UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

DISEÑO Y APLICACIÓN DE UN CÓDIGO PARA RESOLVER LA ECUACIÓN DE POISSON

TESIS

QUE PARA OBTENER EL TÍTULO DE
FÍSICO

PRESENTA:
PEDRO EDUARDO MEDINA GONZALEZ

TUTOR
FABIO DE COLLE

Ciudad de México, 2023

Dedicatoria

Dedico todo mi afán y esfuerzo invertido en esta tesis a mi madre y a mi padre quienes me han apoyado para poder llegar a esta instancia de mi formación, ya que ellos siempre han estado presentes para favorecerme moral y emocionalmente.

A mis hermanos que con mucho amor y cariño siempre he denodado ser una inspiración para ellos.

Agradecimientos

Doy mis más sinceros agradecimientos a la UNAM, que desde niño tuve la ambición de entrar a esta universidad. A la facultad de ciencias, escuela que me asombro y fascino desde mi primer clase. Agradezco a todos mis profesores que durante la carrera de Física lograron compartir sus saberes y experiencias, agradezco a mis compañeros y amigos que he conseguido a lo largo de estos últimos años por su carisma y cariño. Agradezco a mi familia, que gracias a ellos ha sido esto posible, desde el primer día, la primera clase que tomé hasta la conclusión de esta tesis. Y por último agrazeco al proyecto PAPIIT AG100820, gracias a su aportación logré concluir en tiempo y forma mi trabajo.

Resumen

Los sistemas auto-gravitantes son objetos que tienen una estructura definida por su propio campo de gravedad. La auto-gravedad es muy importante en objetos astrofísicos de gran escala, ya que determinan la estructura de las estrellas, de las galaxias y de la mayoría de los objetos observables en el universo. Las estrellas son los objetos auto-gravitantes por excelencia. Desde una perspectiva de equilibrio, una estrella es el resultado del equilibrio entre la presión que ejerce el material de la misma y la acción de la gravedad, que se oponen entre ellas. Al tener estas dos fuerzas en armonía tenemos como resultado una estrella.
En esta tesis abordaremos la solución de la ecuación de Poisson aplicada al potencial gravitacional, que describe como el potencial es determinado por la distribución de masa. La ecuación de Poisson es

$$\nabla^2\phi = 4\pi G\rho\,. \tag{1}$$

Donde G es la constante de gravitación universal, ϕ es el potencial definido en función de la densidad y ϕ es la fnución que define la densidad en cierta región del espacio.
La ecuación de Poisson es una ecuación en derivadas parciales, donde ∇^2 es el operador **Laplaciano**. El Laplaciano es un operador diferencial elíptico de segundo orden. La forma de dicho operador depende de las coordenadas donde el sistema esté definido (cartesianas, esféricas, polares, cilíndricas, etc). El Laplaciano relaciona un campo escalar y un campo vectorial.
Al conocer el potencial gravitacional de estos cuerpos, utilizaremos las ecuaciones de la hidrodinámica (las ecuaciones de Euler), que describen el comportamiento de fluidos. Estas ecuaciones representan la conservación de la masa, del momento y de la energía del fluido. Esto con el propósito de encontrar el estado de equilibrio hidrostático de un perfil de densidad determinado. También mostraremos las soluciones que nuestro código resuelva y las compararemos con soluciones analíticas, bajo distintas distribuciones de masa e indicaremos como es la evolución de las soluciones a través de las iteraciones en las que el algoritmo trabaje.
Resolveremos la ecuación de Poisson mediante distintos métodos iterativos que utilizan la discretización del perfil de densidad tales como:

- El método de Gauss-Seidel.
- El método de las sobre-relajaciones sucesivas.
- Variaciones del método multigrid.

También haremos una comparación entre los distintos métodos, para determinar cuál es el método más adecuado (es decir cuál de todos ellos llega a una solución con la menor cantidad de iteraciones) en función del tamaño de la malla y de la distribución de densidad.
Al igual, mostraremos como funciona una variación de nuestro código con perfiles de densidad de dos dimensiones. Por último, mostraremos los resultados de nuestro acoplamiento hidrodinámico, como se involucran las variables de energía, gravedad y fuente para conseguir un resultado en equilibrio en función a estas variables.
Los resultados de esta tesis serán fundamentales para estudiar a futuro problemas astrofísicos donde la auto-gravedad es importante, en particular la interacción de estrellas en sistemas binarios cerrados.

Índice general

1. Objetivo

El objetivo de esta tesis es diseñar un código en computadora que resuelva la ecuación de Poisson, utilizando diversos métodos, y comparar los métodos implementados. También, resolver la ecuación de Poisson acoplada a un código hidrodinámico para en un futuro poder modelar la evolución de sistemas en los que la auto-gravedad determina el comportamiento del sistema.

2. La ecuación de Poisson

Los sistemas auto-gravitantes son objetos que tienen una estructura definida por su propio campo gravitacional. La auto-gravedad es muy importante en objetos astrofísicos de gran escala. En esta tesis abordaremos la solución de la ecuación de Poisson ($\nabla^2 f = g$) aplicada al potencial gravitacional, que describe como el potencial es determinado por la distribución de masa. La ecuación de Poisson es

$$\nabla^2 \phi = 4\pi G \rho \, . \tag{2.1}$$

La ecuación de Poisson, como se muestra en la ecuación 2.1, es una ecuación en derivadas parciales, donde ∇^2 es el operador **laplaciano**. El Laplaciano es un operador diferencial elíptico de segundo orden. La forma de dicho operador depende de las coordenadas donde el sistema esté definido (cartesianas, esféricas, polares, cilíndricas, etc). El Laplaciano relaciona un campo escalar y un campo vectorial. En este trabajo, analizaremos distribuciones de densidad lineales (una sola dimensión) y superficiales (dos dimensiones).

La auto-gravedad es importante, principalmente, en problemas astrofísicos. Esto se debe a que los cuerpos de gran escala son lo suficientemente masivos como para experimentar efectos de auto-gravedad para ellos mismos o para la materia cercana a ellos. Uno de los efectos mas famosos de la auto-gravedad es conservar la forma esférica de los planetas. La auto-gravedad es importante también en ámbitos mas próximos, como por ejemplo en la sismología. Esto es debido a que la tierra puede experimentar las llamadas olas elásticas, que pueden cambiar el potencial gravitacional de la tierra mientras que las olas interactúan con las estructuras presentes en la superficie.

2.1. Derivación de la ecuación de Poisson

La ecuación de Poisson, específicamente aplicada a la auto-gravedad, se puede derivar directamente de la definición de densidad (masa por unidad de volumen):

$$\rho = \frac{dm}{dV} \, .$$

Suponiendo que el volumen de nuestra distribución de masa es una esfera, podemos aproximar el volumen tomando la superficie de una esfera, multiplicada por un grosor dr, como se muestra en la siguiente imagen.

El volumen está dado por

$$dV = A dr = 4\pi r^2 dr \, .$$

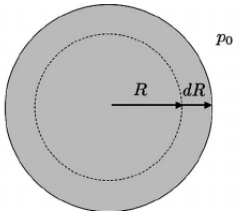

Figura 2.1: Diferencial de volumen de una esfera.

Entonces la definición de densidad se puede escribir como en función del radio de la esfera:

$$\rho = \frac{dm}{dV} = \frac{dm}{4\pi r^2 dr} ,$$
$$\Rightarrow \qquad \frac{dm}{dr} = 4\pi r^2 \rho .$$

Esta es la ecuación de conservación de masa. Ahora, tomemos en cuenta el potencial gravitacional, representado como:

$$-\nabla\phi = -\frac{\partial\phi}{\partial r} = g = -\frac{Gm}{r^2} .$$

Entonces:

$$\frac{\partial\phi}{\partial r} = \frac{Gm}{r^2} ,$$
$$r^2\frac{\partial\phi}{\partial r} = Gm ,$$
$$\frac{\partial}{\partial r}\left(r^2\frac{\partial\phi}{\partial r}\right) = G\frac{\partial m}{\partial r} = 4\pi r^2 G\rho ,$$
$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial\phi}{\partial r}\right) = \frac{G}{r^2}\frac{\partial m}{\partial r} = 4\pi G\rho .$$

El lado izquierdo de esta ecuación puede resumirse utilizando la siguiente definición:

$$\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\frac{\partial}{\partial r}\right) := \nabla^2 .$$

Entonces la ecuación de Poisson queda de la siguiente manera:

$$\nabla^2\phi = 4\pi G\rho .$$

2.2. La ecuación de Poisson en sistemas astrofísicos

En astrofísica y en la ciencia en general la gravedad es una fuerza que varía de importancia en función del propósito de la investigación, análisis o proyecto. Puede tener poca relevancia, si se trata de alguna investigación en el desarrollo de alguna medicina o tratamiento. Puede tener mucha relevancia, como al construir

un edificio o diseñar un mecanismo robótico que automatice algún proceso que involucre cargas excesivas. En otros campos toma el papel principal del estudio. Este trabajo entra en este último caso, en particular si centramos nuestra atención en las estrellas.

Las estrellas son los objetos autogravitantes por excelencia. Desde una perspectiva de equilibrio, una estrella es el resultado del equilibrio entre la presión que ejerce el material de la misma y la acción de la gravedad, que se oponen entre ellas. Al tener estas dos fuerzas en armonía tenemos como resultado una estrella.

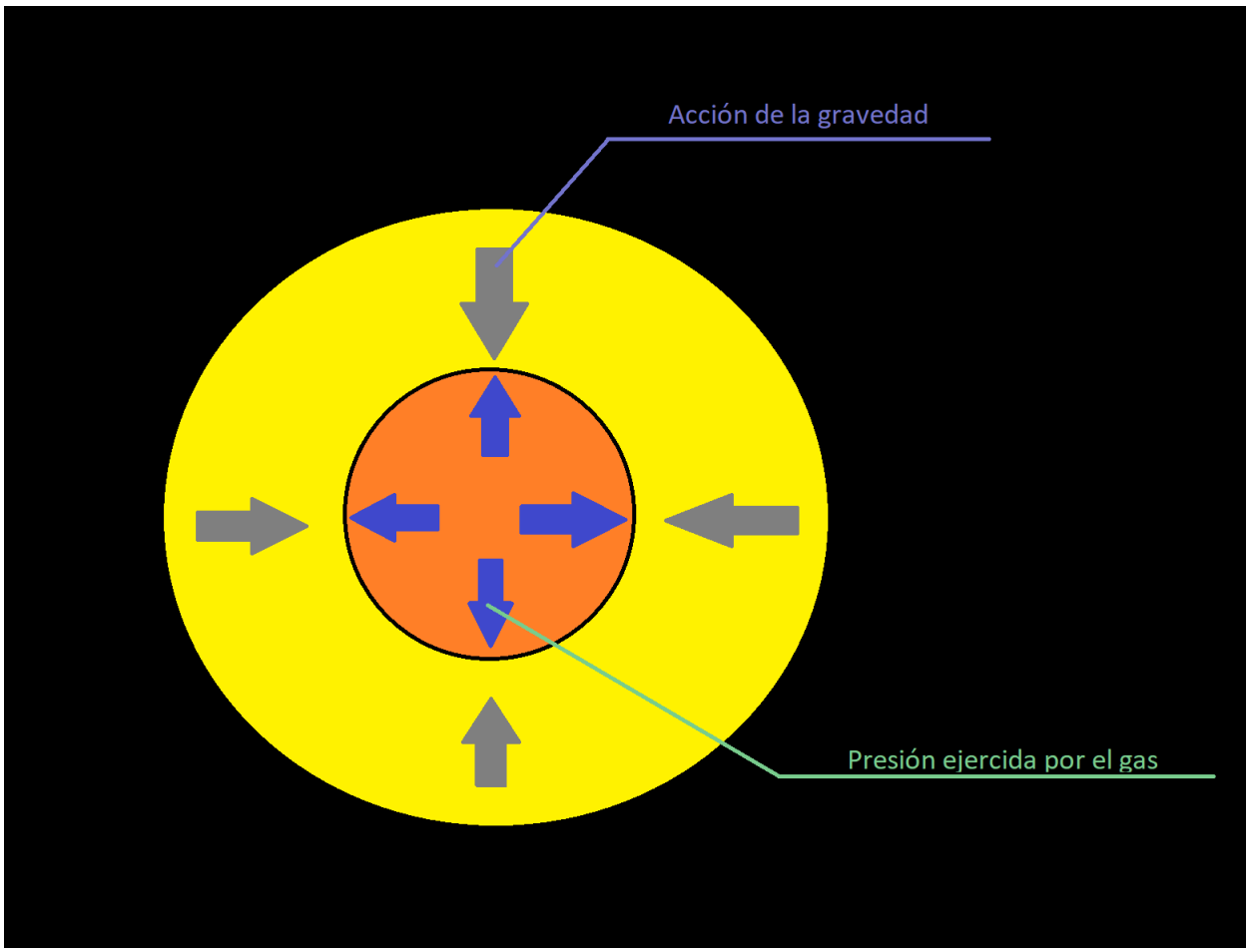

Figura 2.2: Equilibrio hidrostático en una estrella.

Teniendo esta idea en mente, podemos tomar la ecuación de la fuerza de gravedad de Newton, es decir la fuerza que actúa entre dos masas M_1 y M_2 a distancia r_{12} entre ellas:

$$F = -\frac{GM_1M_2}{r_{12}^2} \, .$$

Consideramos ahora la fuerza de gravedad entre la masa incluida en un cierto radio, y una capa delgada de estrella. Esto quiere decir: $M_1 = m(r)$, $M_2 = dm$ y $r_{12} = r$. El problema considerado se ve más claramente en la siguiente figura.

Entonces, la fuerza de gravedad que la masa m ejerce sobre la capa dm es:

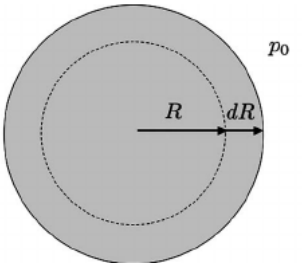

Figura 2.3: Distribución de la masa de una estrella.

$$dF = -\frac{Gm(r)dm}{r^2} \ .$$

Si hacemos uso de la definición de densidad $\rho = dm/dV$, tenemos:

$$dF = -\frac{Gm(r)\rho dV}{r^2} \ ,$$

$$\frac{dF}{dV} = -\frac{Gm(r)\rho}{r^2} \ ,$$

Tomando en consideración que el diferencial de volumen dV puede ser visto como

$$dV = dAdr \ ,$$

y al ser el volumen lo suficientemente suave como función entonces:

$$\frac{d^2}{dAdr} = \frac{d^2}{drdA}$$

Tenemos, entonces:

$$\frac{d^2F}{dAdr} = \frac{d^2F}{drdA} = \frac{d}{dr}\left(\frac{dF}{dA}\right) = \frac{dP}{dr} = -\frac{Gm(r)\rho}{r^2} \ ,$$

$$\frac{dP}{dr} = -\frac{Gm(r)\rho}{r^2} \ .$$

Este último resultado es la ecuación de equilibrio hidrostático, que determina la estructura de presión de una estrella necesaria para balancear perfectamente la fuerza de gravedad.

Para determinar la estructura de una estrella, no es necesario resolver la ecuación de Poisson gracias a la simetría esférica, que hace que la ecuación de Poisson sea equivalente a la ecuación de conservación de masa. Para la demostración de esta equivalencia, véase la sección 2.1.

Un ejemplo de un sistema donde hay que resolver explícitamente la ecuación de Poisson es un sistema binario de estrellas, es decir un sistema formado por dos estrellas que se encuentran girando una alrededor de la otra (o, con más exactitud,

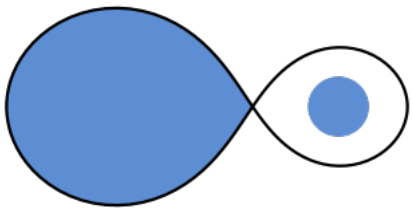

Figura 2.4: Lóbulo de Roche en un sistema binario (imagen tomada de *https://en.wikipedia.org/wiki/Roche_lobe*).

ambas estrellas orbitan entorno al centro de masa del sistema).

Si las estrellas se encuentran cercanas entre ellas, una de las dos puede llegar a llenar su **lóbulo de Roche** (figura 2.4). En astronomía, el lóbulo de Roche es una curva equipotencial, es decir la región sobre la cual el potencial de las dos estrellas y el potencial centrifugo tienen un valor fijo.
El potencial de Roche (en un punto $\vec{r}$ cualesquiera) es representado mediante la ecuación:

$$\Phi(\vec{r}) = -\frac{GM_1}{|\vec{r} - \vec{r_1}|} - \frac{GM_2}{|\vec{r} - \vec{r_2}|} - \frac{1}{2}(\Omega \times \vec{r})$$

donde M_1 y M_2 son las masas de las dos estrellas (con $M_1 \gg M_2$), a la separación orbital, es decir, la distancia entre los centros de las dos estrella, y M es la masa reducida del sistema:

$$\mu = \frac{M_1 * M_2}{M_1 + M_2} \ .$$

Los primeros dos términos de la derecha son los potenciales gravitacionales de cada estrella , y el tercer término es el momento angular del sistema binario. En esta ecuación, estamos considerando la órbita de cada estrella respecto el centro de masa del sistema y además también suponemos rotación síncrona. La velocidad angular Ω está dada por la tercera ley de Kepler:

$$\Omega = \left(\frac{G\mu}{a^3}\right)^{\frac{1}{2}} \ .$$

El momento angular es un vector normal ($\hat{n}$) perpendicular al plano orbital del sistema. Suponemos que el plano orbital es el plano XY, y por lo tanto $\hat{n}$ va en la dirección positiva del eje Z, equivalente a $\hat{k}$. En coordenadas cartesianas, el potencial se escribe como:

$$\Phi = -\frac{GM_1}{\sqrt{(x - Ma)^2 + y^2 + z^2}} - \frac{GM_2}{\sqrt{(x + [1 - M]a)^2 + y^2 + z^2}} - \frac{1}{2}\Omega^2(x^2 + y^2)^2 \ . \tag{2.2}$$

Donde M está dado por la ecuación:

$$M = \frac{M_2}{M_1 + M_2} \, .$$

Dada la ecuación del potencial (2.2) para un punto cualesquiera, si tomamos las curvas de nivel de la función Φ, dichas curvas nos mostrarán los puntos $\vec{r}$ donde el potencial Φ asume valores específicos. En particular, los puntos donde $\nabla\Phi = 0$ corresponden a puntos donde el gradiente del potencial es nulo, o en otras palabras, donde exista un equilibrio de las fuerzas gravitacionales y la fuerza centrípeta que tenga un objeto de prueba (que va en función de su velocidad o energía cinética). Este efecto puede apreciarse mejor en la figura 2.5.

El lóbulo de Roche definido por la región formada alrededor de una estrella en un sistema binario, dentro del cual la materia orbitando, está enlazado a esa estrella. La forma del lóbulo de Roche es similar al de una gota de lluvia.

A continuación se muestra una representación en tres dimensiones del potencial.

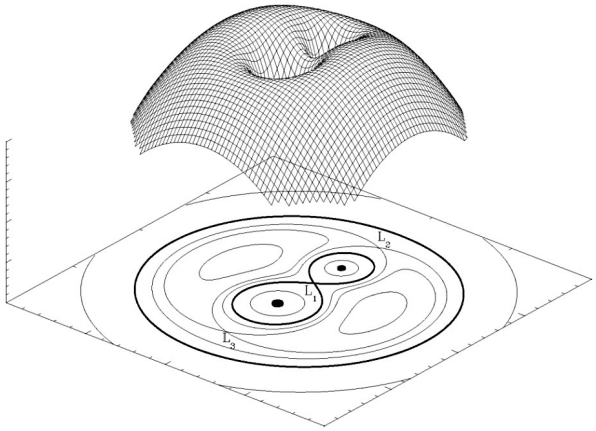

Figura 2.5: Potencial gravitacional generado por un sistema binario de estrellas (tomada de *https://en.wikipedia.org/wiki/Lóbulo_de_Roche*).

Esta figura muestra un pliego de malla, deformado acorde al potencial efectivo (es decir, la suma del potencial gravitacional más la fuerza centrífuga) del sistema binario, y el corte del potencial gravitacional en el plano $Z = 0$. El potencial asume la forma de un número « 8 » pero más estrecho en el centro, con una gota más pequeña que la otra en el caso donde las masas de las estrellas no sean iguales. Este cuerpo geométrico es el resultado de la distribución del potencial de gravedad.

Los puntos de Lagrange, también denominados puntos L o puntos de libración, son las diferentes posiciones donde una partícula de prueba (es decir, un objeto lo suficientemente pequeño para no afectar el potencial alrededor), solo afectado por la gravedad y la fuerza centrífuga, puede quedarse estacionario en el sistema

de referencia no inercial.
Otro efecto interesante relacionado a este tema es el **Roche Lobe overflow**, durante el cual el gas de una de las estrellas desborda su lóbulo de Roche y es acretado sobre la estrella compañera. Cuando hay Roche lobe overflow, el sistema no es ni esféricamente simétrico, ni estático. En este caso, es necesario resolver las ecuaciones de hidrodinámica (las cuales se abordarán en la siguiente sección), y para este caso se necesita encontrar el potencial gravitacional mediante la solución de la ecuación de Poisson.

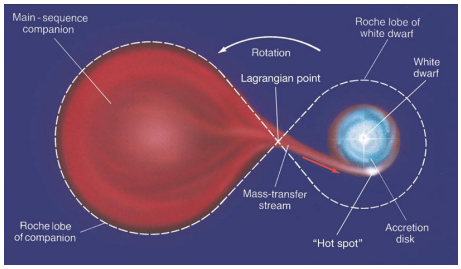

Figura 2.6: Roche Lobe Overflow. (Imagen tomada de *http://web.pd.astro.it/mapelli/lecture4_mapelli.pdf*)

2.3. Ecuaciones de la hidrodinámica

Al conocer el potencial gravitacional de estos cuerpos, utilizaremos las ecuaciones de Euler, que describen el comportamiento de fluidos. Estas ecuaciones representan la conservación de la masa, del momento y de la energía del fluido:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{u}) = 0 \,, \tag{2.3}$$

$$\frac{\partial (\rho \vec{u})}{\partial t} + \nabla \cdot (\rho \vec{u} \vec{u}) + \nabla P = -\rho \nabla \phi \,, \tag{2.4}$$

$$\frac{\partial e}{\partial t} + \nabla \cdot ((e + P)\vec{u}) = -\rho \vec{u} \cdot \nabla \phi \,, \tag{2.5}$$

donde ρ es es la densidad de la distribución de masa, el término $e = p/(\gamma - 1) + \rho v/2$ es la energía total (dada por la suma de las energía térmica y cinética, γ es el coeficiente adiabático de un gas ideal, t es la variable de tiempo, $\nabla\cdot$ es la divergencia de un campo vectorial, ∇ es el gradiente de un campo escalar, ϕ es el potencial gravitacional de la distribución de masa y $\vec{u}$ es el vector de velocidad del flujo.

