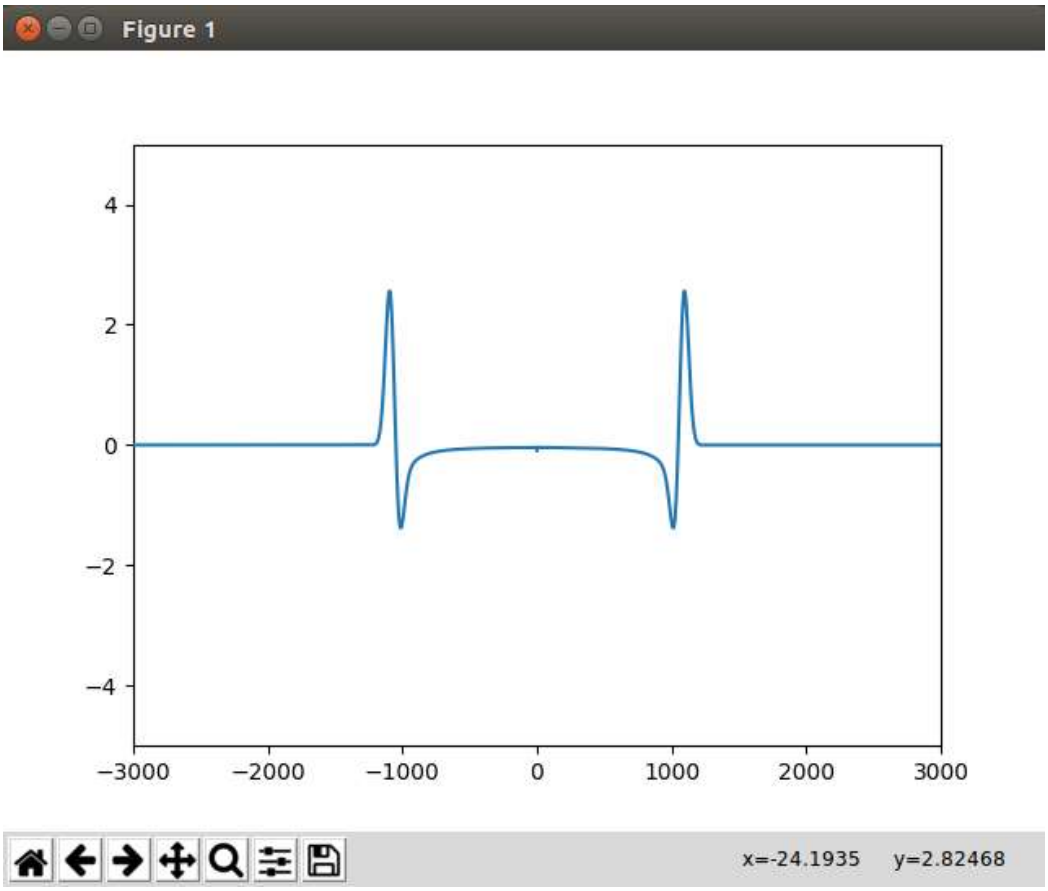
Fuente: (Numérica LTDA., 2018)

1.3.3. 3Dhp90. La universidad de Texas ha desarrollado software para la simulación de propagación de ondas por medio del método de elementos finitos. Este es un software privativo que ha sido desarrollado con el patrocinio de grandes marcas como Shell, Petrobras, ExxonMobil (Castro, Burguener, Paz, & De Bortoli, 2012).

Este software fue desarrollado en Fortran y ha sido objeto de estudio en diversos proyectos a nivel de doctorado en la universidad mencionada.

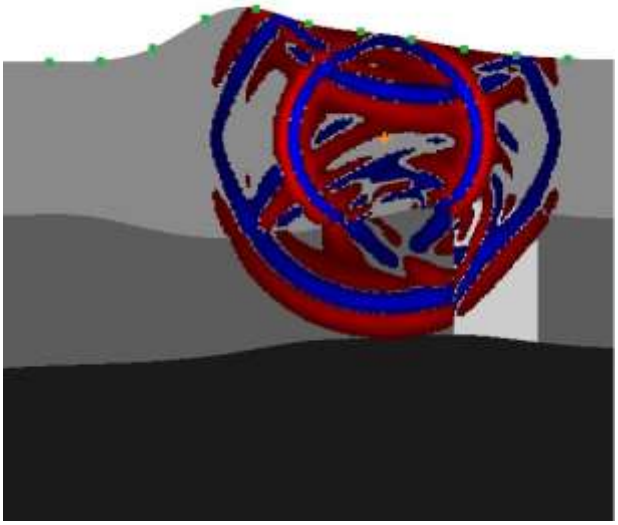
1.3.4. SpecFem2D. Software abierto de simulación de ondas sísmicas escrito en Fortran (Tromp, Komatitsch, & Liu, 2008), este software incluye la implementación de fronteras absorbentes, se mantiene un desarrollo activo aunque no es tan fuerte como el que tiene el software Elmer, sino que este SpecFem2D tiene compatibilidad para simulaciones de petróleo y gas, además tiene un software similar llamado SpecFem3D el cual está siendo más activamente desarrollado, imágenes del programa en funcionamiento pueden ser vistas en la Ilustración 2 y la Ilustración 3.

Ilustración 2. specfem1D



Fuente: Autor

Ilustración 3. specfem2D



Fuente: (SPECFEM2D, 2018)Programas para la generación de la malla

Este software implementa el método de elementos finitos combinado con el método espectral (Hughes T. J., The Finite Element Method, Linear Static and Dynamic Finite Element Analysis, 2012) (Carcione, 2002) y tiene bastantes optimizaciones que permiten que trabaje gran cantidad de datos con menos recursos que la formulación general de elementos finitos (Morgan, Foundations of Wave Theory for Seismic Exploration, 1983) (Komatitsch, 1998).

Para este proyecto en particular la malla que se recibe de entrada se debe generar a partir de un software externo, por lo cual se revisaron diversas opciones.

1.3.5. Gmsh. Es un software que permite la generación, la visualización y el postproceso de una malla con diversos dominios (Geuzaine & Remacle, 2009). Está compuesto por cuatro módulos los que son geometría, mallado, solucionador y post proceso. La especificación de entrada de cada módulo puede ser realizada usando la interfaz gráfica o con archivos de texto en un formato propio de Gmsh.

1.3.6. Cubit. Es un software enfocado en la generación de la malla tanto 2D como 3D (Blacker, Bohnhoff, & Edwards, 1994), también incluye herramientas de suavizado de las formas. Es un software con licenciamiento privativo, que puede ser obtenido gratuitamente dependiendo del uso, en particular para uso académico debe conseguirse la licencia.

1.3.7. GiD. Es un software para de pre y post procesamiento para simulaciones numéricas en ciencia e ingeniería (International Center for Numerical Methods in Engineering (CIMNE)., 2018).

Incluye las siguientes características:

- Modelado Geométrico (CAD)
- Generación de malla
- Definición de datos por analizar
- Transferencia de datos a software de análisis
- Operaciones de post-proceso
- Visualización de resultados

Es un software propietario y su costo para uso académico esta en 550 Euros.

1.4. MODELADO DE LA PROPAGACIÓN DE ONDAS ELÁSTICAS

Las ondas sísmicas pueden tratarse como un problema de ondas elásticas, dicho esto para el problema en particular que es suelos dado que tiene una estructura compleja no es posible realizarlo de una manera directa por lo que se debe realizar con un acercamiento y ahí es donde entran los métodos numéricos.

Como métodos que nos permiten abordar nuestro problema se tienen de tipo directo entre los que están el método de las diferencias finitas, el método pseudoespectral, y el método de elementos finitos, también hay métodos de ecuaciones integrales como el método de Huygens, y métodos asintóticos (Cuervo, 2017).

1.4.1. Métodos directos. Los métodos directos para la simulación de ondas sísmicas se basan en representar el sistema a tratar por medio de una discretización, una de estas maneras de representación es con una malla de puntos se realiza tanto en 1D como 2D y 3D.

1.4.2. Método de elementos finitos. Ha sido ampliamente usado desde que se crearon los primeros computadores, se encarga de aproximar problemas continuos en un dominio por medio de su discretización, dividiendolo en cierta cantidad de puntos también conocidos como nodos que se interconectan en formas definidas como elementos. La sustentación de este método es presentada en el ANEXO A.

1.4.3. Definición del Algoritmo. El diagrama de flujo expuesto en la Ilustración 9 muestra el funcionamiento del algoritmo de elementos finitos.

Discretización del Dominio. El método de elementos finitos funciona subdividiendo el objeto de estudio que llamaremos dominio, por medio de una malla, conformada por una serie de elementos que están compuestos por nodos.

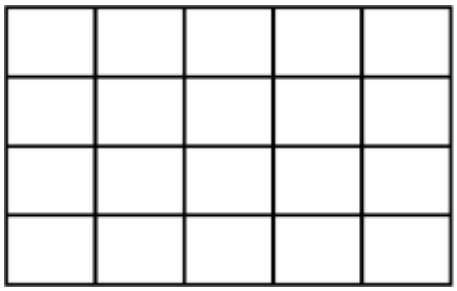
Para el desarrollo del método se inicio asumiendo un dominio con forma rectangular, tal y como se muestra en la Ilustración 4. Suponiendo un perfil de terreno con distancia horizontal y profundidad vertical. La subdivisión del dominio se observa en la Ilustración 5, en donde cada elemento obtenido tiene forma de cuadrado, y cada uno de estos cuadrados se subdivide en dos triángulos como se observa en la

Ilustración 6.

Ilustración 4. Dominio.

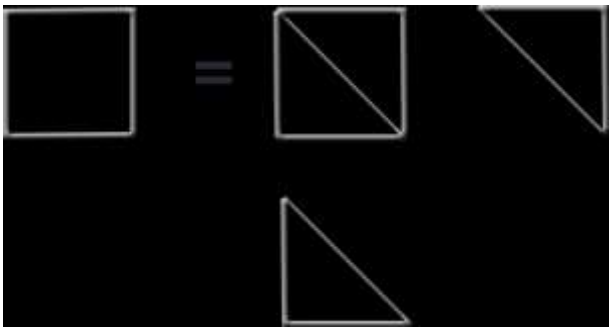
Fuente: Autor

Ilustración 5. Dominio subdividido en elementos.



Fuente: Autor

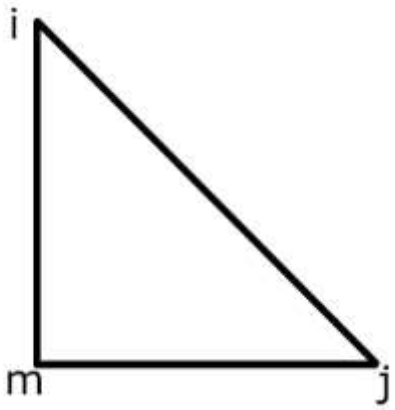
Ilustración 6. Subdivisión del elemento tipo 1 y 2.



Fuente: Autor

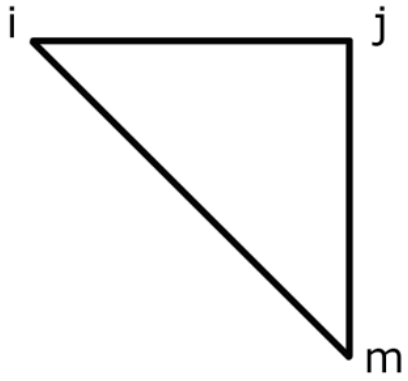
Se seleccionaron los triángulos como elemento tipo para el proceso debido a su simplicidad y a su capacidad de formar figuras cuadradas o rectangulares fácilmente. Cada uno de los triángulos presenta posiciones diferentes, por lo tanto se nombran como elemento tipo 1 y elemento tipo 2 que pueden ser vistos en la Ilustración 7 y la Ilustración 8 respectivamente. Para una versión aún más sencilla del algoritmo se puede ver el ANEXO B donde se describe el método de elementos finitos para una barra.

Ilustración 7. Elemento tipo 1.



Fuente: Autor

Ilustración 8. Elemento tipo 2.



Fuente: Autor

Cada esquina de los elementos, ya sean cuadrados o triangulares, las llamaremos nodos, y vale la pena resaltar que los dos triángulos tipo se relacionan directamente por medio de los nodos de sus diagonales y a su vez los elementos cuadrados se relacionan mediante los nodos compartidos.

Ilustración 9. Diagrama de flujo del algoritmo.

Fuente: Autor

La Ilustración 7 muestra el elemento tipo 1 compuesto de los nodos i, j y m, con la posición de estos nodos podemos generar tanto la matriz de masa como la de rigidez como se muestra en las formulas (50) y (55).

El segundo tipo de elemento mostrado en la Ilustración 8 es un triángulo invertido, el cual al juntarse con el de la Ilustración 8 permite formar una superposición como se muestra en la Ilustración 11. Con los múltiples grupos de estos elementos se puede generar una malla rectangular de cualquier tamaño.

Teniendo como distancia entre los elementos un valor x tendríamos las coordenadas mostradas en la
Ilustración 11.

Antes de iniciar el proceso de creación de las matrices locales de rigidez, se asignan coordenadas de posición para cada nodo teniendo en cuenta sus grados de libertad, de tal forma que para un elemento triangular, compuesto por 3 nodos, en donde cada nodo tiene 2 grados de libertad, correspondientes al desplazamiento en x y y, se obtiene una matriz de dimensiones 6x6, en donde cada campo es nombrado como se observa en la

Ilustración 10. Diagrama de flujo del algoritmo.

	i		j		m	
i	[0,0]	[1,0]	[2,0]	[3,0]	[4,0]	[5,0]
	[0,1]	[1,1]	[2,1]	[3,1]	[4,1]	[5,1]
j	[0,2]	[1,2]	[2,2]	[3,2]	[4,2]	[5,2]
	[0,3]	[1,3]	[2,3]	[3,3]	[4,3]	[5,3]
m	[0,4]	[1,4]	[2,4]	[3,4]	[4,4]	[5,4]
	[0,5]	[1,5]	[2,5]	[3,5]	[4,5]	[5,5]

Fuente: Autor

Luego de conformada la matriz se inicia el proceso matemático para calcular el valor de cada término de la matriz por medio de la ecuación (1) para un elemento triangular de deformación unitaria constante se define la matriz de rigidez. Su aplicación se realiza de forma mecánica dentro del proceso iterativo general, permitiendo observar la generación del valor de cada campo para el elemento tipo 1, y para el elemento tipo 2:

$$[k] = \frac{tE}{4A(1+v)(1-2v)} \times$$

$$\begin{Bmatrix} \beta_i^2 w + \gamma_i^2 r & \beta_i\gamma_i v + \beta_i\gamma_i r & \beta_i\beta_j w + \gamma_i\gamma_j r \\ \beta_i\gamma_i v + \beta_i\gamma_i r & \gamma_i^2 w + \beta_i^2 r & \beta_j\gamma_i v + \beta_i\gamma_j r \\ \beta_i\beta_j w + \gamma_i\gamma_j r & \beta_j\gamma_i v + \beta_i\gamma_j r & \beta_j^2 w + \gamma_j^2 r \\ \beta_i\gamma_j v + \beta_j\gamma_i r & \gamma_i\gamma_j w + \beta_i\beta_j r & \beta_j\gamma_j v + \beta_j\gamma_j r \\ \beta_i\beta_m w + \gamma_i\gamma_m r & \beta_m\gamma_i v + \beta_i\gamma_m r & \beta_j\beta_m w + \gamma_j\gamma_m r \\ \beta_i\gamma_m v + \beta_m\gamma_i r & \gamma_i\gamma_m w + \beta_i\beta_m r & \beta_j\gamma_m v + \gamma_j\beta_m r \end{Bmatrix} \quad (1)$$

$$\begin{Bmatrix} \beta_i\gamma_j v + \beta_j\gamma_i r & \beta_i\beta_m w + \gamma_i\gamma_m r & \beta_i\gamma_m v + \beta_m\gamma_i r \\ \gamma_i\gamma_j w + \beta_i\beta_j r & \beta_m\gamma_i v + \beta_i\gamma_m r & \gamma_i\gamma_m w + \beta_i\beta_m r \\ \beta_j\gamma_j v + \beta_j\gamma_j r & \beta_j\beta_m w + \gamma_j\gamma_m r & \beta_j\gamma_m v + \gamma_j\beta_m r \\ \gamma_j^2 w + \beta_j^2 r & \beta_m\gamma_j v + \beta_j\gamma_m r & \gamma_j\gamma_m w + \beta_j\beta_m r \\ \beta_m\gamma_j v + \beta_j\gamma_m r & \beta_m^2 w + \gamma_m^2 r & \gamma_m\beta_m v + \beta_m\gamma_m r \\ \gamma_j\gamma_m w + \beta_j\beta_m r & \gamma_m\beta_m v + \beta_m\gamma_m r & \gamma_m^2 w + \beta_m^2 r \end{Bmatrix}$$

Donde:

$$\beta_i = y_j - y_m \quad \beta_j = y_m - y_i \quad \beta_m = y_i - y_j$$

$$\gamma_i = x_m - x_j \quad \gamma_j = x_i - x_m \quad \gamma_m = x_j - x_i \quad (2)$$

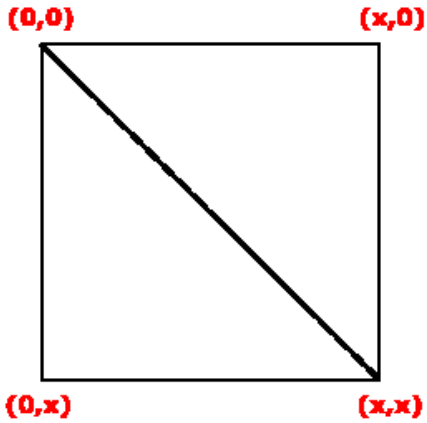
$$w = 1 - v \quad r = \frac{1-2v}{2}$$

A pesar de contar con la posibilidad de ingresar los datos iniciales manualmente en la interfaz de usuario, se resuleve la matriz a modo de ejemplo cambiando el valor de la distancia x por un valor de 30 metros, en este caso los valores para el elemento tipo 1 serían:

$$\beta_i = 0 \quad \beta_j = 30 \quad \beta_m = -30$$

$$\gamma_i = -30 \quad \gamma_j = 0 \quad \gamma_m = 30 \quad (3)$$

Ilustración 11. Coordenadas locales de un grupo de elementos.



Fuente: Autor

Continuando el ejemplo se asigna un espesor de 1 metro, módulo de elasticidad 3x10^9 N/m2, área de 450 m², módulo de Poisson de 0.3, estas características hacen referencia a un suelo de limo arcilloso, de tal forma que la matriz de rigidez para un elemento de la malla sería:

$$[k] = \frac{30 \times 10^9}{4(450)(1+0{,}3)\big(1-2(0{,}3)\big)} \times \begin{bmatrix} 180 & 0 & 0 & -180 & -180 & 180 \\ 0 & 630 & -270 & 0 & -270 & -630 \\ 0 & -270 & 630 & 0 & -630 & 270 \\ -180 & 0 & 0 & 180 & 180 & -180 \\ -180 & -270 & -630 & 180 & 810 & -450 \\ 180 & -630 & 270 & -180 & -450 & 810 \end{bmatrix} \quad (4)$$

La aplicación del software no se limita a un material específico, sino que sus características deben ser ingresadas por el usuario según las condiciones iniciales y del terreno. Sin embargo a modo de ejemplo se desarrolla la ecuación con una densidad de un suelo limo arcilloso (Puppala, Mohammad, & Allen, 1996), con valor de 1680,37 kg/m^3. Al aplicar la densidad de ejemplo a la ecuación (74) se obtiene la ecuación (5) de la matriz de masa para el elemento:

$$[m] = \frac{0{,}73x10^{-3}(450)}{12} \begin{bmatrix} 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 2 \end{bmatrix} \quad (5)$$

En cuanto a la matriz global de rigidez considerando la cantidad de grados de libertad la cual está dada por la ecuación (6) que es dos veces la cantidad de nodos que a su vez se obtiene por medio del procedimiento del ANEXO C, no es viable mostrar una de estas matrices globales pues son matrices de n x n.

$$n = 2xy + 2x + 2y + 2 \quad (6)$$

Donde:
x es la cantidad de elementos a lo ancho (Horizontal)
y es la cantidad de elementos a lo alto (Vertical)
n es la cantidad de elementos
Fuente: Autor

Esta ecuación parte del hecho que el software desarrollado actualmente trabaja solo con grillas rectangulares pero posteriormente puede ser modificado para trabajar con dominios de cualquier forma, igualmente se hace la aclaración que sin importar la forma las matrices globales están en el mismo orden de elementos lo que no permite su visualización en el contenido del presente artículo.

