Universidad Nacional Autónoma de México

Facultad de Ingeniería

Estudio Numérico de la Evolución de un Sistema Ternario usando el Método de Campo de Fase

TESIS

Que para obtener el título de:

Ingeniero Mecánico

P R E S E N T A

Salvador Villarreal López Rebuelta

DIRECTOR DE TESIS

Dr. Marco Antonio Reyes Huesca

Ciudad Universitaria, Cd. Mx., 2020

A mi familia

Resumen

En este documento se presenta un análisis computacional del fenómeno de transporte de masa para sustancias no–miscibles en algunos sistemas ternarios, en una escala en la que es relevante el espesor de la interfase que separa a dichas sustancias. Se revisa el método de Campo de Fase, su aplicaión al problema de interés y su implementación numérica. La solución al problema numérico se realiza en Python.

Las simulaciones obtenidas permiten caracterizar algunas de las propiedades de los sistemas ternarios y comparar estos resultados con el caso más estudiado de sistemas binarios.

Agradecimientos

A mi papá, René, por darme un ejemplo a seguir. Te admiro mucho.

A mi mamá, Ana, por el apoyo y cariño constante.

A mi hermana, Stefania, por todas las enseñanzas.

A mi hermano, René, por entenderme.

A Melissa, por haber llegado a mi vida. Conocerte ha sido indudablemente de las mejores cosas que me han pasado. Gracias por estar conmigo.

A mis amigos y compañeros del Lab. de Reología: Eduardo Leiva, Elizabeth Tenorio, Elsa Reyes, Itzel Julio, Laylet Segoviano y Zaira Torres, gracias por hacer de mi estancia en el Lab. una experiencia muy divertida e interesante.

Al Dr. Marco y al Dr. Geffroy, gracias por guiarme y enseñarme tanto. Fueron una parte crucial del desarrollo de este trabajo en particular y de mi formación profesional en general.

Índice general

Índice de figuras

Índice de tablas

Introducción

Los *sistemas multicomponente* son aquellos que están conformados por dos o más sustancias distintas[1], aquellos constituidos por tres componentes son denominados **sistemas ternarios**. Esta clase de sistemas se encuentran en importantes aplicaciones industriales, tales como emulsiones, petróleos, aleaciones, sistemas agua–surfactante–aceite, etc. así como en otras áreas emergentes como los métodos de captura de carbono.

Debido a su presencia en varias ramas de la industria, los sistemas multicomponente son de gran relevancia. Sin embargo, su estudio presenta un grado de dificultad mayor que los monocomponente ya que requieren contemplar la interacción entre componentes, la presencia de multiples fases y la posible variación de las propiedades del medio en función de la composición, como pueden ser

[1] El número de componentes en consideración puede ser menor que el de sustancias: si hay sustancias de características similares se les puede agrupar en *pseudo-componentes*. Por ejemplo, en la inyección de CO_2 a una reserva de petróleo se tiene un pseudo-componente ligero de CO_2, C_1 y C_2, uno intermedio que agrupa C_3–C_6 y un pseudo-componente pesado para C_{7+}.

su densidad, viscosidad, módulos de elasticidad, etc. El modelo más adecuado dependerá del fenómeno en estudio.

En esta investigación se presenta el estudio de un sistema ternario genérico (*viz.*, sin incluir ecuaciones constitutivas para los componentes) usando un modelo multifásico para tamaños de interfase no despreciables. Se resuelve computacionalmente el modelo a fin de estudiar la evolución del sistema sujeto a transporte de masa.

Se seleccionó el **método de campo de fase** para modelar el sistema ternario. El método es conceptualmente más complejo que aquellos de interfaz abrupta ya que requiere cálculo de variaciones y resulta en una ecuación diferencial no lineal de cuarto orden, sin embargo tiene la ventaja de ser termodinámicamente consistente[2]. En este enfoque, como la interfase ocupa un volumen finito es posible incluir fuerzas interfaciales de manera más natural en la ecuación de balance de momentum. Este tipo de modelos son importantes para el estudio de sólidos y aleaciones. Por ejemplo, para un único constituyente se puede estudiar el proceso de solidificación usando dos campos de fase[3] (Boettinger, Warren, Beckermann & Karma, 2002); para dos componentes hay estudios sobre la solidificación

[2]Se parte de un funcional de energía libre obtenido con teoría de Ginzburg–Landau, esencialmente es una expansión alrededor del punto crítico.

[3]Cada campo de fase representa a un componente distinto, pero no son independientes, están sujetos a una condición local que garantiza que sus fracciones de masa sumen uno.

y transiciones isotérmicas de fase (S. G. Kim, Kim & Suzuki, 1999; Wheeler, Boettinger & McFadden, 1992); en sistemas ternarios hay investigaciones similares como la de solidifiación en Kobayashi, Ode, Kim, Kim y Suzuki, 2003.

Otra área de aplicación para estos modelos es la de fluidos multicomponente. En J. Kim y Lowengrub, 2006, se mencionan numerosas aplicaciones en las que se presentan esta clase de fluidos, incluyendo también alternativas de métodos de modelado. El modelo de campo de fase se ha utilizado también en el modelado de gotas, su evaporación (Safari, Rahimian & Krafczyk, 2014), su rompimiento y formación en junturas microfluídicas (De Menech, 2006; Liu & Zhang, 2009), su dinámica en presencia de surfactantes (Liu & Zhang, 2010), entre otros.

Para estudiar fenómenos complejos, tales como los mencionados arriba, es necesario suplementar al modelo de campo de fase con ecuaciones constitutivas para aportarle al medio las características de un material. Dichas ecuaciones pueden ser de carácter mecánico, termodinámico, electrodinámico o una combinación de éstas.

Habiendo introducido el campo de aplicación y acotado el alcance de este trabajo, se puede establecer su objetivo: *Estudiar la evolución de un sistema ternario usando un método de interfase difusa, el modelo de campo de fase, con la ecuación de Cahn-Morral, que permita ajustar los parámetros de la ecuación y calcular las cantidades*

físicas más importantes del sistema. La implementación del algoritmo se hizo en Python. Si bien el producto del presente trabajo no será suficiente para modelar la dinámica de un material real, sí podrá utilizarse como una base para implementar simulaciones de sistemas con interacciones más complejas.

En cuanto a organización, este trabajo se divide en dos partes. La primera está dedicada a todo lo necesario para implementar el método de campo de fase, incluyendo los aspectos teóricos en el Capítulo 2 y los aspectos prácticos de la implementación en el Capítulo 3 (los programas en sí se encuentran en el Anexo A). La segunda parte esta enfocada en las pruebas que se realizaron al método que se implementó, en el Capítulo 4 se resumen y analizan los resultados obtenidos de las simulaciones mientras que en el Capítulo 5 se analizan los resultados y se presentan conclusiones.

Parte I

Implementación

Capítulo 1

Marco Teórico

En este Capítulo se presentan las bases necesarias de los modelos de interfase difusa junto con las herramientas matemáticas que estos requieren, a fin de presentar de una manera relativamente autosuficiente al modelo, haciendo énfasis en los métodos de campo de fase. Por espacio y enfoque del documento no se profundiza en otros métodos que consideren interfase difusa, una visión más completa se encuentra en el artículo Diffuse Interface Methods in Fluid Mechanics de Anderson, Mcfadden y Wheeler, 1998.

Además, se introducen métodos numéricos para la solución de ecuaciones en derivadas parciales junto con algunos conceptos relevantes como consistencia, estabilidad, convergencia, error de truncamiento y rigidez. Los métodos presentados en este Capítulo se limitan a aquellos que se utilizaron dentro de los programas.

El tratamiento de la separación entre dos materiales distintos ha sido tratada de dos maneras distintas, la interfaz abrupta y la interfase continua.

1.1. Interfaz Abrupta

La naturaleza de la interfase entre dos fluidos ha sido de interés por más de 200 años. A inicios del siglo XIX, Young, Laplace y Gauss consideraban a esta separación como una *superficie matemática* (que se puede representar con una parametrización local, mediante una función $f : U \subset \mathbb{R}^2 \to \mathbb{R}^3$) provista de propiedades como tensión superficial pero sin espesor, a este modelo se le denomina *interfaz*. Esta perspectiva se observa en la ecuación de Young–Laplace, como se aprecia en la ecuación (Anderson y col., 1998).

$$\Delta p = -\sigma \boldsymbol{\nabla} \cdot \mathbf{n} = 2\sigma\kappa_M \tag{1.1}$$

donde Δp es la diferencia de presión entre ambos lados de la interfaz, σ es la tensión superficial entre los dos fluidos, $\mathbf{n}$ es el vector normal a la superficie y κ_M es la curvatura media (el promedio de las curvaturas principales) de la superficie. La ecuación (1.1) ilustra que las propiedades de los fluidos en general no son continuas a través de una interfaz de este tipo, por esto se le denomina *interfaz abrupta*. En particular, el cambio de presión se da a través de la superficie matemática lo que indica que el campo de presión presenta una discontinuidad (en la representación local, ésta se ubicaría justamente sobre la imágen $f(U)$).

1.2. Interfase Difusa

Hacia mediados y finales del siglo XIX, Poisson, Maxwell y Gibbs identificaron a la interfase como una región a través de la cual las propiedades de los fluidos hacen una transición rápida pero continua entre los valores en el bulto. Posteriormente, Rayleigh, van der Waals y Korteweg completaron la teoría. Los **Métodos de Interfase Difusa** aplican esta idea de transición continua entre propiedades materiales a la resolución de problemas múlti-componente.(Anderson y col., 1998).

1.2.1. Modelos de Campo de Fase

Dentro de la categoría de métodos de interfase difusa se encuentran los modelos de campo de fase. Estos se basan en el uso de un parámetro ϕ, que indica la proporción que hay de cada componente. A cada punto del espacio $\mathbf{x}$ se le asigna un valor $\phi(\mathbf{x})$, esto es justamente la definición de un *campo escalar* y de ahí recibe su nombre el modelo. Usualmente la proporción a la que se asocia ϕ es la fracción de masa, sin embargo el campo de fase se puede asociar a cualquier propiedad física de los bultos (que sea distinta entre los constituyentes).

En el caso de dos componentes las ecuaciones de evolución del sistema se obtienen de la forma siguiente. Se parte del funcional de energía libre de Helmholtz

dado por la densidad de energía de Ginzburg-Landau[1]. (Penrose & Fife, 1990).

$$\mathcal{F}[\phi] = \int_{\Omega \subset \mathbb{R}^3} \left[f(\phi(\mathbf{x}, t)) + \frac{1}{2}\kappa \|\nabla\phi(\mathbf{x}, t)\|^2 \right] \mathrm{d}v \tag{1.2}$$

en donde f se puede entender como una densidad local de energía libre de bulto y el segundo término $\frac{1}{2}\kappa\|\nabla\phi(\mathbf{x})\|^2$ se puede entender como la energía asociada a una interfase y Ω es el volumen de integración. Si el sistema se encuentra en equilibrio, entonces $\mathcal{F}$ presentará un valor mínimo. Para poder hablar de mínimos en funciones de funciones (funcionales) como lo es $\mathcal{F}$, es necesario hablar de variaciones, que se entienden como pequeños cambios en ϕ análogos a las diferenciales para el cálculo en dimensiones finitas. Una variación es una función $\zeta(\mathbf{x})$ con segundas derivadas continuas[2] que es cero en la frontera $\partial\Omega$ de Ω (para no alterar las condiciones de frontera) y que sea suficientemente pequeña[3], o en términos matemáticos,

$$\zeta \in H = \left\{ \eta \in C^2(\Omega) \middle| \ \eta(\mathbf{x}) = 0, \forall \mathbf{x} \in \partial\Omega \right\}$$

[1] No se justificará la forma de esta energía ya que ello requeriría el uso de teoría de transiciones de estado de Ginzburg y Landau, lo cual está fuera del alcance de este documento.

[2] La razón por la que se requiere que las variaciones sean de clase C^2 (segundas derivadas continuas) es que se tiene que satisfacer la ecuación de Euler-Lagrange que es una ecuación diferencial de segundo orden. Aunque en realidad se podría relajar esta restricción para incluir también variaciones de clase C^1 (primeras derivadas continuas) si se utiliza la *forma integrada de la ecuación de Euler–Lagrange*. (Kot, 2014).

[3] En un tratamiento riguroso es necesario escoger una norma para el espacio de funciones, y dependiendo de esta elección se modifica el espacio de funciones admisibles. (Kot, 2014). Pero en términos prácticos se puede escoger cualquier función que sea C^2 y desaparezca en la frontera y escalarla mediante una multiplicación por un número ε, que puede ser arbitrariamente pequeño.

van Brunt, 2010, nos dice que si $\mathcal{F}$ presenta un mínimo en ϕ, se cumplirá que para cualquier variación ζ de las características mencionadas arriba,

$$\delta\mathcal{F}(\phi,\zeta) = \int_{\Omega} \zeta \frac{\delta\mathcal{F}}{\delta\phi}\, \mathrm{d}v = 0 \tag{1.3}$$

donde $\delta\mathcal{F}/\delta\phi$ es la derivada variacional de $\mathcal{F}$ con respecto a ϕ, y está dada por:

$$\frac{\delta\mathcal{F}}{\delta\phi} \coloneqq \frac{\partial\mathcal{L}}{\partial\phi} - \boldsymbol{\nabla}\cdot\frac{\partial\mathcal{L}}{\partial(\boldsymbol{\nabla}\phi)} \tag{1.4}$$

donde $\mathcal{L}$ es el integrando de (1.2). Además se usó la notación de derivada "con respecto a un vector",

$$\frac{\partial\mathcal{L}}{\partial(\boldsymbol{\nabla}\phi)} \coloneqq \sum_i \frac{\partial\mathcal{L}}{\partial(\partial\phi/\partial x_i)}\hat{\mathbf{e}}_i$$

La ecuación (1.3) se puede pasar a su versión diferencial usando el lema de *du Bois–Reymond* (también llamado el lema fundamental del cálculo de variaciones), con lo que se obtiene la **ecuación de Euler–Lagrange**,

$$\frac{\delta\mathcal{F}}{\delta\phi} = 0 \tag{1.5}$$

En nuestra energía (1.2) podemos aplicar la ecuación (1.5) y considerando a κ como una constante se obtiene una condición de equilibrio del sistema,

$$f'(\phi) - \kappa\nabla^2\phi = 0 \tag{1.6}$$

El lado izquierdo de (1.6) se denomina como potencial químico μ del sistema (aunque estríctamente no es el potencial químico de la termodinámica de equilibrio, Callen, 1985), ya que éste sólo será cero cuando se alcance el equilibrio. El potencial μ se puede usar para obtener la ecuación de evolución del sistema. (Novick-Cohen & Segel, 1984).

$$\frac{\partial \phi}{\partial t} = \nabla \cdot [M(\phi) \nabla \mu]$$

donde $M(\phi)$ es la movilidad. Al sustituir aquí la ecuación (1.6), se obtiene:

$$\frac{\partial \phi}{\partial t} = \nabla \cdot \left[M(\phi) \nabla \left[f'(\phi) - \kappa \nabla^2 \phi \right] \right] \approx M \nabla^2 \left[f'(\phi) - \kappa \nabla^2 \phi \right] \tag{1.7}$$

donde el término de la derecha se obtiene si se considera que la movilidad $M(\phi)$ es aproximadamente constante para el problema[4]. A (1.7) se le conoce como la **ecuación de Cahn–Hilliard**, es no lineal (cuasilineal) y de cuarto orden.

La ecuación de Cahn–Hilliard se utiliza tanto en sólidos como en fluidos. Si se acopla la mecánica de fluidos es necesario considerar que las propiedades locales cambian también por efecto del flujo y resulta necesario añadir un término convectivo a la derivada temporal para "completar" una derivada material:

$$\frac{\mathrm{D}\phi}{\mathrm{D}t} \equiv \frac{\partial \phi}{\partial t} + \mathbf{v} \cdot \nabla \phi = \nabla \cdot \left[M(\phi) \nabla \left[f'(\phi) - \kappa \nabla^2 \phi \right] \right] \tag{1.8}$$

A esta versión se le conoce como la **Ecuación de Cahn–Hilliard Convectiva**.

[4]Sólo se indicó la dependencia de M con ϕ, pero en un problema general podría variar también con la temperatura que, al aumentar, facilita la difusión. Sin embargo el caso con M variable está fuera del alcance de este documento.

1.2.2. Ecuación de Cahn-Morral para Sistemas Multifásicos

El modelado de sistemas con más de dos fases requiere una versión generalizada de la ecuación (1.7), denominada **ecuación de Cahn-Morral**. Esta ecuación se obtiene de la versión generalizada de la energía de Ginzburg–Landau que para p fases tiene la forma:

$$\mathcal{F}[\boldsymbol{\phi}] = \int_{\Omega} \left[F(\boldsymbol{\phi}(\mathbf{x},t)) + \frac{1}{2}\kappa \sum_{i=1}^{p} \|\boldsymbol{\nabla}\phi_i(\mathbf{x},t)\|^2 \right] \mathrm{d}v \tag{1.9}$$

en donde $\boldsymbol{\phi} = \begin{pmatrix} \phi_1 & \phi_2 & \cdots & \phi_p \end{pmatrix}^T$ es un vector algebraico (una p–tupla, sin significado geométrico) que contiene las fracciones de masa de cada componente. De manera análoga a la ecuación de Cahn-Hilliard, se obtiene la evolución temporal del sistema como el *flujo gradiente* de (1.9). (Tavakoli, 2016). Las fracciones de masa deben sumar la unidad, entonces se deberá satisfacer la restricción equivalente:

$$\phi_p = 1 - \sum_{i=1}^{p-1} \phi_i \tag{1.10}$$

Debido a la restricción, se tendrán $p-1$ ecuaciones independientes de evolución del sistema, de acuerdo a lo desarrollado por Tavakoli, 2016, éstas serán:

$$\frac{\partial \phi_i}{\partial t} = \nabla^2 \left[\frac{\partial F}{\partial \phi_i}(\boldsymbol{\phi}) - \kappa \nabla^2 \phi_i \right] - \nabla^2 \left[\frac{\partial F}{\partial \phi_p}(\boldsymbol{\phi}) - \kappa \nabla^2 \phi_p \right], \quad i = 1, \ldots, p-1 \tag{1.11}$$

Sustituyendo la ecuación (1.10) llegamos a un sistema de $p-1$ ecuaciones con $p-1$ incógnitas.

Estas ecuaciones se podrán resolver si se cuenta con unas condiciones iniciales adecuadas $\phi_0 = \phi(\mathbf{x}; t = 0)$ que cumplan con la restricción (1.10), así como alguna condición de frontera. Se pueden usar distintas condiciones en función de que clase de sistema se desea modelar. Por ejemplo, para paredes impermeables se utilizan condiciones de frontera de von Neumann:

$$\mathbf{n} \cdot \nabla\phi(\mathbf{y}) = \mathbf{0}, \quad \forall\ \mathbf{y} \in \partial\Omega \tag{1.12}$$

donde $\partial\Omega$ es la frontera de Ω y $\mathbf{n}$ es el vector normal a $\partial\Omega$.

En este proyecto se trabajó el caso ternario ($p = 3$) y se tomaron **condiciones de frontera periódicas**, es decir que para cualquier cantidad física $\boldsymbol{\Psi}$ se tiene:

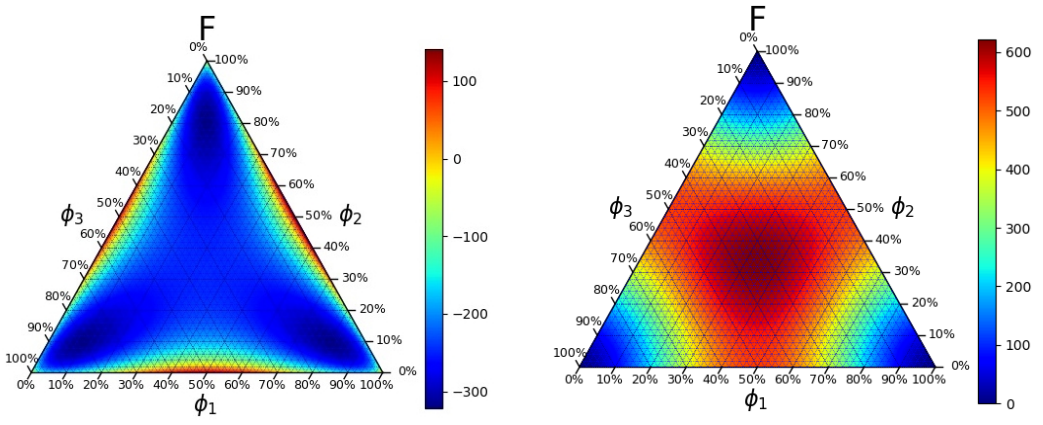
$$\boldsymbol{\Psi}(\mathbf{x}, t) = \boldsymbol{\Psi}(\mathbf{x} + n_1\mathbf{p}_1 + n_2\mathbf{p}_2 + n_3\mathbf{p}_3, t) \tag{1.13}$$

donde $n_i \in \mathbb{Z}$ y $\mathbf{p}_i$ son vectores constantes. En otras palabras, $\boldsymbol{\Psi}$ es periódica en cada una de las direcciones $\mathbf{p}_i/\|\mathbf{p}_i\|$, con periodo $\|\mathbf{p}_i\|$. Este tipo de condición se cumple trivialmente cuando se utilizan métodos espectrales para resolver las ecuaciones diferenciales, ya que las soluciones se aproximan usando series de Fourier que por definición aplican para funciones periódicas. En este proyecto se utilizó este tipo de método.

Matemáticamente, el potencial $F(\phi(\mathbf{x}))$ puede ser arbitrario pero para un sistema físico razonable, se requiere que F tenga pozos dentro de $\mathcal{D} =$

$\left\{\boldsymbol{\phi} \in [0,1]^3 \subset \mathbb{R}^3 \middle| \ \phi_1 + \phi_2 + \phi_3 = 1\right\}$ y aumente monotónicamente al alejarse de este dominio (se podría pensar en una función triestable de manera análoga al caso "biestable" de dos fases), esto garantiza que los campos se mantengan dentro de un rango razonable, que ninguno sea negativo o mayor que la unidad. Una ventaja de la ecuación (1.11) es que su solución cumplirá con la restricción (1.10) cuando $\boldsymbol{\phi}_0$ la cumple, entonces sólo será necesario verificar que se cumpla $\phi_i \in [0,1]$. (Tavakoli, 2016).

Dos ejemplos de pozos de potencial se muestran en la Figura 1.1.



(a) $F(\boldsymbol{\phi}) = RT \sum_{i=1}^{3} \phi_i \ln(\phi_i) + \sum_{i=1}^{3} \sum_{j=i+1}^{3} \Theta_{ij} \phi_i \phi_j$ **(b)** $F(\boldsymbol{\phi}) = \frac{6}{\epsilon} \sum_{i=1}^{3} \Gamma_i \phi_i^2 (1-\phi_i)^2$

Figura 1.1: Gráficas de los dos tipos más usuales de potencial, el logarítmico y el polinómico. En *(a)* se tiene $T = 300K$ y $\Theta_{ij} = 3RT$. *(b)* es un modelo planteado por Boyer, Lapuerta, Minjeaud, Piar y Quintard, 2009, asocia los parámetros Σ_i a las tensiones superficiales σ_{ab} (entre el fluido a y el b), y están dados por $\Gamma_i = \sigma_{ij} + \sigma_{ik} - \sigma_{kj}$ con $i \neq j \neq k \neq i$, mientras que $\epsilon = {}^{4\kappa}/_{3}$ es un parámetro que da el ancho de la interfase, en este caso $\sigma_{ij} = 70^{mN}/_{m}$ y $\epsilon = 0.1mm$

Los pozos en la Fig. 1.1(a) se encuentran al interior del dominio, mientras que en 1.1(b) se encuentran en las esquinas. Esto quiere decir que, en términos de modelado de un sistema, se puede pensar que el potencial graficado en 1.1(a) modela un sistema donde no es posible separar perfectamente las fases, como agua y alcohol, mientras que el de 1.1(b) modela un sistema que se separa completamente, como agua y aceite (a bajas temperaturas).

Se escogió trabajar con una F de tipo logarítmico, como la que se muestra en la Fig. 1.1(a), este potencial corresponde a lo que se conoce en termodinámica como una *mezcla regular*. Por simplicidad trabajaremos con una variante adimensionalizada[5] y con todas las entalpías de mezclado Θ_{ij} iguales

$$F_{\log}(\boldsymbol{\phi}) = \theta \sum_{i=1}^{p} \phi_i \ln(\phi_i) + \theta_c \sum_{i=1}^{p} \sum_{j=i+1}^{p} \phi_i \phi_j \tag{1.14}$$

donde θ, θ_c son constantes reales que cumplen con la restricción $0 < \theta < \frac{1}{2}\theta_c$. Esta condición es necesaria para que el potencial tenga puntos de equilibrio estables dentro del dominio admisible $\mathcal{D}$. Para el caso de tres fases el potencial se reduce a:

$$F_{\log}(\boldsymbol{\phi}) = \theta[\phi_1 \ln(\phi_1) + \phi_2 \ln(\phi_2) + \phi_3 \ln(\phi_3)] + \theta_c(\phi_1\phi_2 + \phi_2\phi_3 + \phi_3\phi_1) \tag{1.15}$$

[5] La $F_{\log}$ de la ecuación (1.14) que se utilizará en este proyecto se obtiene la F en la Fig. 1.1(a) cuando Θ_{ij} son iguales y se divide todo el potencial entre su magnitud (en todas las simulaciones consideramos $\theta_c = 1$). Esta variante se escoge porque lo único que distingue a un sistema de otro (a parte quizá de la velocidad de evolución, la cual numéricamente sólo nos requiere reescalar a t por un factor constante para llegar al mismo resultado) son los valores relativos de los coeficientes y no su valor absoluto. Además condición $\Theta_{ij} = \theta_c$ se escoge para reproducir las condiciones usadas en Tavakoli, 2016, en cuyo modelo se basó este trabajo.

Si tomamos estas consideraciones en cuenta, las ecuaciones (1.11) resultan en:

$$\underbrace{\frac{\partial \phi_i}{\partial t}}_{\partial \phi_i/\partial t + \mathbf{v}\cdot\boldsymbol{\nabla}\phi_i} = \nabla^2\left[\theta \ln\left(\frac{\phi_i}{1-\phi_1-\phi_2}\right) - \theta_c(2\phi_i+\phi_j) - \kappa\nabla^2(2\phi_i+\phi_j)\right] \tag{1.16}$$

donde se tiene que $i, j \in \{1, 2\}$ con $i \neq j$. El término bajo el corchete corresponde a la versión convectiva de la ecuación. Se puede llegar a un modelo completo del sistema ternario si se acopla este sistema de ecuaciones con las ecuaciones de balance. La versión convectiva se utiliza cuando el material presenta flujo[6].

[6]El transporte de masa descrito por (1.16) no presenta flujo neto porque debido a (1.10) se tiene que $\mathbf{J} = \mathbf{j}_1 + \mathbf{j}_2 + \mathbf{j}_3 = 0$

1.3. Análisis Numérico

El análisis numérico es una rama de las matemáticas cuyo objetivo es encontrar aproximaciones con alta *"exactitud"* a las soluciones de problemas cuya solución exacta[7] es imposible o inviable. (Springer Publishing & European Mathematical Society, 2015). Un ejemplo del primer caso es obtener la solución general de las ecuaciones de Navier–Stokes, la cual es imposible obtener con las técnicas matemáticas existentes hasta la fecha, por lo que se recurre a la Mecánica de Fluidos Computacional. Ejemplos del segundo caso son la inversión de matrices o el cálculo de (algunas) transformadas de Fourier, estos se pueden realizar analíticamente pero requieren de mucho mayor tiempo de cómputo que su contraparte numérica.

Esta Sección está organizada de tal modo que se llegue a la aproximación en diferencias finitas del sistema de ecuaciones utilizado en los programas computacionales desarrollados para este proyecto.

[7] Esta oración es extraña en Español por la ambigüedad de la palabra "exactitud". En su primera aparición, exactitud se entiende en el contexto de análisis de errores: que la solución real (analítica) sea cercana o se encuentre en el rango de el[los] resultado[s] obtenido[s] en la solución numérica (en Inglés: *accuracy*), aunque el rango de variación podría ser muy amplio (una variación pequeña en el rango obtenido requiere una alta precisión, en Inglés: *precision*). (Bevington & Robinson, 2003). Por otro lado, la "solución exacta" que se menciona aquí se refiere a la solución analítica del problema.

1.3.1. Partición Convexa–Cóncava del Funcional de Energía

Las ecuaciones (1.16) están estrechamente relacionadas con el problema de minimización del funcional (1.9) ya que son el *flujo gradiente* de $\mathcal{F}$. Esto quiere decir que el sistema, en el espacio de configuraciones, "fluye" hacia un estado de energía mínima a través de la trayectoria de descenso más rápido. En la Figura 1.2 se muestra esquemáticamente este tipo de evolución de un sistema con dos grados de libertad (en nuestro caso hay infinitos ($\aleph_0$[8]) grados de libertad). El estado final puede ser un mínimo local o incluso un punto silla porque el gradiente ahí es cero.

Esto presenta un problema numérico, ya que se espera que el sistema tienda a una energía mínima, sin embargo (1.16) admite la solución trivial de una mezcla homogénea de concentración constante $\phi(\mathbf{x}, t) = \phi_0 \in \mathbb{R}^3$.

Entonces, resulta necesario buscar una forma de modificar el modelo para encontrar mínimos "más fuertes", lo que requiere tomar una idea relevante del cálculo variacional. Se parte de la ecuación de Euler–Lagrange (1.5) para obtener la dinámica de las fases, ésta es una condición necesaria para la minimización, pero no suficiente[9]. El criterio que utilizaremos es el de **convexidad del integrando**:

[8] La cardinalidad del espacio de soluciones es un infinito numerable porque estas deben estar en $C^4(\Omega)$ (la ecuación de Cahn–Morral es de cuarto orden por lo que las ϕs deben ser cuatro veces derivables). Como se sabe que $|C^0(\mathbb{R})| = \aleph_0$ (*i.e.*, una función continua se puede determinar con sólo sus valores racionales porque $\mathbb{Q}$ es denso en $\mathbb{R}$), $|\Omega| = |\mathbb{R}^3| = |\mathbb{R}|$ y $C^4(\mathbb{R}) \subset C^0(\mathbb{R})$ entonces $|C^4(\Omega)| \leq \aleph_0$. Pero $C^4(\Omega)$ es un conjunto infinto entonces sólo se puede tener $|C^0(\mathbb{R})| = \aleph_0$

[9] Se puede hacer una analogía con el caso de funciones: para que haya un extremo debe anularse

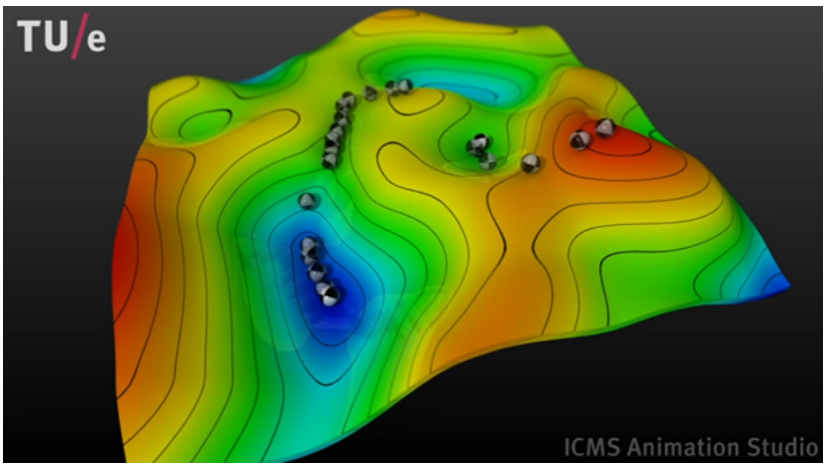


Figura 1.2: Flujo gradiente de un sistema desde dos posiciones iniciales. Ambos llegan al equilibrio en mínimos locales distintos. Imagen del *Institute for Complex Molecular Systems Animation Studio*, de la Eindhoven University of Technology, 16 de diciembre de 2011. Recuperado el 8 de abril de 2019, desde https://www.youtube.com/watch?v=vWFjqgb-ylQ

Teorema 1.3.1 (van Brunt, 2010)**.** *Sean el conjunto* $\Omega \subset \mathbb{R}^n$*, el punto* $\mathbf{x} \in \Omega$ *y la función* $\mathbf{y} \in (C^2(\Omega))^m$*. Sea* $\mathcal{F} : (C^2(\Omega))^m \to \mathbb{R}$ *un funcional de la forma* $\mathcal{F}[\mathbf{y}] = \int_\Omega \mathcal{L}(\mathbf{x}, \mathbf{y}, \nabla\mathbf{y})\, \mathrm{d}v$*. Y sean los conjuntos* $\Omega_{\mathbf{x}} = \{(\mathbf{y}, \nabla\mathbf{y}) \in \mathbb{R}^{m+nm} | \ (\mathbf{x}, \mathbf{y}, \nabla\mathbf{y}) \in \mathrm{dom}(\mathcal{L})\}$.

Suponiendo que para cada $\mathbf{x}$ *el conjunto* $\Omega_{\mathbf{x}}$ *es convexo y* $\mathcal{L}$ *es una función convexa de* $\left\{y_a, \partial y_a / \partial x_i\right\}_{(a,i)=(1,1),\ldots,(m,n)}$*. Si* $\mathbf{y}$ *es un punto estacionario de* $\mathcal{F}$ *(viz.,* $\delta\mathcal{F}(\mathbf{y}, \zeta) = 0$*), entonces* $\mathcal{F}$ *tiene un mínimo en* $\mathbf{y}$*.*

Para poder aplicar este criterio, es necesario es determinar si $\mathcal{L}$ es convexa o no.

la primera derivada, pero hay funciones, como $f(x) = x^3$, que tienen un punto con derivada cero sin ser éste un máximo o mínimo. La analogía termina aquí ya que mientras que para $f : \mathbb{R} \to \mathbb{R}$ la segunda derivada es condición suficiente, para los fucionales $\mathcal{F} : C^2(\Omega) \to \mathbb{R}$ no lo es.

El capítulo 10 de van Brunt, 2010 nos proporciona un modo de determinarlo.

Teorema 1.3.2 (van Brunt, 2010)**.** *Sea $\Omega \subset \mathbb{R}^n$ un conjunto convexo y sea $f : \Omega \to \mathbb{R}$ una función C^2. La función f es convexa en Ω si y sólo si para cada $\mathbf{x} \in \Omega$ la matriz Hessiana*

$$(\mathbf{H}_f(\mathbf{x}))_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Es una matriz positiva definida: $\mathbf{y}^T \mathbf{H}_f(\mathbf{x})\mathbf{y} > 0, \ \ \forall\ \mathbf{y} \in \mathbb{R}^n - \{\mathbf{0}\}$.

La inspección de (1.9) nos lleva a concluir que este funcional tiene dos partes, una convexa $\frac{1}{2}\kappa \sum_{i=1}^{p} \|\nabla\phi_i\|^2$ y una que no es ni cóncava ni convexa $F(\boldsymbol{\phi}(\mathbf{x}))$. Se procede como se muestra en Tavakoli, 2016, se suma un "cero" que no modificará al sistema físico pero sí permitirá tener dos funcionales, ambos convexos. Redefinimos:

$$\mathcal{F}[\boldsymbol{\phi}] \equiv \mathcal{F}_c[\boldsymbol{\phi}] - \mathcal{F}_e[\boldsymbol{\phi}] \tag{1.17}$$

donde los términos están dados por

$$\mathcal{F}_c[\boldsymbol{\phi}] := \int_\Omega \frac{1}{2}\kappa \sum_{i=1}^{p} \|\nabla\phi_i\|^2 \mathrm{d}v + \frac{c}{2}\int_\Omega \sum_{i=1}^{p} \left(\alpha\phi_i^2 + \beta\|\nabla\phi_i\|^2\right)\mathrm{d}v \tag{1.18a}$$

$$\mathcal{F}_e[\boldsymbol{\phi}] := \frac{c}{2}\int_\Omega \sum_{i=1}^{p} \left(\alpha\phi_i^2 + \beta\|\nabla\phi_i\|^2\right)\mathrm{d}v - \int_\Omega F(\boldsymbol{\phi})\mathrm{d}v \tag{1.18b}$$

donde $\alpha, \beta \in \overline{\mathbb{R}^+}$ y $c \in \mathbb{R}^+$ son parámetros de la partición. Tanto el término asociado a α como el asociado a β son convexos en el sentido dado por el teorema 1.3.2.

De esto podemos concluir que si el **parámetro de partición** c es suficientemente grande entonces el integrando de $\mathcal{F}_e$ será convexo[10]. Esto ayudará a evitar la solución trivial para $p < 5$, si $p \geq 5$ es necesario añadir una penalización adicional. (Tavakoli, 2016).

Con esto, las ecuaciones de evolución del sistema (1.11) quedan de la manera siguiente[11]:

$$\frac{\partial \phi_i}{\partial t} = \nabla^2 \left[\left(\frac{\delta \mathcal{F}_c}{\delta \phi_i} - \frac{\delta \mathcal{F}_c}{\delta \phi_p} \right) - \left(\frac{\delta \mathcal{F}_e}{\delta \phi_i} - \frac{\delta \mathcal{F}_e}{\delta \phi_p} \right) \right] \tag{1.19}$$

Se dejan separados los términos correspondientes a $\mathcal{F}_c$ de los de $\mathcal{F}_e$ ya que estos se resolverán de manera distinta, el primero se resolverá de manera implícita y el segundo de manera explícita (ver la Sección 1.3.3).

[10] Esto se observa claramente en la definición de la matriz Hessiana en el teorema 1.3.2. Por la linealidad de las derivadas parciales y de productos de matrices se tiene que $\mathbf{H}_{f+g}(\mathbf{x}) = \mathbf{H}_f(\mathbf{x}) + \mathbf{H}_g(\mathbf{x})$ y por lo tanto $\mathbf{y}^T \mathbf{H}_{f+g}(\mathbf{x})\mathbf{y} = \mathbf{y}^T \mathbf{H}_f(\mathbf{x})\mathbf{y} + \mathbf{y}^T \mathbf{H}_g(\mathbf{x})\mathbf{y}$. Como la multiplicación de matrices está acotada, existe un valor de c que hace que la matriz Hessiana de la suma sea positiva definida.

[11] El lector que se refiera a la fuente original, Tavakoli, 2016, observará diferencias en la notación. Esto se debe a que en el artículo se considera una variante de la derivada variacional (en lo que ahí se denota como ecuaciones (19) y (20)) utilizada para el espacio de Sobolev H^{-1}. En nuestro caso, nos limitamos a la definición usual (1.5) y consideramos que las ecuaciones de evolución se obtienen como se presentó en la Sección 1.2.1. Los términos $-\delta\square/\delta\phi_p$ toman en cuenta el hecho de que $\phi_p = \phi_p(\phi_1, \ldots, \phi_{p-1})$ no es independiente debido a la condición (1.10).

1.3.2. Método del Complemento de Schur

El método de Schur, descrito a continuación, pertenece a la categoría de *Métodos de Descomposición de Dominio*. Esta clase de métodos se utilizan para la resolución de sistemas que han sido descompuestos en varios subdominios, los cuales son más sencillos de resolver por separado de lo que sería resolver el sistema de manera global. (Saad, 2003)

La idea consiste en dividir las ecuaciones en bloques, uno que corresponde a los dominios y otro a la interfase entre los dominios, de acuerdo a lo que se muestra en Saad, 2003 se tiene:

$$\begin{pmatrix} D_1 & & & & & E_1 \\ & D_2 & & & & E_2 \\ & & D_3 & & & E_3 \\ & & & \ddots & & \vdots \\ & & & & D_n & E_n \\ F_1 & F_2 & F_3 & \cdots & F_n & B \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \\ y \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_n \\ s \end{pmatrix} \tag{1.20}$$

donde cada uno de los $\{x_i\}_{i=1,\ldots,n}$ es el subvector (algebraico) con las incógnitas del i–ésimo subdominio, mientras que y contiene las incógnitas de las fronteras. De este modo se puede entender a D_i como la matriz (u operador) que expresa el efecto de una fuente r_i en el interior del subdominio i. Por otra parte, las submatrices $E = (E_1 \ \cdots \ E_n)^T$ y $F = (F_1 \ \cdots \ F_n)$ podrán ser entendidas como el acoplamiento entre los subdominios y las fronteras visto, respectivamente, desde los subdominios y

las fronteras. (Saad, 2003). Podemos expresar este sistema de ecuaciones de una manera más sencilla usando *matrices por bloques*.

$$\begin{pmatrix} D & E \\ F & B \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} r \\ s \end{pmatrix} \tag{1.21}$$

De aquí se puede hacer reducción gaussiana por bloques para obtener:

$$x = D^{-1}(r - Ey) \tag{1.22a}$$

$$(B - FD^{-1}E)y = s - FD^{-1}r \tag{1.22b}$$

La matriz que aparece en el lado izquierdo de (1.22b) es propiamente la **matriz Complemento de Schur** de dónde el método recibe su nombre. La denotaremos por:

$$\mathcal{S} := B - FD^{-1}E \tag{1.23}$$

Un algoritmo que resuelva este sistema de ecuaciones deberá proceder con los pasos siguientes:

1. Obtener $s - FD^{-1}r$.
2. Formar la matriz complemento de Schur $\mathcal{S}$.
3. Resolver la ecuación (1.22b).
4. Obtener todas las demás incógnitas sustituyendo en (1.22a).

Una de las ventajas que presenta este método es que únicamente es necesario resolver directamente la parte de las fronteras y la solución para los dominios se obtiene de una simple sustitución[12].

Con esto surge la pregunta *¿Cómo se aplican los Métodos de Descomposición de Dominio a la ecuación de Cahn-Morral?* Uno podría pensar, inocentemente, que esta descomposición se hace entre los bultos de las distintas fases pero esto sería muy impráctico, habría que registrar la posición de la interfase a cada paso y la geometría, además de cambiar a cada paso de tiempo, puede ser muy compleja.

En realidad, el dominio se descompone de una manera abstracta: Se toman las $p-1$ primeras fases como los subdominios y la última fase será la "frontera" ya que como vimos en (1.11) es ésta la que acopla a las $p-1$ ecuaciones independientes.