Además, estas ecuaciones están acopladas con la ecuación de Poisson:

$$\nabla^2 \phi = 4\pi G \rho \,, \tag{2.6}$$

que permite determinar el gradiente del potencial gravitacional que entra en las ecuaciones de ocnservación de momento y energía.

3. Métodos

En este capítulo describiremos diferentes métodos para resolver la ecuación de Poisson. Estos métodos se basan en la discretización de la ecuación diferencial en una malla.

3.1. Solución de la ecuación de Poisson

Consideramos una malla de dos dimensiones en coordenadas cartesianas. La ecuación de Poisson es:

$$\frac{\partial^2\Phi}{\partial x^2} + \frac{\partial^2\Phi}{\partial y^2} = 4\pi G\rho(x,y) \; .$$

Esta ecuación se puede discretizar como:

$$\frac{\phi_{j+1,k} - 2\phi_{j,k} + \phi_{j-1,k}}{(\Delta x)^2} + \frac{\phi_{j,k+1} - 2\phi_{j,k} + \phi_{j,k-1}}{(\Delta y)^2} = 4\pi G\rho_{j,k} \; ,$$

donde j, el índice en la dirección x, corre desde 0 hasta J, k, el índice en la dirección y, corre desde 0 hasta K, y Δx y Δy son, respectivamente, el tamaño de las celdas a lo largo de los ejes X y Y.
Podemos organizar los valores de Φ en una secuencia unidimensional dejando que el punto (j,k) corresponda al índice m, donde:

$$m = j(K+1) + k \; .$$

Si suponemos constante el tamaño de las celdas $\Delta x = \Delta y$, entonces la ecuación se convierte en:

$$\phi_{m+K+1} + \phi_{m-K-1} + \phi_{m+1} + \phi_{m-1} - 4\phi_m = (\Delta x)^2 4\pi G\rho_m \; ,$$

donde ρ_m son cantidades conocidas. Los valores en la frontera deben también ser especificados. En $j = 0$, en particular, $\phi_B(0,k)$ debe ser especificado para $k = 0$ hasta K o, equivalentemente, $m = 0, 1, ..., K$. En los límites $j = J$, $\phi_B(J,k)$ debe también ser especificado para $m = J(K+1)$ hasta $J(K+1)+K$. Similarmente, en $k = 0$, $\phi_B(j,0)$ tiene que especificarse para $j = 0$ hasta J o $m = 0, K+1, 2(K+1), ..., J(K+1)$ y finalmente para $k = K$, $\phi_B(j,K)$ debe se especificado para $j = 0$ hasta J o $m = K, 2K+1, 3K+2, ..., (J+1)K+J$.

Ahora discutiremos como se discretiza el espacio. Por simplicidad, consideremos una región cuadrada en dos dimensiones. Podríamos discretizar este cuadrado al subdividirlo en rectángulos más pequeños inscritos (figura 3.1). El único inconveniente sería que existirían secciones más grandes que otras y eso haría el esquema

numérico mas complejo. Otra forma de discretizar este cuadrado sería como se ilustra en la figura 3.1 (panel derecho). En este caso la discretización del rectángulo es irregular. Este tipo de malla es útil para estudiar diferentes problemas, como puede ser el análisis de elemento finito al diseñar piezas mecánicas, pero para nuestros propósitos, añade un factor de complejidad innecesario.

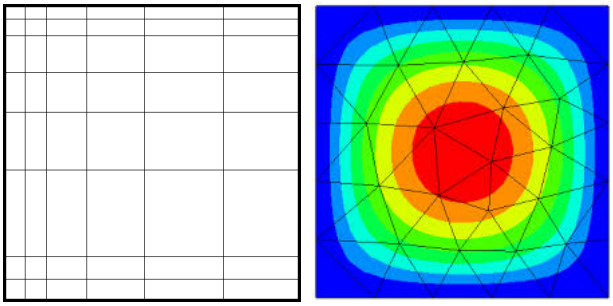

Figura 3.1: Ejemplo de malla estructurada y no estructurada, es decir, con celdas rectangulares (izquierda), y con celdas triangulares (derecha). Imagen tomada de $romsaf.org/GRM-19.php$

Para tener una malla que pueda simplificar los cálculos, tanto analíticos como computacionales, optamos por una malla cuyas celdas sean todas del mismo tamaño, haciendo una similitud con una matriz de dos dimensiones de tamaño $N \times M$, como se muestra en la figura 3.2.

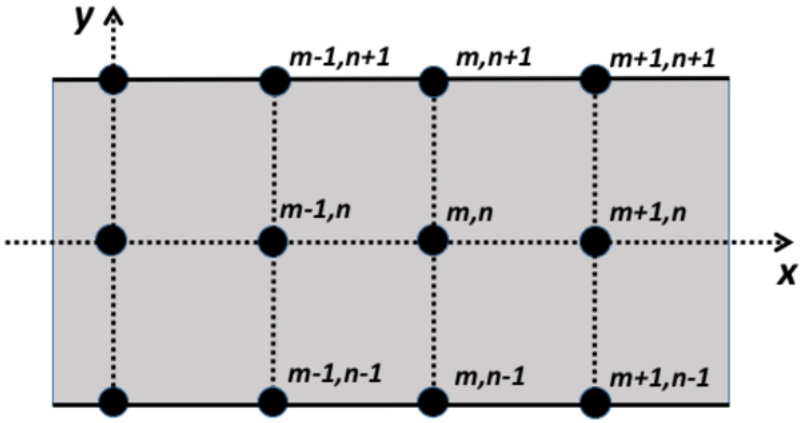

Figura 3.2: Ejemplo de malla regular, con una estructura similar a la de una matriz (imagen tomada de $wikipedia.org/wiki/Lattice\_graph$).

3.2. Condiciones de frontera

En todas las clases de soluciones basadas en ecuaciones diferenciales, las condiciones de frontera deben ser aplicadas por separado como un paso adicional. Como discutimos en la sección de la solución a la ecuación de Poisson, tenemos que especificar como tiene que ser el comportamiento de nuestro potencial efectivo ϕ en las fronteras de nuestra área de estudio. Para el caso específico de la ecuación de Poisson, en esta tesis consideraremos valores nulos de ϕ en la frontera. Es decir,

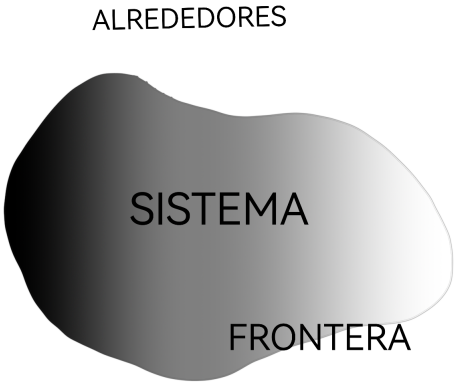

Figura 3.3: Ejemplo de distribución de masa en un plano 2D.

si tenemos una malla de tamaño $L \times K$ la ecuación que debemos resolver se puede ver como:

$$\begin{cases} \nabla^2\phi(x_{ij}) = 4\pi G\rho(x_{ij}) & \text{si} \quad i \neq 0, L \ \mathbf{y} \ j \neq 0, K \\ \\ \phi = 0 & \text{si} \quad i = 0, L \ \mathbf{o} \ j = 0, K \end{cases}$$

La solución para el potencial viene dada por la ecuación de Poisson y la distribución de la masa $\rho(x_{ij})$.

3.3. Método de Jácobi

El método de Jácobi propone una solución iterativa a una ecuación diferencial. En cada paso, la siguiente solución será una combinación lineal de nuestra actual solución y la función fuente de la ecuación a resolver. El método de Jacobi empieza con la ecuación de Poisson. En una dimensión, es:

$$\frac{\partial^2 \phi}{\partial x^2} = 4\pi G\rho \, .$$

Para adaptarla a un método iterativo, reescribimos esta ecuación como:

$$\frac{\partial \phi}{\partial \tau} = \frac{\partial^2 \phi}{\partial x^2} - 4\pi G\rho \, .$$

Esta es una ecuación elíptica, similar por ejemplo a la ecuación de difusión

$$\frac{\partial u}{\partial \tau} = D\frac{\partial^2 u}{\partial x^2} \, ,$$

donde D es el coeficiente de difusión, que para nuestro caso es $D = 1$ (por cómo está estructurada la ecuación de Poisson).

Discretizamos esta ecuación de la siguiente forma:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta\tau} = \frac{1}{\Delta x}\left[\left(\frac{\partial\phi}{\partial x}\right)_{i+1/2} - \left(\frac{\partial\phi}{\partial x}\right)_{i-1/2}\right] - 4\pi G\rho_i \, ,$$

$$\phi_i^{n+1} - \phi_i^n = \frac{\Delta\tau}{\Delta x}\left[\left(\frac{\phi_{i+1}^n - \phi_i^n}{\Delta x}\right) - \left(\frac{\phi_i^n - \phi_{i-1}^n}{\Delta x}\right)\right] - 4\pi G\rho_i \, ,$$

$$\phi_i^{n+1} - \phi_i^n = \frac{\Delta\tau}{(\Delta x)^2}\left[(\phi_{i+1}^n - \phi_i^n) - (\phi_i^n - \phi_{i-1}^n)\right] - 4\pi G\rho_i \, ,$$

$$\phi_i^{n+1} - \phi_i^n = \frac{\Delta\tau}{(\Delta x)^2}\left(\phi_{i+1}^n + \phi_{i-1}^n - 2\phi_i^n\right) - 4\pi G\rho_i \, ,$$

$$\phi_i^{n+1} = \phi_i^n + \frac{\Delta\tau}{(\Delta x)^2}\left(\phi_{i+1}^n + \phi_{i-1}^n - 2\phi_i^n\right) - 4\pi G\rho_i \, .$$

Siendo esta una ecuación elíptica, es estable bajo el criterio:

$$\Delta\tau \leq \frac{1}{2}(\Delta x)^2 \, ,$$

Para garantizar la estabilidad tenemos que escoger un valor para $\Delta\tau$ que obedezca esta desigualdad. Por simplicidad y para que solo trabajemos con la menor cantidad de variables, escogemos un valor de

$$\Delta\tau = \frac{(\Delta x)^2}{2} \, .$$

Dadas estas condiciones entonces nos quedamos solo con la siguiente ecuación:

$$\phi_i^{n+1} = \phi_i^n + \frac{\Delta\tau}{(\Delta x)^2}\left(\phi_{i+1}^n + \phi_{i-1}^n - 2\phi_i^n\right) - 4\pi G\rho_i \ ,$$

$$\phi_i^{n+1} = \phi_i^n + \frac{\frac{(\Delta x)^2}{2}}{(\Delta x)^2}\left(\phi_{i+1}^n + \phi_{i-1}^n - 2\phi_i^n\right) - 4\pi G\rho_i \ ,$$

$$\phi_i^{n+1} = \phi_i^n - \phi_i^n + \frac{1}{2}\left(\phi_{i+1}^n + \phi_{i-1}^n\right) - 4\pi G\rho_i \ ,$$

$$\phi_i^{n+1} = \frac{1}{2}\left(\phi_{i+1}^n + \phi_{i-1}^n\right) - 4\pi G\rho_i \ .$$

Este método nos permite encontrar, mediante una solución ya conocida ϕ^n, otra solución más certera ϕ^{n+1} en un modelo iterativo, el cual converge después de generar cierto número de ciclos o cuando el error entre soluciones $\phi^{n+1} - \phi^n$ es menor a cierto número especificado en el código. Es un método parecido al de Newton-Raphson en el sentido que empezamos desde un valor determinado (para nuestro caso $\phi^0 = 0$) y con el método nos vamos acercando a la solución, no necesariamente una solución que no genere un error nulo, pero sí una que se acerque a la solución exacta.
El método puede ser fácilmente generalizado para problemas en dos dimensiones. En este caso, y suponiendo que $\Delta x = \Delta y$, tenemos que

$$\phi_{j,k}^{n+1} - \phi_{j,k}^n = \frac{\Delta\tau}{(\Delta x)^2}(\phi_{j+1,k}^n + \phi_{j-1,k}^n + \phi_{j,k+1}^n + \phi_{j,k-1}^n - 4\phi_{j,k}^n) - 4\pi G\rho_{j,k}\Delta\tau$$

Esta ecuación puede verse como:

$$\phi_{j,k}^{n+1} = \phi_{j,k}^n\left(1 - \frac{4\Delta\tau}{(\Delta x)^2}\right) + \frac{\Delta\tau}{(\Delta x)^2}(\phi_{j+1,k}^n + \phi_{j-1,k}^n + \phi_{j,k+1}^n + \phi_{j,k-1}^n) - 4\pi G\rho_{j,k}(\Delta x)^2$$

En dos dimensiones, el criterio de estabilidad se convierte en:

$$\frac{\Delta\tau}{(\Delta x)^2} + \frac{\Delta\tau}{(\Delta y)^2} \leq \frac{1}{2}$$

El método de *Jacobi* usa el paso de pseudo-tiempo máximo permitido. Si $\Delta\tau = 0{,}25(\Delta x)^2$, entonces, la ecuación cambia a:

$$\phi_{j,k}^{n+1} = \frac{1}{4}(\phi_{j+1,k}^n + \phi_{j-1,k}^n + \phi_{j,k+1}^n + \phi_{j,k-1}^n) - 4\pi G\rho_{j,k}(\Delta x)^2$$

Entonces, el nuevo estimado para ϕ involucra un promedio de cuatro valores de vecindad en la red, además de un termino fuente.

3.4. Método de Gauss-Seidel

En el área del análisis numérico el **método de Gauss-Seidel** es un método iterativo utilizado para resolver sistemas de ecuaciones diferenciales. El método tiene este nombre en honor a los matemáticos Carl Friedrich Gauss y Phillip Seidel y es similar al método de Jacobi.
Este método puede aplicarse a cualquier tipo de sistema de ecuaciones lineales que pueda ser modelado como una matriz de coeficientes con los elementos de su diagonal no nulos.
De manera breve el método de Gauss-Seidel es una versión acelerada del método de Jacobi. En el método de Jacobi se calculan los valores del potencial ϕ en todas la malla en un tiempo $n+1$ usando los valores al tiempo n. Por otro lado, en el método de Gauss-Seidel se remplaza ϕ_j^n por ϕ_j^{n+1} tan pronto como ha sido calculado (computacionalmente). Un ejemplo del método es el siguiente:

$$\phi_1^{n+1} = \frac{1}{2}\left(\phi_2^n + \phi_0^n\right) - 4\pi G\rho_1 \ ,$$

$$\phi_2^{n+1} = \frac{1}{2}\left(\phi_3^n + \phi_1^{n+1}\right) - 4\pi G\rho_2 \ ,$$

$$\phi_3^{n+1} = \frac{1}{2}\left(\phi_4^n + \phi_2^{n+1}\right) - 4\pi G\rho_3 \ ,$$

$$\ldots$$

Los métodos de Jácobi y Gauss-Seidel son los equivalentes en la solución de ecuaciones lineales al método de aproximadamente sucesivas de ecuaciones de tipo algebraicas. Consiste básicamente en obtener una ecuación de recurrencia y proponer una solución inicial, para posteriormente ejecutar las iteraciones necesarias hasta que la diferencia entre dos vectores consecutivos cumpla con una tolerancia definida.

3.5. Método de las sobre-relajaciones sucesivas

El método de la sobre-relajación sucesiva (*Successive Overrelaxation*, SOR, en inglés), es una variante del método de Gauss-Seidel para estimar la solución de un sistema lineal de ecuaciones, permitiendo una convergencia a la solución más rápida. Este método ha sido propuesto en 1950 por David M. Young.

Este método está relacionado con Gauss-Seidel ya que se toma un valor $\gamma \in [-1, 1]$ y se involucra en la ecuación:

$$\phi^{n+1} = (1-\gamma)\phi^n + \gamma GS(\phi) \ ,$$

donde ϕ^n es la n-sima solución, y $GS(\phi)$ es la solución de la ecuacion de Poisson usando el metodo de Gauss-Seidel. Si $\gamma = 1$ caemos totalmente en el método de Gauss-Siedel.

Si el parámetro γ es menor que 1, el procedimiento es conocido como **Underrelaxation**; por el otro lado, si vale entre 1 y 2, es llamado **Overrelaxation**. Y si es mas grande que 2, el procedimiento es numéricamente inestable. Como una guía útil, Press et al. (1992) ha mostrado que, en una malla cartesiana con J puntos de división en la dirección X de tamaño Δx y K puntos de malla en la dirección Y del mismo tamaño de Δx, el valor óptimo es:

$$\gamma = \frac{2}{1 + (1 - r_s^2)^{\frac{1}{2}}}$$

Donde el radio espectral r está dado por:

$$r_s = \frac{1}{2}\left[\cos(\pi/J) + \cos(\pi/K)\right]$$

Este método es útil, ya que si podemos controlar automáticamente el valor de γ el metodo puede converger con un numero menor de iteraciones.

3.6. Método *Multigrid*

En análisis numérico, el método *multigrid* es un algoritmo para resolver ecuaciones diferenciales usando la jerarquización y la discretización. La idea principal del método *multigrid* es acelerar la convergencia de un método iterativo básico mediante una corrección global de aproximación a la solución en una malla con cierto espesor. Este proceso recursivo es repetido hasta que la solución converge a una función definida por una ecuación diferencial.
El método *multigrid* es una extensión de la solución iterativa con el propósito de acelerar la razón de convergencia. El problema con los métodos estándar es que mientras eliminan aceleradamente errores sobre longitudes de onda cortas (con tamaños correspondientes al tamaño de cada celda en la malla), son ineficientes eliminando el error con longitudes de onda largas.
La razón es que la interacción entre una celda de la malla es solo con su vecindad durante cada iteración. Entonces, para dejar el error en algún punto especifico de la longitud de la onda, la información necesita ser prolongada al menos una vez a través de la onda. Por tanto, para una malla con cientos de céldas, el número de iteraciones necesario para lograr la convergencia es muy alto. Esto puede ser solucionado incluyendo mallas con celdas de distintos tamaños.

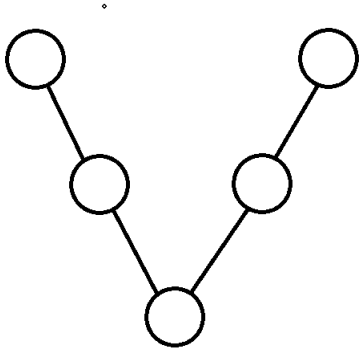

Figura 3.4: Procedimiento tipo V para el método *multigrid*.

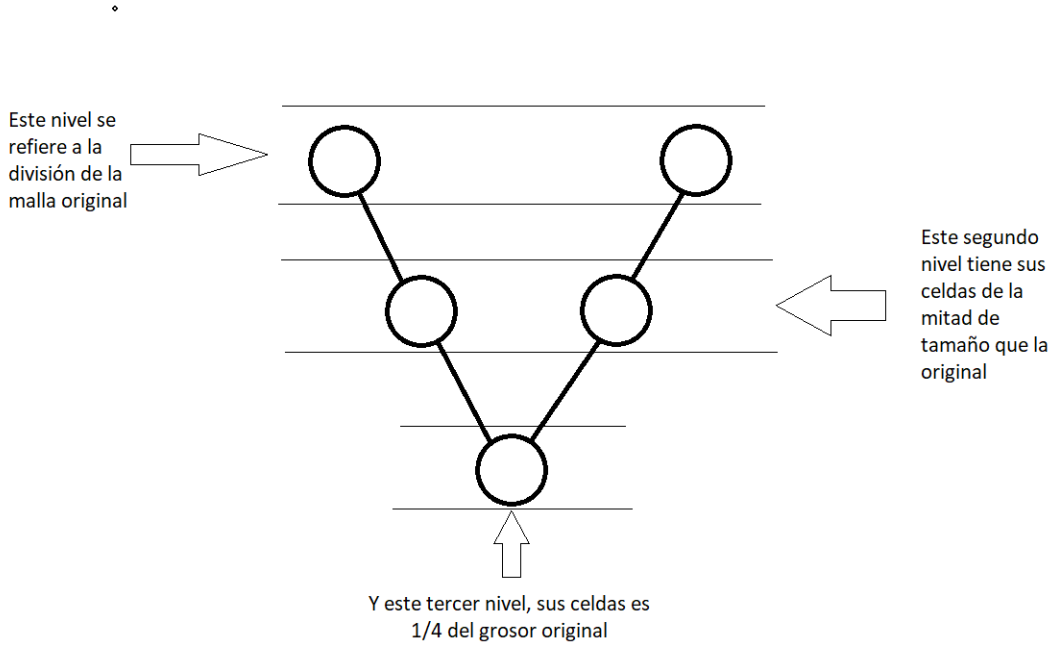

Figura 3.5: Detalle del algoritmo usado en un método *multigrid*.

3.6.1. Multigrid: iteración tipo "V"

El procedimiento mas simple es llamado ciclo-V (véase las figuras 3.4 y 3.5): se comienza con una solución aproximada en la malla más fina, con una iteración de Gauss-Seidel, para después promediar la solución en la siguientes malla y repetir el procedimiento en mallas mas gruesas. En mallas mas gruesas, los errores sobre escalas mas grandes se filtran mas rápidamente, y esta información es enviada de vuelta a las mallas mas finas.
En las figuras 3.4 y 3.5, los círculos representan una solución de la ecuación de Poisson en función del grosor de la malla en cada nivel, y las lineas representan el orden de las soluciones.

