UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"SOFTWARE PARA LA MODELACIÓN DE DATOS GRAVIMÉTRICOS"

TESIS

QUE PARA OBTENER EL TITULO DE:

INGENIERO EN COMPUTACIÓN

PRESENTA:

GUSTAVO HERNÁNDEZ MARTÍNEZ

DIRECTORA DE TESIS:

DRA. ARACELI ZAMORA CAMACHO

CIUDAD UNIVERSITARIA, MÉXICO D.F. 2015

A mi Padre, Roberto Hernández Vera (q.e.p.d.).
Por el apoyo incondicional, su valor mostrado
para salir adelante y por su amor.
Promesa cumplida.

A mi Madre, Irene Martínez.
Por sus consejos, motivación constante
para ser una persona de bien y
por todo el amor que me ha dado.

Agradecimientos

A mis hermanos, Carolina y Roberto Iván, por estar conmigo en cada momento, el apoyo y la gran amistad que nos tenemos, Los quiero mucho.

A la Dra. Araceli Zamora Camacho y al Dr. Juan Manuel Espíndola Castro les agradezco por el apoyo y la paciencia que me tuvieron para realizar esta Tesis y su gran calidez humana durante la estancia en el Instituto de Geofísica.

A mis maestros, gracias por su tiempo, su apoyo, así como el conocimiento transmitido en el desarrollo de mi formación profesional.

A la Universidad Nacional Autónoma de México, así como a la Facultad de Ingeniería, por todo lo que he aprendido y ayudarme a desarrollarme como persona.

INDICE

Capítulo I: Introducción

1.1. Las tecnologías de la información

La tecnología moderna es una gran herramienta para la manipulación de datos y facilitación del trabajo. Su efectividad la hace pervasiva en todas las ramas de la ciencia y en muchos aspectos de la vida cotidiana. Esto se hace muy evidente en cuanto a la computación, tanto en su aspecto de “hardware” como de “software”.

Incluso en problemas cuya solución ya había sido obtenida en el pasado, los adelantos modernos permiten aplicar dichas soluciones con mayor eficiencia resultando en una economía de tiempo y en una mejoría en la interpretación de los resultados. Esto es común para todas las ciencias y en particular para la Geofísica que deriva muchos de sus conocimientos de modelos teóricos que solo pueden resolverse conceptualmente por métodos numéricos y prácticamente por medio de computadoras. En esta Tesis nos enfocaremos en agilizar, manipular, interpretar, facilitar la obtención y modelado de datos en el área de la geofísica de exploración, específicamente en el método de interpretación directa de datos gravimétricos.

El método gravimétrico es ampliamente utilizado en Geofísica, y consiste, sucintamente, en inferir la estructura interna de la tierra por medido de los cambios en la gravedad causados por el contraste de densidad en las rocas del subsuelo. Estos cambios se miden por medio de gravímetros en la superficie y la estructura

se infiere a través de la interpretación de los mismos. Para la interpretación directa se procede obteniendo modelos teóricos, comparándolos con los observados y variando el modelo hasta que se consigue un buen ajuste.

La aplicación es designada como ‘Análisis de Datos Gravimétricos en Áreas Volcánicas’ (ADGAVO) y mide los cambios de gravedad a lo largo de perfiles sobre los cuerpos geométricos que simulan las rocas del subsuelo.

El lenguaje seleccionado para la generación de los datos teóricos es Java y con él se desarrolló un software que además de generar los datos teóricos permite compararlos con los observados y hace que el procedimiento prueba – error para el ajuste de valores gravimétricos, se realice ágilmente.

1.2. La Vulcanología

La vulcanología es una rama de la geología que se encarga del estudio de los volcanes, así como otros fenómenos geológicos relacionados a éstos. Su objetivo último es la predicción de erupciones, lo cual hasta el momento es todavía algo muy lejano. Para su estudio utiliza métodos geofísicos, geodésicos, geoquímicos y geológicos, que se apoyan en tecnologías electrónicas, de comunicaciones e informáticas. El estudio de la vulcanología ha evolucionado principalmente en los últimos dos siglos. Sin embargo, la primera erupción registrada en la historia fue la del volcán Vesubio en el año 79 D.C. y fue descrita por Plinio el Joven. En 1841 Giuseppe Mercalli fundó el primer observatorio vulcanológico del mundo en el reino de las dos sicilias, llamado el observatorio Vesubio. En 1879 se creó el Servicio Geológico de los Estados Unidos para consolidar las ciencias de la tierra desde una sola organización. En 1922 se fundó el Bulletin Volcanologique, un boletín que publica constantemente artículos científicos relacionados a la vulcanología. Este boletín fue creado por la International Association of Volcanology and Chemistry of the Earth's Interior (IAVCEI) que fue creada en 1919 como un esfuerzo internacional por unificar el conocimiento relacionado a la vulcanología. La vulcanología se ayuda de diversas herramientas para sus estudios. Se usan sismógrafos cerca de los volcanes para registrar la sismicidad durante eventos volcánicos. También se monitoriza la deformación de la superficie y se observa los cambios de temperatura, así como las emisiones de gases. Se usan satélites para monitorizar grandes áreas sin necesidad de tener instrumentos

localmente, principalmente esto se hace en zonas de difícil acceso en los que usar otros instrumentos sería demasiado caro. Anteriormente, todos los datos de los instrumentos tenían que ser capturados manualmente en la computadora para luego procesarlos o analizarlos por medio de programas. El desarrollo de la tecnología ha llegado también a estas ciencias y actualmente los equipos pueden almacenar la información electrónicamente, la cual es transferida a memorias externas o por medio de la red. Estos equipos son extremadamente caros, debido a que deben tener una gran precisión. Muchos de los datos recogidos y el análisis que se hace sirve para determinar el nivel de actividad de un volcán, lo que puede ayudar a prevenir a la población en caso de una erupción inminente. Esto es muy difícil de hacer, pues los volcanes son muy impredecibles, sin embargo, a través de los años el conocimiento sobre ellos ha aumentado considerablemente. Sin embargo, es también del interés de los vulcanólogos el análisis de erupciones que ya ocurrieron, pues en gran medida ayudan a entender cómo se puede desarrollar en caso de que vuelva a ocurrir. Para esto se utilizan modelos matemáticos que pueden describir estas erupciones. Constantemente los científicos crean modelos para describir fenómenos vulcanológicos, sin embargo muchos de esos modelos se quedan sólo en la teoría. Tal es el caso del modelo que describe densidad, propuesto por Talwani y sobre el que se hablará más adelante. Descrito a grandes rasgos, este modelo permite obtener la gravedad en ciertos puntos de interés alrededor de un volcán. Aun cuando los investigadores implementan modelos matemáticos como el de Talwani, generalmente lo hacen para uso personal, por lo que sus programas no hacen uso de una interfaz gráfica amigable y es complicado para otra persona usarlos si es que lo requiriera. [7]

1.3. Objetivo

Elaborar software para la obtención rápida de la componente vertical de la atracción gravitacional causada por un cuerpo irregular con cualquier contraste de densidad para la interpretación de datos geofísicos.

1.3.1. Objetivo General

Este programa facilitará la interpretación directa de datos gravimétricos y sería útil en el entrenamiento de los estudiantes de licenciatura y posgrado en Ciencias de la Tierra.

1.3.2. Objetivos Particulares

1.3.2.1. Introducir datos rápidamente.

1.3.2.2. Obtener los resultados de forma gráfica en muy poco tiempo.

1.3.2.3. Introducir los datos observados para su comparación.

1.3.2.4. Cuando el ajuste es satisfactorio, poder obtener además de la gráfica; un archivo de datos para su posterior manipulación.

1.4. Limitaciones

El problema se limitó a vulcanología, aunque se podría aplicar a otras ciencias. Sin embargo, es en ésta se identificó dicho problema por lo que sólo se trabajará en esta área de la física. Además se usará un ejemplo de un modelo para calcular la densidad, para demostrar la manera en la que un Ingeniero en Computación puede aportar a la vulcanología aplicando principalmente los conocimientos de programación orientada a objetos e ingeniería de software, aunque también se aplicaron otros conocimientos.

Capitulo II: Marco Teórico

2.1. Definición del problema.

Describir claramente el problema a resolver en cuanto a contexto, alcance, conexión, con otros problemas, justificación, relevancia y objetivo preciso.

Uno de los métodos más utilizados en geofísica de exploración es el método gravimétrico. Este método consiste en medir, por medio de instrumentos muy sensibles, el cambio en la componente vertical de la gravedad debida a cambios en la densidad de las rocas del subsuelo. A estos cambios se les conoce como anomalías gravimétricas y su determinación permite inferir la estructura del subsuelo. Uno de los métodos de interpretación de los datos se lleva a cabo comparando la respuesta de modelos teóricos de cuerpos con contraste de densidad, con los datos obtenidos. Cuando las anomalías son extensas el análisis se simplifica haciendo un corte transversal e interpretando la sección con un modelo bidimensional. Por mucho tiempo se utilizaron figuras simples cuya atracción gravitacional podía obtenerse analíticamente; sin embargo, este proceder resultaba demasiado limitante por lo que se hizo necesario obtener soluciones para cuerpos de forma irregular. Dado que no existen soluciones analíticas para este tipo de cuerpos se utilizan modelos numéricos para evaluar la respuesta gravimétrica de estos cuerpos. Uno de estos métodos numéricos muy utilizado por su rapidez computacional es el debido a Talwani [1], que consiste en realizar una integración numérica de línea sobre una figura irregular. Por ser una

integración numérica la figura debe simplificarse a un polígono irregular con un número finito de aristas. En la practica el ajuste de los resultados del método de Talwani [1], de aquí en adelante simplificado a Método de Talwani, requiere de un ensayo de prueba y error variando el contraste de gravedad y la geometría del cuerpo hasta lograr un ajuste satisfactorio. Este procedimiento es laborioso y requiere de una inspección visual del ajuste para lograr el mejor modelo.

En la actualidad existen lenguajes eficientes que permiten realizar este procedimiento con notable economía haciendo uso de despliegues gráficos en introducción rápida de los datos. Así, en este proyecto se toma el método Talwani como base para elaborar una aplicación en lenguaje Java.

2.2. Método de solución

Existen varios lenguajes modernos de programación: C, C#, Python, Perl, JAVA, MATLAB, COBOL, por mencionar algunos. Cada uno de ellos tiene ventajas y desventajas de acuerdo con el trabajo al que se dediquen. Para el objetivo del método de Talwani y el uso que se intenta darle el lenguaje más adecuado es JAVA ya que se quiere que los distintos tipos de datos que se usan estén unidos a sus operaciones como se menciona más adelante, se pueda ejecutar en cualquier Sistema Operativo y permita la expresión gráfica de los resultados obtenidos. Java es un lenguaje orientado a objetos, tiene una independencia de la plataforma y con sus bibliotecas pueden añadirse gráficos en una aplicación de manera sencilla y útil.

El modelo numérico en que se basa la aplicación fue desarrollado originalmente por Talwani [1], y se codificaba usualmente en Fortran. Con el desarrollo de las PC modernas se puede lograr una mayor funcionalidad programando el algoritmo de Talwani en lenguaje Java. Teniendo los requerimientos definidos se puede generar en este lenguaje una aplicación que por medio de sus operadores permita generar resultados y por medio de bibliotecas expresarlos de forma gráfica. Eso permite ver los cambios en tiempo real y modificarlos los parámetros del modelo con gran rapidez para el ajuste deseado. La aplicación también puede desarrollarse de manera que se pueda guardar y leer archivos para una fácil obtención de los modelos gravimétricos.

2.3. Método Gravimétrico

El método gravimétrico se basa en la medición en la superficie de pequeñas variaciones del campo gravitacional. Las pequeñas diferencias o distorsiones en este campo de punto a punto sobre la superficie terrestre son causadas por variaciones laterales en la distribución de las masas en el interior de la Tierra. Las variaciones medidas se interpretan en términos de probables distribuciones de masa en el subsuelo, que son la base para inferir las posibles condiciones geológicas existentes [2]

Fundamentos (Ley de Newton y Unidades)

El método consiste básicamente en la medición de valores de la gravedad terrestre para determinar las desviaciones o anomalías del comportamiento normal del campo gravitacional, ocasionado por los cambios de densidad en los materiales del subsuelo. El fundamento fisicomatemático del método descansa en la “Ley de la Gravitación Universal de Newton”, la cual establece que la fuerza (F) de atracción mutua entre dos partículas de masa m1 y m2 es inversamente proporcional al cuadrado de la distancia (r) entre ellas. [2]

$$F = G\frac{m_1 m_2}{r^2}$$

Donde G es la constante de Gravitación Universal:

$$G = 6.67 \times 10^{-8} \frac{cm^3}{gr\ s^2}$$

La aceleración o atracción gravitacional es la fuerza de gravitación que actúa sobre una unidad de masa, y corresponde a la medición del campo gravitacional actuando en cualquier punto:

$$a = g = \frac{F}{m_2} = G\frac{m_1}{r^2}$$

En el Sistema Cegesimal de Unidades (CGS), la gravedad se mide en unidades de longitud sobre tiempo al cuadrado (cm/s^2) que equivale a un “Gal” (en honor a Galileo). Dado que la aceleración gravitatoria en la superficie terrestre es de 980 cm/s2 o 980 gales, y como las anomalías a determinar son del orden de una diezmillonésima de este valor, la unidad estándar en geofísica para gravimetría es el “miliGal” (mGal) que es igual a 10^{-6} gales. [2]

2.3.1. Ejemplo:

En este caso se procede calculando la atracción de diferentes cuerpos geométricos sobre una superficie dada. Notemos de paso que el problema de calcular la atracción de una masa anómala de densidad ρ_1, rodeada por una roca encajonante de densidad ρ_2 es equivalente a encontrar la atracción de una masa de densidad $\Delta\rho=\rho_2-\rho_1$ como se ilustra en la Figura 1.

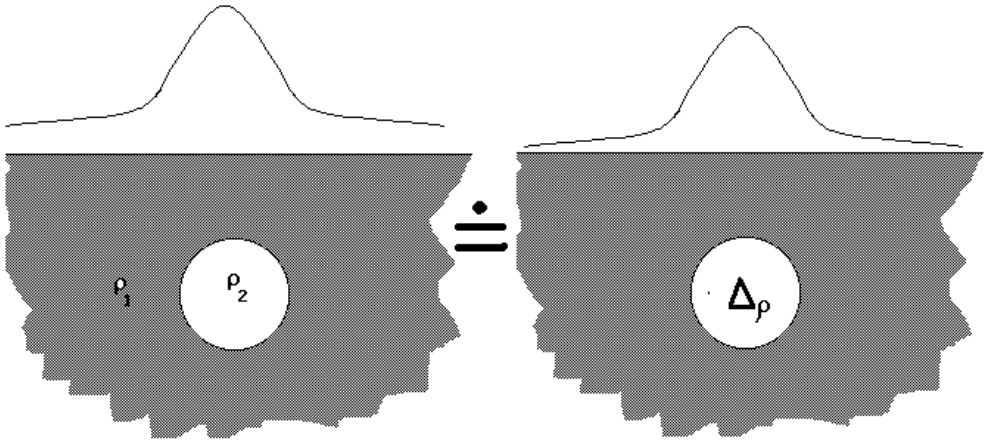


Fig. 1. Se muestra la Diferencial de Densidad equivalente $\Delta\rho=\rho_2-\rho_1$

La comparación entre estos resultados teóricos y los datos observados permite estimar las características de una estructura geológica. Los resultados teóricos se obtenían en el pasado de la solución analítica de la atracción de cuerpos geométricos. Este método aun se utiliza, pero resulta muy limitante en muchos casos en los que las rocas con contraste de densidad difícilmente pueden aproximarse por figuras geométricas simples. Los métodos para calcular la atracción de figuras geométricas irregulares se desarrollaron desde mediados del siglo pasado y permiten obtener modelos de densidad con mejor ajuste y facilidad de operación. Uno de estos métodos, debido a Talwani [1] es ampliamente utilizado y se expone a continuación dado que, como ha sido mencionado, fue utilizado para el presente trabajo.