[12]El algoritmo de uso general más eficiente para la solución de ecuaciones es el de LU. De lo expuesto en Pearce, 2007 podemos calcular que dicho algoritmo requiere $(2n^3+3n^2+n)/3$ *operaciones MAC* (Multiply-and-Accumulate Cycle) para una matriz de $n \times n$. En nuestro caso tenemos tres componentes lo que quiere decir que nuestro sistema de ecuaciones será de $3N \times 3N$ si cada fase tiene N incógnitas (*e.g.* el valor de ϕ_i en cada punto). Esto quiere decir que la solución directa (sin usar el método de Schur) requeriría de $18N^3 + 9N^2 + N$ operaciones MAC, una malla cuadrada modesta de 128 nodos requeriría de 7.92×10^{13} operaciones MAC por cada paso de tiempo. Usar el método de Schur implicará resolver sólo el sistema de $N \times N$, para los pasos 1 y 2 se harán $3N^3 + 3N^2 + N$ operaciones MAC y para el 4 serán $2N^2 + N$; en nuestro pequeño ejemplo se tendrán 1.61×10^{13} operaciones ó 4.91 veces menos.

1.3.3. Métodos de Aproximación por Diferencias Finitas

Para obtener el comportamiento de los sistemas físicos planteados en las secciones pasadas es necesario resolver ecuaciones diferenciales: encontrar una función (o una aproximación a ésta) que satisfaga una relación dada entre varias de sus derivadas sobre una región de interés, así como ciertas condiciones de frontera sobre los extremos de este dominio (*e.g.*, las ecuaciones (1.12) y (1.13)). En general, obtener estas soluciones es muy difícil y exceptuando los casos con múltiples simetrías, hay muy pocas ocasiones donde es posible obtener una fórmula analítica de la solución. (LeVeque, 2007).

Un **método de aproximación por Diferencias Finitas** reemplaza las derivadas en la ecuación diferencial con aproximaciones en diferencias finitas (*i.e.*, no infinitesimales), lo cual resulta en un sistema de ecuaciones **algebraicas** que podrá ser grande pero finito. Y gracias a la era digital[13] es sencillo (aunque puede ser tardado) resolver este sistema mediante una computadora.(LeVeque, 2007).

Se tomaron algunas de las fórmulas más usuales para aproximar derivadas mediante diferencias finitas del libro de LeVeque, 2007 y se resumen en la Tabla 1.1. En la tabla se muestra también el **error de truncamiento** de las fórmulas

[13]El ejemplo en la nota al pie #12 de la página 25 pone en evidencia que sólo el número de operaciones a realizar habría hecho imposible implementar este tipo de esquemas antes de que existieran las computadoras a menos que se contara con una fuerza de trabajo considerable.

correspondientes $D^n f - f^{(n)}$ ($D^n f$ denota la aproximación por diferencias finitas a la enésima derivada de f, mientras que $f^{(n)}$ denota la derivada exacta). Este error corresponde a la "parte" de la derivada que se está omitiendo, *viz.*, a qué orden se está aproximando la derivada.

Cabe mencionar que en la Tabla 1.1 se usa una notación compacta para las fórmulas de diferencias para una malla de separación constante Δx. Colocándose en un punto x_i de esta malla se tiene que:

$$f_{i+n} = f(x_i + n\Delta x), \quad n, i \in \mathbb{Z} \tag{1.24}$$

Primera Derivada	Error de truncamiento
$D_+ f = \dfrac{f_{i+1} - f_i}{\Delta x}$	$\dfrac{\Delta x}{2} f'' + O(\Delta x^2)$
$D_- f = \dfrac{f_i - f_{i-1}}{\Delta x}$	$-\dfrac{\Delta x}{2} f'' + O(\Delta x^2)$
$D_0 f = \dfrac{f_{i+1} - f_{i-1}}{2\Delta x}$	$\dfrac{\Delta x^2}{6} f''' + O(\Delta x^3)$
$D_3 f = \dfrac{2f_{i+1} + 3f_i - 6f_{i-1} + f_{i-2}}{6\Delta x}$	$\dfrac{\Delta x^3}{12} f^{(4)} + O(\Delta x^4)$
Segunda Derivada	
$D^2 f \equiv D_+ D_- f = \dfrac{f_{i+1} - 2f_i + f_{i-1}}{\Delta x^2}$	$\dfrac{\Delta x^2}{12} f^{(4)} + O(\Delta x^3)$

Tabla 1.1: Fórmulas de aproximación por diferencias finitas para la primera y segunda derivada de una función $f : \mathbb{R} \to \mathbb{R}$.

Además de las fórmulas que se resumen en la Tabla 1.1, también se tiene un esquema general para obtener derivadas, el cual consiste en ajustar coeficientes

de una serie de Taylor (este método se detalla en LeVeque, 2007). Para derivadas mayores a $n = 2$ se tienen expresiones generales para los casos de diferencias totalmente atrasadas, totalmente adelantas y centradas, las cuales son:

$$D_-^n = \frac{1}{\Delta x^n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} f_{i-k} \tag{1.25a}$$

$$D_+^n = \frac{1}{\Delta x^n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} f_{i+n-k} \tag{1.25b}$$

$$D_0^n = \frac{1}{\Delta x^n} \sum_{k=0}^{n} (-1)^k \binom{n}{k} f_{i+\frac{n}{2}-k} \tag{1.25c}$$

Es sencillo generalizar este método a funciones de varias variables, se aproxima una derivada parcial aplicando alguno de los operadores de la Tabla 1.1 o las ecuaciones (1.25) sobre el componente a lo largo de cuya dirección se quiere derivar. La notación de (1.24) también se generaliza fácilmente, para una función de N variables $f = f(x_1, \ldots, x_N)$.

$$f_{i_1+n_1, i_2+n_2, \ldots, i_N+n_N} = f((x_1)_{i_1} + n_1 \Delta x_1, \ldots, (x_N)_{i_N} + n_N \Delta x_N), \quad \mathbf{n}, \mathbf{i} \in \mathbb{Z}^N \tag{1.26}$$

donde $(x_a)_i$ es la a–ésima coordenada de un punto de la malla y Δx_a es la separación entre puntos de la malla en la dirección x_a. Con esto podemos obtener derivadas parciales con las expresiones de la tabla, por ejemplo:

$$\frac{\partial f}{\partial x_j} \approx (D_j)_0 f = \frac{f_{i_1, \ldots, i_j+1, \ldots, i_N} - f_{i_1, \ldots, i_j-1, \ldots, i_N}}{2\Delta x_j}$$

Este método se aplicó para la solución de las ecuaciones (1.16): Se aproximó a $\phi(\mathbf{x}, t)$ sobre una malla cuadrada en el espacio y con una separación constante en el tiempo. El lado izquierdo de (1.16) (la derivada temporal) se aproximó mediante diferencias finitas y la parte espacial se resolvió mediante un método de Fourier, presentados a continuación, en la Sección 1.3.4.

La selección de un esquema de diferencias finitas para la solución de una ecuación diferencial no es trivial, se distinguen en métodos *explícitos*, *implícitos* y *semi–implícitos*. Los métodos explícitos usan diferencias atrasadas, como D_-, y para resolverlos sólo hay que sustituir valores obtenidos en pasos anteriores; los implícitos usan diferencias que requieren pasos posteriores, como D_0, y resolverlos implica resolver un sistema de ecuaciones; finalmente, los métodos semi–implícitos resuelven una parte del sistema de forma explícita y otra de forma implícita.

La discretización en el tiempo se hizo de acuerdo al esquema *Semi-Implicit Backward Differentiation Formula* (SBDF) que se encuentra en [la ecuación (13) de] Ascher, Ruuth y Wetton, 1995. Este método se utiliza para ecuaciones que tienen

un término **rígido**[14] y uno no rígido.

$$\frac{\partial y}{\partial t} = \underbrace{g(y, \nabla y, \ldots, t)}_{\text{Rígido}} + \underbrace{f(y, \nabla y, \ldots, t)}_{\text{No-rígido}} \tag{1.27}$$

Para derivadas en el tiempo, la notación introducida en (1.24) suele usar la convención de que el espacio se etiqueta con subíndices[15] y el tiempo con superíndices. Considerando esto, la SBDF es:

$$\frac{y^{n+1} - y^n}{\Delta t} = g(y^{n+1}, \ldots, t_{n+1}) + f(y^n, \ldots, t_n) \tag{1.28}$$

Entre los términos rígidos más usuales están los de difusión lineal. (Ascher y col., 1995). Éste es el caso del biarmónico ($\nabla^4 = \nabla^2 \cdot \nabla^2$) de la ecuación de Cahn–Morral que se busca resolver.

[14] Una ecuación diferencial rígida se caracteriza por tener una solución estable que varía lentamente pero con soluciones cercanas que varían rápidamente, esto hace que errores pequeños alrededor de la solución estable (intrínsecos en cualquier aplicación numérica) causen fluctuaciones importantes. Por esto, una solución explícita se vuelve impráctica en tanto que el paso debe ser muy pequeño para que ésta sea estable. Debido a esto, los términos rígidos deben resolverse de manera implícita, como se observa en (1.28). (MathWorks®, 2003).

[15] En este documento no se usan los subíndices para este propósito salvo por la Sección 1.3.4 y los "elementos matriciales" de la ecuación (1.39), en los demás casos, los subíndices se utilizan para indicar el número del componente de un campo de fase.

1.3.4. Métodos de Fourier

Esta clase de métodos se basa en el uso de la **transformada de Fourier** y su inversa.

$$\hat{f}(\mathbf{k}) = \int_{\mathbb{R}^2} f(\mathbf{x}) e^{-2\pi i \mathbf{k}\cdot\mathbf{x}} \mathrm{d}\mathbf{x} \tag{1.29a}$$

$$f(\mathbf{x}) = \int_{\mathbb{R}^2} \hat{f}(\mathbf{k}) e^{2\pi i \mathbf{k}\cdot\mathbf{x}} \mathrm{d}\mathbf{k} \tag{1.29b}$$

En aplicaciones, como la de este proyecto, que no están directamente interesadas en la densidad espectral, los métodos de Fourier se utilizan como una herramienta computacional eficiente[16]. (Press y col., 1992).

La razón por la cual el dominio de integración se consideró como $\mathbb{R}^2$ (1.29) es que en este proyecto se tomó un sistema bidimensional (aunque la ecuación es igualmente válida si se sustituye $\mathbb{R}^2$ por $\mathbb{R}^n$). Una de las principales ventajas de estos métodos es que las derivadas se reducen a productos, operando con un gradiente en (1.29b) tenemos,

$$\nabla\left[\int_{\mathbb{R}^2} \hat{f}(\mathbf{k}) e^{2\pi i \mathbf{k}\cdot\mathbf{x}} \mathrm{d}\mathbf{k}\right] = \int_{\mathbb{R}^2} \hat{f}(\mathbf{k}) \nabla\left(e^{2\pi i \mathbf{k}\cdot\mathbf{x}}\right) \mathrm{d}\mathbf{k} = \int_{\mathbb{R}^2} \underbrace{2\pi i \mathbf{k} \hat{f}(\mathbf{k})}_{\widehat{\nabla f}} e^{2\pi i \mathbf{k}\cdot\mathbf{x}} \mathrm{d}\mathbf{k}$$

[16] En palabras de Press, Flannery, Teukolsky y Vetterling, 1992, *"Si mejoras un algoritmo no trivial en un factor cercano a un millón, el mundo creará un camino para encontrarle una aplicación útil."* (p. 530).

En la ecuación de la derecha reconocemos nuevamente la forma de la transformada de Fourier (1.29b). Generalizar este resultado es trivial,

$$\widehat{\frac{\partial f}{\partial x_i}}(\mathbf{k}) = 2\pi i k_i \hat{f}(\mathbf{k}),\ \widehat{\nabla f}(\mathbf{k}) = 2\pi i \mathbf{k} \hat{f}(\mathbf{k}),\ \widehat{\nabla^2 f}(\mathbf{k}) = -4\pi^2 k^2 \hat{f}(\mathbf{k}), \ldots \tag{1.30}$$

Este resultado es útil en tanto que las ecuaciones diferenciales se reducen a ecs. algebráicas al pasar al **dominio de la frecuencia** (*i.e.*, el espacio de las transformadas de Fourier). En el contexto de la Sección 1.3.3 esto equivale a pasar de ecuaciones de la Tabla 1.1 a multiplicar por un factor numérico, dado por (1.30).

En una implementación numérica no se usa la forma exacta, se recurre a la **Transformada Discreta de Fourier** (o DFT por sus siglas en inglés), que aproxima las integrales de (1.29). La versión más sencilla[17] consiste en sustituir la integral (1.29a) con sumas de Riemann,

$$\hat{\hat{f}}_{k_1 k_2} = \sum_{i_1=0}^{n-1} \sum_{i_2=0}^{m-1} f_{i_1 i_2} \exp\left[-2\pi i \left(\frac{i_1 k_1}{n} + \frac{i_2 k_2}{m}\right)\right] \tag{1.31}$$

donde $f(\mathbf{x})$ se aproximó sobre una malla de $n \times m$ y se usó la notación de (1.26) para expresarla como $f_{i_1 i_2}$. Similarmente, k_1 y k_2 corren desde cero hasta $n-1$ y $m-1$, respectivamente, por lo que $\hat{\hat{f}}_{k_1 k_2}$ será una aproximación de $\hat{f}(\mathbf{k})$ sobre una malla que también es de $n \times m$.

[17] Esta versión requiere realizar $O(n^2)$ operaciones y no es óptima en el uso de recursos de cómputo. Existe una versión, la **Transformada Rápida de Fourier** (FFT por sus siglas en inglés), que reduce las operaciones a $O(n \log n)$ pero por sencillez este documento se limita a presentar la, matemáticamente equivalente, DFT. El tema de FFT se puede encontrar en Press y col., 1992.

1.3.5. Esquema numérico para la solución de la Ecuación de Cahn-Morral

Con lo presentado en esta Sección se procede a expresar la ecuación de Cahn–Morral en la forma que será utilizada en los programas. Se parte de las ecuaciones (1.16) y (1.19),

$$\frac{\partial \phi_i}{\partial t} = \nabla^2[-(\kappa + c\beta)\nabla^2(2\phi_i + \phi_j) + c\alpha(2\phi_i + \phi_j)] + \nabla^2\left[c\beta\nabla^2(2\phi_i + \phi_j) - c\alpha\nabla^2(2\phi_i + \phi_j) + \theta \ln\left(\frac{\phi_i}{1 - \phi_1 - \phi_2}\right) - \theta_c(2\phi_i + \phi_j)\right] \quad (1.32)$$

y se aproxima $\partial\phi_i/\partial t$ Usando la SBDF. El primero y segundo de los términos en corchetes de (1.32) serán la parte implícita y explícita, respectivamente.

Primero se realiza un cambio de notación para reducir el tamaño de las ecuaciones y permita expresarlas mejor en una forma matricial,

$$\mathfrak{D}_c := (\kappa + c\beta)\nabla^4 - c\alpha\nabla^2$$

$$\mathfrak{D}_e := c\beta\nabla^4 - c\alpha\nabla^2$$

$$f_i(\boldsymbol{\phi}) := \theta(1 + \ln(\phi_i)) + \theta_c \sum_{j \neq i}^{3} \phi_j$$

$$\mathcal{N}(\phi_i) := \mathfrak{D}_e(\phi_i - \phi_3) + \nabla^2(f_i - f_3)$$

La ecuación en diferencias finitas resulta entonces:

$$\frac{\phi_i^{n+1} - \phi_i^n}{\Delta t} = -\mathfrak{D}_c(\phi_i^{n+1} - \phi_3^{n+1}) + \mathcal{N}(\phi_i^n)$$

y despejamos la parte implícita,

$$(\iota + \Delta t\ \mathfrak{D}_c)\phi_i^{n+1} - \Delta t\ \mathfrak{D}_c\phi_3^{n+1} = \phi_i^n + \Delta t\mathcal{N}(\phi_i^n) \tag{1.33}$$

donde ι es el operador identidad $\iota\colon x \mapsto x$. A partir de (1.33) podemos expresar las ecuaciones en forma matricial,

$$\begin{pmatrix} \iota + \Delta t\ \mathfrak{D}_c & 0 & -\Delta t\ \mathfrak{D}_c \\ 0 & \iota + \Delta t\ \mathfrak{D}_c & -\Delta t\ \mathfrak{D}_c \\ \iota & \iota & \iota \end{pmatrix} \begin{pmatrix} \phi_1^{n+1} \\ \phi_2^{n+1} \\ \phi_3^{n+1} \end{pmatrix} = \begin{pmatrix} \phi_1^n + \Delta t\mathcal{N}(\phi_1^n) \\ \phi_2^n + \Delta t\mathcal{N}(\phi_2^n) \\ 1 \end{pmatrix} \tag{1.34}$$

Identificamos en (1.34) la estructura de dominios descompuestos necesaria para la aplicación del método de Schur de la Sección 1.3.2, donde ϕ_1 y ϕ_2 representan el *interior de los dominios* y ϕ_3 la *frontera*. Esto se entiende considerando que ϕ_1 y ϕ_2 evolucionan independientemente salvo por un acoplamiento que está dado por la condición de conservación (1.10), en la que está involucrada ϕ_3.

El complemento de Schur para (1.34) es:

$$\mathfrak{S} = \iota + 2\Delta t\,(\iota + \Delta t\ \mathfrak{D}_c)^{-1}\,\mathfrak{D}_c \tag{1.35}$$

Y usamos las ecuaciones (1.22) para resolver el sistema para las ϕ_i^{n+1},

$$\phi_3^{n+1} = \mathfrak{S}^{-1}\left[1 - (\imath + \Delta t\ \mathfrak{D}_c)^{-1}(r_1^n + r_2^n)\right] \tag{1.36a}$$

$$\phi_i^{n+1} = (\imath + \Delta t\ \mathfrak{D}_c)^{-1}\left[r_i^n + \Delta t\ \mathfrak{D}_c \phi_3^{n+1}\right] \tag{1.36b}$$

donde se tomaron $r_i^n := \phi_i^n + \Delta t \mathcal{N}(\phi_i^n)$, los componentes no triviales del lado derecho de (1.34). Cabe recalcar que $\mathfrak{D}_{c,e}$ son constantes si la malla espacial es constante (*viz.*, si $\Delta x_i(t) = \Delta x_i(0)$, $\forall\ t \in \{t_1, t_2, \ldots, t_N\}$, donde t_i son los tiempos de la discretización). Si además Δt es constante, todos los operadores de (1.36) serán constantes y sólo habrá que calcularlos una vez, al inicio de la simulación[18].

Ahora procedemos a ver qué ocurre con la parte espacial. Como se mencionó anteriormente, se van a utilizar métodos de Fourier. Esto implica que los operadores diferenciales $\mathfrak{D}_c$ y $\mathfrak{D}_e$ pasarán a ser multiplicaciones por un polinomio en el vector de onda, en particular,

$$\mathfrak{D}_c \xrightarrow{f \to \hat{f}} (2\pi k)^4(\kappa + c\beta) - (2\pi k)^2 c\alpha, \quad \mathfrak{D}_e \xrightarrow{f \to \hat{f}} (2\pi k)^4 c\beta - (2\pi k)^2 c\alpha \tag{1.37}$$

Bajo esta transformación, todos los operadores de (1.36) en diferencias finitas son diagonales y al multiplicar por la inversa sólo se divide. Los operadores transformados al espacio de Fourier serán denotados por una tilde. Además,

[18] Estas consideraciones simplifican y aceleran a un programa sencillo de simulación, pero el precio de esto es sacrificar la posibilidad de incluir técnicas más sofisticadas, como multigrid y mallas adaptativas.

definimos $\mathfrak{A} := \iota + \Delta t\ \mathfrak{D}_c$ y también $\mathfrak{B} := -\Delta t\ \mathfrak{D}_c$. Con esto las ecuaciones definitivas serán:

$$\hat{\phi}_3^{n+1}(\mathbf{k}) = \tilde{\mathfrak{S}}^{-1}\left(\hat{1}(\mathbf{k}) - \tilde{\mathfrak{A}}^{-1}[\hat{r}_1^n(\mathbf{k}) + \hat{r}_2^n(\mathbf{k})]\right) \tag{1.38a}$$

$$\hat{\phi}_i^{n+1}(\mathbf{k}) = \tilde{\mathfrak{A}}^{-1}[\hat{r}_i^n(\mathbf{k}) - \tilde{\mathfrak{B}}\hat{\phi}_3^{n+1}] \tag{1.38b}$$

Estas ecuaciones son válidas para cualquier valor del vector de onda $\mathbf{k}$, en una aproximación por diferencias finitas se tienen valores discretos en un arreglo (tipo) matricial y en lugar de (1.38), se deberá multiplicar elemento por elemento (*producto de Hadamard*). Por ejemplo, en dos dimensiones los componentes del vector de onda son $\mathbf{k} = (k_1, k_2)$ y la ecuación (1.38a) lee:

$$\hat{\phi}_3^{n+1}(k_1,k_2) = \frac{\hat{1}(k_1,k_2) - \dfrac{\hat{r}_1^n(k_1,k_2) + \hat{r}_2^n(k_1,k_2)}{1 + \Delta t\ [16\pi^4 k^4(\kappa + c\beta) - 4\pi^2 k^2 c\alpha]}}{1 + 2\Delta t \dfrac{\Delta t\ [16\pi^4 k^4(\kappa + c\beta) - 4\pi^2 k^2 c\alpha]}{1 + \Delta t\ [16\pi^4 k^4(\kappa + c\beta) - 4\pi^2 k^2 c\alpha]}}$$

donde $k^2 := k_1^2 + k_2^2$, y $\hat{1}(k_1, k_2)$ es el valor de la transformada de Fourier de una función constante $f : (x_1, x_2) \mapsto 1$ evaluado en k_1, k_2. Considerando nuestro sistema discreto y denotando por $[A]_{k_1,k_2}$ a los elementos matriciales de A, esto se simplifica a:

$$\left[\hat{\phi}_3^{n+1}\right]_{k_1,k_2} = \frac{\left[\hat{1}\right]_{k_1,k_2} - \left[\tilde{\mathfrak{A}}\right]^{-1}_{k_1,k_2}\left(\left[\hat{r}_1^n\right]_{k_1,k_2} + \left[\hat{r}_2^n\right]_{k_1,k_2}\right)}{\left[\tilde{\mathfrak{S}}\right]_{k_1,k_2}} \tag{1.39}$$

En nuestro caso de una malla de $N \times N$ con una separación homogénea y constante Δx, se tiene que tanto k_1 y k_2 están dadas por:

$$k_1, k_2 \in \left\{ \frac{n}{N\Delta x} \middle| \, n \in \{ {}^{-N}\!/_{2}, {}^{-N}\!/_{2} + 1, \ldots, {}^{N}\!/_{2} - 1 \} \subset \mathbb{Z} \right\}, \text{ para } N \text{ par}$$

$$k_1, k_2 \in \left\{ \frac{n}{N\Delta x} \middle| \, n \in \{ {}^{-(N-1)}\!/_{2}, \ldots, {}^{(N-1)}\!/_{2} \} \subset \mathbb{Z} \right\}, \text{ para } N \text{ impar}$$

El proceso de solución se ilustra en la Figura 1.3, a continuación.

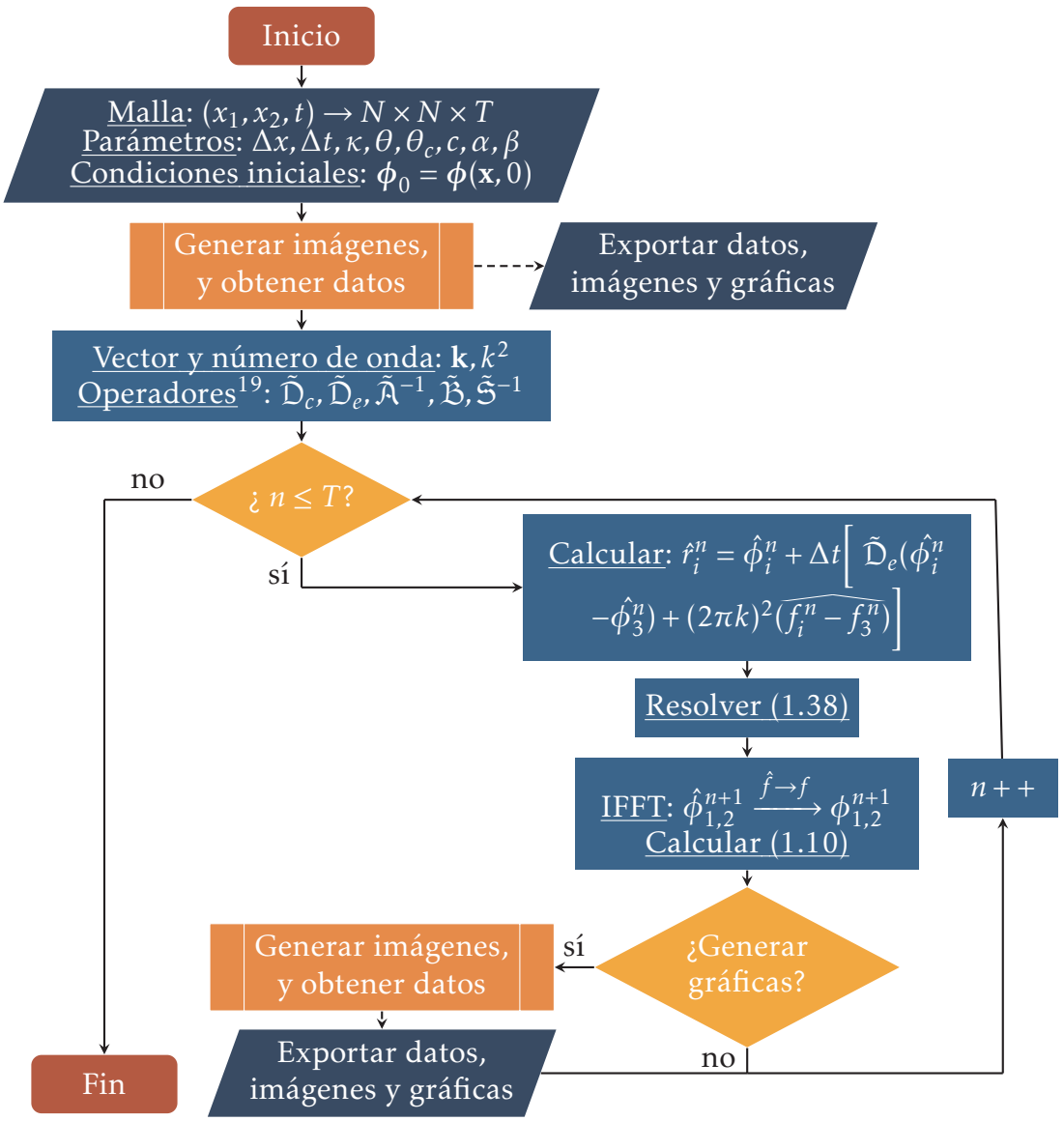


Figura 1.3: Diagrama de Flujo para la solución de las ecuaciones.

[19]Se calcula la inversa por que la multiplicación numérica es más rápida que la división (depende de la arquitectura, pero Granlund, 2017, muestra que dividir es de 5 a 10 veces más lento).

Capítulo 2

Metodología

En este Capítulo se describen y justifican los métodos, técnicas y herramientas utilizados para la realización de este trabajo de investigación. La discusión se limita únicamente a los métodos usados, sin profundizar en otros que podrían usarse en su lugar. El presente Capítulo sólo muestra un listado de los métodos, no se ha planteado como una introducción a ninguno de los temas.

2.1. Características Generales de la Investigación

La investigación de este tema fue de tipo **aplicado** dado que su fin es crear una herramienta computacional que permita modelar un sistema trifásico de fluidos. El carácter es **exploratorio**, ya que se buscó crear un programa cuyos resultados fuesen razonables de acuerdo a lo que se esperaría en teoría.

Los resultados a recopilar son de tipo **cuantitativo**, en la forma de gráficas y diagramas.

El programa permite modificar los parámetros del sistema y observar su efecto en la dinámica del sistema (ese efecto *a priori* no se conoce, por la no–linealidad de las ecuaciones). Por esto la investigación se clasifica, formalmente, como **numérica experimental**.

2.2. Herramientas

Todos los datos recopilados se obtuvieron directamente desde el programa en Python, que se implementó por completo para este proyecto y cuyo código se encuentra en el Anexo A.

Las simulaciones se corrieron en una computadora de escritorio del Instituto de Investigaciones de Materiales, con las siguientes características:

- Sistema Operativo: Windows 10 Enterprise 64-bits
- CPU: Intel® Core™ i7-8700 a 3.19GHz (6 núcleos)
- GPU: NVIDIA® GeForce® GT 620
- Memoria: 16GB de RAM, 969MB de VRAM y 8165MB de memoria gráfica compartida.
- Disco: 930GB HDD.

Para correr las simulaciones se utilizó el editor e interfaz gráfica PyCharm 2018.3.6 (Community Edition).

Se usó la versión de Python 3.7.3 de 64 bits por dos motivos *(i)* La arquitectura de la computadora y el procesador también son de 64 bits y *(ii)* El paquete utilizado

para realizar las transformadas de Fourier, `pyFFTW`, no es compatible con la versión de 32 bits, sólo con la de 64.

La paquetería instalada al interprete de Python constó de:

`cycler v0.10.0`	`kiwisolver v1.0.1`	`llvmlite v0.29.0`
`matplotlib v3.0.2`	`numba v0.44.1`	`numpy v1.16.1`
`pandas v0.24.2`	`Pillow v5.4.1`	`pyFFTW v0.11.1`
`pyparsing v2.3.1`	`python-dateutil v2.8.0`	`python-ternary v1.0.5`
`pytz v2019.1`	`regex v2019.6.8`	`scipy v1.2.0`
`six v1.12.0`	`xlrd v1.2.0`	

De estos, la librería `python-ternary` fue modificada. En particular se cambió la manera en la que se daba formato a las cadenas utilizadas en las etiquetas de los ejes del diagrama ternario[1]: Se reemplazó el operador binario `str % x` por el método `str.format(x)`.

Por simplicidad, se utilizó el entorno físico (se compila utilizando los paquetes de Python directamente instalados en el sistema, con el ejecutable) en lugar de un entorno virtual.

[1] El uso del operador `%` causaba errores al mostrar el texto. No se mostraban algunos caracteres y agregaba comillas redundantes en los extremos de la cadena.

2.3. Métodos

La solución implementada se basó en el articulo de Tavakoli, 2016, éste fue escogido porque, como se menciona, *"La implementación del método presentado es independiente del método de discretización espacial"*. Esto permitió utilizar métodos de Fourier para la parte espacial.

Se utilizó el esquema de discretización SBDF por que la ecuación tiene un término rígido, el biarmónico $\kappa\nabla^4\phi_i$, el cuál requiere resolverse de forma implícita, y la parte no lineal $\nabla^2(f_i - f_3)$ que sólo se puede resolver explícitamente. Los términos del "cero" que se suma en (1.18) se tomarón el positivo, implícito y el negativo, explícito[2].

La parte espacial se resolvió utilizando métodos de Fourier debido a que *(i)* Simplifica el planteamiento de las ecuaciones, *(ii)* En la versión FFT está muy optimizada, haciendola una alternativa rápida para el cálculo, *(iii)* Evita especificar las condiciones de frontera, ya que estas siempre son periódicas, *(iv)* La exactitud de las derivadas calculadas con estos métodos es mayor a lo que se obtiene resolviendo en el espacio físico las ecuaciones en diferencias finitas[3] y *(v)* Es un método que ha

[2]Si las dos se tomaran para el mismo tiempo se cancelarían trivialmente, y si sus coeficientes fuesen distintos, no sería numéricamente consistente con un cero.

[3]La derivada "espectral" (realizada en el espacio de Fourier, como se muestra en las ecuaciones (1.30)) tiene un error numérico de truncamiento del orden $O(N^{-m})$, es decir $O(\Delta x^m)$, para una función m-veces diferenciable. En nuestro caso se tiene que las soluciones de las ecuaciones son

sido usado exitosamente por el equipo del Laboratorio de Reología Óptica, lo que pone a disposición más opciones para la solución de problemas.

La programación se llevó a cabo en Python ya que éste es un lenguaje relativamente sencillo (en comparación con, por ejemplo, FORTRAN) y tiene bastantes opciones de optimización.

Se utilizó el wrapper (en Python, de la *Fastest Fourier Transform in the West*) `pyFFTW`, una de las opciones más rápidas en Python para realizar la FFT, porque se realiza en código máquina y permite paralelizar el proceso. Para paralelizar, se consideraron 12 procesadores lógicos, usando lo que se conoce como *hyperthreading*. Dentro de las opciones de esta librería, se escogió trabajar con la transformada real de Fourier por varios motivos, *(i)* Como se menciona en Frigo y Johnson, 2018: "*Es posible aprovechar estas circunstancias para lograr una mejora en aproximadamente un factor de dos tanto en velocidad como en uso de memoria*", *(ii)* Los arreglos en el espacio de Fourier son de $N \times (\lfloor N/2 \rfloor + 1)$ lo que implica que las operaciones necesarias para resolver (1.38) se reducen casi a la mitad, y *(iii)* Se evita la prescencia de "basura numérica" en la forma de una parte imaginaria cuando se regresa al espacio de las posiciones.

por lo menos 4 veces diferenciables, por ser EDPs de cuarto orden. Esto quiere decir que nuestras derivadas espaciales tienen una exactitud de, por lo menos, una derivada numérica de cuarto orden. (Malyshev, 2005)

Para optimizar más la simulación se utilizó, en todas las subrutinas compatibles, el *just-in-time compiler* (compilador justo-a-tiempo) `numba`, que es un wrapper[4] en Python para código máquina optimizado. Además, se procuró que las operaciones se realizaran con objetos de `numpy`, utilizando las operaciones "vectoriales"[5]. Para el cálculo de gradientes se definió una función (en lugar de usar `numpy.gradient`) para aprovechar la mejora en velocidad que provee `numba`[6].

Para graficar los resultados (se explica en la sección siguiente) se recurrió a tres librerías: para generar las imágenes en RGB, se usó `Pillow` (bifuración de la *Python Imaging Library*); las gráficas fueron generadas mediante `matplotlib`; y para generar los diagramas ternarios se necesitó la librería `python-ternary`.

[4]Una función cuyo propósito es llamar a otra subrutina utilizando la menor cantidad posible de recursos computacionales.

[5]No son vectores en el sentido matemático, así se le denomina a un tipo de operaciones de `numpy`.

[6]La compatibilidad de `numba` con funciones es limitada, `numpy.gradient` no está incluida.

2.4. Generación de Datos

Los datos que se obtuvieron de las simulaciones fueron las concentraciones de cada uno de los componentes sobre todos los puntos de la malla (que son las variables que resuelven las ecuaciones) y la energía libre de Helmholtz, la cual se calculó integrando numéricamente (se usó suma de Riemann de punto medio) la ecuación (1.9).

La información se extrajo directamente del programa durante el curso de la simulación y se llamó directamente a los métodos de graficación (véase el Anexo A) para evitar ocupar espacio en disco con los datos en bruto.

Para cada simulación se presentaron los datos como resultados gráficos de varios tipos, *(i)* Se generaron "fotografías" del sistema asignando a cada una de las tres fases puras un color en la escala RGB mediante la asignación $(\phi_1, \phi_2, \phi_3) \mapsto (\lfloor 255\phi_1 \rfloor, \lfloor 255\phi_2 \rfloor, \lfloor 255\phi_3 \rfloor)$ y se generó una imagen tomando estos valores como pixeles, *(ii)* Se hicieron histogramas donde la frecuencia graficada fue la concentración —de los puntos de la malla y sobre un diagrama ternario, *(iii)* Se obtuvieron gráficas de la energía libre de Helmholtz durante el transcurso de la simulación y, *(iv)* Se crearon gráficas de los valores máximos y mínimos de cada campo de fase durante el transcurso de la simulación.

El análisis de los datos obtenidos se basó en la interpretación de las gráficas mencionadas.[7]. Los resultados obtenidos y su interpretación, tal y como se detalló aquí, se encuentran a continuación en el Capítulo 3.

[7]Cada simulación genera más de 2 mil millones de datos en bruto ($p \times N \times N \times T = 3 \times 384 \times 384 \times 5000$), analizarlos sin darles algún formato gráfico sería imposible.

Parte II

Pruebas

Capítulo 3

Resultados

En este Capítulo se presentan los resultados obtenidos de las simulaciones numéricas del sistema ternario modelado con la ecuación de Cahn–Morral. Asímismo, se interpretan los resultados y se describen las diferencias obtenidas al modificar los parámetros del sistema. También se presenta una descripción de las gráficas utilizadas para reportar los resultados para aclarar exactamente qué se muestra. Algunos de los resultados obtenidos se llegan a comparar con valores obtenidos en otros trabajos donde se llegó a contar con dichos resultados. El rango dentro del cual se modificaron los parámetros fue en ordenes de magnitud cercanos a los valores utilizados en el artículo de Tavakoli, 2016.

3.1. Presentación de los resultados

En esta Sección se introduce el significado de todas las gráficas que se usaron para reportar los resultados de las simulaciones, así como la forma en la que se obtuvieron. Se comienza explicando el caso más simple de un sistema binario y luego se presenta su versión análoga para el caso de sistemas ternarios.

3.1.1. Sistemas Binarios

Para un sistema binario sólo se requiere un campo de fase, esto quiere decir que se puede indicar la composición en cada punto usando únicamente un escalar (en ese caso sólo se denota por ϕ al campo de fase ϕ_1 y se tiene que $\phi_2 = 1 - \phi$). Un ejemplo de la "fotografía" del sistema, adaptada al caso binario, que se mencionó en la Sección 2.4 se muestra en la Figura 3.1. En la barra de color se observa que a cada valor de composición se le asocia un color, el cual se utiliza para la gráfica de ϕ (la "fotografía") a su izquierda.

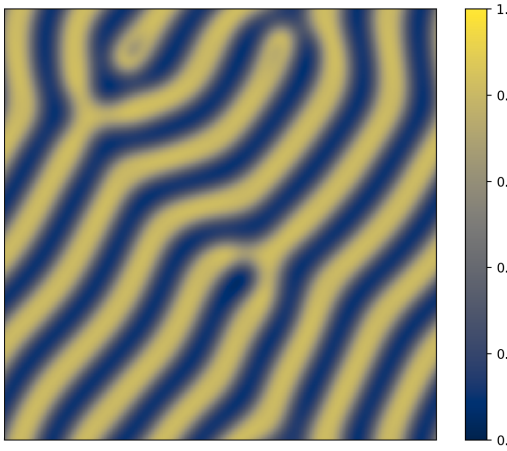


Figura 3.1: Ejemplo de un posible estado de un sistema binario. La imagen fue obtenida de una simulación en Python de la ecuación de Cahn–Hilliard, basada en la implementación de Fiocco, 2012.

Un aspecto que nos interesa es la distribución de las composiciones del sistema. Sabemos que los bultos se encontrarán en una concentración cercana a los mínimos del potencial $F(\phi)$ y las interfases tendrán valores intermedios. Para poder visualizar estos resultados necesitamos una gráfica que permita aproximar cantidades continuas cuando sólo se cuenta con un número limitado de datos. Las gráficas que se utilizan para este propósito son los **histogramas**.

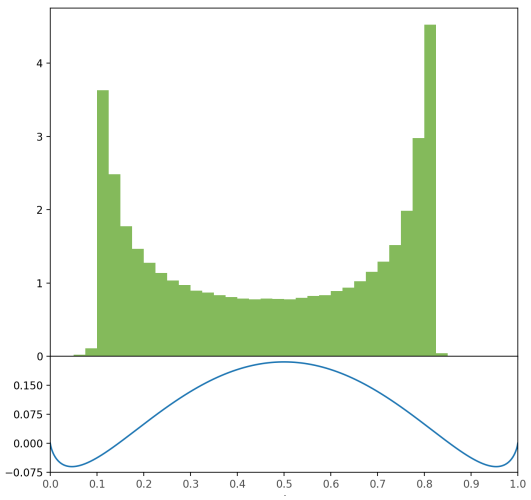


Figura 3.2: Histograma de concentraciones del sistema binario de la Fig. 3.1. Abajo se graficó el potencial utilizado para ese sistema (es del mismo tipo que el usado en este proyecto).

Un histograma es una gráfica de barras de las frecuencias encontradas, en una muestra, de los valores de alguna cantidad dentro de ciertos intervalos. Para que una gráfica de barras pueda ser considerada un histograma los intervalos deben estar ordenados y no deben tener huecos. Un histograma de las concentraciones que se tienen en el sistema de la Fig. 3.1 se muestra en la Figura 3.2.

Gracias al histograma obervamos que sistema de las Figs. 3.1–2 aún no está en un estado de equilibrio, donde observaríamos que los bultos tienen la concentración de los pozos del potencial.

3.1.2. Sistemas Ternarios

Nosotros deseamos reportar, para el caso ternario, resultados muy similares a los que se mencionaron en la Sección anterior. El problema que tenemos es que para nuestro sistema ternario es necesario especificar dos parámetros para poder conocer la concentración del sistema. Para esto se recurre a las gráficas conocidas como **diagramas ternarios**.

En términos matemáticos (de sistemas de coordenadas), los diagramas ternarios son gráficas triangulares de las proporciones entre tres variables cuya suma es constante, que usan coordenadas baricéntricas[1]. (Stover, 2019).

La Figura 3.3 muestra un ejemplo de diagrama ternario (vacío), los ejes indican la concentración de cada componente. Las líneas rectas paralelas a los lados del triángulo corresponden a valores constantes de concentración (que en este caso son las coordenadas baricéntricas) y están numerados en sentido antihorario.

En realidad, ya se utilizaron diagramas ternarios para graficar el término no interfacial de la densidad de energía libre en la Fig. 1.1.

[1] Estas coordenadas se obtienen, para algún punto P al interior del triángulo, colocando masas ficticias m_i en las esquinas cuyos valores estén ajustados para que P sea el centro de masa (baricentro) del arreglo. A cada esquina E_i se le asocia una coordenada ϕ_i cuyo valor será la fracción de la "masa" total que representa la masa ficticia m_i de su esquina. Un corolario de esta definición es que la fracción ϕ_i es proporcional al área formada por el triángulo $\triangle PE_jE_k$ (con $j,k \neq i$) opuesto a su esquina; esto implica que las líneas paralelas a $\overline{E_jE_k}$ corresponden a ϕ_i constante (la base del triángulo es $\|\overline{E_jE_k}\|$ y su altura es la distancia a la paralela, ambas son constantes), como se observa en la Fig. 3.3.