Contando con las matrices locales de cada elemento tipo 1 y 2 es posible generar la matriz global de rigidez, para lo cual se necesita realizar un proceso de conjugación que junte todos los elementos, de tal forma que se obtenga una matriz con el total de grados de libertad del dominio, y que a su vez tenga en cuenta el efecto que puede tener las contribuciones de las matrices locales de diferentes elementos sobre nodos compartidos.

La adición realizada se obtiene procesando elemento a elemento y sobre cada uno conjugando los valores de la matriz local en la matriz global dentro de las iteraciones necesarias para el ancho y alto de los elementos del dominio, basándose en la forma triangular del elemento tipo de acuerdo a la Ilustración 7 y la Ilustración 8 y conjugándolos en la matriz global con las posiciones indicadas en la Ilustración 12 y la Ilustración 13.

Ilustración 12. Conjugación de la matriz de rigidez o de masa local en la matriz rigidez global para el elemento tipo 1

		((c+1)*j+i)*2	((c+1)*j+i)*2+1		((c+1)*(j+1)+i)*2	((c+1)*(j+1)+i)*2+1		((c+1)*(j+1)+(i+1))*2	((c+1)*(j+1)+(i+1))*2+1
((c+1)*j+i)*2		[r1][0][0]	[r1][0][1]		[r1][0][4]	[r1][1][4]		[r1][0][2]	[r1][1][2]
((c+1)*j+i)*2+1		[r1][0][1]	[r1][1][1]		[r1][0][5]	[r1][1][5]		[r1][0][3]	[r1][1][3]
		..							
((c+1)*(j+1)+i)*2		[r1][0][4]	[r1][0][5]		[r1][4][4]	[r1][4][5]		[r1][4][2]	[r1][5][2]
((c+1)*(j+1)+i)*2+1		[r1][1][4]	[r1][1][5]		[r1][4][5]	[r1][5][5]		[r1][4][3]	[r1][5][3]
		..							
((c+1)*(j+1)+(i+1))*2		[r1][0][2]	[r1][0][3]		[r1][4][2]	[r1][4][3]		[r1][2][2]	[r1][2][3]
((c+1)*(j+1)+(i+1))*2+1		[r1][1][2]	[r1][1][3]		[r1][5][2]	[r1][5][3]		[r1][2][3]	[r1][3][3]

Fuente: Autor

Ilustración 13. Conjugación de la matriz de rigidez o de masa local en la matriz rigidez global para el elemento tipo 2

		((c+1)*j+i)*2	((c+1)*j+i)*2+1		((c+1)*j+(i+1))*2	((c+1)*j+(i+1))*2+1		((c+1)*(j+1)+(i+1))*2	((c+1)*(j+1)+(i+1))*2+1
((c+1)*j+i)*2		[r1][0][0]	[r1][0][1]		[r1][0][2]	[r1][1][2]		[r1][0][4]	[r1][1][4]
((c+1)*j+i)*2+1		[r1][0][1]	[r1][1][1]		[r1][0][3]	[r1][1][3]		[r1][0][5]	[r1][1][5]
		..							
((c+1)*j+(i+1))*2		[r1][0][2]	[r1][0][3]		[r1][2][2]	[r1][2][3]		[r1][4][2]	[r1][4][3]
((c+1)*j+(i+1))*2+1		[r1][1][2]	[r1][1][3]		[r1][2][3]	[r1][3][3]		[r1][5][2]	[r1][5][3]
		..							
((c+1)*(j+1)+(i+1))*2		[r1][0][4]	[r1][1][4]		[r1][4][2]	[r1][5][2]		[r1][4][4]	[r1][4][5]
((c+1)*(j+1)+(i+1))*2+1		[r1][0][5]	[r1][1][5]		[r1][4][3]	[r1][5][3]		[r1][4][5]	[r1][5][5]

Fuente: Autor

Donde:

c es la cantidad total de elementos a lo ancho del dominio.

i es la posición horizontal del elemento que cambia a medida que se itera sobre el dominio.

j es la posición vertical del elemento que cambia a medida que se itera sobre el dominio.

A modo de ejemplo se presenta la conjugación de la matriz de rigidez de un elemento tipo 1 con la matriz de rigidez de un elemento tipo 2, las cuales se pueden generar según las

posiciones mostradas en la la Ilustración 12 y la Ilustración 13, la ecuación (7) para el elemento tipo 1 y la (8) para el elemento tipo 2, que luego de la conjugación se ven representadas en la ecuación (9).

$$[k_1] = \begin{bmatrix} a & b & c & d & e & f \\ b & g & h & i & j & k \\ c & h & l & m & n & o \\ d & i & m & p & q & r \\ e & j & n & q & s & t \\ f & k & o & r & t & u \end{bmatrix} \qquad (7)$$

$$[k_2] = \begin{bmatrix} a' & b' & c' & d' & e' & f' \\ b' & g' & h' & i' & j' & k' \\ c' & h' & l' & m' & n' & o' \\ d' & i' & m' & p' & q' & r' \\ e' & j' & n' & q' & s' & t' \\ f' & k' & o' & r' & t' & u' \end{bmatrix} \qquad (8)$$

$$[K] = \begin{bmatrix} a & b & c & d & e & f & 0 & 0 \\ b & g & h & i & j & k & 0 & 0 \\ c & h & l+a' & m+b' & n+c' & o+d' & e' & f' \\ d & i & m+b' & p+g' & q+h' & r+i' & j' & k' \\ e & j & n+c' & q+h' & s+l' & t+m' & n' & o' \\ f & k & o+d' & r+i' & t+m' & u+p' & q' & r' \\ 0 & 0 & e' & j' & n' & q' & s' & t' \\ 0 & 0 & f' & k' & o' & r' & t' & u' \end{bmatrix} \qquad (9)$$

Como se dijo antes, esta conjugación se hace por medio de la ubicación de los nodos locales dentro de la matriz global de rigidez y de masa.

2. MÉTODO DE INVESTIGACIÓN

Para esta investigación se realizó el análisis del método de elementos finitos, se formuló toda la parte teórica incluyendo el análisis de los diversos elementos que iban a conformar el sistema y el análisis de cada uno de los nodos que componen los elementos, se realizó la formulación para hallar la matriz de rigidez y de masa de cada uno de los elementos, como se podría generar la malla de elementos que posteriormente sería la que simularía el sistema.

Se analizó como incluir el procesamiento en el tiempo lo que requería cambiar las fórmulas de elementos finitas básicas para tener en cuenta el desplazamiento y la fuerza en puntos anteriores del tiempo.

También se tuvo que investigar cómo estaba siendo usado en otras aplicaciones de simulación de onda para contrastar la aplicabilidad y las mejores prácticas en una aplicación de este tipo.

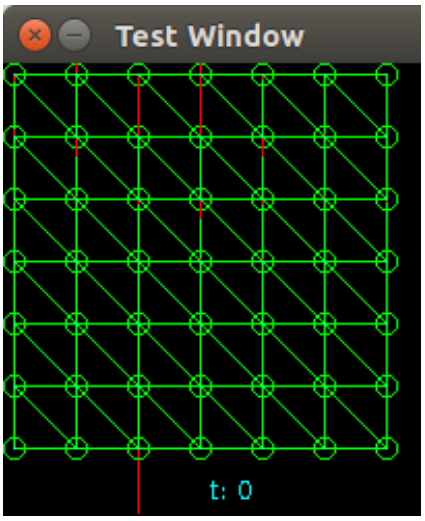
Se planteó que al final de la investigación debía obtenerse una aplicación capaz de simular una onda a partir de una malla inicial y una fuerza ejercida en uno o más puntos.

2.1. SISTEMA DE CARGA DE DATOS

Para el sistema de carga de datos se genera una malla genérica de x unidades de ancho con y unidades de alto, cada unidad con dos elementos triangulares simples que representan un objeto con un material definido, en la carga de datos se encuentran los valores básicos con los que va a correr el modelado, presentados en el capitulo de resultados.

Luego de cargar estos datos se obtiene una malla como la Ilustración 14.

Ilustración 14. Malla cargada



Fuente: Autor

2.2. VECTOR DE FUERZA

Siendo la simulación sobre un sistema inicialmente en estado de reposo es decir con desplazamiento 0 y con velocidad 0, necesita que alguna fuerza externa actúe sobre él para generar algún desplazamiento internamente. Para esto definimos un vector de fuerza el cual viene definido por parejas, fuerza ejercida en el nodo 1 en x, fuerza ejercida en el nodo 1 en y, fuerza ejercida en el nodo 2 en x, etc. La fórmula que describe el vector se puede ver a continuación.

$$\{F\} = \begin{Bmatrix} f_{1x} \\ f_{1y} \\ f_{2x} \\ . \\ . \\ . \\ f_{ny} \end{Bmatrix} \tag{10}$$

2.3. GENERAR ARCHIVO

La generación del archivo se realizó buscando extraer la mayor cantidad de información tanto de lo que entró como de la información de salida, la información generada incluye tanto las

variables de entrada elasticidad, densidad, grosor, área, tiempo, fuerzas aplicadas, como también la matriz de rigidez generada, la matriz de masa y su inversa, y la aceleración, la velocidad y el desplazamiento en cada uno de los puntos.

2.4. DESARROLLO DE LA INTERFAZ GRÁFICA

Se desarrolló una interfaz gráfica que permite la parametrización de los distintos valores que puede tomar el objeto sobre el cual se simulan las ondas, además de esto se hizo una interfaz que muestra los resultados en el sistema general como una interfaz que muestra al detalle los vectores sobre un solo nodo.

2.5. PUBLICACIÓN DEL SOFTWARE EN COMUNIDAD *OPEN SOURCE*

El software se publicó en la gestor de código más reconocido actualmente de software libre el cual es github, como nombre del proyecto se define el nombre inicial que se le habia dado por su origen, EWaveFem https://github.com/pescamillam/EWaveFem en esta ubicación se van a manejar tanto el código como los bugs encontrados y los nuevos features que se quieran realizar como su implementación en modelos 3D.

El software se publica con licencia MIT la cual permite que sea usado como parte de proyectos libres o comerciales, lo que incentivará a que otras personas e incluso empresas puedan colaborar con el código según sus necesidades.

También se creó una lista de correo, la cual suele ser la manera más fácil de comunicarse de manera ordenada en un grupo de este tipo, para esto se creó el correo ewavefem@googlegroups.com donde se espera se realice toda la comunicación formal para el continuo desarrollo de la aplicación, tambien sirve como punto de contacto en caso que se requiera de algún soporte o en caso de que empresas quieran participar o conocer más acerca del proyecto.

3. RESULTADOS

3.1. ALGORITMO MODELACIÓN ONDAS SISMICAS USANDO ELEMENTOS FINITOS

Se desarrolló el algoritmo de elementos finitos mostrado en la Ilustración 9 que permite de manera general obtener el desplazamiento, velocidad y aceleración para cualquier nodo en cualquier tiempo de manera recursiva.

Cada uno de los pasos que se presentan está explicado en el Anexo F donde se hace una explicación de las diferentes ecuaciones que permiten obtener los valores requeridos para esta simulación.

3.2. COMUNIDAD Y REPOSITORIO

Se crea tanto el repositorio público https://github.com/pescamillam/EWaveFem para que cualquier persona pueda usar y modificar el código generado, como un grupo https://groups.google.com/forum/#!forum/ewavefem para tratar el futuro del proyecto y en caso de que alguien tenga dudas tiene acceso a una comunidad de usuarios a la cual acudir.

3.3. ANÁLISIS A MODO COMPARATIVO

Comparando el programa desarrollado con SpecFem2D, se encuentra que SpecFem2D es un software mucho más maduro, con gran cantidad de opciones, que sus archivos generados son archivos de texto e imágenes por cada tiempo que se desee capturar, pero no viene con por ejemplo una animación como actualmente lo tiene EWaveFem que permite una visualización rápida del resultado.

El software se crea en un lenguaje distinto que es ampliamente usado por las comunidades de desarrollo lo que puede lograr que sea fácilmente modificado en el futuro, además de tener una arquitectura extendible para otros tipos de elementos; con solo modificarle la matriz de masa y de rigidez ya sería posible usar otros tipos de malla en la modelación de las ondas.

3.4. DIAGRAMAS DE FLUJO IMPLEMENTACIÓN ALGORITMO

La implementación del algoritmo está directamente escrita en código java, que se encuentra alojado en el repositorio GitHub, que puede encontrarse en https://github.com/pescamillam/EWaveFem/ y su funcionamiento puede ser visto en los siguientes diagramas de flujo, cuyos pasos del algoritmo al código se muestran a continuación:

Funcionamiento general del algoritmo:

Los parámetros que se le solicitan al usuario son los valores utilizados en el proceso, estos valores corresponden a:

Ancho_dominio: indica en metros la distancia a lo ancho de la placa de suelo a simular
Alto_dominio: indica en metros la distancia a lo alto de la placa de suelo a simular
Cantidad_iteraciones: el número de pasos en el tiempo que se van a procesar
Cantidad_ancho: el número de elementos en los que se divide el dominio horizontalmente
Cantidad_alto: el número de elementos en los que se divide el dominio verticalmente
Espesor: la profundidad de la placa del dominio, está dado en metros
Elasticidad: el valor que indica la propiedad elástica del elemento, está dada en psi
Densidad: el valor del peso sobre una unidad de volumen, está dada en Kg/m3.
Área: el espacio ocupado por cada elemento, está dada en m².
Poisson: es la constante elástica que se obtiene cuando al ensanchar un elemento con respecto a lo que se encoje en la dirección perpendicular
Delta_tiempo: la diferencia de tiempo entre cada instante, está dado en segundos
Desplazamiento_inicial: valor en metros que se aplica sobre la matriz con el desplazamiento inicial de los nodos de la primera fila.

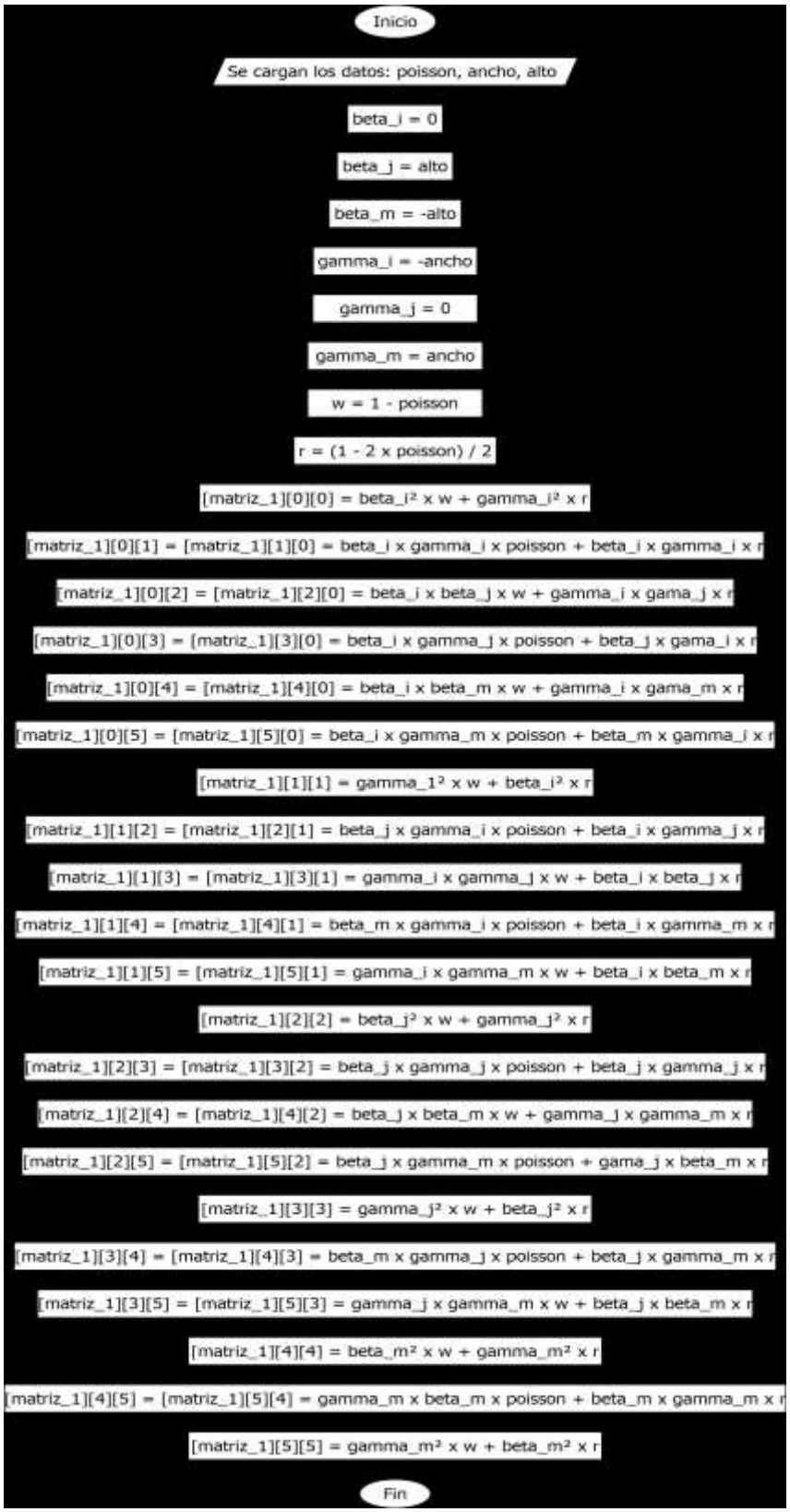
El funcionamiento general puede observarse en la Ilustración 15, la generación de la matriz de rigidez plara el elemento tipo 1 puede verse en la Ilustración 16, la generación de la matriz de rigidez local para el elemento tipo 2 puede verse en la Ilustración 17, la adición de la matriz local de rigidez local para el elemento tipo 1 en la matriz de rigidez global puede verse en la Ilustración 18, Ilustración 19, Ilustración 20 e Ilustración 21, la adición de la matriz de local de

rigidez local para el elemento tipo 1 en la matriz de rigidez global puede verse en la Ilustración 22, Ilustración 23, Ilustración 24 e Ilustración 25, la adición de la matriz de masa local para el elemento tipo 1 en la matriz de masa global puede verse en la Ilustración 26 e Ilustración 27, la adición de la matriz de masa local para el elemento tipo 2 en la matriz de masa global puede verse en la Ilustración 28 e Ilustración 29

Ilustración 15. Diagrama de flujo funcionamiento general del algoritmo

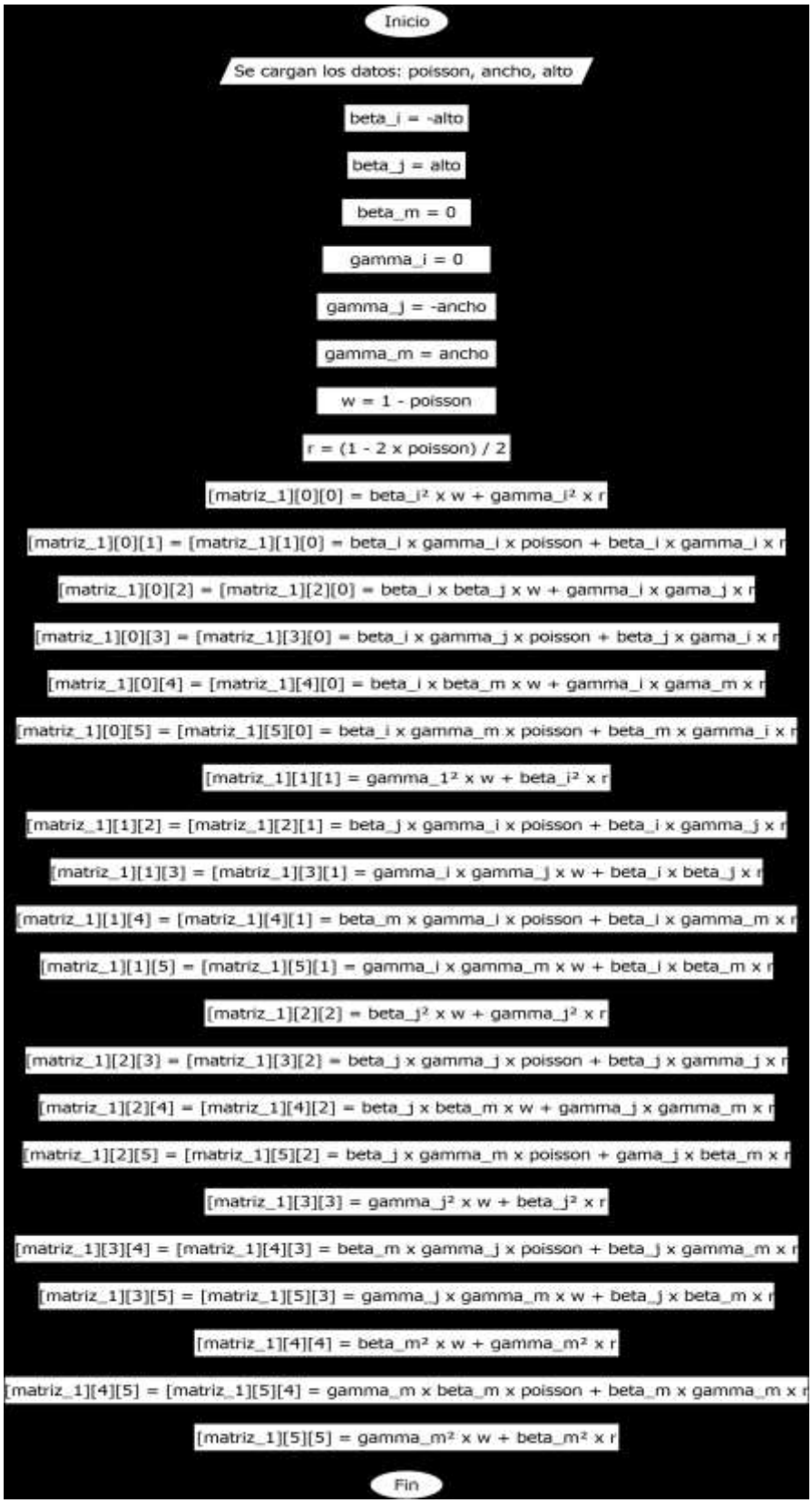
Fuente: Autor

Ilustración 16. Diagrama de flujo generación de la matriz de rigidez local para el elemento tipo 1