3.6.2. Multigrid: iteración tipo "W"

Otra de estas variaciones viene dada por el método tipo 'W':

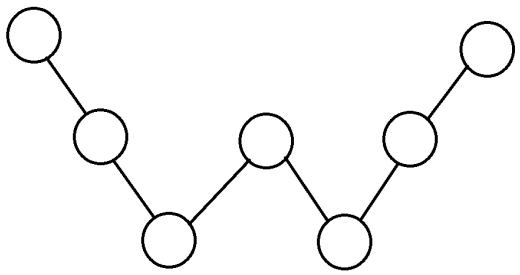

Figura 3.6: Procedimiento tipo W multigrid.

Con este esquema representamos un procedimiento que empieza con el primer círculo de la esquina superior izquierda, aquí se hace una iteración usando el método SOR (Sobre Relajación Sucesivas). El resultado se proyecta sobre una malla mas gruesa (segundo círculo desde la izquierda) promediando los valores obtenidos con la primera iteración. En esta malla mas gruesa se hace una nueva iteración usando el método SOR, repitiendo el proceso. Una vez que se tiene una solución sobre una malla fina, esta se extrapola sobre una malla gruesa promediando los valores. Cuando se pasa la información de una malla gruesa a una fina, por otro lado, se interpolan los valores obtenidos. Si la malla original tiene por ejemplo 128 celdas, el proceso completo consiste en hacer una iteración en una malla de 128 celdas, en una malla de 64, una de 32, una de 64, una de 32, una de 64 y, finalmente, una de 128.

3.7. Acoplamiento con el código hidrodinámico

Las ecuaciones de Euler describen el comportamiento de un fluido sin viscosidad. En dos dimensiones, en coordenadas cartesianas estas son:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho v_x}{\partial x} + \frac{\partial \rho v_y}{\partial y} = 0 \,, \tag{3.1}$$

$$\frac{\partial \rho v_x}{\partial t} + \frac{\partial (\rho v_x^2 + P)}{\partial x} + \frac{\partial \rho v_x v_y}{\partial y} = -\rho \frac{\partial \phi}{\partial x} \,, \tag{3.2}$$

$$\frac{\partial \rho v_y}{\partial t} + \frac{\partial (\rho v_y^2 + P)}{\partial y} + \frac{\partial \rho v_x v_y}{\partial x} = -\rho \frac{\partial \phi}{\partial y} \,, \tag{3.3}$$

$$\frac{\partial e}{\partial t} + \frac{\partial (e + P) v_x}{\partial x} + \frac{\partial (e + P) v_y}{\partial y} = -\rho v_x \frac{\partial \phi}{\partial x} - \rho v_y \frac{\partial \phi}{\partial y} \,. \tag{3.4}$$

Las ecuaciones de la hidrodinámica se pueden también escribir en forma de ley de conservación como:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = S \,, \tag{3.5}$$

donde el vector de las variables conservadas U se define como

$$U = (\rho, \rho v_x, \rho v_y, e) \,, \tag{3.6}$$

los vectores F y G son los flujos de las variables conservadas:

$$F = (\rho v_x, \rho v_x^2 + P, \rho v_x v_y, (e + P) v_x) \,, \tag{3.7}$$

$$G = (\rho v_y, \rho v_x v_y, \rho v_y^2 + P, (e + P) v_y) \,, \tag{3.8}$$

y el vector S representa los términos fuente

$$S = \left(0, -\rho \frac{\partial \phi}{\partial x}, -\rho \frac{\partial \phi}{\partial y}, -\rho v_x \frac{\partial \phi}{\partial x} - \rho v_y \frac{\partial \phi}{\partial y}\right) \,. \tag{3.9}$$

En estos términos está incluido el potencial debido al perfil de masa con el que estamos trabajando. En estas ecuaciones, la energía (por unidad de volumen) está dada por

$$e = \frac{1}{2} \rho v^2 + \frac{P}{\gamma - 1} \,, \tag{3.10}$$

donde γ es el índice adiabático, que en el caso de un gas adiabático, mono-atómico vale $5/3$.

Estas ecuaciones están acopladas a la ecuación de Poisson

$$\nabla^2 \phi = 4\pi G \rho \,. \tag{3.11}$$

La derivada temporal se discretiza como:

$$\frac{\partial U}{\partial t} = \frac{U_{ij}^{n+1} - U_{ij}^{n}}{\Delta t} \,, \tag{3.12}$$

donde el valor de Δt viene dada por

$$\Delta t = \frac{\text{mín}_{i,j}(\Delta x_i, \Delta y_j)}{\text{máx}_{i,j}(v_{i,j} + c_{s,i,j})} \,, \tag{3.13}$$

donde $\Delta x_i, \Delta y_j$ son los tamaños de las celdas, $v_{i,j}$ es la velocidad definida en cada punto de la malla, y $c_{s,i,j}$ es la velocidad del sonido definida como

$$c_{s,i,j} = \sqrt{\frac{\gamma P_{i,j}}{\rho_{i,j}}} \,. \tag{3.14}$$

Para simplificar las ecuaciones, y considerando que los tamanõs de las celdas $\Delta x_i = \Delta y_j = \Delta x$ son iguales, tomamos

$$\Delta t = \frac{\Delta x}{\text{máx}_{i,j}(v_{i,j} + c_{s,i,j})} \,, \tag{3.15}$$

Ahora consideramos las derivadas espaciales de los flujos

$$\frac{\partial F}{\partial x} \,. \tag{3.16}$$

En este caso la discretización es sobre una variable espacial. Para determinar el cambio diferencial que expresa esta derivada tomamos en cuenta los valores de las variables que tendríamos entre celdas. Este efecto lo podemos ver en la figura 3.7:

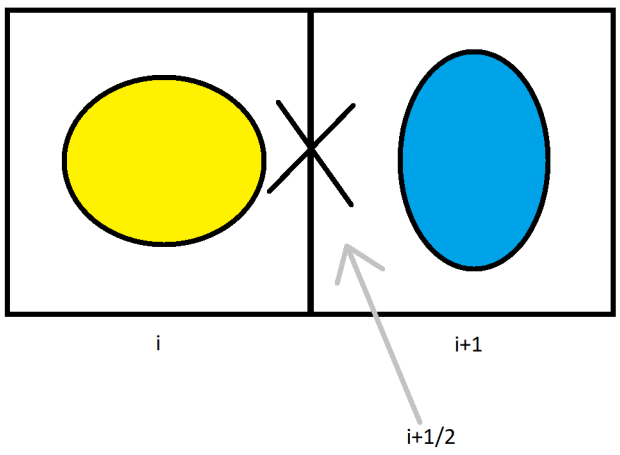

Figura 3.7: Esquema de dos celdas usadas para discretizar la derivada espacial.

La discretización entonces involucra los términos $F_{i+1/2}^{n}$ y $F_{i-1/2}^{n}$, evaluados en las interfaces entre celdas. Tenemos:

$$\frac{\partial F}{\partial x} = \frac{F_{i+1/2}^{n} - F_{i-1/2}^{n}}{\Delta x} \,, \tag{3.17}$$

Análogamente para los flujos en la direccion y, tenemos

$$\frac{\partial G}{\partial y} = \frac{G^n_{j+1/2} - G^n_{j-1/2}}{\Delta y} \ . \tag{3.18}$$

Con todas estas consideraciones y tomando los términos fuentes como $S(U^n)$, las ecuaciones de la hidrodinámica se discretizan como

$$U^{n+1}_{ij} = U^n_{ij} - \frac{F^n_{i+1/2,j} - F^n_{i-1/2,n}}{\Delta x} - \frac{G^n_{i,j+1/2} - G^n_{i,j-1/2}}{\Delta y} + S(U^n_{i,j}) \ , \tag{3.19}$$

Para calcular los términos fuente, el diferencial del potencial se discretiza como:

$$\frac{\partial \phi}{\partial x} = \frac{\phi_{i+1,j} - \phi_{i-1,j}}{2\Delta x} \ , \tag{3.20}$$

$$\frac{\partial \phi}{\partial y} = \frac{\phi_{i,j+1} - \phi_{i,j-1}}{2\Delta y} \ . \tag{3.21}$$

4. Resultados

4.1. Implementación del método de Gauss-Seidel

Para encontrar la solución numérica de la ecuación de Poisson, necesitamos definir la distribución de densidad y la malla computacional sobre la cual integrarla. Para simplificar más los cálculos, tomaremos los valores de la densidad ρ y la posición $\vec{x}$ como magnitudes sin dimensiones. Consideramos una densidad variable como función de la posición X, en una sola dimensión. Empezamos con la densidad representada en la figura 4.1.

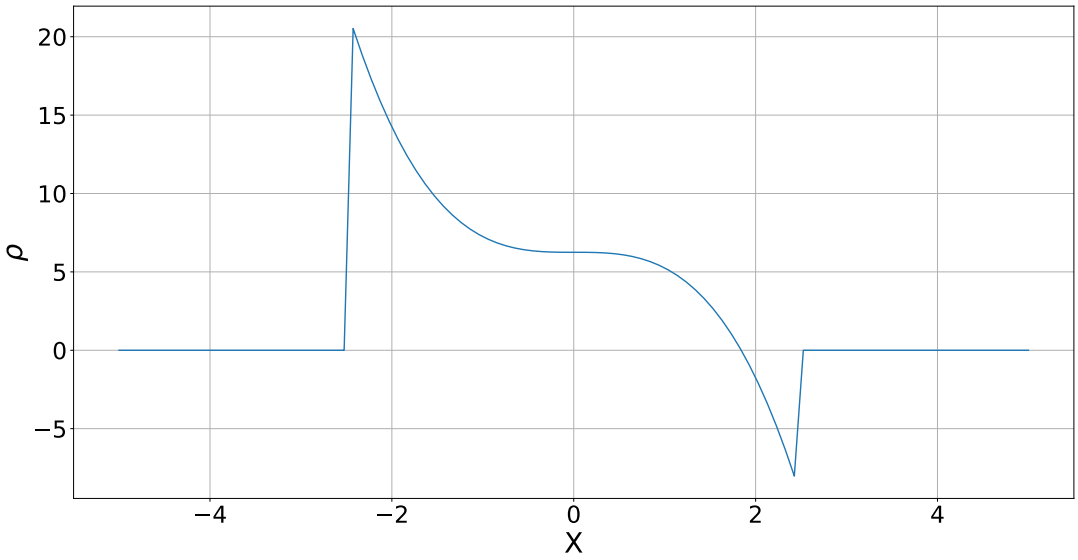

Figura 4.1: Perfil de densidad usado para el calculo del potencial gravitacional por medio de la ecuación de Poisson.

Dicha distribución está dada por la función:

$$\rho(x) = \begin{cases} -x^3 + 5/2 & \text{si} \quad x \in [-2.5, 2.5] \\ 0 & \text{si} \quad x \notin [-2.5, 2.5] \end{cases}$$

Dividimos el eje $\vec{X} \in [-5, 5]$ en 199 celdas (además de incluir dos celdas para las condiciones de frontera) del mismo tamaño. Cada celda tiene un tamaño de 0.1 (en unidades del código) y está centrada en:

$$\vec{X} = [-4.95, \text{-}4.90, \text{-}4.85, \cdots, 4.85, 4.90, 4.95]$$

Además de declarar el vector de densidad, se definen tres vectores ϕ_1, ϕ_2 y ϕ que representan el potencial efectivo y que nos apoyan con el calculo de la solución, como se describirá en detalle más adelante.
El siguiente paso para encontrar la solución es determinar una iteración o ciclo en el cual se puedan ejecutar las instrucciones necesarias. El algoritmo es el siguiente:

- Defino los valores de frontera del vector solución $\phi(0)$ y $\phi(N)$ iguales a cero, donde 0 y N son el primero y último elemento del vector $\vec{X}$.
- Defino un sub-ciclo donde se determinan los valores de ϕ por medio de la ecuación:
$$\phi_2(i) = \frac{\phi_1(i-1) + \phi_1(i+1) - \rho(i)}{2} \,,$$
donde ϕ_1 está definido inicialmente como un vector puramente de ceros y ρ es el vector de distribución para la densidad (en lo que sigue, vamos a considerar por simplicidad $4\pi G = 1$).
- Al final de cada ciclo el vector ϕ_1 hereda los valores de ϕ_2 para que al empezar la siguiente iteración se puedan tomar en cuenta los nuevos cambios.
- Se itera empezando un nuevo ciclo hasta que se cumplan un determinado número de pasos o que el error (definido abajo) sea menor que un valor crítico.

Para la distribución de densidad mostrada anteriormente, la solución de la ecuación de Poisson se muestra a continuación:

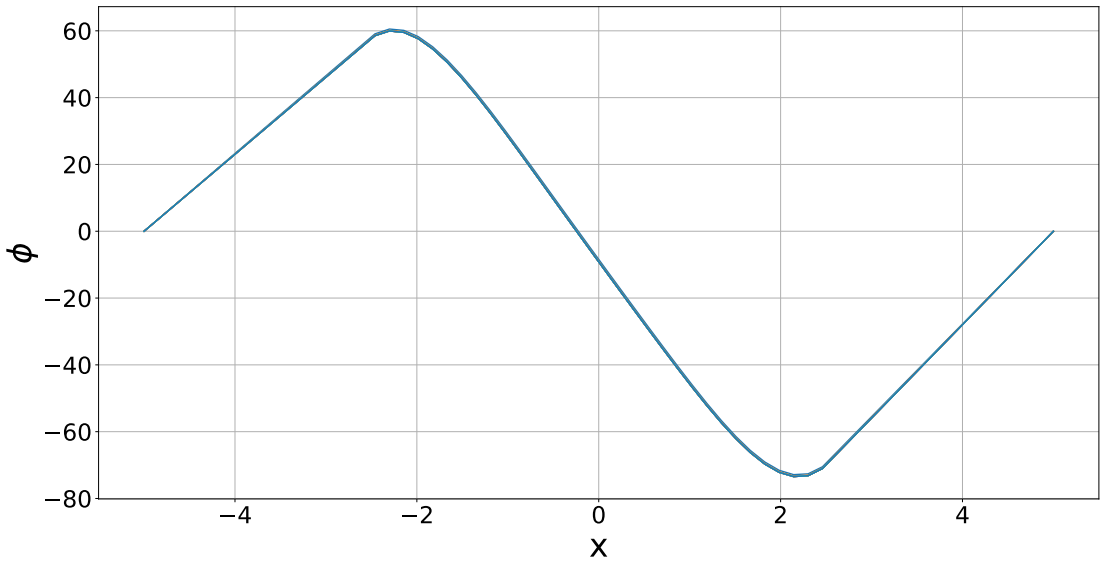

Figura 4.2: Potencial gravitacional obtenido para el perfil de densidad mostrado en la figura 4.1 después de 5000 iteraciones.

Esta solución se obtuvo como parte de un proceso iterativo. En la figura 4.3 se observa como avanza la solución durante las diferentes iteraciones para el perfil de densidad definido como $\rho(x) = -x^3 + 5/2$ (figura 4.1).

Observar la evolución y la convergencia de la solución durante distintas iteraciones nos ayudará a comprender como es que cambia y evoluciona el comportamiento de la función solución a través de los diferentes ciclos en los que nuestro

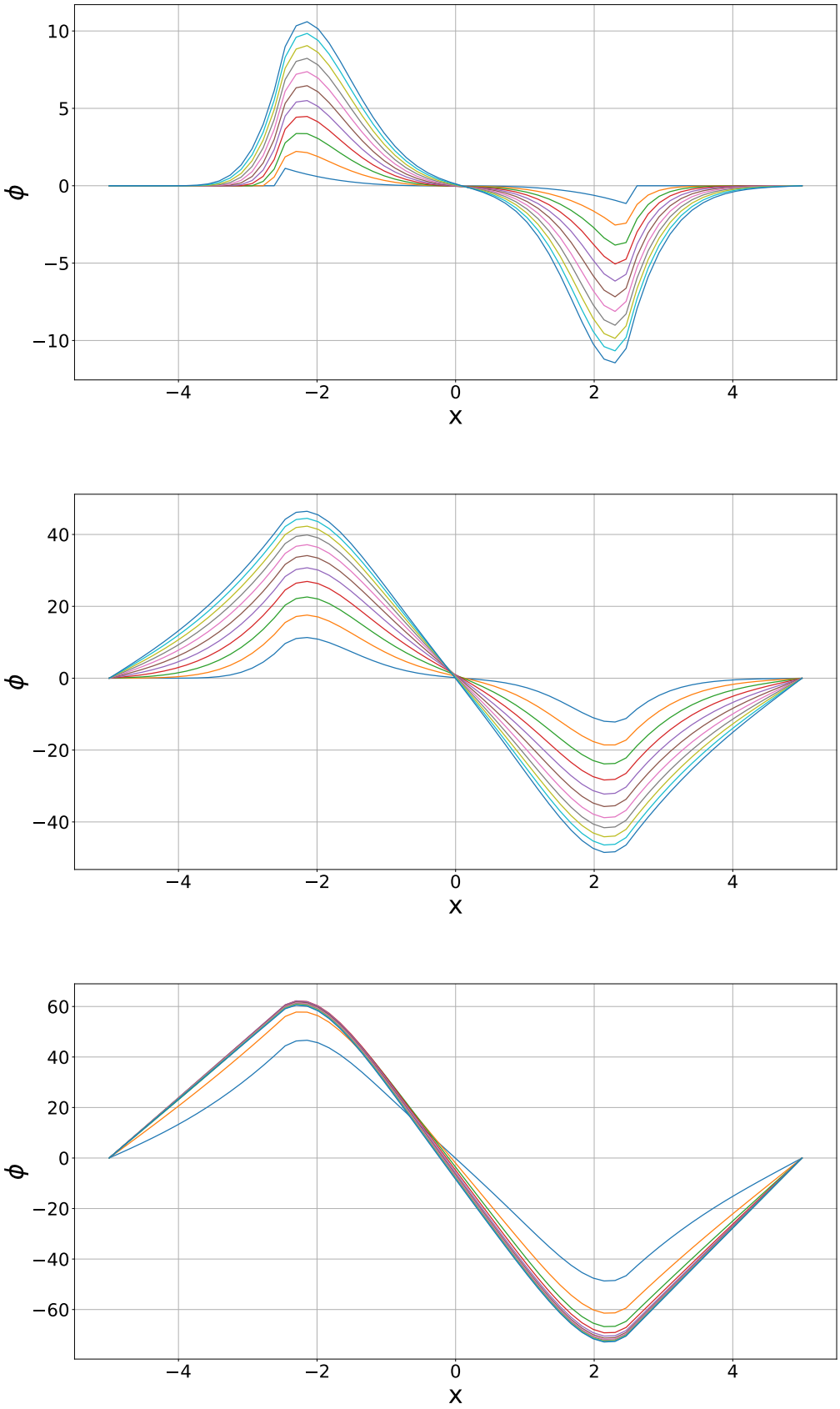

Figura 4.3: Potencial gravitacional obtenido para el perfil de densidad mostrado en figura 4.1 durante las primeras 10 iteraciones (primer panel), 100 iteraciones (segundo panel) y 1000 iteraciones (tercer panel).

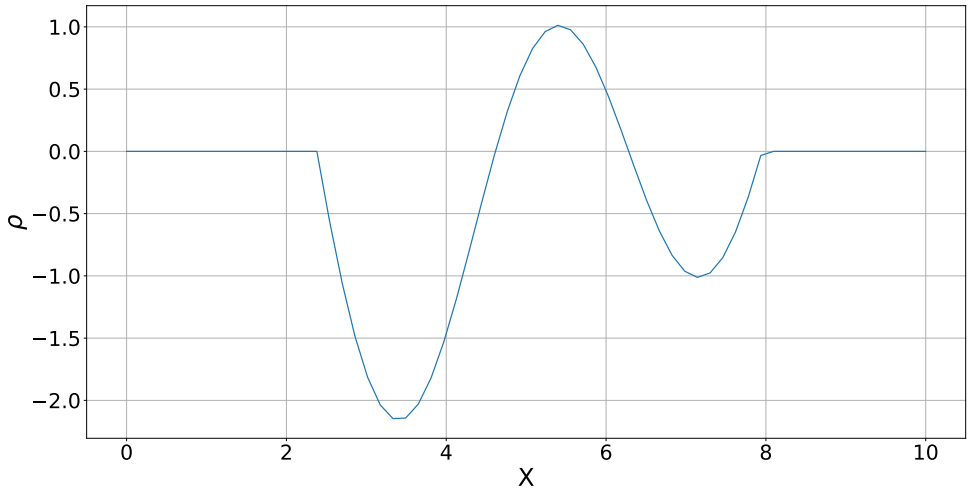

Figura 4.4: Perfil de densidad basado en una suma de funciones sinusoidales.

programa se esté ejecutando. Gracias a esto podremos determinar los resultados y mostrar problemas en la programación o lógica del programa y analizar más detalladamente nuestras observaciones.

4.1.1. Cálculo del error relativo

Antes de pasar al análisis de los resultados es importante tener en cuenta como están estructuradas las imágenes (por ejemplo la que se visualiza en la figura 4.3), que representa la evolución de la solución a través del avance en las iteraciones del proceso.

La figura 4.3 muestra el avance de la función ϕ en un rango de 5000 iteraciones, correspondientes al ciclo número 0, 10, 20, 30, $\cdots$, 100 (primer panel), 100, 200, $\cdots$, 1000 (segundo panel), 1000, 2000, $\cdots$, 5000 (tercer panel). Para estas gráficas el orden siempre es el mismo. En función al color, el orden es: azul marino, naranja, verde periquito, rojo, violeta, café, rosa, gris claro, verde pera, cían, y morado.

En el eje de las "X" siempre tendremos el espacio donde se distribuye el perfil de densidad y en el eje de las "Y" el valor del potencial ϕ calculado para esa iteración.