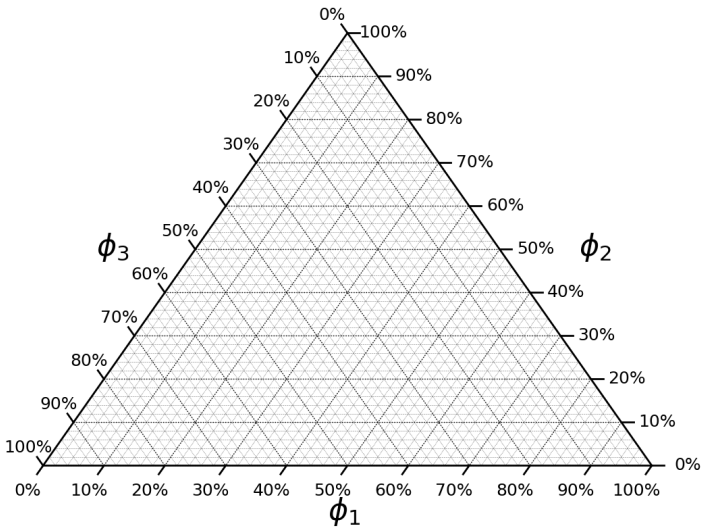


Figura 3.3: Diagrama ternario genérico [2].

Los datos en bruto que se obtienen de la simulación son los valores de los campos de fase en cada punto de la malla numérica. Escoger valores para estos campos de fase es equivalente a seleccionar un punto sobre el diagrama ternario. Para conocer la concentración de los puntos de la malla con una imágen, de manera análoga a lo que se muestra en la Fig. 3.1, se recurre a asignar colores en escala RGB a cada una de las concentraciones. En particular se usó el mapa

$$\phi_i \mapsto \lfloor 255\phi_i \rfloor \tag{3.1}$$

Este mapa se muestra gráficamente en la Figura 3.4. El diagrama mostrado ahí será el equivalente ternario de la barra de colores del lado derecho de la Fig. 3.1.

[2]Matemáticamente, representa al espacio $\Delta^2 = \left\{\boldsymbol{\phi} \in [0,1]^3 \middle| \sum_i \phi_i = 1\right\}$

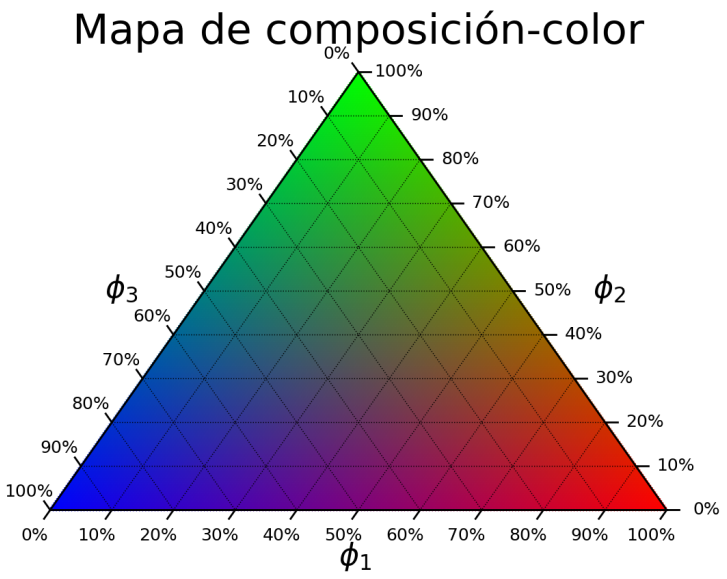


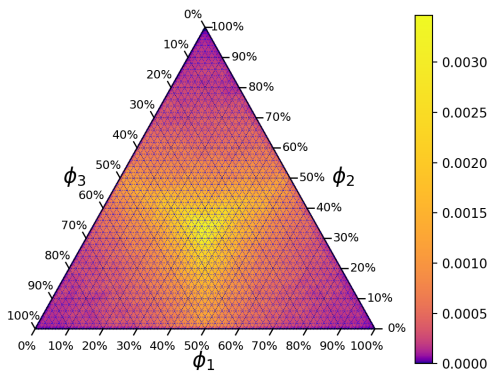
Figura 3.4: Diagrama ternario del mapa de concentraciones a color, utilizado para generar imágenes del estado del sistema.

También se adaptó el histograma de concentraciones para el caso ternario. En este caso se graficaron las "barras" directamente encima del diagrama ternario. Para los "intervalos" de cada barra se dividió el diagrama ternario en triángulos (como los que dividen al diagrama ternario de la Fig. 3.3) y se le asignó una barra a cada uno. Para dar la altura de las barras se utilizó una barra de color.

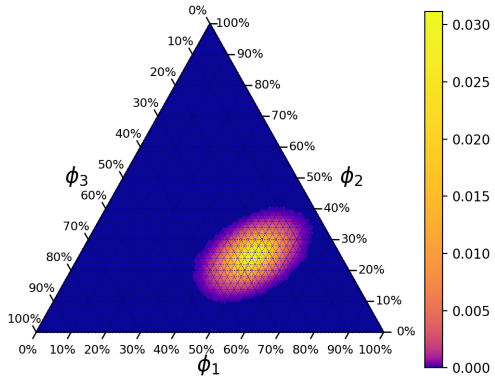
Los triángulos en cuestión se definieron dividiendo el diagrama ternario en intervalos de cierto porcentaje[3] y se hizo un conteo de los nodos de la malla con

[3]El porcentaje que se utlizó al final fue obtenido por prueba y error. Un intervalo muy grande no permitiría apreciar los detalles de la distribución, mientras que un intervalo muy pequeño tendría mucho ruido.

concentraciones dentro de cada área. El valor obtenido se normalizó divididendo entre el número total de nodos de la malla para obtener la *distribución* de los puntos sobre el diagrama ternario. Ejemplos de este tipo de gráficas se muestran en la Figura 3.5, estos ejemplos se crearon generando números aleatorios en una malla de 384×384 (el mismo tamaño de malla que se usó en las simulaciones) y con la misma rutina utilizada en las simulaciones.



(a) Histograma para la distribución $\Phi_i = {}^{X_i}/_{(X_1+X_2+X_3)}$ donde X_i tienen una función de densidad de probabilidad uniforme $f_{X_i}(x_i) = 1, x_i \in [0,1)$



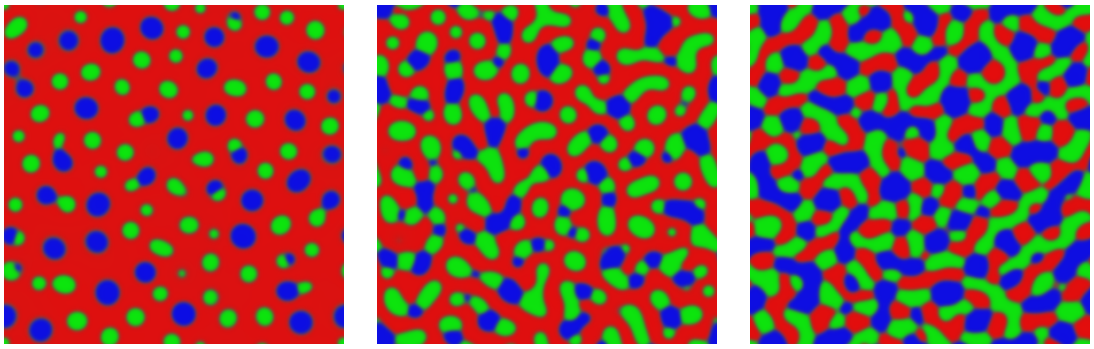
(b) Histograma para una distribución normal multivariante sin correlación en ϕ_1, ϕ_2 con $\mu_1 = {}^1/_2, \mu_2 = {}^1/_4$ y $\sigma_1 = \sigma_2 = {}^9/_{20}$. ϕ_3 se obtiene por cerradura.

Figura 3.5: Histogramas creados con Python mediante un generador de números pseudo-aleatorios.

Las demás gráficas utilizadas fueron relativamente sencillas. Se graficaron los valores máximos y mínimos de cada campo de fase a lo largo de la simulación; se calculó y graficó el funcional de energía libre en cada paso de tiempo; y en una de las pruebas se graficaron los campos de fase sobre una línea de corte (para observar la evolución del ancho de la interfase).

3.2. Descomposición de una Mezcla Homogénea

La separación de las fases se puede dar como descomposición espinodal o en forma metaestable[4], los casos se observan en la Figura 3.6. Se realizaron 36 simulaciones, haciendo un barrido sobre el diagrama ternario para las concentraciones iniciales.



(a) Descomposición metaestable. La composición inicial en cada punto fue $\phi_0 = (2/3 \pm 0.05, 1/6 \pm 0.05, 1/6 \pm 0.05)$

(b) Descomposicón espinodal entre una y dos fases. La composición inicial puntual fue $\phi_0 = (0.5 \pm 0.05, 0.3 \pm 0.05, 0.2 \pm 0.05)$

(c) Descomposición espinodal entre las tres fases, con composición inicial de $\phi_0 = (1/3 \pm 0.05, 1/3 \pm 0.05, 1/3 \pm 0.05)$

Figura 3.6: Resultados de tres casos posibles de la descomposición de una mezcla inicialmente homogénea, con los mismos parámetros y en el mismo paso de tiempo.

Las concentraciones iniciales que se utilizaron en las simulaciones se observan en la Figura 3.7. Se excluyeron los puntos con alguna de las $\phi_i = 0$ por que estos se reducen a sistemas binarios.

[4]La distinción entre estos modos es que la descomposición metaestable se da por nucleación y crecimiento mientras que la espinodal se da de manera homogénea en todo el dominio.

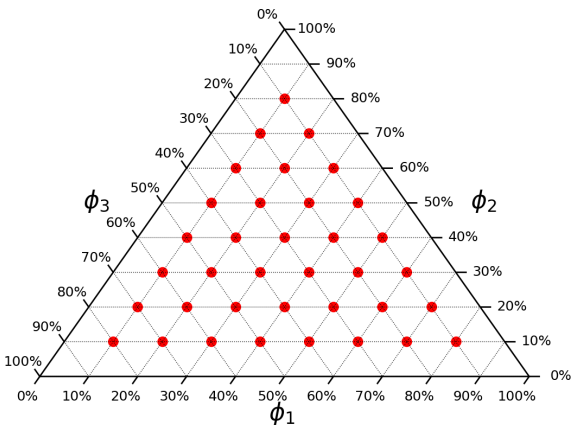


Figura 3.7: Se muestran en el diagrama ternario los puntos de composición inicial utilizados en las simulaciones de descomposición.

En principio, la dinámica del sistema debería ser la misma bajo el intercambio[5] entre las ϕs debido a que el término F_{log} no distingue entre los campos de fase y la energía interfacial $\kappa/2\|\nabla\phi_i\|$ tiene el mismo coeficiente para todas las ϕs. Sin embargo, se simularon todos los puntos de la Fig. 3.7 porque las ecuaciones en diferencias no son simétricas en este sentido: el método de complemento de Schur resuelve de manera distinta a ϕ_3 y a las ecuaciones de $\phi_{1,2}$ se les sumó un *cero formal* (el término de la partición de la Sección 1.3.1, es cero en el límite $\Delta t \to 0$) que para la discretización no es exactamente cero. Esto permitirá validar si la simulación es adecuada o si hay una discrepancia debido a una discretización burda.

La situación descrita en el párrafo anterior implicó que varias simulaciones sean redundantes en los casos en los que no hubo discrepancias apreciables. Los resultados que se reportan en las secciones siguientes sólo incluyen un representante de los sistemas en estos casos.

[5] Formalmente, las ecuaciones son simétricas ante la acción del grupo de permutaciones $\text{Sym}(\Phi)$ donde $\Phi = \{\phi_1, \phi_2, \phi_3\}$.

3.2.1. Parámetros Numéricos

En estas simulaciones únicamente se observó el efecto de cambiar la concentración inicial promedio de la mezcla. Por esto, los demás parámetros se mantuvieron constantes. Sus valores se resumen en la Tabla 3.1

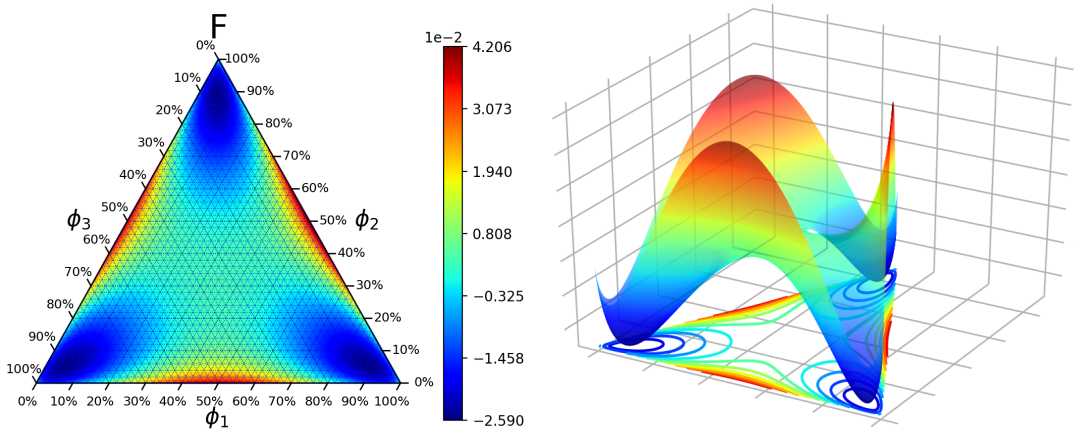
Tabla 3.1: Valores de los parámetros que se tomaron para las simulaciones del caso 1 ($\kappa = 0.125$) de descomposición de mezclas.

κ	θ	θ_c	c	α	β	Δx	Δt	N	T
1	0.3	1	4	1	0	1	10	384	5000

En la malla se escogió: $\Delta x = 1$, por ser una escala adecuada para la longitud característica del sistema (con estos parámetros); $\Delta t = 10$, para reducir el tiempo de cómputo para aproximarse al equilibrio; $N = 384$, para tener un desempeño óptimo de la FFTW; $T = 5000$, como un tiempo suficiente para que el sistema se acerque al equilibrio. Para la partición de (1.18) se utilizó $c = 4$ para asegurar una solución no-trivial y para (α, β) se usaron los valores predeterminados[6] $(1, 0)$ del estudio de Tavakoli, 2016.

El potencial de estos sistemas, como se mencionó anteriormente, es simétrico y se puede observar su gráfica en la Figura 3.8.

[6]Tiene sentido usar $\beta = 0$ porque la energía interfacial, que es el término que involucra a $\partial\phi_i/\partial x_j$, ya es convexa. Esto hace que el único coeficiente relevante de la partición sea $c\alpha$, dado que se está variando c se deja $\alpha = 1$.



(a) Mapa de calor del potencial. **(b)** Gráfica tridimensional del potencial.

Figura 3.8: Potencial utilizado en las simulaciones de decomposición. Ambas gráficas usan el mismo mapa de color, en la barra de color de (a) se indica la magnitud (el "1e-2" es un factor de 10^{-2}, *i.e.*, el máximo es 0.0421 y el mínimo es −0.0259).

Haciendo nuevamente un paralelismo con el caso binario, ahí se sabe (Pego, 1989) que se puede determinar si un estado presentará descomposición, y de qué tipo, a partir de la derivada del potencial F, como se observa en la Figura 3.9.

Generalizar este criterio al caso ternario no es trivial ya que no se sabe, *a priori*, cómo influirá un tercer componente en la descomposicón de los otros. Además, se tienen tres posibles derivadas parciales de F con las cuales se tendría que dividir el diagrama ternario en regiones similares a las de la Fig. 3.9 y, como se observa en la referencia, las concentraciones de equilibrio no coinciden necesariamente con los mínimos del potencial, incluso para el caso binario.

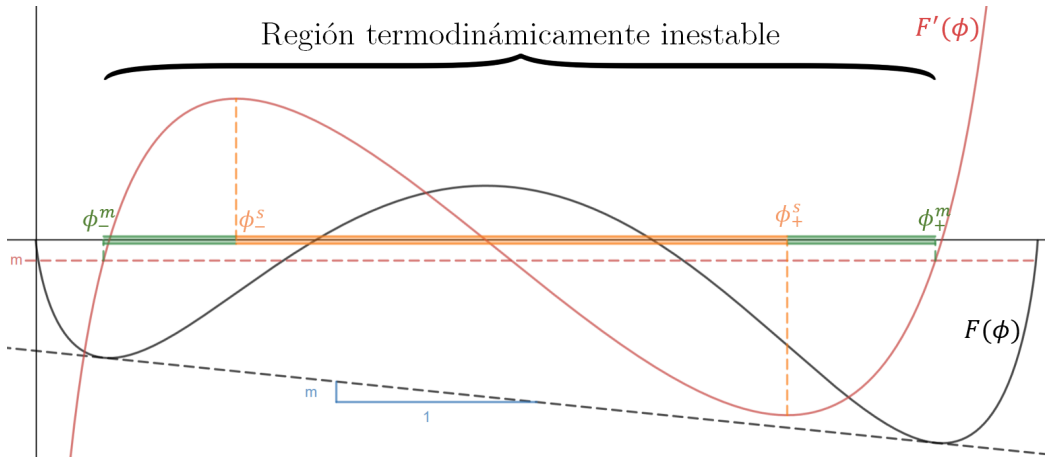


Figura 3.9: En negro, el potencial $F(\phi)$ y una tangente en dos de sus puntos (cuyas concentraciones $\phi_\pm^m$ corresponden al bulto de las fases). Para que haya descomposición de cualquier tipo, el sistema debe estar entre las concentraciones con $F'(\phi) = \mathrm{m}$, la pendiente de la tangente. A los valores externos se les denomina $\phi_\pm^m$. Las abscisas de los puntos críticos de $F'(\phi)$ se denotan por $\phi_\pm^s$. Entre $\phi_\pm^m$ y $\phi_\pm^s$ se presenta descomposición metaestable, mientras que entre ϕ_-^s y ϕ_+^s hay descomposición espinodal. (Pego, 1989)

Debido a estas complicaciones, no se proporcionará un criterio teórico como el de Pego, 1989, sino que se observará lo que ocurre para cada uno de los casos analizados.

3.2.2. Simulaciones de Descomposición

Se reportan aquí los resultados obtenidos de las simulaciones de decomposición. Para emular fluctuaciones térmicas[7] se introdujo una variación (pseudo-)aleatoria a la condición inicial de ±0.05 a los campos de fase, en cada punto[8]. Las simulaciónes de concentraciones con valores iniciales permutados son muy similares por lo que se agruparon en lo que se denominó como *familias*.

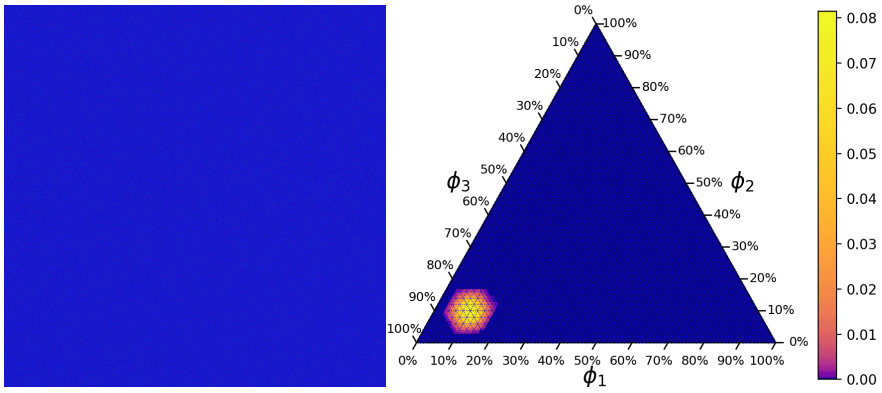
Familia 8:1:1

Para esta familia se muestra la condición $\overline{\phi_0} \approx (0.1, 0.1, 0.8)$ no se observa separación de fases, únicamente se homogeneizan gradualmente las fluctuaciones iniciales. La evolución del sistema se observa en la Figura 3.10 (se denota por $\tau := t/\Delta t$ al tiempo adimensional que corresponde al paso de simulación).

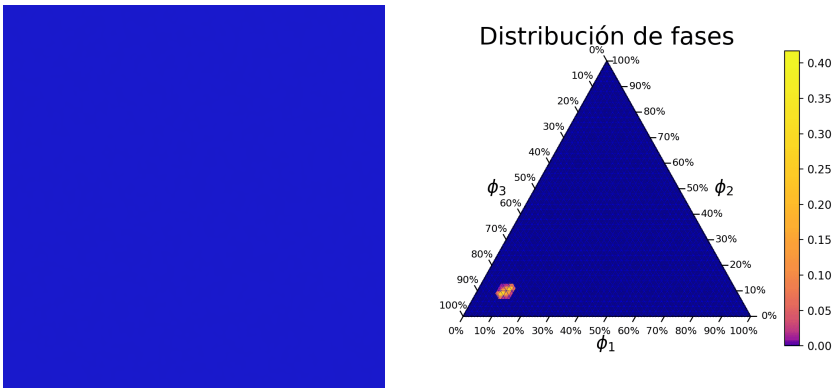
Los estados del sistema se ven muy similares y resultan más ilustrativos los diagramas ternarios. En 3.10(a) se observa la condición inicial con fluctuaciones aleatorias de ±5%. Al inicio, éstas se reducen rápidamente y luego se homogeneiza gradualmente la mezcla hasta que todos los puntos se encuentran en el mismo intervalo de concentraciones del histograma, como se observa en 3.10(f).

[7] Estas fluctuaciones son necesarias para que se formen las fases del sistema. Si se inicia la simulación con una concentración perfectamente homogénea no se daría la decomposición porque no habría sitios preferentes para formar fases y ninguna variable del modelo que permite que se formen "espontáneamente". En otras términos, la condición homogénea es una solución trivial a las ecuaciones de Cahn–Morral, por que el lado derecho de ellas es cero.

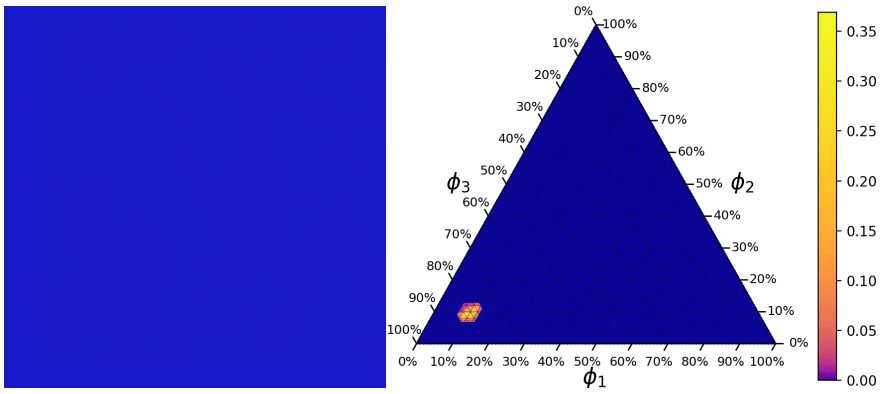
[8] Esto implica una desviación en valor promedio, del orden de $\Delta\phi/\sqrt{N_{\text{Total}}} = 0.05/384 \approx 0.0001$.



(a) Condición inicial

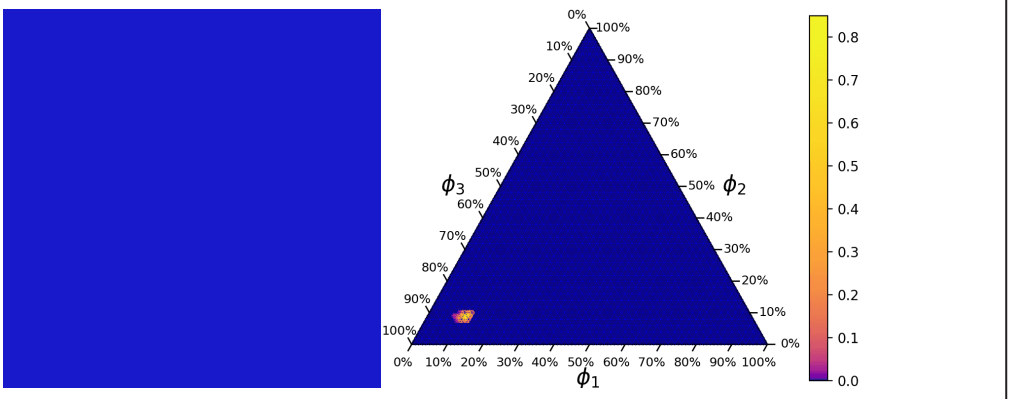


(b) $\tau = 10$

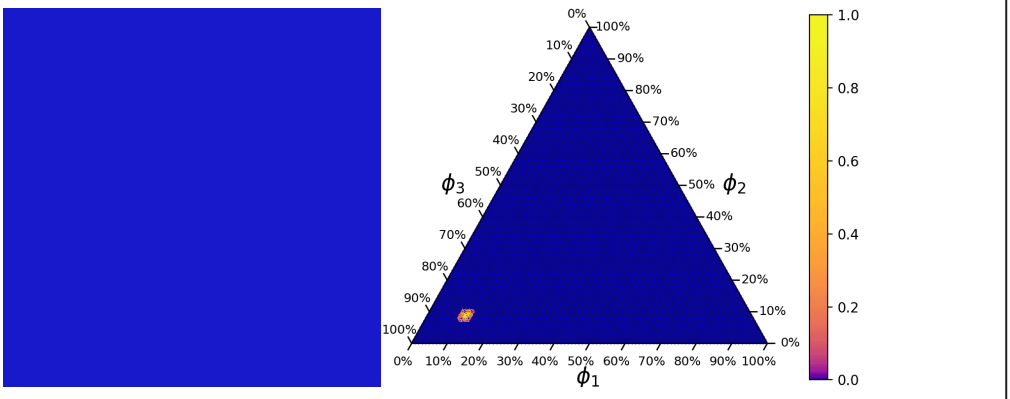


(c) $\tau = 40$

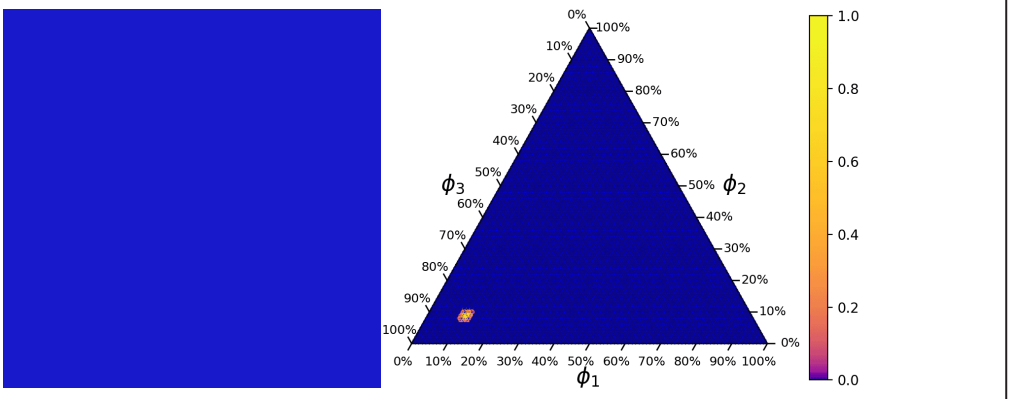
Figura 3.10: Estado del sistema y distribución de las concentraciones durante el transcurso de la simulación. (Continúa en la página siguiente).



(d) $\tau = 665$



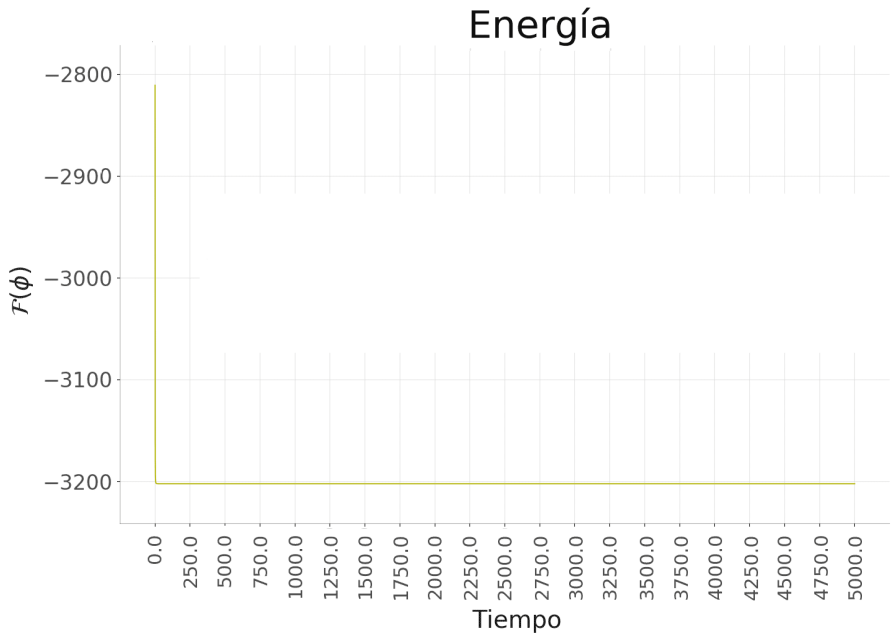
(e) $\tau = 1211$



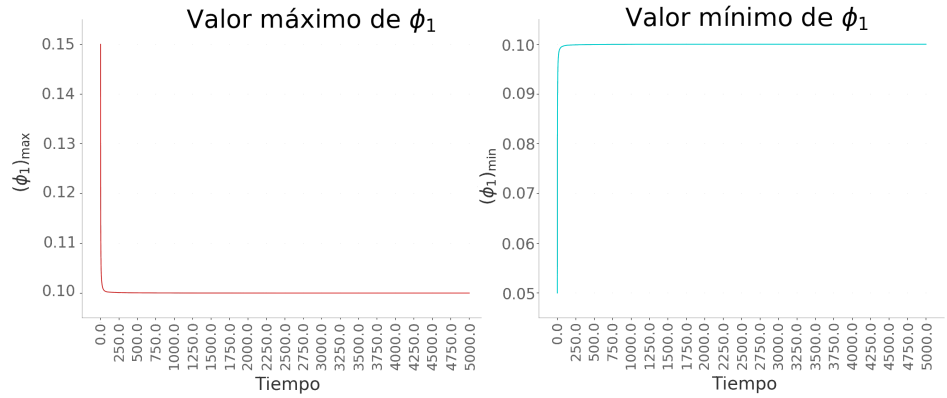
(f) $\tau = 5000$

Figura 3.10: (Continuación) El valor de 1 en el histograma para $\tau = 5000$ indica que todos los puntos de la malla se encuentran en el mismo intervalo de concentraciones.

Se crearon gráficas del potencial de Helmholtz (1.9) y, los máximos y mínimos de cada campo de fase durante la simulación. Estos se muestran en la Figura 3.11

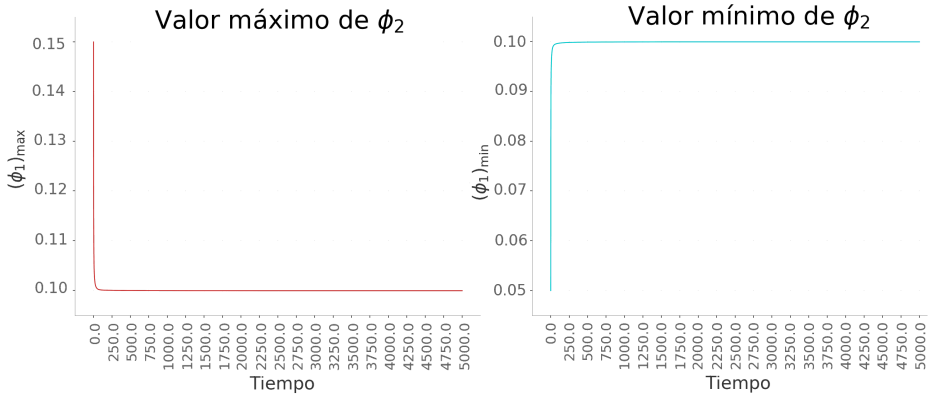


(a) Energía libre de Helmholtz

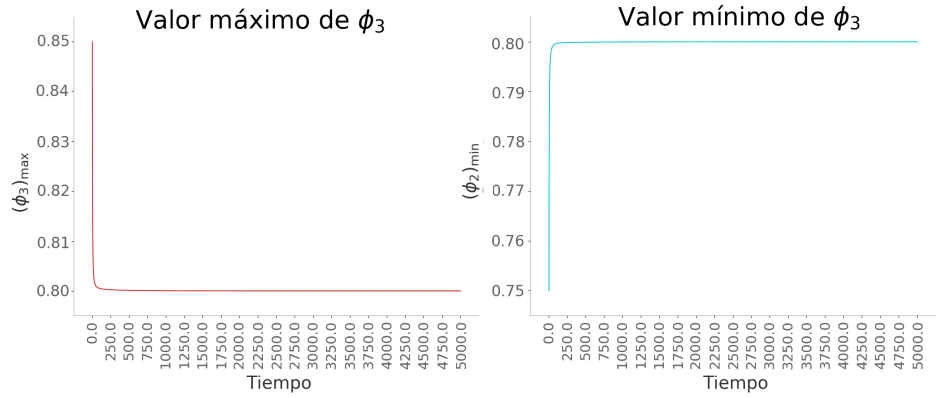


(b) Máximos y mínimos de ϕ_1

Figura 3.11: Energía, máximos y mínimos durante el transcurso de la simulación. Se observa como el sistema se estabiliza rápidamente.



(c) Máximos y mínimos de ϕ_2



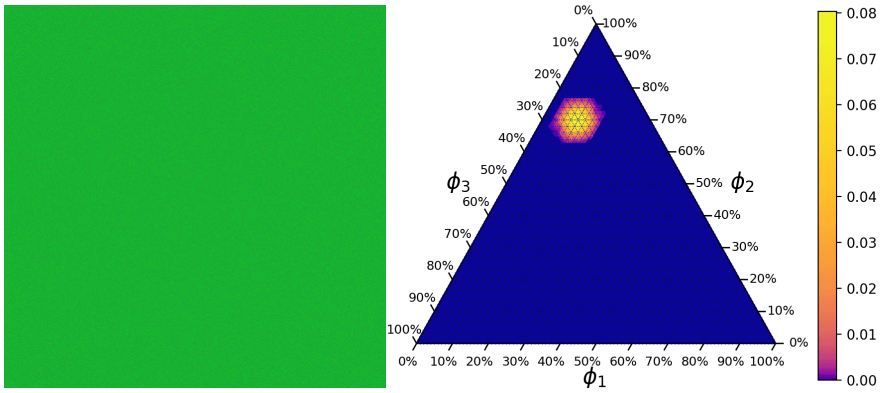
(d) Máximos y mínimos de ϕ_3

Las Figs. 3.11(b–d) muestran claramente que la evolución del sistema consistió unicamente en eliminar las fluctuaciones. Tanto los máximos como los mínimos convergen al valor inicial promedio.

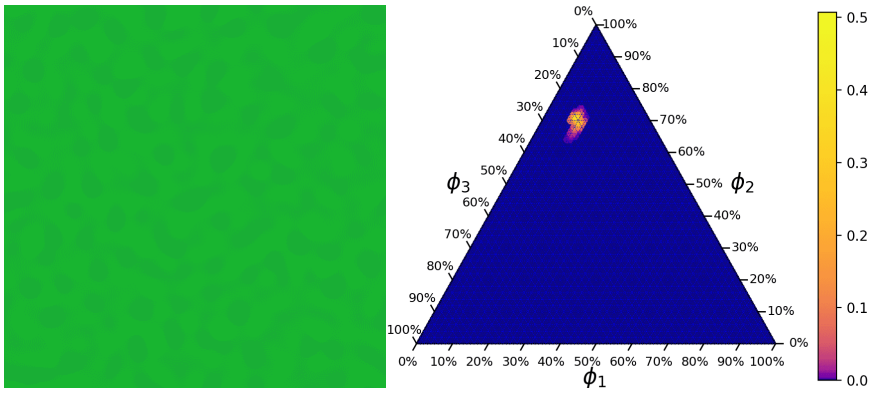
Los casos "permutados", con $\overline{\phi_0} \approx (0.1, 0.8, 0.1), (0.1, 0.1, 0.8)$, de la familia 8:1:1 tampoco se separaron en fases. Por ser muy similares a éste, se omitirán.

Familia 7:2:1

Para todos los casos de esta familia se encontró una decomposición de tipo metaestable. A continuación se presenta el caso con concentración inicial promedio $\overline{\phi_0} \approx (0.1, 0.7, 0.2)$, en la Figura 3.12.

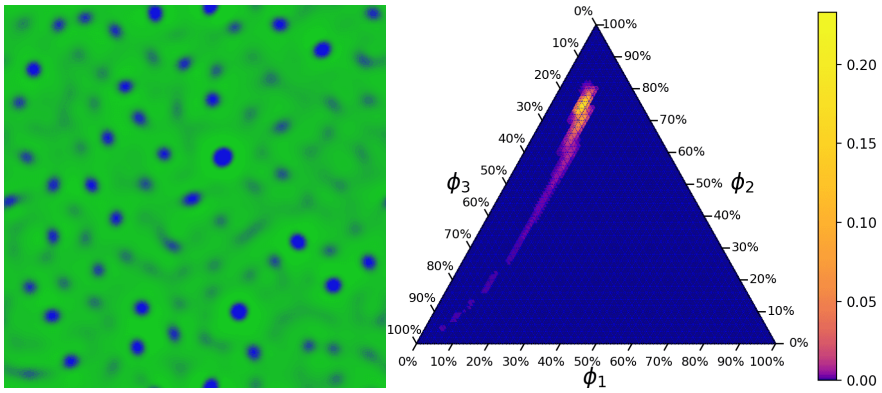


(a) Condición inicial

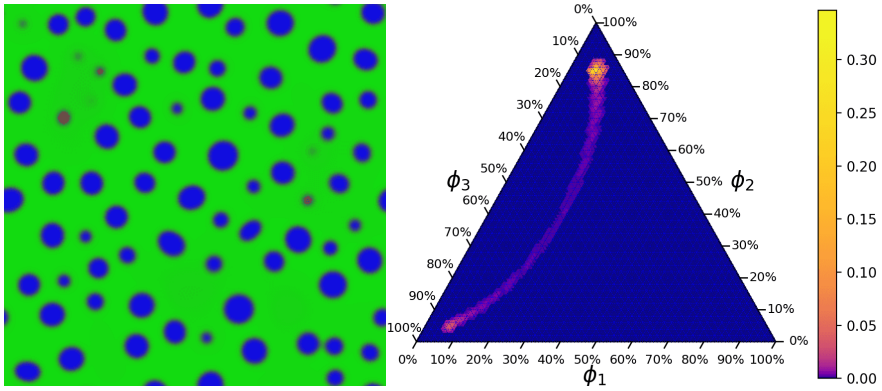


(b) $\tau = 206$

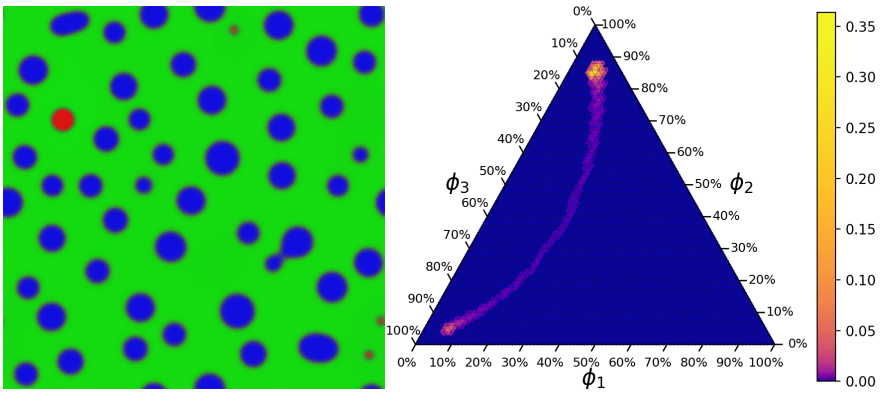
Figura 3.12: Estado y distribución de concentraciones para $\overline{\phi_0} \approx (0.1, 0.7, 0.2)$. En (a) se observa la condición inicial y en (b) se comienzan a formar sitios con mayor concentracón de ϕ_3, dónde se puede dar nucleación. (Continúa en la página siguiente).



(c) $\tau = 318$

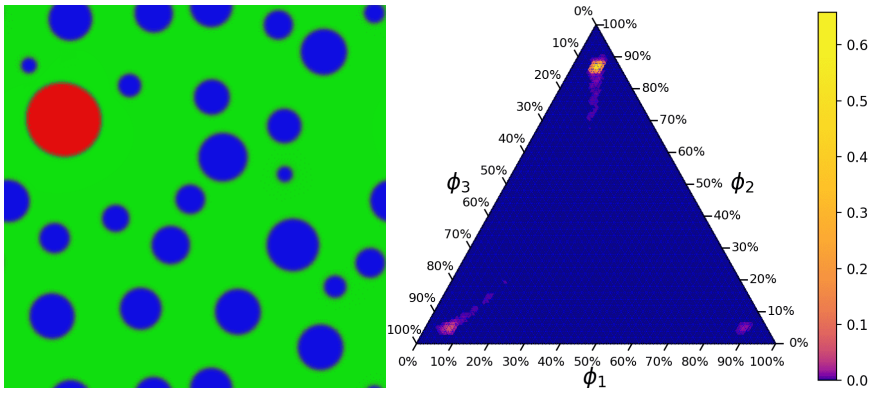


(d) $\tau = 523$



(e) $\tau = 1260$

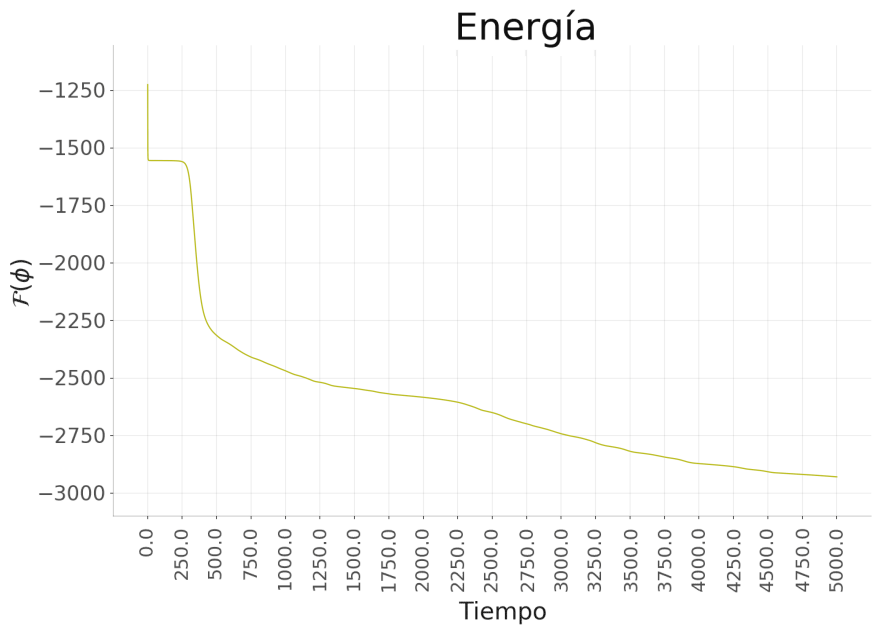
Figura 3.12: (Continuación) En (c) se observa la formación de la fase asociada a ϕ_3 por nucleación, en (d) ocurre lo mismo pero para la fase de ϕ_1. En (e) se aprecia el crecimiento de dicha fase.