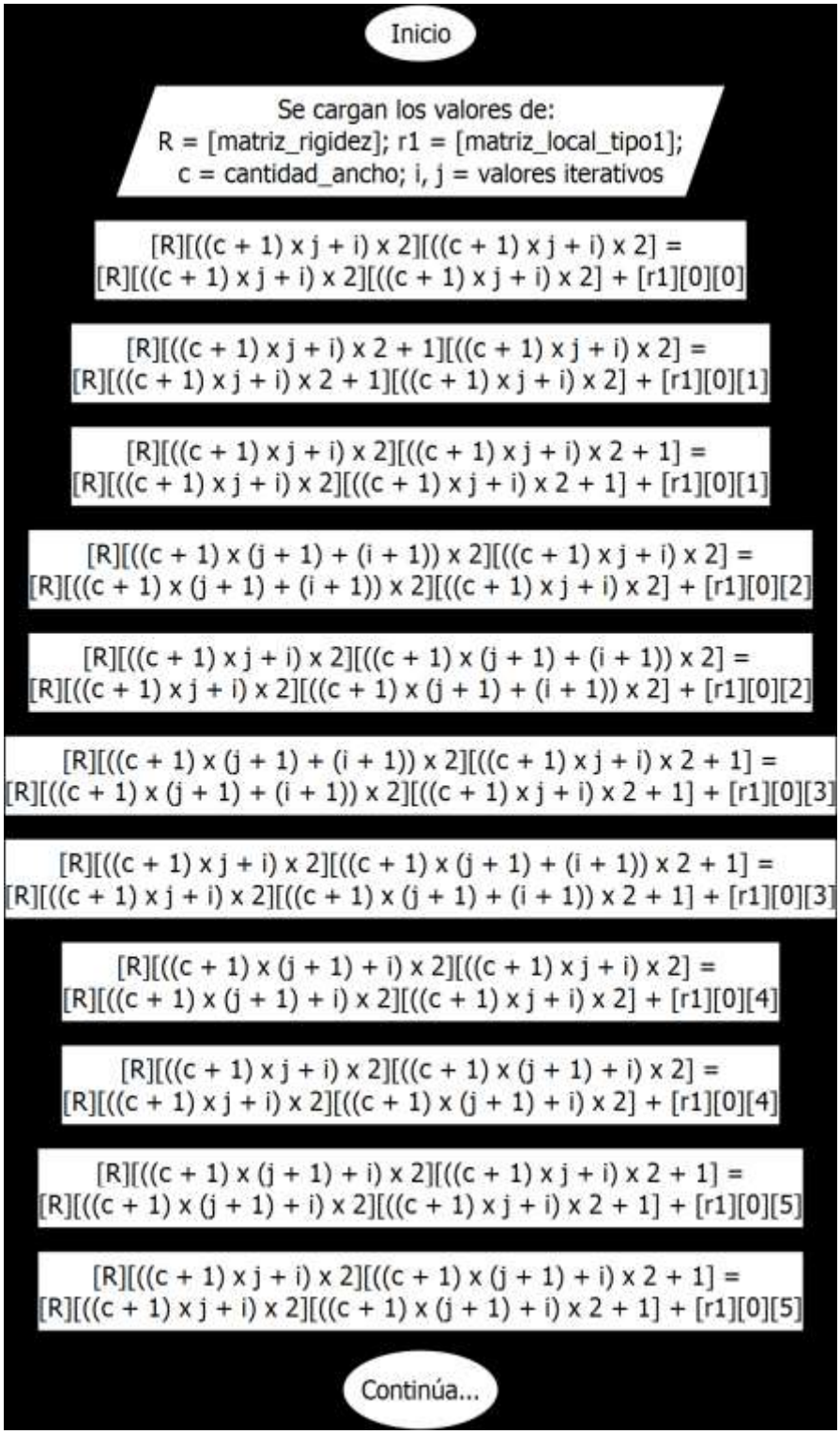
Fuente: Autor

Ilustración 17. Diagrama de flujo generación de la matriz de rigidez local para el elemento tipo 2



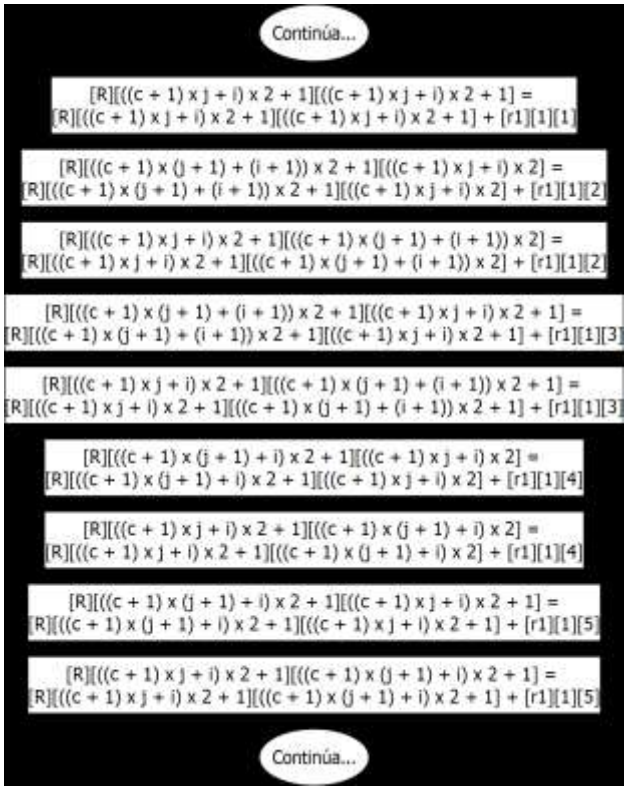
Fuente: Autor

Ilustración 18. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 1 columna 0 en la matriz de rigidez global



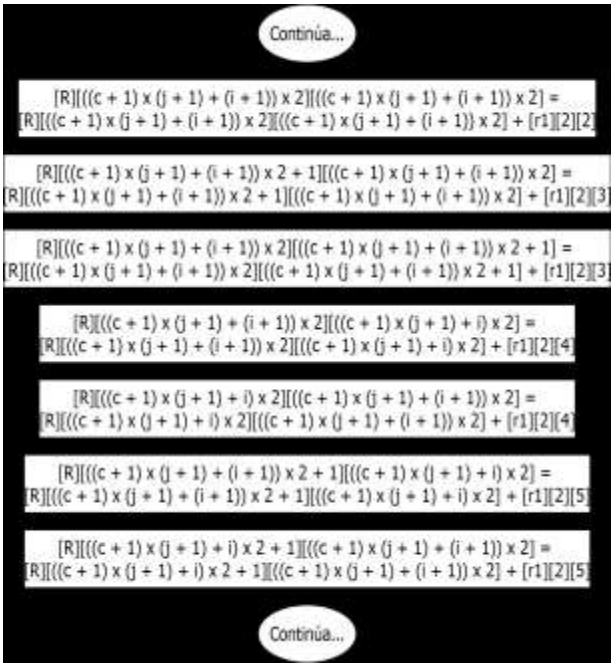
Fuente: Autor

Ilustración 19. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 1 columna 1 en la matriz de rigidez global



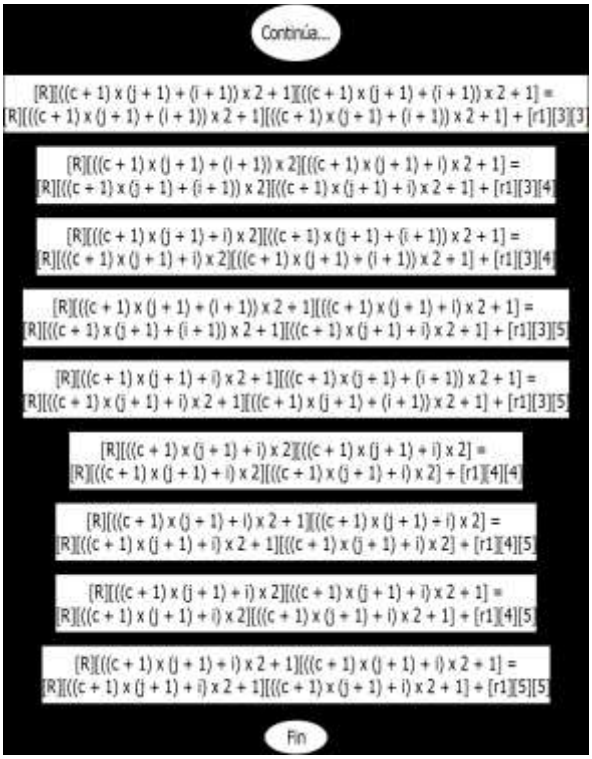
Fuente: Autor

Ilustración 20. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 1 columna 2 en la matriz de rigidez global



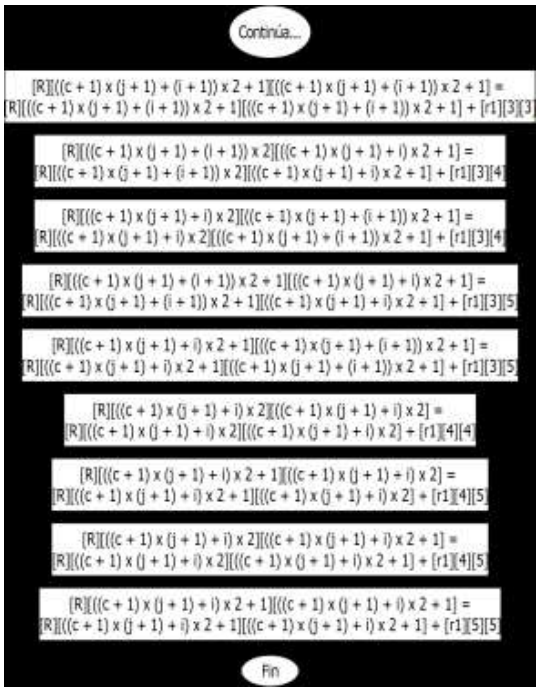
Fuente: Autor

Ilustración 21. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 1 columnas 3, 4, 5 en la matriz de rigidez global



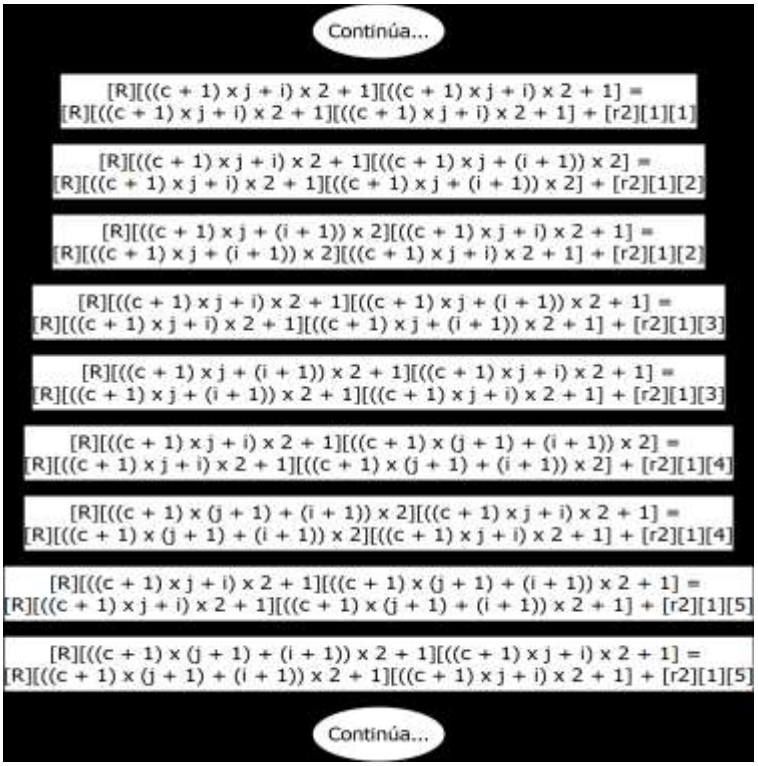
Fuente: Autor

Ilustración 22. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 2 columna 0 en la matriz de rigidez global



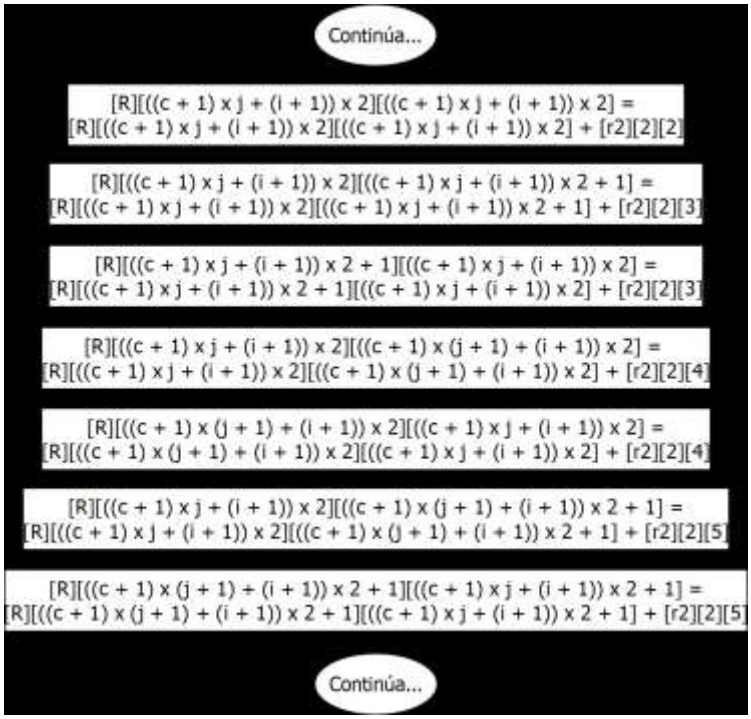
Fuente: Autor

Ilustración 23. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 2 columna 1 en la matriz de rigidez global



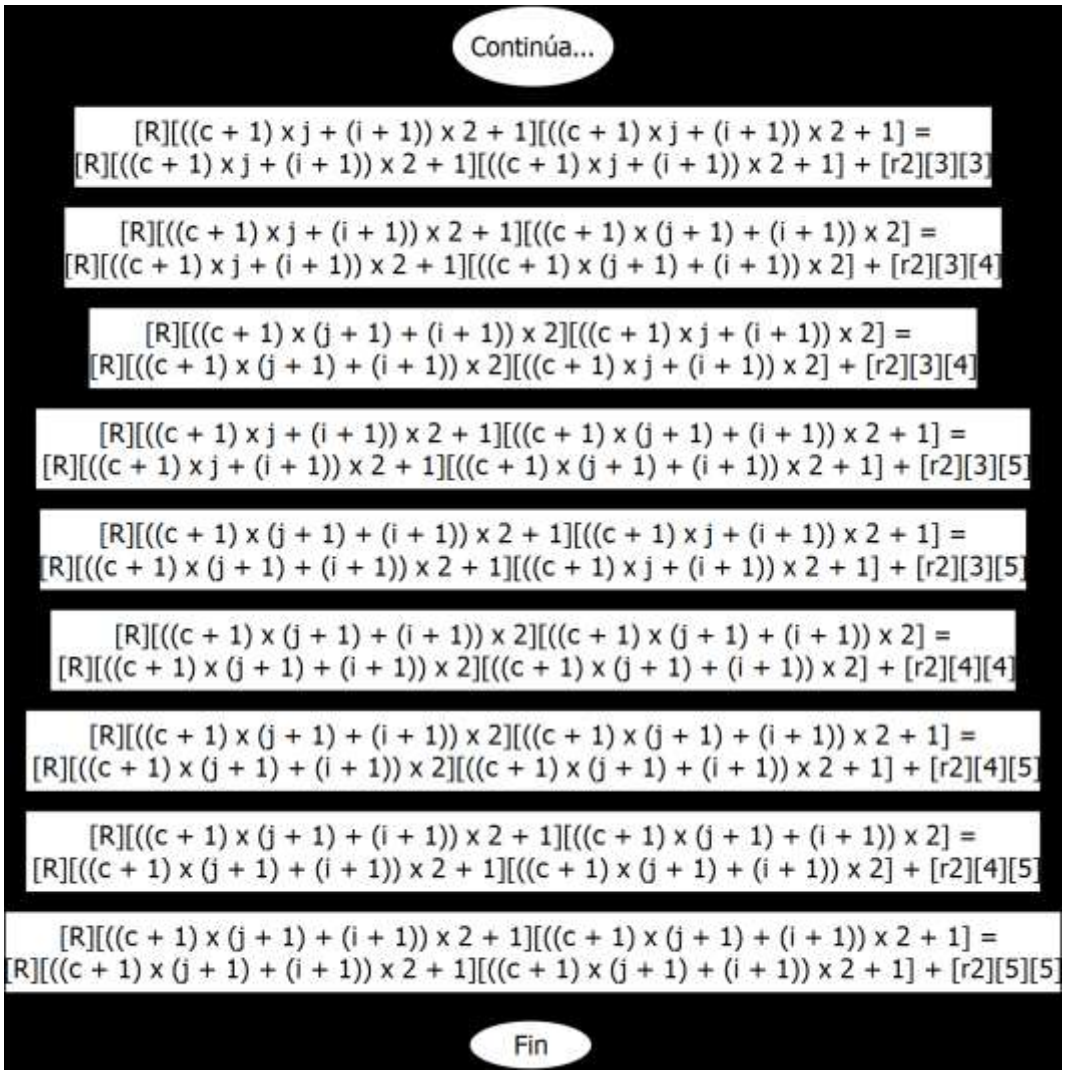
Fuente: Autor

Ilustración 24. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 2 columna 2 en la matriz de rigidez global