En esta sección consideraremos la misma metodología que la sección anteriores. Representaremos a la función de densidad como una combinación de funciones cosinusoidales, tales como se muestra en la figura 4.4, la cual viene dada por la ecuación:

$$f(x) = \begin{cases} 2\sin\left(\frac{3}{2}x\right) + \frac{6}{5}\sin(x) & \text{si} \quad x \in [2{,}5, 8] \\ 0 & \text{si} \quad x \notin [2{,}5, 8] \end{cases}$$

Usando el método de Gauss-Seidel y el mismo algoritmo para encontrar la solución, conseguimos el proceso de iteración dividido en gráficas (figuras 4.5 y 4.6) que convergen a la solución final mostrada en la figura 4.6, correspondiente al

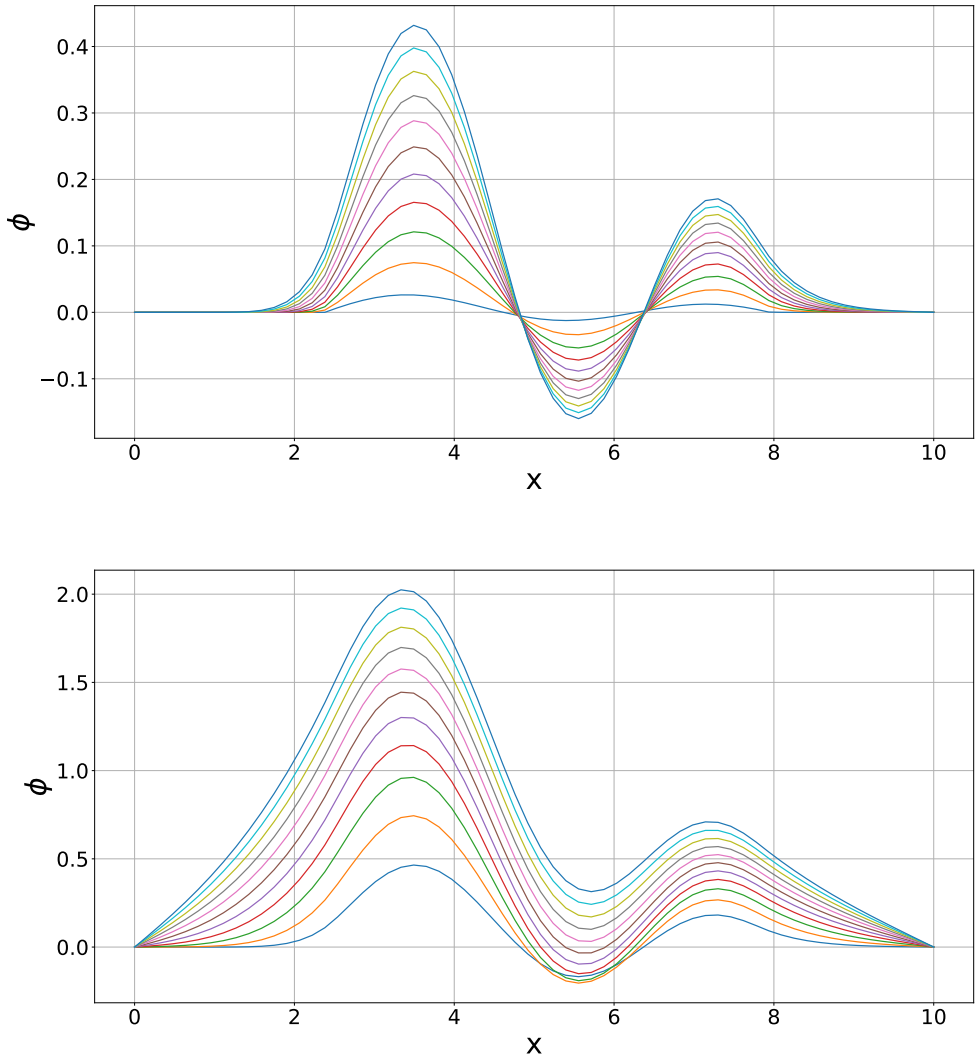

Figura 4.5: Evolución del potencial gravitacional obtenido integrando la ecuación de Poisson, obtenido para el perfil de densidad mostrado en figura 4.4 después de 10 (primer panel) o 100 iteraciones (segundo panel).

ciclo número 15000.

Ahora, queremos ver como evoluciona el error que aparece en el proceso recursivo. Para fines de esta tesis decidí tomar el error que se genera entre la iteración n-ésima y la siguiente. Para esto se puede tomar el vector ϕ^n y compararlo con los valores de ϕ^{n-1}. El error se define como

$$\text{N-ésimo error} = \int_{j=0}^{j=N} (\phi_j^n - \phi_j^{n-1}) dx \ ,$$

donde $dx = x_j - x_{j-1}$ es el tamaño de cada celda.
La figura 4.7 muestra el error como función de la iteración (o ciclo) n. Nótese que los ejes poseen una representación logarítmica. Como se puede notar, el error cae a valores extremadamente pequeños (del orden de 10^{-8}) sobre escalas de miles de iteraciones.

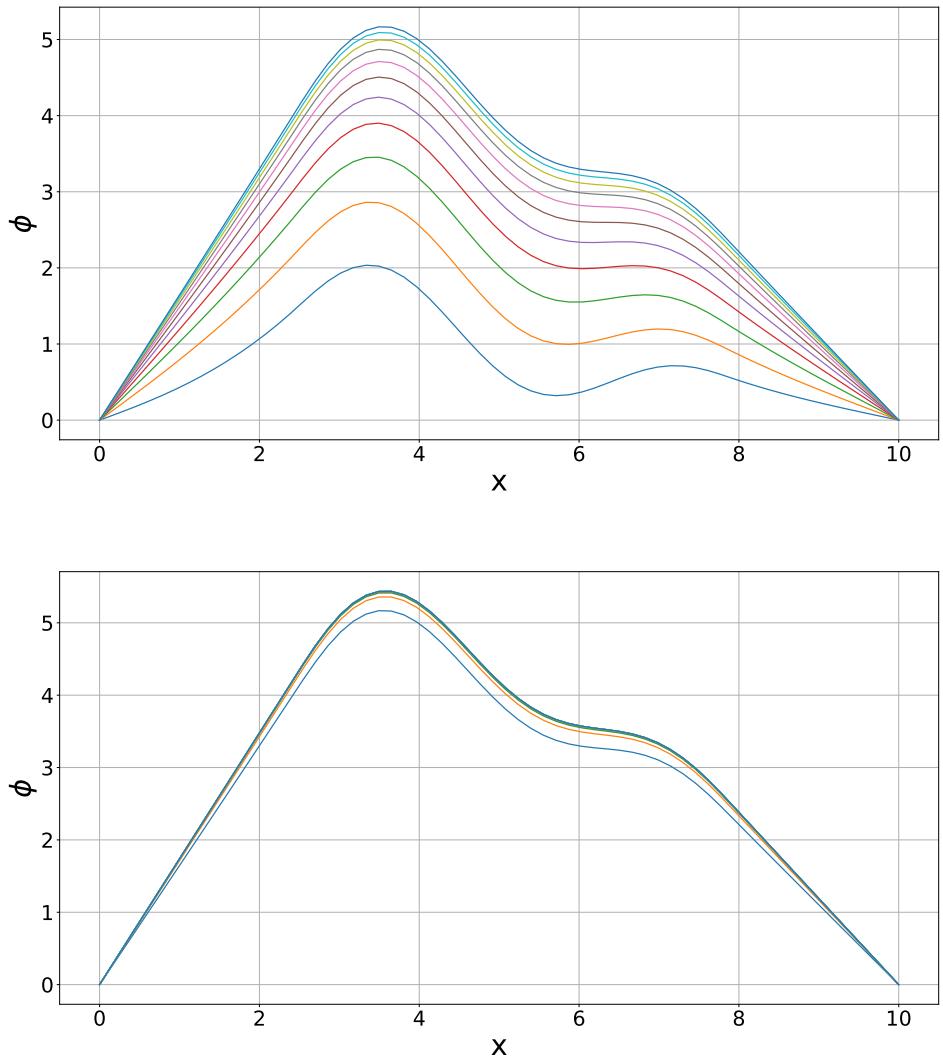

Figura 4.6: Evolución del potencial gravitacional obtenido integrando la ecuación de Poisson, obtenido para el perfil de densidad mostrado en figura 4.4 después de 1000 iteraciones (primer panel) y 5000 iteraciones (segundo panel).

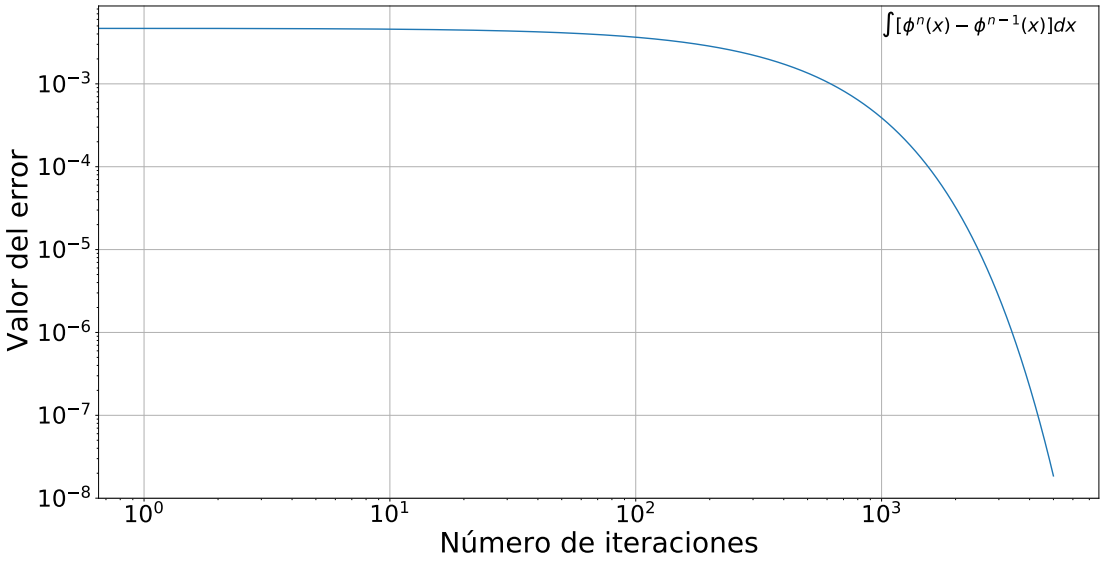

Figura 4.7: Gráfica del error calculado para el perfil de densidad mostrado en figura 4.4.

4.1.2. Convergencia del método para diferentes perfiles de densidad

A continuación presentaremos una serie de gráficas mostrando el perfil de densidad, el potencial obtenido como resultado del proceso iterativo, y el error relativo (como función del ciclo iterativo).

En la figura 4.8 (primer panel) se muestra un perfil de densidad definido como una distribución de tipo $\sin(2x)$. En la figura 4.8 (segundo panel), se muestra que la solución converge en la iteración 5000. El error se muestra en la figura 4.8 (tercer panel). Cabe destacar que los métodos usados para resolver la ecuación de Poisson toman en cuenta toda la distribución de masa (por ser el resultado de una ecuación elíptica), es decir para su solución también se consideran las partes de la distribución que tienen valor $\rho(x) = 0$.

En el caso de densidad constante ($\rho(x) = 1$), la ecuación de Poisson puede ser integrada fácilmente de manera analítica. En este caso, $\phi(x) = \frac{1}{2}(x-5)^5$. La comparación entre la solución analítica y la solución numérica (obtenida como resultado del proceso iterativo) se muestra en la figura 4.9. Como se puede notar en esta figura tenemos las dos soluciones superpuestas, la de color azul representa la solución analítica y la de color naranja es la solución calculada mediante el método de Gauss-Seidel. Estas funciones son muy parecidas, tanto que no es sencillo diferenciarlas a simple vista.

4.2. Implementación del método SOR

En las secciones anteriores se ha integrado la ecuación de Poisson por medio del métodos de Gauss-Seidel. Ahora veamos la solución de la ecuación de Poisson

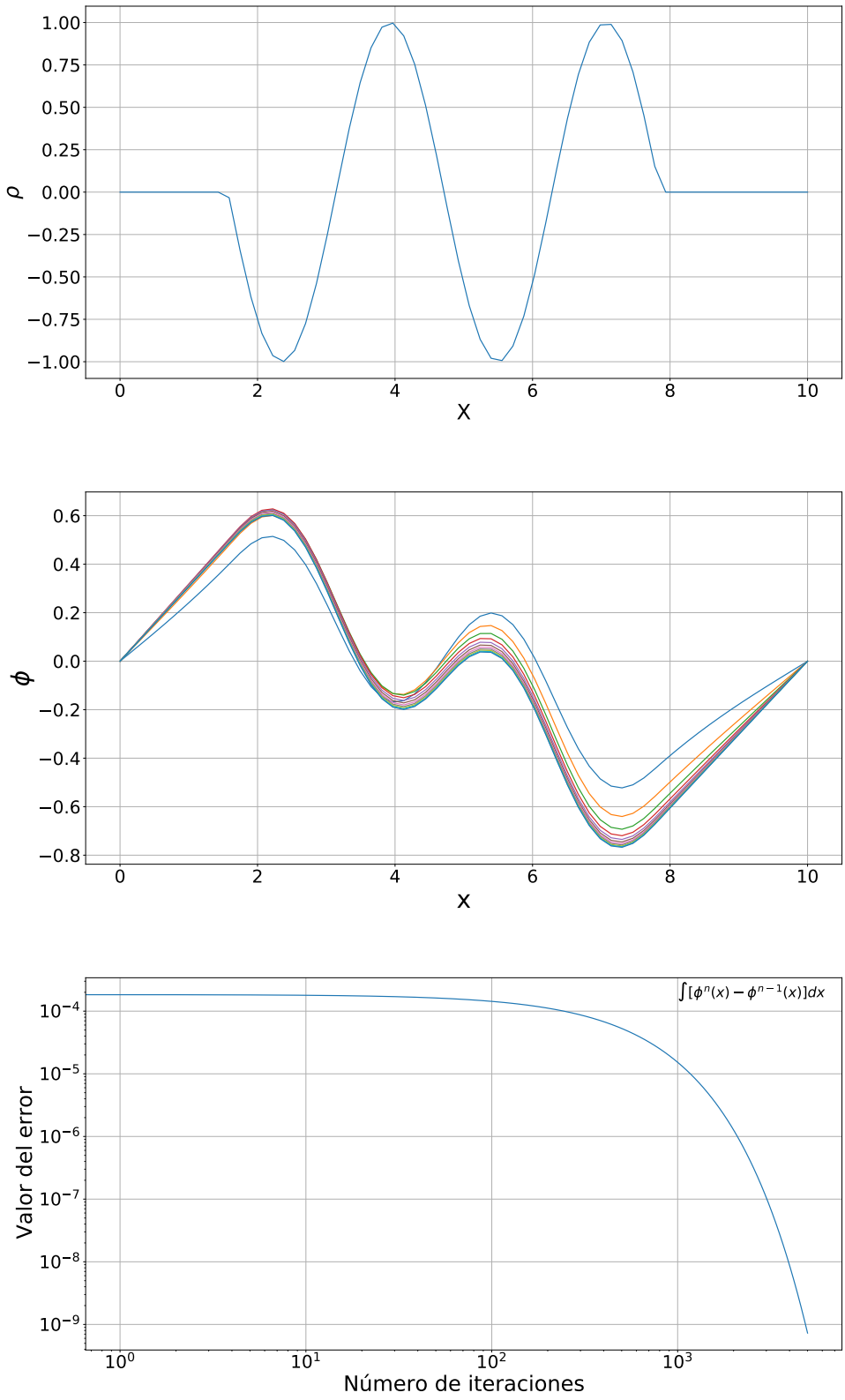

Figura 4.8: Solución de la ecuación de Poisson para un perfil de densidad $\rho(x) = \sin(2x)$ (primer panel). Avance de la función para las primeras 1000 iteraciones (segundo panel). Gráfica del error (tercer panel)

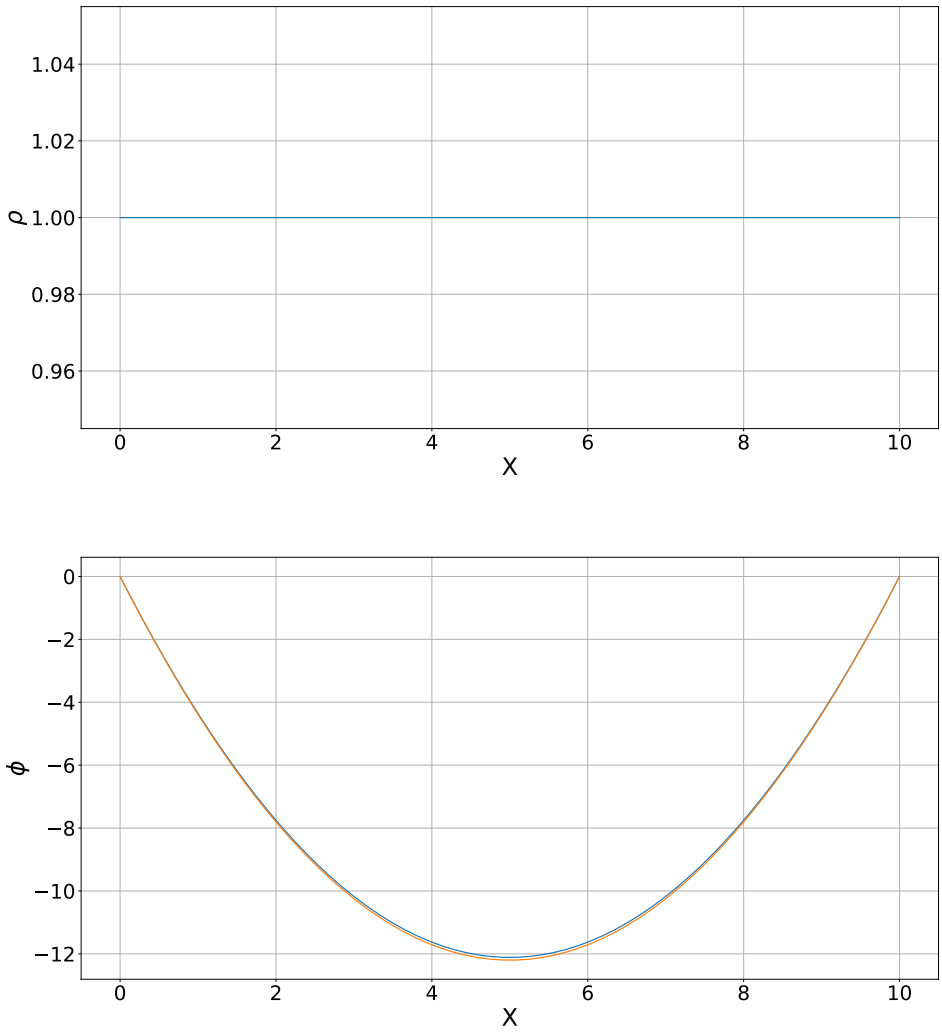

Figura 4.9: Perfil de densidad constante (primer panel). Comparativa de las soluciones calculada (curva naranja) y analítica (curva azul, segundo panel).

a través del método SOR. De la misma manera que en la sección anterior, la ecuación se integra hasta encontrar el vector ϕ_2 determinado en cualquier ciclo. En el método SOR, se define una variable γ para así determinar una nueva solución por medio de la siguiente ecuación:

$$\phi^{\text{Actual}} = \gamma\phi_i^2 + (1-\gamma)\phi_i^1 \ .$$

Un ejemplo de integración de la ecuación de Poisson por medio del método SOR se muestra en la figura 4.10. En este caso, el perfil de densidad es la función $\cos(x)$ con $x \in [2, 8]$. Dadas estas condiciones el programa determina los siguientes resultados que se visualizan en la figura 4.10. En este caso se ha considerado un parámetro $\gamma = 3/2$ constante.

Ahora, a simple vista este proceso parece muy similar a los anteriores utilizado. Para poder medir el coste computacional y tiempo efectuado en el programa, aplicaremos una condición de salida al programa para que una vez alcanzado un cierto error, determinado por el programador, mande un mensaje de conclusión y detenga el programa, como se vera en la sección 4.2.3.

4.2.1. Comparativa entre distintos valores para la constante γ. Teórico y simulado.

Una vez teniendo en cuenta la implementación del método SOR con Overelaxation, tomamos en cuenta también el valor γ que optimiza la solución del problema. Para esta comparativa utilizamos la función constante $\rho(x) = 1$.

Cansiderando ya esta función de prueba, encontramos la solución de la ecuación de Poisson. Se hicieron 200 simulaciones cambiando los valores de γ de 0 a 1 espaciados uniformemente. Estas simulaciones se calcularon considerando una malla compuesta por 2^{10} celdas. La figura 4.11 representa la comparativa entre el valor de la constante γ y el número de iteraciones necesarias para llegar a un error relativo entre la iteración ϕ_{n+1} y ϕ_n menor que 10^{-8} que es el valor que entrega la integral $\int[\phi^{n+1}(x) - \phi^n(x)]dx$, es decir:

$$\int[\phi^{n+1}(x) - \phi^n(x)]dx < 10^{-8},$$

Entonces, dadas estas condiciones, repetimos esta misma operación variando el número de celdas en la malla computacional, utilizando la misma función de prueba (es decir, una densidad constante). En la figura 4.12, mostramos una comparativa entre el valor de gamma encontrado en las simulaciones numéricas, y el valor esperado para la gamma dado por la ecuación:

$$\gamma = \cos\left(\frac{\pi}{J}\right) + 1 \ ,$$

donde J es el número de celdas consideradas.