2.4. Algoritmo de Talwani

Las expresiones se derivan para los componentes verticales y horizontales de la atracción gravitatoria debida a un cuerpo bidimensional de forma arbitraria mediante la aproximación a un polígono de ***n*** lados. Estas expresiones se ponen en formas adecuadas para la solución por computadora. Como un ejemplo de la aplicación de este método, la sección de la corteza a través de la zona de fractura Mendocino se deduce de las anomalías de la gravedad. Suponiendo que la corteza consiste en una única capa homogénea, cubierta por el agua y los sedimentos, se encontró que unos tres kilómetros de más grosor al norte de la zona de fractura que al sur de la misma. [1]

Muchas estructuras geológicas son aproximadamente lineales, y los problemas relacionados con ellos se pueden resolver con las formas de dos dimensiones de análisis. Para los cálculos de gravedad, Nettleton [2] ha determinado los criterios para hacer un cálculo de dos dimensiones adecuadas. Existen varios métodos para el cálculo de la atracción gravitacional causado por cuerpos bidimensionales de forma irregular. Estos métodos se pueden dividir en dos categorías. En la primera categoría se encuentran aquellos que involucran el uso de retículas, hacer listas, u otras ayudas gráficas de computación. Mientras que, en teoría, estos métodos se pueden hacer tan preciso como uno quiera, simplemente mediante el aumento de la escala a la que se construye la retícula, en la práctica real esto puede ser difícil, si no imposible.

En la segunda categoría se encuentran aquellos métodos que involucran destruir los cuerpos de forma irregular en varios cuerpos más pequeños de diferentes tamaños pero de formas que son regulares y para el que la atracción gravitacional se puede calcular fácilmente. Una forma conveniente de cuerpo regular para su uso es el bloque rectangular, según lo propuesto por Vening Meinesz [5]. Una vez más el método puede hacerse tan preciso como mejor parezca utilizando un número suficientemente grande de pequeños bloques. Sin embargo, los cálculos se hacen cada vez más tedioso como el número de bloques es aumentar Además, los bloques pueden ser tan pequeñas que sus contribuciones individuales en puntos distantes se descuidan a pesar de que la suma total de sus pequeñas contribuciones es apreciable. Esto puede causar considerables errores en los cálculos. La periferia de cualquier cuerpo de dos dimensiones se puede aproximar de cerca por un polígono, por lo que el número de lados de este polígono es suficientemente grande. Las expresiones analíticas se pueden obtener tanto para los componentes verticales y horizontales de la atracción gravitacional debido a este polígono en cualquier punto dado. Estas expresiones, a continuación, se pueden utilizar sin ningún tipo de limitaciones en el tamaño o la posición del cuerpo. El presente método implica el uso de estas expresiones. La precisión depende sólo de cómo el polígono se ajusta el cuerpo dado, y se puede aumentar mediante el aumento del número de lados del polígono. Se reconocerá que un cuerpo de dos dimensiones de forma irregular puede ser más fácilmente aproximada por un polígono que por bloques rectangulares. Los cálculos necesarios para la resolución de las expresiones para obtener los componentes de la atracción gravitatoria son largos y tediosos, pero al ser iterativo se programan

fácilmente para la solución por una computadora digital. Un programa para su uso con el IBM 650 se ha hecho, y el tiempo de máquina, necesario para la obtención tanto de los componentes verticales y horizontales de la atracción gravitatoria de un polígono de n lados en un solo punto es aproximadamente igual a 2,5 N segundos.

Sea un polígono ABCDEF (Fig. 2) de n lados y sea P el punto en que la atracción a este polígono tiene que ser determinada.

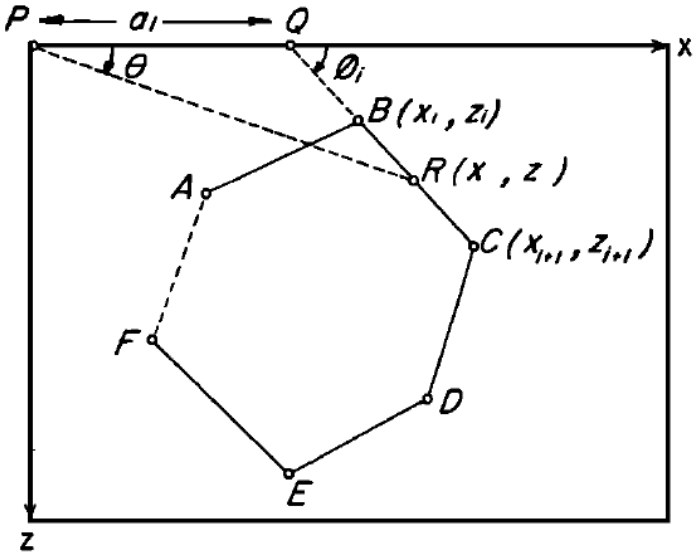


Fig. 2 .Elementos geométricos que intervienen en
la atracción gravitacional de un polígono con 'n' lados

Sea P el origen de un sistema de coordenadas XZ, donde el polígono también se encuentra en el plano XZ. Sea Z positivas hacia abajo y dejar que 0 se medirá desde el eje X positivo hacia el eje Z positivo.

Se ha demostrado por Hubbert [6] que la componente vertical de la atracción gravitacional debido a tal cuerpo de dos dimensiones es, en el origen, igual a

$$2G\rho \oint z d\theta,$$

La integral de línea está tomada a lo largo de su periferia, donde G es la constante universal de la gravitación y ρ es la densidad de volumen del cuerpo. Puede ser demostrada por un método similar al de Hubbert [6], de ese la expresión correspondiente para la horizontal se da la componente de atracción gravitatoria por $2G\rho \oint x d\theta$. [8]

Evaluemos las dos integrales $\oint z\ d\theta$ y $\oint x\ d\theta$ por el polígono de arriba. El aporte a $\oint z\ d\theta$, digamos, el lado *BC* del polígono se puede calcular en primer lugar. Producir *CB* para satisfacer el eje *x* en *Q* en un ángulo ϕ_i. Sea *PQ* = a_i. Ahora

$$z = x \tan\theta \qquad (1)$$

Para cualquier punto R arbitrario en BC. También

$$z = (x - a_i) \tan \phi_i \qquad (2)$$

La componente vertical de la atracción gravitatoria V y el componente H horizontal, debido a todo el polígono, vienen dadas respectivamente por

$$V = 2G\rho \sum_{i=1}^{n} Z_i \qquad (3)$$

y

$$H = 2G\rho \sum_{i=1}^{n} X_i, \qquad (4)$$

Las sumas que se realizan a través de los ‘n’ lados del polígono como se muestra en las ecuaciones (3) y (4).

Ahora queda por resolver las integrales que intervienen en la expresión para Z_i y X_i.

En el caso más general se puede demostrar que

$$Z_i = a_i \sin\phi_i \cos\phi_i \left[\theta_i - \theta_{i+1} + tan\ \phi_i \log_e \frac{\cos\theta_i(\tan\theta_i - \tan\ \phi_i)}{\cos\theta_{i+1}\tan\theta_{i+1}\tan\ \phi_i}\right] \qquad (5)$$

$$X_i = a_i \sin\phi_i \cos\phi_i \left[tan\ \phi_i(\theta_{i+1} - \theta_i) + \log_e \frac{\cos\theta_i(\tan\theta_i - \tan\ \phi_i)}{\cos\theta_{i+1}\tan\theta_{i+1}\tan\ \phi_i}\right] \qquad (6)$$

Dónde

$$\theta_i = \tan^{-1}\frac{z_i}{x_i},$$

$$\phi_i = \tan^{-1}\frac{z_{i+1} - z_i}{x_{i+1} - x_i},$$

$$\theta_{i+1} = \tan^{-1}\frac{z_{i+1}}{x_{i+1}},$$

y

$$a_i = x_{i+1} + z_{i+1}\frac{x_{i+1} - x_i}{z_i - z_{i+1}}$$

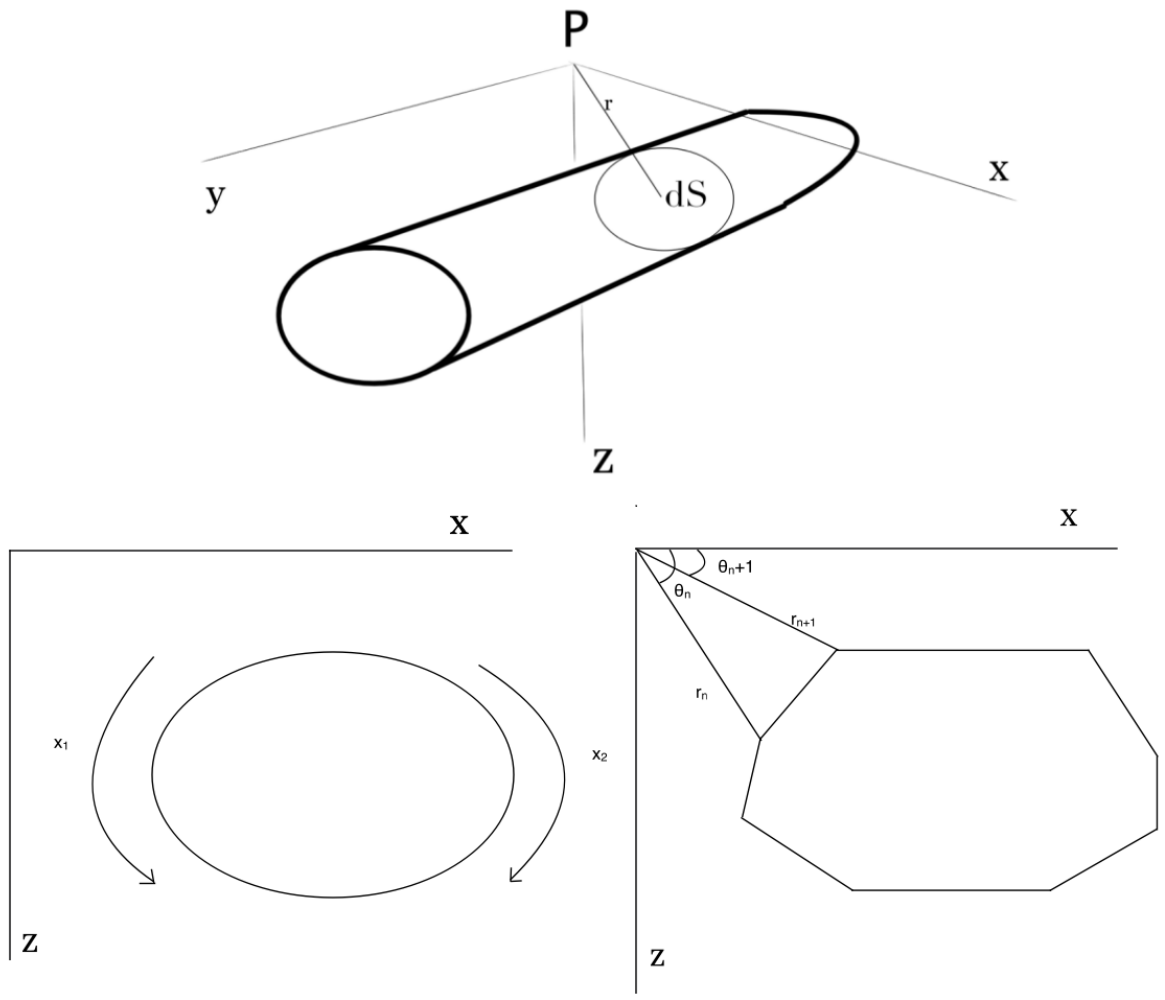


Fig. 3. Aproximación de cuerpo en dos dimensiones por un polígono de N lados

La siguiente derivación ofrece los mismos resultados que Talwani [1] de una manera ligeramente diferente, pero con muchas similitudes con la derivación de la sección anterior.

$$U = 2\gamma \int_S \rho(S) \log\frac{1}{r} dS, \qquad (7)$$

Donde la integración es sobre la superficie S de la sección transversal y en donde *r* es la distancia perpendicular a un elemento del cuerpo, dado por

$$r = \sqrt{(x - x')^2 + (z - z')^2}$$

Para simplificar las cosas, ahora movemos el punto de observación para el origen y sabemos que la densidad es constante. La atracción de gravedad vertical está dada por

$$g(P) = \frac{\delta U}{\delta z} = 2\gamma\rho \int\int \frac{z' dx' dz'}{x'^2 + z'^2} \qquad (8)$$

y la integración sobre los rendimientos *x'*

$$g = 2\,\gamma\rho \int \left[\tan^{-1}\frac{x'_2}{z'} - \tan^{-1}\frac{x'_1}{z'}\right] dz', \qquad (9)$$

Donde x'_1 y x'_2 son ambas funciones de *z'* y, como se muestra en la figura 3, representan caminos separados alrededor del perímetro de la cruz, la superficie de corte. Estos dos caminos parciales, en su conjunto y teniendo en cuenta el cambio de signo, la cantidad de una sola integración en sentido horario alrededor del perímetro, que es.

$$g = 2\gamma\rho \oint \tan^{-1}\frac{x'}{z'} dz'. \qquad (10)$$

Ahora reemplazamos el perímetro con un polígono de N lados de la ecuación 10 que se convierte en

$$g = 2\gamma\rho \sum_{n=1}^{N} \int_{z_n}^{z_{n+1}} \tan^{-1}\frac{x'}{z'} dz', \qquad (11)$$

Donde z_n y z_{n+1} son las coordenadas *z* de los dos extremos del lado *n*. Antes de continuar, tenemos una expresión para *x'* en términos de *z'*, y esta proporcionada por la ecuación de una línea recta:

$$x' = \alpha_n z' + \beta_n, \tag{12}$$

donde

$$\alpha_n = \frac{x_{n+1} - x_n}{z_{n+1} - z_n},$$

$$\beta_n = x_n - \alpha_n z_n.$$

Substituyendo la ecuación 12 en la ecuación 11 obtenemos

$$g = 2\gamma\rho \sum_{n=1}^{N} \int_{z_n}^{z_{n+1}} \tan^{-1} \frac{\alpha_n z' + \beta_n}{z'} dz'$$

$$g = 2\gamma\rho \sum_{n=1}^{N} \left\{ \frac{\pi}{2}(z_{n+1} - z_n) + \left(z_n \tan^{-1} \frac{z_n}{x_n} - z_{n+1} \tan^{-1} \frac{z_{n+1}}{x_{n+1}} \right) + \frac{\beta_n}{1 + \alpha_n^2} \left[log \frac{\sqrt{x_{n+1}^2 + z_{n+1}^2}}{\sqrt{x_n^2 + z_n^2}} - \alpha_n \left(\tan^{-1} \frac{z_{n+1}}{x_{n+1}} - \tan^{-1} \frac{z_n}{x_n} \right) \right] \right\}$$

Los dos primeros términos entre paréntesis de la sumatoria se suman a cero alrededor de cualquier polígono cerrado, por lo que la ecuación anterior se simplifica a la ecuación principal:

$$g = 2\gamma\rho \sum_{n=1}^{N} \frac{\beta_n}{1 + \alpha_n^2} \left[log \frac{r_{n+1}}{r_n} - \alpha_n((\theta_{n+1} - \theta_n) \right] \tag{13}$$

En tales casos donde r_n y θ_n se definen como se muestra en la figura 3

Capitulo III: Propuesta

3.1 Lenguajes de programación

En general, se puede clasificar a los lenguajes de programación en tres categorías: 1) Lenguajes compilados 2) Lenguajes interpretados 3) Híbridos entre los dos anteriores.

En la primera categoría, tenemos a lenguajes como Fortran o C/C++. Estos lenguajes se distinguen porque usan un compilador que traduce el código de alto nivel a código máquina. Este código máquina generado está en el ejecutable (en Windows se distingue por la extensión "*.exe*") y es dependiente de la máquina para la que se compiló. Es decir, si queremos correr un programa en un lenguaje compilado en varios sistemas, debemos compilar para cada uno de ellos. La principal ventaja de estos lenguajes es que una vez que se tiene el ejecutable este trabaja directamente con lenguaje máquina, por lo que la ejecución en general es más eficiente. En la segunda categoría tenemos a los lenguajes interpretados, los cuales son compilados en tiempo de ejecución línea por línea. Son lenguajes en los que al ejecutar un programa cada línea es compilada y luego ejecutada. Ejemplos de lenguajes interpretados son Smalltalk, JavaScript y PHP. Estos lenguajes usan un intérprete que va ejecutando los programas línea por línea. La ventaja de esto es que el programa se vuelve independiente de la plataforma, por lo que puede correr en cualquier sistema que tenga un intérprete para el lenguaje en el que está escrito el programa. Sin embargo, estos lenguajes son en general

menos eficientes, pues cada instrucción tiene que ser interpretada antes de ejecutarse. La tercera categoría es cuando se utilizan técnicas para hacer más eficientes los lenguajes interpretados. La mayoría de los lenguajes catalogados como interpretados usan ahora estas técnicas para solucionar el problema de la eficiencia. Generalmente lo que hacen es combinar la compilación con la interpretación. Por ejemplo, Java traduce el código a un código intermedio (bytecode) que interpreta la Máquina Virtual de Java (JVM), pero también usa la Compilación Justo a Tiempo (Just-in-time Compilation o JIT Compilation) para pasar a código máquina secciones de código que se usan regularmente. Los lenguajes de .NET son otro ejemplo; en dichos lenguajes se compila a código intermedio (Common Intermediate Language o CIL) que el intérprete CLR (Common Language Runtime) compila en tiempo de ejecución.