(f) $\tau = 5000$

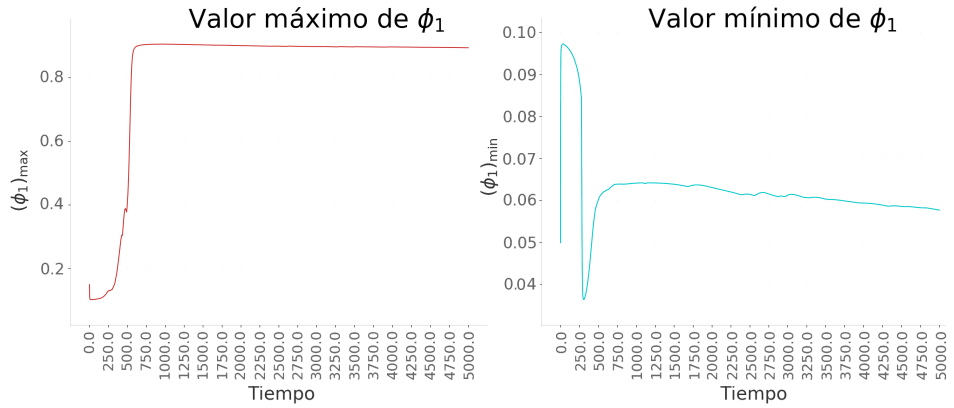
Figura 3.12: (Continuación) En (f) se tiene la condición final.

La energía libre de este sistema y los máximos y mínimos de cada campo de fase se muestran en la Figura 3.13

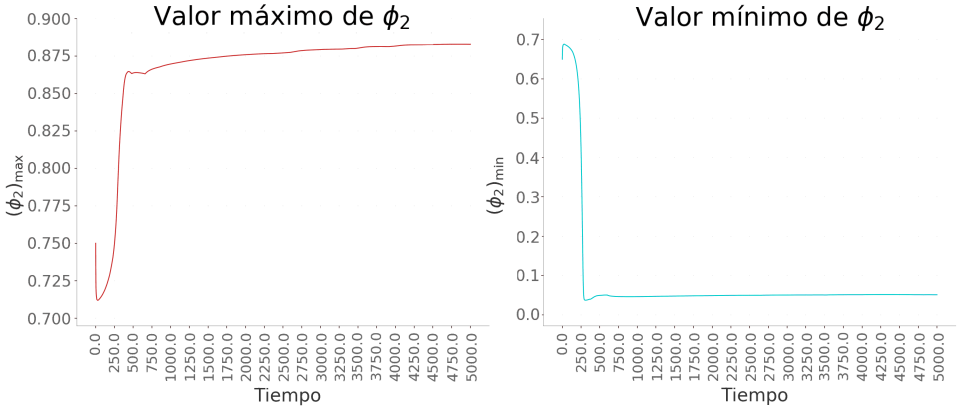


(a) Energía libre de Helmholtz

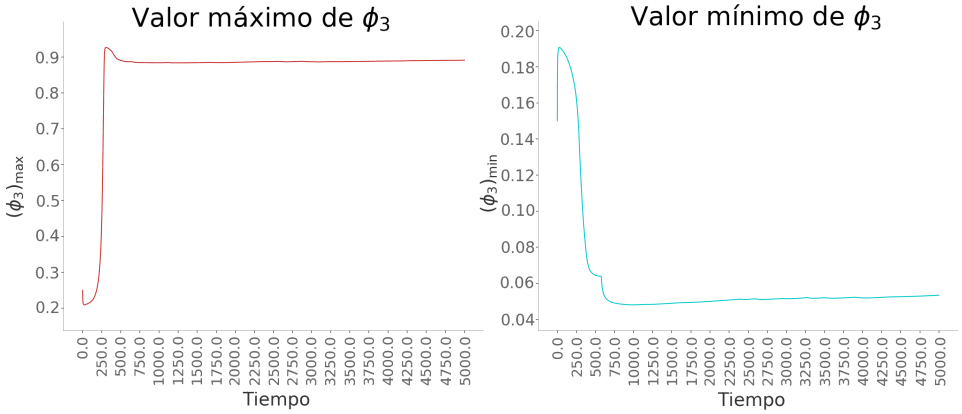
Figura 3.13: Energía, máximos y mínimos durante el transcurso de la simulación.



(b) Máximos y mínimos de ϕ_1



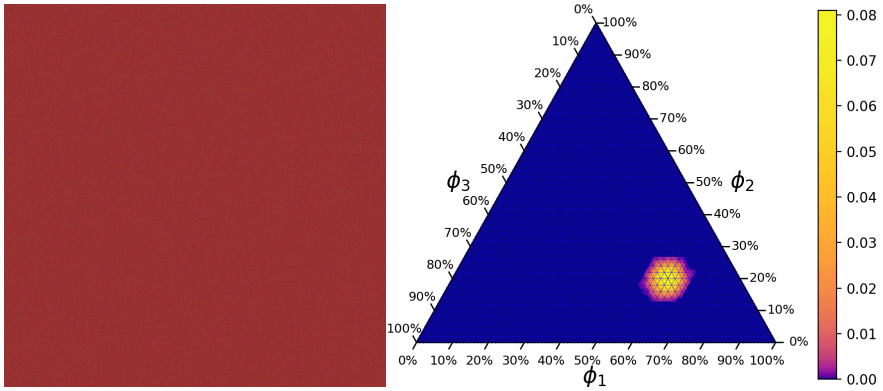
(c) Máximos y mínimos de ϕ_2



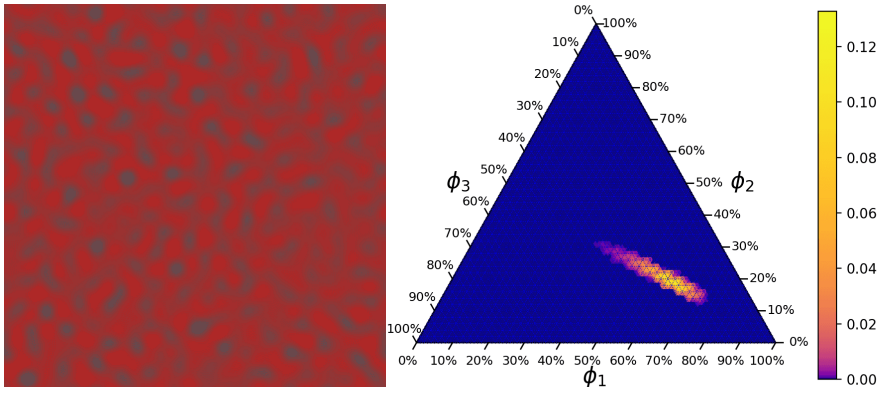
(d) Máximos y mínimos de ϕ_3

Familia 6:2:2

En estos sistemas se obtuvo una descomposición de tipo espinodal entre 1 y 2 componentes como la que se observa en la Fig. 3.6(b). La evolución del caso $\overline{\phi_0} \approx (0.6, 0.2, 0.2)$ se muestra en la Figura 3.14.

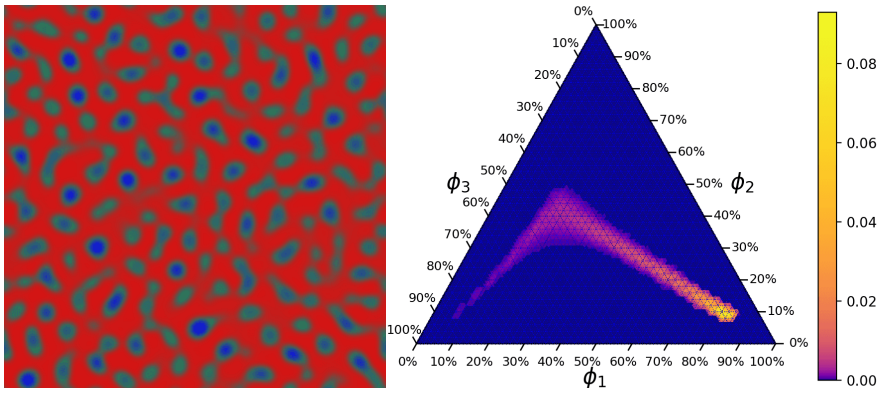


(a) Condición inicial

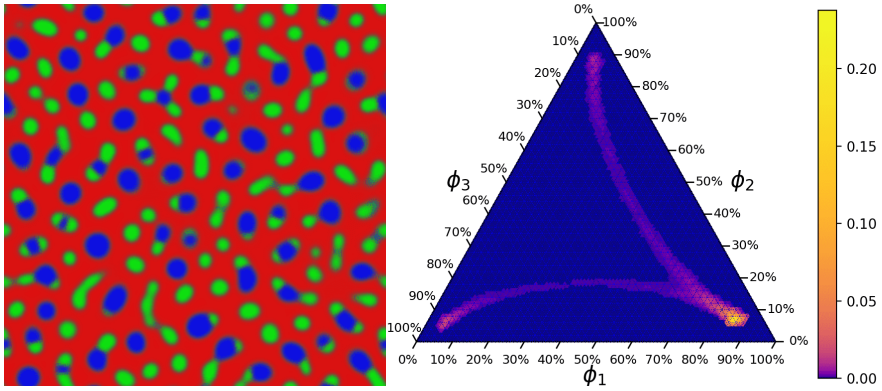


(b) $\tau = 184$

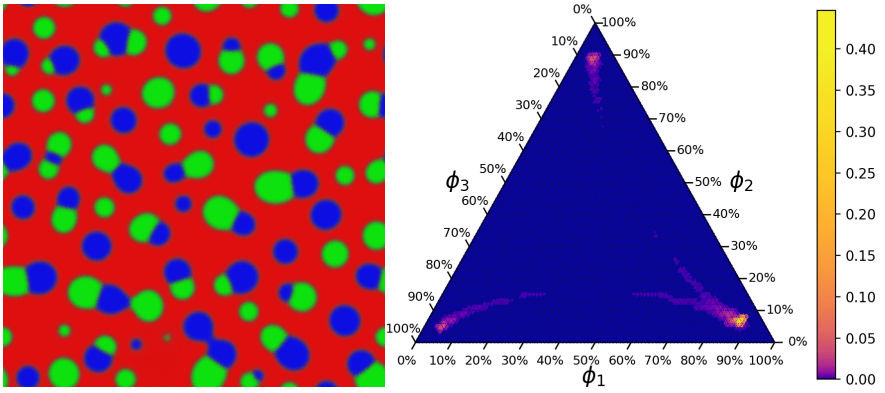
Figura 3.14: Evolución del sistema con concentración inicial $\overline{\phi_0} \approx (0.6, 0.2, 0.2)$. En (b) se observa la separación entre la fase de ϕ_1 y una mezcla de $\phi_{2,3}$. (Continúa en la página siguiente).



(c) $\tau = 291$

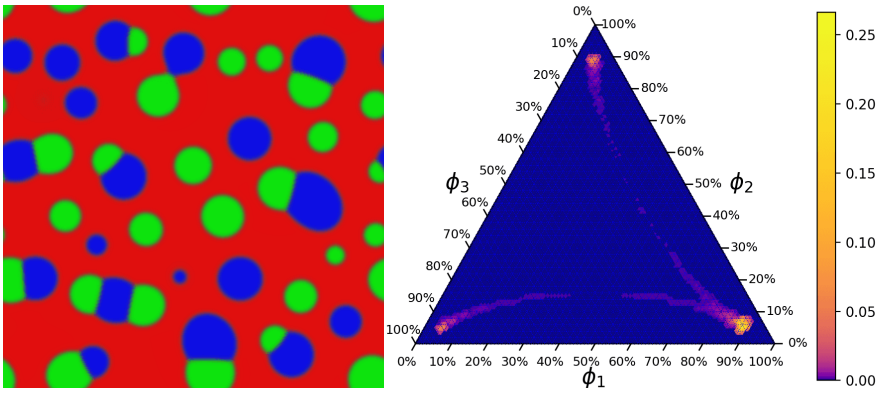


(d) $\tau = 410$



(e) $\tau = 1865$

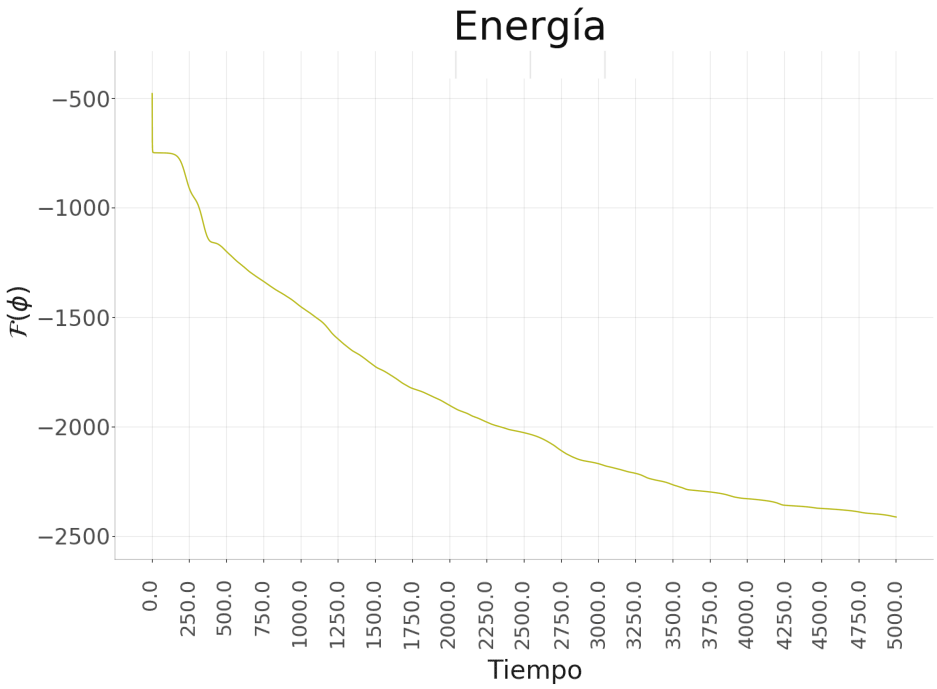
Figura 3.14: (Continuación) En (c) y (d) se observa el proceso de separación de las fases asociadas a ϕ_2 y ϕ_3 y su crecimiento en (e). (Continúa en la página siguiente).



(f) $\tau = 5000$

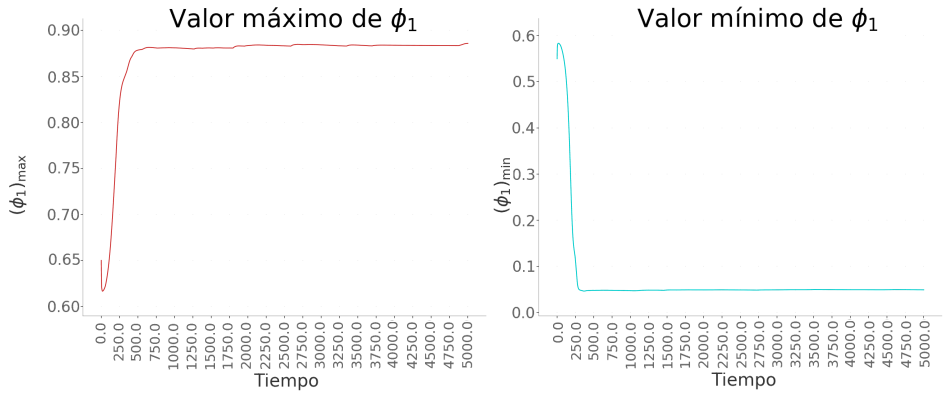
Figura 3.14: (Continuación) En (f) está la condición final.

Las gráficas de cantidades globales del sistema se muestran en la Figura 3.15.

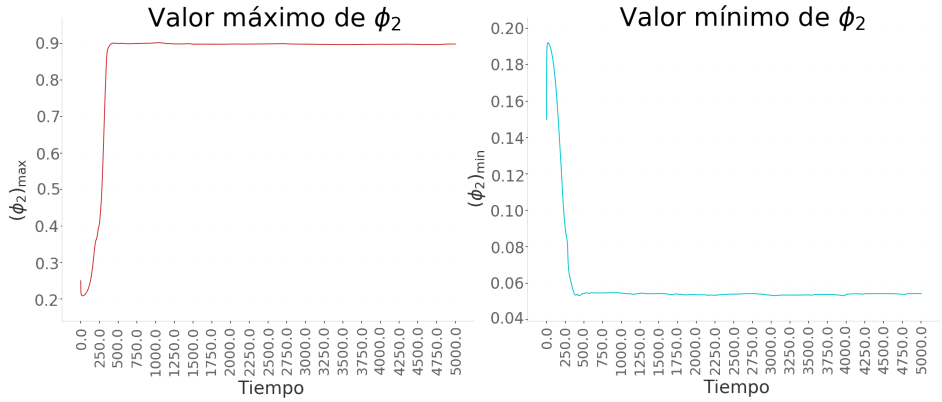


(a) Energía libre de Helmholtz

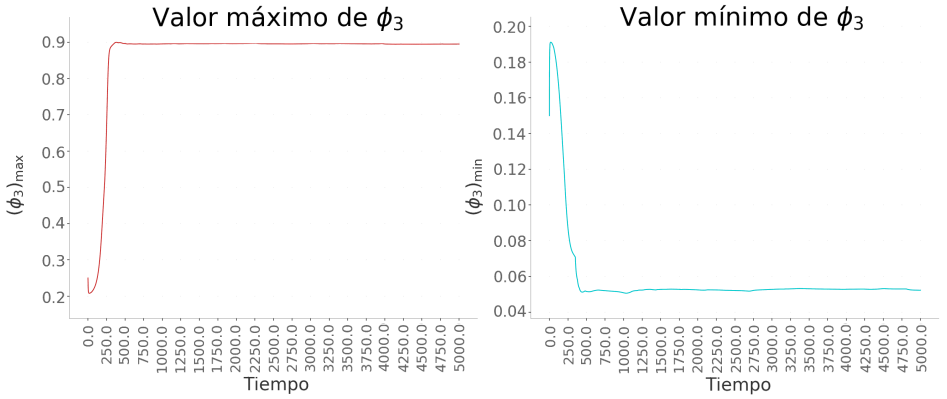
Figura 3.15: Energía, máximos y mínimos durante el transcurso de la simulación.



(b) Máximos y mínimos de ϕ_1

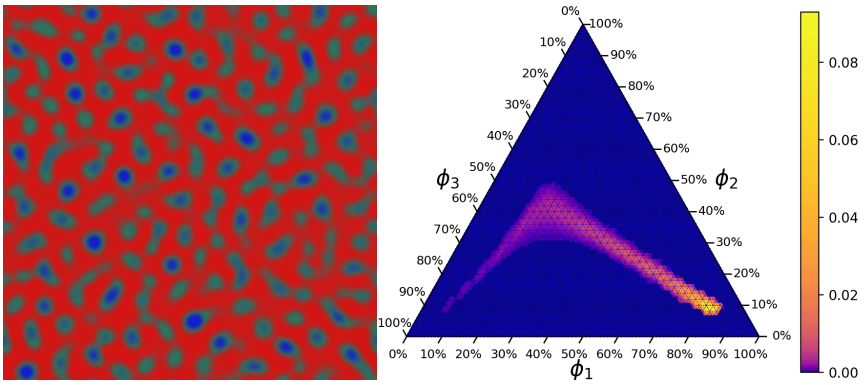


(c) Máximos y mínimos de ϕ_2

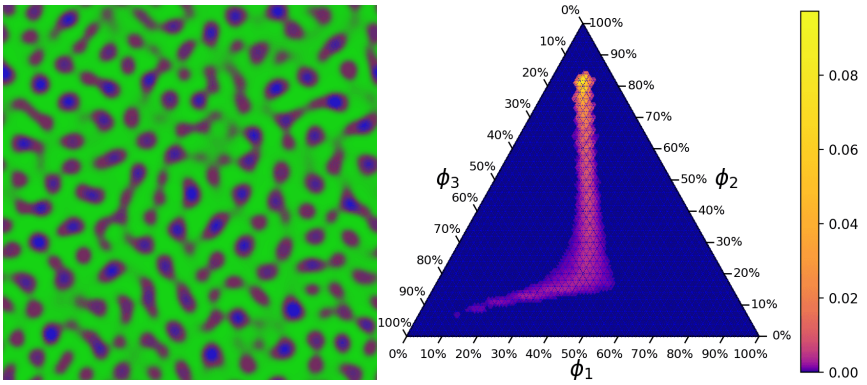


(d) Máximos y mínimos de ϕ_3

Como se mencionó al inicio de esta Sección, los casos con concentraciones iniciales permutadas tienen que ser equivalentes para una solución exacta. Sin embargo en este caso se observó cierta distinción, en particular el componente ϕ_3 se separa más rápidamente que los otros dos, como se ve en la Figura 3.16 [9].



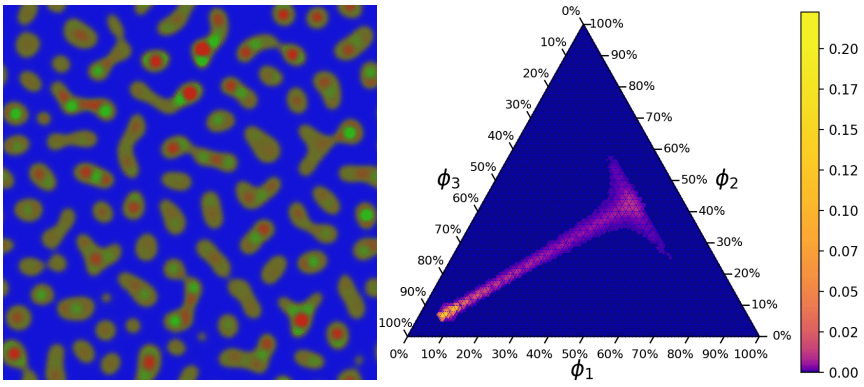
(a) Caso con $\overline{\phi_0} \approx (0.6, 0.2, 0.2)$ en el paso $\tau = 291$.



(b) Caso con $\overline{\phi_0} \approx (0.2, 0.6, 0.2)$ en el paso $\tau = 291$.

Figura 3.16: Hay descomposición espinodal en los tres casos. (Continúa en la página siguiente).

[9] Viendo la Figura se puede concluir que la fase asociada a ϕ_3 se separa más rápidamente porque en 3.16(c) se observa que la formación de las fases asociadas a ϕ_1 y ϕ_2 se dan a un tiempo posterior que su contraparte en 3.16(a) y 3.16(b), además que en estas últimas dos es notorio que en la "fase" mezclada (en tonos más grises) se presenta primero ϕ_3.

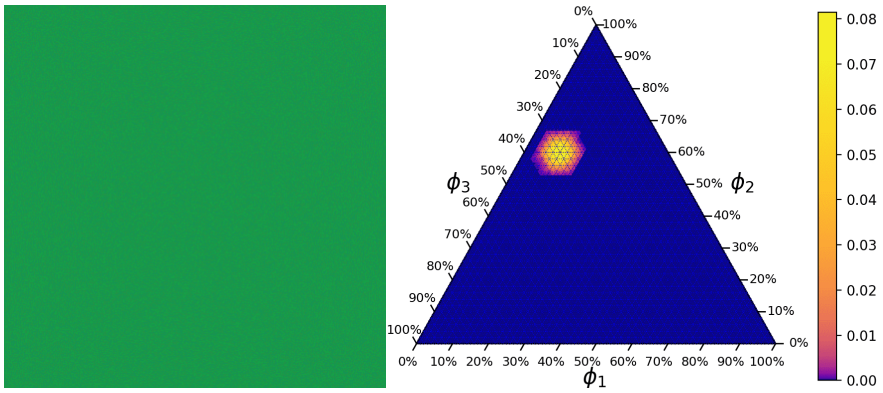


(c) Caso con $\overline{\phi_0} \approx (0.2, 0.2, 0.6)$ en el paso $\tau = 377$.

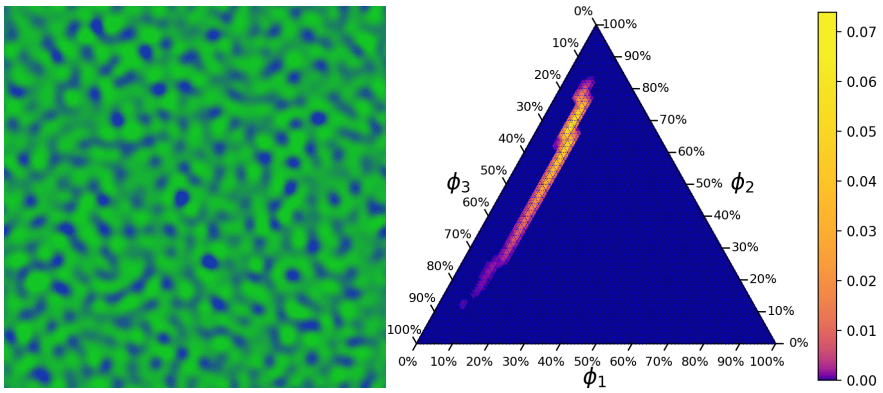
Figura 3.16: (Continuación) En (a) y (b) se observa una separación preferente hacia ϕ_3 en contraste con (c), donde las fases menos abundantes se forman simetricamente.

Familia 6:3:1

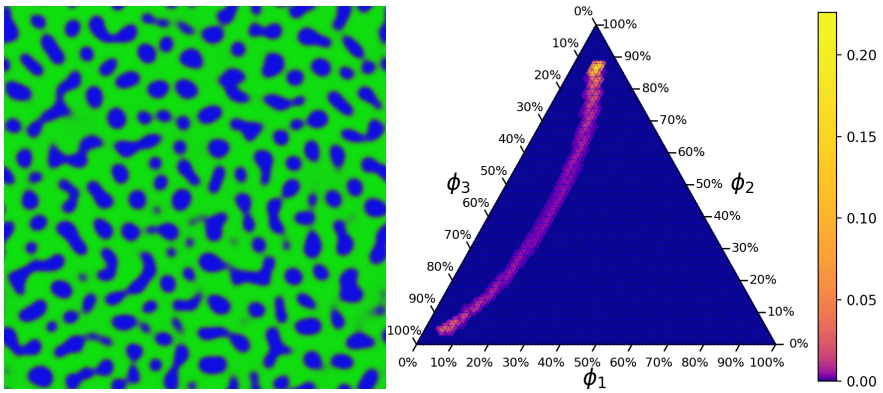
La *familia 6:3:1* presentó un caso distinto a los mencionados en la Figura 3.6, se observó una separación espinodal entre dos fases (ϕ_2 y ϕ_3 en la Figura 3.17) y posteriormente se formó un dominio de la fase asociada a la tercera fase por nucleación. Algo notorio de este resultado es que la formación del dominio rojo se dió en un tiempo característico considerablemente más largo en simulación, a comparación de la separación en los demás casos analizados hasta ahora. En la Fig. 3.17 se muestran los resultados para el caso de $\overline{\phi_0} \approx (0.1, 0.6, 0.3)$, la fase asociada a ϕ_1 es la que se forma posteriormente por nucleación.



(a) Condición inicial

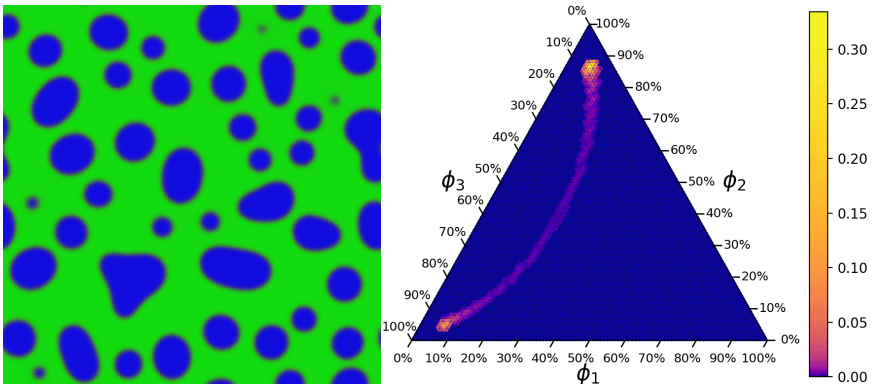


(b) $\tau = 110$

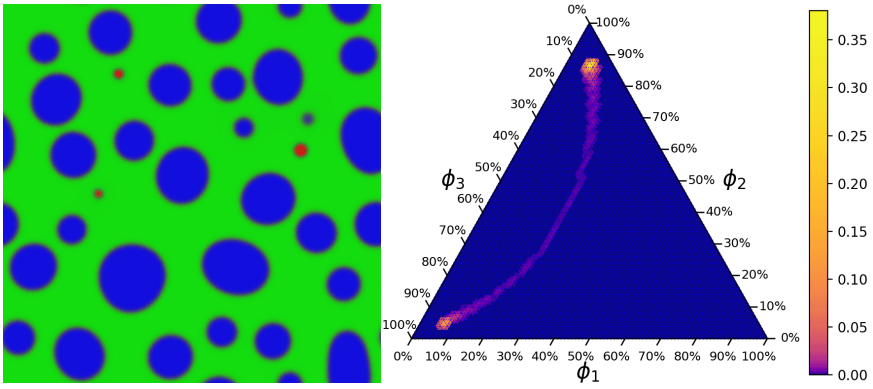


(c) $\tau = 195$

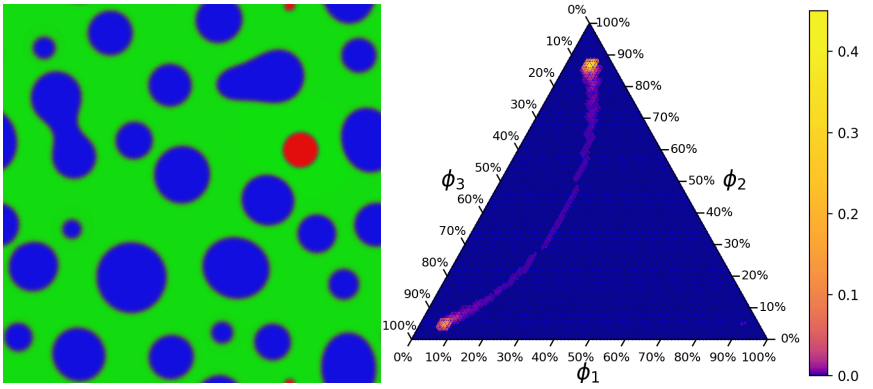
Figura 3.17: Estado del sistema y distribución de las concentraciones durante el transcurso de la simulación. (a) es la condición inicial, (b) se comienza a formar la fase asociada a ϕ_3 en todo el dominio (espinodal) y en (c) se observan dominios bien definidos para esta fase. (Continúa en la página siguiente).



(d) $\tau = 1794$



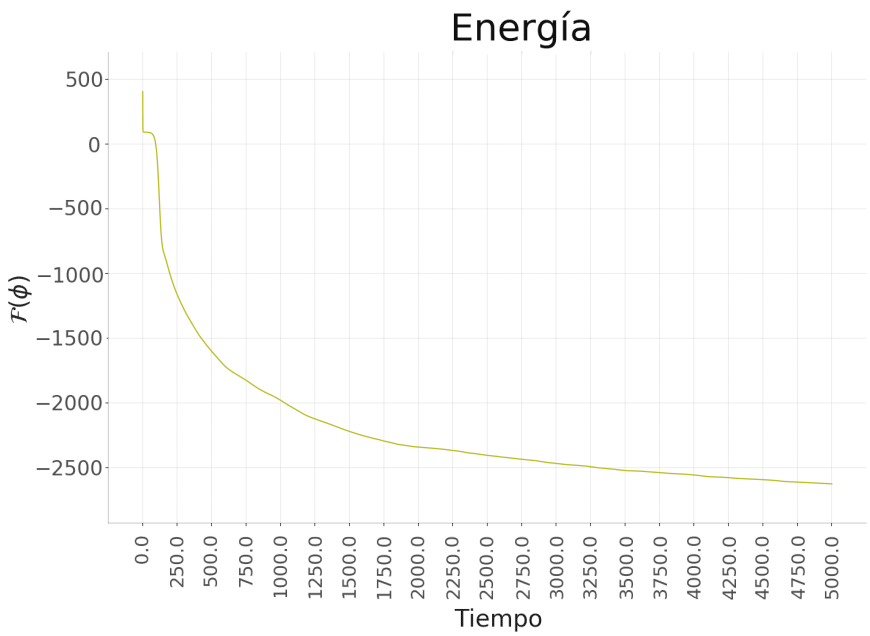
(e) $\tau = 3652$



(f) $\tau = 5000$

Figura 3.17: (Continuación) En (d) se aprecia el crecimiento de los dominios de la fase menos abundante (ϕ_3). En (e) se muestra la formación de sitios de nucleación de ϕ_1 y su crecimiento se observa en la condición final en (f).

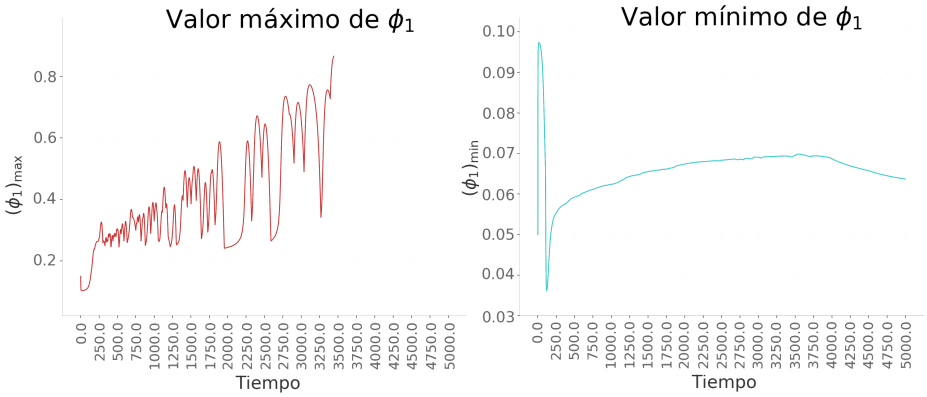
Para este caso, la función de Helmholtz, los máximos y los mínimos se muestran en la Figura 3.18. En 3.18(b) se observan muchas fluctuaciones en el máximo de ϕ_1 antes de formarse su fase asociada. Analizando con cuidado los resultados se encontró que los picos coinciden con eventos de coalescencia entre gotas y desaparición de éstas por difusión en contra de gradientes de concentración[10].



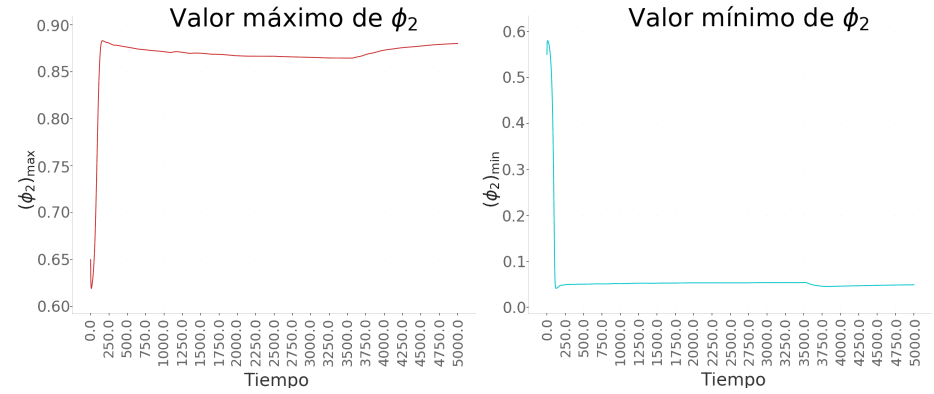
(a) Energía libre de Helmholtz

Figura 3.18: Energía, máximos y mínimos durante el transcurso de la simulación.

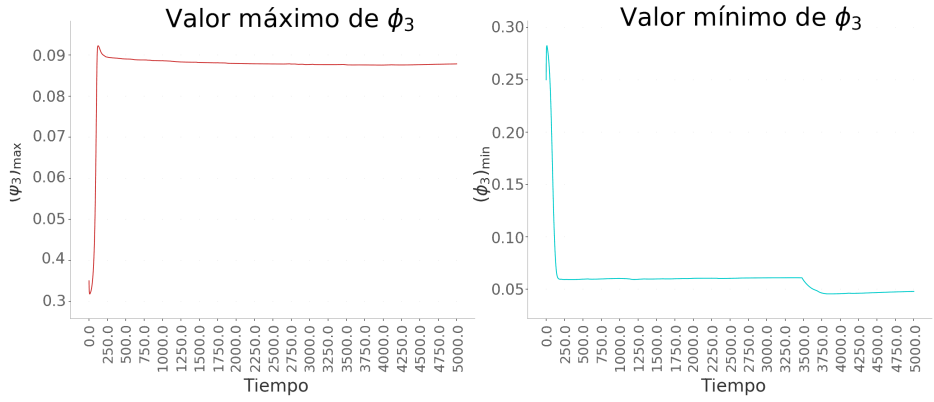
[10]Algo que tienen en común estos dos procesos es que se reduce en uno el número de gotas. Para los picos más aislados se observó claramente que coincidían con alguno de estos dos eventos, los más altos se registraron para la desaparición de gotas por difusión (en contra de gradientes de concentración). Para los otros picos, se llegan a apreciar alrededor de 50 en la gráfica 3.18(b), se decidió contar el número de eventos de coalescencia o desaparicion de gotas por difusión entre los tiempos $\tau = 250$ y $\tau = 3500$, donde se observa la mayor cantidad de picos. Se observaron 95 de estos eventos, pero también se observó que varios de ellos ocurrían simultáneamente. Esto nos indica que es verosímil la asociación entre los picos y la reducción en el número de gotas.



(b) Máximos y mínimos de ϕ_1



(c) Máximos y mínimos de ϕ_2



(d) Máximos y mínimos de ϕ_3

Una pregunta importante con respecto a estos resultados sería *¿Por qué se tarda tanto en formar la fase asociada a ϕ_1 en esta simulación, pero no en el caso similar de 7:2:1?* Un factor importante es que para este caso se observó descomposición espinodal, lo que ocasionó que el número de dominios de la fase ϕ_3 fuera mayor (casi el doble). Esto implica que había un mayor volumen interfacial, pero con la misma cantidad total de ϕ_1 en todo el dominio lo que quiere decir que el valor de ϕ_1 en la interfase es necesariamente menor[11]. Fue hasta que el sistema logró reducir el número de interfases que se llegó a formar la fase de ϕ_1. Esto también se observa en el hecho de que el valor mínimo[12] de ϕ_1 aumentó gradualmente hasta el momento en el que se formó su fase y en ese momento comenzó a decrecer.

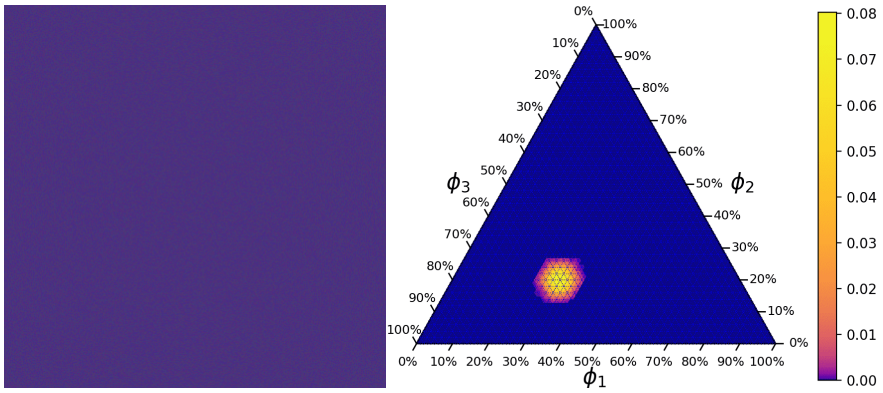
Familia 5:3:2

Se presenta separación espinodal entre una y dos fases, como en la Fig. 3.6(b).

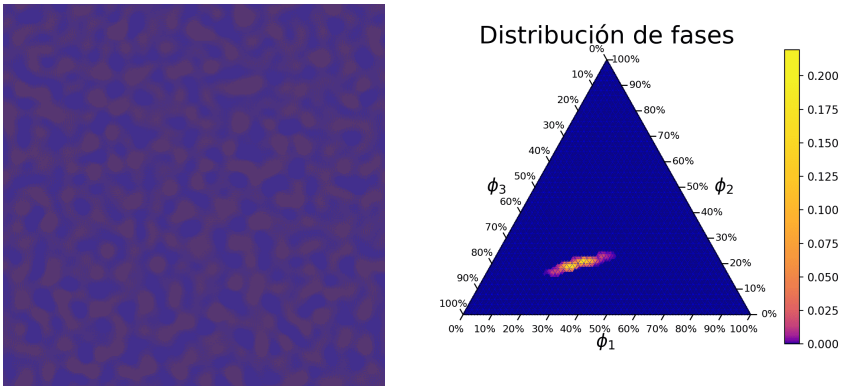
El caso que se reporta es el correspondiente a una concentración inicial promedio de $\overline{\boldsymbol{\phi}_0} \approx (0.3, 0.2, 0.5)$. Para mostrar su evolución, algunos estados de este sistema se muestran en la Figura 3.19

[11] Como se verá en las gráficas de la sección siguiente, fuera del bulto de una fase el valor más alto que alcanza su campo de fase se encuentra sobre la interfase entre los otros dos componentes. Esto también se observa en en los histogramas de distribución de concentraciones, las interfases se encuentran más cerca del centro del diagrama debido a la forma del potencial F.