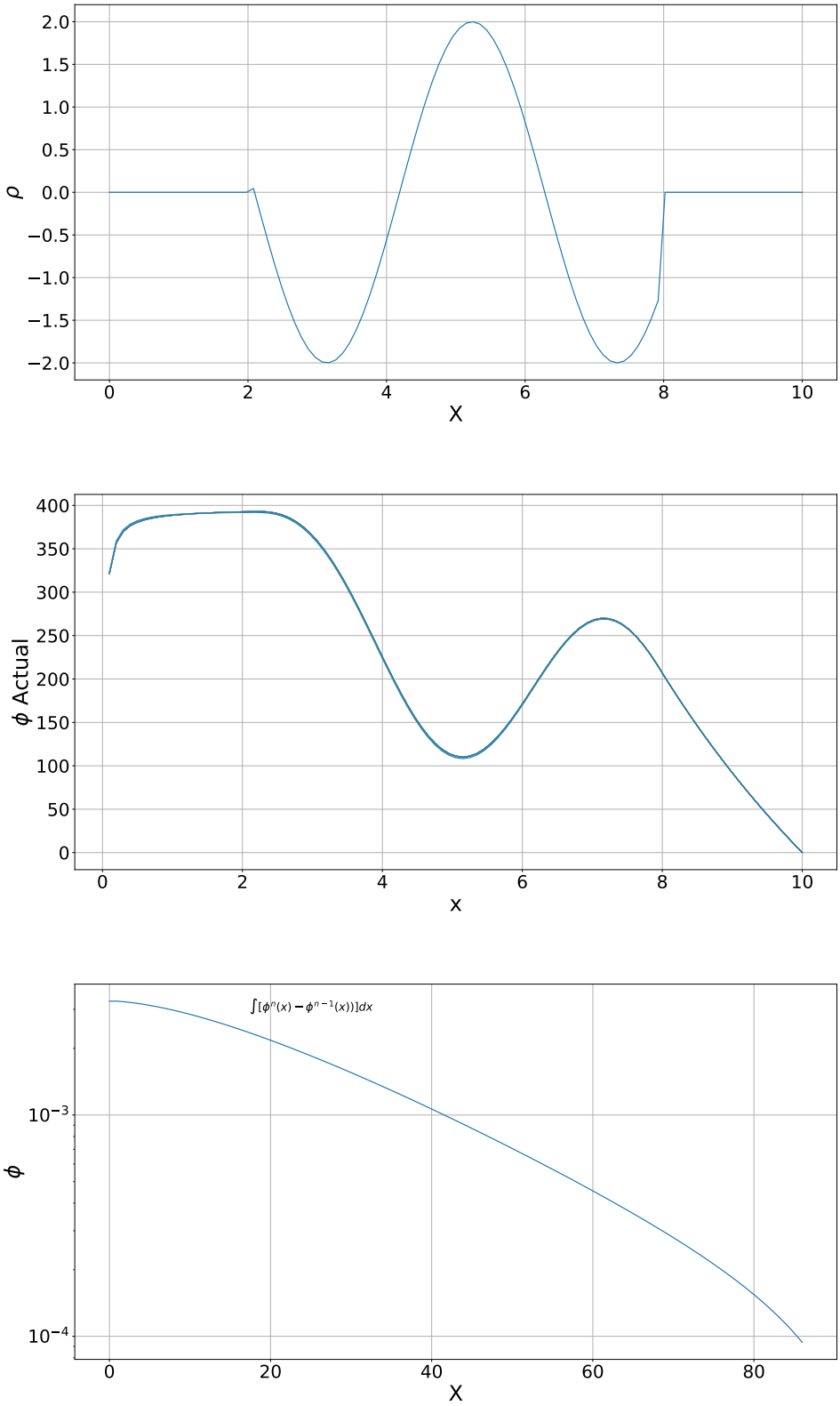

Figura 4.10: Perfil de densidad sinusoidal (primer panel). Últimas iteraciones del método SOR (segundo panel). Error relativo usando el método SOR (tercer panel).

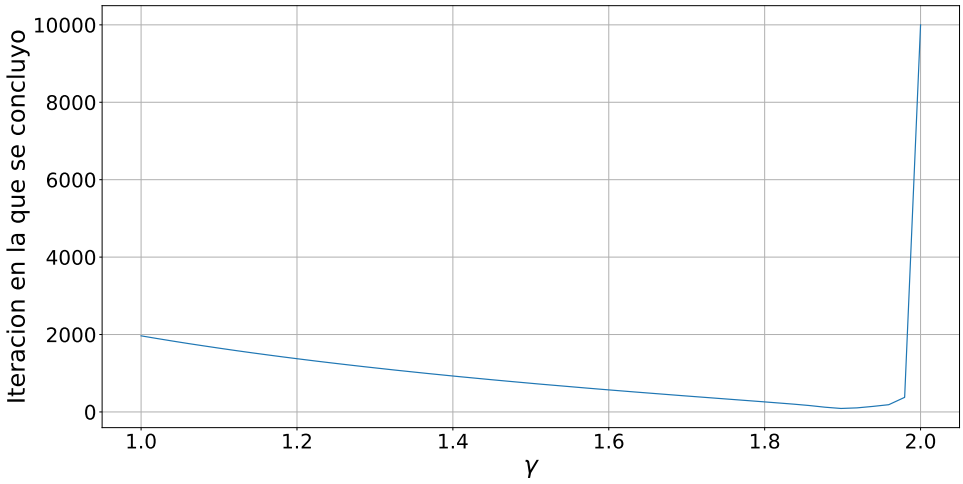

Figura 4.11: Número de iteraciones necesarias para obtener convergencia del método, como función del parámetro γ del método SOR.

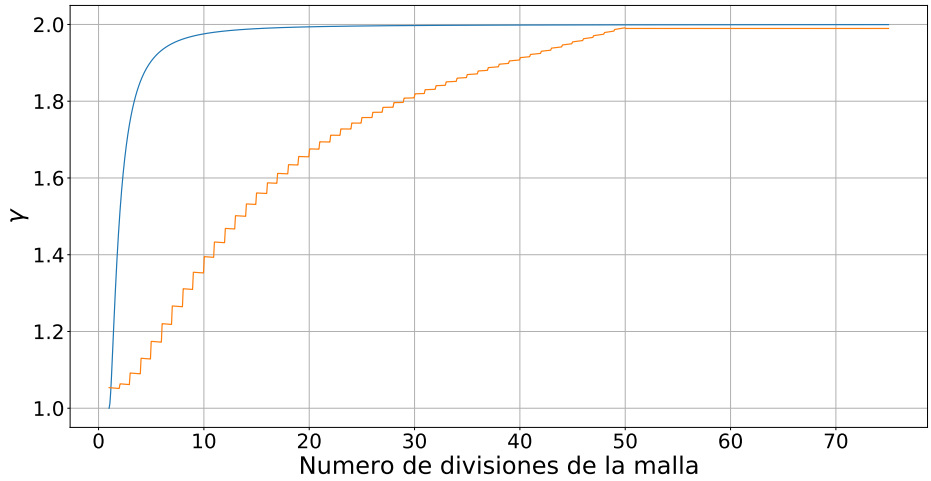

Figura 4.12: Valor óptimo (es decir, que minimiza el número de iteraciones) del parámetro γ como función del número de celdas de la malla computacional. En naranja se muestran los valores de γ obtenidos en las simulaciones. En azul se muestran los valores esperados.

En la figura 4.12, la curva de color azul representa el valor de γ calculada con esta función analítica, mientras que la curva naranja muestra los valores de γ "óptimos" encontrados por medio de las simulaciones numéricas. Como se puede ver en la figura, la formula analítica describe bien el comportamiento de γ para mallas con mas de ~ 50 celdas.

4.3. Método "Multigrid"

La idea principal del método "multigrid" es acelerar la convergencia de un método iterativo básico (conocido como relajación, que generalmente reduce el error de longitud de onda corta) mediante la solución de la ecuación de Poisson sobre malla con un número distinto de celdas. Las mallas mas finas, más baratas de resolver, permiten reducir los errores sobre longitudes de onda mayores, acelerando la convergencia del proceso iterativo. El método entonces consiste en encontrar la solución de la ecuación de Poisson en distintas mallas, interpolando las soluciones entre los mallas. La aplicación típica de multigrid es la solución numérica de ecuaciones diferenciales parciales elípticas en dos o más dimensiones.
En esta sección mostraré la aplicación del método multigrid a una malla de una dimensión, con una distribución de densidad dada por una combinación lineal de senos y cosenos (figura 4.13) . Empezamos considerando una implementación sencilla del método, es decir, tomamos una malla fina y una malla gruesa. En cada malla resolvemos una vez la ecuación y el resultado lo interpolamos a la siguiente malla, gruesa o fina, para trabajar con ella.

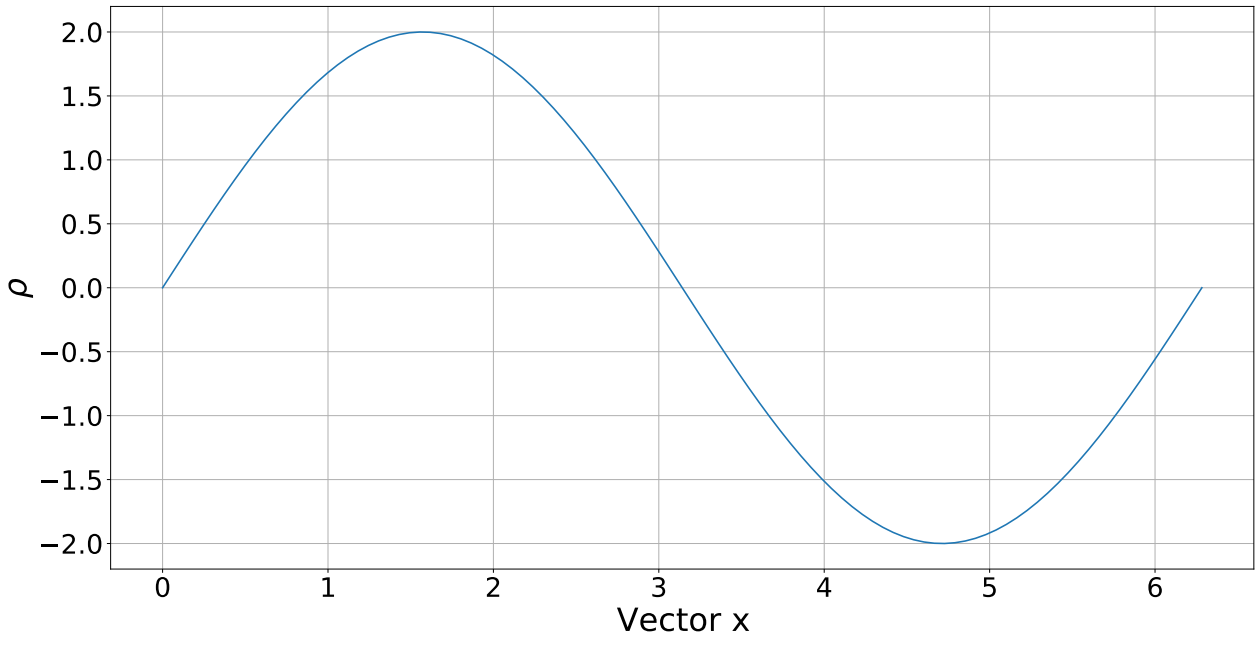

Figura 4.13: Distribución de densidad usada el el método multigrid.

La figura 4.14 muestra el perfil de densidad usado, el potencial gravitacionales durante 100 iteraciones, y la comparación entre la solución analítica y la solución obtenida numéricamente.

Esta figura analiza la solución a la ecuación en tres distintos niveles de mallas, que va ensanchándose cada vez más para llegar a un límite y volver a hacerse fina, hasta volver a su grosor original. El efecto en la evolución de cada ciclo puede verse en la figura 4.15 para una función constante junto a la gráfica del error correspondiente a este método.

Las figuras 4.16 y 4.17 presentan los resultados obtenidos usando el método multigrid con ciclos de tipo "W".

4.3.1. Comparativa de soluciones

Para determinar la eficiencia de cada método usado para integrar la ecuación de Poisson, consideramos la eficiencia de cada uno de los diferentes tipos de soluciones o marcos de soluciones. Podemos determinar su eficiencia usando directamente la gráfica de los errores, y en particular el número de ciclos que se necesitan para llegar a una solución estable (es decir, con un error inferior a un cierto valor).
Para esta sección tomaremos la ya clásica función constante $f : [0, 10] \to \mathbf{R}$ con $\rho(x) = 1$. Tomando de prueba dicha función, en los códigos que calculan la solución, agregamos una rutina que con base en el error relativo tenga una bandera de alto tal que si el error relativo de n-ésima iteración ($\int \phi^n - \phi^{n-1}$) es menor que una diez milésima parte del primer error relativo ($\int \phi^1 - \phi^0$), entonces el código se detiene y registra el número de iteraciones efectuadas. Los resultados se muestra en la tabla 4.1.
En estas tablas se realiza la comparación de los métodos, en el título de la tabla se indican el número de divisiones de la malla, que se tomaron como potencias de 2 (el número de celdas es $= 2^N$). En la tabla 4.1 tenemos varias columnas, una con el método usado para integrar la ecuación de Possion, y en las siguientes columnas están presentados las mallas utilizadas en función al número de celdas que utilizamos, en las celdas de la tabla 4.1 ponemos el valor que corresponde a la iteración donde el error $\int \phi^n - \phi^{n-1}$ llegó al valor límite mínimo, del cual hablamos en el párrafo pasado, para el método correspondiente. Hicimos este procedimiento para cuatro distintas divisiones de la malla.

Desde las tablas, podemos concluir que los métodos de tipo SOR son buenos para una malla gruesa en una dimensión, pero mientras más fina sea la malla mas el método multigrid tiene mejores resultados, en cuestión de velocidad. En el caso de una malla fina, con $2^{12} = 4048$ celdas, por ejemplo, el método multigrid es alrededor de 30 veces mas rápido que el método SOR.

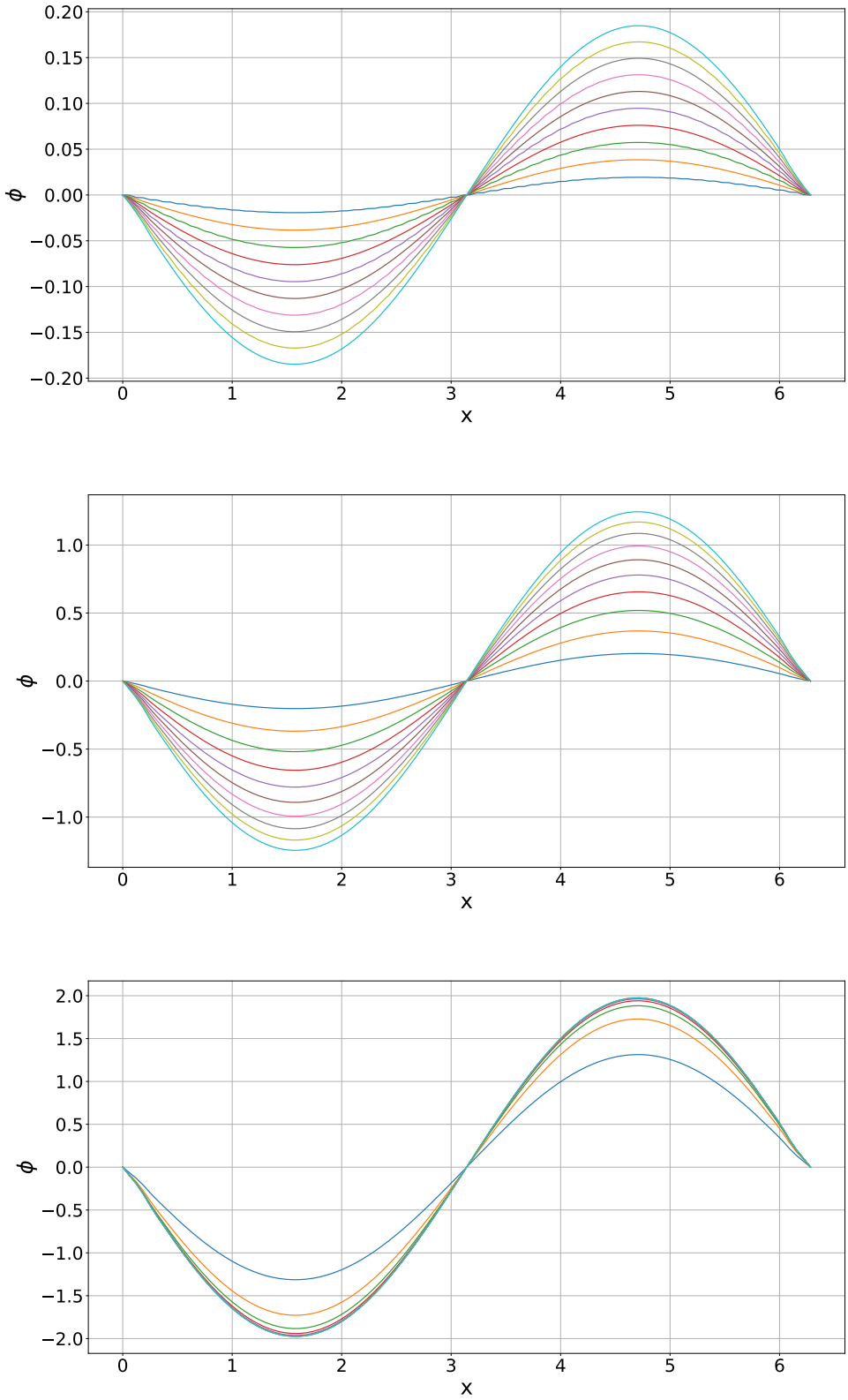

Figura 4.14: Evolución del potencial gravitacional obtenido integrando la ecuación de Poisson después de 10 ciclos (primer panel), 100 ciclos (segundo panel) y 500 ciclos (tercer panel), para el perfil de densidad mostrado en la figura 4.13.

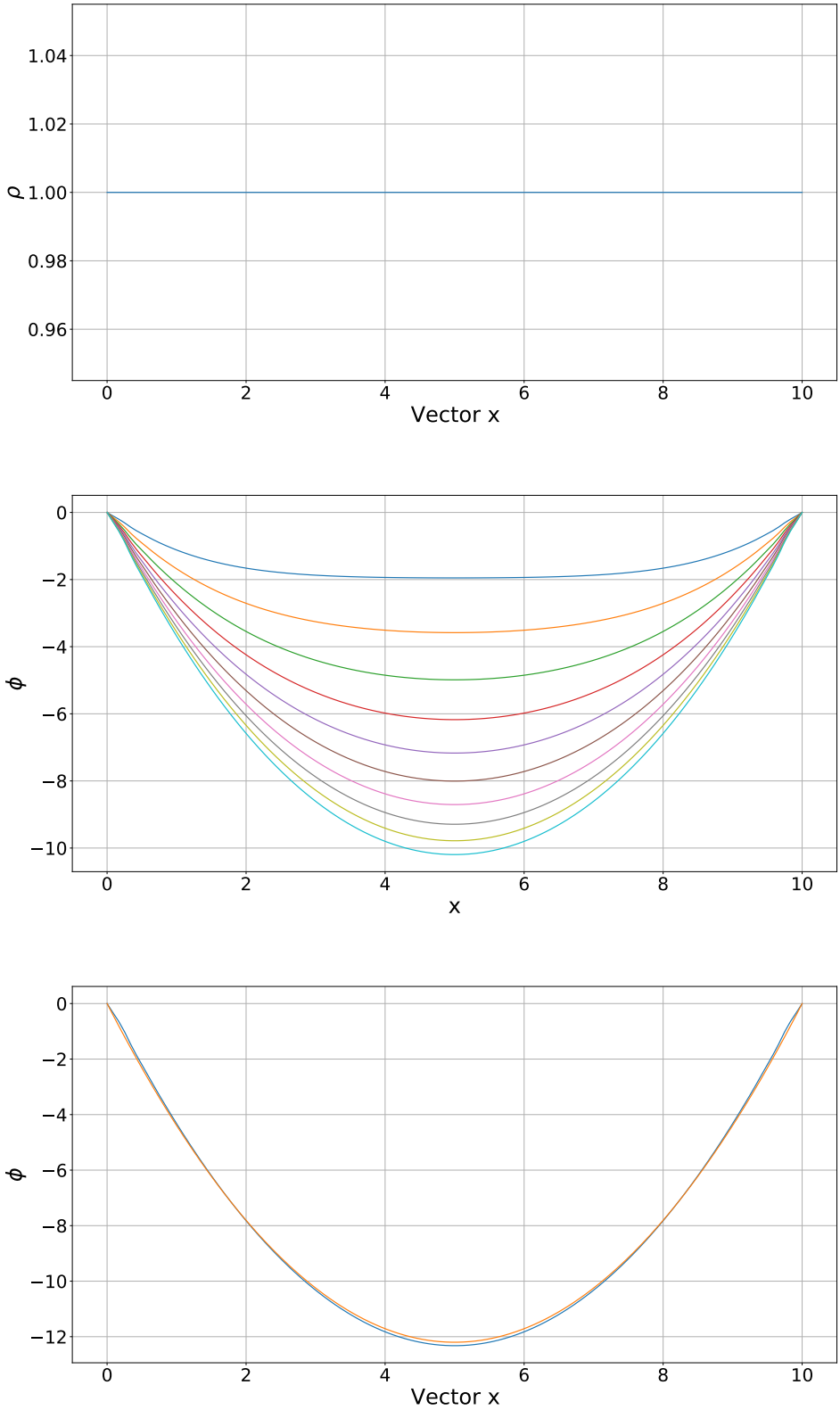

Figura 4.15: Función de prueba para la comparativa de soluciones en el método multigrid. Perfil de densidad constante (primer panel). Potencial gravitacional obtenido después de 100 iteraciones (segundo panel). Comparativa entre la solución numérica y la solución analítica (tercer panel).