Se pudo utilizar cualquiera de los siguientes lenguajes.

- **Fortran**

Fue desarrollado por IBM para usarse en aplicaciones científicas y de ingeniería con cálculos matemáticos complejos.

Optimizado para realizar operaciones sobre arreglos.

Popular en el ámbito científico-

Permite cómputo en paralelo.

Múltiples bibliotecas para uso científico.

- **C**

Popular por ser el lenguaje de desarrollo del sistema operativo UNIX. Se usa generalmente para programar los sistemas operativos de propósito general.

Lenguaje estructurado.

Eficiencia en cálculos al ser un lenguaje compilado.

Permite manipulación directa de la memoria con el uso de apuntadores.

Existen muchas bibliotecas de funciones en este lenguaje

- **Java**

Lenguaje de programación orientado a objetos y multiplataforma.

Programación Orientada a Objetos.

Multiplataforma, seguro y Robusto.

La deficiencia en Fortran es la falta de una interfaz gráfica amigable para el usuario. Es por ello que para la elección del lenguaje de programación se tomará en cuenta principalmente este punto. La carencia de interfaces gráficas en los programas de uso científico es muy común. Los científicos en su mayoría programan en lenguajes como C/C++ o Fortran para realizar programas que los apoyen en sus investigaciones. Estos lenguajes son los preferidos en el ámbito científico debido a que son lenguajes compilados, por lo que son más eficientes

para programas que realizan muchas operaciones matemáticas. Existen bibliotecas gráficas en lenguajes como C y Fortran que podrían permitir a los científicos dar mayor vistosidad a los programas con el uso de interfaces gráficas. Sin embargo, hay poca información sobre la implementación de interfaces con estos lenguajes, sus interfaces no son tan atractivas gráficamente y son más complicadas de implementar. Es por eso que estos lenguajes se descartan para la implementación de la interfaz gráfica.

Java tiene la ventaja de estar basados en el paradigma de Programación Orientada a Objetos (POO). Ésta es una gran ventaja pues la POO ayuda al programa a ser fácil de mantener y de hacer modificaciones. Además debido a la popularidad de este lenguaje, se cuenta con entornos de desarrollo muy potentes que pueden ser usados para acelerar aún más el proceso de la creación de interfaces gráficas

3.2. Lenguaje Java

El lenguaje de programación Java fue liberado por Sun en 1995. Es un lenguaje de programación concurrente, basado en clases y orientado a objetos. Fue creado con el lema “write once, run anywhere” (WORA), es decir, está pensado para que los programas corran en cualquier máquina sin la necesidad de compilar para cada una de ellas. Para esto se usa la Java Virtual Machine (JVM) que tiene que estar instalada en la máquina en donde se desee correr un programa de Java.

A continuación se muestran algunas estadísticas que publica Oracle en la página oficial de Java *(http://www.java.com/es/about)*

- El 97% de los escritorios empresariales ejecutan Java
- El 89% de los escritorios (o computadoras) en Estados Unidos ejecutan Java
- 9 millones de desarrolladores de Java en todo el mundo
- La primera opción para los desarrolladores
- La primera plataforma de desarrollo
- 3 mil millones de teléfonos móviles ejecutan Java
- El 100% de los reproductores de Blu-ray incluyen Java
- 5 mil millones de Java Cards en uso
- 125 millones de dispositivos de televisión ejecutan Java
- 5 de los 5 principales fabricantes de equipos originales utilizan Java ME

El proyecto del lenguaje Java fue iniciado en 1991 por James Gosling. Mike Sheridan y Patrick Naughton. Su nombre inicial fue Oak, pero al final fue llamado

Java, nombre tomado del café al que muy a menudo iban los creadores del lenguaje. La primera versión liberada fue Java 1.0 en 1995. En 1998 se liberó la versión de Java 2, a partir de la cual ya había versiones diferentes para diferentes tipos de plataformas: J2SE (Standard Edition), J2EE (Enterprise Edition) y J2ME (Micro Edition). Entre 2006 y 2007 Sun liberó la mayor parte del código de Java usando la licencia GNU GLP (General Public License) con lo que lo hizo Free and Open Source Software (FOSS). Entre el 2009 y el 2010 Oracle adquirió la compañía de Sun Microsystems con lo que adquirió entre otras tecnologías a Java.

3.3. Programación Orientada a Objetos (POO)

Se trata de un paradigma que tiene como objetivo permitir el diseño de sistemas de software modulares y reutilizables. El paradigma orientado a objetos está basado en conceptos como la programación estructurada y los tipos abstractos de datos. Un objeto es un tipo abstracto de dato que tiene como agregados el polimorfismo y la herencia. En la programación estructura se hace una separación entre código y datos, cuando en el programación orientada a objetos ambos conceptos se unen para forma un objeto, el cual tiene un estado (atributos) y un comportamiento (métodos). Algunas de las ventajas que ofrece la programación orientada a objetos son:

- Permite definir el programa de una manera más natural con el uso de objetos que describen el problema
- Facilitar el mantenimiento del software al hacer programas más claros.
- Facilitar la evolución, extensión o mejoramiento del software por su modularidad.
- Fomenta la reutilización del código al programarse en módulos que pueden ser usados por otros programas.
- Ahorra tiempo de desarrollo al permitir desarrollar en paralelo módulos independientes del software.

Algunos conceptos relacionados a la programación orientada a objetos son:
Clase.- Definición de los atributos y el comportamiento de un tipo de objeto.

Instancia.- Se trata de la creación de un objeto a partir de las reglas definidas en la clase.

Atributos.- Son las características del objeto (datos) que se definen en la clase.

Métodos.- Son subrutinas asociadas a un objeto y que se definen en la clase para describir el comportamiento del objeto.

Abstracción.- Es la definición de un objeto en base a sus características esenciales (atributos y comportamiento). La abstracción es una de las características más importantes en el diseño orientado a objetos, pues permite modelar un problema en base a un conjunto de clases.

Modularidad.- Al crear clases, el software se separa naturalmente en módulos, que pueden ser utilizados como parte de varios programas.

Mensajes.- Es la comunicación hacia un objeto que le indica que ejecute un método con los parámetros indicados en el mensaje.

Las características básicas que un lenguaje de programación debe tener para entrar en el paradigma orientado a objetos son: encapsulamiento, herencia y polimorfismo. [9]

3.3.1 Encapsulamiento

El encapsulamiento se refiere a agregar dentro de un objeto todos los recursos necesarios para que el objeto funcione, esto es, agregar los métodos y los atributos que requiere. La idea del encapsulamiento es que el código no influencie en otro código, ni sea influenciado por otro código. Las clases permiten que los datos y los métodos estén estrechamente ligados, formando una “cápsula” congruente con el objeto que se está definiendo. El encapsulamiento es usado para esconder los valores de los atributos dentro de una clase, de modo que sólo el objeto pueda tener acceso a ellas y no lo tengan otros objetos externos. Para esto los atributos deben ser definidos como privados, es decir, sólo los métodos de la clase pueden acceder a ellos o modificarlos. Para acceder a los atributos desde otras clases se proporcionan en la clase métodos específicos para esto:

- Para consultar el valor de un atributo desde una clase externa se tiene que definir un método get para este atributo.
- Para modificar el valor de un atributo desde una clase externa se tiene que definir un método set para este atributo. [9]

3.3.2. Herencia

La herencia es una de las características más poderosas de la programación orientada a objetos, pues fomenta la reutilización del código. La herencia se refiere a que se pueden crear clases que son hijas de otra clase. A la clase padre se le llama superclase, la cual contiene atributos y métodos comunes para todas las clases hijas. Las clases hijas se llaman subclases y son las que heredan estos elementos comunes de las superclases, pero contienen también atributos y métodos propios. De esta manera, se puede ahorrar mucho código por ejemplo al implementar métodos una sola vez en la superclase, además de que el código es más fácil de mantener, pues sólo se modifica en una parte y esta modificación pasa automáticamente a las subclases. Es importante mencionar que los métodos heredados por las subclases pueden ser reimplementados por ellas para que se adecuen a la subclase en específico. [9]

3.3.3. Polimorfismo

El polimorfismo está muy ligado a la herencia. El polimorfismo se refiere a que dentro del código se puede tratar a todas las subclases como si se tratara de su superclase. Esto simplifica mucho la programación y la hace más clara. El enfoque que se da es el de tratar los problemas de manera general y no de manera específica.

Las mayores ventajas de este paradigma de programación son brindar claridad al código, de manera que sea más fácil desarrollar y mantener aplicaciones. [9]

3.4. Ambiente de desarrollo

Para este proyecto se usará Netbeans, pues facilita en gran medida la creación de interfaces gráficas.

Es un entorno de desarrollo escrito en Java que permite programar en múltiples lenguajes de programación. Cuenta con diferentes Plugins que se pueden descargar dependiendo de las herramientas que necesitemos para nuestro desarrollo.

3.5. Arquitectura – Modelo Vista Controlador

Para la arquitectura del sistema, se eligió la arquitectura Modelo Vista Controlador (MVC). Esta arquitectura está pensada para el desarrollo de interfaces gráficas de usuario. Con esta arquitectura, se puede llevar un mejor control de los elementos que componen al sistema, de modo que sea más fácil el desarrollo y mantenimiento del mismo. Este tipo de arquitectura o patrón de diseño está muy ligado a la programación orientada a objetos, pues se basa en la abstracción y también en la reutilización de código.

El MVC tiene tres componentes que interactúan entre sí: el modelo, la vista y el controlador

Modelo - Representa los datos de la aplicación y las reglas, la lógica y las funciones. El modelo notifica a la vista y al controlador cuando hay un cambio en

el estado de la aplicación. que gobiernan el acceso a y la actualización de los datos.

Vista - Especifica cómo deben ser presentados los datos del modelo. Si los datos del modelo cambian, la vista debe actualizar la presentación de los datos en el momento. Para esto se puede usar un push model, donde el modelo manda los cambios en todo momento, o un pull model, donde la vista es responsable de llamar al modelo cuando necesite actualizar a los datos más recientes.

Controlador - Se encarga de interpretar las acciones que el modelo debe efectuar, cuando el usuario interactúa con la Vista.

Modelo

Contiene el núcleo de la funcionalidad (dominio) de la aplicación.

Encapsula el estado de la aplicación.

A continuación se muestra el modelo en donde se especifica cada botón, campo y sección del programa:

1) Datos Generales - (Sección para el ingreso de los datos generales)
    a) X Inicial – (Punto de origen donde se iniciara la medición)
    b) N° Ops.- (Número de puntos de observación, se mide Kilómetros)
    c) Dx – (Diferencial de *X,* distancia entre cada punto de observación, se mide en Kilómetros)
    d) N° Cuerpos - (Cantidad de cuerpos que se medirá su gravedad)

2) Datos de Cuerpo - (Sección donde se especifican los datos de cada cuerpo)
    a) Densidad - (Densidad del cuerpo a medir)
    b) Número de aristas - (Número de Aristas que tiene cada cuerpo)
    c) Aristas - (Coordenadas [x, y] de cada una de las aristas del cuerpo)
3) Guardar – (Campo que guarda los datos ingresados para su posterior uso)
4) Calcular – (Campo que utiliza los datos para su cálculo de gravedad)
5) Borrar – (Campo que borra los Datos ingresado para un nuevo cálculo)
6) Comparar – (Campo que abre una ventana para realizar la búsqueda de un archivo con datos guardado y lo compare con datos ya cargados)
7) Examinar – (Campo que abre una ventana para realizar la búsqueda de un archivo con datos guardados y calcule la gravedad con el algoritmo)
8) Resultados – (Sección que muestra los datos calculados con el algoritmo de Talwani)
    a) Gráfica – (Ventana que despliega los resultados en una gráfica)
    b) Cuerpos - (Ventana que despliega los cuerpos utilizados para calcular la gravedad)

Vista

Es la presentación del Modelo.

Puede acceder al Modelo pero nunca cambiar su estado.

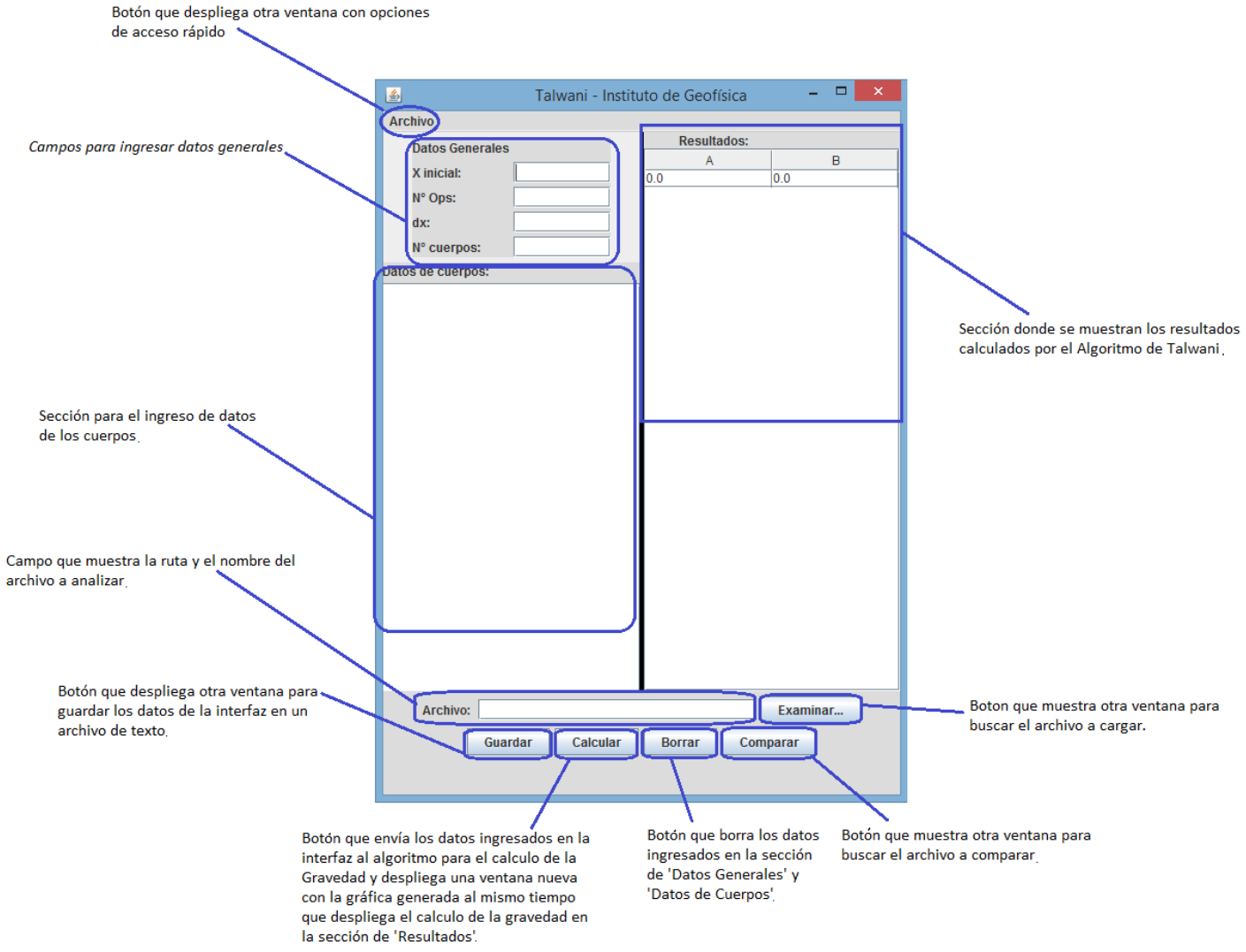


Fig. 4. Ventana principal que muestra la función de cada una de las secciones de la aplicación.