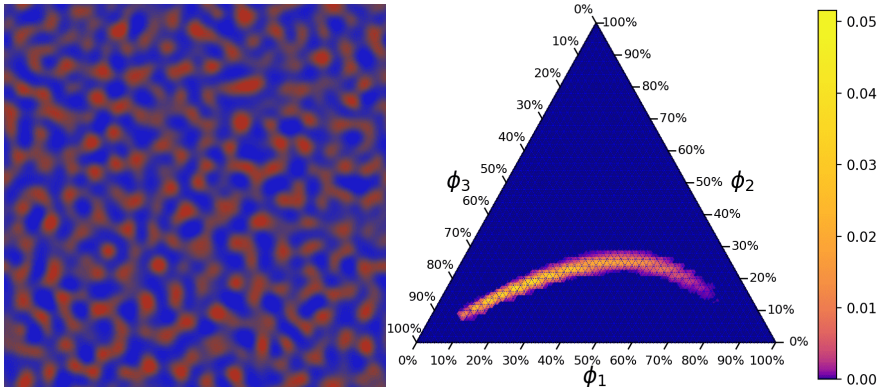
[12] Éste se da en los bultos de $\phi_{2,3}$, porque aquí las composiciones se acercan lo más posible a los mínimos del potencial, que en este caso se da en $\phi_1 \approx 0.055$, pero el valor promedio de ϕ_1 es 0.1 lo que causa que la concentración de los bultos se separe ligeramente de los mínimos.



(a) Condición inicial

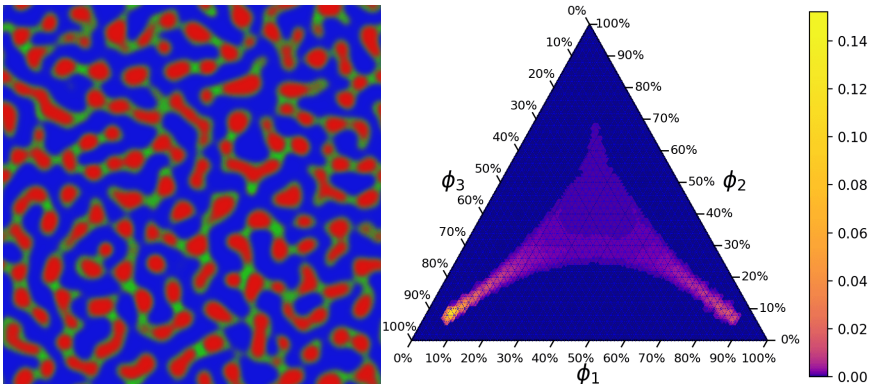


(b) $\tau = 90$

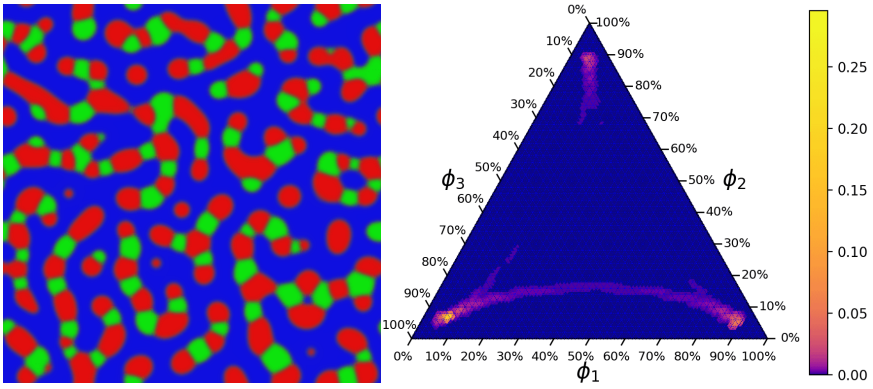


(c) $\tau = 141$

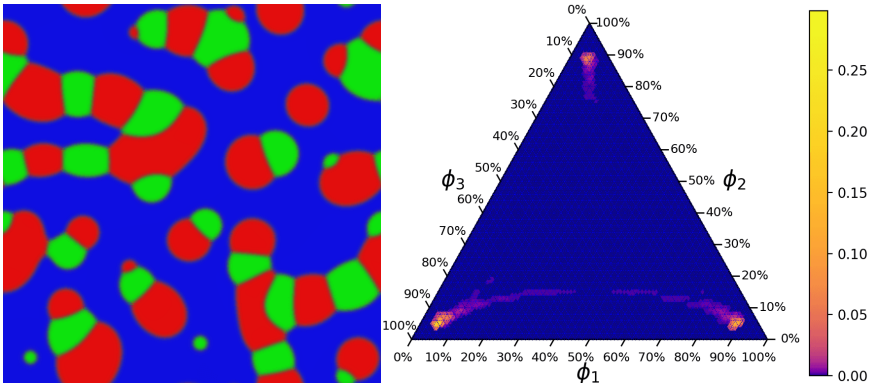
Figura 3.19: Estados del sistema para concentración inicial $\overline{\phi_0} \approx (0.3, 0.2, 0.5)$. En (a) se tiene la concentración inicial, en (b) se observa como comienza la decomposición espinodal entre ϕ_3 y una mezcla entre ϕ_1 y ϕ_2 y en (c) se comienzan a formar dominios de la fase de ϕ_1 dentro de la mezcla. (Continúa en la página siguiente).



(d) $\tau = 229$



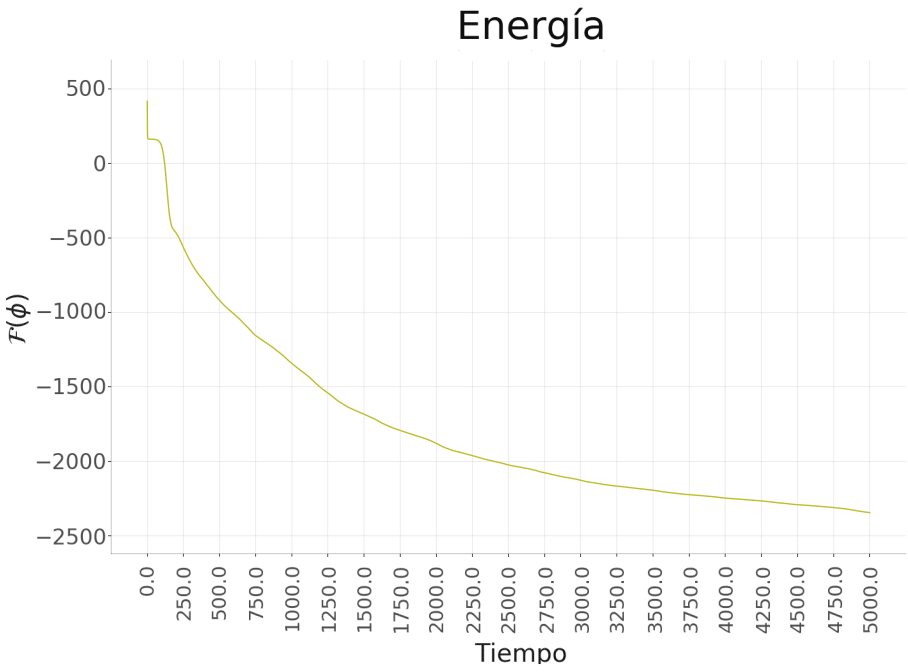
(e) $\tau = 692$



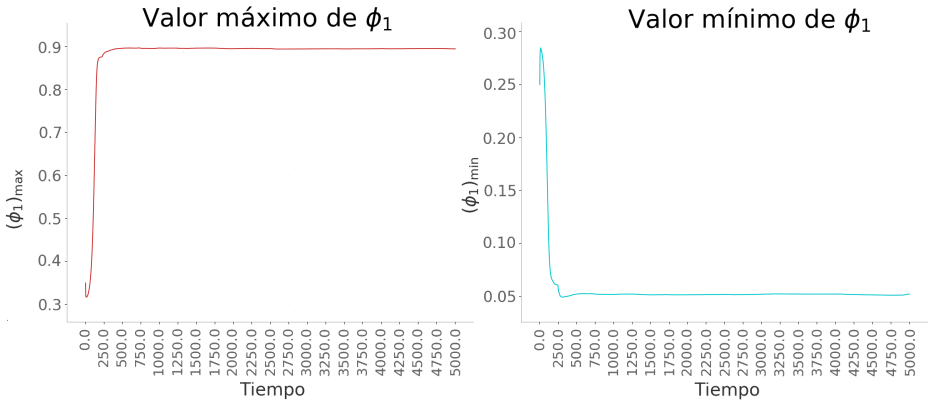
(f) $\tau = 5000$

Figura 3.19: (Continuación) En (d) se observa la formación la fase asociada a ϕ_2 dentro de la parte que era la mezcla y en (e) se observan completamente formadas todas las fases. En (f) se observa el crecimiento de los dominios en la condición final.

La energía libre de Helmholtz del sistema y los máximos y mínimos de los campos de fase se muestran en la Figura 3.20.

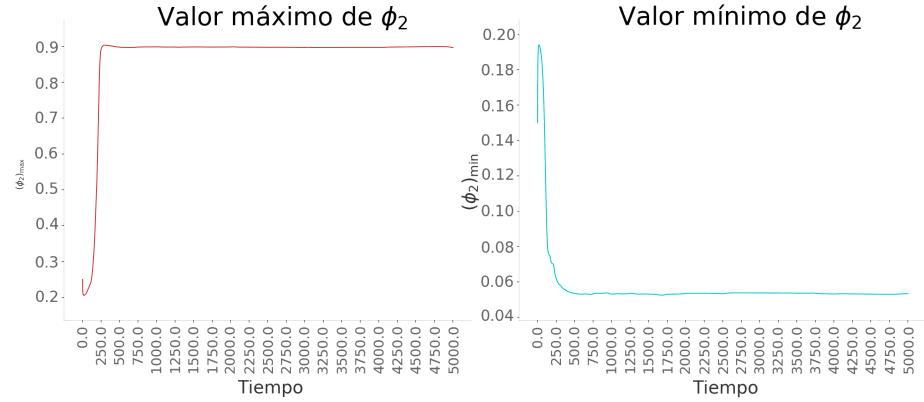


(a) Energía libre de Helmholtz

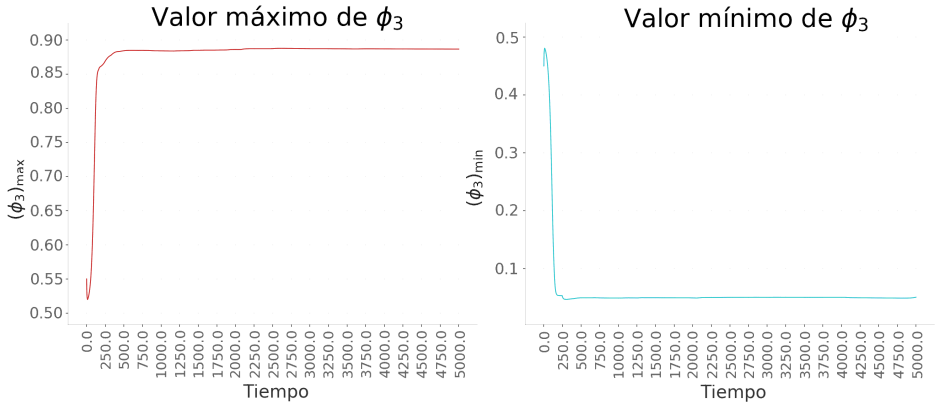


(b) Máximos y mínimos de ϕ_1

Figura 3.20: Energía, máximos y mínimos durante el transcurso de la simulación.



(c) Máximos y mínimos de ϕ_2

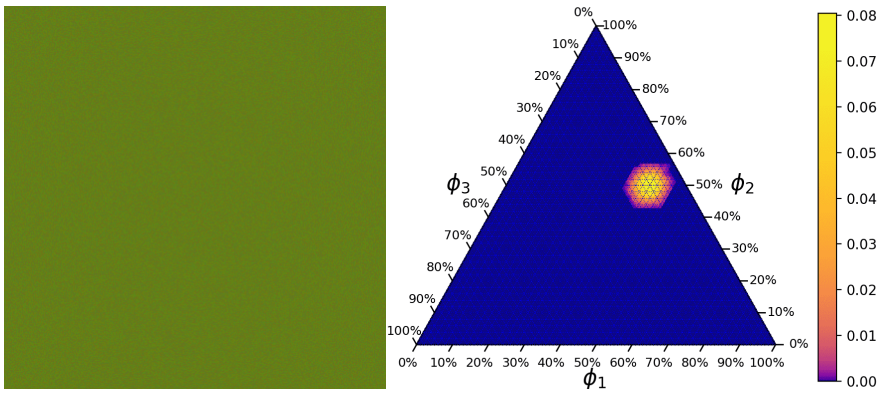


(d) Máximos y mínimos de ϕ_3

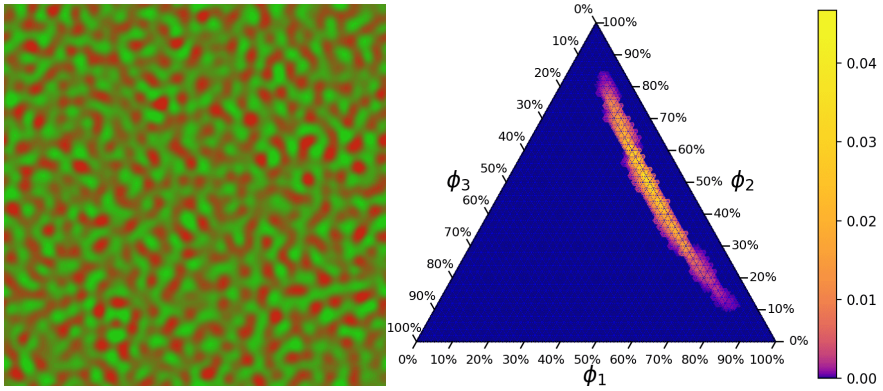
Las gráficas en la Fig. 3.20 son un caso prototipo de lo que esperamos encontrar en una simulación de este estilo (son muy similares a lo que se obtiene en una simulación binaria, si se deseara extrapolar los resultados de dichos sistemas esto sería una primera aproximación). La energía decrece exponencialmente (aparentemente) una vez que se formaron las fases y los máximos y mínimos convergen a los valores de bulto (las concentraciones sobre los pozos del potencial).

Familia 5:4:1

La separación que se observó es similar al caso anterior de la familia 6:3:1, con una decomposición espinodal inicial entre los componentes más saturados y una eventual formación de la fase menos saturada. La evolución del caso particular del sistema de composición inicial $\overline{\phi_0} \approx (0.4, 0.5, 0.1)$ se muestra en la Figura 3.21.

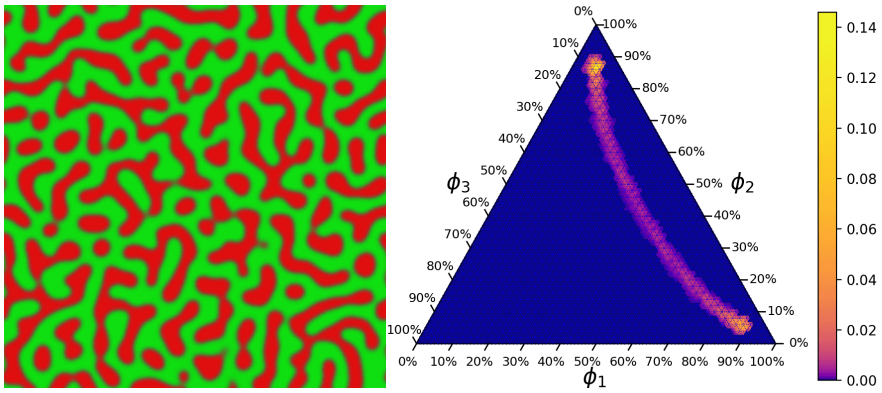


(a) Condición inicial

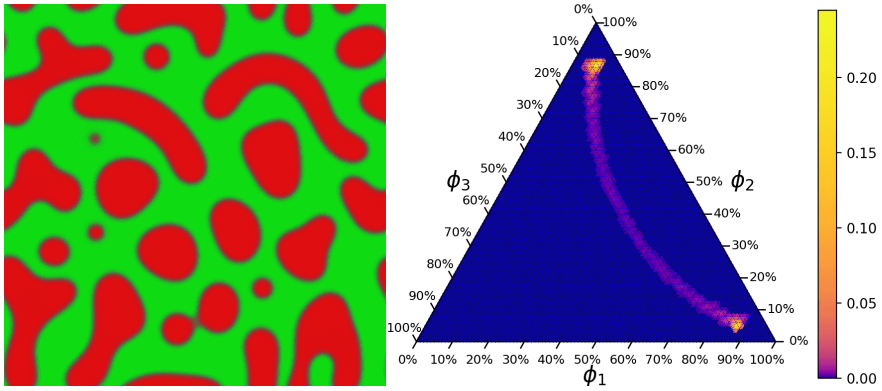


(b) $\tau = 100$

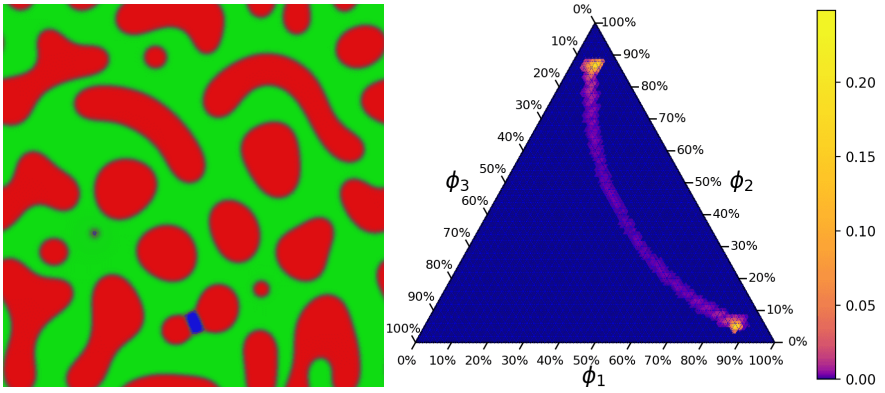
Figura 3.21: Estados del sistema con C.I. $\overline{\phi_0} \approx (0.4, 0.5, 0.1)$. Entre (b) y (c) se muestra la separación espinodal entre las fases de ϕ_1 y ϕ_2. (Continúa en la página siguiente).



(c) $\tau = 318$

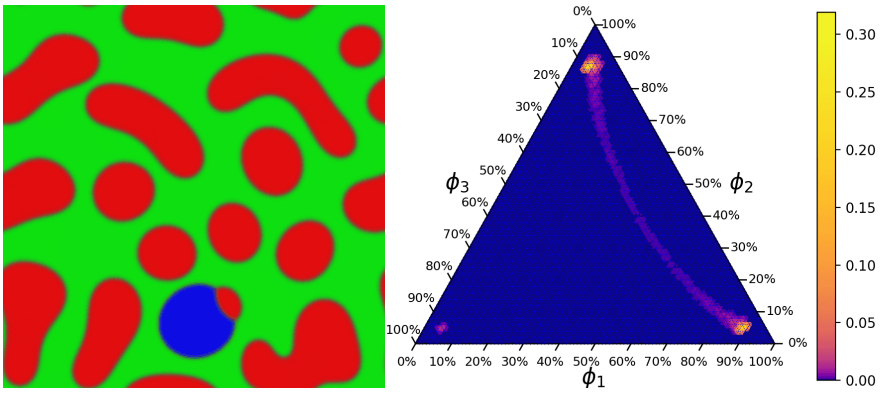


(d) $\tau = 2728$



(e) $\tau = 3048$

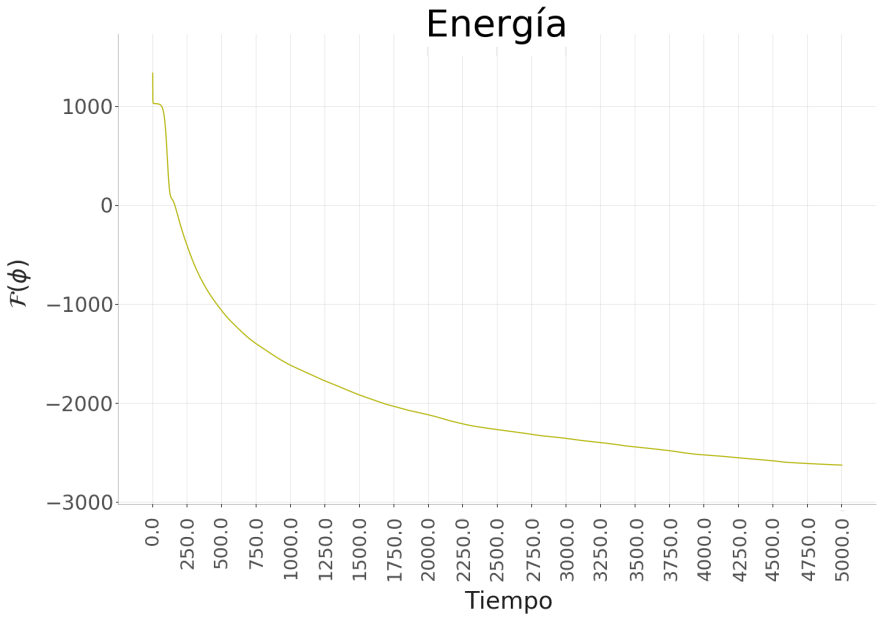
Figura 3.21: (Continuación) En (d) se observa el crecimiento de los dominios de ambas fases. En (e) se aprecia que se comienza a formar un dominio de la fase asociada a ϕ_3. (Continúa en la página siguiente).



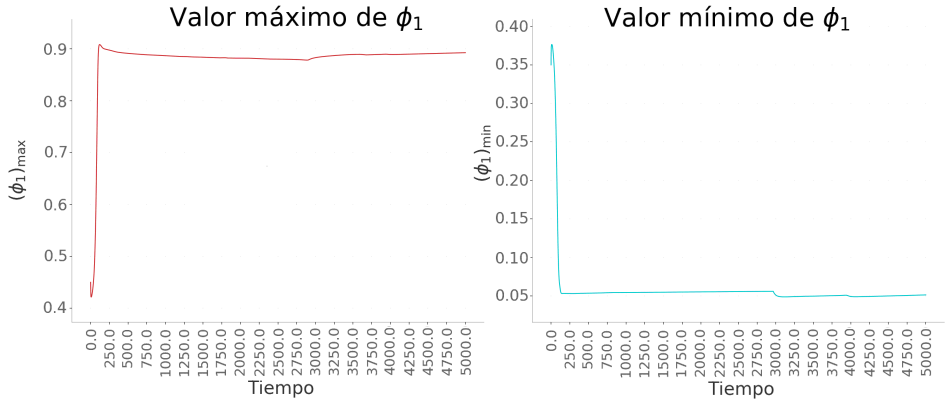
(f) $\tau = 5000$

Figura 3.21: (Continuación) En (f), la condición final, se ve el crecimiento de la fase asociada a ϕ_3.

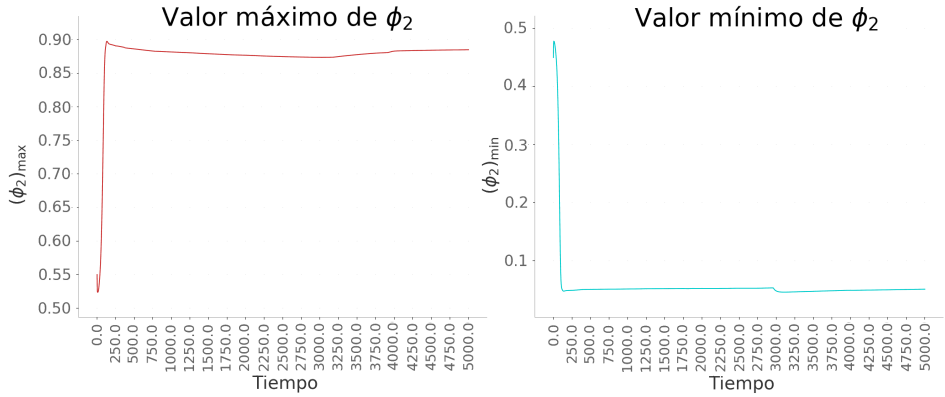
La energía de Helmholtz junto con los máximos y mínimos de los campos de fase se observan en la Figura 3.2.2. De forma similar a lo observado en la Fig. 3.18(b), se observan fluctuaciones importantes en el valor máximo del componente menos saturado hasta el momento en el que se forma su fase. Un aspecto de este sistema que contrasta con el caso anterior es que se observan considerablemente menos picos en la gráfica de $(\phi_1)_{\max}$, esto se puede atribuir a que el número de dominios (simplemente) conexos en este caso es menor y por lo tanto hubieron menos eventos de coalescencia que ocasionaran dichos picos. También se observa que la formación de la tercera fase (ϕ_3) se da en un tiempo anterior al otro caso. Nuestra suposición de que la nucleación se da para cierto valor de "saturación" en las interfases concuerda con la formación adelantada de este caso, ya que la existencia de menos dominios implica menos interfases.



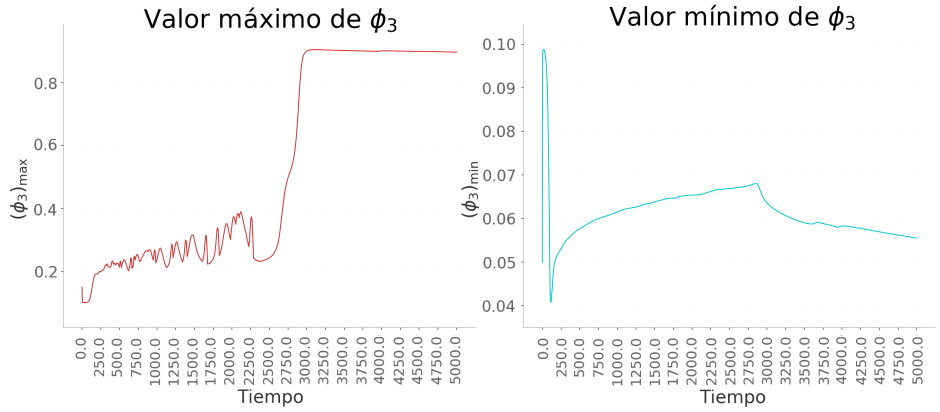
(a) Energía libre de Helmholtz



(b) Máximos y mínimos de ϕ_1



(c) Máximos y mínimos de ϕ_2



(d) Máximos y mínimos de ϕ_3

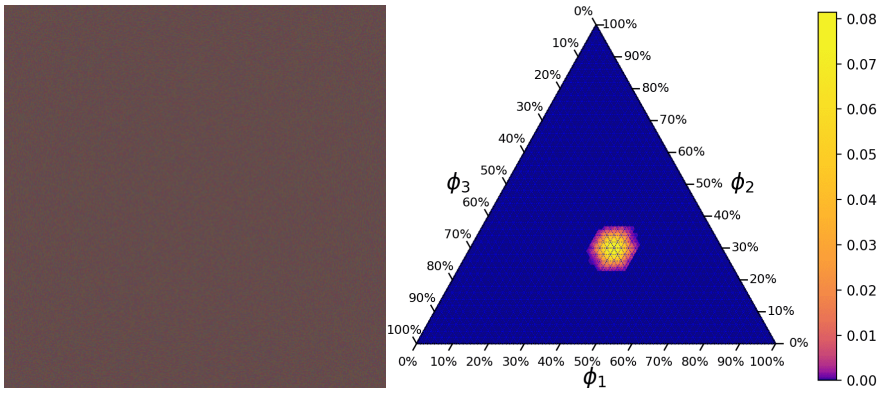
Figura 3.21: Energía, máximos y mínimos durante el transcurso de la simulación.

Nuevamente se observó en los resultados una coincidencia de los eventos de coalescencia y desaparición de gotas con la presencia de picos en la gráfica de los máximos[13].

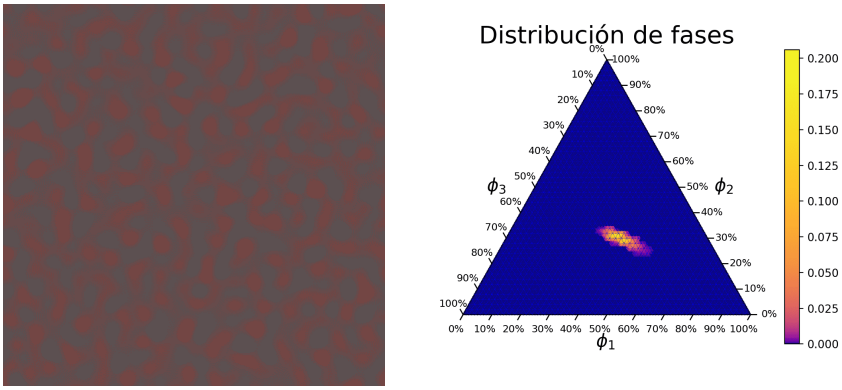
Familia 4:3:3

Como caso representativo se seleccionó al sistema de concentraciones iniciales promedio $\overline{\phi_0} \approx (0.4, 0.3, 0.3)$, donde se observó separación espinodal entre los tres componentes, como la de la Fig. 3.6(c). Se observan las imágenes de los estados del sistema en la Figura 3.22.

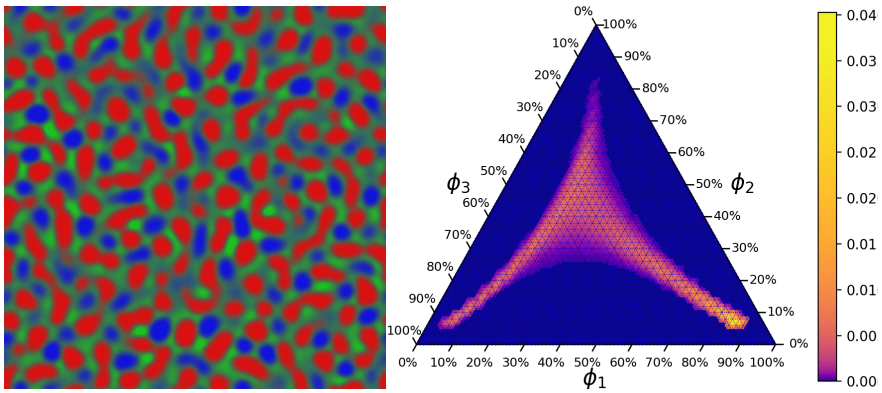
[13] A pesar de que hay menos picos y es más sencillo relacionarlos con estos eventos, no es exacta esta asignación. Por ejemplo, entre los tiempos $\tau \approx 2000$ y $\tau \approx 2300$ se observan 4 picos en la gráfica: el último claramente corresponde a la desaparición de una gota, pero los primeros tres coinciden con cuatro eventos, una coalesencia que ocurre primero y otras dos coalesencias que ocurren concurrentemente con la desaparición de una gota. Por estos casos "encimados" no podemos estar completamente seguros de que esta correlación sea correcta, pero resulta bastante convincente.



(a) Condición inicial

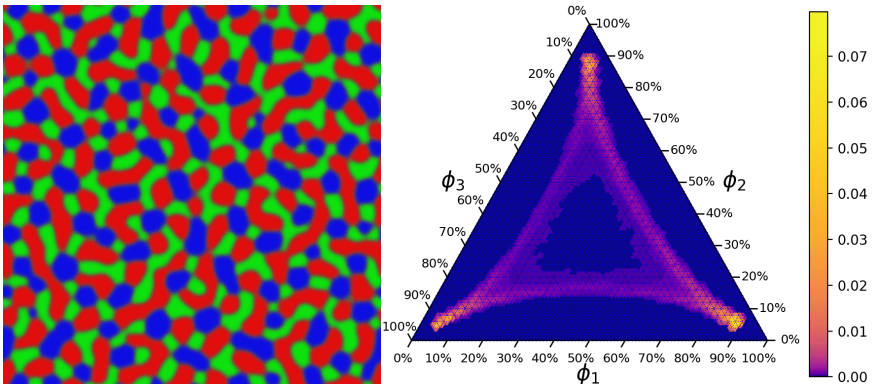


(b) $\tau = 152$

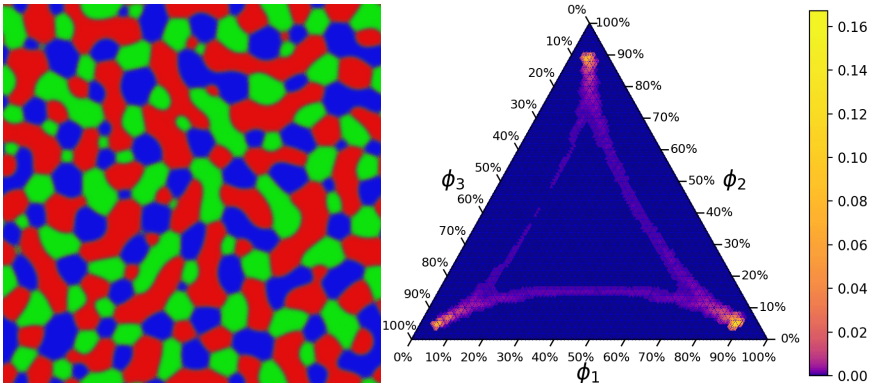


(c) $\tau = 304$

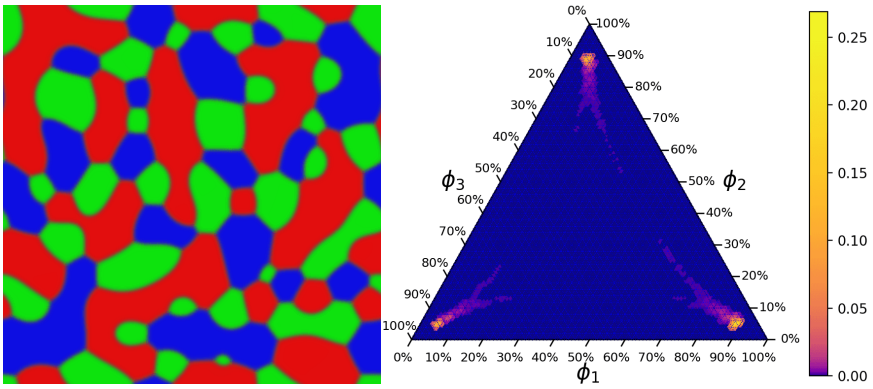
Figura 3.22: Estados del sistema para concentración inicial $\overline{\phi_0} \approx (0.4, 0.3, 0.3)$. En (a) se tiene la concentración inicial, en (b) se observa como comienza la descomposición espinodal con ϕ_1, cuya fase es más saturada y en (c) se comienzan a separar también las otras dos fases. (Continúa en la página siguiente).



(d) $\tau = 444$



(e) $\tau = 1364$

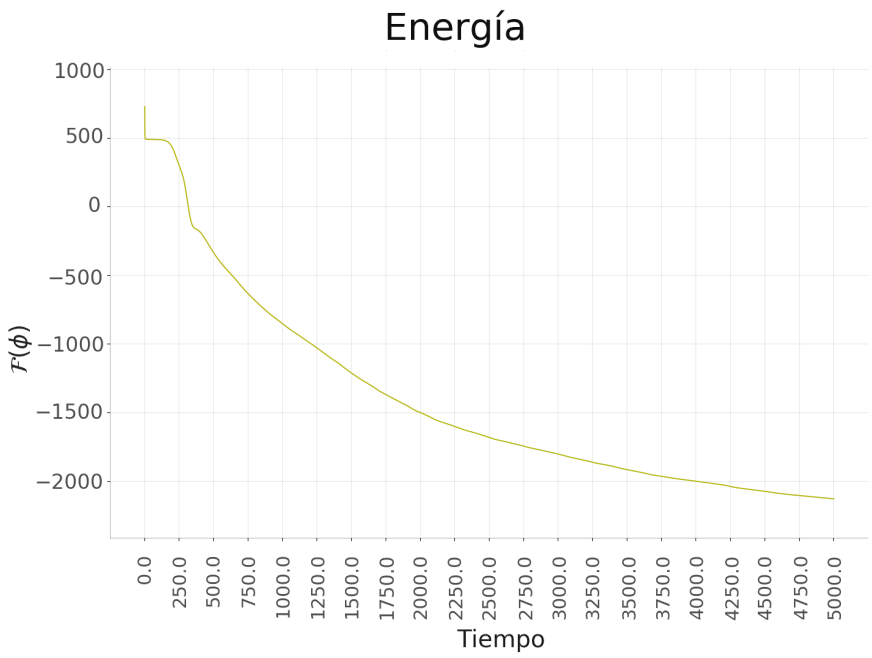


(f) $\tau = 5000$

Figura 3.22: (Continuación) En (d) se observan completamente formados los dominios de las fases. En (e) y (f) se observa el crecimiento de ellas (y la condición final).

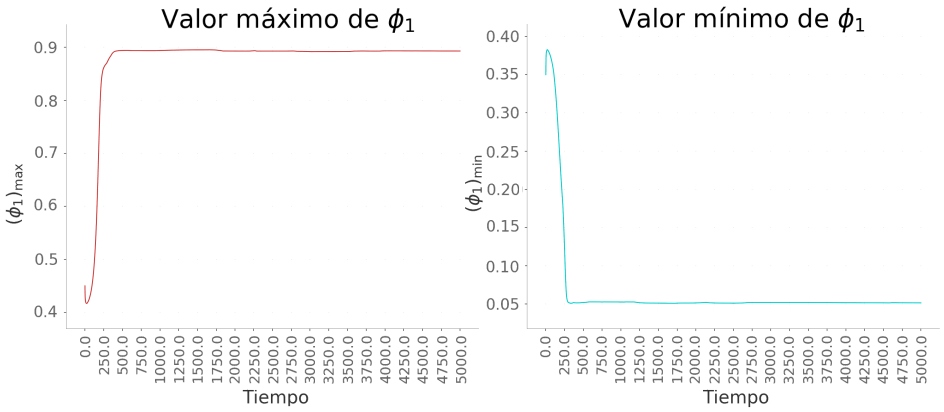
Un aspecto interesante de este sistema es que si se observan las Figs. 3.22(d–f) parece ser que se hubiera hecho un "zoom" a alguna región, ya que las estructuras son muy similares. Esto contrasta con las otras simulaciones donde es más común que los dominios tiendan a volverse esferas.

Algo que se alcanza a observar es que aquí también se presentó la asimetría con respecto a ϕ_3 que se mostró en la Fig. 3.16. Las gráficas de las cantidades globales del sistema se encuentran en la Figura 3.23.

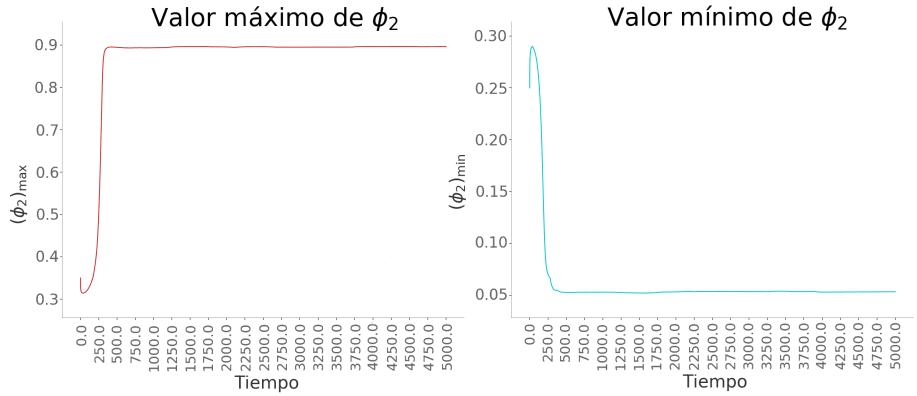


(a) Energía libre de Helmholtz

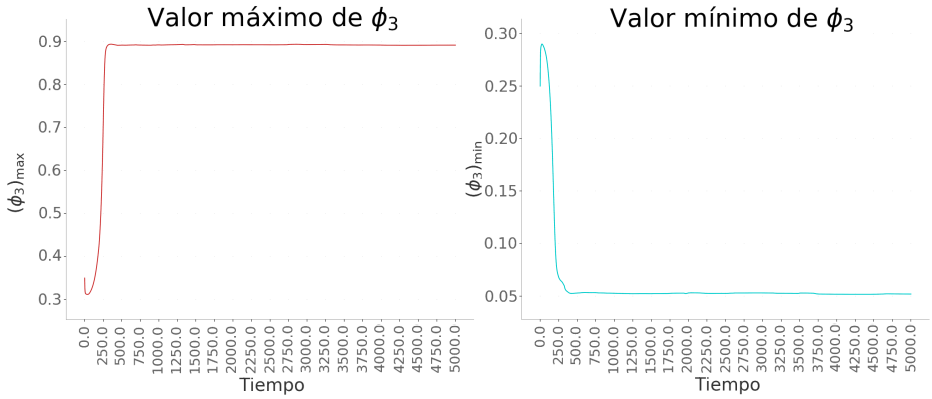
Figura 3.23: Energía, máximos y mínimos durante el transcurso de la simulación.