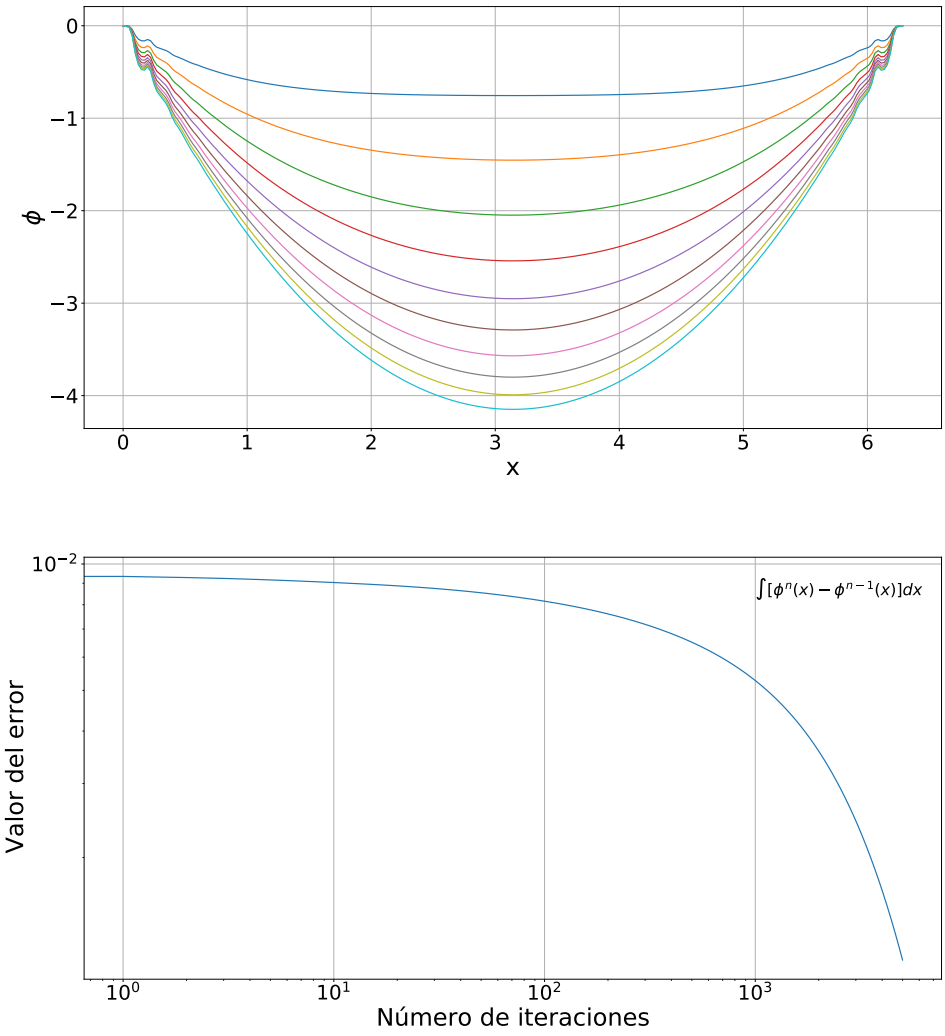

Figura 4.16: Potencial gravitacional obtenido después de 100 iteraciones para un perfil de densidad constante (segundo panel). Evolución del error como función del número de iteraciones (segundo panel) para el método multigrid tipo V.

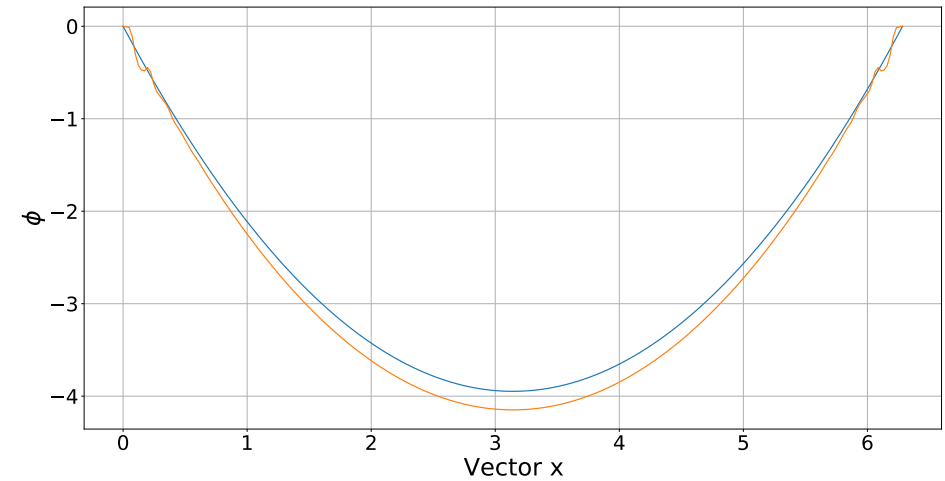

Figura 4.17: Comparación entre la solución analítica y la solución numérica para el método multigrid tipo W.

Método	32 celdas	256 celdas	4096 celdas	32768 celdas
Overelaxation	55	785	15485	120692
SOR con $\gamma = 1{,}2$	45	1255	11255	101046
SOR con $\gamma = 1{,}8$	33	484	4790	90356
Multigrid simple	125	3500	2780	8921
Multigrid tipo 'V'	541	5288	4278	6318
Multigrid tipo 'W'	440	5579	3697	4078

Cuadro 4.1: Tabla comparativa para el método usado y número de iteraciones para cada caso

4.3.2. Malla en 2D

Ahora bien dados los criterios de solución anteriores podemos introducir las mallas en dos dimensiones, es decir en un plano, en el cual definiremos una distribución de masa especifica para obtener la solución a la ecuación de Poisson. Empezaremos con una distribución que obedezca a una función cosenoidal, cuya distribución en dos dimensiones viene dada por.

$$\rho = \cos(x^2 + y^2)$$

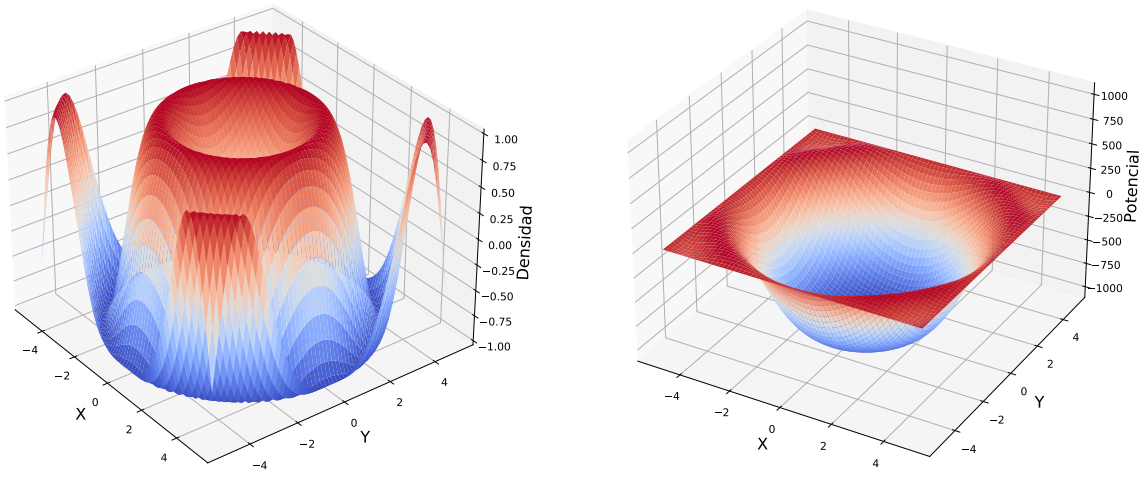

Figura 4.18: Distribución de densidad de una función cosinusoidale (panel izquierdo) y potencial gravitacional (panel derecho) para una malla en dos dimensiones.

Una vista más detallada de la solución se muestra a continuación en las figuras 4.19 y 4.18.

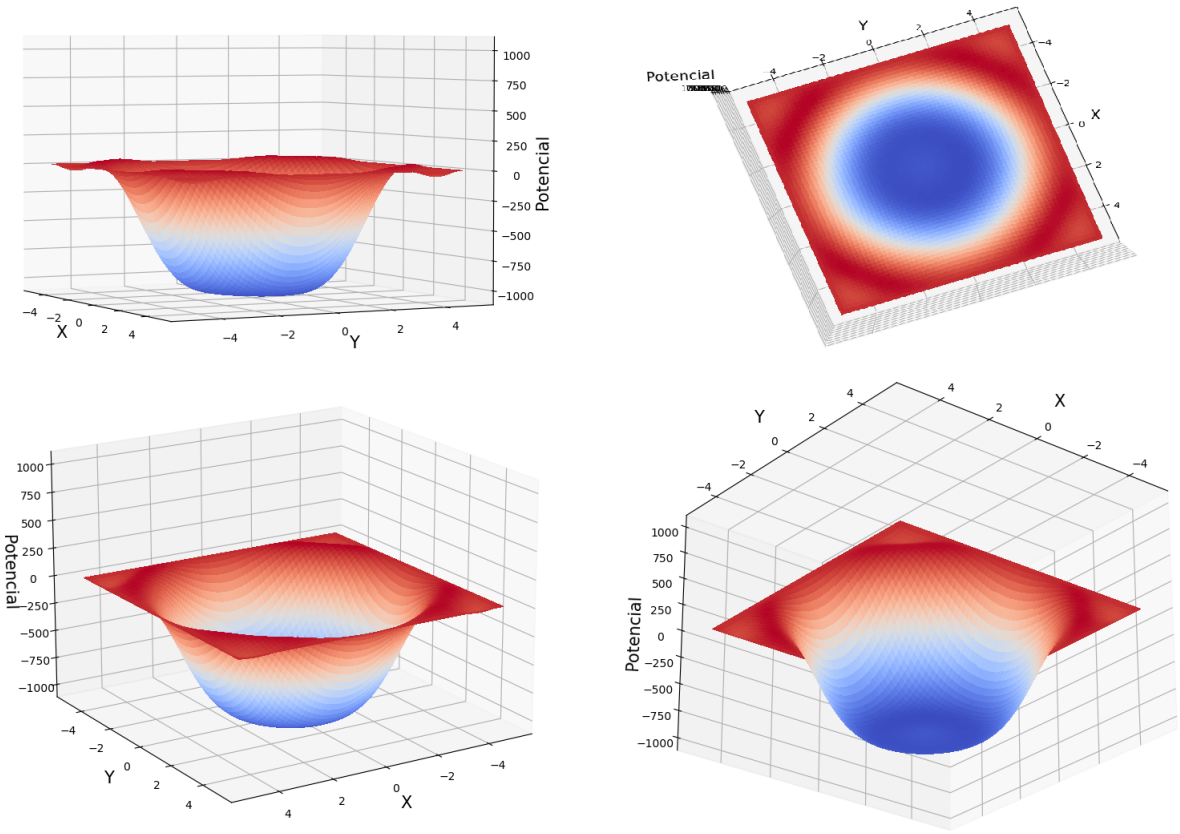

Figura 4.19: Distintas proyecciones del potencial gravitacional obtenido considerando una densidad sinusoidal, para una malla en dos dimensiones.

Por otro lado, tomando una densidad descrita por la función sinusoidal, es decir

$$\rho = \sin(x^2 + y^2) \ ,$$

y con un número de iteraciones igual a 500, encontrar la solución mostrada en la figura 4.20.

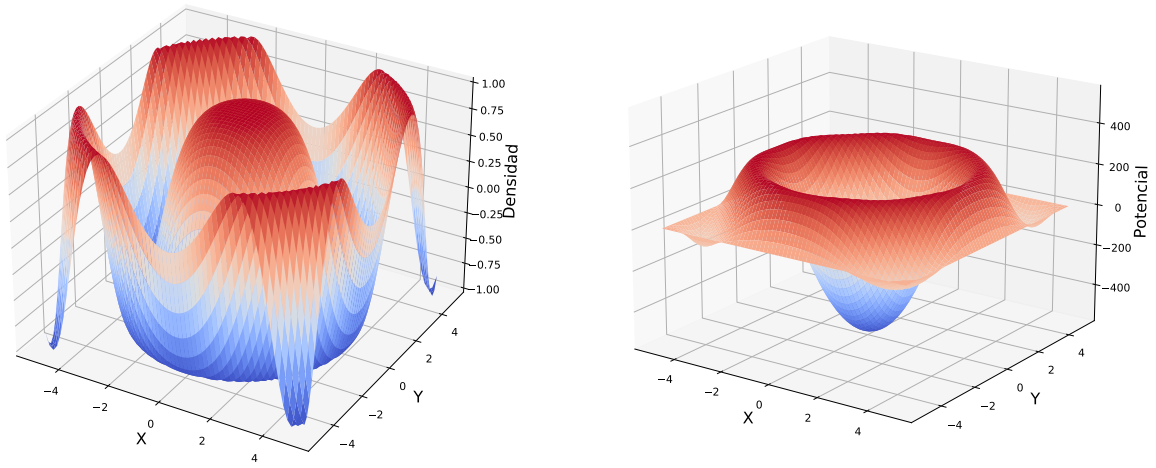

Figura 4.20: Distribución de densidad de una función sinusoidal (panel izquierdo) y potencial gravitacional (panel derecho) para una malla en dos dimensiones.

4.4. Inclusión de la auto-gravedad en el código hidrodinámico

Uno de los objetivos de esta tesis es acoplar el calculo de la auto-gravedad con un código hidrodinámico. En esta sección, describiré brevemente los resultados obtenidos. Considero un perfil sencillo de densidad:

$$\rho = \rho_0(1 - x^2 - y^2) \ , \tag{4.1}$$

donde $\rho_0 = 1$. Dada esta distribución de densidad, se puede calcular el potencial gravitacional analíticamente. En dos dimensiones, en coordenadas cartesianas, el potencial es:

$$\phi = -\frac{3}{16} + \frac{r^2}{4} - \frac{r^4}{16} \ , \tag{4.2}$$

donde $r = \sqrt{x^2 + y^2}$. Considerando la ecuación de equilibrio hidrostático,

$$\frac{\partial p}{\partial r} = -\rho \frac{\partial \phi}{\partial r} \ , \tag{4.3}$$

también podemos calcular la presión analíticamente. Tenemos

$$\frac{\partial p}{\partial r} = -\rho \frac{\partial \phi}{\partial r} = -\frac{r}{2} + \frac{3r^3}{4} - \frac{r^5}{4} \ . \tag{4.4}$$

Entonces, la presión es

$$p = p_0 - \frac{r^2}{4} + \frac{3r^4}{16} - \frac{r^6}{24} \ . \tag{4.5}$$

Tomamos $p_0 = 1$, y usamos esta formula para definir la presión inicial en el código hidrodinámico. Siendo que esta presión corresponde al equilibrio hidrostático, en principio el sistema debería mantenerse en equilibrio hidrostático a todos los tiempos.

La figura 4.21 muestra el potencial gravitacional calculado analíticamente (primer panel), el potencial gravitacional calculado en el código (segundo panel) y una figura comparativa entre las dos soluciones. Esta figura muestra que el método desarrollado para calcular el potencial gravitacional logra reproducir de manera precisa la solución analítica.

En la figura 4.22 se muestra la presión inicial (es decir en $t = 0s$). La presión se define por medio de la ecuación de equilibrio hidrostático. En principio, debería mantener por un tiempo indefinido este equilibrio.

Teniendo esta distribución para la presión en el tiempo inicial, ahora ejecutamos el código para verificar la evolución de la presión (y de los otros parámetros físicos). En la figura 4.23 se muestra la evolución temporal de la presión. Como podemos observar de esta última gráfica, podemos concluir que el sistema no conserva el equilibrio de las variables, es decir, al dejar pasar el tiempo en el código las cantidades del sistema empiezan a desviarse del equilibrio. También se puede notar que este efecto se manifiesta inicialmente cerca de la frontera de la malla computacional y se propaga después hacia la parte interna de la malla. No obstante que la parte interna tiene gradientes de presiones mayores, estos no producen una significativa perturbación desde el equilibrio (sobre las escalas de tiempo estudiadas). Por otro lado, las condiciones de frontera sencillas que hemos implementado (en las que simplemente se copia el valor afuera de la malla computacional igual al primer valor adentro de la misma) no son suficientemente precisas como para mantener el equilibrio hidrostático. Condiciones mas complejas en la frontera se implementarán en el futuro.

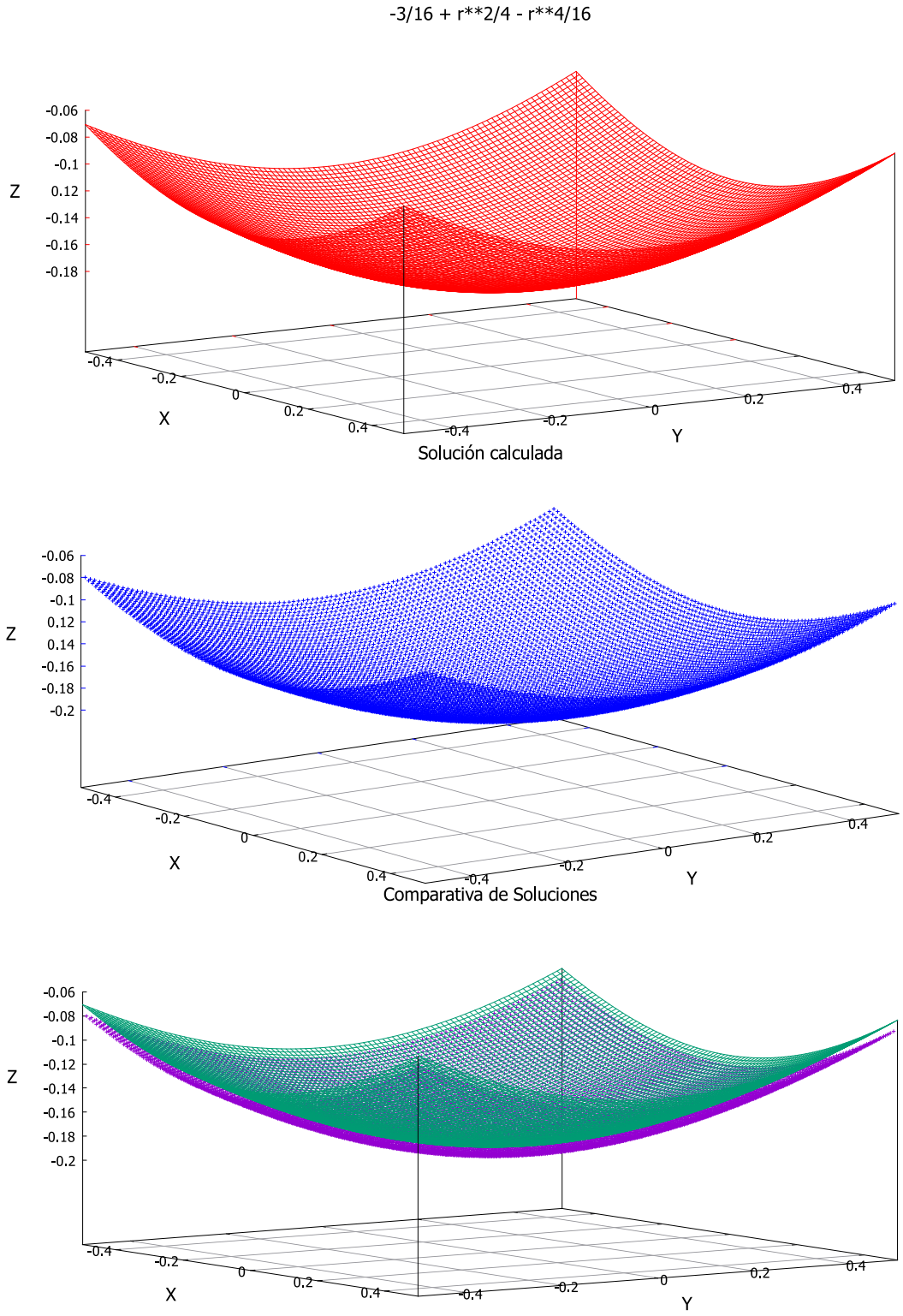

Figura 4.21: De arriba a abajo: potencial gravitacional obtenido analíticamente considerando el perfil de densidad $\rho = (1 - x^2 - y^2)$ (panel de arriba), calculado en el código hidrodinámico, y comparación entre las dos soluciones.

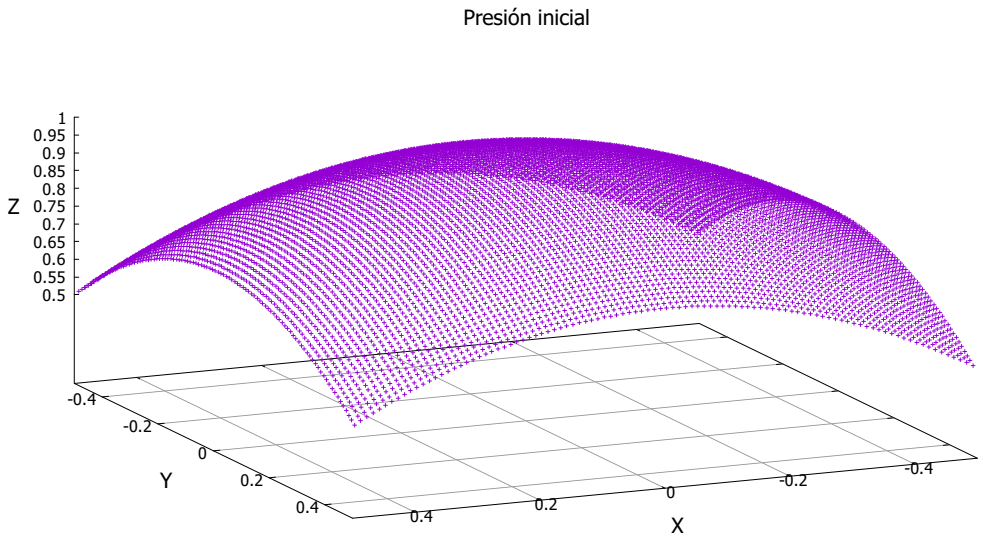

Figura 4.22: Presión inicial, determinada considerando equilibrio hidrostático.

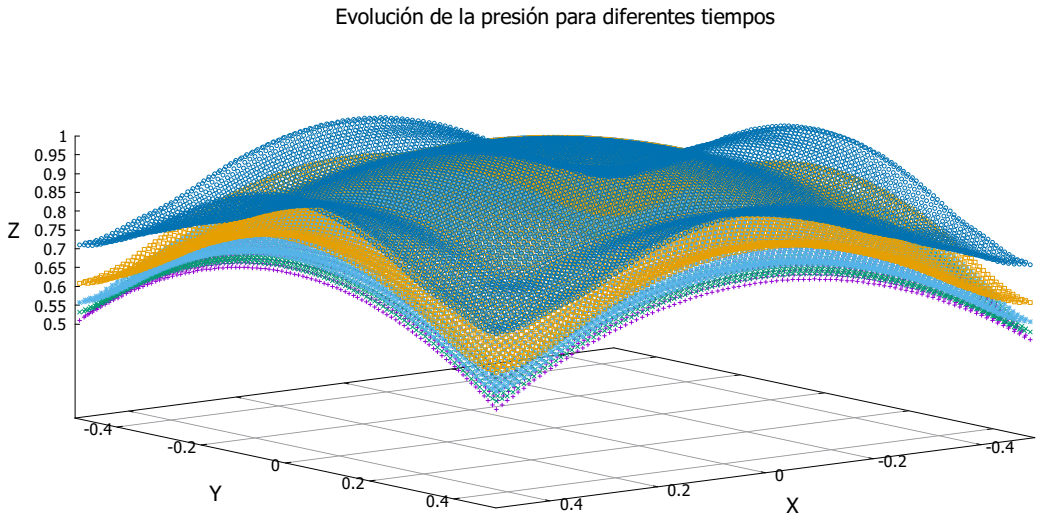

Figura 4.23: Evolución de la presión para $t = 0$, $t = 0{,}01$, $t = 0{,}05$, $t = 0{,}075$, $t = 0{,}1$.