Controlador

Código que obtiene datos dinámicamente y genera el contenido gráficamente por medio del algoritmo de Talwani

```
package talwaniunam.modelo;

public class Algoritmo
{
	static double GAMMA = 6.67e-11, SI2MG = 1e5, KM2M = 1e3, DENOM0 = 1e-6;

	int nCuerpos, nOps;
	double xInicial, dx;

	static double PRECALCULO;

	double[][] resultado;

	/*
	 * Constructor que inicializa todos los campos necesarios para el desarrollo
	 * del algoritmo
	 */
	public Algoritmo(int nCuerpos, int nOps, double xInicial, double dx)
	{
		this.nCuerpos = nCuerpos;
		this.nOps = nOps;
		this.xInicial = xInicial;
		this.dx = dx;
		resultado = new double[nOps + 1][2];

		PRECALCULO = GAMMA * SI2MG * KM2M;
	}

	public double[][] calcula(Cuerpo cuerpo)
	{

		double xOb = xInicial;

		for (int i = 0; i <= nOps; i++)
		{
			resultado[i][0] = xOb;
			resultado[i][1] = 2 * PRECALCULO * cuerpo.getDensidad() *
gravedadCuerpo(cuerpo, xOb);

			xOb += dx;
		}

		return resultado;
	}

	public double[][] calcula(Cuerpo[] cuerpos)
	{
		double[][] temporal;

		for (int i = 0; i < resultado.length; i++)
			for (int j = 0; j < resultado[0].length; j++)
				resultado[i][j] = 0;

		for (int i = 0; i < cuerpos.length; i++)
		{
			temporal = calcula(cuerpos[i]);

			for (int j = 0; j < temporal.length; j++)
```

```
                                        resultado[j][1] += temporal[j][1];
                }

                return resultado;
        }

        private double gravedadCuerpo(Cuerpo cuerpo, double xOb)
        {
                double suma = 0;

                for(int n = 0; n < cuerpo.getNumeroAristas(); n++)
                {
                        int n2;
                        double x1, x2, z1, z2;

                        n2 = (n == cuerpo.getNumeroAristas() - 1) ? 0 : n + 1; // If-else version
PRO

                        x1 = cuerpo.getArista(n)[0] - xOb;
                        z1 = cuerpo.getArista(n)[1];

                        //System.out.println("n: " + n + "\nn2: " + n2);
                        x2 = cuerpo.getArista(n2)[0] - xOb;
                        z2 = cuerpo.getArista(n2)[1];

                        double denom;

                        denom = (z2 - z1 == 0) ? DENOM0 : z2 -z1;
                        //System.out.println("z2: " + z2 + "z1: " + z1 + "denom: " + denom);

                        double alfa = (x2 - x1) / denom;
                        double beta = (x1 * z2 - x2 * z1) / denom;

                        double factor = beta / (1 + alfa * alfa);

                        double r1sq = x1 * x1 + z1 * z1;
                        double r2sq = x2 * x2 + z2 * z2;
                        double term1 = 0.5 * (Math.log(r2sq) - Math.log(r1sq) );
                        double term2 = Math.atan2(z2,x2) - Math.atan2(z1,x1);

                        suma += factor * (term1 - alfa * term2);
                        //System.out.println("Suma: " + suma);

                } // Fin de <for(i)>

                // System.out.println("Cuerpo calculado, SUMA: " +  suma);
                return suma;
        } // Fin de <gravedadCuerpo>
```

Esta función se es tomado a partir de la ecuación del algoritmo de Talwani::

$$g = 2\gamma\rho \sum_{n=1}^{N} \frac{\beta_n}{1+\alpha_n^2}\left[log\frac{r_{n+1}}{r_n} - \alpha_n((\theta_{n+1} - \theta_n)\right]$$

Debido a que se trata de un patrón de diseño abstracto, es posible implementarlo de diferentes maneras. En Java, una de las implementaciones que se puede hacer es creando directamente las clases para el Modelo, la Vista y el Controlador. Esto aplicaría para un proyecto pequeño, en el que no hay muchos elementos que manejar y en el que fueran suficientes estas tres clases para todo el sistema. Pero para un sistema más grande, este enfoque no sería conveniente.

3.6. Desarrollo

Para hacer un desarrollo efectivo de un programa, no sólo es necesario saber programar, también es importante tener conocimiento de las técnicas de la ingeniería de software. La ingeniería de software es la aplicación de la ingeniería al diseño, desarrollo y mantenimiento del software. Se podría decir que es el conjunto de conocimientos que se han ido adquiriendo en las recientes décadas sobre las mejores prácticas para el desarrollo de software. Desde el punto de vista de la ingeniería de software, es importante elegir un modelo de desarrollo antes de empezar a escribir el código. Un modelo de desarrollo es una serie de actividades relacionadas que van a llevar a la producción de un producto de software. Dentro de los diferentes modelos que existen, todos deben incluir las siguientes cuatro actividades:

- Especificación del software: también llamada la etapa de análisis, es donde se identifican los problemas que han de resolverse y se hace una especificación de requerimientos.

- Diseño e implementación del software: en esta etapa se crea el producto de software. En ocasiones es más claro separar esta etapa en dos, para identificar lo que se debe hacer en cada etapa.
- Validación del software: se llama validación a las pruebas que hace el usuario final (cliente) al software para asegurarse de que cumple con los requerimientos.
- Evolución del software: esta etapa es llamada también de operación y mantenimiento. Conforme el software se use puede haber modificaciones para cumplir requerimientos que no se habían especificado en un principio.

3.7. Método Cascada

El modelo que se utilizará en este proyecto es el modelo de cascada. En este modelo se planean las actividades desde un principio y una vez definido todo el proceso, se desarrollan las actividades. En este modelo las actividades de desarrollo fundamentales son:

- Análisis y definición de requerimientos: Se debe hacer un análisis a detalle de los requerimientos del sistema. En esta etapa es importante consultar al usuario final del sistema para identificar sus necesidades y poder hacer una especificación de requerimientos detallada.

- Diseño del sistema: usando la especificación de requerimientos se diseña el sistema tomando en cuenta los requerimientos de hardware y software. En esta etapa se diseña la arquitectura del sistema, de modo que al iniciar con el desarrollo (implementación) se tenga un esquema sobre el cual trabajar.

- Integración y pruebas del sistema: en esta etapa ya todos los elementos se han probado individualmente y se unen para formar el sistema.

- En esta etapa se prueba el sistema completo y se corrigen errores que se pudieran producir durante la integración. Después de las pruebas se libera el software al cliente o usuario final.

- Operación y mantenimiento: en esta etapa el software es usado por el usuario final, que es a lo que se refiere la operación. El mantenimiento implica la corrección de errores o identificación de requerimientos que no fueron descubiertos durante las otras etapas.

3.8. Control de versiones

Durante todas las etapas del proceso de desarrollo es importante documentar los cambios que se van haciendo conforme se va avanzando en el proyecto. De esta manera se puede cuantificar los avances conforme transcurre el tiempo y saber si se tendrá el software en el tiempo estimado. Esto se puede realizar simplemente llevando una bitácora de los cambios que se van haciendo, aunque hay herramientas que permiten hacer esto y agregan otras características interesantes. A este tipo de herramientas se les llama sistemas de control de versiones o System Version Control (SVC) en inglés. Con estos sistemas podemos ir documentando el avance del proyecto, pero además nos permite hacer un regreso a una versión anterior en caso de que se requiera. Entre los SVCs más populares tenemos:

- CVS (Control Version System): muy popular para desarrollo de software libre. Usa la licencia GPL.
- Subversion: esta herramienta es de código abierto y muy popular para el desarrollo de software de código abierto
- Team Foundation Version Control: es una herramienta que viene integrada en la suite de desarrollo de Visual Studio de Microsoft.
- Bazaar: esta herramienta es patrocinada por canonical ltd, la empresa creadora de Ubuntu.
- Git: este SVC fue diseñado por Linus Torvalds, el creador del núcleo de Linux, pensado para proyectos de código abierto en donde el número de archivos es muy extenso y también para el desarrollo colaborativo.

3.8.1 GIT

Para lograr una administración del programa, se eligió GIT para llevar a cabo un buen manejo versiones. En cada clase, estructura o función se guarda un avance de tal forma que si se quiere modificar cualquier parte del código, solo con un commit se regresa a esa parte de la versión y se modifica. [4]

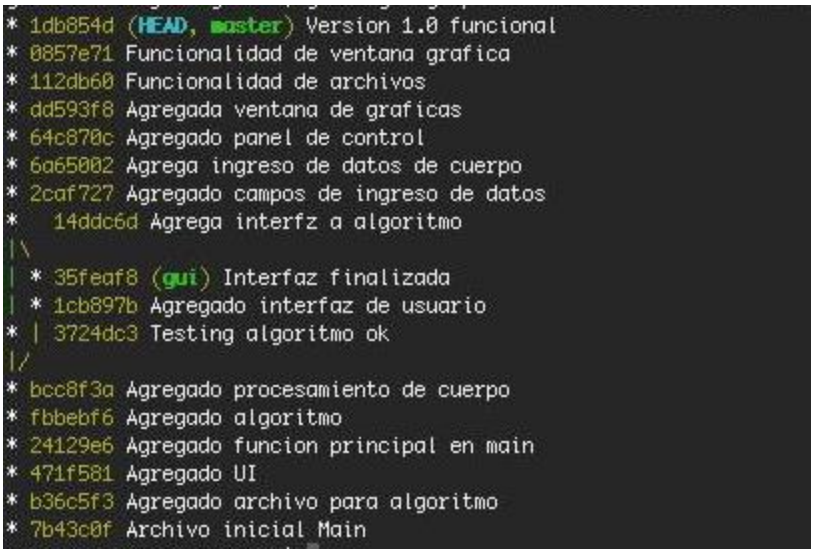


Fig. 7. Control de versiones centralizado.

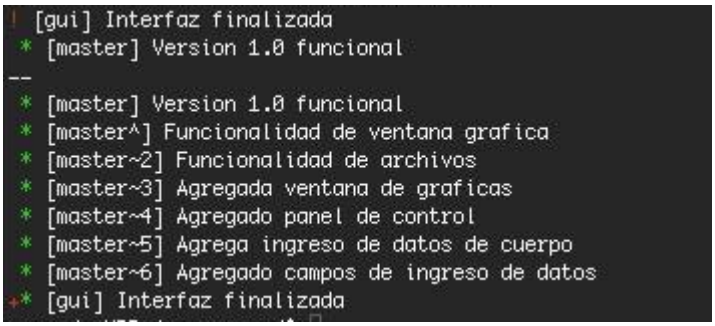


Fig. 8. Diagrama de control de versiones centralizado

Al consultar el histórico de las versiones se puede ver la descripción que se puso y las diferencias entre los archivos de esa versión y la versión anterior. Si se usa un repositorio, todos estos datos se podrán consultar también desde ahí.

Utilidades del control de versiones GIT

-Diseñado para manejar proyectos grandes.

- Es extremadamente rápido.

- Autentificación criptográfica del historial.

- Formato de archivo muy sencillo y compacto.

- Se puede sincronizar por cualquier medio

3.9. Modelo SCRUM

Es una manera para el usuario trabaje para desarrollar un producto para construir exactamente y sólo lo que se necesita.

Más específicamente, Scrum es un marco sencillo para la organización eficaz en proyectos complejos, proporciona un pequeño conjunto de reglas que crean suficiente estructura para que los equipos puedan centrar su innovación. [8]

Backlog

Tarea	Descripción	Tiempo (horas)	Sprint	Semana 1	Semana 2	Semana 3	Semana 4	semana 5	Semana 6
Main	Función principal para generar el programa	1	1	X					
Algoritmo	Implementación del algoritmo en java	40	1	X					
Cuerpo	implementación para procesar cuerpo	20	1		x				
Testing en consola	Verificar que los algoritmos, y cuerpo funcionen correctamente en consola	5	1		x				
Interfaz	Implementar interfaz gráfica para el programa	10	2		x				
Datos Generales	Implementar funcionalidad de ingreso de datos generales	10	2		x	x			
Datos de cuerpos	Implementar funcionalidad de panel de datos de cuerpos	10	2			x			
Resultados	Implementar funcionalidad de panel de resultados	10	2			x			
Testing de interfaz	Testing de ingreso de datos en interfaz gráfica	5	2			x			
Panel de control	Implementacion de sección en interfaz para abrir y guardar archivos, así como limpiar datos, comparar y calcular	20	3			x	x		
Ventana de gráficos	Implementación de ventana para mostrar gráficos	20	3				x		
Archivos	Implementa funcionalidad para manejar los archivos en el programa	20	3				x	x	
gráficas	implementa funcionalidad para generar gráficas en ventana de gráficos	30	4					x	
Testing	Prueba de funcionamiento general	12	4						x
		213							

Fig. 5. Cuadro que muestra los tiempos en realizar cada sección del programa

Burndown Chart

Gráfica que muestra los datos obtenidos en el Backlog del modelo Scrum y va reduciendo las horas en las que se realizó el programa.

Cada Sprint es una fase de la programación como se muestra en el Backlog y se puede llegar a la conclusión que se concluyó el programa a tiempo para el inicio de cursos.

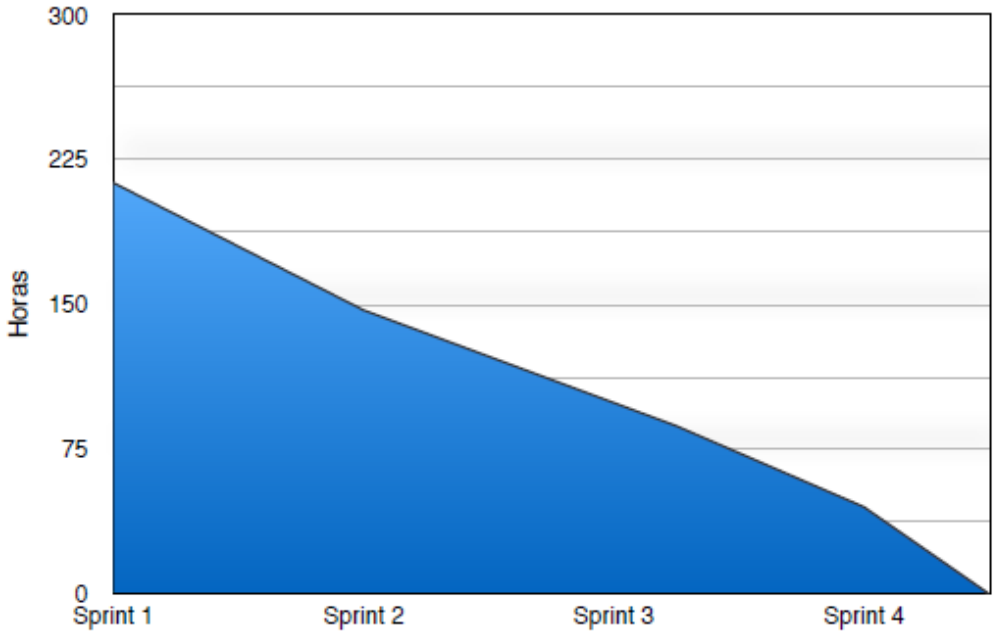


Fig. 6. Muestra la velocidad a la que se está completando los objetivos/requisitos