(b) Máximos y mínimos de ϕ_1



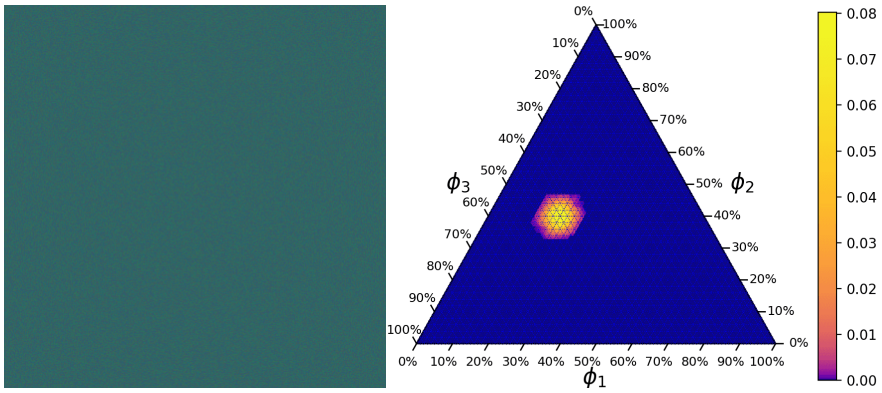
(c) Máximos y mínimos de ϕ_2



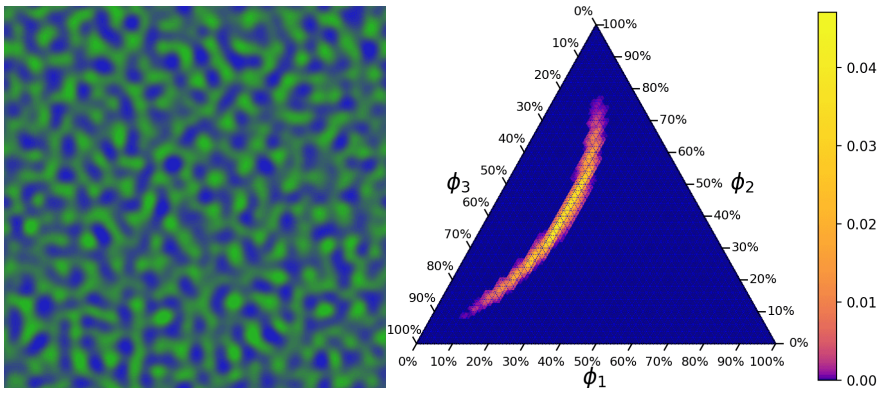
(d) Máximos y mínimos de ϕ_3

Familia 4:4:2

Finalmente, se concluye esta Sección con los resultados del último bloque de sistemas que se simuló. En este caso se observó una separación espinodal entre 3 componentes, como la del caso anterior. El caso que se reporta aquí es el correspondiente a $\overline{\phi_0} \approx (0.2, 0.4, 0.4)$, su evolución se muestra en la Figura 3.24

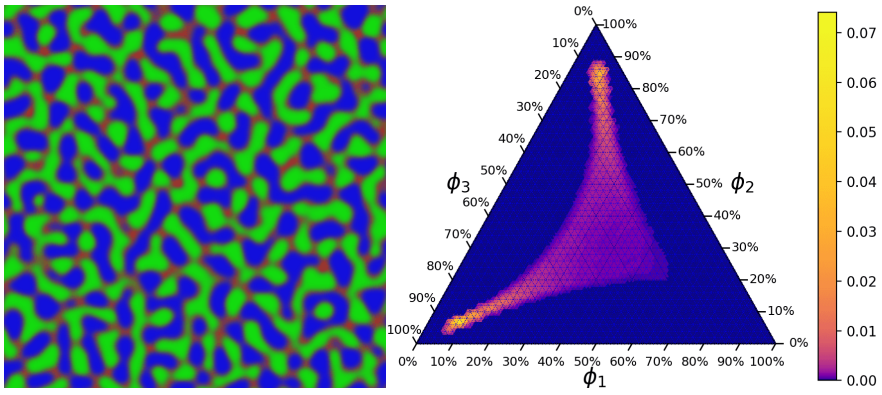


(a) Condición inicial

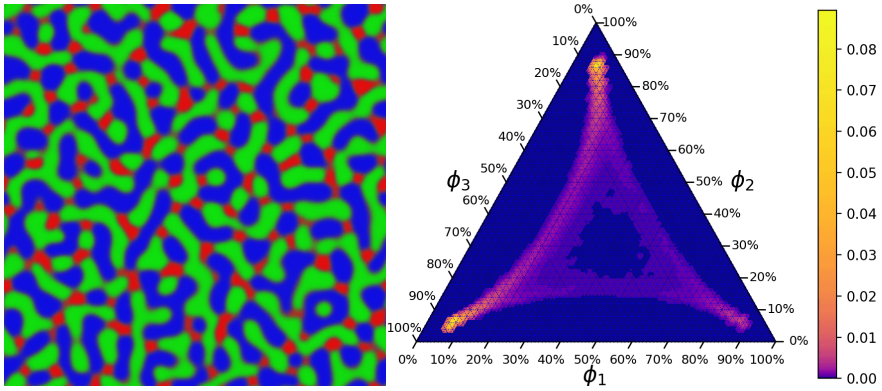


(b) $\tau = 120$

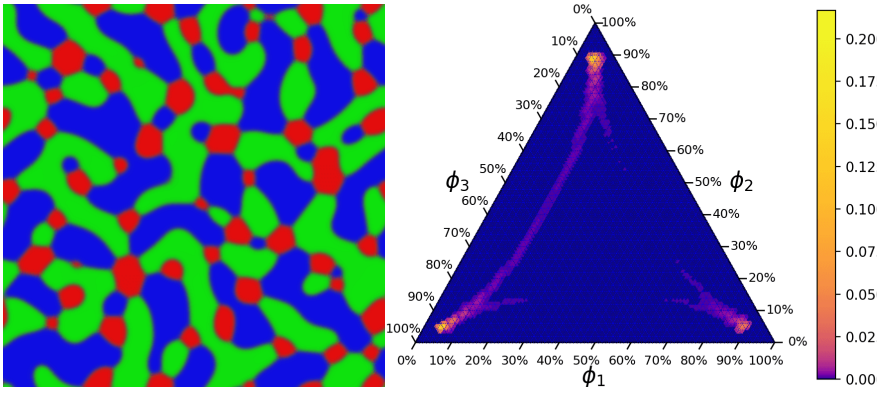
Figura 3.24: Estados del sistema $\overline{\phi_0} \approx (0.2, 0.4, 0.4)$. En (a) se observa la condición inicial y en (b) se muestra el inicio de la descomposición espinodal, principalmente entre ϕ_2 y ϕ_3, que son las fases más saturadas. (Continúa en la página siguiente).



(c) $\tau = 206$

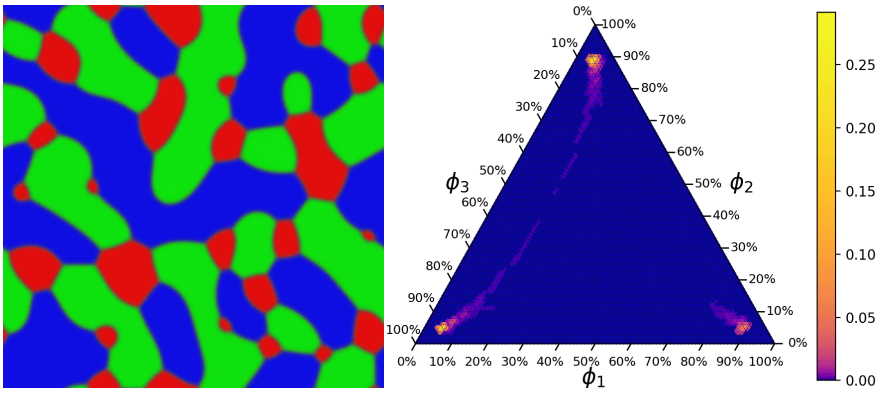


(d) $\tau = 291$



(e) $\tau = 1660$

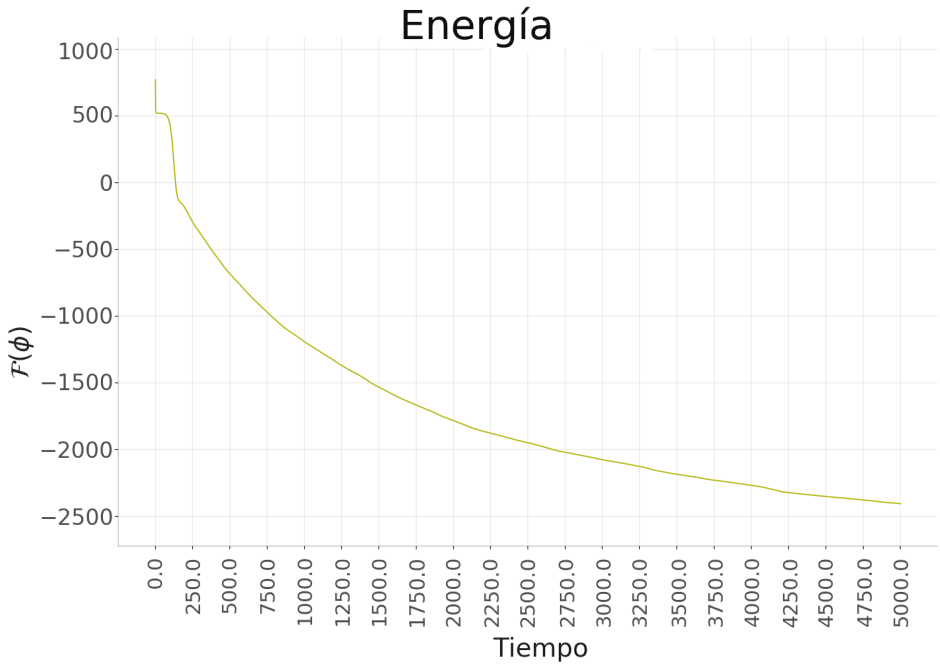
Figura 3.24: (Continuación) En (c) se observan los inicios de la formación de la fase de ϕ_1 y en (e) se ve este proceso terminado. En (e) muestra el crecimiento de los dominios. (Continúa en la página siguiente).



(f) $\tau = 5000$

Figura 3.24: (Continuación) En (f) se tiene la condición final.

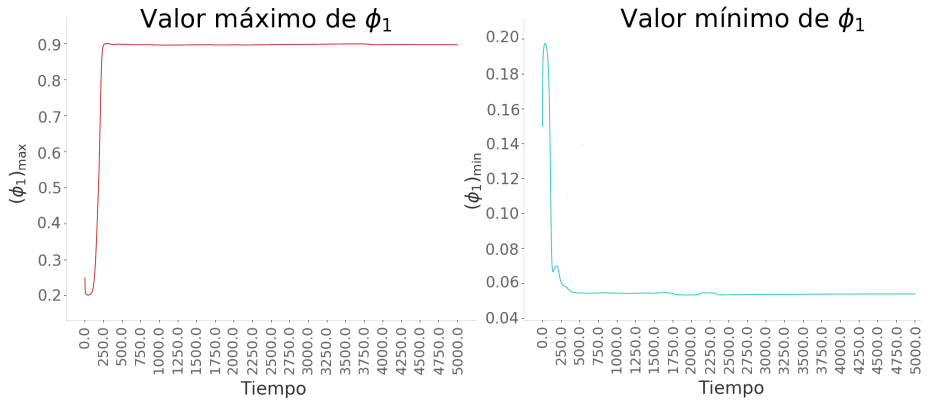
Las Figs. 3.24(d–f) también son muy similares entre sí, como si se observara lo mismo a menor escala[14]. Las cantidades globales se observan en la Figura 3.25.



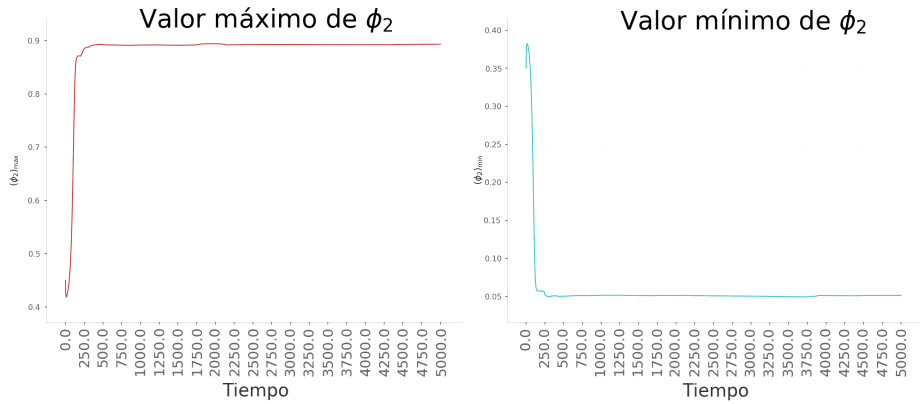
(a) Energía libre de Helmholtz

Figura 3.25: Energía, máximos y mínimos durante el transcurso de la simulación.

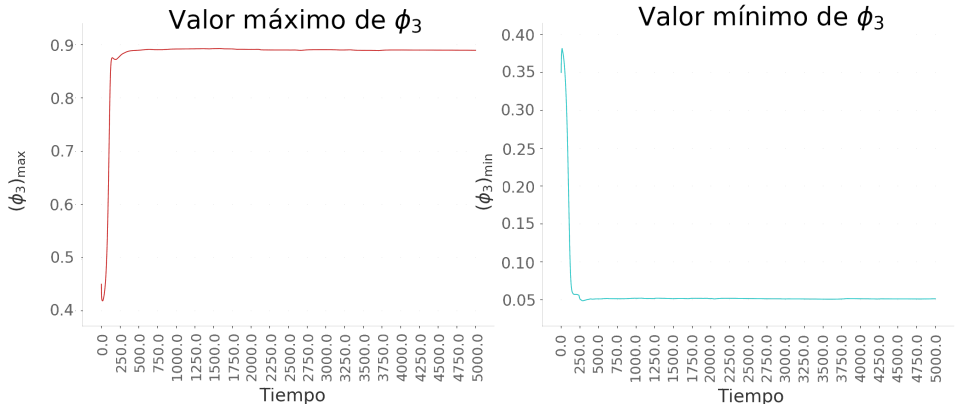
[14]Si se detiene la difusión (bajando la temperatura) en un tiempo adecuado se pueden obtener dominios (granos en aleaciones) del tamaño deseado, manteniendo las demás características.



(b) Máximos y mínimos de ϕ_1



(c) Máximos y mínimos de ϕ_2



(d) Máximos y mínimos de ϕ_3

Para concluir esta sección se destaca que dentro de los resultados obtenidos se observa cualitativamente que los casos que presentaron descomposición espinodal presentan una formación de fases más rápida que los casos con separación metaestable. Esta observación coincide con lo que encontraron experimentalmente en Rogers y col., 2019. Lamentablemente no es posible cotejar nuestros resultados cuantitativamente porque en Rogers y col., 2019, utilizan técnicas de difracción para caracterizar la separación de las fases, cuyo cálculo requiere del conocimiento de propiedades ópticas (en particular el índice de refracción, la constante dieléctrica y su dependencia con la densidad, de acuerdo a lo que se describe en Abdel-Azim y Munk, 1987), las cuales no están especificadas en este trabajo.

3.3. Obtención del Espesor de Interfase

El espesor de la interfase ξ es un parámetro importante del sistema, su valor depende de un "equilibrio" entre los términos de energía libre de bulto y de interfase que componen al potencial de Helmholtz del sistema. Recordando que esta energía está dada por la ecuación (1.9),

$$\mathcal{F}[\boldsymbol{\phi}] = \int \left[F(\boldsymbol{\phi}) + \frac{1}{2}\kappa \sum_{i=1}^{3} \|\boldsymbol{\nabla}\phi_i\|^2 \right] \mathrm{d}v$$

El primer término $F(\boldsymbol{\phi})$ tiene pozos en el bulto de las fases, por lo que en un proceso que minimice la energía este término promoverá la minimización del ancho de la interfase[15], mientras que al segundo se le puede asociar una energía del orden de

$$\int \|\boldsymbol{\nabla}\phi_i\|^2 \, \mathrm{d}v \sim \int \xi^{-2} \mathrm{d}v \sim \xi^{-1}$$

lo que implica que, en un proceso que minimiza el potencial de Helmholtz, este término promueve que aumente el ancho de interfase.

El ancho final estará dado entonces por un equilibrio entre estos dos efectos y podrá depender de la geometría de la interfase, particularmente de la curvatura.

[15] En la interfase F es mayor, lo que hace que este término tenga una mayor contribución a la energía cuando hay regiones interfaciales más extensas.

En nuestro caso con $M(\phi) \approx \text{cte}$, la condición de equilibrio se puede expresar como[16] $\nabla^2 \mu_i = 0$.

Por simiplicidad, se consideró únicamente el caso de interfases sin curvatura (planas). Para estos experimentos numéricos se varió el valor de κ (para cambiar el peso relativo entre los dos términos) y se calculó el valor de ξ para cada caso.

La transición de fases se da de una manera continua y suave[17], por lo que hay que asignar un valor de corte para lo que se considera una región de interfase. Aquí se consideró el 90 % central del rango. Es decir,

$$\mathcal{I} = \{\mathbf{x} \in \Omega \mid \phi_- + 0.05\,(\phi_+ - \phi_-) \leq \phi(\mathbf{x}) \leq \phi_+ - 0.05\,(\phi_+ - \phi_-)\}$$

$$\mathcal{B} = \{\mathbf{x} \in \Omega \mid \mathbf{x} \notin \mathcal{I}\}$$

[16] La interpretacón de este laplaciano cero es que, para cualquier punto en el espacio $\mathbf{x}_0$, el potencial químico toma un valor en el punto igual al valor promedio alrededor de ese punto. En términos más rigurosos, se tiene que si $\langle \mu_i \rangle_{U \ni \mathbf{x}_0} \coloneqq \frac{1}{\Delta U} \int_U \mu_i(\mathbf{x}) \mathrm{d}v$, con $\Delta U \coloneqq \int_U \mathrm{d}v$ el volumen ocupado por U, entonces

$$\nabla^2 \mu_i(\mathbf{x}_0) = 0 \Leftrightarrow \left[\lim_{\Delta U \to 0} \langle \mu_i \rangle_{U \ni \mathbf{x}_0} = \mu_i(\mathbf{x}_0) \right]$$

Físicamente esto quiere decir que no hay flujo másico neto hacia un punto si el potencial químico en él es igual, en promedio, al de sus alrededores. Esta descripción e interpretación del laplaciano se puede encontrar en Byron y R.W., 1992, y es análoga al equilibrio térmico (la diferencia es que en transferencia de calor, la temperatura misma juega también el papel de potencial, mientras que en nuestro caso se requiere introducir al potencial químico. La analogía es perfecta si se considera el caso de difusión lineal, gobernada por las leyes de Fick).

[17] Para el caso de dos fases en una dimensión, la solución está dada por una tangente hiperbólica, que sólo toma el "valor de bulto" en el infinito.

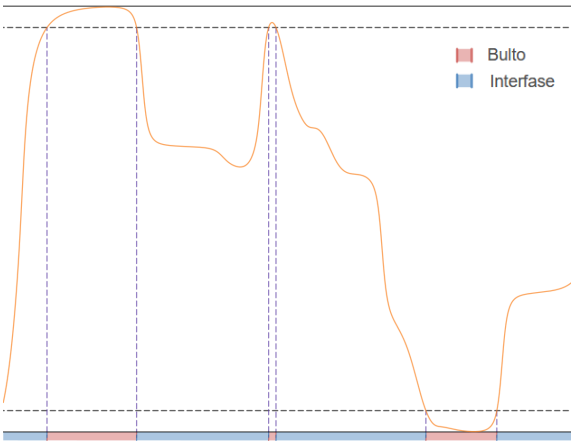


Figura 3.26: Ejemplo de una concentración en una dimensión, generada mediante una función con parámetros aleatorios. La imágen se generó usando el graficador Desmos.

La región de interfase es $\mathcal{I}$, mientras que $\mathcal{B}$ es el bulto. Se generó una imagen ejemplo para ilustrar cómo se hace la clasificación de las regiones. Ésta se muestra en la Figura 3.26.

La selección de un valor de corte es arbitraria[18] y el ancho que se obtenga para la interfase depende de esto.

Para evaluar ξ se partió de una condición inicial de interfaz abrupta y se permitió evolucionar al sistema hasta alcanzar un valor estable del espesor. La función $F(\phi)$ se dejó constante y se varió el valor de κ.

Como condición inicial se introdujeron tres regiones con interfase plana. En cada una de éstas, a uno de los campos de fase se le asignó el valor de $\phi_+(\approx 0.8898935)$ y a los otros dos se les asignó $\phi_-(\approx 0.05505326)$. Estos son los valores de concentración en los mínimos de F. La condición inicial se ilustra en la Figura 3.27.

[18]Esta arbitrariedad es relativa. Una selección adecuada del valor de corte incluiría únicamente la región donde la magnitud de $\nabla\phi$ es significativa.

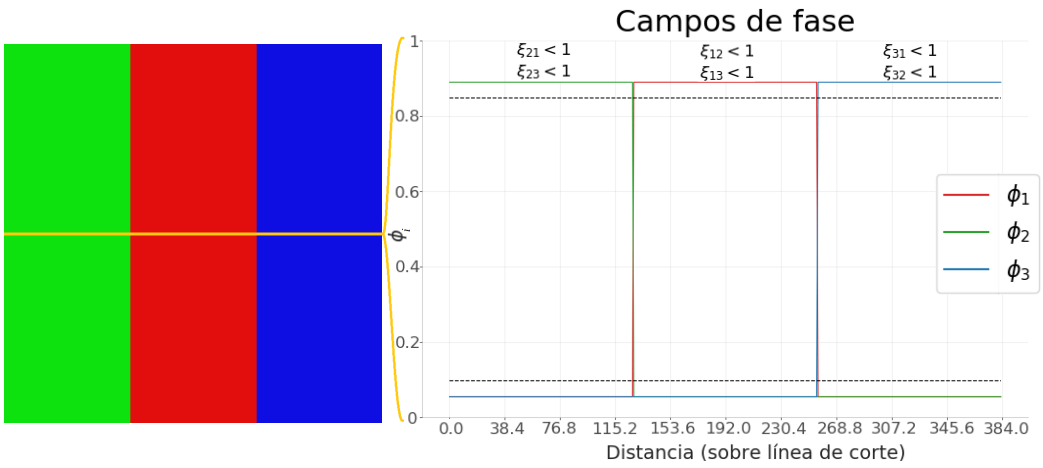


Figura 3.27: Condición inicial utilizada en los experimentos de cálculo de espesor de interfase. Dicho cálculo se realizó a lo largo de una linea, centrada verticalmente en el dominio (en amarillo). La gráfica de la derecha muestra individualmente los campos de fase sobre la línea, junto con el valor de corte escogido.

Dado que la solución es numérica y no exacta, se decidió evaluar cada ancho de interfase por separado. Se denotó por ξ_{ij} al espesor de la interfase formada entre los componentes i y j vista desde el componente i. Es decir que en las fronteras de esa interfase se cumple que

$$(\phi_i(x) = \phi_+) \Leftrightarrow \left(\phi_i\left(x \overset{+}{(-)} \xi_{ij}\right) = \phi_-\right) \wedge \left(\phi_j\left(x \overset{+}{(-)} \xi_{ij}\right) = \phi_+\right)$$

donde $\overset{+}{(-)}$ quiere decir "más **o** menos" y $\wedge$ es el operador lógico "y".

Con los resultados de estas simulaciones se obtuvo una expresión por mínimos cuadrados para ξ en función del valor de κ. El ajuste se encuentra en la Figura 3.28

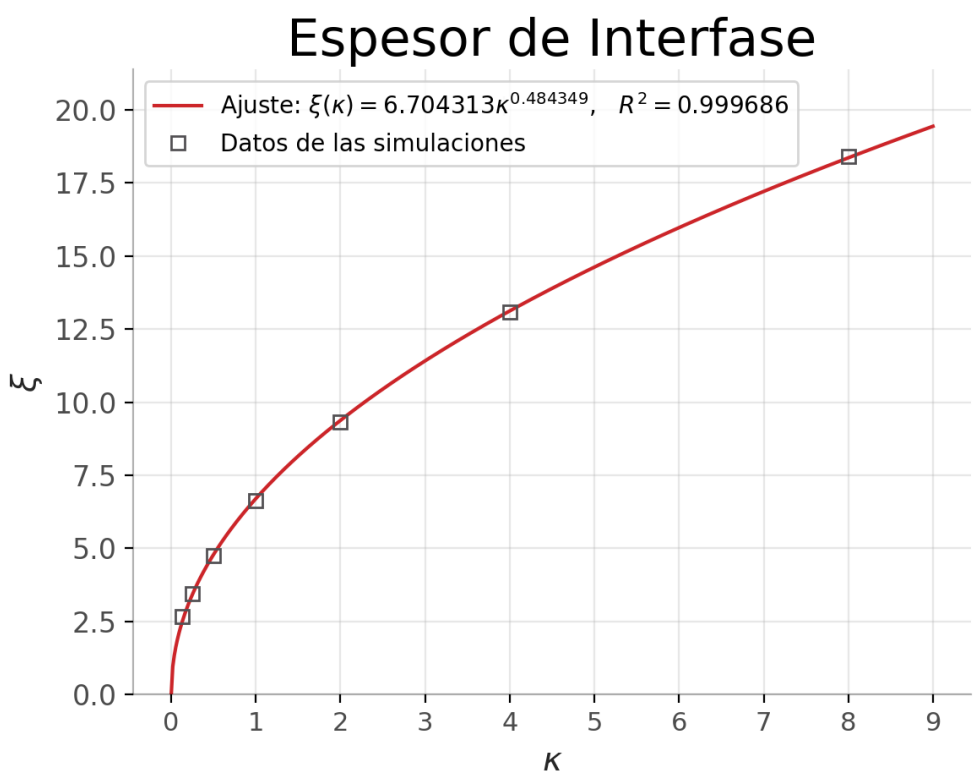


Figura 3.28: Gráfica del espesor de interfase ξ como función del coeficiente de energía interfacial κ. Las dimensiones de ξ están dadas como $[\xi] = [\Delta x]$, es decir son unidades del paso de malla, mientras que κ está en $[\kappa] = u_E$, con u_E las unidades de energía que se le deseen asignar al potencial adimesionalizado (1.14).

En general sí se observó una desviación entre los distintos ξ_{ij}, pero para el valor final que se graficó en la Fig. 3.28 dicha desviación (estándar) fue menor a 0.3 % en todos los casos. Sin embargo, la discrepancia en el "espesor de interfase" aparente[19] fue mayor en el régimen transitorio.

[19] El espesor de interfase ξ está propiamente definido como una cantidad de equilibrio, el valor que se fue calculando durante las simulaciones tiende asintóticamente a ξ.

Un detalle importante que cabe mencionar es el valor de ξ para $\kappa = 2^{-3}$, en este caso se tuvo $\xi \approx 2.666$, esto indica que había menos de 3 puntos de malla en la interfase.En la ecuación de Cahn- -Morral, el término asociado a la energía interfacial se traduce a una cuarta derivada, pero numéricamente una derivada de cuarto orden requiere cinco puntos (para nosotros, tres en la interfase y uno en cada bulto). Debido a esta situación el método puede perder precisión si se tienen menos de tres nodos al interior de la región interfacial[20].

Un aspecto interesante de la gráfica 3.28 es que se encontró una ley de potencia con un exponente muy cercano a 1/2. Para el caso de dos fases la teoría nos dice que el exponente debe tomar exactamente este valor (Pego, 1989). *A priori*, no era posible saber si el añadir un tercer componente haría que cambiase esta dependencia, pero nuestros resultados parecen indicar que no hay diferencia con el caso binario en una primera aproximación.

A continuación se muestran las gráficas de ancho de interfase durante el transcurso de la simulación.

[20]Considerando esta situación se hizo otro ajuste para la curva de la Fig. 3.28 omitiendo el dato de $\kappa = 2^{-3}$, se observó un mejor ajuste por un orden de magnitud ($R^2 - 1$ fue un orden de magnitud menor, $R^2 = 0.9999$). La misma mejoría no se obtuvo al omitir ningún otro de los datos, incluso llegando a empeorar. Esta observación refuerza la posibilidad de pérdida de precisión debajo de $\xi = 3$.

En la Figura 3.29 se observa que la discrepancia transitoria aumenta con la mayor de las κs. Otra característica notoria es que el sistema llega más lentamente al valor estacionario para mayor κ, esto ocurre por que el sistema evoluciona a través de las interfases y al ser éstas más grandes se facilita la interacción. En los demás sistemas también se observa esta misma tendencia de que una κ mayor presenta una mayor discrepancia así como un tiempo característico mayor, esto se presenta en las Figuras 3.30–3.32.

Una excepción, observada en todas las figuras, a la discrepancia mencionada es ϕ_3, que presenta ambos espesores con el mismo valor exactamente. Esto ocurre por que la ecuación en diferencias es simétrica bajo intercambios de ϕ_1 y ϕ_2 y por lo tanto $\xi_{31} \equiv \xi_{32}$.

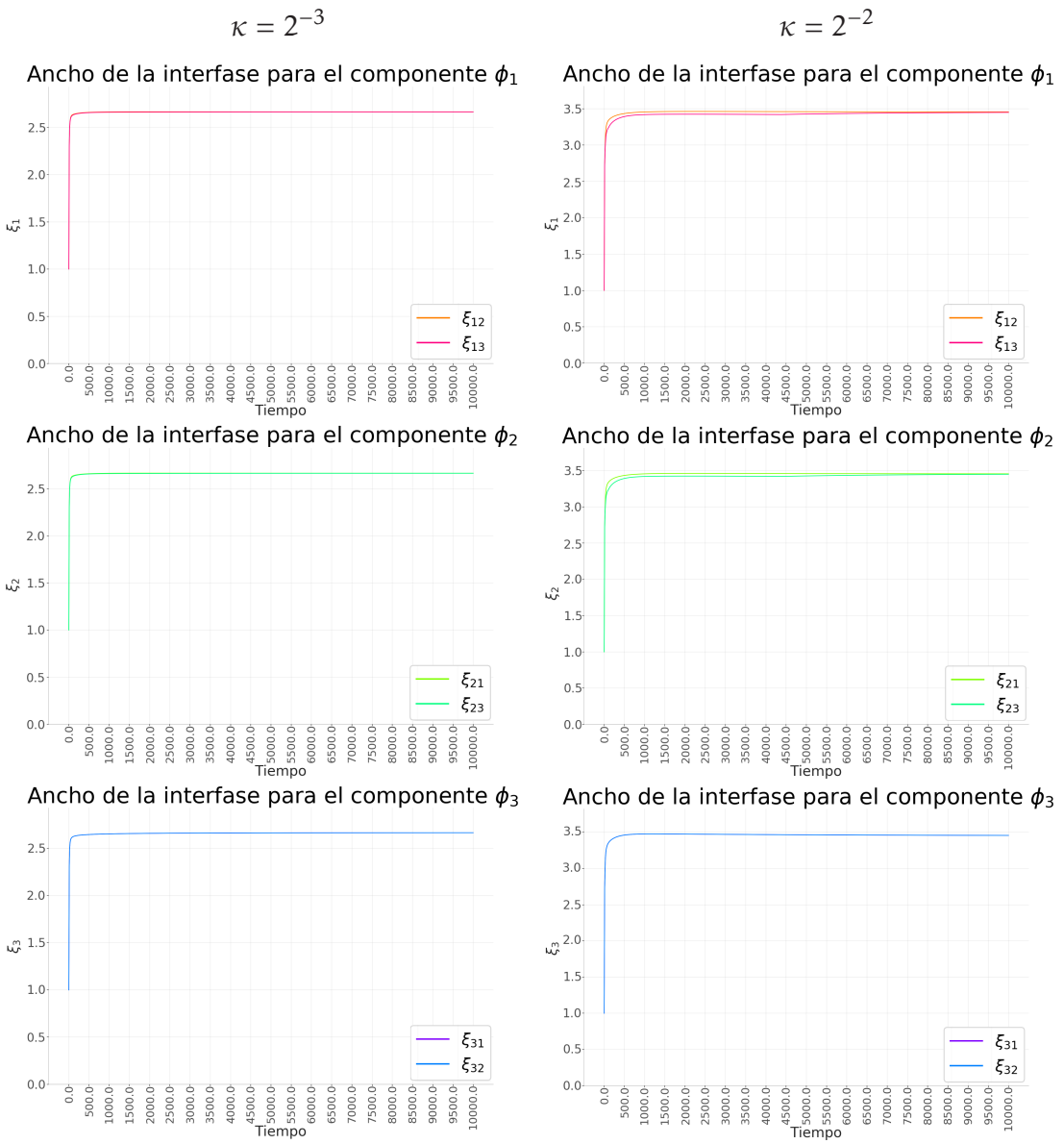


Figura 3.29: Gráficas del espesor de Interfase durante el transcurso de las simulaciones. Del lado izquierdo se tienen las gráficas para la simulación con $\kappa = 2^{-3}$ y del lado derecho $\kappa = 2^{-2}$.

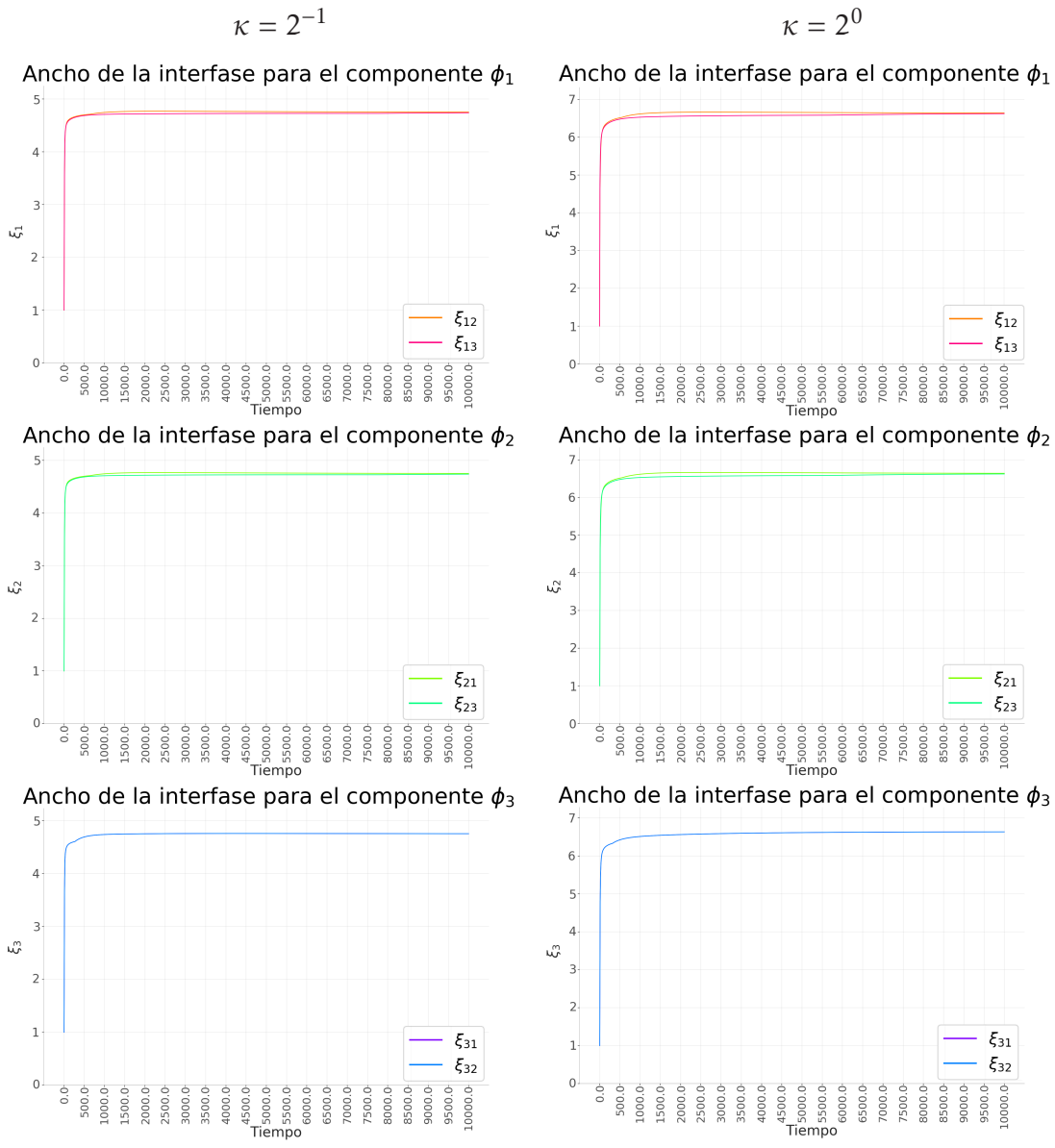


Figura 3.30: Gráficas del espesor de Interfase para, del lado izquierdo, $\kappa = 2^{-1}$ y, del lado derecho, $\kappa = 1$.

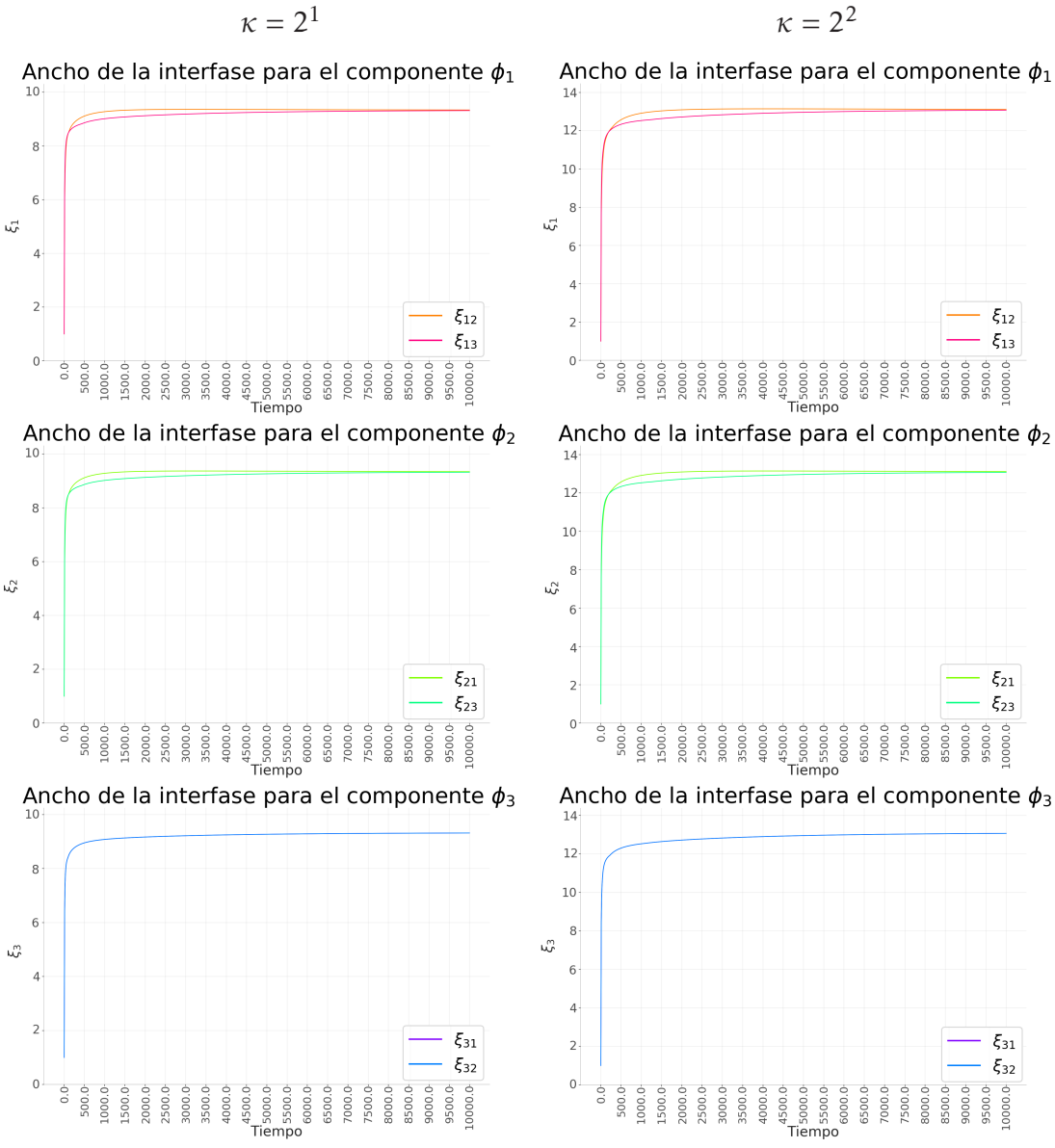


Figura 3.31: Espesor de Interfase para $\kappa = 2^1$ (izquierda) y $\kappa = 2^2$ (derecha).

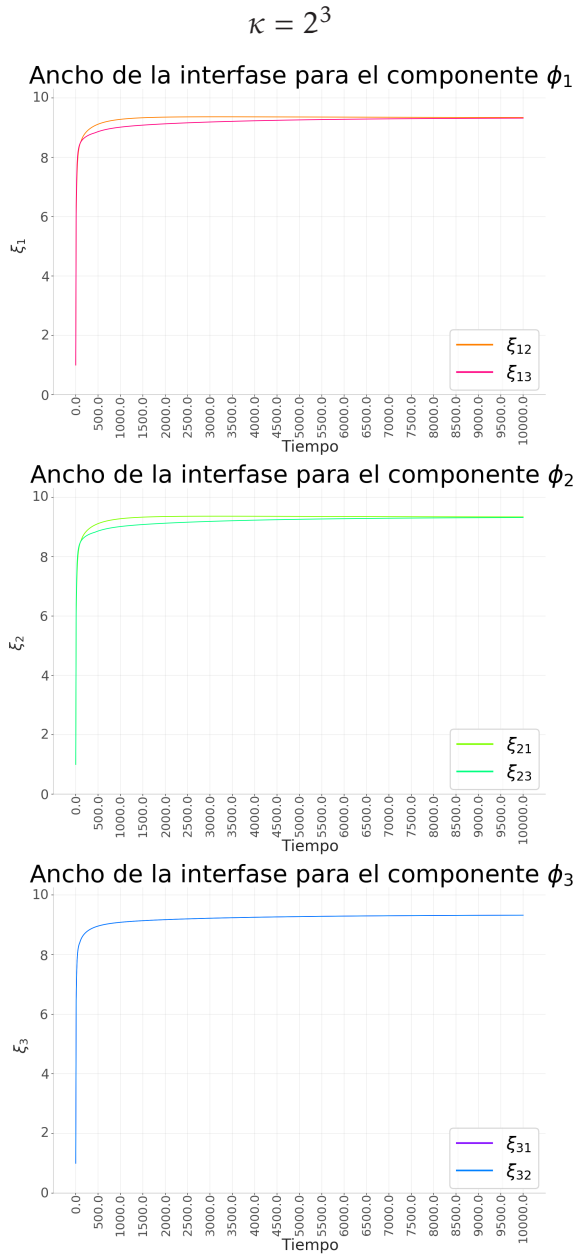


Figura 3.32: Espesor de Interfase para $\kappa = 2^3$.