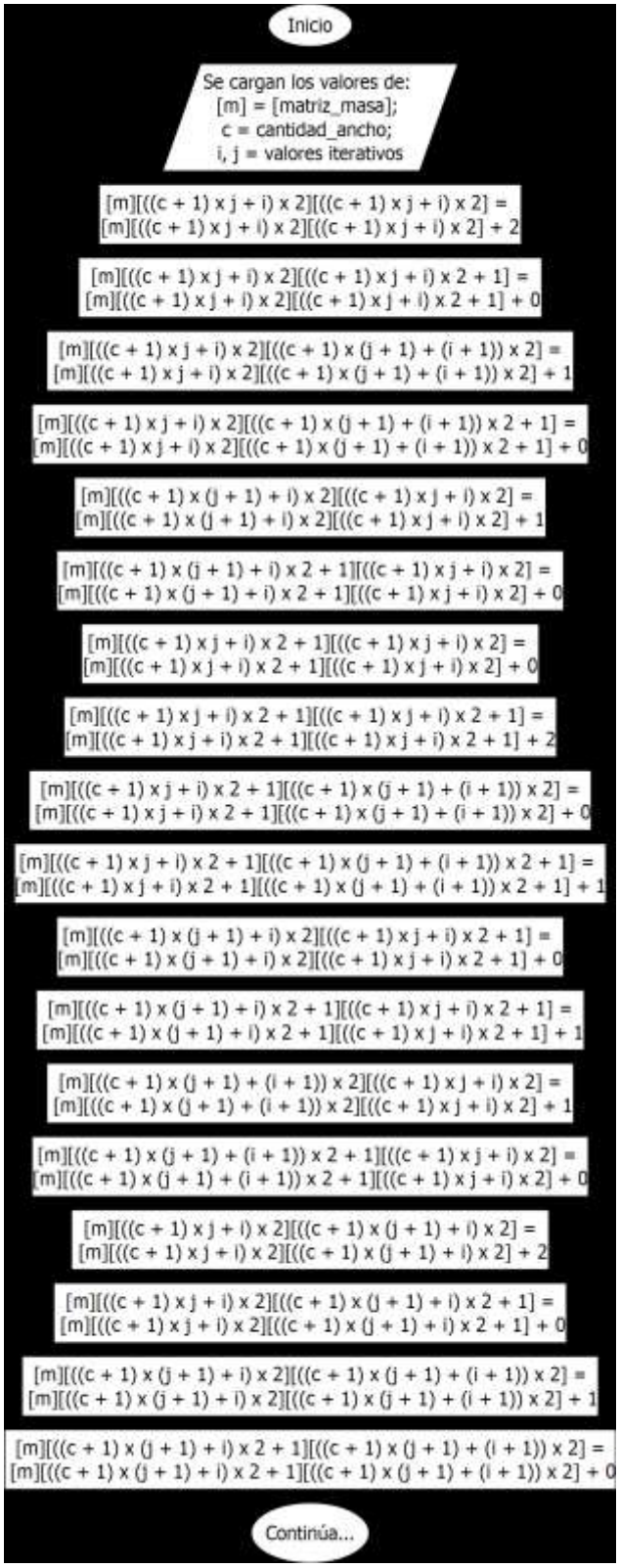
Fuente: Autor

Ilustración 25. Diagrama de flujo adición de la matriz de rigidez local para el elemento tipo 2 columnas 3, 4, 5 en la matriz de rigidez global



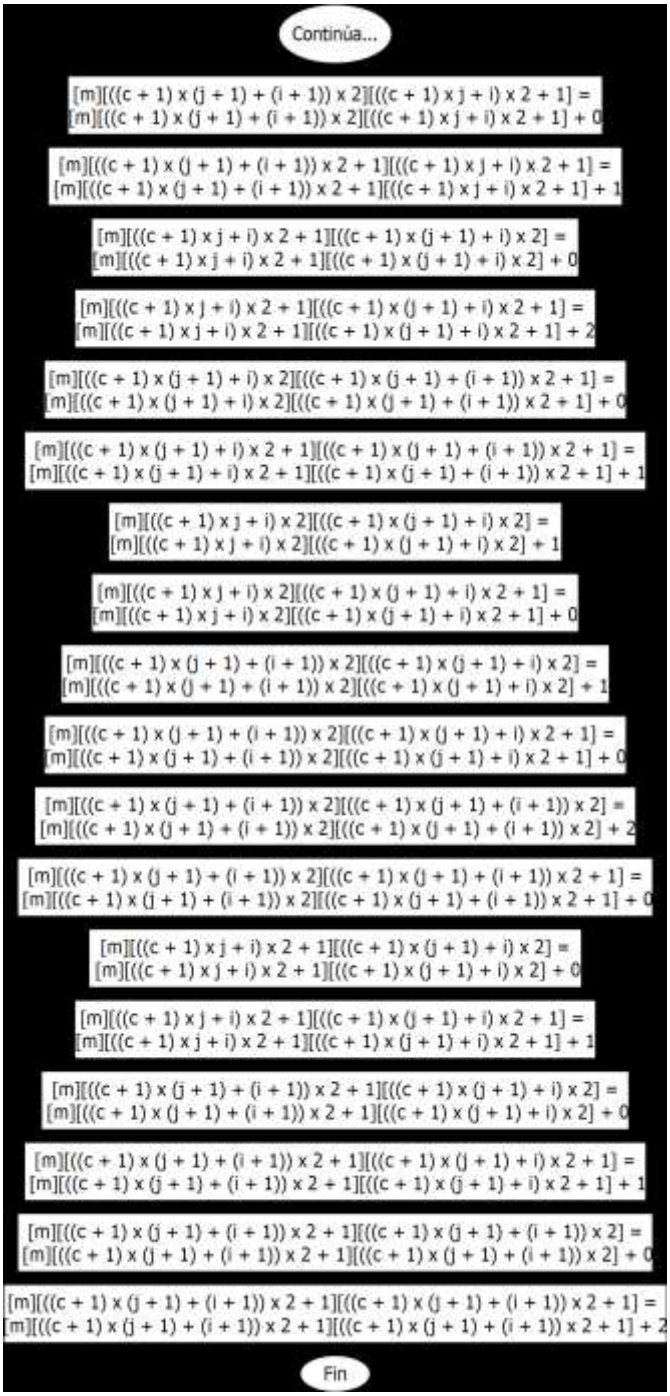
Fuente: Autor

Ilustración 26. Diagrama de flujo adición de la matriz de masa local para el elemento tipo 1 columnas 1, 2, 3 en la matriz de masa global



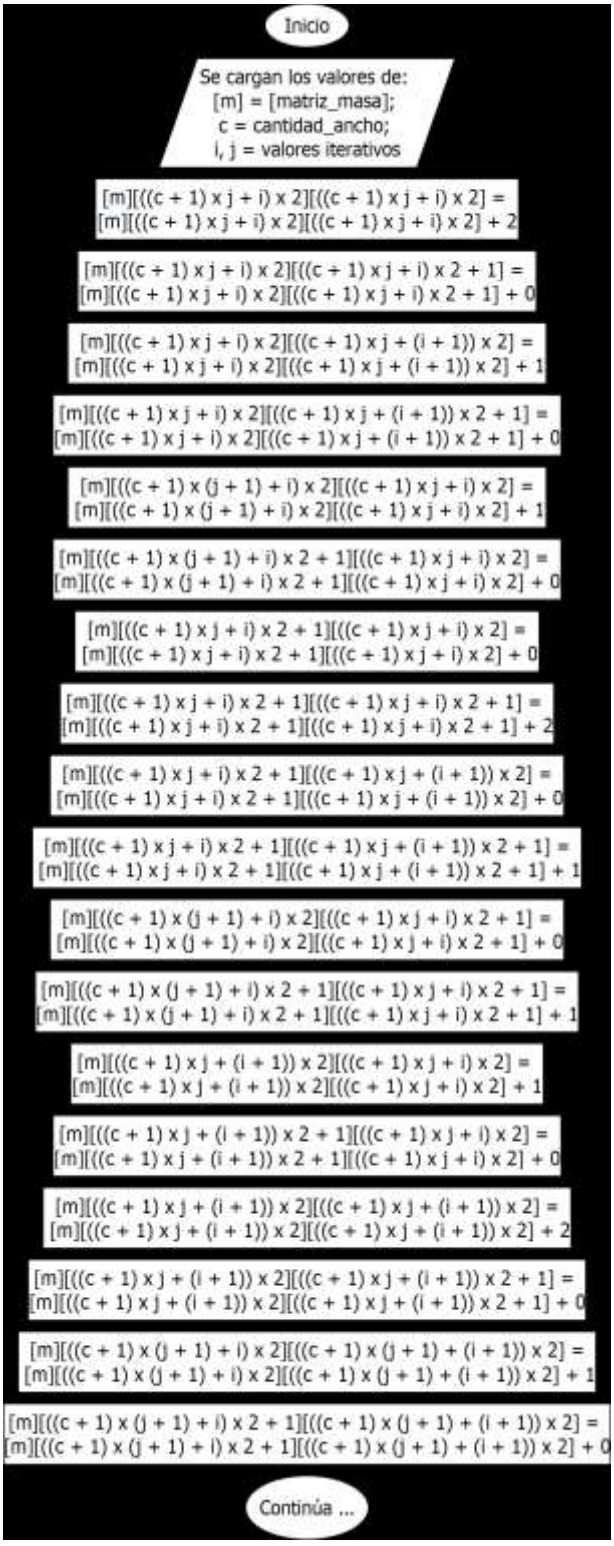
Fuente: Autor

Ilustración 27. Diagrama de flujo adición de la matriz de masa local para el elemento tipo 1 columnas 4, 5, 6 en la matriz de masa global



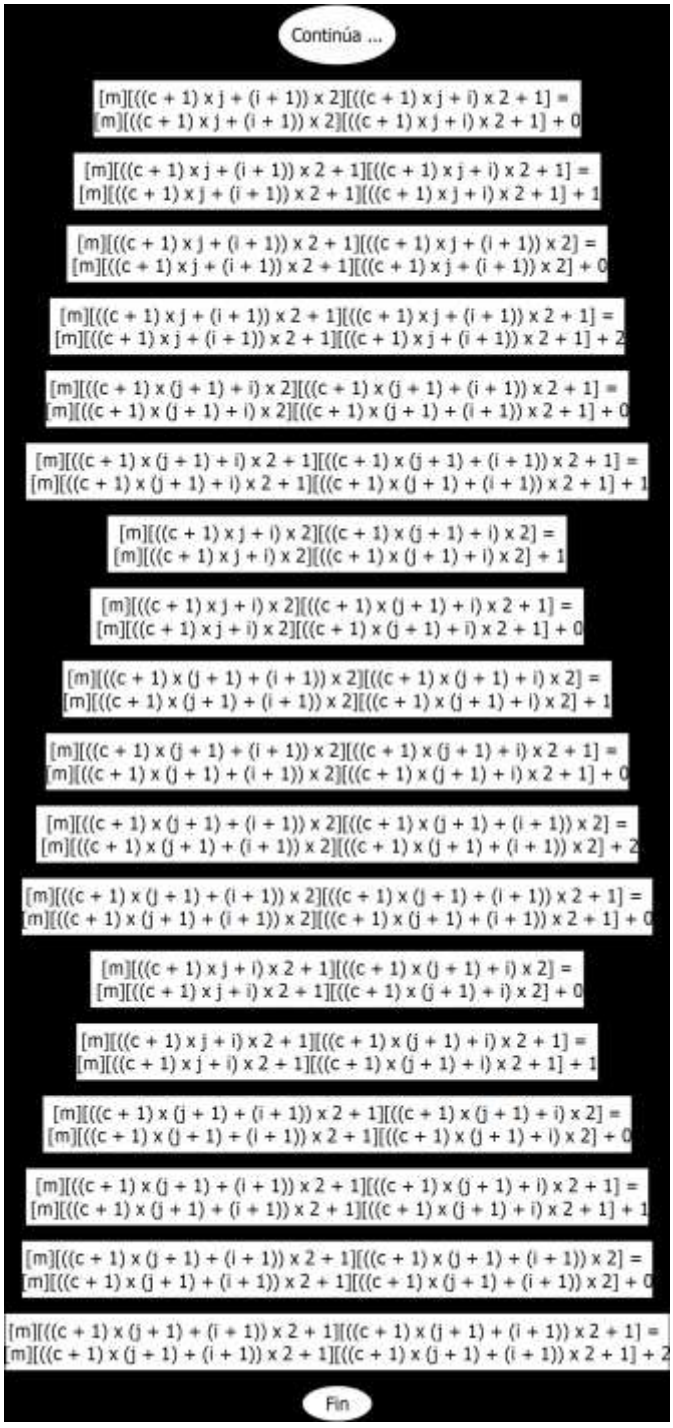
Fuente: Autor

Ilustración 28. Diagrama de flujo adición de la matriz de masa local para el elemento tipo 2 columnas 1, 2, 3 en la matriz de masa global



Fuente: Autor

Ilustración 29. Diagrama de flujo adición de la matriz de masa local para el elemento tipo 2 columnas 4, 5, 6 en la matriz de masa global



Fuente: Autor

3.5. INTERFAZ DE USUARIO GENERADA

Se obtiene un software libre que implementa el algoritmo desarrollado el cual simula una onda en 2D con una malla de cualquier tamaño con cualquier cantidad de elementos y con las propiedades de los elementos definidos libremente por el usuario y con un vector de fuerza cualquiera incluyendo uno que cambia en función del tiempo.

Se crea tanto el repositorio público https://github.com/pescamillam/EWaveFem para que cualquier persona pueda usar y modificar el código generado como un grupo https://groups.google.com/forum/#!forum/ewavefem para tratar el futuro del proyecto y en caso que alguien tenga dudas tiene personas a las cuales acudir.

El software se crea en un lenguaje distinto que es ampliamente usado por las comunidades de desarrollo lo que puede lograr que sea fácilmente modificado en el futuro, además de tener una arquitectura extendible para otros tipos de elementos, con solo modificarle la matriz de masa y de rigidez ya sería posible usar otros tipos de malla en la modelación de las ondas.

Al iniciar el programa la interfaz permite ingresar los valores con los cuales se va a ejecutar el programa.

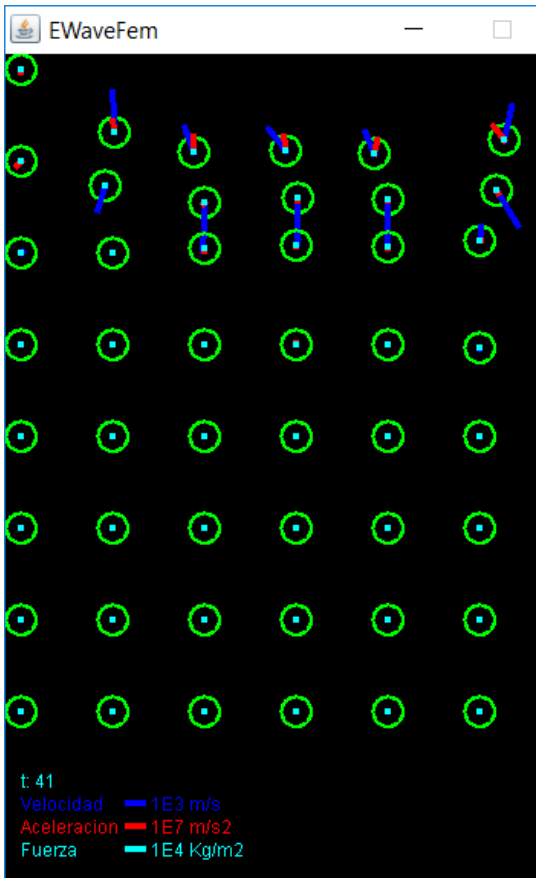
Ilustración 30. Interfaz gráfica configuración parámetros iniciales

Fuente: Autor

La interfaz gráfica cuenta con una malla con todos sus nodos, además de esto se genera una linea representando cada vector que impacta el nodo, es decir, una linea roja para la aceleración, una linea cyan para la fuerza externa aplicada, una linea azul para la velocidad sobre el nodo, además de que se muestra el número de la iteración en el tiempo actual como se puede ver en la Ilustración 31.

La malla se pinta cada 200 milisegundos generando una animación bastante fluida que muestra el movimiento de los nodos a través del tiempo.

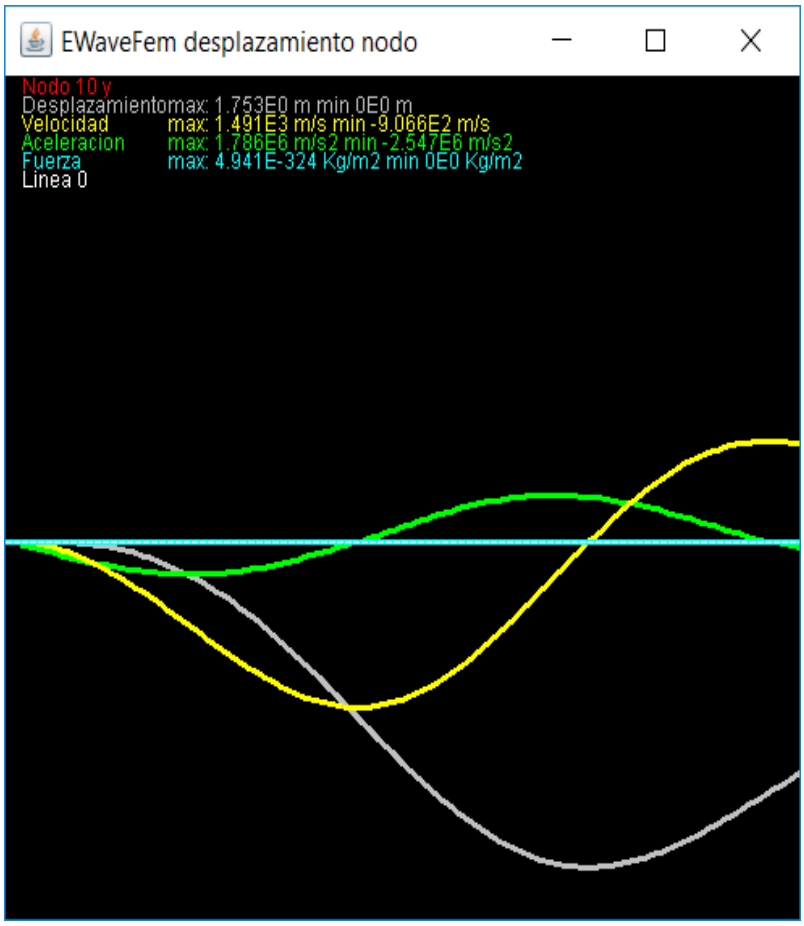
Ilustración 31. Interfaz gráfica con distintos vectores



Fuente: Autor

También cuenta con una interfaz que en vez de mostrar la animación de todos los nodos permite mostrar el comportamiento de un nodo particular durante todo el tiempo, como puede observarse en laIlustración 31 Ilustración 32 existen diferentes colores según el dato graficado de la siguiente forma: color gris: desplazamiento, color verde: aceleración, color amarillo: velocidad, color cyan: fuerza externa

Ilustración 32. Iinterfaz gráfica con datos de un solo nodo



Fuente: Autor

3.6. PROGRAMAS EXISTENTES Y ASPECTOS DIFERENCIADORES

Comparándolo contra SpecFem2D se encuentra que specfem es un software mucho más maduro con gran cantidad de opciones pero difícil de instalar y de usar, que sus archivos generados son archivos de texto e imágenes por cada tiempo que se desee capturar pero no viene con por ejemplo una animación como actualmente lo tiene EWaveFem que permite una visualización rápida del resultado.

Tabla 1. Comparación entre diversas herramientas actuales para simulación por elementos finitos.

	Licencia	Método	Lenguaje	Pros	Cons
SpecFem2D	Libre – GNU / GPL	Métodos espectrales	Fortran	Permite cualquier tipo de forma, incluso tiene una versión para 3D	Está en Fortran el cual es un lenguaje que pocos conocen
EcoElast2D	Comercial	Diferencias finitas	N/A	Es facil de usar, siendo Colombiano esta pensado para el caso de uso local	Es comercial por lo que no se puede modificar libremente y se debe pagar por usarlo
Elmer	Libre – GNU / GPL	Elementos finitos	Fortran	Tiene una variedad de herramientas que permite sea usado de manera independiente para el mallado, el procesamiento o el postprocesamiento	Esta pensado para soluciones generales de diferencias parciales Requiere compilación especifica en la maquina que se va a usar por lo que puede ser complejo de usar para algunos usuarios
3Dhp90	Comercial	Elementos finitos	Fortran	Trabaja con elementos en 3D	Es comercial por lo que no se puede modificar libremente y se debe pagar por usarlo
OpenFem	Libre – GNU / LGPL	Elementos finitos	MatLab	Su uso es bastante sencillo Siendo licencia LGPL los programas que lo usen como librería pueden ser cerrados	Se debe contar con una licencia de matlab para su uso

Fuente: Autor

4. CONCLUSIONES

Una ventaja muy importante en un software para el modelado de propagación de ondas elásticas por el método de elementos finitos y más aún escrito en Java es la posibilidad de modificar su código para ser adaptado según las necesidades requeridas, dado que Java es de los lenguajes más ampliamente usados y el método de elementos finitos permite ajustar fácilmente las condiciones de frontera y los diversos comportamientos de cualquier grupo de nodos, ya sea en un problema específico o en un campo de acción diferente.

Se obtiene un software de modelado de ondas por elementos finitos libre y escalable susceptible de ampliación y mejoras en sus capacidades. Esta primera versión tiene la generación de las matrices de masa y rigidez, recibe todos los parámetros de la definición de los elementos incluyendo el material y sus dimensiones. Este software funciona inicialmente con mallas en 2D y en medios isotrópicos.