Capítulo IV: Análisis y Desarrollo del Sistema

4.1. Requerimientos

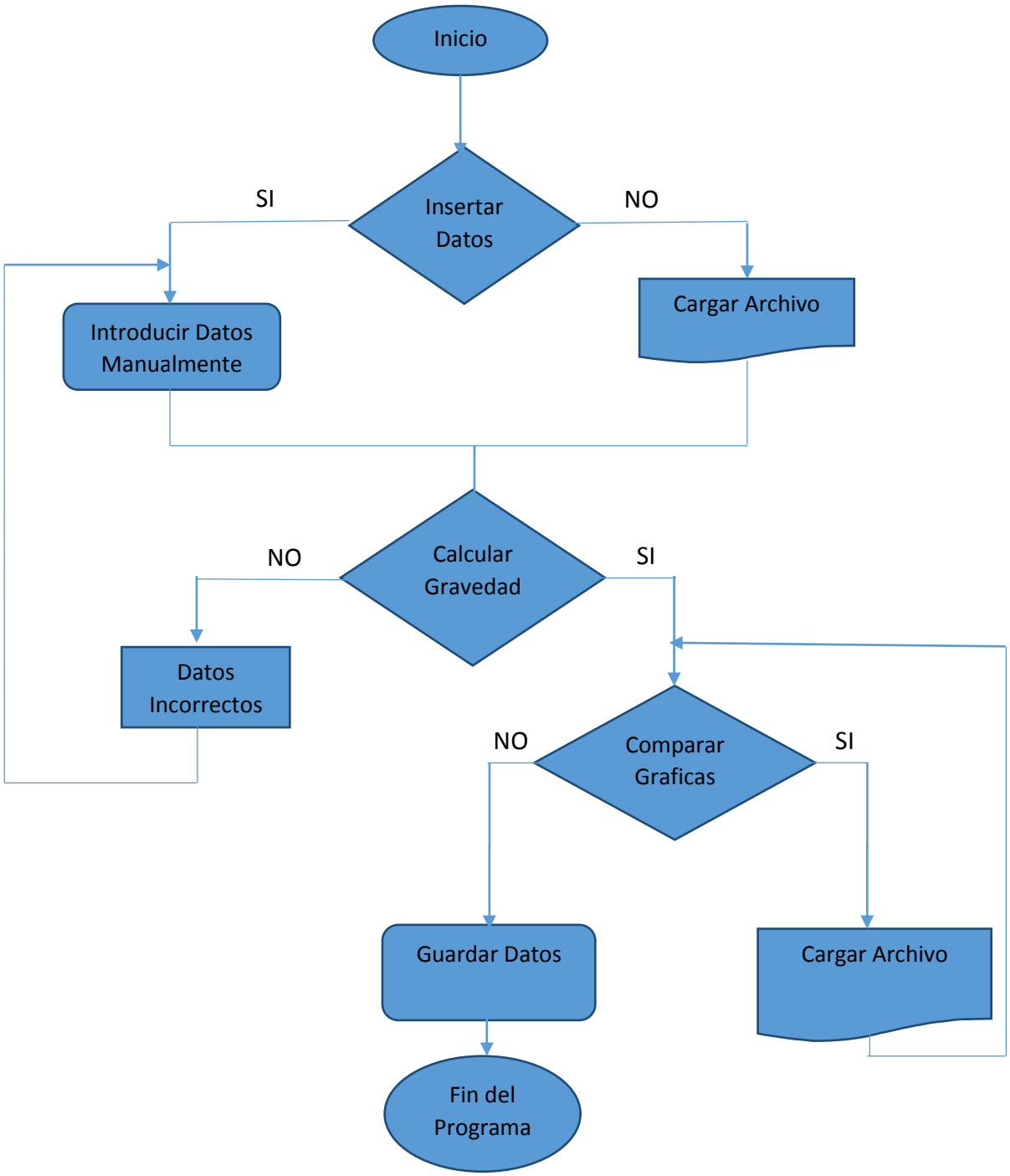
Los requerimientos específicos fueron los siguientes:

- Interfaz gráfica desde la cual se puedan introducir los datos y observar los resultados de los cálculos.
- Diagrama a escala en el que se pueda observar los polígonos y la gráfica de resultados usando los datos calculados por el programa.
- Posibilidad de guardar datos de entrada en un archivo para poder cargarlos en otro momento en que se ejecutara la aplicación.
- Funcionalidad para exportar la gráfica y los polígonos en un archivo de imagen.
- Funcionalidad para poder comparar dos lecturas diferentes en una misma gráfica.
- Poder modificar los datos en tiempo real para poder visualizar cambios en la gráfica así como en los polígonos.
- Manual de usuario para entender lo que hace el sistema.

4.2. Diagrama de flujo

Diagrama de flujo del programa ADGAVO (Análisis de Datos Gravimétricos en Áreas Volcánicas).

En el diagrama de flujo se muestra la funcionalidad de la aplicación paso a paso



4.3. Manual de Usuario

ADGAVO es una aplicación de fácil uso e interacción con los usuarios con un manejo de datos en tiempo real y una óptima forma de mostrar datos gráficamente.

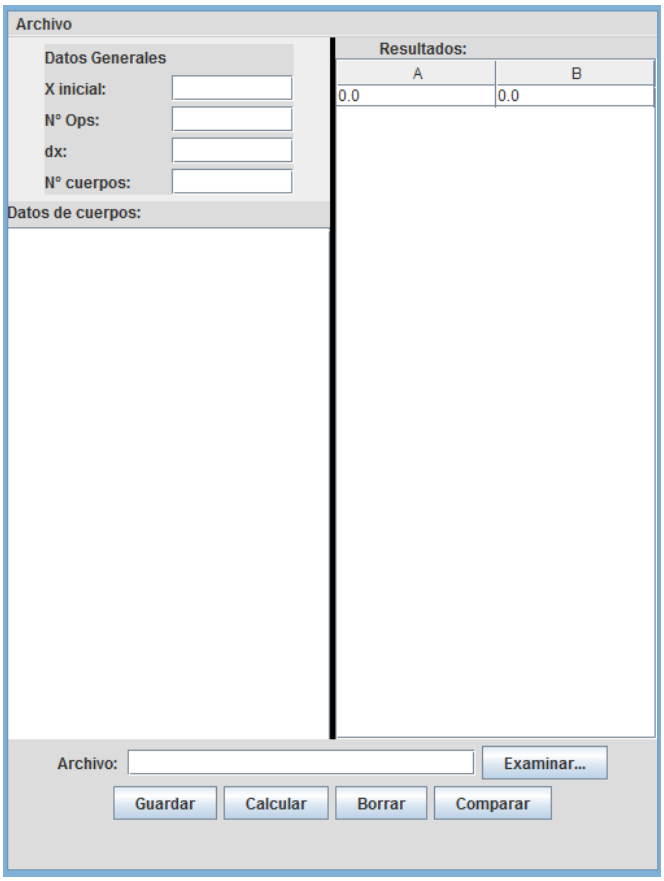


Fig. 9. Ventana principal de la aplicación

Datos Generales.

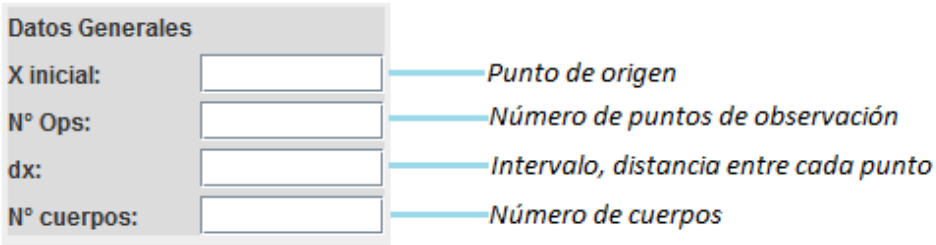


Fig. 10. Sección de ingreso de Datos Generales

4.4. Pruebas

Un cuerpo con 10 puntos de observación. Cada punto tendrá un intervalo de 200 km. Con un origen en cero.

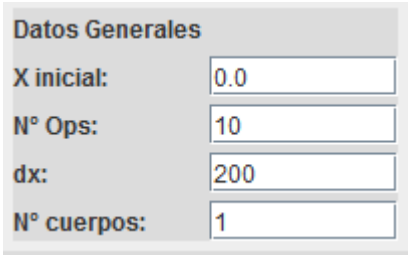


Fig. 11. Prueba de llenado de datos, de la sección 'Datos Generales'

Datos de Cuerpos.

Fig. 12. Sección en dónde se especifican los datos de los cuerpos

*Es necesario aumentar el número de aristas + 1, para cerrar el polígono o cuerpo a medir utilizando la primera Arista.

Prueba 1: Se ingresan los datos para calcular la gravedad de un cuerpo alrededor de un volcán y determinar su posible hundimiento a partir del cálculo.

Un cuerpo de 4 Aristas, con las coordenadas (100,100), (200,100), (200,200) y (100,200).

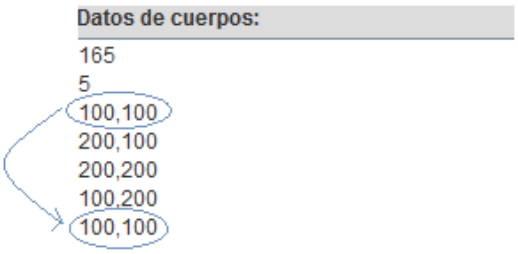


Fig. 13. Prueba de ingreso de datos en la sección Datos de Cuerpo

Incluyendo los 'Datos Generales' y 'Datos de Cuerpos' se procede a ***Calcular.*** Fig. 19.

Resultados.

Fig. 14. Sección donde se muestran los resultados obtenidos

Utilizando los datos de las pruebas anteriores se muestra el resultado calculado.

Resultados:

A	B
0.0	73.43063482216458
200.0	132.07533843933714
400.0	38.83551304431696
600.0	14.672474767611359
800.0	7.419187529641219
1000.0	4.431687018865752
1200.0	2.934782810281321
1400.0	2.0830539337000373
1600.0	1.553715349359154
1800.0	1.2027859943215886
2000.0	0.9583886679831843

Fig. 15. Resultados obtenidos mediante el Algoritmo de Talwani

Gráfica.

Realizando el cálculo se muestra automáticamente la gráfica con los resultados obtenidos.

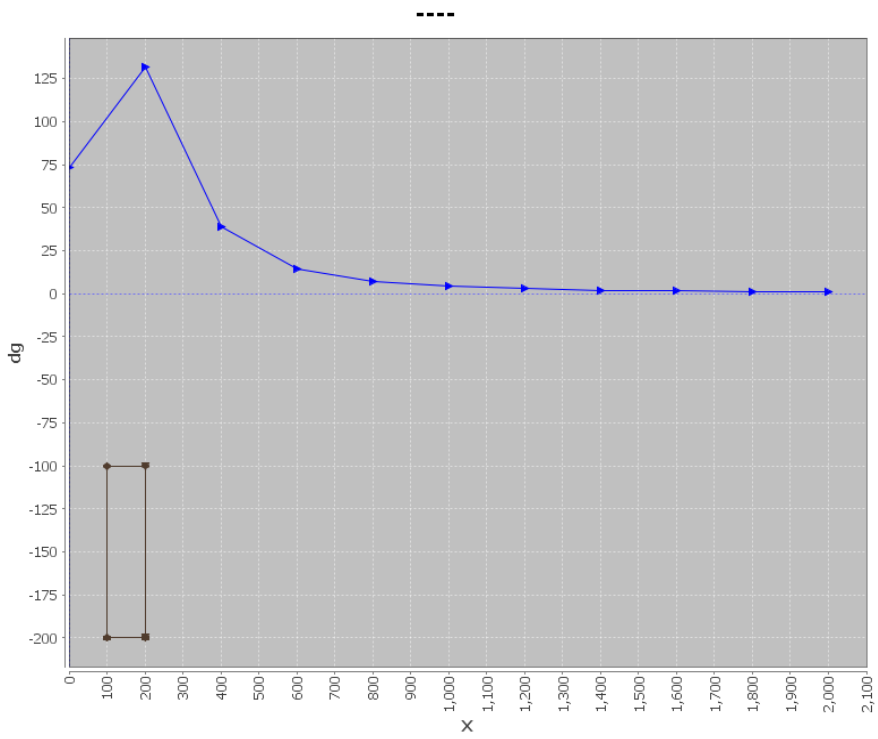


Fig. 16. Ventana que muestra la gráfica obtenida de la sección de 'Resultados' y cuerpos de la sección de 'Datos de Cuerpos'

La gráfica se divide en dos partes. Se crea una línea imaginaria sobre el eje X.

-En la parte superior se observa la gráfica obtenida de la sección 'Resultados'.

-En la parte inferior se observan los polígonos generados a partir de las aristas.

La grafica muestra la gravedad de los polígonos insertados y en una posible erupción el hundimiento en ciertos puntos a partir de los cuerpos y su densidad.

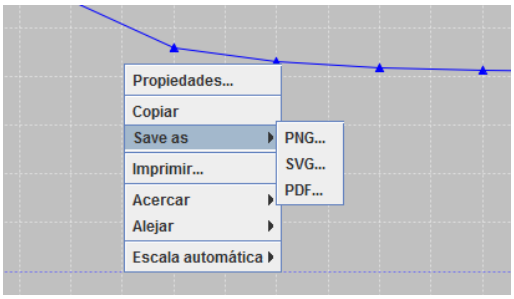
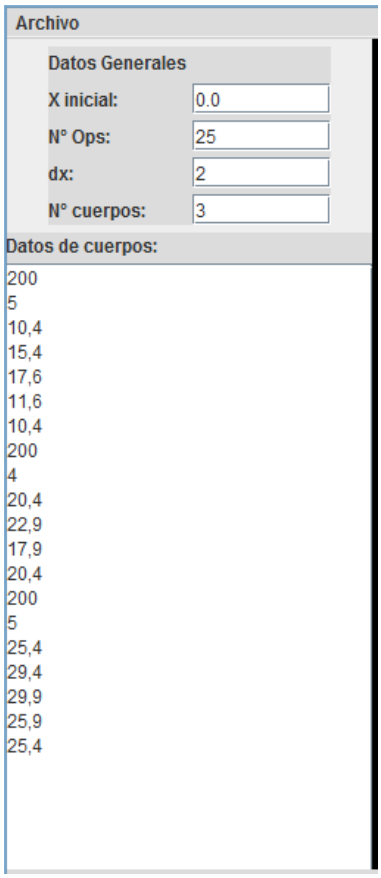


Fig. 17. Ventana que aparece para guardar gráfica

Caso de dos o más cuerpos.

Para el caso de ingresar dos o más cuerpos el programa sigue con el mismo ingreso de datos. Especificando, aquí se muestra una prueba:



Se inicia en el origen. X inicial = 0

Número de puntos de observación = 25

Intervalo, distancia entre cada punto = 2

Número de cuerpos = 3

Datos de cuerpos

Cuerpo 1: *Densidad*
Número de Aristas (+1)
Arista Inicial
Aristas
(...)
Arista Inicial

Cuerpo 2: *Densidad*
Número de Aristas (+1)
Arista Inicial
Aristas
(...)
Arista Inicial

Cuerpo 3: *Densidad*
Número de Aristas (+1)
Arista Inicial
Aristas
(...)
Arista Inicial

Fig. 18. Prueba para ingresar datos de tres cuerpos

Llenando los campos correctamente se procede a calcular y obtener Resultados.

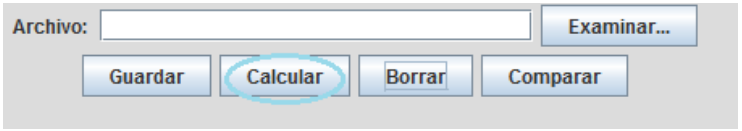
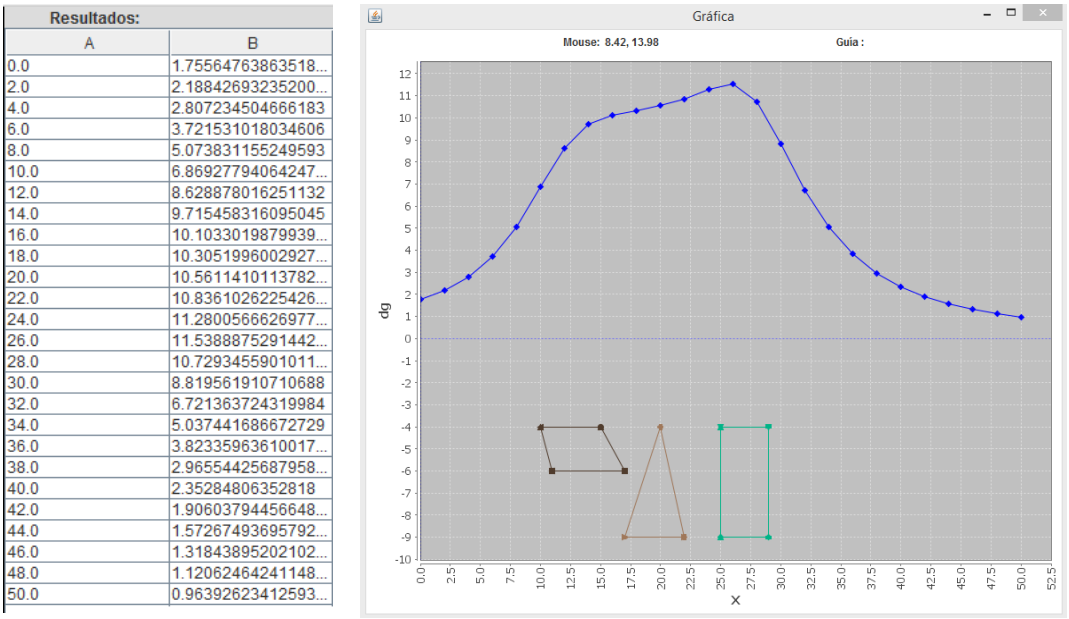


Fig. 19. Sección en donde se procede a ejecutar la acción de Calcular

Se obtiene el resultado y la gráfica de varios cuerpos.



Resultados:

A	B
0.0	1.75564763863518...
2.0	2.18842693235200...
4.0	2.807234504666183
6.0	3.721531018034606
8.0	5.073831155249593
10.0	6.86927794064247...
12.0	8.628878016251132
14.0	9.715458316095045
16.0	10.1033019879939...
18.0	10.3051996002927...
20.0	10.5611410113782...
22.0	10.8361026225426...
24.0	11.2800566626977...
26.0	11.5388875291442...
28.0	10.7293455901011...
30.0	8.819561910710688
32.0	6.721363724319984
34.0	5.037441686672729
36.0	3.82335963610017...
38.0	2.96554425687958...
40.0	2.35284806352818
42.0	1.90603794456648...
44.0	1.57267493695792...
46.0	1.31843895202102...
48.0	1.12062464241148...
50.0	0.96392623412593...