Para explicar la discrepancia entre los valores del ancho de interfase en el régimen transitorio se proponen dos posibilidades, *(i)* El parámetro de partición c es muy elevado y la parte que se añadió generó una dinámica errónea *(ii)* La malla numérica no es suficientemente fina[21] y es necesario disminuir Δt. Para determinar si alguna de estas opciones es correcta, es necesario probar si disminuir estos parámetros disminuye el error.

Las pruebas para determinar el efecto de los parámetros en la discrepancia se hicieron comenzando por el que tiene menor impacto a la implementación numérica.

[21] Esto puede explicar que una mayor κ presenta mayor discrepancia, ya que esto hace que el lado derecho de la ecuación sea mayor y en consecuencia también $\Delta\phi$, pero Δt y Δx son los mismos. Esto ocasiona que haya un mayor error en el método numérico.

Primero se quiso observar si el parámetro de partición influye en la discrepancia. Se corrieron simulaciones con $\Delta t = 1$ y $\Delta x = 1$ (manteniendo todos los demás parámetros de la Tab. 3.1) y se graficó la discrepancia en ξ_1. (que es igual a la discrepancia en ξ_2) dada por $|\xi_{12} - \xi_{13}|$. Se escogió c primero porque este sería el mejor de los casos, ya que el tiempo de cómputo va como $O(c^0) = O(1)$. Los resultados de estas simulaciones se muestran a continuación en la Figura 3.33.

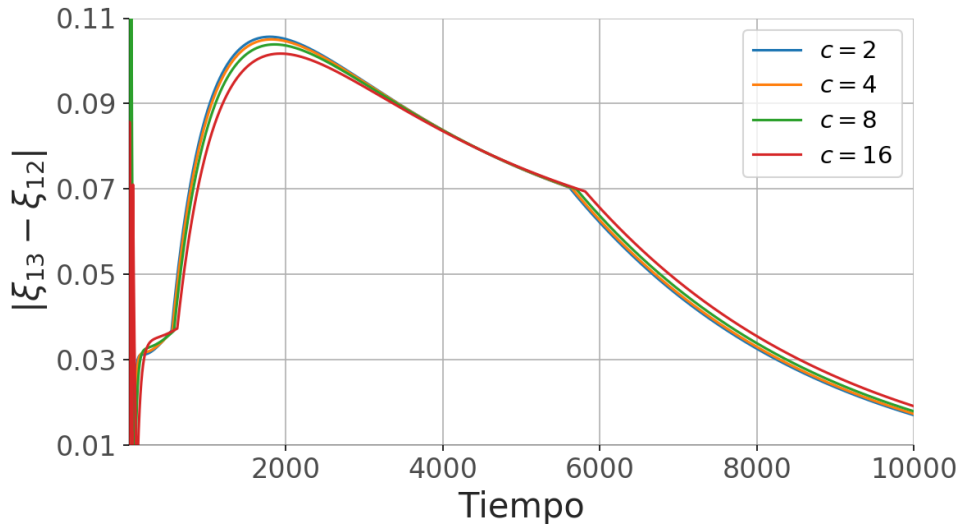


Figura 3.33: Gráficas de la discrepancia en el espesor de interfase ξ_1 para distintos valores del parámetro de partición c.

Estos resultados indican que el efecto de c en la discrepancia es mínimo. Un cambio en casi un orden de magnitud cambio la discrepancia es menos del 5%. Por esto se prosiguió a observar la dependencia en Δt, para el cual el impacto computacional es del órden $O(\Delta t^{-1})$. Las gráficas se ven en la Figura 3.34.

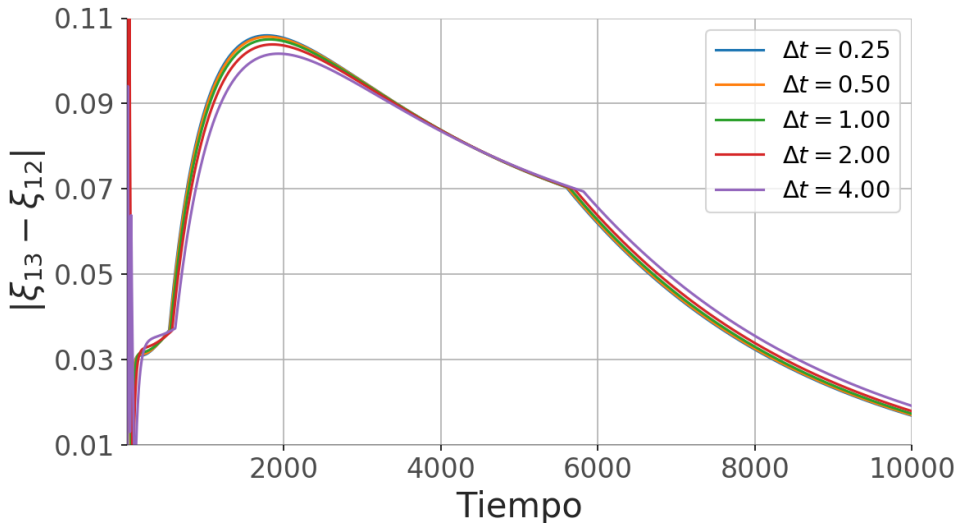


Figura 3.34: Gráficas de la discrepancia en el espesor de interfase ξ_1 para distintos valores de Δt, en el mallado temporal.

Al igual que como se observó para c, el efecto obtenido al variar Δt es pequeño. Se procede a evaluar la opción computacionalmente más pesada de reducir Δx, para la cual el tiempo de cómputo va como $O(\Delta x^{-2})$. Los resultados se muestran en la Figura 3.35.

Se observa que aumentar Δx claramente aumenta la discrepancia, sin embargo al disminuirla se observa un comportamiento más complejo: Hay un crecimiento inicial un poco mayor (hay una diferencia de alrededor de 10 % entre el máximo de la curva para $\Delta x = 1$ y la de 0.25) pero posteriormente disminuye más rápido la discrepancia para mayor Δx.

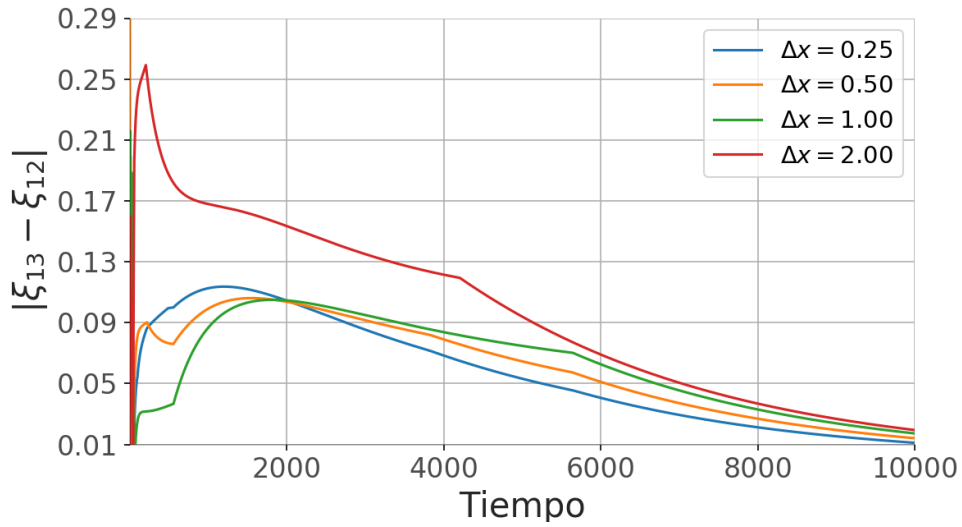


Figura 3.35: Gráficas de la discrepancia en el espesor de interfase ξ_1 para distintos valores de Δx de la malla espacial.

Estos resultados muestran que sólo disminuir Δx no es suficiente para eliminar la discrepancia, pero si se observa cuidadosamente la Fig. 3.34 se aprecia que el crecimiento inicial en la discrepancia no es tan rápido (el "hombro" a la izquierda del máximo se reduce al reducir Δt). Esto parecería indicar que reducir conjuntamente las dos deltas podría corregir la discrepancia. Sin embargo, un estudio de convergencia esta fuera del alcance de este documento, tanto por espacio como enfoque. Nos conformaremos con mencionar que el modelo numérico, por ser consistente y estable, debe ser convergente[22] (por lo que $\lim_{\Delta x, \Delta t \to 0} |\xi_{12} - \xi_{13}| = 0$).

[22] De acuerdo a lo que se muestra en Linz, 1979, una aproximación a una ecuación de operadores no–lineal es convergente si y sólo si es estable y consistente, y el orden de convergencia es mayor o igual al orden de consistencia. Este resultado es aplicable a ecuaciones de operadores en espacios de Banach, lo podemos traducir a nuestro caso convirtiendo a la ecuación diferencial en una ecuación de operadores no–lineal con una Transformada de Fourier en 4 dimensiones $(t, \mathbf{x} \mapsto \omega, \mathbf{k})$.

Un aspecto adicional que cabe mencionar es que en las Figs. 3.33-35 se presenta un cambio en la pendiente de la curva alrededor de $\tau \approx 5000$, sin embargo este cambio desaparece para los valores de Δx menores. Esto parece sugerir que el método pierde algo de precisión por no tener suficientes nodos en la malla (recordando que el cálculo de una cuarta derivada requiere al menos cinco nodos). Esto resulta interesante por que en estas simulaciones se tiene que $\xi \approx 6.63$ corresponde al número de nodos que abarca la región interfacial (para $\Delta x = 1$), lo cual debería ser suficiente para obtener un cálculo efectivo (si bien rudimentario) de la derivada. Para verificar si este puede ser un problema, se evalúa el error en la cuarta para un caso muy sencillo en la Figura 3.36.

Como se indica en la Fig. 3.36, el error en la cuarta derivada puede llegar a ser significativo y bien podría explicar el problema encontrado en las gráficas anteriores. Habiendo dicho esto, cabe recalcar que en las simulaciones se utilizan métodos espectrales para calcular las derivadas, los cuales tienen exactitud superior a la de las diferencias finitas usadas en la referencia (esto fue mencionado brevemente en la nota al pie 3 del Capítulo 2, pp. 44).

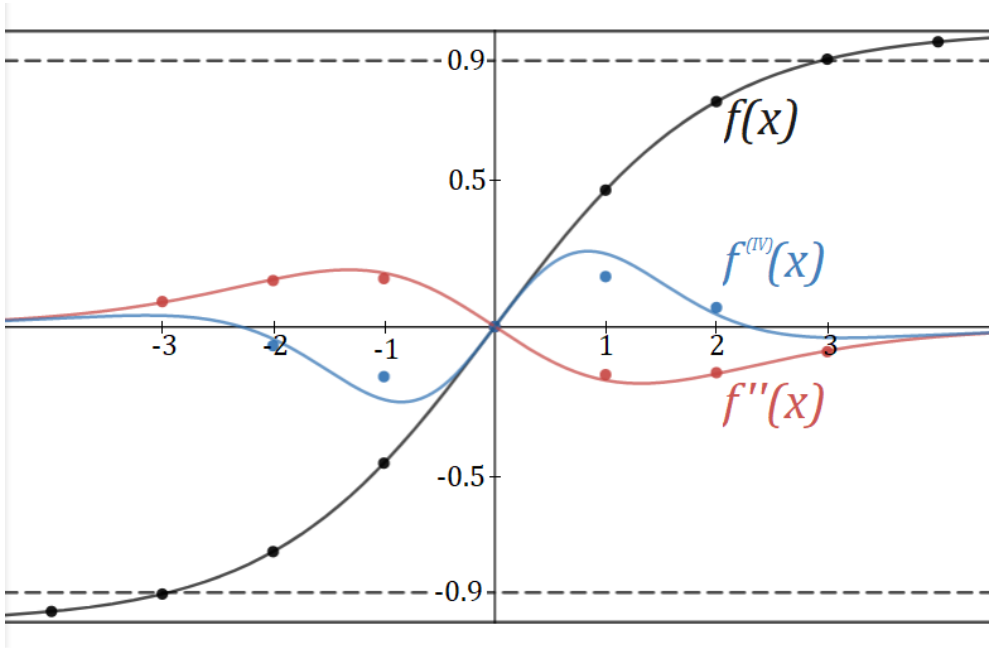


Figura 3.36: Usando el graficador Desmos se generó la gráfica de la función $f(x) = \tanh(x/2)$ y se tabularon los puntos usando $\Delta x = 1$. Se utiliza la función tangente hiperbólica por ser la forma de equilibrio prototípica de una interfase en una dimensión y con el argumento que se escogió se tiene $\xi \approx 5.89$. Posteriormente se aplicó la segunda derivada dos veces tanto para la gráfica "exacta" como para los datos tabulados (utilizando D^2 de la Tab. 1.1 de forma sucesiva). La segunda derivada tiene un error razonable, que no excede el 10.5%. Sin embargo, para la cuarta derivada se tienen valores de 31.4% y 55.2% en los errores no triviales (el error relativo para los puntos que se anulan está indeterminado y diverge para cualquier discrepancia). Para el caso que se analizó de $\xi \approx 6.63$ y $\Delta x = 1$ se tendría que esperar un error similar a éste y, por lo que se observa aquí, en contraste con lo que se supuso anteriormente, puede ser significativo.

Capítulo 4

Conclusiones

Finalmente, se concluye este proyecto con una discusión sobre los resultados obtenidos en términos del objetivo planteado inicialmente.

Se logró implementar exitosamente el modelo numérico del sistema ternario con interfase difusa. Esto permitió estudiar las propiedades del sistema, sujeto a cambios en algunos de los parámetros del sistema y comparándolos con resultados experimentales donde se tuvo la oportunidad. Como una primera aproximación al modelado de un material ternario, el presente trabajo resulta satisfactorio, sin embargo cabe aclarar que se hicieron muchas simplificaciones para llegar a él. Un trabajo futuro podría contemplar la adición de una energía libre de bulto más realista (por ejemplo usando entalpías de mezclado distintas, o un modelo que no sea mezcla regular) para aproximar el diagrama ternario de una aleación real; una movilidad variable, obtenida fenomenológicamente, que permita emular la dinámica de formación de fases (o solidificación) de un material real;

el acoplamiento con las ecuaciones de movimiento para analizar fenómenos más complejos como dinámica de gotas o propiedades mecánicas.

En cuanto a los resultados se observó una coincidencia razonable con lo que se esperaría en un estudio experimental: hay minimización de superficies, como se esperaría en presencia tensión superficial; el crecimiento de los dominios se va frenando a medida que evoluciona el sistema (Fiocco, 2012, menciona una tendencia de $\sim t^{1/4}$, aunque esto sólo sería válido para dominios mucho más pequeños que la región en estudio); se encontraron las regiones del diagrama ternario en las que el sistema presenta distintos tipos de separación y estas corresponden con lo que se esperaría de una generalización del modelo de Cahn–Hilliard; la dependencia del ancho de interfase con κ fue la misma que la de un sistema binario, en retrospectiva esto tiene sentido ya que $\mathcal{F}$ mantuvo la misma forma para la energía interfacial (una para cada fase) mientras que a la energía de bulto no se le añadió ningún término de interacción entre las tres substancias (la parte $\phi_i \ln(\phi_i)$ es un término de entropía y la interacción entre los componentes se manifiesta en los términos de tipo $\phi_i \phi_j$. Como se observa en 1.9 se consideró a la interacción total tan sólo como la suma de las interacciones binarias); la solubilidad observada de una fase en una mezcla de otras dos parece disminuir cuando se acercan las concentraciones de las fases de la mezcla, este tipo de comportamiento sí

se llega a apreciar en algunos sistemas pero en otros se observa el comportamiento contrario, donde la solubilidad de C en A aumenta sí se le mezcla B (por ejemplo SiO_2–CaO–Fe_2O_3 presenta ambos comportamientos en la solubilidad para distintas regiones de su diagrama ternario isotérmico a 1623K), esta limitante posiblemente se debe a la forma tan sencilla que se utilizó para $f(\phi)$, la cual es insuficiente para modelar sistemas reales más complejos.

En cuanto al desempeño de la implementación numérica, se obtuvo un programa razonablemente eficiente. No se realizó propiamente un benchmarking (por que estaría fuera del enfoque de este estudio) pero se observó que el tiempo de cómputo utilizado en la solución de las ecuaciones fue del mismo orden que el usado por una implementación en FORTRAN de las mismas ecuaciones utilizadas en el grupo de trabajo de Reología Óptica[1]. En cuanto a la parte de análisis numérico, en particular la convergencia del método, se observó que modificar los parámetros de la malla puede disminuir el error en los límites de t grande

[1]Como comparación, para una malla de 256×256 y 3100 pasos de tiempo con condiciones iniciales aleatorias, el programa en FORTRAN tomó 7 minutos con 31 segundos en tiempo de reloj, mientras que el programa de Python tardó 13 minutos con 58 segundos (de adaptó para que ambos programas exportaran los datos en crudo a documentos de texto, el de FORTRAN a `.dat` y el de Python a `.csv`, con el mismo acomodo). Lo que realmente requirió de la mayoría del tiempo de cómputo fue el procesamiento de los resultados en gráficas (aumentando el tiempo de algunos minutos a varias horas para cada sistema), especialmente la generación de diagramas ternarios la cuál se hizo con la paquetería `python-ternary`. Este paquete se utilizó por simplicidad, para evitar tener que implementarlo manualmente, sin embargo por cómo está escrito (usando estructuras de datos de tipo `dict` en lugar de `np.ndarray`) no fue posible optimizarlo como se hizo con el resto del código.

(reduciendo Δx) y también de t pequeña (reduciendo Δt), pero la no–linealidad de las ecuaciones hace de éste un problema complejo que no se pudo abordar por completo.

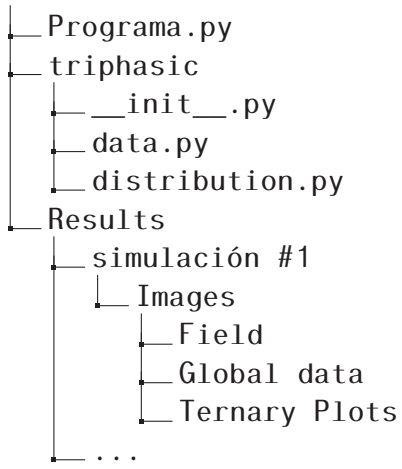
En resumen, se creó un programa capaz de simular sistemas ternarios con interfase difusa cuyas predicciones —salvo por algunas limitantes ocasionadas por las simplificaciones al modelo— se alinean con resultados existentes tanto teóricos como experimentales así como lo que se esperaría de una generalización de un modelo de campo de fase. Entonces, el producto de este trabajo puede ser utilizado como un componente en la simulación de sistemas más complejos, donde sea pertinente acoplar un fenómeno de difusión no–lineal o donde se desee tratar a las interacciones interfaciales de forma volumétrica, para poder incluirlas más naturalmente a las ecuaciones de movimiento de medios continuos.

Anexos

Anexo A

Códigos

En esta sección se incluyen los códigos en Python utilizados. Hay cuatro programas: el programa principal, que implementa la solución de las ecuaciones; dos programas de apoyo, que generan imágenes, gráficas y diagramas ternarios; y un programa `__init__.py` que permite llamar a los programas de apoyo como subrutinas. El árbol de directorios fue:



A.1. Programa principal

El código utilizado para la solución de las ecuaciones. En la versión que se incluye aquí, se itera sobre distintas concentraciones iniciales con una fluctuación aleatoria buscando caracterizar el comportamiento del sistema sobre todo el diagrama ternario en incrementos de 10 % (*viz.*, la concentración incial está dada por $\boldsymbol{\phi_0} \in \left\{ (\frac{n}{10}, \frac{m}{10}, \frac{l}{10}) \middle| \, n, m, l \in \mathbb{Z}^+, \; n+m+l = 10 \right\}$).

```
# -*- coding: utf-8 -*-
"""
Archivo: Programa.py
Autor: Salvador Villarreal. Basado en los programas de: Tavakoli, R. (2016). Unconditionally energy stable time stepping
       scheme for Cahn-Morral equation: Application to multi-component spinodal decomposition and optimal space tiling.
       Journal of Computational Physics, 304, 441-464. doi:10.1016/j.jcp.2015.10.018
Última edición: 14 de Abril de 2020.
Descripción: Este programa resuelve la ecuación de Cahn-Morral usando un método de Fourier en el espacio y un esquema de
             diferencias finitas a primer orden en el tiempo. Los resultados son graficados directamente desde el pro-
             grama durante la ejecución del método y se guardan en el directorio donde se ubica el código.
Uso: Es necesario ajustar las condiciones iniciales y los directorios manualmente. Hay dos tipos de codición inicial que
     se pueden usar, para usar una hay que dejar la otra como un comentario en bloque. En los métodos fftw_ e ifftw_ se
     usan 12 procesadores lógicos para una computadora con 6 núcleos físicos, llamado hyperthreading, acelerando al
     programa. Este número deberá modificarse si se desea correr en otro equipo. Ver: https://medium.com/data-design/
     destroying-the-myth-of-number-of-threads-number-of-physical-cores-762ad3919880
"""
from pylab import *
from PIL import Image
import numpy as np
import random as rand
from numba import jit, njit, int64, float64, complex128, prange  # Python -> código máquina para acelerar la ejecución
import pyfftw  # Wrapper de FFTW para Python, necesario por que Numba no es compatible con numpy.fft
import triphasic  # Paquete creado por el autor con el que se crean las figuras de los resultados
import os
import errno
```

```
26 import warnings as warn
27 from itertools import permutations
28
29
30 ################################################
31 # - F U N C I O N E S   D E   A P O Y O - #
32 ################################################
33
34 # - Transformadas de Fourier - #
35 def fftw_(y: np.ndarray) -> np.ndarray:
36     a = pyfftw.empty_aligned((N, N), dtype=np.float64)
37     b = pyfftw.empty_aligned((N, N//2+1), dtype=np.complex128)
38     fft_object = pyfftw.FFTW(a, b, axes=(0, 1), direction='FFTW_FORWARD', flags=('FFTW_PATIENT',), threads=12)
39     y_hat = fft_object(y)
40     return y_hat
41
42
43 def ifftw_(y_hat: np.ndarray) -> np.ndarray:
44     a = pyfftw.empty_aligned((N, N//2+1), dtype=np.complex128)
45     b = pyfftw.empty_aligned((N, N), dtype=np.float64)
46     fft_object = pyfftw.FFTW(a, b, axes=(0, 1), direction='FFTW_BACKWARD', flags=('FFTW_PATIENT',), threads=12)
47     y = fft_object(y_hat)
48     return y
49
50
51 # - Condición inicial - #
52 @njit
53 def cond_in(mesh_p: np.ndarray, random_p: np.ndarray) -> np.ndarray:
54     ini = np.zeros((3, N, N))
55     nn2 = N**2
56     nn2_inv = 1/nn2
57     for i in prange(nn2):
58         if 0 <= i*nn2_inv < 0.33333:
59             ini[0][mesh_p[i][0]][mesh_p[i][1]] = random_p[0][mesh_p[i][0]][mesh_p[i][1]]
60             ini[1][mesh_p[i][0]][mesh_p[i][1]] = random_p[1][mesh_p[i][0]][mesh_p[i][1]]
61             ini[2][mesh_p[i][0]][mesh_p[i][1]] = -ini[0][mesh_p[i][0]][mesh_p[i][1]]-ini[1][mesh_p[i][0]][mesh_p[i][1]]
```

```
62          elif 0.33333 <= i*nn2_inv < 0.66667:
63              ini[1][mesh_p[i][0]][mesh_p[i][1]] = random_p[0][mesh_p[i][0]][mesh_p[i][1]]
64              ini[2][mesh_p[i][0]][mesh_p[i][1]] = random_p[1][mesh_p[i][0]][mesh_p[i][1]]
65              ini[0][mesh_p[i][0]][mesh_p[i][1]] = -ini[1][mesh_p[i][0]][mesh_p[i][1]]-ini[2][mesh_p[i][0]][mesh_p[i][1]]
66          elif 0.66667 <= i*nn2_inv < 1:
67              ini[2][mesh_p[i][0]][mesh_p[i][1]] = random_p[0][mesh_p[i][0]][mesh_p[i][1]]
68              ini[0][mesh_p[i][0]][mesh_p[i][1]] = random_p[1][mesh_p[i][0]][mesh_p[i][1]]
69              ini[1][mesh_p[i][0]][mesh_p[i][1]] = -ini[2][mesh_p[i][0]][mesh_p[i][1]]-ini[0][mesh_p[i][0]][mesh_p[i][1]]
70      return ini
71
72
73  # - Función para guardar valores de campo máximos y mínimos - #
74  def maxmin(phi_max: np.ndarray, phi_min: np.ndarray, phi: np.ndarray):
75      for i in range(3):
76          phi_max[i] = np.append(phi_max[i], np.max(phi[i]))
77          phi_min[i] = np.append(phi_min[i], np.min(phi[i]))
78      return phi_max, phi_min
79
80
81  # - Derivadas del potencial F - #
82  @njit((float64[:, :, :], int64, float64, float64))
83  def fi(phi: np.ndarray, i: int, theta: float, theta_c: float) -> np.ndarray:  # Suma de derivadas del potencial
84      return theta*(np.log(phi[i]/phi[2]))+theta_c*(phi[2]-phi[i])  # log(x/y) es más rápido(39%) y varía menos(6x)
85
86
87  ##############################################################################################
88  # - F U N C I O N E S   P A R A   R E S O L V E R   L A S   E C U A C I O N E S - #
89  ##############################################################################################
90
91  # - Actualizar el lado derecho de las ecuaciones - #
92  def act_r(phi_hat: np.ndarray, phi: np.ndarray, theta: float, theta_c: float) -> np.ndarray:  # Actualizar lado derecho
93      ret = np.zeros((2, N, N//2+1), dtype=np.complex128)
94      for i in range(2):
95          ret[i] = phi_hat[i]+dt*(De*(phi_hat[i]-phi_hat[2])+Lap*fftw_(fi(phi, i, theta, theta_c)))
96      return ret
97
```

```
98
99  # - Actualizar los campos de fase - #
100 @njit((complex128[:, :, :], int64, complex128[:, :], complex128[:, :], complex128[:, :], complex128[:, :]))
101 def act_phi(rhs: np.ndarray, m: int, s: np.ndarray, a: np.ndarray, b: np.ndarray, unos: np.ndarray) -> np.ndarray:
102     phi = np.zeros((3, m, m//2+1), dtype=np.complex128)
103     phi[2] = s*(unos-a*(rhs[0]+rhs[1]))  # Método de Schur: Frontera.
104     for i in range(2):  # Método de Schur: Dominios.
105         phi[i] = a*(rhs[i]-b*phi[2])
106     return phi
107
108
109 ##############################################################################################
110 # - P A R Á M E T R O S   A J U S T A B L E S   P O R   E L   U S U A R I O - #
111 ##############################################################################################
112
113 # - Parámetros de la malla - #
114 N = 1536  # Numero de nodos de la malla.
115 T = 10000  # Numero de pasos de tiempo (PdT).
116 dx = 0.25  # Distancia entre nodos.
117 dt = 1  # Tiempo entre PdT.
118 dat, img, csec = 1, 1, 1  # Numero de PdT para adquisición de Datos e Imágenes.
119
120 # - Parámetros para generar diagramas ternarios - #
121 n_ter = -1  # (Número de diagramas ternarios a generar)-1
122 ter_step = ((T/5)*(4*(np.arange(n_ter+1)/n_ter)**4+(np.arange(n_ter+1)/n_ter))).astype(int)  # Distorsionado para
123 # tener pasos largos en tiempos grandes (hay menor variación en el sistema al acercarse al estado estacionario)
124
125 # - Parámetros del sistema - #
126 kap = 1  # Penalización de interfase.
127 thet, thet_c = 0.3, 1.
128 c, alpha, beta = 4., 1., 0.  # Parámetros de partición.
129 fluc = 0.05  # Amplitud de fluctuaciones de la Composición inicial.
130
131 # - Ruta de los directorios donde se guardarán los resultados - #
132 dirs = ('Images/Field/', 'Images/Ternary Plots/', 'Images/Global data/', 'Images/Cross Section/')
133
```

```
134
135 ##########################################################################################################
136 # - A R R E G L O S    P A R A    O P E R A D O R E S    D E    L A S    E C U A C I O N E S - #
137 ##########################################################################################################
138 kx, ky = np.meshgrid(np.fft.rfftfreq(N, dx)+0.j, np.fft.fftfreq(N, dx)+0.j)  # Componentes del vector de onda.
139 Lap = -4*np.pi**2*(kx*kx+ky*ky)  # Operador para el Laplaciano.
140 Bih = Lap**2  # Operador para el bi-armónico
141 uno = fftw_(np.ones((N, N), dtype=np.float64))  # Transformada de "unos" (que se suman en todas las celdas)
142
143 Dc = (kap+c*beta)*Bih-c*alpha*Lap
144 De = c*beta*Bih-c*alpha*Lap
145 A_inv = 1./(1.+dt*Dc)  # Operador A^{-1} (no se utiliza "A", únicamente su inversa)
146 B = -dt*Dc  # Operador B.
147 S_inv = 1./(1.-2.*A_inv*B)  # Operador para S^{-1}, la inversa del complemento de Schur (no se utiliza "S")
148
149 '''
150 ###############################################################################
151 # - C O N D I C I O N E S    I N I C I A L E S    A L E A T O R I A S - #
152 ###############################################################################
153
154 # - Concentraciones iniciales promedio - #
155 phi0 = []  # Lista de las concentraciones iniciales con las que se va a experimentar
156 # NOTA: phi0[i]>fluc para evitar que salga del rango.
157 for i in range(1, 10):
158     for j in range(1, 10-i):
159         phi0.append(np.around([i/10, j/10, 1-i/10-j/10], decimals=1))  # Valores desde 0.1 hasta 0.8 en todos los campos
160
161
162 # - Ciclo sobre las concentraciones - #
163 for p0 in phi0:
164
165     # - Crear un directorio individual y las rutas para los resultados con concentraciones p0[0], p0[1] y p0[2] - #
166     p10 = [int(10*p0[0]), int(10*p0[1]), int(10*p0[2])]
167     path = 'Results/({:d}, {:d}, {:d})'.format(p10[0], p10[1], p10[2])
168     for d in dirs:
169         if d == dirs[3]:  # No se crea directorio para las gráficas de sección transversal por que no se van a generar
```

```
            continue
        try:  # Intenta crear directorio si este no existe
            os.makedirs(path+'{:s}'.format(d))
        except OSError as e:  # Si el directorio existe, se omite la creación
            if e.errno != errno.EEXIST:
                raise

    # - Generar condiciones iniciales aleatorias - #
    p = np.arange(N)
    p = np.c_[np.repeat(p, N), np.tile(p, N)]
    p = np.asarray(rand.sample(list(p), N**2))
    rd = [np.random.uniform(low=-fluc, high=fluc, size=(N, N)), np.random.uniform(low=-fluc, high=fluc, size=(N, N))]
    rd = np.array([0.5*(rd[0]+rd[1]), 0.5*(rd[1]-rd[0])])  # Rotación de 45° y escalamiento de 1/sqrt(2)
    ph = np.array([p0[0]*np.ones((N, N)), p0[1]*np.ones((N, N)), p0[2]*np.ones((N, N))],dtype=np.float64)+cond_in(p, rd)
    ph = np.where(ph < 1e-40, 1e-40, ph)

    ph_fou = np.zeros((3, N, N//2+1), dtype=np.complex128)
    ph_fou[0], ph_fou[1], ph_fou[2] = fftw_(ph[0]), fftw_(ph[1]), fftw_(ph[2])  # El campo de fase en Fourier.
    r = np.zeros((2, N, N//2+1), dtype=np.complex128)  # Lado derecho de las ecuaciones.

    # - Generar gráficas y exportar datos iniciales - #
    triphasic.gen_img(ph, 0, N=N, path=path+dirs[0])
    ener = np.array([triphasic.energy(ph, dx, N=N, thet=thet, thet_c=thet_c)])
    ph_max = [np.max(ph[0]), np.max(ph[1]), np.max(ph[2])]
    ph_min = [np.min(ph[0]), np.min(ph[1]), np.min(ph[2])]
    if 0 in ter_step:
        triphasic.gen_dist(ph, 0, r=50, path=path+dirs[1])

    # - Ciclo principal - #
    for n in range(1, T+1):  # Se inicia en el PdT #1 (el #0 es Cond. Inicial).
        r = act_r(ph_fou, ph, thet, thet_c)
        ph_fou = act_phi(r, N, S_inv, A_inv, B, uno)
        ph[0], ph[1] = ifftw_(ph_fou[0]), ifftw_(ph_fou[1])  # Regresa al espacio de las posiciones.
        ph[2] = 1-ph[0]-ph[1]  # Aplica condición de conservación.

        ph_fou[0], ph_fou[1], ph_fou[2] = fftw_(ph[0]), fftw_(ph[1]), fftw_(ph[2])
```

```
206
207         if mod(n, dat) == 0:  # Guardar datos
208             ener = np.append(ener, triphasic.energy(ph, dx, N=N, thet=thet, thet_c=thet_c))
209             ph_max, ph_min = maxmin(ph_max, ph_min, ph)
210         if mod(n, img) == 0:  # Generar imagen(es)
211             triphasic.gen_img(ph, n, N=N, path=path+dirs[0])
212         if n in ter_step:  # Generar diagrama ternario
213             triphasic.gen_dist(ph, n, r=50, path=path+dirs[1])
214
215     # - Generar gráficas - #
216     triphasic.plots(ener, ph_max, ph_min, dat, dt, path=path+dirs[2])
217
218     # - Crear GIF animado de las imágenes del campo - #
219     n_f = [path+dirs[0]+'/t{:d}.tiff'.format(i) for i in range(0, T+1, 20)]
220     n_f.append(path+dirs[0]+'/t{:d}.tiff'.format(T))  # Repite el últmo cuadro de la animación
221     triphasic.animation(path=path+dirs[2], names=n_f, filename='Animado_campo.gif')
222
223     # - Crear GIF animado de los diagramas ternarios - #
224     names_ter = []
225     for x in ter_step:
226         names_ter.append(path+dirs[1]+'/Dist{:04d}.tiff'.format(x))
227     triphasic.animation(path=path+dirs[2], names=names_ter, filename='Animado_ternario.gif')
228 '''
229
230 ##########################################################################
231 # C O N D I C I O N E S   I N I C I A L E S   E N   F R A N J A S #
232 ##########################################################################
233
234 lin_trans_init = 2  # Numero de puntos para interpolar la frontera de la condición inicial (sólo se implementó 1 y 2)
235
236 # - Ruta para guardar resultados - #
237 path = 'Results/'
238
239 # - Generar condiciones iniciales - #
240 phi_low = 0.055053256  # Valor "bajo" de los campos de fase en los mínimos del potencial
241 phi_high = 0.889893488  # Valor "alto" de los campos de fase en los mínimos del potencial
```

```
242 if N % 3 != 0:
243     warn.warn("No se puede usar la condición inicial de franjas por que el número de nodos (N) no es múltiplo de 3")
244     print("El valor de N será reemplazado por el siguiente múltiplo de 3")
245     N += 3-(N % 3)
246 Np = N//3
247 
248 perm = list(permutations((0, 1, 2)))[2]  # Permutaciones de (1, 2, 3) para cambiar el orden de las franjas
249 #  [0] -> (0, 1, 2), [1] -> (0, 2, 1), [2] -> (1, 0, 2), [3] -> (1, 2, 0), [4] -> (2, 0, 1), [5] -> (2, 1, 0),
250 
251 ph = np.zeros((3, N, N))
252 for a in range(3):
253     ap = perm[a]  # Se usa la permutación para cambiar el orden de las franjas
254     for i in range(ap*Np, (ap+1)*Np):
255         for j in range(N):
256             ph[a][i][j] = phi_high
257             for b in range(3):
258                 if b == a:
259                     continue
260                 ph[b][i][j] = phi_low
261 
262 if lin_trans_init == 1:
263     for a in range(3):
264         for i in (x*Np-1 for x in range(1, 4)):
265             for j in range(N):
266                 ph[a][i][j] = 0.5*(ph[a][i-1][j]+ph[a][mod(i+1, N)][j])
267 
268 if lin_trans_init == 2:
269     for a in range(3):
270         for i in (x * Np - 2 for x in range(1, 4)):
271             for j in range(N):
272                 ph[a][i][j] = 0.5 * (ph[a][i][j] + ph[a][mod(i + 2, N)][j])
273         for i in (x * Np - 1 for x in range(1, 4)):
274             for j in range(N):
275                 ph[a][i][j] = 0.5 * (ph[a][i - 1][j] + ph[a][mod(i + 1, N)][j])
276         for i in (x * Np - 3 for x in range(1, 4)):
277             for j in range(N):
```

```
278                 ph[a][i][j] = 0.5 * (ph[a][i - 1][j] + ph[a][mod(i + 1, N)][j])
279
280 ph_fou = np.zeros((3, N, N//2+1), dtype=np.complex128)
281 ph_fou[0], ph_fou[1], ph_fou[2] = fftw_(ph[0]), fftw_(ph[1]), fftw_(ph[2])  # El campo de fase en Fourier.
282 r = np.zeros((2, N, N//2+1), dtype=np.complex128)  # Lado derecho de las ecuaciones.
283
284 # - Crear los directorios donde se guardarán los resultados - #
285 for d in dirs:
286     try:  # Intenta crear directorio si este no existe
287         os.makedirs(path+'{:s}'.format(d))
288     except OSError as e:  # Si el directorio existe, se omite la creación
289         if e.errno != errno.EEXIST:
290             raise
291
292 # - Generar gráficas y exportar datos iniciales - #
293 triphasic.gen_img(ph, 0, N=N, path=path+dirs[0])
294 i_len = triphasic.cross_sec(ph, phi_high, phi_low, 0, N=N, path=path+dirs[3], perm=perm, dx=dx)
295 inter_len = [[i_len[0][0], i_len[0][1], i_len[0][2]],
296              [i_len[1][0], i_len[1][1], i_len[1][2]],
297              [i_len[2][0], i_len[2][1], i_len[2][2]]]  # Se crea como lista en lugar de ndarray para poder añadir datos
298 len_diff = abs(i_len[0][1]-i_len[0][2])  # La diferencia entre los espesores de interfase
299 print(abs(i_len[0][1]-i_len[0][2]))
300 ener = np.array([triphasic.energy(ph, dx, N=N, thet=thet, thet_c=thet_c)])
301 ph_max = [np.max(ph[0]), np.max(ph[1]), np.max(ph[2])]
302 ph_min = [np.min(ph[0]), np.min(ph[1]), np.min(ph[2])]
303 if 0 in ter_step:
304     triphasic.gen_dist(ph, 0, r=50, path=path+dirs[1])
305
306
307 # - Ciclo Principal - #
308 for n in range(1, T+1):  # Se inicia en el PdT #1 (el #0 es Cond. Inicial).
309     r = act_r(ph_fou, ph, thet, thet_c)
310     ph_fou = act_phi(r, N, S_inv, A_inv, B, uno)
311     ph[0], ph[1] = ifftw_(ph_fou[0]), ifftw_(ph_fou[1])  # Regresa al espacio de las posiciones.
312     ph[2] = 1-ph[0]-ph[1]  # Aplica condición de conservación.
313
```

```
314     ph_fou[0], ph_fou[1], ph_fou[2] = fftw_(ph[0]), fftw_(ph[1]), fftw_(ph[2])
315 
316     if mod(n, dat) == 0:  # Guardar datos
317         ener = np.append(ener, triphasic.energy(ph, dx, N=N, thet=thet, thet_c=thet_c))
318         ph_max, ph_min = maxmin(ph_max, ph_min, ph)
319     if mod(n, img) == 0:  # Generar imagen(es)
320         triphasic.gen_img(ph, n, N=N, path=path+dirs[0])
321     if n in ter_step:  # Generar diagrama ternario
322         triphasic.gen_dist(ph, n, r=50, path=path+dirs[1])
323     if mod(n, csec) == 0:
324         i_len = triphasic.cross_sec(ph, phi_high, phi_low, n, N=N, path=path+dirs[3], perm=perm, dx=dx)
325         for a in range(3):
326             for b in (c for c in range(3) if c != a):  # Se omiten los valores de "interfase entre 'a' y 'a'"
327                 inter_len[a][b] = np.append(inter_len[a][b], i_len[a][b])
328         len_diff = np.append(len_diff, abs(i_len[0][1]-i_len[0][2]))  # La diferencia entre los espesores de interfase
329         print(abs(i_len[0][1]-i_len[0][2]))
330 
331     # - Generar gráficas - #
332 triphasic.plots(ener, ph_max, ph_min, dat, dt, path=path+dirs[2])
333 triphasic.interph_plot(inter_len, csec, dt, path=path+dirs[2])
334 
335 np.savetxt('Results/interphase_thickness.txt', i_len, delimiter=' ')
336 np.savetxt('Results/len_diff.txt', len_diff, delimiter=' ')
337 
338 
339 print("FIN DE LA SIMULACIÓN")
```

A.2. El archivo __init__.py

Permite utilizar los programas de las secciones A.3 y A.4 como un módulo importado en el programa principal.

```
# -*- coding: utf-8 -*-
"""
Archivo: __init__.py
Autor: Salvador Villarreal.
Última edición:14 de Abril de 2020.
Descripción: Este programa se usa para poder llamar subrutinas de los archivos data.py y distribution.py como un módulo
             importado en el programa principal
Uso: Este programa no está diseñado para ser utilizado directamente por el usuario.
"""
import ternary
import numpy as np
from scipy.interpolate import griddata
from matplotlib import colors
import matplotlib.pyplot as plt
from numba import jit, njit, prange
from ternary.helpers import simplex_iterator
import warnings as warn

from .distribution import gen_dist
from .data import (energy, gen_img, plots, animation, cross_sec, interph_plot)

from operator import mod
```