5. Conclusiones

En esta tesis desarrollé un código para resolver la ecuación de Poisson. Implementé varios métodos tales como el método de Jacobi, el de Gauss-Siedel, el SOR y el método multigrid. Además, acoplé este código a un código hidrodinámico. Los resultados principales de esta tesis son los siguientes:

1. Para la solución a la ecuación de Poisson, es muy importante tener en cuenta que, al momento de la discretización, el tamaño de las celdas δs tiene que ser lo suficientemente pequeño para que la solución converja a una solución correcta. Tal es el caso del potencial gravitacional ϕ en el método de Gauss-Seidel, definido como:

$$\phi[i] = \frac{1}{2}\left(\phi[i-1] + \phi[i+1] - \rho[i]\Delta x^2\right) \ .$$

2. Los cambios en las evoluciones del potencial efectivo ϕ_n en función de la iteración llegan a ser muy acelerados para $n's$ pequeños (al rededor de 10^1 o 10^2), ya que su error disminuye rápidamente. Pero, para n ya más grandes (para decenas de miles de repeticiones), el error decrece cada vez más lentamente.

3. Para el método de SOR, el valor óptimo del parámetro γ es muy próximo al valor teórico. Las diferencias muestran que la solución teórica es aproximada.

4. La eficiencia de los diferentes métodos para solucionar la ecuación de Poisson es función del número de celdas. Es decir, dependiendo que tan fina sea nuestra malla el método más adecuado para llegar a una solución cambia. Ejemplificando esto, para una malla de pocas celdas (al rededor de 32) el método de Gaus-Seidel es el más rápido. En cambio para mallas más finas ($\gtrsim$ 1000 partes) el método multigrid tipo «W» mostró ser el más rápido. Siendo que en simulaciones hidrodinámicas se usan típicamente millones de celdas, el método multigrid es varios de ordenes de magnitud mas rápido que los otros métodos.

5. Respecto al acoplamiento del código hidrodinámico con el "Poisson solver", cuando se empieza con equilibrio hidrostático y dejamos avanzar la simulación, las variables del sistema se mantienen cercanas a dicha estabilidad cuando el tiempo de simulación es corto. Cuando trabajamos con tiempos mayores se puede presentar un deslizamiento y llevar al sistema fuera de el estado estable. Esto puede deberse a los errores numéricos del programa en empleado y al tipo de discretización (principalmente de la frontera) que

usamos. Dejamos para trabajos futuros explorar otras condiciones de frontera que puedan mantener de manera mas precisa el equilibrio hidrostático. Dos soluciones posibles para este inconveniente son: colocar condiciones de frontera periódicas, o bien, rodear el sistema fisico con un "gas caliente", que tenga gradientes densidad muy suaves a causa de la gravedad, y que este gas casi homogeneo sea el que esté en contacto con las fronteras en nuestro análisis computacional.

6. Bibliografia

1. Numerical Methods in Astrophysics, P. B., Laughlin, M. R., Yorke. 2007, Taylor & Francis, 1st ed.

2. Numerical Recipes in Fortran 77, W. H., Teukolsky, S. A., Vetterling, W. T., et al. 1992, Cambridge: University Press, 2nd ed.

3. Classical Mechanics , H. Goldstein, Co. 1980, Addison-Wesley Publ, 2nd ed.

4. Mecánica Vol , L.D. Landau- E.M. Lifshitz, 1970, Ed. Reverté, 2nd ed.

5. Análisis numérico, R. L. Burden, J. D. Faires, 2010 Thompson Learning.

6. Numerical Recipes in C; The art of scientific computing, W. H.Press, B. P. Flannery, et al. Cambridge university Press, 2005, 1st ed.

7. Introduction to Programming with Fortran, Ian Chivers, Springer, 2014 1st ed.

8. Introducing Python: Modern Computing in Simple Packages, Bill Lubanovic, O'Reilly, 2019, 3th ed.

Apéndice

6.1. Códigos

A continuación presentamos los códigos desarrollados.

6.1.1. Método de Jacobi

```
"""@author: Pedro Eduardo Medina Gonzalez"""
import matplotlib.pyplot as plt
import numpy as np
import imageio
from os import remove
import os
from os.path import isfile, join

def Pol(Coef,x):
    v = 0
    for i in Coef:
        v += i*x**(Coef.index(i))
    return v

def IntArr(x,arr):
    sum =0
    if len(x) == len(arr):
        for i in range(len(x)-1):
            sum += (x[i+1]-x[i])*((arr[i+1]+arr[i])/2)
    else:
        print("Funcion no valida,
        no coincide las dimensiones de los arreglos")
    return sum

x = np.linspace(-5, 5, 102)
rho = []
phi1=np.zeros(len(x))
phi2 = np.zeros(len(x))
phi = np.zeros(len(x))

for i in x:
    if (i>=-2.5 and i<=2.5):
        rho.append( Pol([2.5**2,0,0,-1],i) )
```

```
    else:
        rho.append(0)

plt.figure()
plt.plot(x, rho)
plt.ylabel(r'$\rho$',fontsize=30)
plt.xlabel('X',fontsize=30)
plt.tick_params(labelsize = 25)
plt.grid()

Ran = 10000  #Numero de Iteraciones
NumofPic=50 #Especifica el Numofit
Numofit = int(Ran/NumofPic)
path = "C:/Ubicacion"
intrie = []

contenido = os.listdir(path)
archivos =
[nombre for nombre in contenido if
isfile(join(path,nombre))]
for c in archivos:
    remove(path+"/"+c)

plt.figure()
for s in range(Ran+1):
    phi1[0] = 0
    phi1[len(x)-1] =0

    for i in range(1,len(x)-1):
        phi2[i] = 0.5*(phi1[i-1]+phi1[i+1]-rho[i])
    if(s%int((Ran/10))==0 and s!=0):
        print("Avance de los calculos en un:
        "+str(100*s/Ran)+"%")
    if(s%Numofit==0):
        plt.plot(x, phi2)
        pname=path+"/Imagen-"+
        str(s).rjust(len(str(Ran)),'0')+".png"
        plt.tick_params(labelsize = 30)
        plt.title('Recursion Numero:'+str(s))
        plt.ylabel(r'$\phi$')
```

```
            plt.xlabel('X')
            plt.grid()
            plt.savefig(pname)
            #plt.clf()
        phi1=phi2
contenido = os.listdir(path)
filenames = [nombre for nombre in contenido
if isfile(join(path,nombre))]

images = []
for filename in filenames:
    images.append(imageio.imread(path+"/"+filename))
imageio.mimsave(path+'/movie.gif', images)

plt.figure()
plt.plot(x, phi2)
plt.tick_params(labelsize = 25)
plt.ylabel(r'$\phi$',fontsize=30)
plt.xlabel('X',fontsize=30)
plt.grid()
```

6.1.2. Método de Gauss-Siedel

```
"""@author: Pedro Eduardo Medina Gonzalez"""
import matplotlib.pyplot as plt
import numpy as np
import math

plt.close("all")
def Pol(Coef,x):
    v = 0
    for i in Coef:
        v += i*x**(Coef.index(i))
    return v

def IntArr(x,arr):
    sum =0.0000
    if len(x) == len(arr):
        for i in range(len(x)-1):
            sum += (x[i+1]-x[i])*((arr[i+1]+arr[i])/2)
    else:
```

```
        print("Funcion no valida,
        no coincide las dimensiones de los arreglos")
    return sum
def Fseno(Arr,x):
    return Arr[0]*math.sin(Arr[1]*x + Arr[2])
def Fcoseno(Arr,x):
    return Arr[0]*math.cos(Arr[1]*x + Arr[2])
def Expon(A,arg):
    return A*math.exp(arg)



Inicio = 0
Final= 10
Numdedivisiones=2**6
x = np.linspace(Inicio, Final, Numdedivisiones)
rho = []
phi1=np.zeros(len(x))
phi2 = np.zeros(len(x))
dif = np.zeros(len(x))

minVal = math.pi/2
maxVal = 5*math.pi/2
for i in x:
    if (i>=minVal and i<= maxVal):
        rho.append(Fseno([1,2,0],i))
    else:
        rho.append(0)

Ran = [10,100,1000,5000]
NumofPic=10
path = "C:\Ubicacion"
intrie = []

plt.figure()
plt.plot(x, rho)
plt.tick_params(labelsize = 25)
plt.ylabel(r'$\rho$',fontsize=30)
plt.xlabel('X',fontsize=30)
plt.grid()
```

```
dx = (Final-Inicio)/Numdedivisiones
error = 1e-4
for m in Ran:
    y=[]
    Numofit = int(m/NumofPic)

    print("\n\n\n\n\n\n\n\n
    Calculo para la iteracion de:"+str(m))
    plt.figure()
    for s in range(m+1):
        phi1[0] = 0
        phi1[len(x)-1] =0
        for i in range(1,len(x)-1):
            phi2[i] =
            0.5*(phi1[i-1]+phi1[i+1]-rho[i]*(dx**2))
        if(s%int((m/10))==0 and s!=0):
            print("Avance de los calculos en un:
            "+str(100*s/m)+"%")
        if(s%Numofit==0):
            plt.plot(x, phi2)
            plt.ylabel('$\phi$',fontsize=35)
            plt.xlabel('x',fontsize=35)
            plt.tick_params(labelsize = 25)
            plt.grid()
        phi1=phi2
        (y.append(abs(IntArr(x,phi2))))
        if(len(y)==0)
        else(y.append(abs(IntArr(x,phi2))))

z = []
print("\n\n\n\n\n\n\nCalculo del error:")
for i in y:
    if (y.index(i))%int((len(y)/10))==0:
        print("Avance de los calculos del error:
        "+str(int(100*y.index(i)/len(y)+1))+"%")
    z.append(abs(i-y[y.index(i)-1])) if(y.index(i)!=0)
    else z.append(abs(y[1]-y[0]))

plt.figure()
```

```
plt.plot(x, phi2,label='Solucion calculada')
plt.tick_params(labelsize = 25)
plt.ylabel('$\phi$',fontsize=20)
plt.xlabel('x',fontsize=20)

solucion =[]
for a in x:
    solucion.append((Pol([0,-10,1],a))*12.20/25)

plt.plot(x, solucion,label='Solucion Analitica')
plt.tick_params(labelsize = 25)
plt.ylabel(r'$\phi$',fontsize=30)
plt.xlabel('X',fontsize=30)
plt.grid()



plt.figure()
plt.plot(z)
plt.annotate('$\int [\phi^n(x) -
\phi^{n-1}(x)]dx$',(len(z)/5,max(z)*0.9)
,fontsize=20)
plt.tick_params(labelsize = 25)
plt.ylabel('Valor del error',fontsize=30)
plt.xlabel('Numero de iteracion',fontsize=30)
plt.xscale("log")
plt.yscale("log")
plt.grid()
r = []
```

6.1.3. Método de sobre-relajación sucesiva - parametro γ fijo

```
"""@author: Pedro Eduardo Medina Gonzalez"""
import matplotlib.pyplot as plt
import numpy as np
import math
from tkinter import messagebox as mb

plt.close("all")
#Funciones Auxiliares
def Pol(Coef,x):
```

```
    v = 0
    for i in Coef:
        v += i*x**(Coef.index(i))
    return v

def IntArr(x,arr):
    sum =0.0000
    if len(x) == len(arr):
        for i in range(len(x)-1):
            sum += (x[i+1]-x[i])*((arr[i+1]+arr[i])/2)
    else:
        print("Funcion no valida, no coincide
        las dimensiones de los arreglos")
    return sum
def Fseno(Arr,x):
    return Arr[0]*math.sin(Arr[1]*x + Arr[2])
def Fcoseno(Arr,x):
    return Arr[0]*math.cos(Arr[1]*x + Arr[2])
def Expon(A,arg):
    return A*math.exp(arg)

Inicio =0
Final=10
Numdedivisiones=102
x = np.linspace(Inicio, Final, Numdedivisiones)
rho = []
phi1=np.zeros(len(x))
phi2 = np.zeros(len(x))
dif = np.zeros(len(x))

for i in x:
    if (i>=2 and i<=8):
        rho.append( Pol([-16,10,-1],i) )
    else:
        rho.append(0)

Ran = [10,100,1000,10000]  #Numero de Iteraciones
NumofPic=10 #Especifica el Numofit
path = "C:/Ubicacion"
intrie = []
```

```
plt.figure()
plt.plot(x, rho)
plt.title('Grafica de densidad',fontsize=30)
plt.ylabel(r'$\rho$',fontsize=30)
plt.xlabel('Vector x',fontsize=30)
plt.grid()

z= []
error = 1e-4
for m in Ran:
    y=[]
    Numofit = int(m/NumofPic)

    print("\n\n\n\n\n\n\n\n Calculo para
    la iteracion de:"+str(m))
    plt.figure()
    for s in range(m+1):
        phi1[0] = 0
        phi1[len(x)-1] =0
        for i in range(1,len(x)-1):
            R_i12 = x[i] + (x[i+1]-x[i])/2
            r_i12 = x[i] - (x[i+1]-x[i])/2
            phi2[i] = phi1[i]*(1- (R_i12**2 +
            r_i12**2)/(2*(x[i])**2) ) +
            0.5*(((R_i12**2)*phi1[i+1]
            + (r_i12**2)*phi1[i-1]  )
            /((x[i])**2)-rho[i])
        if(s%int((m/10))==0 and s!=0):
            print("Avance de los calculos en un: "+str(100*s/m)+"%
        if(s%Numofit==0):
            plt.plot(x, phi2)
            plt.title('Recursion hasta:'+str(s),
            fontsize=30)
            plt.ylabel('$\phi$',fontsize=30)
            plt.xlabel('R',fontsize=30)
            plt.grid()
            pname = path___N
            plt.savefig(pname)
        phi1=phi2
```

```
        (y.append(abs(IntArr(x,phi2)))) if(len(y)==0)
        else(y.append(abs(IntArr(x,phi2))))
        if (y[len(y)-1] - y[len(y)-2]) <=
        error and y[len(y)-1] - y[len(y)-2]>0:
            plt.grid()
            mb.showinfo("Termino de Iteraciones",
            "El proceso termino en la iteracion: "
            +str(s)+".
            Con un error de: "+str(error))
            break

print("\n\n\n\n\n\n\nCalculo del error:")
for i in y:
    if (y.index(i))%int((len(y)/10))==0:
        print("Avance de los calculos del error:
        "+str(int(100*y.index(i)/len(y)+1))+"%")
    z.append(abs(i-y[y.index(i)-1])) if(y.index(i)!=0)
    else z.append(abs(y[1]-y[0]))


plt.figure()
plt.plot(z)
plt.title('Grafica de la integral del error',
fontsize=30)
plt.annotate('$\int (\phi_n -
\phi_{n-1})dx$',(len(z)/5,max(z)*0.9),fontsize=30)
plt.xscale("log")
plt.grid()
```

6.1.4. Método de sobre-relajación sucesiva - parámetro γ variable

```
"""@author: power"""

import matplotlib.pyplot as plt
import numpy as np
import math
from tkinter import messagebox as mb


plt.close("all")
def Pol(Coef,x):
```

```
    v = 0
    for i in Coef:
        v += i*x**(Coef.index(i))
    return v

def IntArr(x,arr):
    sum =0.0000
    if len(x) == len(arr):
        for i in range(len(x)-1):
            sum += (x[i+1]-x[i])*((arr[i+1]+arr[i])/2)
    else:
        print("Funcion no valida,
        no coincide las dimensiones de los arreglos")
    return sum
def Fseno(Arr,x):
    return Arr[0]*math.sin(Arr[1]*x + Arr[2])
def Fcoseno(Arr,x):
    return Arr[0]*math.cos(Arr[1]*x + Arr[2])
def Expon(A,arg):
    return A*math.exp(arg)



Inicio = 0
Final=10
Numdedivisiones=102
x = np.linspace(Inicio, Final, Numdedivisiones)
rho = []
phi1=np.zeros(len(x))
phi2 = np.zeros(len(x))
phiA = np.zeros(len(x))
dif = np.zeros(len(x))

for i in x:
    if (i>=2 and i<=8):
        rho.append(Fseno([2,1.5,0],i))
    else:
        rho.append(0)

Ran = [10,100,1000,15000]  #Numero de Iteraciones
```

```
NumofPic=10 #Especifica el Numofit
path = "C:/Ubicacion"
intrie = []

plt.figure()
plt.plot(x, rho)
plt.tick_params(labelsize = 25)
plt.ylabel(r'$\rho$',fontsize=30)
plt.xlabel('X',fontsize=30)
plt.grid()

gamma = 1.945  #Valor para GAMMA
error = 1e-4
romper = False
for m in Ran:
    if romper :
        break
    y=[]
    Numofit = int(m/NumofPic)

    print("\n\n\n\n\n\n\n\n Calculo
    para la iteracion de:"+str(m))
    plt.figure()
    for s in range(m+1):
        phi1[0] = 0
        phi1[len(x)-1] = 0
        for i in range(1,len(x)-1):
            R_i12 = x[i] + (x[i+1]-x[i])/2
            r_i12 = x[i] - (x[i+1]-x[i])/2
            phi2[i] = phi1[i]*(1- (R_i12**2 + r_i12**2)
            /(2*(x[i])**2) )
            + 0.5*(((R_i12**2)*phi1[i+1] + (r_i12**2)
            *phi1[i-1]
            )/((x[i])**2)-rho[i])
            gamma = 2/(1+s*(math.pi**2)
            /((Numdedivisiones-2)**2))
            phiA[i] = gamma*phi2[i]+(1-gamma)*phi1[i]
        if(s%int((m/10))==0 and s!=0):
            print("Avance de los calculos en un: "
            +str(100*s/m)+"%")
```

```
        if(s%Numofit==0):
            plt.plot(x[1:], phiA[1:])
            plt.tick_params(labelsize = 25)
            plt.ylabel('$\phi$ Actual',fontsize=30)
            plt.xlabel('X',fontsize=30)
            plt.grid()
            path = "C:/Ubicacion"
            pname=path___N
            plt.savefig(pname)
        phi1=phiA
        (y.append(abs(IntArr(x,phiA))))
        if(len(y)==0)
        else(y.append(abs(IntArr(x,phiA))))
        if (y[len(y)-1] - y[len(y)-2])
        <= error and y[len(y)-1] - y[len(y)-2]>0:
            plt.grid()
            mb.showinfo("Termino de Iteraciones"
            ,"El proceso termino
            en la iteracion: "+str(s)+
            ". Con un error de: "
            +str(error))
            print("El proceso termino en la iteracion:
            "+str(s)+". Con un error de: "+str(error))
            romper = True
            break
z= []
print("\n\n\n\n\n\nCalculo del error:")
for i in y:
    if (y.index(i))%int((len(y)/10))==0:
        print("Avance de los calculos del error:
        "+str(int(100*y.index(i)/len(y)))+"%")
    z.append(abs(i-y[y.index(i)-1])) if(y.index(i)!=0)
    else z.append(abs(y[1]-y[0]))


plt.figure()
plt.plot(z)
plt.tick_params(labelsize = 25)
plt.ylabel(r'$\phi$',fontsize=30)
plt.xlabel('X',fontsize=30)
```

```
plt.annotate('$\int [\phi^n(x) -
\phi^{n-1}(x))]dx$',(len(z)/5
,max(z)*0.9),fontsize=16)
plt.yscale("log")
plt.grid()
```

6.1.5. Método multigrid - tipo V

```
"""@author: Eduardo Medina"""

import matplotlib.pyplot as plt
import numpy as np
import math

plt.close("all")
def Pol(Coef,x):
    v = 0
    for i in Coef:
        v += i*x**(Coef.index(i))
    return v

def IntArr(x,arr):
    sum =0.0000
    if len(x) == len(arr):
        for i in range(len(x)-1):
            sum += (x[i+1]-x[i])*((arr[i+1]+arr[i])/2)
    else:
        print("Funcion no valida,
        no coincide las dimensiones de los arreglos")
    return sum
def Fseno(Arr,x):
    return Arr[0]*math.sin(Arr[1]*x + Arr[2])
def Fcoseno(Arr,x):
    return Arr[0]*math.cos(Arr[1]*x + Arr[2])
def Expon(A,arg):
    return A*math.exp(arg)
def Malla_Gruesarho(vec):
    sol= []
    for i in range(0,int((len(vec))/2)):
        sol.append((vec[i*2]+vec[i*2+1] )/2)
    return sol
```

```
def Malla_Gruesa(vec):
    sol= []
    for i in range(0,int((len(vec))/2)):
        sol.append((vec[i*2]+vec[i*2+1] )/2)
    sol[0] = 0
    sol[len(sol)-1]=0
    return sol
def Malla_Fina(vec,vecs,vecg):
    sol= []
    for i in range(0,len(vecg)):
        sol.append(vecg[i] + vecs[int(i/2)]
        -vec[int(i/2)])
    return sol

def Nueva_Solucion(r, p1,delx):
    p2 = np.zeros(len(r))
    for i in range(1,len(r)-1):
        p2[i] = 0.5*(p1[i-1]+p1[i+1]-r[i]*(delx**2))
    return p2
Inicio =0
Final=2*math.pi
Numdedivisiones=2**8
x = np.linspace(Inicio, Final, Numdedivisiones)
rho = []
phi1=np.zeros(len(x))
phi2 = np.zeros(len(x))

Rep=5000
for i in x:
    if (i>=0 and i<=2*math.pi):
        rho.append(1)
    else:
        rho.append(0)
NumofPic=10
path = "C:/Ubicacion"

y=[]
Numofit=10
error = 1e-4
```

```
savearrays = []
dx = abs(x[2]-x[1])
for s in range(Rep):
    phi1 = Nueva_Solucion(rho,phi1,dx)

    savearrays.append(phi1)
    phi1 = Malla_Gruesa(phi1)

    phi2 = Nueva_Solucion(Malla_Gruesarho(rho),
    phi1,2*dx)
    savearrays.append(phi2)
    phi1 = Malla_Gruesa(phi2)

    phi1 = Malla_Fina(phi1,phi2,savearrays[1])

    savearrays.pop()
    phi2 = Nueva_Solucion(Malla_Gruesarho(rho),
    phi1,2*dx)

    phi1 = Malla_Fina(phi1,phi2,savearrays[0])
    savearrays.pop()
    phi2 =phi1
    if((s+1)%int((Rep/10))==0 and s!=0):
            print("Avance de los calculos en un:
            "+str(int(100*s/Rep)+1)+"%")
            plt.plot(x, phi2)
            plt.grid()
            plt.tick_params(labelsize = 25)
            plt.ylabel('$\phi$',fontsize=30)
            plt.xlabel('x',fontsize=30)
    (y.append(abs(IntArr(x,phi2)))) if(len(y)==0)
    else(y.append(abs(IntArr(x,phi2))))


solucion =[]
for a in x:
    solucion.append(Pol([0,-2*math.pi,1],a)*0.4)
plt.figure()
plt.plot(x, solucion,label='Solucion Analitica')
plt.plot(x, phi1)
```

```
plt.tick_params(labelsize = 25)
plt.ylabel(r'$\phi$',fontsize=30)
plt.xlabel('Vector x',fontsize=30)
plt.grid()

z = []
print("\n\n\n\n\n\n\nCalculo del error:")
for i in y:
    if (y.index(i))%int((len(y)/10))==0:
        print("Avance de los calculos del error:
        "+str(int(100*y.index(i)/len(y)+1))+"%")
    z.append(abs(i-y[y.index(i)-1])) if(y.index(i)!=0)
    else z.append(abs(y[1]-y[0]))


plt.figure()
plt.plot(z)
plt.tick_params(labelsize = 25)
plt.ylabel('Valor del error',fontsize=30)
plt.xlabel('Numero de iteraciones',fontsize=30)
plt.annotate('$\int [\phi^n(x) -
\phi^{n-1}(x)]dx$',(len(z)/5,max(z)*0.9),fontsize=20)
plt.xscale("log")
plt.yscale("log")
plt.grid()
```