Mostrar una animación como parte del resultado obtenido permite apreciar rápidamente los datos para verificar el comportamiento del material simulado donde se pueden ver las ondas en movimiento causado por una fuerza aplicada sobre la fila superior.

Se identificaron los programas actuales de simulación de ondas donde se detectó que la mayoría son privativos y de alto costo, y que aunque hay algunos libres los lenguajes en los que estos están desarrollados no son muy comunes.

5. TRABAJOS FUTUROS

Como trabajos futuros se tienen varias lineas a seguir trabajando sobre este proyecto, las principales serían optimización de procesos, paso a procesamiento en 3D, procesamiento en paralelo, aumentar variedad de elementos.

El caso de la optimización de procesos es que dado que el proyecto buscaba ser un producto mínimo viable, buena parte de los procesos internos pueden mejorarse en gran medida aplicando teoremas matemáticos que disminuyan considerablemente el tiempo de ejecución, también se puede mejorar el manejo de memoria usado en las variables para que sea mejor usado el “garbage collector” y se evite el alto consumo de memoría que actualmente se tiene, además de cambiar algunas variables y disponer de otras como el caso de que al momento de pintar los resultados solo son necesario enteros y no decimales tan exactos como actualmente se tiene.

El caso del paso a procesamiento en 3D es para permitir que el programa no trabaje solo en dos dimensiones como está actualmente sino que permita tres dimensiones esto requiere una revisión desde la formulación matemática y agregar los nuevos tipos de elementos que estarían en tres dimensiones además de que los vectores de desplazamiento, aceleración, velocidad y fuerza tendrían tres grados de libertad por cada nodo.

Dado que el tiempo que tardar en procesar es muy alto es indispensable pensar en ajustar el programa para realizar procesamientos en paralelo, sea aprovechando todos los procesadores de un computador o incluso aprovechando diversos computadores y enviando los resultados por red, esto puede disminuir considerablemente tanto el tiempo que demora un sistema en proceso como los requisitos de las capacidades del computador que va a realizar la simulación.

Y finalmente un cambio ligeramente más sencillo que se puede trabajar es la inclusión de diversos elementos en el programa como figuras cuadradas de nueve nodos los que permitirían una mayor precisión sobre los resultados de la simulación.

BIBLIOGRAFÍA

Bartels, R. H., & Golub, G. H. (1969). The simplex method of linear programming using LU decomposition. *Communications of the ACM, Volume 12, Issue 5*, 266-268 .

Blacker, T. D., Bohnhoff, W. J., & Edwards, T. L. (1994). *CUBIT mesh generation environment. Volume 1: Users manual.* Albuquerque, Estados Unidos: Sandia National Labs.

Bull, J. M., Smith, L. A., Pottage, L., & Freeman, R. (2001). Benchmarking Java against C and Fortran for scientific applications. *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande* (págs. 97-105). New York, Estados Unidos: Association for Computing Machinery Inc.

Carcione, J. M. (2002). Seismic Modeling. *Geophysics*, 1304-1325.

Castro, H. G., Burguener, H., Paz, R. R., & De Bortoli, M. E. (2012). *Desarrollo de una interfaz gráfica para un código abierto de elementos finitos.* Resistencia, Argentina: Universidad Tecnológica Nacional.

Cuervo, O. A. (2017). Solución numérica de la ecuación de onda en medios heterogéneos y aleatorios en 1 dimensión. Boletín de Matemáticas. *Boletín de Matemáticas, 24*(1), 37-55.

Free Software Foundation, Inc. (2007). *GNU Operating System*. Obtenido de https://www.gnu.org/licenses/gpl-3.0.html

Galeano, C., Mantilla, J., Duque, C., & Mejía, M. (2007). Herramientas de software con licencia pública general para el modelado por elementos finitos. *Revista DYNA, Universidad Nacional de Colombia, Volumen 74, Número 153*, 313-324.

Geuzaine, C., & Remacle, J. F. (2009). Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 1-24.

GitHub. (2018). Obtenido de https://github.com/

Hortua Bayona, D. C., & Martinez Abaunza, V. E. (2006). *Diseño e implementación de la versión paralela del programa para el modelamiento de propagación de ondas elásticas 2d en cuerpos sólidos usando el método de diferencias finitas ecoelas2d. (Tesis pregrado).* Bucaramanga, Colombia: Universidad Industrial de Santander UIS.

Hughes, T. J. (1987). *The finite element method, Linear Static and Dynamic Finite Element Analysis.* New Jersey, Estados Unidos: Prentice-Hall Inc.

Hughes, T. J. (1987). *The Finite Element Method, Linear Static and Dynamic Finite Element Analysis.* New Jersey, Estados Unidos: Prentice-Hall Inc.

Hughes, T. J. (2012). *The Finite Element Method, Linear Static and Dynamic Finite Element Analysis.* Courier Corporation.

International Center for Numerical Methods in Engineering (CIMNE). (2018). *GiD - The personal pre and post processor*. Obtenido de https://www.gidhome.com/

Koc University. (20 de 06 de 2016). *Koc University.* Obtenido de http://home.ku.edu.tr/~akabakcioglu/teaching/math506/FEM.pdf

Komatitsch, D. &. (1998). The spectral element method: an efficient tool to simulate the seismic response of 2D and 3D geological structures. 368-392.

Logan, D. L. (2011). *A first course in the finite element method.* Cengage Learning.

Logan, D. L. (2011). *A first course in the finite element method.* Stamford, Estados Unidos: Cengage Learning.

Lyly, M., Ruokolainen, J., & Jarvinen, E. (1999). ELMER--a finite element solver for multiphysics. *CSC-report on scientific computing vol.2000*, 156--159. Obtenido de https://www.csc.fi/web/elmer

Morgan, T. R. (1983). *Foundations of Wave Theory for Seismic Exploration.* Springer.

Morgan, T. R. (1983). *Foundations of Wave Theory for Seismic Exploration.* Heidelberg, Alemania: Springer.

Neil , M. (2013). *Study: Most projects on GitHub not open source licensed.* Obtenido de https://www.theregister.co.uk/2013/04/18/github_licensing_study/

Numérica LTDA. (2018). *Numérica desarrollando ideas*. Obtenido de http://www.numerica.com.co/projects/EcoElast2D/main.htm

Open Source Initiative and others. (2006). The MIT License. Haettu. Obtenido de https://opensource.org/licenses/MIT

Organización Internacional del Trabajo (OIT). (2001). *Enciclopedia de salud y seguridad en el trabajo.* Madrid, España: Chantal Dufresne, BA.

Pownuk, A. M. (2013). *Matrix Vector Multiplication Benchmark in Different Computer Languages.* Texas, Estados Unidos: Digital Commons UTEP.

Puppala, A., Mohammad, L., & Allen, A. (1996). Engineering behavior of lime-treated Louisiana subgrade soil. *Transportation Research Record: Journal of the Transportation Research Board*(1546), 24-31.

SDTools. (2009). OpenFEM. SD Tools Vibration software & consulting. Obtenido de https://www.sdtools.com/openfem/

Software Freedom Conservancy, Inc. (2018). *Git*. Obtenido de https://git-scm.com/

SPECFEM2D. (2018). University of California, Estados Unidos: Computational Infrastructure for Geodynamics .

Tromp, J., Komatitsch, D., & Liu, Q. (2008). Spectral-Element and Adjoint Methods in Seismology. *Communications in Computational Physics, Vol.3, No.1*, 1-32.

ANEXOS

ANEXO A. Definición formula elementos finitos

El método de elementos finitos ha sido ampliamente usado desde que se crearon los primeros computadores, se basa en que un problema complejo del cual no se conocen sus detalles completos puede ser simplificado en muchos problemas en puntos específicos en los cuales se conocen mejor sus propiedades.

La sustentación de este método es la siguiente: (tomado del libro de Hughes (Hughes T. J., The Finite Element Method, Linear Static and Dynamic Finite Element Analysis, 1987).

Se quiere solucionar la siguiente ecuación diferencial:

$$d^2\,u/dx^2 + f = 0 \tag{11}$$

Donde f es un función definida en el intervalo [0, 1] y tenemos los valores iniciales

$$u(1) = g \tag{12}$$

$$-du/dx\,(0) = h \tag{13}$$

Donde g y h son constantes
A partir de estas buscamos la solución como con la ecuación variacional

$$\int_0^1 w_{,x} u_{,x} dx = \int_0^1 wf dx + w(0)h \tag{14}$$

Integrando por partes obtenemos la ecuación

$$0 = \int_0^1 w\big(u_{,xx} + f\big)\,dx + w(0)\big[u_{,x}(0) + h\big] \tag{15}$$

Usando la ecuación de Galerkin tenemos la formula

$$\sum_{B=1}^{n} a(N_A, N_B) d_B = (N_A, f) + N_A(0)h - a(N_A, N_{n+1})g \quad (16)$$

A partir de la cual podemos reescribir

$$K_{AB} = a(N_A, N_B) \quad (17)$$

$$F_A = (N_A, f) + N_A(0)h - a(N_A, N_{n+a})g \quad (18)$$

Con lo que nos quedaría la formula

$\sum_{B=1}^{n} K_{AB} d_B = F_A$, A=1, 2, … , n (19)

La cual se puede simplificar con notación matricial

$$K = [K_{AB}] = \begin{bmatrix} K_{11} & K_{12} & \cdots & K_{1n} \\ K_{21} & K_{22} & \cdots & K_{2n} \\ \vdots & \vdots & & \vdots \\ K_{n1} & K_{n2} & \cdots & K_{nn} \end{bmatrix} \quad (20)$$

$$F = \{F_A\} = \begin{Bmatrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{Bmatrix} \quad (21)$$

$$d = \{d_B\} = \begin{Bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{Bmatrix} \quad (22)$$

Con esto obtenemos la formula general que se expresa de la forma:

$$F = kd \quad (23)$$

Donde d es la matriz de desplazamiento en cada punto que es el dato que nos interesa, k es la matriz de rigidez y F es la fuerza a la que está sometida cada punto.

ANEXO B. Ejemplo del algoritmo en una barra

A continuación se presenta el uso del algoritmo de elementos finitos en una barra representada en la Ilustración 33.

Ilustración 33. Barra dividida en dos elementos

Fuente: Autor

Primero obtenemos la matriz de rigidez y de masa para los nodos de una barra.

$$[m] = \frac{\rho AL}{2}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} \tag{24}$$

Matriz concentrada de masa de un elemento de dos nodos una barra

$$[k] = \frac{AE}{L}\begin{bmatrix}1 & -1\\ -1 & 1\end{bmatrix} \tag{25}$$

Matriz de rigidez de un elemento de dos nodos en un barra

Ahora tomando como ejemplo una barra de 3 nodos tendríamos las siguientes matrices globales.

$$[M] = \frac{\rho AL}{2}\begin{bmatrix}1 & 0 & 0\\ 0 & 2 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{26}$$

Matriz global de masa para una barra de tres nodos

$$[K] = \frac{AE}{L}\begin{bmatrix}1 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 1\end{bmatrix} \tag{27}$$

Matriz global de rigidez para una barra de tres nodos

Teniendo la matriz global de masa se debe calcular su inversa el cual es el proceso más dispendioso de este método

$$[M]^{-1} = \frac{2}{\rho AL}\begin{bmatrix}1 & 0 & 0\\ 0 & 1/2 & 0\\ 0 & 0 & 1\end{bmatrix} \tag{28}$$

Para este ejemplo vamos a utilizar las siguientes variables densidad $\rho = 0{,}00073lb - s^2/\text{in}^4$, área $A = 1\text{in}^2$, elasticidad $E = 30x10^6psi$, longitud $L = 100\text{in}$, desplazamiento inicial$\{d_0\} = 0$, velocidad inicial $\{\dot{d}_0\} = 0$y fuerza en cada tiempo $F = \begin{bmatrix}0\\ 0\\ 1000\end{bmatrix}$

Ahora se puede calcular la aceleración

$$\{\ddot{d}_0\} = \frac{2}{\rho AL}\begin{bmatrix}1 & 0 & 0\\ 0 & 1/2 & 0\\ 0 & 0 & 1\end{bmatrix}\left(\begin{bmatrix}0\\ 0\\ 1000\end{bmatrix} - \frac{AE}{L}\begin{bmatrix}1 & -1 & 0\\ -1 & 2 & -1\\ 0 & -1 & 1\end{bmatrix}\begin{Bmatrix}0\\ 0\\ 1000\end{Bmatrix}\right) \tag{29}$$

$$\{\ddot{d}_0\} = \begin{Bmatrix}0\\ 0\\ 27397\end{Bmatrix} \tag{30}$$

Con estos datos podemos calcular el desplazamiento en el tiempo -1.

$$\{d_{-1}\} = \{0\} - 0{,}25x10^{-3}\{0\} + \frac{(0{,}25x10^{-3})^2}{2}27400\begin{Bmatrix}0\\0\\1\end{Bmatrix} \tag{31}$$

$$\{d_{-1}\} = \begin{Bmatrix}0\\ 0\\ 0{,}856x10^{-3}\end{Bmatrix}\text{in} \tag{32}$$

Ahora podemos calcular el desplazamiento, la aceleración y la velocidad en cada tiempo a continuación la tabla con los resultados para los primeros 10 tiempos.

Tabla 2. Desplazamiento de los nodos de una barra en el tiempo

t	*Desplazamiento*
1	$\{0 \quad 0 \quad 0{,}856x10^{-3}\}$
2	$\{0 \quad 0{,}219x10^{-3} \quad 2{,}98x10^{-3}\}$
3	$\{0{,}113x10^{-3} \quad 1{,}093x10^{-3} \quad 5{,}405x10^{-3}\}$
4	$\{0{,}73x10^{-3} \quad 2{,}823x10^{-3} \quad 7{,}323x10^{-3}\}$
5	$\{2{,}421x10^{-3} \quad 5{,}17x10^{-3} \quad 8{,}642x10^{-3}\}$
6	$\{5{,}525x10^{-3} \quad 7{,}704x10^{-3} \quad 9{,}889x10^{-3}\}$
7	$\{9{,}748x10^{-3} \quad 10{,}239x10^{-3} \quad 11{,}726x10^{-3}\}$
8	$\{14{,}223x10^{-3} \quad 13{,}03x10^{-3} \quad 14{,}511x10^{-3}\}$
9	$\{18{,}085x10^{-3} \quad 16{,}508x10^{-3} \quad 18{,}248x10^{-3}\}$
10	$\{21{,}137x10^{-3} \quad 20{,}838x10^{-3} \quad 22{,}803x10^{-3}\}$

Fuente: Autor

Tabla 3. Velocidad de los nodos de una barra en el tiempo

t	*Velocidad*
1	$\{0 \quad 0{,}44 \quad 5{,}97\}$
2	$\{0{,}226 \quad 2{,}187 \quad 9{,}099\}$
3	$\{1{,}459 \quad 4{,}205 \quad 8{,}677\}$
4	$\{4{,}617 \quad 8{,}154 \quad 6{,}472\}$
5	$\{9{,}592 \quad 9{,}762 \quad 5{,}131\}$
6	$\{14{,}654 \quad 10{,}136 \quad 6{,}169\}$
7	$\{17{,}395 \quad 10{,}653 \quad 9{,}245\}$
8	$\{16{,}673 \quad 12{,}539 \quad 13{,}044\}$
9	$\{13{,}827 \quad 15{,}617 \quad 16{,}583\}$
10	$\{11{,}9 \quad 18{,}483 \quad 19{,}626\}$

Fuente: Autor

Tabla 4. Aceleración de los nodos de una barra en el tiempo

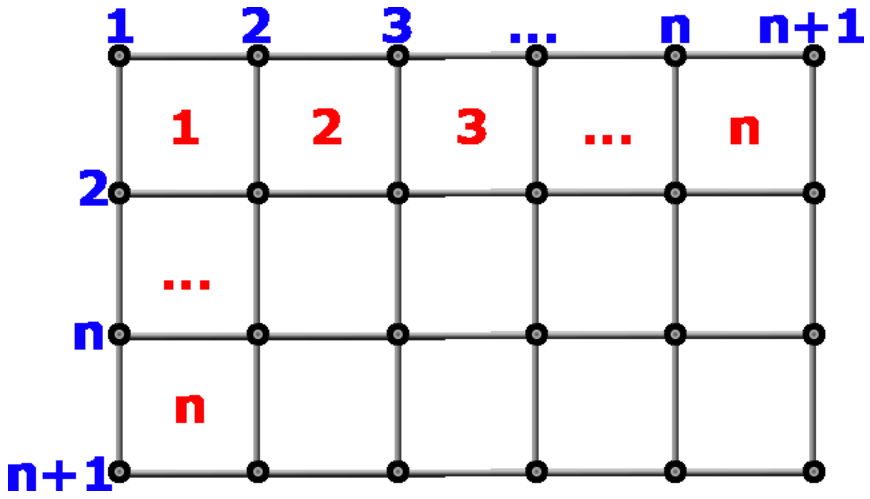
t	*Aceleración*
1	$\{0 \quad 3518 \quad 20360\}$
2	$\{1807 \quad 10459 \quad 4671\}$
3	$\{8059 \quad 13690 \quad -8043\}$
4	$\{17204 \quad 9893 \quad -9594\}$
5	$\{22593 \quad 2968 \quad -1133\}$
6	$\{17901 \quad 30 \quad 9434\}$
7	$\{4029 \quad 4098 \quad 15171\}$
8	$\{-9807 \quad 10992 \quad 15220\}$
9	$\{-12959 \quad 13629 \quad 13097\}$
10	$\{-2453 \quad 9301 \quad 11247\}$

Fuente: Autor

ANEXO C. Calculo número de nodos para ewavefem

Para obtener la cantidad de nodos se utilizan como valores de entrada la cantidad de elementos a lo ancho y la cantidad de elementos a lo alto como se puede ver en la Ilustración 34.