A.3. Programa para diagramas ternarios

Código utilizado para generar diagramas como los de la sección...

```
"""
Archivo: distribution.py
Autor: Salvador Villarreal.
Última edición: 31 de Julio de 2019.
Descripción: Este programa contiene un método para generar diagramas ternario de distribución, el cual es utilizado por
             "Programa.py".
Uso: Este programa no está diseñado para ser utilizado directamente por el usuario.
"""
import ternary
import numpy as np
from scipy.interpolate import griddata
from matplotlib import colors
import matplotlib.pyplot as plt
from numba import njit  # Compilador "Just-in-time" para acelerar el llamado de funciones.
from ternary.helpers import simplex_iterator
# coding=utf-8


############################
# - S U B R U T I N A S - #
############################
def distort_cmap(cm, inv):
    """
    Función que genera un mapa de color no-lineal a partir de uno pre-existente (objeto cmap de matplotlib).

    cm: Mapa de color pre-existente.
    inv: Función de distorsión. Puede ser def or lambda.
    """

    cdict = {'red': [], 'green': [], 'blue': []}
```

```
31      i = 0
32      for rgb in cm.__dict__['colors']:  # Note that the encoding of colors can vary in different colormaps
33          cdict['red'].append((inv(i / 255), rgb[0], rgb[0]))
34          cdict['green'].append((inv(i / 255), rgb[1], rgb[1]))
35          cdict['blue'].append((inv(i / 255), rgb[2], rgb[2]))
36          i += 1
37
38      name = 'new_'+cm.__dict__['name']
39      return colors.LinearSegmentedColormap(name, cdict)
40
41
42  @njit
43  def retval(minv: float, maxv: float, val: float) -> float:
44      """
45      Método que determina si un valor "val" se encuentra entre "minv" y "maxv". Si el valor se encuentra al interior, se
46      cuenta como "1", si se encuentra en los extremos del intervalo se cuenta a la mitad (regresa 0.5).
47
48      minv: Límite inferior del intervalo.
49      maxv: Límite superior del intervalo.
50      val: Valor a "probar".
51      """
52      if minv < val < maxv:
53          return 1
54      elif minv == val or maxv == val:
55          if minv == 0 or maxv == 1:
56              return 1
57          else:
58              return 0.5
59      else:
60          return 0
61
62
63  @njit
64  def get_histogram_data(r: int, m: int, sr: float, hist_dat: np.ndarray, pha: np.ndarray, phb: np.ndarray) -> np.ndarray:
65      """
66      Genera los valores del histograma que se va a graficar. Cuenta cuántos puntos de la malla tienen concentraciones
```

```
    dentro de cierto rango de porcentajes (dividido entre el número total de puntos).
    NOTA: Los parámetros pha y phb se podrían importar también con pandas.

    r: Número de divisiones en cada dirección del histograma.
    m: Número de puntos de la malla en cada dirección.
    sr: Inverso del total de puntos de la malla (para 2D, sr = 1/(m**2)).
    hist_dat: Arreglo (vacío) de tamaño r**2 donde se guardan los datos del histograma.
    pha: Valores de \phi_{1} sobre la malla (obtenidos desde el programa principal).
    phb: Valores de \phi_{2} sobre la malla (obtenidos desde el programa principal).
    """
    count = 0
    for a in range(r):
        for b in range(r):
            for i in range(m):
                for j in range(m):
                    hist_dat[count] += retval(a/r, (a+1)/r, pha[i][j])*retval(b/r, (b+1)/r, phb[i][j])*sr
            count += 1
    return hist_dat


def hist_gen(g_z: np.ndarray, scale: int) -> dict:
    """
    Genera un diccionario que permite graficar los datos interpolados "g_z" sobre el "simplex" del diagrama ternario.

    g_z: Datos del histograma, interpolados para que el arreglo tenga el mismo tamaño que requiere el diagrama ternario.
    scale:  Número de puntos "por lado" que tendrá el diagrama ternario.
    """
    dic = dict()
    for (i, j, k) in simplex_iterator(scale):
        dic[(i, j)] = g_z[i][j]
    return dic


def rescale(y: float) -> float:
    """
    Función de distorsión para el mapa de color no-lineal. Debe ser un homeomorfismo f:[0, 1]->[0, 1]. En donde la fun-
```

```
103     ción sea más "plana", el color cambiará más rapidamente; y donde la función sea más "empinada" el color cambiará más
104     lentamente.
105
106     y: Variable de la función de distorsion utilizada para generar el cmap distorsionado.
107     """
108     return y*50**(y-1)
109
110
111 ##########################################
112 # - S U B R U T I N A    E X T E R N A - #
113 ##########################################
114 # Para llamar en el programa principal.  #
115 ##########################################
116 def gen_dist(phi, n, r=50, path=""):
117     """
118     Método que genera una gráfica de la distribucón de concentraciones como un diagrama ternario.
119
120     phi: numpy.ndarray o tuple que contiene los valores de concentracion sobre una malla cuadrada homogénea.
121     n: Número que se añadirá al nombre del archivo como identificador (e.g. el número del paso de tiempo).
122     r: Número de divisiones usadas para clasificar los datos en el hisograma (el número de "barras" en el histograma
123        hacia cada dirección).
124     path: Ruta del directorio donde se guardará la imagen. Si se deja vacío, la imágen se guardará en el directorio
125           donde se encuentra el programa
126     """
127     cmap = plt.get_cmap("plasma")
128     cmap = distort_cmap(cmap, inv=rescale)
129
130     scale = 100
131     grid_x, grid_y = np.mgrid[0:1:(scale+1)*(0+1j), 0:1:(scale+1)*(0+1j)]  # Se usa scale+1 porque el diagrama ternario
132     #                                                                        requiere indices desde 0 hasta "scale"
133
134     r_inv = 1/r
135     points = np.arange(0.5*r_inv, 1, r_inv)
136     points = np.c_[np.repeat(points, r), np.tile(points, r)]
137
138     # - Leer los datos (puede usarse la librería pandas para leerlo desde un archivo) - #
```

```
139     print("\nGenerando histograma para el ciclo {:04d}...".format(n))
140     pha = phi[0]
141     phb = phi[1]
142 
143     N = len(pha)
144     SR = 1/(N**2)
145 
146     # - Crear un histograma con los datos leídos - #
147     print("Generando histograma...")
148     hist_dat = np.zeros(r ** 2)
149     hist_dat = get_histogram_data(r, N, SR, hist_dat,  pha, phb)
150 
151     # - Interpolar el histograma al tamaño (número de puntos: "scale") del diagrama deseado - #
152     print("Interpolando...")
153     grid_z = griddata(points, hist_dat, (grid_x, grid_y), method='linear', fill_value=0)
154     d = hist_gen(grid_z, scale)
155 
156     # - Inicializar figura - #
157     figure, tax = ternary.figure(scale=scale)
158 
159     # - Borde y líneas guía - #
160     tax.boundary(linewidth=1.0)
161     tax.gridlines(color="black", multiple=scale/10)
162     tax.gridlines(color="blue", multiple=scale/50, linewidth=0.5)  # REVISAR: "multiple" debe ser entero
163 
164     # - Título de la gráfica y leyendas de los ejes - #
165     fszL = 16  # Tamaño de letra para las leyendas.
166     fszT = 22  # Tamaño de letra para el título.
167     fnm = "Cambria"  # Tipo de letra de las leyendas.
168     tax.left_axis_label("$\\phi_3$", fontsize=fszL, fontname=fnm, offset=0.15, rotation=0)
169     tax.right_axis_label("$\\phi_2$", fontsize=fszL, fontname=fnm, offset=0.17, rotation=0)
170     tax.bottom_axis_label("$\\phi_1$", fontsize=fszL, fontname=fnm, offset=0.02)
171     tax.set_title("Distribución de fases", fontsize=fszT)
172 
173     # - Etiquetas sobre los ejes - #
174     labels = [str(int(i*10))+"%" for i in range(11)]
```

```
175     tax.ticks(ticks=labels, axis="lbr", linewidth=1, tick_formats="{:s}")
176     tax.clear_matplotlib_ticks()  # Elimina los ejes predeterminados de matplotlib
177 
178     # - Eliminar el marco de matplotlib alrededor de la figura - #
179     ls = "top", "bottom", "left", "right"
180     for x in ls:
181         tax.ax.spines[x].set_visible(False)
182 
183     # - Generar el mapa-de-calor - #
184     print("Generando figura...")
185     tax.heatmap(d, style="triangular", cmap=cmap, scientific=False)
186 
187     # - Guardar el mapa-de-calor - #
188     print("Guardando figura...")
189     tax.savefig(path+"Dist{:04d}.tiff".format(n), format="tiff")
190 
191     # - Cerrar la figura para prevenir una fuga de memoria - #
192     tax.close()
193     print("_____________________")
```

A.4. Programa para imágenes y gráficas

Incluye métodos para crear las imágenes de los campos y las gráficas de la energía, mínimos y máximos.

```
# -*- coding: utf-8 -*-
"""
Archivo: data.py
Autor: Salvador Villarreal.
Última edición: 14 de Abril de 2020.
Descripción: Este programa contiene varios métodos para analizar los datos generados por el "Programa.py".
Uso: Este programa no está diseñado para ser utilizado directamente por el usuario.
"""
from PIL import Image
from numba import njit, prange
import numpy as np
import matplotlib.pyplot as plt
import warnings as warn
from operator import mod


############################
# - S U B R U T I N A S - #
############################
@njit
def grad(phi: np.ndarray, N:int) -> np.ndarray:
    """
    Calcula el gradiente de cada uno de los componentes de phi.

    phi: Campo de fase, es un objeto np.ndarray de (3, N, N).
    N: Número de puntos de la malla en cada dirección.
    """
    g_ph = np.zeros((3, 2, N, N))  # 3 componentes, 2 direcciones, malla de N por N.
    for a in prange(3):  # prange() es un objeto exclusivo de numba y es más rápido que usar range().
        for i in prange(N):
```

```
            g_ph[a][0][0][i] = phi[a][1][i]-phi[a][0][i]
        for i in prange(N):
            g_ph[a][1][i][0] = phi[a][i][1]-phi[a][i][0]
        for i in prange(N):
            g_ph[a][0][N-1][i] = phi[a][N-1][i]-phi[a][N-2][i]
        for i in prange(N):
            g_ph[a][1][i][N-1] = phi[a][i][N-1]-phi[a][i][N-2]
        for i in prange(N):
            for j in prange(1, N-1):
                g_ph[a][0][j][i] = (phi[a][j+1][i]-phi[a][j-1][i])*0.5
                g_ph[a][1][i][j] = (phi[a][i][j+1]-phi[a][i][j-1])*0.5
    return g_ph


@njit
def pot(phi: np.ndarray, thet: float=0.3, thet_c: float=1.) -> np.ndarray:
    """
    Calcula el potencial F (adentro de la integral de energía libre) sobre cada punto de la malla.

    phi: Campo de fase, es un objeto np.ndarray de (3, N, N).
    thet: Parámetro del potencial F, corresponde a la parte de la entropía.
    thet_c: Parámetro del potencial F, corresponde a una "penalización" por interfase.
    """
    return thet*np.sum(phi*np.log(phi), axis=0)+thet_c*(phi[0]*phi[1]+phi[1]*phi[2]+phi[2]*phi[0])


@njit
def energy(phi: np.ndarray, dx: float, N: int=0, thet=0.3, thet_c=1) -> float:
    """
    Calcula la energía libre de Helmholtz del sistema.

    phi: Campo de fase, es un objeto np.ndarray de (3, N, N).
    dx: Elemento (discreto) de línea, distancia entre nodos de la malla.
    N: Número de puntos de la malla en cada dirección.
    thet: Parámetro del potencial F, corresponde a la parte de la entropía.
    thet_c: Parámetro del potencial F, corresponde a una "penalización" por interfase.
```

```
67      """
68      dph = grad(phi, N=N)  # Se omite la división entre dx por que ésta se cancela con la dx**2 de la integral.
69      bulk = pot(phi, thet=thet, thet_c=thet_c)*dx**2  # Se multiplica por el elemento de área para no repetirlo al sumar.
70      return np.sum(bulk)+np.sum(dph**2)
71
72
73  def gen_img(phi: np.ndarray, n: int, N:int=0, path: str=""):
74      """
75      Genera imágenes del campo de fase, asignando a cada componente puro un color primario de la escala RGB: El componen-
76      te 1 en rojo, el 2 en verde y el 3 en azul.
77
78      phi: Campo de fase, es un objeto np.ndarray de (3, N, N).
79      n: Número de paso de tiempo, para nombrar al archivo generado.
80      N: Número de puntos de la malla en cada dirección.
81      path: Ruta donde se guardarán las gráficas.
82      """
83      im = Image.new('RGB', (N, N))
84      bmp = im.load()
85      for l in range(N):
86          for m in range(N):
87              bmp[l, m] = (int(abs(phi[0][l][m])*255), int(abs(phi[1][l][m])*255), int(abs(phi[2][l][m])*255))
88      im.save(path+"t"+str(n)+".tiff", format='tiff', compression='tiff_adobe_deflate')
89
90
91  def interph(phi_cs: np.ndarray, p95: float, p05: float, N: int):
92      """
93      Calcula el ancho de la interfase a graficar y, si se tienen varias interfases, también la desviación estándar de
94      dicho valor. Sólo funciona correctamente si todas las fases tienen un mismo valor en su campo de fase predominante.
95
96      phi_cs: Campo de fase a lo largo de la línea de corte, es un objeto np.ndarray de (3, N).
97      p95: Cota superior del campo de fase para que se considere a una región como interfase.
98      p05: Umbral del campo de fase para que se considere a una región como interfase.
99      N: Número de puntos de la malla en cada dirección.
100     n: Número de paso de tiempo
101     """
102     lon = np.zeros((3, 3))  # Arreglo para guardar los anchos de interfase. Es de 3x3 pero la diagonal debe quedar como
```

```
103     #                           cero si se hizo bien (es el ancho de interfase de un campo consigo mismo)
104     n_lon = np.zeros((3, 3))  # El número de valores registrados en cada entrada de "lon" (usada para promediar)
105
106     # - Listas de apoyo - #
107     ph95 = [0, 0, 0]  # Nodos inmediatamente precedentes a una intersección con p95
108     ph05 = [0, 0, 0]  # Nodos inmediatamente precedentes a una intersección con p05
109     ind = np.zeros((3, 3))  # Indicadores para casos especiales
110     isc = [0, 0, 0]  # Todas las intersecciones, ordenadas
111     iph = [0, 0, 0]  # Pares de nodos entre los que hay una interfase
112     iph2 = [0, 0, 0]  # Coordenadas de las intersecciones (interpoladas a partir de iph)
113     iph_second = [0, 0, 0]  # Arreglo para registrar el campo de fase con el que iph[a][i] forma una interfase
114     rep95 = [0, 0, 0]  # Arreglo con valores repetidos para el arreglo de intersecciones con p95
115     rep05 = [0, 0, 0]  # Arreglo con valores repetidos para el arreglo de intersecciones con p05
116
117     # - Calcular nodos de intersecciones - #
118     for a in range(3):
119         ph95[a] = np.argwhere(np.diff(np.sign(phi_cs[a]-p95))).flatten()
120         if np.sign(phi_cs[a][0]-p95) != np.sign(phi_cs[a][N-1]-p95):
121             ph95[a] = np.append(ph95[a], N-1)
122         ph05[a] = np.argwhere(np.diff(np.sign(phi_cs[a]-p05))).flatten()
123         if np.sign(phi_cs[a][0]-p05) != np.sign(phi_cs[a][N-1]-p05):
124             ph05[a] = np.append(ph05[a], N-1)
125
126         if len(ph95[a]) == 0 or len(ph05[a]) == 0:  # Si no se encuentra ninguna intersección para el campo "a"...
127             ind[a] = -1  # ... Se guarda un "indicador" (no hay interfase para el campo "a"), ...
128             continue  # ... Y se sale del ciclo
129         # NOTA: Se usa "or" porque si el campo de fase cruza sólo uno de los valores pero no el otro, entonces no hay
130         #       interfase (e.g., cuando hay una interfase entre A y B con el potencial logarítmico (t,t_c == 0.3,1), el
131         #       campo de fase de C sobrepasa el valor ph05 dentro de la interfase A<->B)
132
133         # - Puntos repetidos (interfases con espesor menor al ancho de malla y no se puede calcular) - #
134         rep95[a], rep05[a] = np.argwhere(np.in1d(ph95[a], ph05[a])).flatten(), np.argwhere(np.in1d(ph05[a], ph95[a])).flatten()
135
136     for a in range(3):
137         for b in (c for c in range(3) if c > a):  # Los dos "for" hacen un ciclo sobre (a,b)=(1,2),(1,3),(2,3)
138             if np.in1d(ph95[a][rep95[a]], ph95[b][rep95[b]]).any():  # Si algún punto repetido de ph_a coincide con ph_b
```

```
139                 num = np.sum(np.in1d(ph95[a][rep95[a]], ph95[b][rep95[b]]))  # Núm. de interfases entre a y b con \xi<1
140                 ind[a][b] = ind[b][a] = num  # Se añade un indicador para la interfase de a con b (y b con a)
141                 lon[a][b] = lon[b][a] = 1  # Se "define" el ancho de interfase 1 porque es la mínima resolución que hay
142 
143             ph95[a], ph05[a] = np.delete(ph95[a], rep95[a]), np.delete(ph05[a], rep05[a])  # Se eliminan los valores repeti-
144             #                                                                                dos que ya se registraron
145 
146             isc[a] = np.sort(np.append(ph95[a], ph05[a]))  # Todas las intersecciones en orden ascendente, sin repetidos
147 
148         # - Encontrar pares de intersecciones que forman una interfase (no intersectan al mismo valor) - #
149         for a in range(3):
150             if (ind[a] != 0).all():
151                 continue  # Se sale del ciclo si no hay interfase o ésta no se puede medir
152             iph[a] = np.delete(iph[a], 0)  # Se elimina el "0" del arreglo que se creo al declararlo
153             iph_second[a] = np.delete(iph_second[a], 0)  # IDEM
154 
155             # NOTA: numpy es compatible con indexar con arreglos, e.g., x[[0, 2]] da un arreglo con el primer y tercer ele-
156             #       mento de "x", y si se indexa con un arreglo vacío se obtiene un vacío x[[]]=[]. Además ndarray.any() no
157             #       considera verdades vacuas.
158             for i in range(len(isc[a])):
159                 if (isc[a][i-1] in ph95[a] and isc[a][i] in ph05[a]) or (isc[a][i-1] in ph05[a] and isc[a][i] in ph95[a]):
160                     if i == 0:  # El índice -1 se usa para el último elemento del arreglo
161                         iph[a] = np.append(iph[a], [isc[a][-1]-N, isc[a][0]])  # Se ajusta considerando CI periódicas
162                     else:
163                         iph[a] = np.append(iph[a], [isc[a][i-1], isc[a][i]])
164                     append = True  # Sí se añadió un elemento a la lista en este ciclo
165                 else:
166                     append = False  # No se añadió un elemento a la lista en este ciclo
167 
168                 if append:  # Si es que se añadió un elemento a la lista, se quiere encontrar con qué fase hay interfase
169                     b = a  # Inicializando variable para almacenar valor de iph_second
170                     diff = 0  # Inicializando diferencia entre campos de fase
171                     index = [-2, -1]  # Índices para hacer la comparación en una línea
172                     for c in (d for d in range(3) if d != a):  # Ciclo para las phi != phi[a]
173                         if (phi_cs[c][iph[a][index]]-phi_cs[a][iph[a][index]] > diff).any():
174                             b = c
```

```
                    diff = (phi_cs[c][iph[a][index]]-phi_cs[a][iph[a][index]]).max()
                if b != a:
                    iph_second[a] = np.append(iph_second[a], b)
                else:
                    warn.warn("Error al registrar con qué fase forma una interfase el campo phi{:d}".format(a))
                    iph_second[a] = np.append(iph_second[a], -1)  # Se agrega -1 para identificar dónde ocurre el error

        for b in (c for c in range(3) if c != a):
            if (len(iph[a]) == 0) and (ind[a][b] == 0):  # Si no hay pares de intersección que formen una interfase ...
                ind[a] = -1  # ... Se asigna el indicador de "no interfase", ...
                continue  # ... Y se sale del ciclo

        iph2[a] = iph[a].astype(float)  # Numpy tiene tipos estáticos, hay que usar otro arreglo para cambiarlo a float


    # - Interpolar para encontrar la abcisa de intersección - #
    for a in range(3):
        if (ind[a] != 0).all():
            continue  # Se sale del ciclo si no hay interfase o ésta no se puede medir
        for i in range(len(iph[a])):
            if np.where(iph[a] < 0, iph[a]+N, iph[a])[i] in ph95[a]:  # Se usa la condición periódica en iph[a]
                iph2[a][i] = iph[a][i]+(p95-phi_cs[a][iph[a][i]])/(phi_cs[a][mod(iph[a][i]+1, N)]-phi_cs[a][iph[a][i]])
            elif np.where(iph[a] < 0, iph[a]+N, iph[a])[i] in ph05[a]:
                iph2[a][i] = iph[a][i]+(p05-phi_cs[a][iph[a][i]])/(phi_cs[a][mod(iph[a][i]+1, N)]-phi_cs[a][iph[a][i]])
            else:
                warn.warn("Error al interpolar abscisa de intersección en el cálculo de ancho de interfase")

    # - Calcular el ancho de la interfase y su desviación estándar (si aplica) - #
    for a in range(3):
        if (ind[a] != 0).all():
            continue  # Se sale del ciclo si no hay interfase o ésta no se puede medir

        for i in range(len(iph[a])//2):
            if iph_second[a][i] != -1:  # Sólo se registra el ancho de interfase si se sabe con qué fase se forma
                lon[a][iph_second[a][i]] += iph2[a][2*i+1]-iph2[a][2*i]
                n_lon[a][iph_second[a][i]] += 1
```

```
211         np.where(n_lon > 0, lon/n_lon, 0)  # Se promedian los anchos de interfase (por la forma en la que se calcula,
212         #                                    Python arrojará un "RuntimeWarning", pero el NaN no se queda en el arreglo
213         #                                    final)
214
215     return lon, n_lon, ind
216
217
218 ##########################################
219 # - S U B R U T I N A   E X T E R N A - #
220 ##########################################
221 # Para llamar en el programa principal.  #
222 ##########################################
223 def plots(ener, phi_max, phi_min, dat, dt, path: str=""):
224     """
225     Genera gráficas con la información obtenida por el programa.
226
227     ener: Objeto np.ndarray donde se guardan las energías de Helmholtz calculadas.
228     phi_max: Objeto np.ndarray donde se guardan los máximos del campo de fase.
229     phi_min: Objeto np.ndarray donde se guardan los mínimos del campo de fase.
230     dat: Número de pasos de tiempo entre cada dato (separados regularmente).
231     path: Ruta donde se guardarán las gráficas.
232     """
233     L = len(ener)  # Tamaño del arreglo, se guarda para no volverlo a calcular
234     t = np.arange(0, L*dt*dat, dt*dat)
235
236     ######################
237     # - E N E R G Í A - #
238     ######################
239
240     # - Dibujar Gráfica - #
241     plt.figure(figsize=(16, 10), dpi=200)
242     plt.plot(t, ener, color='tab:olive')
243
244     # - Apariencia - #
245     ymin = np.min(ener)*1.1-np.max(ener)*0.1  # Límite inferior de la gráfica 10% debajo del mínimo.
246     ymax = np.max(ener)*1.1-np.min(ener)*0.1  # Límite superior de la gráfica 10% encima del máximo.
```

```
247     plt.ylim(ymin, ymax)
248     xtick_loc = [i/20*(L-1)*dt*dat for i in range(21)]
249     xtick_lab = [np.around(i/20*(L-1)*dt*dat, decimals=0) for i in range(21)]
250     plt.xticks(ticks=xtick_loc, labels=xtick_lab, rotation=90, fontsize=11, horizontalalignment='center', alpha=.7)
251     plt.xlabel("Tiempo", fontsize=14, alpha=.85)
252     plt.yticks(fontsize=12, alpha=.7)
253     plt.ylabel("$\\mathcal{F}(\\phi)$", fontsize=14, alpha=.85)
254     plt.title("Energía", fontsize=22)
255     plt.grid(axis='both', alpha=.3)
256 
257     # - Quitar marco - #
258     plt.gca().spines["top"].set_alpha(0.0)
259     plt.gca().spines["bottom"].set_alpha(0.3)
260     plt.gca().spines["right"].set_alpha(0.0)
261     plt.gca().spines["left"].set_alpha(0.3)
262 
263     # - Guardar figura - #
264     plt.savefig(path+"Ener"+".tiff", format='tiff')
265     plt.close()
266 
267     for i in range(3):
268         plt.figure(figsize=(10, 16), dpi=200)
269 
270         ######################
271         # - M Á X I M O S - #
272         ######################
273 
274         # - Dibujar Gráfica - #
275         plt.subplot(2, 1, 1)
276         plt.plot(t, phi_max[i], color='tab:red')
277 
278         # - Apariencia - #
279         ymin = np.min(phi_max[i])*1.1-np.max(phi_max[i])*0.1  # Límite inferior de la gráfica 10% debajo del mínimo.
280         ymax = np.max(phi_max[i])*1.1-np.min(phi_max[i])*0.1  # Límite superior de la gráfica 10% encima del máximo.
281         plt.ylim(ymin, ymax)
282         xtick_loc = [j/20*(L-1)*dt*dat for j in range(21)]
```

```
283         xtick_lab = [np.around(j/20*(L-1)*dt*dat, decimals=1) for j in range(21)]
284         plt.xticks(ticks=xtick_loc, labels=xtick_lab, rotation=90, fontsize=11, horizontalalignment='center', alpha=.7)
285         plt.xlabel("Tiempo", fontsize=14, alpha=.85)
286         plt.yticks(fontsize=12, alpha=.7)
287         plt.ylabel("$(\\phi_{%d})_\\mathrm{max}$" % (i+1), fontsize=14, alpha=.85)
288         plt.title("Valor máximo de $\\phi_{:d}$".format(i+1), fontsize=20)
289         plt.grid(axis='both', alpha=.3)
290 
291         # - Quitar marco - #
292         plt.gca().spines["top"].set_alpha(0.0)
293         plt.gca().spines["bottom"].set_alpha(0.3)
294         plt.gca().spines["right"].set_alpha(0.0)
295         plt.gca().spines["left"].set_alpha(0.3)
296 
297         ######################
298         # - M Í N I M O S - #
299         ######################
300 
301         # - Dibujar Gráfica - #
302         plt.subplot(2, 1, 2)
303         plt.plot(t, phi_min[i], color='tab:cyan')
304 
305         # - Apariencia - #
306         ymin = np.min(phi_min[i])*1.1-np.max(phi_min[i])*0.1  # Límite inferior de la gráfica 10% debajo del mínimo.
307         ymax = np.max(phi_min[i])*1.1-np.min(phi_min[i])*0.1  # Límite superior de la gráfica 10% encima del máximo.
308         plt.ylim(ymin, ymax)
309         xtick_loc = [j/20*(L-1)*dt*dat for j in range(21)]
310         xtick_lab = [np.around(j/20*(L-1)*dt*dat, decimals=1) for j in range(21)]
311         plt.xticks(ticks=xtick_loc, labels=xtick_lab, rotation=90, fontsize=11, horizontalalignment='center', alpha=.7)
312         plt.xlabel("Tiempo", fontsize=14, alpha=.85)
313         plt.yticks(fontsize=12, alpha=.7)
314         plt.ylabel("$(\\phi_{%d})_\\mathrm{min}$" % (i+1), fontsize=14, alpha=.85)
315         plt.title("Valor mínimo de $\\phi_{:d}$".format(i+1), fontsize=20)
316         plt.grid(axis='both', alpha=.3)
317 
318         # - Quitar marco - #
```

```
319         plt.gca().spines["top"].set_alpha(0.0)
320         plt.gca().spines["bottom"].set_alpha(0.3)
321         plt.gca().spines["right"].set_alpha(0.0)
322         plt.gca().spines["left"].set_alpha(0.3)
323 
324         plt.tight_layout()
325 
326         # - Guardar figura - #
327         plt.savefig(path+"phi{:d}_maxmin".format(i+1)+".tiff", format='tiff')
328         plt.close()
329 
330 
331 def animation(path='', names=None, filename=''):
332     """
333     Crea un GIF animado de las imágenes en "names" y lo guarda en la ruta "path".
334 
335     path: Ruta donde se guardará la animación.
336     names: Lista de los nombres de los archivos de imágenes que se animarán.
337     filename: Nombre del archivo de destino.
338     """
339     images = []
340     for n in names:
341         frame = Image.open(n)
342         images.append(frame)
343     images[0].save(path+filename, save_all=True, append_images=images[1:], duration=50, loop=0)  # 1000/duration = fps.
344 
345 
346 def cross_sec(phi: np.ndarray, phi_high: float, phi_low: float, n: int, N: int, path: str="", perm: tuple=(0, 1, 2),
347               dx: float=1):
348     """
349     Genera gráficas del campo de fase a lo largo de un corte horizontal a la mitad del dominio. (Si N es par, se toma
350     el corte en la mitad "de arriba")
351 
352     phi: Campo de fase, es un objeto np.ndarray de (3, N, N).
353     phi_high: Valor que toman los componentes del campo de fase en la fase asociada a su mismo componente.
354     phi_low: Valor que toman los componentes del campo de fase en una fase asociada a un componente distinto.
```

```
355     n: Número de paso de tiempo, para nombrar al archivo generado.
356     N: Número de puntos de la malla en cada dirección.
357     path: Ruta donde se guardarán las gráficas.
358     perm: Permutación de los índices usada para crear las gráficas. Usada para colocar el ancho calculado de la interfa-
359           se sobre su respectiva franja.
360     dx: Separación espacial entre los puntos de la malla.
361     """
362     M = N//2+1
363
364     p95 = phi_high-0.05*(phi_high-phi_low)  # Valor de "corte" superior para la interfase
365     p05 = phi_low+0.05*(phi_high-phi_low)  # Valor de "corte" inferior para la interfase
366
367     phi_cs = np.zeros((3, N))
368     for a in range(3):
369         for i in range(N):
370             phi_cs[a][i] = phi[a][i][M]
371
372     x = np.arange(N)
373
374     # - Dibujar Gráficas - #
375     plt.figure(figsize=(16, 10), dpi=100)
376     plt.plot(x, phi_cs[0], color='tab:red')
377     plt.plot(x, phi_cs[1], color='tab:green')
378     plt.plot(x, phi_cs[2], color='tab:blue')
379
380     plt.legend(("$\\phi_{1}$", "$\\phi_{2}$", "$\\phi_{3}$"), loc='right', fontsize=16)
381
382     # - Líneas al 5 y 95% del rango entre las concentraciones de equilibrio - #
383     plt.plot(x, p95*np.ones(N), color='black', linestyle='dashed')
384     plt.plot(x, p05*np.ones(N), color='black', linestyle='dashed')
385
386     # - Calcular ancho de la interfase - #
387     inter_len, inter_num, ind = interph(phi_cs, p95, p05, N)
388
389     inter_len = inter_len*dx
390
```

```
391     if (inter_num > 1).any():
392         print("Los anchos de interfase para el PdT #{:d} fueron promediados\n".format(n)+str(inter_num))
393 
394     tex = ['', '', '']  # Arreglo para el texto
395     for a in range(3):
396         for b in (c for c in range(3) if c != a):
397             if ind[a][b] == 0:
398                 tex[a] = tex[a]+"$\\xi_{:s} = {:.6g}$\n".format("{"+str(a+1)+str(b+1)+"}", inter_len[a][b])
399             if ind[a][b] == 1:
400                 tex[a] = tex[a]+"$\\xi_{:s} < 1$\n".format("{"+str(a+1)+str(b+1)+"}")
401             if ind[a][b] == -1:
402                 tex[a] = tex[a]+"No se encontró \ninterfase de $\\phi_{:d}$ con $\\phi_{:d}$".format(a+1, b+1)
403 
404     # - Apariencia - #
405     ymin = 0
406     ymax = 1
407     plt.ylim(ymin, ymax)
408     xtick_loc = [i/10*N for i in range(11)]
409     xtick_lab = [np.around(i/10*N, decimals=1) for i in range(11)]
410     plt.xticks(ticks=xtick_loc, labels=xtick_lab, rotation=0, fontsize=12, horizontalalignment='center', alpha=.7)
411     plt.xlabel("Distancia (sobre línea de corte)", fontsize=14, alpha=.85)
412     plt.yticks(fontsize=12, alpha=.7)
413     plt.ylabel("$\\phi_{1}, \\phi_{2}, \\phi_{3}$", fontsize=14, alpha=.85)
414     plt.title("Campos de fase", fontsize=22)
415     for a in range(3):
416         ap = perm[a]
417         plt.text((2*ap+1)*N/6, 0.9, tex[a], fontsize=12, ha='center')
418     plt.grid(axis='both', alpha=.3)
419 
420     # - Quitar marco - #
421     plt.gca().spines["top"].set_alpha(0.0)
422     plt.gca().spines["bottom"].set_alpha(0.3)
423     plt.gca().spines["right"].set_alpha(0.0)
424     plt.gca().spines["left"].set_alpha(0.3)
425 
426     # - Guardar figura - #
```

```
427     plt.savefig(path+"sec_tr{:d}".format(n)+".tiff", format='tiff')
428     plt.close()
429 
430     return inter_len
431 
432 
433 def interph_plot(inter_len: np.ndarray, csec: float, dt: float, path: str=""):
434     L = len(inter_len[0][1])  # Tamaño del arreglo, se guarda para no volverlo a calcular
435     for a in range(3):
436         plt.figure(figsize=(16, 10), dpi=200)
437         t = np.arange(0, L*dt*csec, dt*csec)  # Arreglo con los pasos de tiempo
438         maxv = 0  # Valor máximo de las gráficas
439         legend = []  # Lista vacía para las leyendas
440         for b in (c for c in range(3) if c != a):
441             color = [0, 0, 0]  # Color de la gráfica del ancho de interfase
442             color[a], color[b] = 1, 0.5
443             color = tuple(color)
444             plt.plot(t, inter_len[a][b], color=color)
445             maxv = np.maximum(maxv, inter_len[a][b].max())
446             legend = legend+["$\\xi_{:s}$".format("{"+str(a+1)+str(b+1)+"}")]
447 
448         plt.legend(tuple(legend), loc='lower right', fontsize=16)
449 
450         # - Apariencia - #
451         ymin = 0
452         ymax = maxv*1.1  # Límite superior de la gráfica 10% encima del máximo.
453         plt.ylim(ymin, ymax)
454         xtick_loc = [i/20*(L-1)*csec for i in range(21)]
455         xtick_lab = [np.around(i/20*(L-1)*csec, decimals=0) for i in range(21)]
456         plt.xticks(ticks=xtick_loc, labels=xtick_lab, rotation=90, fontsize=11, horizontalalignment='center', alpha=.7)
457         plt.xlabel("Tiempo", fontsize=14, alpha=.85)
458         plt.yticks(fontsize=12, alpha=.7)
459         plt.ylabel("$\\xi_{:d}$".format(a+1), fontsize=14, alpha=.85)
460         plt.title("Ancho de la interfase para el componente $\\phi_{:d}$".format(a+1), fontsize=22)
461         plt.grid(axis='both', alpha=.3)
462 
```

```
463         # - Quitar marco - #
464         plt.gca().spines["top"].set_alpha(0.0)
465         plt.gca().spines["bottom"].set_alpha(0.3)
466         plt.gca().spines["right"].set_alpha(0.0)
467         plt.gca().spines["left"].set_alpha(0.3)
468
469         # - Guardar figura - #
470         plt.savefig(path+"inter_len{:d}".format(a+1) + ".png", format='png')
471         plt.close()
```

Referencias

Abdel-Azim, A.-A. A. & Munk, P. (1987). Light Scattering of Liquids and Liquid Mixtures. 1. Compressibility of Pure Liquids. *The Journal of Physical Chemistry*, *91*(14), 3910-3914. doi:10.1021/j100298a036

Anderson, D. M., Mcfadden, G. B. & Wheeler, A. A. (1998). Diffuse-Interface Methods in Fluid Mechanics. *Annual Review of Fluid Mechanics*, *30*(1), 139-165. doi:10.1146/annurev.fluid.30.1.139

Ascher, U. M., Ruuth, S. J. & Wetton, B. T. R. (1995). Implicit-Explicit Methods for Time-Dependent Partial Differential Equations. *SIAM Journal on Numerical Analysis*, *32*(3), 797-823. doi:10.1137/0732037

Badalassi, V., Ceniceros, H. & Banerjee, S. (2003). Computation of multiphase systems with phase field models. *Journal of Computational Physics*, *190*, 371-397. doi:10.1016/S0021-9991(03)00280-8

Bevington, P. R. & Robinson, D. K. (2003). *Data Reduction and Error Analysis for the Physical Sciences*. Boston: McGraw-Hill.

Boettinger, W. J., Warren, J. A., Beckermann, C. & Karma, A. (2002). Phase-Field Simulation of Solidification. *Annual Review of Materials Research*, *32*(1), 163-194.

doi:10.1146/annurev.matsci.32.101901.155803. eprint: https://doi.org/10.1146/annurev.matsci.32.101901.155803

Boyer, F., Lapuerta, C., Minjeaud, S., Piar, B. & Quintard, M. (2009). Cahn–Hilliard/Navier–Stokes Model for the Simulation of Three-Phase Flows. *Transport in Porous Media*, *82*(3), 463-483. doi:10.1007/s11242-009-9408-z

Byron, F. & R.W., F. (1992). *Mathematics of Classical and Quantum Physics*. New York: Dover Publications.

Callen, H. (1985). *Thermodynamics and an Introduction to Thermostatistics*. Toronto: John Wiley & Sons.

De Menech, M. (2006). Modeling of droplet breakup in a microfluidic T-shaped junction with a phase-field model. *Physical Review E*, *73*, 031505. doi:10.1103/PhysRevE.73.031505

Fiocco, D. (2012). Phase Field Modelling of Phase Separation using the Cahn–Hilliard Equation. Recuperado el 7 de febrero de 2019, desde https://documents.epfl.ch/users/f/fi/fiocco/www/JournalClubFiles/ModelB.pdf

Frigo, M. & Johnson, S. G. (2018). *FFTW: for version 3.3.8*. Massachusetts Institute of Technology.

Granlund, T. (2017). *Instruction latencies and throughput for AMD and Intel x86 Processors*. Recuperado el 20 de mayo de 2019, del sitio web de The GNU Multiple Precision Arithmetic Library: https://gmplib.org/~tege/x86-timing.pdf.

Kim, J. & Lowengrub, J. (2006). Interfaces and Multicomponent Fluids. *Encyclopedia of Mathematical Physics*, 135-144. doi:10.1016/b0-12-512666-2/00275-3

Kim, S. G., Kim, W. T. & Suzuki, T. (1999). Phase-field model for binary alloys. *Physical Review E*, *60*, 7186-7197. doi:10.1103/PhysRevE.60.7186

Kobayashi, H., Ode, M., Kim, S. G., Kim, W. T. & Suzuki, T. (2003). Phase-field model for solidification of ternary alloys coupled with thermodynamic database. *Scripta Materialia*, *48*, 689-694. doi:10.1016/S1359-6462(02)00557-2

Kot, M. (2014). *A First Course in the Calculus of Variations*. Rhode Island: American Mathematical Society.

LeVeque, R. J. (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. Philadelphia: Society for Industrial and Applied Mathematics.

Linz, P. (1979). *Theoretical Numerical Analysis: An Introduction to Advanced Techniques*. John Wiley & Sons.

Liu, H. & Zhang, Y. (2009). Droplet formation in a T-shaped microfluidic junction. *Journal of Applied Physics*, *106*, 034906. doi:10.1063/1.3187831

Liu, H. & Zhang, Y. (2010). Phase-field modeling droplet dynamics with soluble surfactants. *Journal of Computational Physics*, *229*, 9166-9187. doi:10.1016/j.jcp.2010.08.031

Malyshev, A. (2005). On the Spectral Differentiation. Recuperado el 4 de marzo de 2020, desde http://www.ii.uib.no/~sasha/INF263/spectral.pdf

MathWorks®. (2003). Stiff Differential Equations. *Technical Articles and Newsletters*. Recuperado el 14 de abril de 2019, desde https://www.mathworks.com/company/newsletters/articles/stiff-differential-equations.html

Novick-Cohen, A. & Segel, L. A. (1984). Nonlinear aspects of the Cahn-Hilliard equation. *Physica D: Nonlinear Phenomena*, *10*(3), 277-298. doi:10.1016/0167-2789(84)90180-5

Pearce, D. A. J. (2007). *GSW... Counting the Costs*, University of York. Texto no publicado disponible en: http://www-users.york.ac.uk/~dajp1/Introductions/index.html.

Pego, R. (1989). Front migration in the nonlinear Cahn-Hilliard equation. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, *422*(1863), 261-278. doi:10.1098/rspa.1989.0027

Penrose, O. & Fife, P. C. (1990). Thermodynamically consistent models of Phase-Field type for the kinetic of Phase Transitions. *Physica D: Nonlinear Phenomena*, *43*(1), 44-62. doi:10.1016/0167-2789(90)90015-h

Press, W. H., Flannery, W. H., Teukolsky, S. A. & Vetterling, W. T. (1992). *Fortran Numerical Recipes*:*Vol. 1. Numerical Recipes in Fortran 77: The Art of Scientific Computing* (Vols. 2). Cambridge: Cambridge University Press.

Rogers, B. A., Rembert, K. B., Poyton, M. F., Okur, H. I., Kale, A. R., Yang, T., ... Cremer, P. S. (2019). A stepwise mechanism for aqueous two-phase system formation in concentrated antibody solutions. *Proceedings of the National Academy of Sciences*, *116*(32), 15784-15791. doi:10.1073/pnas.1900886116

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. Philadelphia: Society for Industrial and Applied Mathematics.

Safari, H., Rahimian, M. H. & Krafczyk, M. (2014). Consistent simulation of droplet evaporation based on the phase-field multiphase lattice Boltzmann method. *Physical Review E*, *90*, 033305. doi:10.1103/PhysRevE.90.033305

Springer Publishing & European Mathematical Society. (2015). Numerical Analysis. *Encyclopedia of Mathematics*. Recuperado el 12 de marzo de 2019, desde https://www.encyclopediaofmath.org/index.php?title=Numerical_analysis

Stover, C. (2019). Ternary Diagram – *from Wolfram MathWorld*. Recuperado el 5 de noviembre de 2019, desde http://mathworld.wolfram.com/TernaryDiagram.html

Tavakoli, R. (2016). Unconditionally energy stable time stepping scheme for Cahn–Morral equation: Application to multi-component spinodal decomposition and optimal space tiling. *Journal of Computational Physics*, *304*, 441-464. doi:10.1016/j.jcp.2015.10.018

van Brunt, B. (2010). *The Calculus of Variations*. New York: Springer.

Wheeler, A. A., Boettinger, W. J. & McFadden, G. B. (1992). Phase-field model for isothermal phase transitions in binary alloys. *Physical Review A*, *45*, 7424-7439. doi:10.1103/PhysRevA.45.7424