Fig. 19. Prueba al calcular 3 cuerpos en la aplicación

Guardado de Datos.

Teniendo los datos cargados en ADGAVO, se tiene la opción de guardar los datos utilizados para posteriormente utilizarlos o compararlos con más datos.

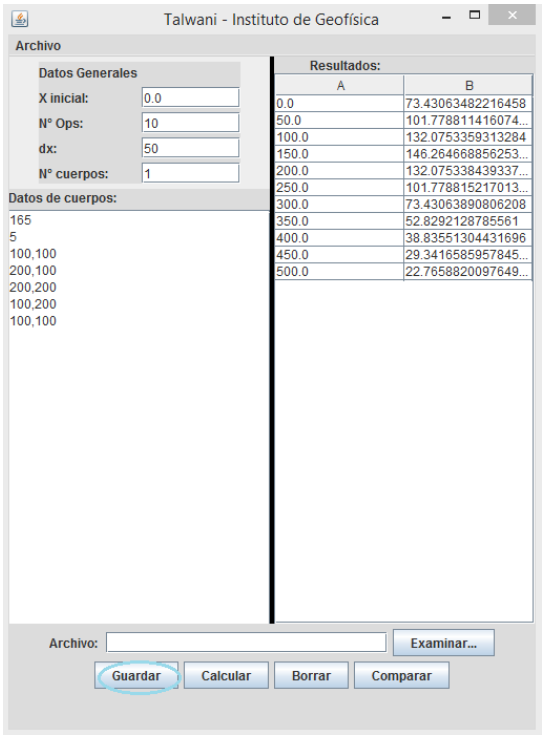


Fig. 20. Para guardar los datos ingresados, se da click en el botón de 'Guardar'

Se guardan los datos especificando la ruta y extensión '.txt'

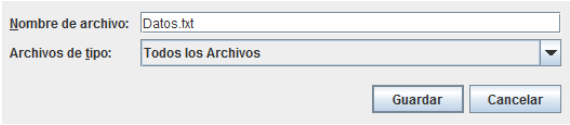


Fig. 21. Se especifica la ruta y extensión del archivo a guaradar

Los datos se guardan con el siguiente formato:

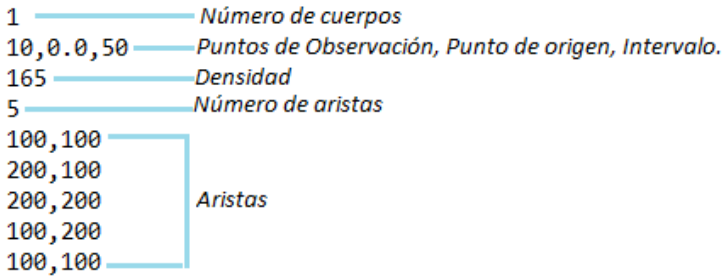


Fig. 22. Sección Datos de Cuerpo, un solo cuerpo

Carga de Datos.

Para la carga de datos de debe de tener un archivo con las características de la *Figura 22,* posteriormente se carga el archivo existente con formato txt. Dando click en la casilla ***'Examinar'***

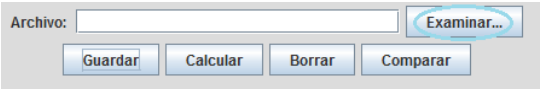


Fig. 23. Botón 'Examinar' para abrir archivo

Para cargar un archivo de dos o más cuerpos se utiliza el formato de la figura *A*.

- En número de cuerpos se especifica cuantos van a utilizarse.

- A partir de la última arista del primer cuerpo se especifica la densidad del segundo cuerpo, enseguida el número de aristas y así sucesivamente.

Prueba 2: Se ingresan los datos para calcular la gravedad de dos cuerpos alrededor de un volcán y determinar su posible hundimiento a partir del cálculo.

```
2 ———— Número de cuerpos
10,0.0,50 ———— Puntos de observación,Punto de origen, Intervalo.
485 ———— Densidad de cuerpo 1
4 ———— Número de aristas de cuerpo 1
0,200
100,150
100,300          Aristas cuerpo 1
0,200
235 ———— Densidad cuerpo 2
5 ———— Número de aristas de cuerpo 2
300,100
350,100
400,200          Aristas cuerpo 2
250,150
300,100
```

Fig. 24. Sección Datos de Cuerpo, dos cuerpos

Se escoge el archivo e inmediatamente se muestran los resultados.

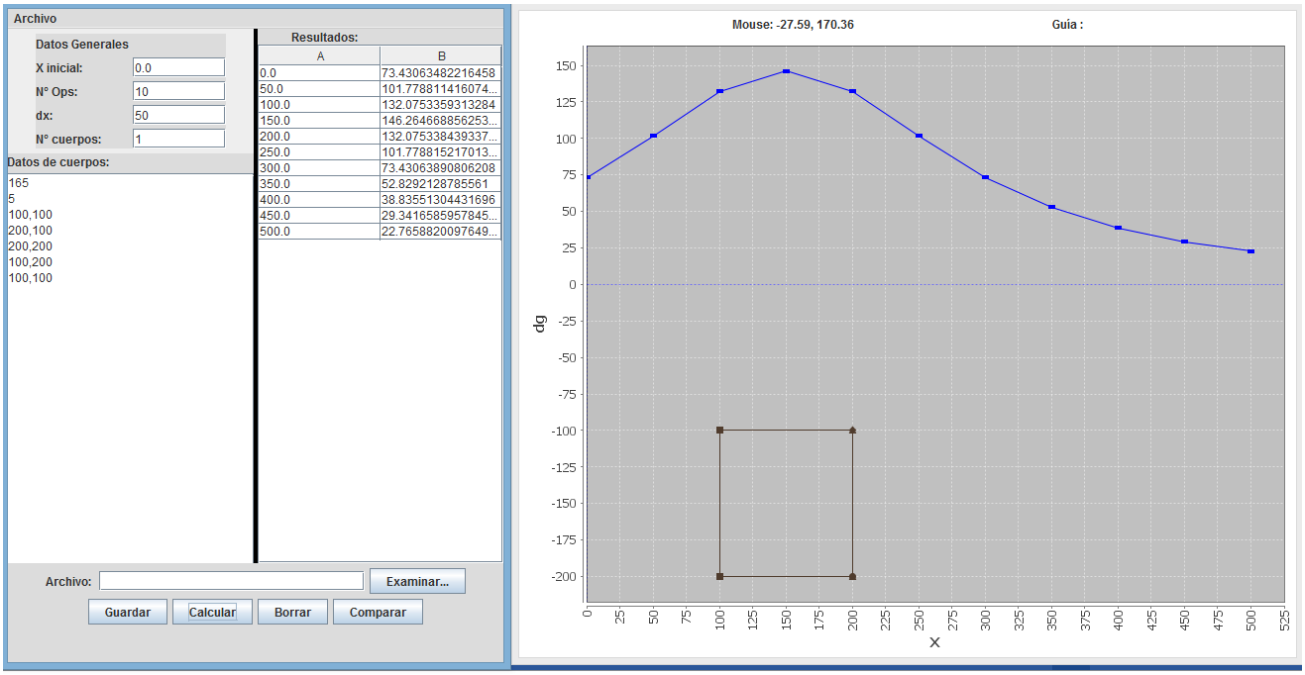


Fig. 25. Carga de datos en automático y desplegado de gráfica al abrir un archivo existente con datos

Comparación de dos gráficas para su análisis.

Cuando se quiere comparar dos gráficas, se debe tener una carga de datos en ADGAVO. Para llevar a cabo la comparación se da click en el campo ***'Comparar'*** como se muestra a continuación.

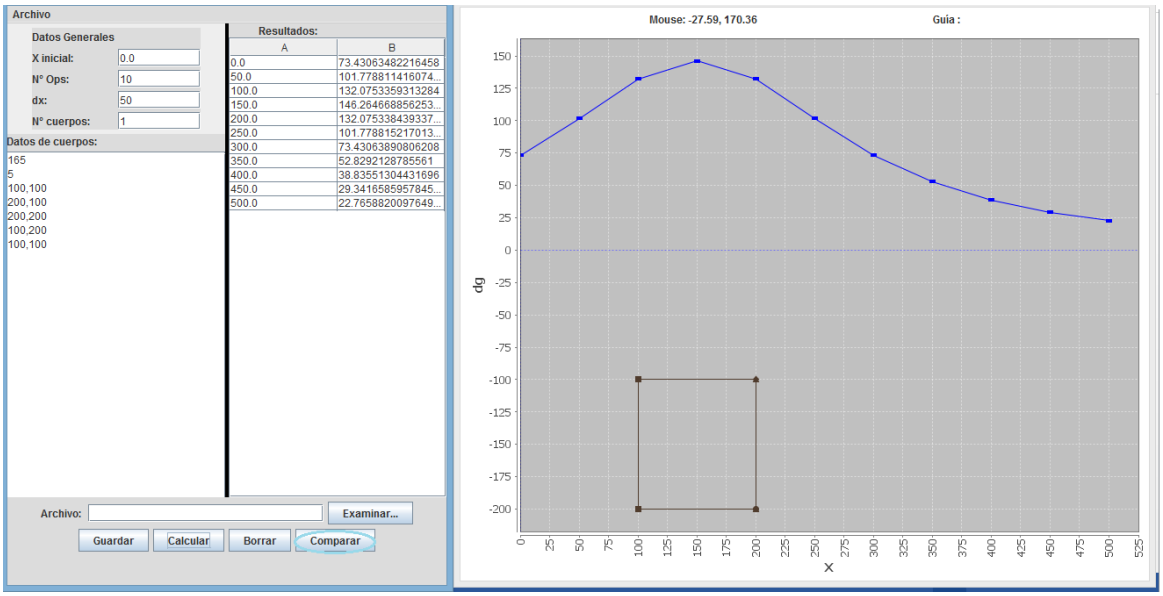


Fig. 26. Ventanas con datos obtenidos con la aplicación

Teniendo el resultado de un análisis de datos, se procede a buscar un archivo existente dando click en el botón ‘comparar’ y se muestra la comparación en la siguiente figura

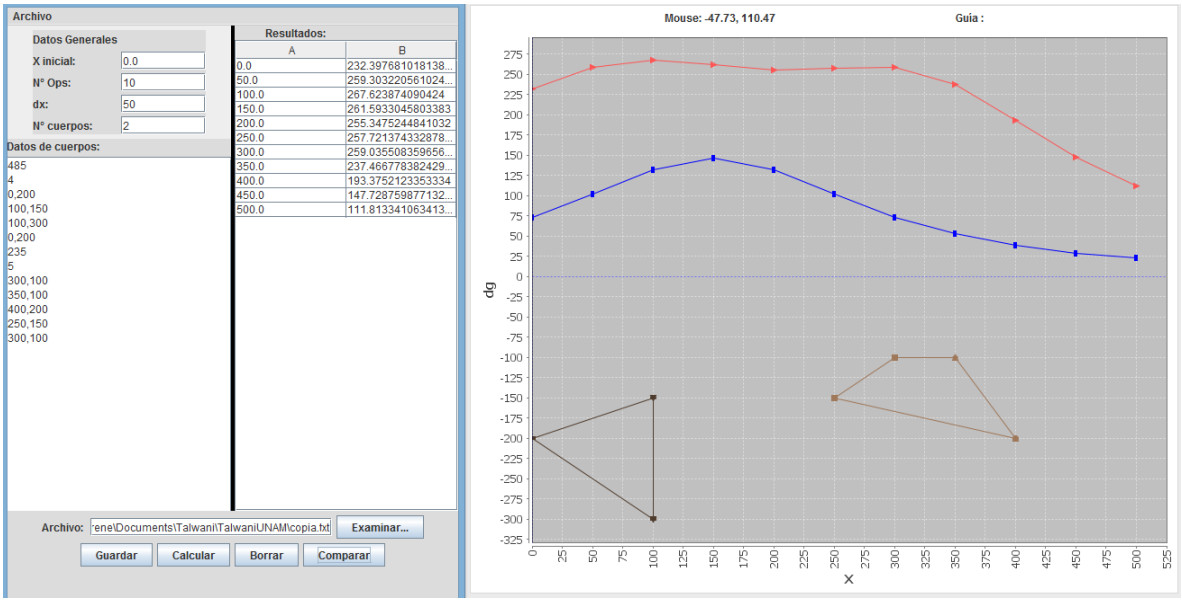


Fig. 27. Gráfica donde se compara el análisis de dos cuerpos con otra de un solo cuerpo

Se muestra la comparación de datos donde las gráficas muestran la curva de gravedad y se analiza su posible hundimiento en una erupción del volcán a partir de la densidad de los cuerpos ingresados y su análisis por parte de los especialistas en la materia, como los investigadores del Instituto de Geofísica.

Los cuerpos se pueden interpretar como rocas o placas por debajo de la tierra y su principal uso del sistema, es la identificación de Áreas en riesgo de hundimiento a partir de una erupción de un volcán.

Una de las ventajas de utilizar ADGAVO es que se pueden manipular los datos en tiempo real. Modificando coordenadas, densidad o incluir cuerpos, solo se modifican los datos en la sección de 'Datos Generales', 'Datos de Cuerpos', se presiona el botón de '*Calcula'* y automáticamente se actualiza la ventana de '*Gráfica'*.

Acceso rápido

En la parte superior de la aplicación se encuentra una apartado que dice ‘Archivo’, al dar click despliega un catálogo de la funciones rápidas que se pueden utilizar durante la ejecución de la aplicación

Fig. 28. Ventana de Archivo donde se muestran los accesos rápidos a la funciones

‘F1’ – Abrir archivo

‘F2’ – Calcular

‘F3’ – Borrar

‘F4’ – Salir

‘F5’ – Guardar

‘F6’ – Comparar

Capítulo V: Conclusiones

La aplicación obtenida en este trabajo es de gran utilidad porque nos permite obtener resultados gráficos con gran rapidez, con lo cual la labor de análisis de los datos se simplifica considerablemente. Esto se debe a que permite ir ajustando los datos teóricos a los observados manipulando la geometría de los cuerpos modelados en tiempo real, de manera que el proceso de prueba y error se realiza en corto tiempo; por otro lado, también presenta los resultados en forma tabular para permitir su manipulación posterior, si así lo desea el analista. Estos aspectos se detallan en el manual del usuario. Por consiguiente, programa ADGAVO cumple con los objetivos establecidos y puede ser modificado fácilmente para su ejecución en diferentes sistemas operativos.

Tomando en cuenta los objetivos planteados en esta tesis, los resultados más importantes son los siguientes:

- Se establecieron los elementos necesarios para desarrollar una aplicación para vulcanología con éxito. Para este punto se aplicaron las técnicas más novedosas que beneficiarían al proyecto, entre las que se pueden destacar el paradigma orientado a objetos, el modelo vista controlador y el sistema de control de versiones.

Capítulo VI: Anexo – Código

En esta parte se muestran las clases y objetos más importantes de la programación que fueron fundamentales para desplegar ventanas y realizar cálculos en base al algoritmo establecido por Talwani.