6.1.6. Método multigrid - tipo W

```
"""@author: Eduardo Medina"""

import matplotlib.pyplot as plt
import numpy as np
import math

plt.close("all")
def Pol(Coef,x):
    v = 0
    for i in Coef:
        v += i*x**(Coef.index(i))
    return v
```

```
def IntArr(x,arr):
    sum =0.0000
    if len(x) == len(arr):
        for i in range(len(x)-1):
            sum += (x[i+1]-x[i])*((arr[i+1]+arr[i])/2)
    else:
        print("Funcion no valida, no coincide
        las dimensiones de los arreglos")
    return sum
def Fseno(Arr,x):
    return Arr[0]*math.sin(Arr[1]*x + Arr[2])
def Fcoseno(Arr,x):
    return Arr[0]*math.cos(Arr[1]*x + Arr[2])
def Expon(A,arg):
    return A*math.exp(arg)
def Malla_Gruesarho(vec):
    sol= []
    for i in range(0,int((len(vec))/2)):
        sol.append((vec[i*2]+vec[i*2+1] )/2)
    return sol

def Malla_Gruesa(vec):
    sol= []
    for i in range(0,int((len(vec))/2)):
        sol.append((vec[i*2]+vec[i*2+1] )/2)
    sol[0] = 0
    sol[len(sol)-1]=0
    return sol
def Malla_Fina(vec,vecs,vecg):
    sol= []
    for i in range(0,len(vecg)):
        sol.append(vecg[i] + vecs[int(i/2)]
        -vec[int(i/2)])
    return sol

def Nueva_Solucion(r, p1,delx):
    p2 = np.zeros(len(r))
    for i in range(1,len(r)-1):
        p2[i] = 0.5*(p1[i-1]+p1[i+1]-r[i]*(delx**2))
    return p2
```

```
Inicio =0
Final=2*math.pi
Numdedivisiones=2**8
x = np.linspace(Inicio, Final, Numdedivisiones)
rho = []
phi1=np.zeros(len(x))
phi2 = np.zeros(len(x))

Rep=5000
for i in x:
    if (i>=0 and i<=2*math.pi):
        rho.append(1)
    else:
        rho.append(0)
NumofPic=10 #Especifica el Numofit
path = "C:/Ubicacion"

y=[]
Numofit=10
error = 1e-4
savearrays = []
dx = abs(x[2]-x[1])
for s in range(Rep):
    phi1 = Nueva_Solucion(rho,phi1,dx)

    savearrays.append(phi1) #Salvar el arreglo
    phi1 = Malla_Gruesa(phi1) #Hacer malla gruesa

    phi2 = Nueva_Solucion(Malla_Gruesarho(rho),
    phi1,2*dx)
    savearrays.append(phi2)
    phi1 = Malla_Gruesa(phi2)

    phi2 = Nueva_Solucion(Malla_Gruesarho
    (Malla_Gruesarho(rho)),phi1,4*dx)
    phi1 = Malla_Fina(phi1,phi2,savearrays[1])
    savearrays.pop()
    phi2 = Nueva_Solucion(Malla_Gruesarho(rho)
    ,phi1,2*dx)
```

```
    savearrays.append(phi2)
    phi1 = Malla_Gruesa(phi2)

    phi1 = Malla_Fina(phi1,phi2,savearrays[1])
    savearrays.pop()
    phi2 = Nueva_Solucion(Malla_Gruesarho(rho)
    ,phi1,2*dx)

    phi1 = Malla_Fina(phi1,phi2,savearrays[0])
    savearrays.pop()
    phi2 =phi1
    if((s+1)%int((Rep/10))==0 and s!=0) or s==1:
            print("Avance de los calculos en un:
            "+str(int(100*s/Rep)+1)+"%")
            plt.plot(x, phi2)
            plt.title('Recursion hasta:'+str(s+1),
            fontsize=30)
            plt.ylabel('$\phi$',fontsize=30)
            plt.xlabel('x',fontsize=30)
            plt.grid()
    (y.append(abs(IntArr(x,phi2)))) if(len(y)==0)
    else(y.append(abs(IntArr(x,phi2))))


plt.grid()
solucion =[]
for a in x:
    solucion.append(Pol([0,-2*math.pi,1],a)*0.44)
plt.figure()
plt.plot(x, solucion,label='Solucion Analitica')
plt.title('Solucion calculada vs Solucion Analitica',
fontsize=20)
plt.ylabel(r'$\rho$',fontsize=30)
plt.xlabel('Vector x',fontsize=30)

plt.plot(x, phi1)
plt.title('Grafica de la solucion',fontsize=30)
plt.ylabel(r'$\phi$',fontsize=30)
plt.xlabel('Vector x',fontsize=30)
plt.grid()
```

```
z = []
print("\n\n\n\n\n\n\nCalculo del error:")
for i in y:
    if (y.index(i))%int((len(y)/10))==0:
        print("Avance de los calculos del error:
        "+str(int(100*y.index(i)/len(y)+1))+"%")
    z.append(abs(i-y[y.index(i)-1])) if(y.index(i)!=0)
    else z.append(abs(y[1]-y[0]))


plt.figure()
plt.plot(z)
plt.title('Grafica de la integral del error',
fontsize=30)
plt.annotate('$\int (\phi_n -
\phi_{n-1})dx$',(len(z)/5,max(z)*0.9),fontsize=30)
plt.xscale("log")
plt.yscale("log")
plt.grid()
```

6.1.7. Método SOR para una malla 2D

```
"""@author: Eduardo Medina"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from tkinter import messagebox as mb
plt.close("all")
fig = plt.figure(figsize=plt.figaspect(0.5))


def error(A):
    s = 0
    for i in range(len(A)):
        for j in range(len(A)):
            s += A[i][j]
    return s

Inicio =-5
```

```
Final= 5
Numdedivisiones=2**6
X = np.linspace(Inicio, Final, Numdedivisiones)
Y = X
Xp=X
dx = (X[1]-X[0])
R = np.array([ [ 0 for i in range(Numdedivisiones) ]
for j in range(Numdedivisiones) ], dtype=float)
Z = np.array([ [ 0 for i in range(Numdedivisiones) ]
for j in range(Numdedivisiones) ], dtype=float)
Zsol = np.array([ [ 0 for i in range(Numdedivisiones) ]
for j in range(Numdedivisiones) ], dtype=float)
GP = np.array([ [ 0 for i in range(Numdedivisiones) ]
for j in range(Numdedivisiones) ], dtype=float)
for i,x in enumerate(X):
    for j,y in enumerate(Y):
        R[i,j] = np.cos((x**2 + y**2))

X, Y = np.meshgrid(X, Y)
ax = fig.add_subplot(1, 3, 1, projection='3d')
surface = ax.plot_surface(X, Y, R, rstride=1,
cstride=1,cmap=cm.coolwarm,
linewidth=0, antialiased=False)
plt.show()


flag = 0.01
E = []
N=1500
for s in range(N):
    if (s+1)%(int(N/10))== 0:
        print("Avance del proceso en :"
        +str(int(s/N*100)+1)+"%" )
    for i in range(1,len(Xp)-1):
        for j in range(1,len(Xp)-1):
            Zsol[i,j] = 0.25*( Z[i+1,j] + Z[i-1,j]
            + Z[i,j+1] + Z[i,j-1] - R[i,j]*(dx**2))
    Zsol[:,0] = np.zeros(len(Xp))
    Zsol[:,len(Xp)-1] = np.zeros(len(Xp))
    Zsol[0,:] = np.zeros(len(Xp))
```

```
        Zsol[len(Xp)-1,:] = np.zeros(len(Xp))
        Z = Zsol
        E.append(abs(error(Zsol)))
        if (E[len(E)-1] - E[len(E)-2])/E[0]
        <= flag and E[len(E)-1] - E[len(E)-2]>0:
            plt.grid()
            mb.showinfo("Termino de Iteraciones",
            "El proceso termino en la iteracion:
            "+str(s)+". Con un error relativo de:
            "+str(flag))
            break

z = []
for i in E:
    z.append(abs(i-E[E.index(i)-1])) if(E.index(i)!=0)
    else z.append(abs(E[1]-E[0]))

sv = []
for i in range(len(Z[0,:])):
    sv.append(max(abs(Z[i,:])))
s = max(sv)

ax = fig.add_subplot(1, 3, 2, projection='3d')
surface = ax.plot_surface(X, Y, Z, rstride=1,
cstride=1, cmap=cm.coolwarm,
linewidth=0, antialiased=False)
ax.set_zlim(-s, s)
plt.show()

for i in range(1,len(Xp)-1):
        for j in range(1,len(Xp)-1):
            GP[i,j] = ((Z[i+1,j] - Z[i-1,j])
            /(Xp[i+1]-Xp[i-1]))+ ((Z[i,j+1]
            - Z[i,j-1])/(Xp[j+1]-Xp[j-1]))

ax = fig.add_subplot(1, 3, 3, projection='3d')
surface = ax.plot_surface(X, Y, GP, rstride=1,
cstride=1, cmap=cm.coolwarm, linewidth=0,
antialiased=False)
plt.show()
```

6.1.8. Código hidrodinámico

```
! variables:
 ! primitive: pressure, density, vx, vy
 ! conservative: energy, density, rho*vx, rho*vy

!=======================================================================
 module runpars
   implicit none
   integer, parameter :: nx = 100
   integer, parameter :: ny = 100
   integer, parameter :: neq = 4
   real*8 :: size_min(2) = [-0.5d0, -0.5d0]
   real*8 :: size_max(2) = [ 0.5d0,  0.5d0]
   real*8 :: final_time = 0.01d0 ! 0.1d0
   real*8 :: u (neq,0:nx+1,0:ny+1)
   real*8 :: up(neq,0:nx+1,0:ny+1)
   real*8 :: p (neq,0:nx+1,0:ny+1)
   real*8 :: phi(0:nx+1,0:ny+1)
   real*8 :: dphi(2,0:nx+1,0:ny+1)
   real*8 :: S(neq,0:nx+1,0:ny+1)
   logical, parameter :: include_HD = .true.
   character(len=50) :: fileout = 'test001.dat'
   character(len=70) :: fn
 end module runpars
!=======================================================================

!=======================================================================
 program mezcal
   use runpars
   implicit none
   real*8 :: dt, pr(neq), pl(neq), dx
             , dy, x(0:nx+1), y(0:ny+1)
   real*8 :: integration_time = 0d0
   integer :: n_steps = 0, i, j, iter
   real*8  :: vxcf, cs, r_0
   real*8 :: fx(neq,0:nx,0:ny)
   real*8 :: fy(neq,0:nx,0:ny)
   real*8 :: rad
```

```
! size of the cells
dx = (size_max(1)-size_min(1))/ nx
dy = (size_max(2)-size_min(2))/ ny

! coordinates at the center of each cell
do i = 0, nx + 1
  x(i) = size_min(1) + dx*(i-0.5d0)
end do
do j = 0, ny + 1
  y(j) = size_min(2) + dy*(j-0.5d0)
end do
r_0 = 0.4

! initial conditions
do i = 1, nx
  do j = 1, ny
    rad = sqrt( x(i)**2 + y(j)**2)
    p(:,i,j) = 0d0
    p(2,i,j) = 1d0 - rad**2
    p(1,i,j) = 1d0 - 0.25d0*rad**2
    * (1d0 - 0.75d0*rad**2 + rad**4/6d0)
  end do
end do !Modificar la densidad

! compute the conservative variables
do i = 1, nx
  do j = 1, ny
    call primu(neq, p(:,i,j), up(:,i,j))
  end do
end do

phi = 0d0
! compute the gravitational potential (gravity step)
call poisson_solver(nx,ny,x,y,dx,up(2,1:nx,1:ny),
phi,iter)
call S_Solver(dx,dy,nx,ny,up,phi,S)

! saving the output
print*,'writing output file: ',fileout
open(unit = 10, file = 'test0.dat',
```

```
status = 'unknown')
write(10,*) '# x, y, p(:), phi, S(:)'
do i = 1, nx
  do j = 1, ny
    write(10,*) x(i), y(j), phi(i,j),
    p(2,i,j), S(:,i,j)
  end do
end do
close(unit=10)

! time iteration
n_steps = -1
do while((integration_time.lt.final_time))
  n_steps = n_steps + 1
  dt = 1d30
  do i = 1, nx
    do j = 1, ny
      cs = sqrt(5d0/3d0 * p(1,i,j) / p(2,i,j))  ! sound speed
      vxcf = abs(p(3,i,j)) + cs
      dt = min(dt,dx/vxcf)
    end do
  end do
  dt = dt*0.4d0

  u = up

  ! print the iteration number
  integration_time = integration_time + dt
  print*,n_steps, integration_time, final_time

  ! boundary conditions
  do j = 0,ny+1
    p(:, 0  ,j) = p(:, 1,j)
    p(:,nx+1,j) = p(:,nx,j)
  end do
  do i = 0,nx+1
    p(:,i, 0  ) = p(:,i, 1)
    p(:,i,ny+1) = p(:,i,ny)
  end do
```

```
    ! compute the flux along the x direction
    do i = 0, nx
      do j = 1, ny
        call fluxes(neq, p(:,i,j),
        p(:,i+1,j),fx(:,i,j))
      end do
    end do

    ! compute the flux along the y direction
    do i = 1, nx
      do j = 0, ny
        pl(:) = p(:,i,j);  pr(:) = p(:,i,j+1)
        call split(pl(3),pl(4)) ;
        call split(pr(3),pr(4))
        call fluxes(neq, pl(:),pr(:),fy(:,i,j))
        call split(fy(3,i,j),fy(4,i,j))
      end do
    end do

    call poisson_solver(nx,ny,x,y,
    dx,u(2,1:nx,1:ny),phi,iter)
    call S_Solver(dx,dy,nx,ny,u,phi,S)

    ! evolve the conserved variables (HD step)
    do i = 1, nx
      do j = 1, ny
        up(:,i,j) = u(:,i,j)  &
&                   - (fx(:,i,j)-fx(:,i-1,j)) * dt/dx  &
&                   - (fy(:,i,j)-fy(:,i,j-1)) * dt/dy  &
                    + S(:,i,j) * dt
      end do
    end do

    do i = 1, nx
      do j = 1, ny
        call uprim(neq,up(:,i,j),p(:,i,j))
      end do
    end do

  end do !Este end Do es el que termina el ciclo.
```

```
  ! saving the output
  print*,'writing output file: ',fileout
  open(unit = 10, file = fileout, status = 'unknown')
  write(10,*) '# x, y, pr, de, vx, vy, S'
  do i = 1, nx
    do j = 1, ny
      write(10,*) x(i),y(j),p(:,i,j), phi(i,j),
      dphi(1,i,j),dphi(2,i,j),S(:,i,j)
    end do
  end do
  close(unit=10)

subroutine split(a,b)
  implicit none
  real*8, intent(inout) :: a, b
  real*8 :: tmp
  tmp = a
  a = b
  b = tmp
end subroutine split

subroutine fluxes(neq, pl,pr,f)
  implicit none
  integer, intent(in) :: neq
  real*8, intent(in)  :: pl(neq), pr(neq)
  real*8, intent(out) :: f(neq)
  real*8 :: ul(neq), ur(neq), fl(neq), fr(neq)
  real*8 :: ap, am, cflmin, cflmax, cfrmin, cfrmax
  call primf(neq,pl,fl)
  call primf(neq,pr,fr)
  call primu(neq,pl,ul)
  call primu(neq,pr,ur)
  cflmin = pl(3) - sqrt(5d0/3d0 * pl(1) / pl(2))
  cflmax = pl(3) + sqrt(5d0/3d0 * pl(1) / pl(2))
  cfrmin = pr(3) - sqrt(5d0/3d0 * pr(1) / pr(2))
  cfrmax = pr(3) + sqrt(5d0/3d0 * pr(1) / pr(2))
  ap = max(0d0,cflmax,cfrmax)
  am = min(0d0,cflmin,cfrmin)
  f(:) = (ap*fl(:)-am*fr(:)+ap*am*(ur(:)-ul(:)))
```

```
  /(ap-am)
end subroutine fluxes

subroutine primf(neq,p,f)
  implicit none
  integer, intent(in) :: neq
  real*8, intent(in)  :: p(neq)
  real*8, intent(out) :: f(neq)
  f(1)  = (2.5d0*p(1)+0.5d0*p(2)*(p(3)**2+p(4)**2))
  * p(3)
  f(2)  = p(3)*p(2)
  f(3)  = p(3)*p(2)*p(3) + p(1)
  f(4)  = p(3)*p(2)*p(4)
end subroutine primf

subroutine primu(neq,p,u)
  implicit none
  integer, intent(in) :: neq
  real*8, intent(in)  :: p(neq)
  real*8, intent(out) :: u(neq)
  u(1)   = 0.5d0*p(2)*sum(p(3:4)**2) + 1.5d0*p(1)
  u(2)   = p(2)
  u(3:4) = p(3:4)*p(2)
end subroutine primu

subroutine uprim(neq,u,p)
  implicit none
  integer, intent(in) :: neq
  real*8, intent(in)  :: u(neq)
  real*8, intent(out) :: p(neq)
  p(1)   = 2d0/3d0 * ( u(1) - (u(3)**2+u(4)**2)*
  0.5d0/u(2) )
  p(2)   = u(2)
  p(3:4) = u(3:4)/u(2)
end subroutine uprim

subroutine poisson_solver(nx,ny,x,
y,dx,dens,phi,iter)
  implicit none
  integer, intent(in)  :: nx, ny
```

```
integer, intent(out) :: iter
real*8 , parameter   :: erro = 1d-8
real*8 , intent(in)  :: dens(1:nx,1:ny)
real*8 , intent(inout) ::
phi(0:nx+1,0:ny+1)
real*8 , intent(in)  ::
dx, x(0:nx+1), y(0:nx+1)
real*8  :: Dif
real*8  :: phi0(0:nx+1,0:ny+1), rad
integer :: i, j
integer, parameter   ::
numerodeciclos = 1000
phi0 = phi

Dif = 1000d0
! dirichlet conditions
iter = 0
do while(Dif>erro)
  iter = iter+1
  do j = 1, ny
    rad = sqrt( x(0)**2 + y(j)**2)
    phi0(0,j) = rad**2/4d0 -
    rad**4/16d0 - 3d0/16d0
    rad = sqrt( x(nx+1)**2
    + y(j)**2)
    phi0(nx+1,j) = rad**2/4d0
    - rad**4/16d0 - 3d0/16d0
  end do
  do i = 1, nx
    rad = sqrt( x(i)**2 + y(0)**2)
    phi0(i,0) = rad**2/4d0 -
    rad**4/16d0 - 3d0/16d0
    rad = sqrt( x(i)**2
    + y(ny+1)**2)
    phi0(i,ny+1) = rad**2/4d0
    - rad**4/16d0 - 3d0/16d0
  end do

  do i = 1, nx
    do j = 1, ny
```

```
          phi(i,j) = 0.25d0*( phi0(i+1,j)
          + phi0(i-1,j)  + phi0(i,j+1)
          + phi0(i,j-1) - dens(i,j)*dx**2 )
        end do
      end do
      Dif = 0.d0
      do i = 1, nx
        do j = 1, ny
          Dif = Dif + abs(phi(i,j)-phi0(i,j))
        end do
      end do
      phi0 = phi
    end do
  end subroutine poisson_solver

  subroutine S_Solver(dx,dy,nx,ny,up,phi,S)
     implicit none
     integer :: i, j
     integer, intent(in)  :: nx,ny
     real*8 , intent(in)  :: dx,dy
     real*8 , intent(in)  :: phi(0:nx+1,0:ny+1)
     real*8 , intent(in)  :: up(4,0:nx+1,0:ny+1)
     real*8 , intent(out) :: S(4,0:nx+1,0:ny+1)
     real*8 ::  FG(2,0:nx+1,0:ny+1)

    FG = 0.d0
    do i = 1, nx
       do j = 1, ny
        FG(1,i,j) = ( phi(i+1,j)-phi(i-1,j) )
        /(2*dx)
        FG(2,i,j) = ( phi(i,j+1)-phi(i,j-1) )
        /(2*dy)

        S(1,i,j)  = -up(3,i,j)*FG(1,i,j)
        - up(4,i,j)*FG(2,i,j)
        S(2,i,j)  = 0d0
        S(3,i,j)  = -up(2,i,j)*FG(1,i,j)
        S(4,i,j)  = -up(2,i,j)*FG(2,i,j)
      end do
    end do
```

```
end subroutine S_Solver
```