Ilustración 34. Malla con elementos y nodos



Fuente: Autor

Con esto se tiene:

$$n = (w + 1) * (h + 1) \quad (33)$$

Expandiendo esa ecuación se tiene

$$n = wh + h + w + 1 \quad (34)$$

Donde:

n es el número de nodos
w es la cantidad de elementos a lo ancho
h es la cantidad de elementos a lo alto

Clase principal: Application.java

```
package com.pescamillam.fem;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import com.pescamillam.fem.util.UtilWindow;
import org.apache.commons.math3.linear.FieldLUDecomposition;
import org.apache.commons.math3.linear.FieldMatrix;
import org.apache.commons.math3.linear.MatrixUtils;
import org.apache.commons.math3.util.BigReal;

import com.pescamillam.fem.element.Cst;
import com.pescamillam.fem.element.cst.ElementOne;
import com.pescamillam.fem.element.cst.ElementTwo;
import com.pescamillam.fem.model.InputValues;
import com.pescamillam.fem.util.UtilWriter;
import com.pescamillam.fem.window.InitialFormWindow;

import static com.pescamillam.fem.util.Constants.FOUR;
import static com.pescamillam.fem.util.Constants.MINUS_ONE;
import static com.pescamillam.fem.util.Constants.TWELVE;
import static com.pescamillam.fem.util.Constants.TWO;
import static com.pescamillam.fem.util.Constants.DF;
import static org.apache.commons.math3.util.BigReal.ONE;
import static org.apache.commons.math3.util.BigReal.ZERO;

/**
 * Class that executes the application
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class Application {

    /**
     * Method that starts the application
     *
     * @param args not used
     */
    public static void main(String[] args) {
        InitialFormWindow.createFormInitialValues();
    }

    /**
     * Executes the finite element method with given parameters
     *
     * @param input Initial parameters to be used by Finite element method
     */
    @SuppressWarnings("unchecked")
    public static void executeFiniteElementProcess(InputValues input) {

        //Records the initial execution time
```

```
        Long startTime = System.nanoTime();

        //parameters
        //thickness t
        BigReal thickness = new BigReal(input.getThickness());
        //elasticity module E
        BigReal elasticity = new BigReal(input.getElasticity());
        //density p
        BigReal density = new BigReal(input.getDensity());
        //Area A
        BigReal area = new BigReal(input.getArea());
        //poisson ratio v
        BigReal poisson = new BigReal(input.getPoisson());

        //delta time
        BigReal deltaTime = new BigReal(input.getDeltaTime());
        //delta time squared (used to simplify operations
        BigReal deltaTimeSquare = deltaTime.multiply(deltaTime);

        //number of horizontal elements
        Integer numX = new Integer(input.getNumX());
        //number of vertical elements
        Integer numY = new Integer(input.getNumY());
        //number of execution times
        Integer numTimes = new Integer(input.getNumTimes());

        //prints initial parameters
        writeToFile("=== Constants ===");
        writeToFile("area: " + area.bigDecimalValue().toPlainString());
        writeToFile("thickness: " + thickness.bigDecimalValue().toPlainString());
        writeToFile("elasticity: " + elasticity.bigDecimalValue().toPlainString());
        writeToFile("density: " + density.bigDecimalValue().toPlainString());
        writeToFile("poisson: " + poisson.bigDecimalValue().toPlainString());
        writeToFile("\n\n");

        //(1-2v)
        BigReal poisson1m2v = poisson.multiply(TWO).negate().add(ONE);

        //(1+v)
        BigReal poisson1pv = poisson.add(ONE);

        //variables
        //acceleration
        FieldMatrix<BigReal>[] acceleration = new FieldMatrix[numTimes];

        List<Cst> elements = new ArrayList<>();

        //Stiffness matrix
        BigReal[][] stiffnessMatrix = new BigReal[2* numX* numY+2* numX+2* numY+2][2* numX* numY+2* numX+2*
numY+2];

        //mass matrix
        BigReal[][] massMatrix = new BigReal[2* numX* numY+2* numX+2* numY+2][2* numX* numY+2* numX+2*
numY+2];

        for (BigReal[] row : stiffnessMatrix) {
```

```
            Arrays.fill(row, ZERO);
        }

        for (BigReal[] row : massMatrix) {
            Arrays.fill(row, ZERO);
        }


        //Stiffness matrix for local element one
        BigReal[][] localMatrixElemOne = ElementOne.getLocalStiffnessMatrix(poisson);

        //Stiffness matrix for local element two
        BigReal[][] localMatrixElemTwo = ElementTwo.getLocalStiffnessMatrix(poisson);

        //points matrix
        for (int i = 0; i < numX; i++) {
            for (int j = 0; j < numY; j++) {
                //adds local matrix to general stiffness matrix
                ElementOne.appendElementOneToStiffnessMatrix(elements, stiffnessMatrix, localMatrixElemOne, i, j, numX);
                ElementTwo.appendElementTwoToStiffnessMatrix(elements, stiffnessMatrix, localMatrixElemTwo, i, j, numX);

                //adds local mass matrix to general mass matrix
                ElementOne.appendElementOneToMassMatrix(massMatrix, i, j, numX);
                ElementTwo.appendElementTwoToMassMatrix(massMatrix, i, j, numX);
            }
        }

        //Field matrix to easily process the mass matrix
        FieldMatrix<BigReal> massFieldMatrix = MatrixUtils.createFieldMatrix(massMatrix);
        massFieldMatrix = massFieldMatrix.scalarMultiply(density.multiply(thickness).multiply(area).divide(TWELVE));

        writeToFile("=== Starting inversing mass matrix ===");
        Long startInverseTime = System.nanoTime();

        //calculates the inverse mass matrix
        FieldMatrix<BigReal> inverseMassMatrix = new
FieldLUDecomposition<BigReal>(massFieldMatrix).getSolver().getInverse();
        writeToFile("=== Finished inversing mass matrix " + (System.nanoTime() - startInverseTime)/1000000000.0 + "s
===");

        FieldMatrix<BigReal> stiffnessFieldMatrix = MatrixUtils.createFieldMatrix(stiffnessMatrix);

        //multiplies the stiffness matrix with the constant values
        stiffnessFieldMatrix =
stiffnessFieldMatrix.scalarMultiply(thickness.multiply(elasticity).divide(FOUR.multiply(area).multiply(poisson1pv).multip
ly(poisson1m2v)));
        writeToFile("=== stiffness matrix ===");
        printMatrix(stiffnessFieldMatrix);

        //creates the force vector
        BigReal ONE_HUNDRED = new BigReal("30000");
        BigReal[] force0Vector = new BigReal[2 * numX * numY + 2 * numX + 2 * numY + 2];
        Arrays.fill(force0Vector, ZERO);
        force0Vector[numX * 2 + 2 + ((numX * 2) / 2 + 1 + 6)] = ONE_HUNDRED;

        FieldMatrix<BigReal>[] force = new FieldMatrix[numTimes];
```

```
        for (int i = 0; i < numTimes; i++) {

            //the force vector is a sinosoidal function
            force[i] = MatrixUtils.createColumnFieldMatrix(force0Vector)
                    .scalarMultiply(new BigReal(Math.sin(i/8.0)));
        }

        //acceleration = mass inverse * (F0 - [K]{d0}) as {d0} = 0
        //acceleration = mass inverse * (F0)
        writeToFile("=== acceleration 0 ===");

        //calculates acceleration for time 0
        acceleration[0] = inverseMassMatrix.multiply(force[0]);
        printMatrix(acceleration[0]);

        //calculates displacement for time -1
        FieldMatrix<BigReal> displacementM1 = acceleration[0].scalarMultiply(deltaTimeSquare.divide(TWO));

        writeToFile("=== displacement -1 ===");
        printMatrix(displacementM1);

        //calculates displacement for time 0
        BigReal[] displacement0 = new BigReal[2* numX* numY+2* numX+2* numY+2];
        Arrays.fill(displacement0, ZERO);
        FieldMatrix<BigReal>[] displacement = new FieldMatrix[numTimes];
        FieldMatrix<BigReal>[] speed = new FieldMatrix[numTimes];
        displacement[0] = MatrixUtils.createColumnFieldMatrix(displacement0);

        writeToFile("=== displacement 0 ===");
        printMatrix(displacement[0]);

        //calculates displacement for time 1
        displacement[1] = inverseMassMatrix.multiply(
                force[0].scalarMultiply(deltaTimeSquare)

//.add(massFieldMatrix.scalarMultiply(TWO_BIG_REAL).add(stiffnessFieldMatrix.scalarMultiply(deltaTime.multiply(delt
aTime))).multiply(displacementM1))
                .add(massFieldMatrix.multiply(displacementM1).scalarMultiply(MINUS_ONE))
                );

        writeToFile("=== displacement 1 ===");
        printMatrix(displacement[1]);

        for (int i = 2; i < numTimes; i++) {

            //calculates displacement for time i
            displacement[i] = inverseMassMatrix.multiply(force[i-1].scalarMultiply(deltaTimeSquare)
                            .add(massFieldMatrix.scalarMultiply(TWO).add(stiffnessFieldMatrix.scalarMultiply(deltaTimeSquare.
multiply(MINUS_ONE))).multiply(displacement[i-1]))
                            .add(massFieldMatrix.multiply(displacement[i-2]).scalarMultiply(MINUS_ONE))
                            );

            writeToFile("==== displacement " + i + " ====");
            printMatrix(displacement[i]);

            //calculates speed for time i-1
```

```
            speed[i-1] = displacement[i].add(displacement[i-
2].scalarMultiply(MINUS_ONE)).scalarMultiply(ONE.divide(TWO.multiply(deltaTime)));

            writeToFile("=== speed " + (i-1) + " ===");
            printMatrix(speed[i-1]);

            //calculates acceleration for time i
            acceleration[i] =
inverseMassMatrix.multiply(force[i].add(stiffnessFieldMatrix.multiply(displacement[i]).scalarMultiply(MINUS_ONE)));
            writeToFile("=== acceleration " + i + " ===");
            printMatrix(acceleration[i]);

            writeToFile("=== force " + i + " ===");
            printMatrix(force[i]);
        }

        writeToFile("Total time: " + (System.nanoTime() - startTime)/1000000000.0 + "s");
        new Thread(new Runnable() {
            @Override
            public void run() {
                UtilWindow.printElements(elements, displacement, speed, acceleration, force, numX, numY, numTimes);
            }
        }).start();
        new Thread(new Runnable() {
            @Override
            public void run() {
                UtilWindow.printMovingNode(elements, displacement, speed, acceleration, force, numTimes);
            }
        }).start();
    }

    private static void printMatrix(FieldMatrix<BigReal> fieldMatrix) {

        writeToFile("====  ====");
        StringBuilder str = new StringBuilder();
        for (BigReal[] column : fieldMatrix.getData()) {

            for (BigReal unit : column) {
                str.append(DF.format(unit.bigDecimalValue())).append("\t");
            }
            str.append("\n");
        }
        writeToFile(str.toString());

    }

    private static void writeToFile(String string) {
        UtilWriter.writeToFile(string);
        UtilWriter.writeToFile("\n");
    }
}
```

Ventana para ingresar los datos iniciales: InitialFormWindow.java

```
package com.pescamillam.fem.window;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;

import com.pescamillam.fem.Application;
import com.pescamillam.fem.model.InputValues;
import com.pescamillam.fem.util.Constants;


/**
 * Class that shows the initial form to insert the constants to be used
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class InitialFormWindow {

    /** Label for number of iterations */
    private static JLabel numTimesLabel = new JLabel("Número de iteraciones");
    /** Field text for number of iterations */
    private static JTextField numTimesText = new JTextField(String.valueOf(Constants.NUM_TIMES));

    /** Label for number of elements in X */
    private static JLabel numXLabel = new JLabel("Número de elementos a lo ancho");
    /** Field text for number of elements in X */
    private static JTextField numXText = new JTextField(String.valueOf(Constants.NUM_X));

    /** Label for number of elements in Y */
    private static JLabel numYLabel = new JLabel("Número de elementos a lo alto");
    /** Text field for number of elements in Y */
    private static JTextField numYText = new JTextField(String.valueOf(Constants.NUM_Y));

    /** Label for thickness field */
    private static JLabel thicknessLabel = new JLabel("Grosor");
    /** Text Field for thickness value */
    private static JTextField thicknessText = new JTextField(Constants.THICKNESS);

    /** Label for elasticity field */
    private static JLabel elasticityLabel = new JLabel("Elasticidad");
    /** Text field for elasticity value */
    private static JTextField elasticityText = new JTextField(Constants.ELASTICITY);

    /** Label for density field */
    private static JLabel densityLabel = new JLabel("Densidad");
```

```
/** Text field for density value */
private static JTextField densityText = new JTextField(Constants.DENSITY);

/** Label for element area field */
private static JLabel areaLabel = new JLabel("Area");
/** Text field for element area value */
private static JTextField areaText = new JTextField(Constants.AREA);

/** Label for poisson field */
private static JLabel poissonLabel = new JLabel("Poisson");
/** Text field for poisson value */
private static JTextField poissonText = new JTextField(Constants.POISSON);

/** Label for delta time field */
private static JLabel deltaTimeLabel = new JLabel("Delta Time");
/** Text field for delta time value */
private static JTextField deltaTimeText = new JTextField(Constants.DELTA_TIME);

/** Frame object where the form will be shown */
private static JFrame frame;

/** Creates the form with a squared layout */
public static void createFormInitialValues() {
   //Assigns a grid layout of 12 rows and 2 columns
   JPanel p = new JPanel(new GridLayout(12, 2, 10, 10));

   p.add(numTimesLabel);
   p.add(numTimesText);

   p.add(numXLabel);
   p.add(numXText);

   p.add(numYLabel);
   p.add(numYText);

   p.add(thicknessLabel);
   p.add(thicknessText);

   p.add(elasticityLabel);
   p.add(elasticityText);

   p.add(densityLabel);
   p.add(densityText);

   p.add(areaLabel);
   p.add(areaText);

   p.add(poissonLabel);
   p.add(poissonText);

   p.add(deltaTimeLabel);
   p.add(deltaTimeText);

   // button to start the process
   JButton button = new JButton("Ejecutar");
   p.add(button);
```

```
        // assigns the action of making the processing to the button
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                button.setEnabled(false);
                //Starts the processing in background
                new Thread(new ExecutionThread()).start();
                JOptionPane.showMessageDialog(frame, "Ejecutando por favor espere");
                frame.dispose();
            }
        });
        //Create and set up the window.
        frame = new JFrame("Ingresar valores iniciales");

        //Set up the content pane.
        p.setOpaque(true);  //content panes must be opaque
        frame.setContentPane(p);

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    //Thread with the processing part to be executed in background
    private static class ExecutionThread implements Runnable {
        @Override
        public void run() {
            // Starts the processing with the entered values
            Application.executeFiniteElementProcess(new InputValues.Builder()
                    .withArea(areaText.getText())
                    .withThickness(thicknessText.getText())
                    .withDensity(densityText.getText())
                    .withElasticity(elasticityText.getText())
                    .withPoisson(poissonText.getText())
                    .withDeltaTime(deltaTimeText.getText())
                    .withNumX(numXText.getText())
                    .withNumY(numYText.getText())
                    .withNumTimes(numTimesText.getText())
                    .build());
        }
    }
}
```

Ventana que gráfica tanto un nodo individual como la malla completa y su movimiento en el tiempo: UtilWindow.java

```
package com.pescamillam.fem.util;

import com.pescamillam.fem.element.Cst;
import org.apache.commons.math3.linear.FieldMatrix;
import org.apache.commons.math3.util.BigReal;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferStrategy;
```

```java
import java.math.BigDecimal;
import java.util.List;

/**
 * Util class to show result windows
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class UtilWindow {

    /**
     * Shows an image with the displacement, speed, acceleration and force of a node
     *
     * @param elements List with all the elements of the grid
     * @param displacement vector of displacement
     * @param speed vector of speed
     * @param acceleration vector of acceleration
     * @param force vector of force
     * @param numTimes number of iterations
     */
    public static void printMovingNode(List<Cst> elements, FieldMatrix<BigReal>[] displacement,
            FieldMatrix<BigReal>[] speed, FieldMatrix<BigReal>[] acceleration, FieldMatrix<BigReal>[] force, Integer
numTimes) {
        //assigns the title of the window
        final String title = "EWaveFem node displacement";
        //defines the size of the window
        final int width = 1200;
        final int height = 500;

        //creates the window with the given size
        JFrame frame = new JFrame(title);
        frame.setSize(width, height);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setResizable(true);
        frame.setVisible(true);

        //Creating the canvas where the vectors will be drawn
        Canvas canvas = new Canvas();

        canvas.setSize(width, height);
        canvas.setBackground(Color.BLACK);
        canvas.setVisible(true);
        canvas.setFocusable(false);

        frame.add(canvas);

        canvas.createBufferStrategy(3);

        BufferStrategy bufferStrategy;
        Graphics graphics = canvas.getGraphics();
        bufferStrategy = canvas.getBufferStrategy();
        graphics = bufferStrategy.getDrawGraphics();
        //keeps drawing the vectors
        while (true) {
            graphics.setColor(Color.GREEN);
```

```
//iterates for every possible time
for (int i = 0; i < numTimes-1; i++) {
   for (int j = 1; j < 1 + 1; j++) {
      //Draws the displacement
      graphics.setColor(Color.LIGHT_GRAY);
      ((Graphics2D)graphics).setStroke(new BasicStroke(3));
      int y1 = displacement[i].getData()[j][0].bigDecimalValue()
         .multiply(new BigDecimal("100"))
         .intValue();
      int y2 = displacement[i+1].getData()[j][0].bigDecimalValue()
            .multiply(new BigDecimal("100"))
            .intValue();
      graphics.drawLine(i*5, y1+250, (i+1)*5, y2+250);

      //Draws the acceleration
      if (acceleration[i] != null && acceleration[i+1] != null) {
         graphics.setColor(Color.GREEN);
         y1 = acceleration[i].getData()[j][0].bigDecimalValue()
               .multiply(new BigDecimal("0.001"))
               .intValue();
         y2 = acceleration[i+1].getData()[j][0].bigDecimalValue()
               .multiply(new BigDecimal("0.001"))
               .intValue();
         graphics.drawLine(i*5, y1+250, (i+1)*5, y2+250);
      }

      //Draws the speed
      if (speed[i] != null && speed[i+1] != null) {
         graphics.setColor(Color.YELLOW);
         y1 = speed[i].getData()[j][0].bigDecimalValue()
               .multiply(new BigDecimal("10"))
               .intValue();
         y2 = speed[i+1].getData()[j][0].bigDecimalValue()
               .multiply(new BigDecimal("10"))
               .intValue();
         graphics.drawLine(i*5, y1+250, (i+1)*5, y2+250);
      }

      //Draws the force
      graphics.setColor(Color.CYAN);
      y1 = force[i].getData()[j][0].bigDecimalValue()
            .multiply(new BigDecimal("0.002"))
            .intValue();
      y2 = force[i+1].getData()[j][0].bigDecimalValue()
            .multiply(new BigDecimal("0.002"))
            .intValue();
      graphics.drawLine(i*5, y1+250, (i+1)*5, y2+250);

      //Draws a 0 line for reference
      graphics.setColor(Color.WHITE);
      ((Graphics2D)graphics).setStroke(new BasicStroke(1));
      graphics.drawLine(i*5, 250, (i+1)*5, 250);
   }

}
```

```
            bufferStrategy.show();
            graphics.dispose();

            try {
                //keeps drawing every 0.2 seconds
                Thread.sleep(200L);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }

    /**
     * Shows a window with an animation of all the nodes with a line representing speed, acceleration and force for
     * each node
     *
     * @param elements List with all the elements of the grid
     * @param displacement vector of displacement
     * @param speed vector of speed
     * @param acceleration vector of acceleration
     * @param force vector of force
     * @param numX number of elements in X
     * @param numY number of elements in Y
     * @param numTimes number of iterations
     */
    public static void printElements(List<Cst> elements, FieldMatrix<BigReal>[] displacement,
                                     FieldMatrix<BigReal>[] speed, FieldMatrix<BigReal>[] acceleration, FieldMatrix<BigReal>[]
force,
                                     Integer numX, Integer numY, Integer numTimes) {
        //assigns the title of the window
        final String title = "EWaveFem";
        final int width = 30*(numX+2);
        final int height = 30*(numY+3);

        //Creating the frame.
        JFrame frame = new JFrame(title);

        frame.setSize(width, height);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);
        frame.setResizable(false);
        frame.setVisible(true);

        //Creating the canvas.
        Canvas canvas = new Canvas();

        canvas.setSize(width, height);
        canvas.setBackground(Color.BLACK);
        canvas.setVisible(true);
        canvas.setFocusable(false);


        //Putting it all together.
        frame.add(canvas);
```

```
canvas.createBufferStrategy(3);

boolean running = true;

BufferStrategy bufferStrategy;
Graphics graphics;
int i = 0;

//keeps executing to show the animation
while (running) {
   bufferStrategy = canvas.getBufferStrategy();
   graphics = bufferStrategy.getDrawGraphics();
   graphics.clearRect(0, 0, width, height);

   graphics.setColor(Color.GREEN);
   graphics.clearRect(0, 0, 30*(numX+2), 30*(numY+3));

   for (int m = 0; m <= numY; m++) {
      for (int n = 0; n <= numX; n++) {

         //places the node based on the displacement
         int x = n*30 + displacement[i].getData()[m*(numX+1)*2+n*2][0].bigDecimalValue()
               .intValue();
         int y = m*30 + displacement[i].getData()[m*(numX+1)*2+n*2+1][0].bigDecimalValue()
               .intValue();

         //draws the node
         graphics.setColor(Color.GREEN);
         graphics.drawOval(x, y, 10, 10);

         //draws the speed vector
         if (speed[i] != null) {
            graphics.setColor(Color.BLUE);
            graphics.drawLine(x + 5, y + 5,
                  x + 5 + speed[i].getData()[m*(numX+1)*2+n*2][0].bigDecimalValue()
                        .multiply(new BigDecimal("0.01"))
                        .intValue(),
                  y + 5 + speed[i].getData()[m*(numX+1)*2+n*2+1][0].bigDecimalValue()
                        .multiply(new BigDecimal("0.01"))
                        .intValue());
         }

         //draws the acceleration vector
         if (acceleration[i] != null) {
            graphics.setColor(Color.RED);
            graphics.drawLine(x + 5, y + 5,
                  x + 5 + acceleration[i].getData()[m*(numX+1)*2+n*2][0].bigDecimalValue()
                        .multiply(new BigDecimal("0.0001"))
                        .intValue(),
                  y + 5 + acceleration[i].getData()[m*(numX+1)*2+n*2+1][0].bigDecimalValue()
                        .multiply(new BigDecimal("0.0001"))
                        .intValue());
         }

         //draws the force vector
         if (force[i] != null) {
```

```
                graphics.setColor(Color.CYAN);
                graphics.drawLine(x + 5, y + 5,
                        x + 5 + force[i].getData()[m*(numX+1)*2+n*2][0].bigDecimalValue()
                                .multiply(new BigDecimal("0.001"))
                                .intValue(),
                        y + 5 + force[i].getData()[m*(numX+1)*2+n*2+1][0].bigDecimalValue()
                                .multiply(new BigDecimal("0.001"))
                                .intValue());
            }
        }
    }

    //draws the current time iteration
    graphics.drawString("t: " + i, 100, 30*(numY+1));

    bufferStrategy.show();
    graphics.dispose();
    try {
        Thread.sleep(200L);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    if (i < numTimes - 1) {
        i++;
    } else {
        //restarts the unit count when reaches the end
        i = 0;
    }
  }
 }
}
```