<u>TalwaniUNAM.java</u> – Es la clase principal del Proyecto (main). Desde aquí se manda crear la ventana princioal, definida por 'VentanaDatos.java'

```
package talwaniunam;

import talwaniunam.interfaz.Splash;
import talwaniunam.interfaz.VentanaDatos;

public class TalwaniUNAM
{

	public static void main(String[] args)
	{
		Splash pantallaInicio = new Splash();
		javax.swing.SwingUtilities.invokeLater(new Runnable() // Falta cerrar
		{
			public void run()
			{
				VentanaDatos programa = new VentanaDatos();
				programa.muestraInterfaz();
			}
		}
		); // Se cerró
	}
}
```

ASISTENTES

<u>MANEJADOR DE ARCHIVOS</u> – Con esta clase se implementa la parte de leer archivos guardados en la computadora.

```
package talwaniunam.asistentes;

import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;

public class ManejadorArchivos
{
	String cadena = new String();

	public String lee(String archivo)
	{
		try
		{
			StringBuffer strb = new StringBuffer();
```

```
                        FileReader lector = new FileReader(archivo);
                        int c;
                        while((c = lector.read() )!= -1)
                        {
                                strb.append((char)c);
                        }

                        cadena = strb.toString();

                        if(lector != null)
                        {
                                try
                                {
                                        lector.close();
                                }
                                catch(IOException ioe){      }
                        }

                }
                catch (IOException ioe)
                {
                        System.out.println(ioe);
                }
                return cadena;
        }

        public void escribe(String cadena, String archivo)
        {
                try
                {
                        FileOutputStream escribano = new FileOutputStream(archivo, true);
                        PrintStream lapiz = new PrintStream(escribano);

                        lapiz.println(cadena);
                        lapiz.close();

                }
                catch(IOException ioe)
                {
                        ioe.printStackTrace();
                }


        }
}
```

POLIGONO *– Clase que lee los datos de cada poligono para el cálculo de la gravedad*

```
package talwaniunam.asistentes;

import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;

public class Poligono
{
        int indice;
        double[][] coordenadas;
```

```
	public Poligono(int indice, double[][] coordenadas)
	{
		this.indice = indice;
		this.coordenadas = coordenadas;
	}

	public XYDataset creaDataset()
	{
		XYSeries[] series = new XYSeries[coordenadas.length];

		for (int i = 0; i < series.length - 1; i++)
		{
			series[i] = new XYSeries("Cuerpo " + i);
			series[i].add(coordenadas[i][0], coordenadas[i][1] * -1);
			series[i].add(coordenadas[i + 1][0], coordenadas[i + 1][1] * -1);
		}

		XYSeriesCollection coleccion = new XYSeriesCollection();
			for (int i = 0; i < series.length; i++)
				if(series[i] != null)
					coleccion.addSeries(series[i]);

		return coleccion;
	}
}
```

INTERFAZ

PANEL CONTROL *– Clase que manda llamar cada función de los botones de la interfaz*

```
package talwaniunam.interfaz;

import java.awt.Color;
import java.awt.Dimension;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class PanelControles extends javax.swing.JPanel
{
	JLabel etiquetaArchivo = new JLabel("Archivo: ");
	JTextField campoArchivo      = new JTextField(24);
	JButton botonExaminar = new JButton("Examinar...");
	JButton botonGuardar = new JButton("Guardar");
	JButton botonCalcular = new JButton("Calcular");
	JButton botonBorrar = new JButton("Borrar");
	JButton botonComparar = new JButton("Comparar");

	public void inicializar(int ancho, int alto, Color color)
	{
		setPreferredSize(new Dimension(ancho, alto));
		setBackground (color);

		add(etiquetaArchivo);
		add(campoArchivo);
		add(botonExaminar);
		add(botonGuardar);
		add(botonCalcular);
```

```
                add(botonBorrar);
                add(botonComparar);
        }
}
```

***PANEL DE DATOS DEL CUERPO** – Sección donde se introducen los datos de los cuerpos.*

```
package talwaniunam.interfaz;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;

public class PanelDatosCuerpos extends javax.swing.JPanel
{
        JTextArea datos = new JTextArea(12, 12);
        JScrollPane scroll = new JScrollPane(      datos,

        ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,

        ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NEEDED
                                                                        );

        public void inicializa(int ancho, int alto, Color color)
        {
                setLayout             (new BorderLayout(4, 4) );
                setPreferredSize(new Dimension(ancho, alto));
                setBackground  (color);

                add(new JLabel("Datos de cuerpos:"), BorderLayout.PAGE_START );

                scroll.setSize(1, 1);
                add(scroll, BorderLayout.CENTER);

        }
}
```

***PANEL DE DATOS GENERALES** – Sección donde se introducen los datos iniciales*

```
package talwaniunam.interfaz;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class PanelDatosGenerales extends javax.swing.JPanel
{
        JTextField xIni              = new JTextField();
        JTextField nOps              = new JTextField();
        JTextField dX         = new JTextField();
        JTextField nCuerpos = new JTextField();

        public void inicializa(int ancho, int alto, Color color)
```

```
	{
		setLayout		(new GridLayout(5, 2, 4, 4) );
		setMaximumSize(new Dimension(ancho, alto));
		setBackground (color);

		add(new JLabel("Datos Generales") );
		add(new JLabel("") );

		add(new JLabel("X inicial:") );
		xIni.setSize(86, 30);
		add(xIni);

		add(new JLabel("N° Ops:") );
		nOps.setSize(86, 30);
		add(nOps);

		add(new JLabel("dx:") );
		dX.setSize(86, 30);
		add(dX);

		add(new JLabel("N° cuerpos:") );
		nCuerpos.setSize(86, 30);
		add(nCuerpos);
	}
}
```

PANEL DE GRÁFICAS *– Parte donde se crea una nueva ventana para los resultados gráficos-*

```
package talwaniunam.interfaz;

import java.awt.BorderLayout;
import java.awt.Color;
import javax.swing.JLabel;
import javax.swing.JPanel;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartMouseEvent;
import org.jfree.chart.ChartMouseListener;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYItemRenderer;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.DefaultXYDataset;

public class PanelGraficas extends JPanel
{
	DefaultXYDataset datosXY = new DefaultXYDataset();
	JFreeChart grafica = ChartFactory.createScatterPlot("Gráfica", "X", "dg", datosXY,
PlotOrientation.VERTICAL, true, false, false);
	ChartPanel panelGrafica = new ChartPanel(grafica);

	public void inicializa(String nombre, double[][] datos, boolean ejeVertical,
boolean ejeHorizontal)
	{
		panelGrafica.setMaximumDrawHeight(1080);
		panelGrafica.setMinimumDrawHeight(0);
```

```
            panelGrafica.setMaximumDrawWidth(1920);
            panelGrafica.setMinimumDrawWidth(0);

            panelGrafica.setHorizontalAxisTrace(ejeHorizontal);
            panelGrafica.setVerticalAxisTrace(ejeVertical);
            panelGrafica.setToolTipText(null);

            final JLabel coords = new JLabel("COORDENADAS: ");
            panelGrafica.add(coords, BorderLayout.NORTH);

            XYPlot xyPlot = (XYPlot) grafica.getPlot();
            xyPlot.setDomainCrosshairVisible(true);
            xyPlot.setRangeCrosshairVisible(true);
            XYItemRenderer renderer = xyPlot.getRenderer();
            renderer.setSeriesPaint(0, Color.blue);
            NumberAxis domain = (NumberAxis) xyPlot.getDomainAxis();
            domain.setVerticalTickLabels(true);

            datosXY.addSeries(nombre, datos);

            XYPlot plot = (XYPlot) grafica.getPlot();
            XYLineAndShapeRenderer rendererForma = new XYLineAndShapeRenderer();
            rendererForma.setSeriesLinesVisible(0, true);
            plot.setRenderer(rendererForma);

            panelGrafica.addChartMouseListener( new ChartMouseListener()
                                                                        {

       public void chartMouseMoved(ChartMouseEvent e)
                                                                        {



       System.out.println("Coordenadas:" +
e.getChart().getXYPlot().getDomainAxisLocation() );
                                                                        }


       @Override

       public void chartMouseClicked(ChartMouseEvent cme)
                                                                        {

       throw new UnsupportedOperationException("Not supported yet."); //To change body of
generated methods, choose Tools | Templates.
                                                                        }
                                                                    });
       }


}
```

<u>PANEL DE RESULTADOS</u> – Ventana en donde se muestran los resultados obtenidos a partir del algoritmo de Talwani

```
package talwaniunam.interfaz;

import java.awt.Color;
import java.awt.Dimension;
import javax.swing.BoxLayout;
```

```
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import talwaniunam.asistentes.ModeloTabla;

public class PanelResultados extends javax.swing.JPanel
{
	String[] columnasTabla = {"X", "dg"};
	double[][] datosTabla;
	JTable tablaResultados;
	JScrollPane scroll;
	ModeloTabla modeloTabla;

	public void inicializa(int ancho, int alto, Color color)
	{
		setLayout		(new BoxLayout(this, BoxLayout.PAGE_AXIS) );
		setPreferredSize(new Dimension(ancho, alto));
		setBackground (color);

		add(new JLabel("Resultados:") );
		add(new JLabel("") );

		modeloTabla = new ModeloTabla();
		tablaResultados = new JTable(modeloTabla);

		scroll = new JScrollPane(tablaResultados);
		tablaResultados.setFillsViewportHeight(true);
		tablaResultados.setFocusable(false);

		add(scroll);
	}
}
```

***GIF INICIO** – Gif que se muestra antes de abrir la ventana principal de la aplicación*

```
package talwaniunam.interfaz;

import java.awt.*;
import java.awt.event.*;

public class Splash implements ActionListener
{
    static void renderSplashFrame(Graphics2D g, int frame)
       {
        final String[] comps = {"máquina virtual", "módulos", "ventanas"};
        g.setComposite(AlphaComposite.Clear);
        g.fillRect(120,140,200,40);
        g.setPaintMode();
        g.setColor(Color.BLACK);
        g.drawString("Cargando "+comps[(frame/5)%3]+"...", 120, 150);
    }

    public Splash()
       {
        final SplashScreen splash = SplashScreen.getSplashScreen();
        if (splash == null) {
            System.out.println("SplashScreen.getSplashScreen() returned null");
            return;
        }
```

```
        Graphics2D g = splash.createGraphics();
        if (g == null) {
            System.out.println("g is null");
            return;
        }
        for(int i=0; i<100; i++) {
            renderSplashFrame(g, i);
            splash.update();
            try {
                Thread.sleep(90);
            }
            catch(InterruptedException e) {
            }
        }
        splash.close();
        //setVisible(true);
        //toFront();
    }
    public void actionPerformed(ActionEvent ae) {
        System.exit(0);
    }

    private static WindowListener closeWindow = new WindowAdapter()
       {
        public void windowClosing(WindowEvent e)
              {
            e.getWindow().dispose();
        }
    };
}
```

<u>VENTANA DE DATOS</u> – Por medio de coordenadas y colores se implementa la interfaz gráfica de la ventana principal de la aplicación

```
package talwaniunam.interfaz;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.KeyStroke;
import talwaniunam.asistentes.Arreglos;
import talwaniunam.asistentes.ManejadorArchivos;
import talwaniunam.modelo.Algoritmo;
import talwaniunam.modelo.Cuerpo;

public class VentanaDatos extends javax.swing.JFrame implements
java.awt.event.ActionListener
{
       // Constantes de dimensiones de la ventana.
       public static final int ANCHO_VENTANA = 500,
```

```
                                                ALTO_VENTANA = 640,
                                                ALTO_INFERIOR = 100,
                                                ALTO_SUPERIOR = ALTO_VENTANA -
ALTO_INFERIOR,
                                                ALTO_DATOS_GENERALES = 120,
                                                ALTO_DATOS_CUERPOS = ALTO_SUPERIOR -
ALTO_DATOS_GENERALES;

        // Elementos gráficos de la ventana.

        // Barra de menu y sus elementos.
        JMenuBar barraMenu = new JMenuBar();
                JMenu menuArchivo = new JMenu("Archivo");
                        JMenuItem menuAbrirArchivo = new JMenuItem("Abrir archivo...");
                        JMenuItem menuCalcular = new JMenuItem("Calcular");
                        JMenuItem menuBorrar = new JMenuItem("Borrar");
                        JMenuItem menuGuardar = new JMenuItem("Guardar");
                        JMenuItem menuComparar = new JMenuItem("Comparar");
                        JMenuItem menuSalir = new JMenuItem("Salir");

        // Paneles
        JPanel panelSuperior = new JPanel();
                JPanel panelDatos = new JPanel();
                        PanelDatosGenerales datosGenerales = new PanelDatosGenerales();
                        PanelDatosCuerpos datosCuerpos = new PanelDatosCuerpos();
                PanelResultados resultados = new PanelResultados();
        PanelControles controles = new PanelControles();

        VentanaGraficas ventanaGrafica = new VentanaGraficas();

        public void muestraInterfaz()
        {
                Color colorPaneles = new Color(220, 220, 220);

                setTitle("Talwani - Instituto de Geofísica");
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setMaximumSize(new Dimension(ANCHO_VENTANA, ALTO_VENTANA) );

                panelSuperior.setLayout                    (new GridLayout(1,2,4,4) );
                panelSuperior.setPreferredSize      (new Dimension(ANCHO_VENTANA,
ALTO_SUPERIOR));
                panelSuperior.setBackground         (new Color(0, 0, 0));

                // Inicializar elementos independientes.
                datosGenerales.inicializa(ANCHO_VENTANA / 2, ALTO_DATOS_GENERALES,
colorPaneles);
                datosCuerpos.inicializa(ANCHO_VENTANA / 2, ALTO_DATOS_CUERPOS,
colorPaneles);
                resultados.inicializa(ANCHO_VENTANA / 2, ALTO_SUPERIOR, colorPaneles);

                controles.inicializar(ANCHO_VENTANA, ALTO_INFERIOR, colorPaneles);
                        controles.botonExaminar.addActionListener(this);
                        controles.botonGuardar.addActionListener(this);
                        controles.botonCalcular.addActionListener(this);
                        controles.botonBorrar.addActionListener(this);
                        controles.botonComparar.addActionListener(this);

                barraMenu.setOpaque(true);
                barraMenu.setBackground(colorPaneles);
                barraMenu.setPreferredSize(new Dimension(200, 20) );
                barraMenu.add(menuArchivo);
```

```
                menuArchivo.add(menuAbrirArchivo);
                        menuAbrirArchivo.addActionListener(this);

        menuAbrirArchivo.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F1, 0) );
                menuArchivo.add(menuCalcular);
                        menuCalcular.addActionListener(this);

        menuCalcular.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F2, 0) );
                menuArchivo.add(menuBorrar);
                        menuBorrar.addActionListener(this);

        menuBorrar.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F3, 0) );
                menuArchivo.add(menuGuardar);
                        menuGuardar.addActionListener(this);

        menuGuardar.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F5, 0) );
                menuArchivo.add(menuComparar);
                        menuComparar.addActionListener(this);

        menuComparar.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F6, 0) );

                menuArchivo.addSeparator();
                menuArchivo.add(menuSalir);
                        menuSalir.addActionListener(this);

        menuSalir.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F4, 0) );

                setJMenuBar(barraMenu);
                getContentPane().add  (panelSuperior,       "North");
                        panelSuperior.add(panelDatos);
                                panelDatos.add(datosGenerales);
                                panelDatos.add(datosCuerpos);
                        panelSuperior.add(resultados);
                getContentPane().add  (controles,    "South");
                pack();
                setLocation(100, 100);
                setVisible(true);
        }

        /*
         * Abre el archivo seleccionado desde el botón examinar y llena los campos
         * correspondientes en la interfaz grafica
         */
        private void abreArchivo()
        {
                JFileChooser elector = new JFileChooser(System.getProperty("user.dir"));
                elector.showOpenDialog(this);
                File archivoAbierto = elector.getSelectedFile();

                if(elector.getSelectedFile() != null)
                {

        controles.campoArchivo.setText(elector.getSelectedFile().getAbsolutePath());
                        ManejadorArchivos archivo = new ManejadorArchivos();

                        String entrada =
archivo.lee(elector.getSelectedFile().getAbsolutePath() );
                        String[] datosPre = entrada.split("\n");
                        String[] datos = new String[datosPre.length];

                        for(int i = 0; i < datosPre.length; i++)
```

```
                        datos[i] = datosPre[i].substring(0, datosPre[i].length() -
1);

                String[] datosN = datos[1].split(",");
                System.out.println("datosN: " + datosN.length);
                for (int i = 0; i < datosN.length; i++)
                {
                        System.out.println(i + ": " + datosN[i]);

                }
                datosGenerales.nCuerpos.setText(datos[0]);
                datosGenerales.nOps.  setText(datosN[0]);
                datosGenerales.xIni.  setText(datosN[1]);
                datosGenerales.dX.           setText(datosN[2]);

                datosCuerpos.datos.setText("");
                for (int i = 2; i < datos.length; i++)
                        datosCuerpos.datos.append(datos[i] + "\n");
        }
    }

    /*
     * Realiza el calculo de los resultados a partir de los datos mostrados en
     * la interfaz grafica
     */
    private void calcula(boolean compara)
    {
            Algoritmo talwani = new Algoritmo(
    Integer.parseInt(datosGenerales.nCuerpos.getText() ),

    Integer.parseInt(datosGenerales.nOps.getText() ),

    Double.parseDouble(datosGenerales.xIni.getText() ),

    Double.parseDouble(datosGenerales.dX.getText() )
                                                                        );
            String cadenaTemporal = datosCuerpos.datos.getText();

            // Crear arreglo de cadenas con renglones del JTextArea
            String[] datosCuerpos = cadenaTemporal.split("\n");
            int indice = 0;

            Cuerpo[] cuerpos = new
Cuerpo[Integer.parseInt(datosGenerales.nCuerpos.getText() )];
            //System.out.println(cuerpos.length);
            for (int i = 0; i < cuerpos.length; i++)
            {
                    double densidad = Double.parseDouble(datosCuerpos[indice++]);
                    int nAristas = Integer.parseInt(datosCuerpos[indice++]);

                    cuerpos[i] = new Cuerpo      (densidad, nAristas);
                    // System.out.println("Calculando cuerpo N° " + i);
                    for (int j = 0; j < nAristas; j++)
                    {
                            double[] coordenadas = new double[2];
                            String[] coordenadasTexto =
datosCuerpos[indice++].split(",");
                            for (int k = 0; k < 2; k++)
                                    coordenadas[k] =
Double.parseDouble(coordenadasTexto[k]);
```

```
                        cuerpos[i].setArista(coordenadas[0], coordenadas[1], j);
                        //System.out.println("Guardando coordenada " + j);
                    }
                    //System.out.println(cuerpos[i]);
                }

                double[][] resultado = new
double[Integer.parseInt(datosGenerales.nOps.getText() ) + 1][2];
                double[][] temp;

                for (int i = 0; i < cuerpos.length; i++)
                {
                    temp = talwani.calcula(cuerpos[i]);
                        //System.out.println("- - - TEMP(" + i +"): - - -");
Arreglos.imprime(temp);

                    if (i == 0)
                        Arreglos.suma(resultado, temp);
                    else
                        Arreglos.sumaUno(resultado, temp);
                    //System.out.println("- - - RESULTADO SUMADO(" + i + "): - - -");
Arreglos.imprime(resultado);
                }

                ventanaGrafica.setFocusableWindowState(false);
                /*
                 *  CAMBIAR A MOSTRAR LA PRIMERA VEZ, ACTUALIZAR LAS SIGUIENTES
                 */

                if(compara)
                    ventanaGrafica.actualizaVentana(Arreglos.invierte(resultado),
cuerpos, "Gravity");
                else
                    ventanaGrafica.muestraVentana(Arreglos.invierte(resultado),
cuerpos);

                resultados.datosTabla = resultado;
                resultados.modeloTabla.actualizar(resultados.datosTabla);
                resultados.modeloTabla.fireTableDataChanged();

                revalida();
        }

        private void revalida()
        {
                resultados.scroll.revalidate();
                resultados.revalidate();
                barraMenu.revalidate();
                datosCuerpos.scroll.revalidate();
                validate();
        }

        @Override
        public void actionPerformed(ActionEvent evento)
        {
                if(evento.getSource() == controles.botonExaminar || evento.getSource() ==
menuAbrirArchivo)// Agregar teclas de funcion
                {
                    abreArchivo();
                    calcula(false);
                }
```

```
		if(evento.getSource() == controles.botonGuardar)
		{
			String archivo = "";
			archivo +=	datosGenerales.nCuerpos.getText() + "\r\n" +
					datosGenerales.nOps.getText() + "," +
					datosGenerales.xIni.getText() + "," +
					datosGenerales.dX.getText() + "\r\n" +
					datosCuerpos.datos.getText();

			JFileChooser elector = new
JFileChooser(System.getProperty("user.dir"));
				elector.showSaveDialog(this);
				File nuevo = elector.getSelectedFile();
				try
				{
					FileWriter escritor = new FileWriter(nuevo);
					escritor.write(archivo);
					escritor.flush();
					escritor.close();
				}
				catch(IOException e)
				{
					System.out.println("Excepcion IO");
				}


		}

		if(evento.getSource() == controles.botonCalcular || evento.getSource() ==
menuCalcular)
		{
			calcula(false);
		}

		if(evento.getSource() == menuSalir)
			System.exit(0);

		if(evento.getSource() == menuBorrar || evento.getSource() ==
controles.botonBorrar)
		{
			datosGenerales.xIni.setText("");
			datosGenerales.nOps.setText("");
			datosGenerales.dX.setText("");
			datosGenerales.nCuerpos.setText("");

			datosCuerpos.datos.setText("");

			Arreglos.inicializa(resultados.datosTabla);
			resultados.modeloTabla.actualizar(resultados.datosTabla);
			resultados.modeloTabla.fireTableDataChanged();

			revalida();

		}

		if(evento.getSource() == controles.botonComparar || evento.getSource() ==
menuComparar)
		{
			abreArchivo();
			calcula(true);
```

```
		}
	}


}
```