Archivo de constantes: Constants.java

```
package com.pescamillam.fem.util;

import java.text.DecimalFormat;

import org.apache.commons.math3.util.BigReal;

/**
 * Constants with initial values and default decimal format
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class Constants {
    public static final BigReal FOUR = new BigReal("4");
    public static final BigReal TWELVE = new BigReal("12");
    public static final BigReal TWO = new BigReal("2");
    public static final BigReal MINUS_ONE = new BigReal("-1");

    public static final String THICKNESS = "1";
    public static final String ELASTICITY = "3000000";
    public static final String DENSITY = "0.00073";
```

```
    public static final String AREA = "450";
    public static final String POISSON = "0.3";
    public static final String DELTA_TIME = "0.00005";
    public static final String NUM_X = "8";
    public static final String NUM_Y = "8";
    public static final String NUM_TIMES = "200";

    public static DecimalFormat DF = new DecimalFormat();
    static {
        DF.setMaximumFractionDigits(10);
        DF.setMinimumFractionDigits(0);
        DF.setGroupingUsed(false);
    }
}
```

Archivo para generar el archivo de resultado: UtilWriter.java

```
package com.pescamillam.fem.util;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Class to write to output file
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class UtilWriter {

    private static BufferedWriter bw;


    /**
     * Prints to file the received string
     *
     * @param stringToWrite string to write to file
     */
    public static void writeToFile(String stringToWrite) {
        initializeWriterIfNecessary();
        try {
            bw.write(stringToWrite);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * In case the writer doesn't exists this creates it
     */
    private static void initializeWriterIfNecessary() {
        if (bw == null) {
            try {
                bw = new BufferedWriter(new FileWriter("output.txt"));
            } catch (IOException e) {
                e.printStackTrace();
```

```
        }
      }
    }
}
```

Archivo para modelar los datos de entrada: InputValues.java

```
package com.pescamillam.fem.model;

/**
 * Pojo class with the input values that the program can receive
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class InputValues {
    private String thickness;
    private String area;
    private String elasticity;
    private String density;
    private String poisson;
    private String deltaTime;
    private String numTimes;
    private String numX;
    private String numY;

    public InputValues(String thickness, String area, String elasticity, String density,
            String poisson, String deltaTime, String numTimes, String numX, String numY) {
        this.thickness = thickness;
        this.area = area;
        this.elasticity = elasticity;
        this.density = density;
        this.poisson = poisson;
        this.deltaTime = deltaTime;
        this.numTimes = numTimes;
        this.numX = numX;
        this.numY = numY;
    }

    public String getThickness() {
        return thickness;
    }



    public String getArea() {
        return area;
    }



    public String getElasticity() {
        return elasticity;
    }
```

```
public String getDensity() {
   return density;
}



public String getPoisson() {
   return poisson;
}



public String getDeltaTime() {
   return deltaTime;
}



public String getNumTimes() {
   return numTimes;
}

public String getNumX() {
   return numX;
}

public String getNumY() {
   return numY;
}

/**
 * Builder to create a InputValues instance easily
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public static class Builder {
   private String thickness;
   private String area;
   private String elasticity;
   private String density;
   private String poisson;
   private String deltaTime;
   private String numTimes;
   private String numX;
   private String numY;

   public Builder withThickness(String thickness) {
      this.thickness = thickness;
      return this;
   }

   public Builder withArea(String area) {
      this.area = area;
      return this;
   }
```

```
        public Builder withElasticity(String elasticity) {
            this.elasticity = elasticity;
            return this;
        }

        public Builder withDensity(String density) {
            this.density = density;
            return this;
        }

        public Builder withPoisson(String poisson) {
            this.poisson = poisson;
            return this;
        }

        public Builder withDeltaTime(String deltaTime) {
            this.deltaTime = deltaTime;
            return this;
        }

        public Builder withNumTimes(String numTimes) {
            this.numTimes = numTimes;
            return this;
        }

        public Builder withNumX(String numX) {
            this.numX = numX;
            return this;
        }

        public Builder withNumY(String numY) {
            this.numY = numY;
            return this;
        }

        public InputValues build() {
            return new InputValues(thickness, area, elasticity, density, poisson, deltaTime, numTimes, numX, numY);
        }

    }
}
```

Archivo que genera las matrices del elemento 1: ElementOne.java

```
package com.pescamillam.fem.element.cst;

import static com.pescamillam.fem.util.Constants.TWO;
import static org.apache.commons.math3.util.BigReal.ONE;
import static org.apache.commons.math3.util.BigReal.ZERO;

import java.math.BigDecimal;
import java.util.List;
```

```
import org.apache.commons.math3.util.BigReal;

import com.pescamillam.fem.element.Cst;
import com.pescamillam.fem.element.Point;

/**
 * Util class to manage the element one type
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class ElementOne {

   /**
    * Gets the local stiffness matrix for a element one
    * @param poisson poisson module as given by input values
    * @return local stiffness matrix of the element one
    */
   public static BigReal[][] getLocalStiffnessMatrix(BigReal poisson) {

      BigReal[][] localMatrix = new BigReal[6][6];

      //(1-v)
      BigReal poisson1mv = poisson.negate().add(new BigReal("1"));

      //(1-2v)/2
      BigReal poisson1m2vo2 = poisson.multiply(new BigReal("2")).negate().add(new BigReal("1")).divide(new
BigReal("2"));

      //element 1
      //
      // i
      // |\
      // | \
      // |__\
      // m   j

      //beta i: y_j - y_m
      BigReal betaIel1 = new BigReal("0");
      //beta j: y_m - y_i
      BigReal betaJel1 = new BigReal("30");
      //beta m: y_i - y_j
      BigReal betaMel1 = new BigReal("-30");

      //gamma_i: x_m - x_j
      BigReal gammaIel1 = new BigReal("-30");
      //gamma_j: x_i - x_m
      BigReal gammaJel1 = new BigReal("0");
      //gamma_m: x_j - x_i
      BigReal gammaMel1 = new BigReal("30");

      //1-1
      localMatrix[0][0] =
betaIel1.multiply(betaIel1).multiply(poisson1mv).add(gammaIel1.multiply(gammaIel1).multiply(poisson1m2vo2));
      //1-2
      localMatrix[0][1] =
betaIel1.multiply(betaIel1).multiply(poisson).add(betaIel1.multiply(gammaIel1).multiply(poisson1m2vo2));
```

```
        //1-3
        localMatrix[0][2] =
betaIel1.multiply(betaJel1).multiply(poisson1mv).add(gammaIel1.multiply(gammaJel1).multiply(poisson1m2vo2));
        //1-4
        localMatrix[0][3] =
betaIel1.multiply(gammaJel1).multiply(poisson).add(betaJel1.multiply(gammaIel1).multiply(poisson1m2vo2));
        //1-5
        localMatrix[0][4] =
betaIel1.multiply(betaMel1).multiply(poisson1mv).add(gammaIel1.multiply(gammaMel1).multiply(poisson1m2vo2));
        //1-6
        localMatrix[0][5] =
betaIel1.multiply(gammaMel1).multiply(poisson).add(betaMel1.multiply(gammaIel1).multiply(poisson1m2vo2));

        //2-1
        localMatrix[1][0] = localMatrix[0][1];
        //2-2
        localMatrix[1][1] =
gammaIel1.multiply(gammaIel1).multiply(poisson1mv).add(betaIel1.multiply(betaIel1).multiply(poisson1m2vo2));
        //2-3
        localMatrix[1][2] =
betaJel1.multiply(gammaIel1).multiply(poisson).add(betaIel1.multiply(gammaJel1).multiply(poisson1m2vo2));
        //2-4
        localMatrix[1][3] =
gammaIel1.multiply(gammaJel1).multiply(poisson1mv).add(betaIel1.multiply(betaJel1).multiply(poisson1m2vo2));
        //2-5
        localMatrix[1][4] =
betaMel1.multiply(gammaIel1).multiply(poisson).add(betaIel1.multiply(gammaMel1).multiply(poisson1m2vo2));
        //2-6
        localMatrix[1][5] =
gammaIel1.multiply(gammaMel1).multiply(poisson1mv).add(betaIel1.multiply(betaMel1).multiply(poisson1m2vo2));

        //3-3
        localMatrix[2][2] =
betaJel1.multiply(betaJel1).multiply(poisson1mv).add(gammaJel1.multiply(gammaJel1).multiply(poisson1m2vo2));
        //3-4
        localMatrix[2][3] =
betaJel1.multiply(gammaJel1).multiply(poisson).add(betaJel1.multiply(gammaJel1).multiply(poisson1m2vo2));
        //3-5
        localMatrix[2][4] =
betaJel1.multiply(betaMel1).multiply(poisson1mv).add(gammaJel1.multiply(gammaMel1).multiply(poisson1m2vo2));
        //3-6
        localMatrix[2][5] =
betaJel1.multiply(gammaMel1).multiply(poisson).add(betaMel1.multiply(gammaJel1).multiply(poisson1m2vo2));

        //4-3
        localMatrix[3][2] = localMatrix[2][3];
        //4-4
        localMatrix[3][3] =
gammaJel1.multiply(gammaJel1).multiply(poisson1mv).add(betaJel1.multiply(betaJel1).multiply(poisson1m2vo2));
        //4-5
        localMatrix[3][4] =
betaMel1.multiply(gammaJel1).multiply(poisson).add(betaJel1.multiply(gammaMel1).multiply(poisson1m2vo2));
        //4-6
        localMatrix[3][5] =
gammaIel1.multiply(gammaMel1).multiply(poisson1mv).add(betaJel1.multiply(betaMel1).multiply(poisson1m2vo2));
```

```
        //5-5
        localMatrix[4][4] =
betaMel1.multiply(betaMel1).multiply(poisson1mv).add(gammaMel1.multiply(gammaMel1).multiply(poisson1m2vo2));
        //5-6
        localMatrix[4][5] =
betaMel1.multiply(gammaMel1).multiply(poisson).add(betaMel1.multiply(gammaMel1).multiply(poisson1m2vo2));

        //6-5
        localMatrix[5][4] = localMatrix[4][5];
        //6-6
        localMatrix[5][5] =
gammaMel1.multiply(gammaMel1).multiply(poisson1mv).add(betaMel1.multiply(betaMel1).multiply(poisson1m2vo2));

        return localMatrix;
    }


    /**
     * Adds a local matrix element one to the global stiffness matrix
     *
     * @param elements list of elements
     * @param stiffnessMatrix global stiffness matrix
     * @param localMatrixElementOne local stiffness matrix to add
     * @param i element placement in X
     * @param j element placement in Y
     * @param numX number of elements in X
     */
    public static void appendElementOneToStiffnessMatrix(List<Cst> elements,
            BigReal[][] stiffnessMatrix, BigReal[][] localMatrixElementOne, int i, int j, Integer numX) {
        Cst element = new Cst(new Point(new BigDecimal(i*10), new BigDecimal(j*10)),
                new Point(new BigDecimal(i*10+10), new BigDecimal(j*10+10)),
                new Point(new BigDecimal(i*10), new BigDecimal(j*10+10)));
        elements.add(element);

        //top left corner
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2].add(localMatrixElementOne[0][0]);
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1].add(localMatrixElementOne[0][1]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2].add(localMatrixElementOne[0][1]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1].add(localMatrixElementOne[1][1]);

        //bottom left corner
        stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2].add(localMatrixElementOne[4][4]);
        stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2+1].add(localMatrixElementOne[4][5]);
        stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2].add(localMatrixElementOne[5][4]);
        stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2+1].add(localMatrixElementOne[5][5]);

        //bottom right corner
```

```
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementOne[2][2]);
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementOne[2][3]);
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementOne[2][3]);
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementOne[3][3]);

        //ij
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementOne[0][2]);
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementOne[0][3]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementOne[1][2]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementOne[1][3]);

        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1];

        //im
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2].add(localMatrixElementOne[0][4]);
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2+1].add(localMatrixElementOne[0][5]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2].add(localMatrixElementOne[1][4]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2+1].add(localMatrixElementOne[1][5]);

        stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2];
        stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2+1];
        stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2];
        stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2+1];

        //jm
        stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementOne[2][4]);
        stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementOne[3][4]);
        stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementOne[2][5]);
        stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementOne[3][5]);
```

```
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+i)*2] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+i)*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+i)*2] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2+1];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+i)*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1];
    }

    /**
     * Adds a local mass matrix element one to the global mass matrix
     *
     * @param massMatrix global mass matrix
     * @param i element placement in x
     * @param j element placement in y
     * @param numX total number of elements in X
     */
    public static void appendElementOneToMassMatrix(BigReal[][] massMatrix, int i, int j, Integer numX) {
        //element 1
        //
        // i
        // |\
        // | \
        // |__\
        // m   j

        //top left corner
        massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2].add(TWO);
        massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1].add(TWO);

        //bottom left corner
        massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2] =
massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2].add(TWO);
        massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2+1] =
massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+i)*2+1].add(ZERO);
        massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2] =
massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2].add(ZERO);
        massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2+1] =
massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+i)*2+1].add(TWO);

        //bottom right corner
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2].add(TWO);
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1].add(ZERO);
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2].add(ZERO);
```

```
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(TWO);

        //ij
        massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2].add(ONE);
        massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1].add(BigReal.ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2].add(BigReal.ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(ONE);

        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1];
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1];

        //im
        massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2].add(ONE);
        massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2+1] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2+1].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2+1].add(ONE);

        massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*j+i)*2] = massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2];
        massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+i)*2+1];
        massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2];
        massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+i)*2+1];

        //jm
        massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2].add(ONE);
        massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2].add(ZERO);
        massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2+1].add(ZERO);
        massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(ONE);

        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+i)*2] =
massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+i)*2+1] =
massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+i)*2] =
massMatrix[((numX+1)*(j+1)+i)*2][((numX+1)*(j+1)+(i+1))*2+1];
```

```
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+i)*2+1] =
massMatrix[((numX+1)*(j+1)+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1];
    }
}
```

Archivo que genera las matrices del elemento 2: ElementTwo.java

```
package com.pescamillam.fem.element.cst;

import static com.pescamillam.fem.util.Constants.TWO;
import static org.apache.commons.math3.util.BigReal.ONE;
import static org.apache.commons.math3.util.BigReal.ZERO;

import java.math.BigDecimal;
import java.util.List;

import org.apache.commons.math3.util.BigReal;

import com.pescamillam.fem.element.Cst;
import com.pescamillam.fem.element.Point;

/**
 * Util class to manage the element two type
 *
 * @author Peter Escamilla (pescamilla@unab.edu.co)
 */
public class ElementTwo {

    /**
     * Gets the local stiffness matrix for an element two
     * @param poisson poisson module as given by input values
     * @return local stiffness matrix of the element two
     */
    public static BigReal[][] getLocalStiffnessMatrix(BigReal poisson) {

        BigReal[][] localMatrix = new BigReal[6][6];

        //(1-v)
        BigReal poisson1mv = poisson.negate().add(new BigReal("1"));

        //(1-2v)/2
        BigReal poisson1m2vo2 = poisson.multiply(new BigReal("2")).negate().add(new BigReal("1")).divide(new
BigReal("2"));

        //element 2
        //
        // i___j
        //  \  |
        //   \ |
        //    \|
        //     m

        //beta i: y_j - y_m
        BigReal betaIel2 = new BigReal("-30");
        //beta j: y_m - y_i
        BigReal betaJel2 = new BigReal("30");
```

```
        //beta m: y_i - y_j
        BigReal betaMel2 = new BigReal("0");

        //gamma_i: x_m - x_j
        BigReal gammaIel2 = new BigReal("0");
        //gamma_j: x_i - x_m
        BigReal gammaJel2 = new BigReal("-30");
        //gamma_m: x_j - x_i
        BigReal gammaMel2 = new BigReal("30");

        //1-1
        localMatrix[0][0] =
betaIel2.multiply(betaIel2).multiply(poisson1mv).add(gammaIel2.multiply(gammaIel2).multiply(poisson1m2vo2));
        //1-2
        localMatrix[0][1] =
betaIel2.multiply(betaIel2).multiply(poisson).add(betaIel2.multiply(gammaIel2).multiply(poisson1m2vo2));
        //1-3
        localMatrix[0][2] =
betaIel2.multiply(betaJel2).multiply(poisson1mv).add(gammaIel2.multiply(gammaJel2).multiply(poisson1m2vo2));
        //1-4
        localMatrix[0][3] =
betaIel2.multiply(gammaJel2).multiply(poisson).add(betaJel2.multiply(gammaIel2).multiply(poisson1m2vo2));
        //1-5
        localMatrix[0][4] =
betaIel2.multiply(betaMel2).multiply(poisson1mv).add(gammaIel2.multiply(gammaMel2).multiply(poisson1m2vo2));
        //1-6
        localMatrix[0][5] =
betaIel2.multiply(gammaMel2).multiply(poisson).add(betaMel2.multiply(gammaIel2).multiply(poisson1m2vo2));

        //2-2
        localMatrix[1][1] =
gammaIel2.multiply(gammaIel2).multiply(poisson1mv).add(betaIel2.multiply(betaIel2).multiply(poisson1m2vo2));
        //2-3
        localMatrix[1][2] =
betaJel2.multiply(gammaIel2).multiply(poisson).add(betaIel2.multiply(gammaJel2).multiply(poisson1m2vo2));
        //2-4
        localMatrix[1][3] =
gammaIel2.multiply(gammaJel2).multiply(poisson1mv).add(betaIel2.multiply(betaJel2).multiply(poisson1m2vo2));
        //2-5
        localMatrix[1][4] =
betaMel2.multiply(gammaIel2).multiply(poisson).add(betaIel2.multiply(gammaMel2).multiply(poisson1m2vo2));
        //2-6
        localMatrix[1][5] =
gammaIel2.multiply(gammaMel2).multiply(poisson1mv).add(betaIel2.multiply(betaMel2).multiply(poisson1m2vo2));

        //3-3
        localMatrix[2][2] =
betaJel2.multiply(betaJel2).multiply(poisson1mv).add(gammaJel2.multiply(gammaJel2).multiply(poisson1m2vo2));
        //3-4
        localMatrix[2][3] =
betaJel2.multiply(gammaJel2).multiply(poisson).add(betaJel2.multiply(gammaJel2).multiply(poisson1m2vo2));
        //3-5
        localMatrix[2][4] =
betaJel2.multiply(betaMel2).multiply(poisson1mv).add(gammaJel2.multiply(gammaMel2).multiply(poisson1m2vo2));
        //3-6
```

```
        localMatrix[2][5] =
betaJel2.multiply(gammaMel2).multiply(poisson).add(betaMel2.multiply(gammaJel2).multiply(poisson1m2vo2));

        //4-4
        localMatrix[3][3] =
gammaJel2.multiply(gammaJel2).multiply(poisson1mv).add(betaJel2.multiply(betaJel2).multiply(poisson1m2vo2));
        //4-5
        localMatrix[3][4] =
betaMel2.multiply(gammaJel2).multiply(poisson).add(betaJel2.multiply(gammaMel2).multiply(poisson1m2vo2));
        //4-6
        localMatrix[3][5] =
gammaIel2.multiply(gammaMel2).multiply(poisson1mv).add(betaJel2.multiply(betaMel2).multiply(poisson1m2vo2));

        //5-5
        localMatrix[4][4] =
betaMel2.multiply(betaMel2).multiply(poisson1mv).add(gammaMel2.multiply(gammaMel2).multiply(poisson1m2vo2));
        //5-6
        localMatrix[4][5] =
betaMel2.multiply(gammaMel2).multiply(poisson).add(betaMel2.multiply(gammaMel2).multiply(poisson1m2vo2));

        //6-6
        localMatrix[5][5] =
gammaMel2.multiply(gammaMel2).multiply(poisson1mv).add(betaMel2.multiply(betaMel2).multiply(poisson1m2vo2));
        return localMatrix;
    }

    /**
     * Adds a local matrix element two to the global stiffness matrix
     *
     * @param elements list of elements
     * @param stiffnessMatrix global stiffness matrix
     * @param localMatrixElementTwo local stiffness matrix to add
     * @param i element placement in X
     * @param j element placement in Y
     * @param numX number of elements in X
     */
    public static void appendElementTwoToStiffnessMatrix(List<Cst> elements,
            BigReal[][] stiffnessMatrix, BigReal[][] localMatrixElementTwo, int i, int j, Integer numX) {
        Cst element2 = new Cst(new Point(new BigDecimal(i*10), new BigDecimal(j*10)),
                new Point(new BigDecimal(i*10+10), new BigDecimal(j*10)),
                new Point(new BigDecimal(i*10+10), new BigDecimal(j*10+10)));
        elements.add(element2);
        //top left corner
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2].add(localMatrixElementTwo[0][0]);
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1].add(localMatrixElementTwo[0][1]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2].add(localMatrixElementTwo[0][1]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1].add(localMatrixElementTwo[1][1]);

        //top right corner
        stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2].add(localMatrixElementTwo[2][2]);
```

```
        stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2+1].add(localMatrixElementTwo[2][3]);
        stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2].add(localMatrixElementTwo[2][3]);
        stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2+1].add(localMatrixElementTwo[3][3]);

        //bottom right corner
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementTwo[4][4]);
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementTwo[4][5]);
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementTwo[4][5]);
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementTwo[5][5]);

        //ij
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2].add(localMatrixElementTwo[0][2]);
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2+1].add(localMatrixElementTwo[0][3]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2].add(localMatrixElementTwo[1][2]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2+1].add(localMatrixElementTwo[1][3]);


        stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2];
        stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2+1];
        stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2];
        stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2+1];

        //im
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementTwo[0][4]);
        stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementTwo[0][5]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementTwo[1][4]);
        stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementTwo[1][5]);

        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2] =
stiffnessMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2+1] =
stiffnessMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1];
```

```
        //jm
        stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementTwo[2][4]);
        stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementTwo[2][5]);
        stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2].add(localMatrixElementTwo[3][4]);
        stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(localMatrixElementTwo[3][5]);

        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+(i+1))*2] =
stiffnessMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2];
        stiffnessMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+(i+1))*2+1] =
stiffnessMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1];
    }

    /**
     * Adds a local mass matrix element two to the global mass matrix
     *
     * @param massMatrix global mass matrix
     * @param i element placement in x
     * @param j element placement in y
     * @param numX total number of elements in X
     */
    public static void appendElementTwoToMassMatrix(BigReal[][] massMatrix, int i, int j, Integer numX) {
        //element 2
        //
        // i___j
        //  \  |
        //   \ |
        //    \|
        //     m

        //top left corner
        massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2].add(TWO);
        massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*j+i)*2+1].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+i)*2+1].add(TWO);

        //top right corner
        massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2] =
massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2].add(TWO);
        massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2+1] =
massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+(i+1))*2+1].add(ZERO);
        massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2] =
massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2].add(ZERO);
        massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2+1] =
massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+(i+1))*2+1].add(TWO);
```

```
        //bottom right corner
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2].add(TWO);
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1].add(ZERO);
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2].add(ZERO);
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(TWO);

        //ij
        massMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2].add(ONE);
        massMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2+1] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2+1].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2+1].add(ONE);


        massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+i)*2] = massMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2];
        massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*j+(i+1))*2+1];
        massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2];
        massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*j+(i+1))*2+1];

        //im
        massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2].add(ONE);
        massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2].add(ZERO);
        massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(ONE);

        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2] =
massMatrix[((numX+1)*j+i)*2][((numX+1)*(j+1)+(i+1))*2+1];
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+i)*2+1] =
massMatrix[((numX+1)*j+i)*2+1][((numX+1)*(j+1)+(i+1))*2+1];

        //jm
        massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2].add(ONE);
        massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1].add(ZERO);
        massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2] =
massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2].add(ZERO);
```

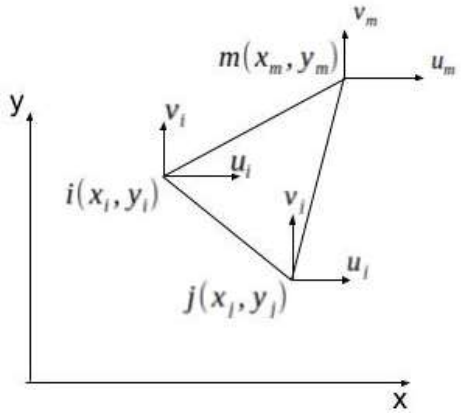
```
        massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1] =
massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1].add(ONE);

        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+(i+1))*2] =
massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+(i+1))*2] =
massMatrix[((numX+1)*j+(i+1))*2][((numX+1)*(j+1)+(i+1))*2+1];
        massMatrix[((numX+1)*(j+1)+(i+1))*2][((numX+1)*j+(i+1))*2+1] =
massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2];
        massMatrix[((numX+1)*(j+1)+(i+1))*2+1][((numX+1)*j+(i+1))*2+1] =
massMatrix[((numX+1)*j+(i+1))*2+1][((numX+1)*(j+1)+(i+1))*2+1];
    }

}
```

ANEXO E. Formulación matriz de rigidez

Esta matriz se halla a partir de la siguiente formulación extraída del libro de Logan (Logan, A first course in the finite element method, 2011):

Tomamos uno de los elementos triangulares sobre los cuales vamos a definir el elemento general y obtenemos sus grados de libertad.

Ilustración 35. Elemento triangular con grados de libertad.



Fuente: (Logan, A first course in the finite element method, 2011)

En la Ilustración 35 se ven los diferentes nodos del elemento, i, j, m con sus respectivas posiciones identificadas por $i(x_i, y_i)j(x_j, y_j)m(x_m, y_m)$y con los grados de libertad $v_i u_i v_j u_j v_m u_m$ sobre cada nodo respectivamente.

Con esta definición podemos generar las funciones lineales de movimiento

$$\begin{aligned} u(x,y) &= a_1 + a_2 x + a_3 y \\ v(x,y) &= a_4 + a_5 x + a_6 y \end{aligned} \tag{35}$$

Y con estas dos funciones podemos obtener la función de desplazamiento general Ψ

$$\Psi = \begin{Bmatrix} u(x,y) = a_1 + a_2 x + a_3 y \\ v(x,y) = a_4 + a_5 x + a_6 y \end{Bmatrix} = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & x & y \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{Bmatrix} \tag{36}$$

Con esta función podemos obtener el valor de $a_x$tomando las diferentes funciones de desplazamiento para cada nodo

$$\begin{gathered} u_i = u(x_i, y_i) = a_1 + a_2 x_i + a_3 y_i \\ u_j = u(x_j, y_j) = a_1 + a_2 x_j + a_3 y_j \\ u_m = u(x_m, y_m) = a_1 + a_2 x_m + a_3 y_m \\ v_i = v(x_i, y_i) = a_4 + a_5 x_i + a_6 y_i \\ v_j = v(x_j, y_j) = a_4 + a_5 x_j + a_6 y_j \\ v_m = v(x_m, y_m) = a_4 + a_5 x_m + a_6 y_m \end{gathered} \tag{37}$$

Tomando las primeras 3 ecuaciones tenemos

$$\begin{Bmatrix} u_i \\ u_j \\ u_m \end{Bmatrix} = \begin{bmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_m & y_m \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ a_3 \end{Bmatrix} \tag{38}$$

A partir de la cual se obtiene que

$$[x]^{-1} = \frac{1}{2A} \begin{bmatrix} \alpha_i & \alpha_j & \alpha_m \\ \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix} \tag{39}$$

Donde

$$2A = \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_m & y_m \end{vmatrix} \tag{40}$$

A es el área del triángulo y

$$\begin{array}{lll} \alpha_i = x_j y_m - x_m y_j & \alpha_j = x_m y_i - x_i y_m & \alpha_m = x_i y_j - y_i x_j \\ \beta_i = y_j - y_m & \beta_j = y_m - y_i & \beta_m = y_i - y_j \\ \gamma_i = x_m - x_j & \gamma_j = x_i - x_m & \gamma_m = x_j - x_i \end{array} \tag{41}$$

Ahora se puede derivar la función de desplazamiento$u(x, y)$de $\{\Psi\}$

Tendríamos la función:

$$\{u\} = \frac{1}{2A}[1 \quad x \quad y]\begin{bmatrix} \alpha_i & \alpha_j & \alpha_m \\ \beta_i & \beta_j & \beta_m \\ \gamma_i & \gamma_j & \gamma_m \end{bmatrix}\begin{Bmatrix} u_i \\ u_j \\ u_m \end{Bmatrix} \tag{42}$$

La cual al resolver nos daría:

$$\begin{gathered} u(x,y) = \frac{1}{2A}\{(\alpha_i + \beta_i x + \gamma_i y)u_i \\ +\big(\alpha_j + \beta_j x + \gamma_j y\big)u_j + (\alpha_m + \beta_m x + \gamma_m y)u_m\big\} \end{gathered} \tag{43}$$

De la misma manera se resuelve para v(x, y) así:

$$\begin{gathered} v(x,y) = \frac{1}{2A}\{(\alpha_i + \beta_i x + \gamma_i y)v_i \\ +\big(\alpha_j + \beta_j x + \gamma_j y\big)v_j + (\alpha_m + \beta_m x + \gamma_m y)v_m\big\} \end{gathered} \tag{44}$$

Para simplificar estas dos ecuaciones se puede realizar la siguiente definición

$$\begin{aligned} N_i &= \frac{1}{2A}(\alpha_i + \beta_i x + \gamma_i y) \\ N_j &= \frac{1}{2A}\big(\alpha_j + \beta_j x + \gamma_j y\big) \\ N_m &= \frac{1}{2A}(\alpha_m + \beta_m x + \gamma_m y) \end{aligned} \tag{45}$$

Con lo que se podrían reescribir las ecuaciones (93) y (94)

$$\begin{aligned} u(x,y) &= N_i u_i + N_j u_j + N_m u_m \\ v(x,y) &= N_i v_i + N_j v_j + N_m v_m \end{aligned} \tag{46}$$

Lo cual podría escribirse de manera abreviada en forma matricial como

$$\{\Psi\} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_m & 0 \\ 0 & N_i & 0 & N_j & 0 & N_m \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix} \tag{47}$$

La cual a su vez abreviada puede ser vista como

$$\{\Psi\} = [N]\{d\} \tag{48}$$

Ahora definiendo el vector tensor

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\delta u}{\delta x} \\ \dfrac{\delta v}{\delta y} \\ \dfrac{\delta u}{\delta y} + \dfrac{\delta v}{\delta x} \end{Bmatrix} \tag{49}$$

Donde usando la ecuación (14) se tendría que

$$\varepsilon_y = a_6 \gamma_{xy} = a_3 + a_5 \tag{50}$$

Ahora podemos derivar la ecuación de movimiento así:

$$u_{,x} = N_{i,x}u_i + N_{j,x}u_j + N_{m,x}u_m \tag{51}$$

Y derivando las funciones de forma tendríamos:

$$N_{i,x} = \frac{1}{2A}\frac{\delta}{\delta x}(\alpha_i + \beta_i x + \gamma_i y) = \frac{\beta_i}{2A} \tag{52}$$

De la misma forma se obtiene:

$$N_{j,x} = \frac{\beta_j}{2A}\, N_{m,x} = \frac{\beta_m}{2A} \tag{53}$$

Con lo que tendríamos la formulas:

$$\begin{gathered} \frac{\delta u}{\delta x} = \frac{1}{2A}\left(\beta_i u_i + \beta_j u_j + \beta_m u_m\right) \\ \frac{\delta v}{\delta y} = \frac{1}{2A}\left(\gamma_i v_i + \gamma_j v_j + \gamma_m v_m\right) \\ \frac{\delta u}{\delta y} + \frac{\delta v}{\delta x} = \frac{1}{2A}\left(\gamma_i u_i + \beta_i v_i + \gamma_j u_j + \beta_j v_j + \gamma_m u_m + \beta_m v_m\right) \end{gathered} \tag{54}$$

Estas fórmulas pueden pasarse a modo matricial indicando el vector tensor así:

$$\{\varepsilon\} = \frac{1}{2A}\begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_m \\ v_m \end{Bmatrix} \tag{55}$$

Esto simplificado nos daría la formula

$$\{\varepsilon\} = [B]\{d\} \tag{56}$$

Ahora la relación de Tensor Tensión se formularia como

$$\{\sigma\} = [D]\{\varepsilon\} \tag{57}$$

Lo que sería igual a

$$\{\sigma\} = [D][B]\{d\} \tag{58}$$

Ahora usando el principio de la mínima energía potencial y teniendo en cuenta que la energía tensora es

$$U = \frac{1}{2}\iiint_V \{\varepsilon\}^T \{\sigma\} dV \tag{59}$$

Tenemos que la energía potencial está dada por

$$\Omega_b = -\iiint_V \{\Psi\}^T \{X\} dV \tag{60}$$

Con esto tenemos la carga total del sistema {f} que es

$$\{f\} = \iiint_V [N]^T X dV + P + \iint_S [N_S]^T \{T_S\} dS \tag{61}$$

Con esto podemos obtener

$$[k] = \iiint_V [B]^T [D][B] dV \tag{62}$$

Para un elemento con grosor constante se sacaría la constante quedando

$$[k] = t \iint_A [B]^T [D][B] dxdy \tag{63}$$

Así que la matriz de rigidez es igual a:

$$[k] = tA[B]^T[D][B] \tag{64}$$

Donde para un elemento triangular la matriz de Tensor Tensión [D] es igual a:

$$[D] = \frac{E}{(1+v)(1-2v)} \begin{bmatrix} 1-v & v & 0 \\ v & 1-v & 0 \\ 0 & 0 & \frac{1-2v}{2} \end{bmatrix} \tag{65}$$

Y la matriz de gradiente [B] es igual a:

$$[B] = \left[[B_i] [B_j] [B_m] \right] \tag{66}$$

Donde:

$$[B_i] = \frac{1}{2A} \begin{bmatrix} \beta_i & 0 \\ 0 & \gamma_i \\ \gamma_i & \beta_i \end{bmatrix} \tag{67}$$

$$[B_j] = \frac{1}{2A} \begin{bmatrix} \beta_j & 0 \\ 0 & \gamma_j \\ \gamma_j & \beta_j \end{bmatrix} \tag{68}$$

$$[B_m] = \frac{1}{2A} \begin{bmatrix} \beta_m & 0 \\ 0 & \gamma_m \\ \gamma_m & \beta_m \end{bmatrix} \tag{69}$$

Se define la matriz de rigidez para un elemento triangular:

$$[k] = \frac{tE}{4A(1+v)(1-2v)} \begin{bmatrix} \beta_i & 0 & \gamma_i \\ 0 & \gamma_i & \beta_i \\ \beta_j & 0 & \gamma_j \\ 0 & \gamma_j & \beta_j \\ \beta_m & 0 & \gamma_m \\ 0 & \gamma_m & \beta_m \end{bmatrix} \times \tag{70}$$

$$\begin{bmatrix} 1-v & v & 0 \\ v & 1-v & 0 \\ 0 & 0 & \frac{1-2v}{2} \end{bmatrix} \begin{bmatrix} \beta_i & 0 & \beta_j & 0 & \beta_m & 0 \\ 0 & \gamma_i & 0 & \gamma_j & 0 & \gamma_m \\ \gamma_i & \beta_i & \gamma_j & \beta_j & \gamma_m & \beta_m \end{bmatrix}$$

ANEXO F. Formulación matriz de masa

Esta matriz se halla a partir de la siguiente formulación extraída del libro de Logan (Logan, A first course in the finite element method, 2011):

Para la formulación de la matriz de masa primero tomamos la matriz de función de forma

$$[N] = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \tag{71}$$

Y la fórmula de la matriz de masa del elemento

$$[m] = \iiint_V \rho\,[N]^T[N]dV \tag{72}$$

Para obtener la matriz de masa del elemento triangular simple, teniendo en cuenta que:

$$dV = tdA \int_A N_1^2\,dA = \frac{1}{6}A \int_A N_1\,N_2 dA = \frac{1}{12}A \tag{73}$$

Se tiene que la matriz de masa sería:

$$[m] = \frac{\rho tA}{12}\begin{bmatrix} 2 & 0 & 1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 1 \\ 1 & 0 & 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 & 0 & 1 \\ 1 & 0 & 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 & 0 & 2 \end{bmatrix} \tag{74}$$

ANEXO G. Algoritmo de elementos finitos en el tiempo

El algoritmo expuesto a continuación es principalmente tomado del libro de Morgan (Morgan, Foundations of Wave Theory for Seismic Exploration, 1983):

Lo primero es obtener cada uno de los elementos individuales y a estos obtenerles la matriz de rigidez y de masa por medio de las ecuaciones del ANEXO D y ANEXO E.

$[k_i]$: Matriz de rigidez para el elemento i
$[m_i]$: Matriz de masa para el elemento i

Luego de tener esto se deben calcular tanto la matriz global de masa como la matriz global de rigidez

La matriz global de rigidez y la matriz global de masa se obtienen por medio de la adición de sus respectivas matrices locales tipo 1 y tipo 2 cuyo proceso está detallado en el ANEXO H y cuyo diagrama de flujo puede ser visto en el capítulo 3.4.

Estas sumatorias se realizan teniendo en cuenta los nodos que representa cada fila / columna de la matriz

Luego de tener las matrices de masa y de rigidez se puede definir a gusto propio el vector de fuerza que va a afectar el sistema

$\{F\}$: Vector de fuerza
Al tener estos valores debemos realizar la operación más costosa la cual es invertir la matriz global de masa

$[M]^{-1}$: Inversa de la matriz global de masa

Ahora se puede determinar para el tiempo 0 un vector de desplazamiento cualquiera, por ejemplo 0.

$\{d_0\}$: Vector de desplazamiento en el tiempo 0

Y se podría determinar para cada tiempo la aceleración, el desplazamiento y la velocidad en cada nodo por medio de descomposición LU (Bartels & Golub, 1969).

La primera ecuación para determinar estos valores es la de la aceleración en el tiempo 0.

$$\{\ddot{d}_0\} = [M]^{-1}(\{F_0\} - [K]\{d_0\}) \tag{75}$$

Luego de esto y determinando una velocidad inicial cualquiera se puede hallar el desplazamiento en el tiempo -1 (es necesario para las siguientes ecuaciones).

$$\{d_{-1}\} = \{d_0\} - \Delta t\{\dot{d}_0\} + \frac{(\Delta t)^2}{2}\{\ddot{d}_0\} \tag{76}$$

Ahora iteramos según necesitemos sobre las siguientes ecuaciones para determinar en el tiempo i el desplazamiento, la aceleración y la velocidad.

$$\begin{gathered}\{d_i\} = [M]^{-1}\{(\Delta t)^2\{F_{i-1}\} \\ +\left[2[M] - (\Delta t)^2[K]\right]\{d_{i-1}\} - [M]\{d_{i-2}\}\}\end{gathered} \tag{77}$$

$$\{\ddot{d}_i\} = [M]^{-1}(\{F_i\} - [K]\{d_i\}) \tag{78}$$

$$\{\dot{d}_i\} = \frac{\{d_{i+1}\} - \{d_{i-1}\}}{2(\Delta t)} \tag{79}$$

Esta es la solución general para cualquier modelado de ondas por medio de elementos finitos en el tiempo.

ANEXO H. Adición de matrices de cada elemento

Para adicionar las matrices tanto de rigidez como de masa hay que tener en cuenta los nodos compartidos y el orden de estos.

Teniendo en cuenta la Ilustración 7 y la Ilustración 8 se pueden generar por ejemplo las ecuaciones (138) y (139) que al adicionar se verían como la ecuación (140)

$$\begin{bmatrix} a & b & c & d & e & f \\ b & g & h & i & j & k \\ c & h & l & m & n & o \\ d & i & m & p & q & r \\ e & j & n & q & s & t \\ f & k & o & r & t & u \end{bmatrix} \tag{80}$$

$$\begin{bmatrix} a' & b' & c' & d' & e' & f' \\ b' & g' & h' & i' & j' & k' \\ c' & h' & l' & m' & n' & o' \\ d' & i' & m' & p' & q' & r' \\ e' & j' & n' & q' & s' & t' \\ f' & k' & o' & r' & t' & u' \end{bmatrix} \tag{81}$$

$$\begin{bmatrix} a & b & c & d & e & f & 0 & 0 \\ b & g & h & i & j & k & 0 & 0 \\ c & h & l+a' & m+b' & n+c' & o+d' & e' & f' \\ d & i & m+b' & p+g' & q+h' & r+i' & j' & k' \\ e & j & n+c' & q+h' & s+l' & t+m' & n' & o' \\ f & k & o+d' & r+i' & t+m' & u+p' & q' & r' \\ 0 & 0 & e' & j' & n' & q' & s' & t' \\ 0 & 0 & f' & k' & o' & r' & t' & u' \end{bmatrix} \tag{82}$$

Esta suma se hace haciendo una representación general de nodos del sistema global y reordenando según los nodos de cada elemento en la matriz general