<u>VENTANA DE GRAFICAS</u> – Se crea la ventana de gráficos a partir de coordenadas y colores declarados

```
package talwaniunam.interfaz;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import javax.swing.JLabel;
import javax.swing.JSeparator;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartMouseEvent;
import org.jfree.chart.ChartMouseListener;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.DefaultXYDataset;
import org.jfree.data.xy.XYDataset;
import talwaniunam.asistentes.Poligono;
import talwaniunam.modelo.Cuerpo;

public class VentanaGraficas extends javax.swing.JFrame
{
		// Constantes de dimensiones de la ventanas de la aplicacion.
	public static final int ANCHO_VENTANA = 800,
                                                ALTO_VENTANA = 692,
                                                COLOR_R = 80,
                                                COLOR_G = 60,
                                                COLOR_B = 45;


	DefaultXYDataset datosXY = new DefaultXYDataset();
	JFreeChart graficaGravedad = ChartFactory.createScatterPlot("----", "X", "dg",
datosXY,

                            PlotOrientation.VERTICAL, rootPaneCheckingEnabled,
rootPaneCheckingEnabled,

                            rootPaneCheckingEnabled);
	ChartPanel panelGrafica = new ChartPanel(graficaGravedad);
	final JLabel coordsMouse = new JLabel("Mouse: ");
	final JLabel coordsGuia = new JLabel("
Guía : ");

	public void muestraVentana(double[][] datos, Cuerpo[] cuerpos)
	{
		setTitle("Gráfica");
```

```
                getContentPane().add(panelGrafica);

                setMaximumSize(new Dimension(ANCHO_VENTANA, ALTO_VENTANA) );
                setPreferredSize(new Dimension(ANCHO_VENTANA, ALTO_VENTANA) );

                panelGrafica.setMaximumDrawHeight(1080);
                panelGrafica.setMinimumDrawHeight(0);
                panelGrafica.setMaximumDrawWidth(1920);
                panelGrafica.setMinimumDrawWidth(0);

                panelGrafica.setHorizontalAxisTrace(false);
                panelGrafica.setVerticalAxisTrace(false);
                panelGrafica.setToolTipText(null);
                panelGrafica.getChart().removeLegend();


                panelGrafica.add(coordsMouse);
                       coordsMouse.setOpaque(true);
                       coordsMouse.setBackground(Color.white);
                panelGrafica.add(coordsGuia, BorderLayout.NORTH);
                       coordsGuia.setOpaque(true);
                       coordsGuia.setBackground(Color.white);

                panelGrafica.addChartMouseListener( new ChartMouseListener()
                                                                                {

       public void chartMouseMoved(ChartMouseEvent evento)
                                                                                {

       //coords.setText("                      Coordenadas:" + e.getTrigger().getX() +
", " + e.getTrigger().getY());



       Point2D p = panelGrafica.translateScreenToJava2D(evento.getTrigger().getPoint());

       Rectangle2D plotArea = panelGrafica.getScreenDataArea();

       XYPlot plot = (XYPlot) graficaGravedad.getPlot(); // your plot

       double chartX = plot.getDomainAxis().java2DToValue(p.getX(), plotArea,
plot.getDomainAxisEdge());

       double chartY = plot.getRangeAxis().java2DToValue(p.getY(), plotArea,
plot.getRangeAxisEdge());

       coordsMouse.setText(" Mouse: " + String.format("%5.2f", chartX) + ", " +
String.format("%5.2f", chartY) + " ");
                                                                                }


       @Override

       public void chartMouseClicked(ChartMouseEvent evento)
                                                                                {

       coordsGuia.setText("
Guía : " + String.format("%5.2f", evento.getChart().getXYPlot().getDomainCrosshairValue()
) + ", "+ String.format("%5.2f", evento.getChart().getXYPlot().getRangeCrosshairValue() )
);
                                                                                }
```

```
                                                                        });

                actualizaVentana(datos, cuerpos, "Gravedad");

                pack();
                setLocation(610, 100);
                setVisible(true);
        }

        public void actualizaVentana(double[][] datos, Cuerpo[] cuerpos, String nombre)
        {
                XYPlot plotGravedad = (XYPlot) graficaGravedad.getPlot();
                        // Mostrar guias horizontal y vertical
                        plotGravedad.setDomainCrosshairVisible(true);
                        plotGravedad.setRangeCrosshairVisible(true);
                // Mostrar escala de dpminio en forma vertical
                NumberAxis dominio = (NumberAxis) plotGravedad.getDomainAxis();
                        dominio.setVerticalTickLabels(true);

                datosXY.addSeries(nombre, datos);
                XYLineAndShapeRenderer[] renderers = new
XYLineAndShapeRenderer[cuerpos.length + 1];
                for (int i = 0; i < renderers.length; i++)
                {
                        renderers[i] = new XYLineAndShapeRenderer();
                        renderers[i].setSeriesLinesVisible(0, true);
                        plotGravedad.setRenderers(renderers);
                }

                // Renderer de la grafica
                renderers[0].setSeriesPaint(0, Color.BLUE);

                        for (int i = 0; i < cuerpos.length; i++)
                        {
                                plotGravedad.setDataset(i + 1, creaCuerpos(i, cuerpos[i]) );
                                for (int j = 0; j < cuerpos[i].getNumeroAristas() - 1; j++)

        plotGravedad.getRendererForDataset(plotGravedad.getDataset(i + 1)
).setSeriesPaint(j, new Color(COLOR_R * ( (i + 1) % 3), COLOR_G * ( (i + 1) % 4), COLOR_B
* ( (i + 1) % 5) ) );
                        }
        } // Actualiza Ventana

        /*
         *     Mostrar los cuerpos abajo de la gráfica
         */
        public XYDataset creaCuerpos(int indice, Cuerpo cuerpo)
        {
                Poligono poligono = new Poligono(indice, cuerpo.getAristas() );
                return poligono.creaDataset();

        }
}
```

MODELO TALWANI

***ALGORITMO** – Clase principal que realiza el cálculo para mostrar en la ventana de resultados y desplegar la grafica*

```
package talwaniunam.modelo;
```

```
public class Algoritmo
{
	static double GAMMA = 6.67e-11, SI2MG = 1e5, KM2M = 1e3, DENOM0 = 1e-6;

	int nCuerpos, nOps;
	double xInicial, dx;

	static double PRECALCULO;

	double[][] resultado;

	/*
	 * Constructor que incializa todos los campos necesarios para el desarrollo
	 * del algoritmo
	 */
	public Algoritmo(int nCuerpos, int nOps, double xInicial, double dx)
	{
		this.nCuerpos = nCuerpos;
		this.nOps = nOps;
		this.xInicial = xInicial;
		this.dx = dx;
		resultado = new double[nOps + 1][2];

		PRECALCULO = GAMMA * SI2MG * KM2M;
	}

	public double[][] calcula(Cuerpo cuerpo)
	{

		double xOb = xInicial;

		for (int i = 0; i <= nOps; i++)
		{
			resultado[i][0] = xOb;
			resultado[i][1] = 2 * PRECALCULO * cuerpo.getDensidad() *
gravedadCuerpo(cuerpo, xOb);

			xOb += dx;
		}

		return resultado;
	}

	public double[][] calcula(Cuerpo[] cuerpos)
	{
		double[][] temporal;

		for (int i = 0; i < resultado.length; i++)
			for (int j = 0; j < resultado[0].length; j++)
				resultado[i][j] = 0;

		for (int i = 0; i < cuerpos.length; i++)
		{
			temporal = calcula(cuerpos[i]);

			for (int j = 0; j < temporal.length; j++)
					resultado[j][1] += temporal[j][1];
		}

		return resultado;
```

```
	}

	private double gravedadCuerpo(Cuerpo cuerpo, double xOb)
	{
		double suma = 0;

		for(int n = 0; n < cuerpo.getNumeroAristas(); n++)
		{
			int n2;
			double x1, x2, z1, z2;

			n2 = (n == cuerpo.getNumeroAristas() - 1) ? 0 : n + 1; // If-else
version PRO

			x1 = cuerpo.getArista(n)[0] - xOb;
			z1 = cuerpo.getArista(n)[1];

			//System.out.println("n: " + n + "\nn2: " + n2);
			x2 = cuerpo.getArista(n2)[0] - xOb;
			z2 = cuerpo.getArista(n2)[1];

			double denom;

			denom = (z2 - z1 == 0) ? DENOM0 : z2 -z1;
			//System.out.println("z2: " + z2 + "z1: " + z1 + "denom: " +
denom);

			double alfa = (x2 - x1) / denom;
			double beta = (x1 * z2 - x2 * z1) / denom;

			double factor = beta / (1 + alfa * alfa);

			double r1sq = x1 * x1 + z1 * z1;
			double r2sq = x2 * x2 + z2 * z2;
			double term1 = 0.5 * (Math.log(r2sq) - Math.log(r1sq) );
			double term2 = Math.atan2(z2,x2) - Math.atan2(z1,x1);

			suma += factor * (term1 - alfa * term2);
			//System.out.println("Suma: " + suma);

		} // Fin de <for(i)>

		// System.out.println("Cuerpo calculado, SUMA: " +  suma);
		return suma;
	} // Fin de <gravedadCuerpo>
}
```

CUERPO* – *Clase que dibuja los cuerpos en la ventana de gráfica

```
package talwaniunam.modelo;

public class Cuerpo
{
	private double densidad;
	private double[][] aristas;


	/*
	 * Constructor con densidad y numero de aristas
	 */
```

```
public Cuerpo(double densidad, int nAristas)
{
	this.densidad = densidad;
	aristas = new double[nAristas][2];
	for(int i = 0; i < aristas.length; i++)
		for (int j = 0; j < 2; j++)
			aristas[i][j] = 0;
}

/*
 * Constructor que recibe la densidad del cuerpo y un arreglo con las
 * coordenadas de las aristas
 */
public Cuerpo(double densidad, double[][] aristas)
{
	this.densidad = densidad;
	this.aristas = aristas;
}

/*
 * Setters
 */
public void setDensidad(double densidad)
{
	this.densidad = densidad;
}


public void setArista(double x, double z, int indice)
{
	aristas[indice][0] = x;
	aristas[indice][1] = z;
}

public void setAristas(double[][] aristas)
{
	this.aristas = aristas;
}

/*
 * Getters
 */
public double getDensidad()
{
	return densidad;
}

public int getNumeroAristas()
{
	return aristas.length;
}

public double[] getArista(int indice)
{
	return aristas[indice];
}

public double[][] getAristas()
{
	return aristas;
}
```

```
	/*
	 * Sobreescritura del metodo toString() para imprimir en consola la informacion
	 * del objeto
	 */
	@Override
	public String toString()
	{
		return "Densidad: " + densidad + "\nNúmero de Aristas: " + aristas.length;

	}
}
```

Capítulo VII: Referencias

[1] TALWANI, M., WORZEL J. L., AND LANDISMAN M. (1959). “Rapid Gravity Computations for Two-Dimensional Bodies with Application to the Mendocino Submarine Fracture Zone”, J. Geophys. Res. 64, 1, 49-59

[2] NETTLETON, L. L., (1940). Gravity and Magnetics in Oil Prospecting. McGraw Hill, International Series in The Earth and Planetary Sciences. 444 pp.

[3] BLAKELY, Richard J. (1996). Potential Theory in Gravity and Magnetics Applications, Cambridge University Press. Res. 58-59, 193-194

[4] GIT Controlador de versiones. http://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones

[5] VENING MEINESZ, F. A., J. H. F. UMBGROVE AND PH. H. KUENEN, Gravity expedition at sea. 1923-1932, vol. 2, Pub. Netherl. Geod. Comm., 208 pp. 1934.

[6] HUBBERT, M. KING, A line-integral method of computing the gravimetric effects of two-dimensional masses, Geophysics, 13, 215-225, 1948.

[7] International Association of Volcanology and Chemistry of the Earth, disponible en línea, http://en.wikipedia.org/wiki/International_Association_of_Volcanology_and_Chemistry_o f_the_Earth%27s_Interior

[8] Modelo SCRUM, disponible en línea. http://www.proyectosagiles.org/que-es-scrum

[9] Introduction to Object Oriented Programming Concepts (OOP) and More, disponible en línea, http://www.codeproject.com/Articles/22769/Introduction-to-Object-OrientedProgramming-Concep#Conclusion