UNIVERSIDAD INTERNACIONAL DE LA RIOJA

Universidad Internacional de La Rioja
Máster universitario en Ingeniería Matemática y Computación

Diseño y desarrollo de un simulador de yacimientos de petróleo negro convencional con paralelización GPU basado en JAVA

Trabajo Fin de Máster

Tipo de trabajo: Desarrollo práctico

Presentado por: Ramos R., Jorge Luis

Director/a: Chicharro, Francisco

Ciudad: Bogotá/ Colombia

Fecha:23 de Julio del 2020

Resumen

Este proyecto tiene como meta, demostrar la viabilidad del uso de las librerías OpenCL en JAVA para el desarrollo de un simulador de yacimiento de petróleo negro.

El desarrollo de este proyecto cubrió la codificación de los módulos CPU en JAVA y los módulos GPU en JAVA mediante el uso de la librería JOCL que permitió implementar el paquete OpenCL.

Posteriormente, se realizó el desarrollo de una interface mediante el uso de la librería JAVAFX para el diseño de las ventanas de los datos de entrada para facilitar la configuración del caso de simulación, así como las ventanas para la visualización de los datos 2D y 3D de salida.

Finalmente, se realizaron dos casos de evaluación de la aplicación desarrollada cuyos resultados fueron comparados con el simulador comercial IMEX de la Compañía Computer Modeling Group mostrando resultados muy similares al programa comercial.

Palabras Clave: JOCL, OpenCL, Paralelización, Simulación de Yacimientos, Java.

Abstract

This project aims to demonstrate the feasibility of using OpenCL libraries in JAVA for the development of a black oil reservoir simulator.

The development of this project covered the coding of the CPU modules in JAVA and the GPU Modules in JAVA Through the use of the JOCL library that allowed to implement the OpenCL Package.

Subsequently, an interface was developed using the JAVAFX library for the design of the input data windows to facilitate the configuration of the simulation case, as well as the windows for the visualization of the 2D and 3D output data.

Finally, two evaluation cases of the application developed were carried out, the results of which were compared with the IMEX commercial simulator of Computer Modeling Group Company, showing results very similar to the commercial program.

Keywords: JOCL, OpenCL, Parallelization, Reservoir Simulation, Java.

Tabla de contenido

Índice de ilustraciones

Índice de Ecuaciones

Índice de Tablas

1. Introducción.

A continuación, se describirán los aspectos relacionados a la motivación, planteamiento de trabajo, estructura del trabajo, objetivos generales y específicos del proyecto:

1.1. Motivación.

Con el actual avance que se ha realizado en el modelamiento y paralelización de modelamiento matemático basado en GPU (Unidad de Procesamiento Gráfico por sus siglas en inglés) ha permitido en la actualidad de una manera mucho más económica el desarrollo de aplicaciones que requiere el desarrollo de millones de cálculos de una forma rápida. Se sabe que la paralelización mediante CPU (Unidad de Procesamiento Central por sus siglas en inglés) ha sido por excelencia la manera tradicional de desarrollar aplicativos que requieren muchos cálculos complejos, donde usualmente se requiere se computadores muy potentes y costosos, basados en CPU para el modelaje de procesos en un tiempo relativamente corto. Sin embargo, con el advenimiento de uso del GPU en el cálculo matemático de procesos complejos surge una nueva manera de procesar información de una forma más económica, ya que actualmente casi cualquier dispositivo inteligente posee unidades de procesamiento gráficos muy eficientes y potentes, donde usualmente las unidades GPU son más baratas que las CPU.

En el área de petróleo y gas existe muchas herramientas comerciales que permite de la predicción del comportamiento futuro de los reservorios basados en la Ecuación de Difusividad llamados Simuladores de Yacimientos, los cuales son altamente costosos, y usualmente estaban basados en procesamiento paralelo CPU.

La ecuación base de todo simulador de yacimientos se denomina Ecuación de Difusividad, la cual está basada en un balance de materia, y la Ley de Darcy desarrollada por Henry Darcy (1856), la cual viene dada en forma vectorial de la siguiente manera:

$$v_{ox} = -\frac{k_{ox}}{\mu_o} * \frac{\partial P_o}{\partial x}$$

Esta ecuación es la base fundamental para el modelaje de flujo de fluidos en medios porosos. Para un modelo de fluido tipo Black Oíl, la ecuación diferencial en derivadas parciales de la forma, por ejemplo:

- Para la Fase Petróleo:

$$\frac{\partial(\frac{\rho_o * k_{ox}}{\mu_o} * \frac{\partial P_o}{\partial x})}{\partial x} + \frac{\partial(\frac{\rho_o * k_{oy}}{\mu_o} * \frac{\partial P_o}{\partial y})}{\partial y} + \frac{\partial(\frac{\rho_o * k_{oz}}{\mu_o} * \frac{\partial(P_o \pm \rho_o * \Delta z * g)}{\partial z})}{\partial z} = \frac{\partial(\rho_o * \emptyset * S_o)}{\partial t}$$

- Para la Fase Agua:

$$\frac{\partial(\frac{\rho_w * k_{wx}}{\mu_w} * \frac{\partial P_w}{\partial x})}{\partial x} + \frac{\partial(\frac{\rho_w * k_{wy}}{\mu_o} * \frac{\partial P_w}{\partial y})}{\partial y} + \frac{\partial(\frac{\rho_w * k_{wz}}{\mu_w} * \frac{\partial(P_w \pm \rho_w * \Delta z * g)}{\partial z})}{\partial z} = \frac{\partial(\rho_w * \emptyset * S_w)}{\partial t}$$

- Para la Fase Gas:

$$\frac{\partial(\frac{\rho_g * k_{gx}}{\mu_g} * \frac{\partial P_g}{\partial x})}{\partial x} + \frac{\partial(\frac{\rho_g * k_{gy}}{\mu_g} * \frac{\partial P_g}{\partial y})}{\partial y} + \frac{\partial(\frac{\rho_g * k_{gz}}{\mu_g} * \frac{\partial(P_g \pm \rho_g * \Delta z * g)}{\partial z})}{\partial z} = \frac{\partial(\rho_g * \emptyset * S_g)}{\partial t}$$

Como puede ser evidenciado, estas ecuaciones en Derivadas Parciales (EDP), son de tipo parabólicas, las cuales pueden ser resueltas por los métodos explícitos, implícitos y de Crank Nicholson, evidenciando el uso de matemáticas avanzadas en el mismo, sin embargo, para este proyecto específico presenta una propuesta de solución denominada IMPES (Implícito en Presión–Explicito en Saturación), la cual es de uso común en los simuladores de yacimientos. Es importante resaltar que la ecuación de Difusividad es un caso particular de la Ecuación de Navier-Stokes, la cual es muy usada en procesos de flujo de fluidos, flujo de fluidos compresibles e incompresibles, y análisis CDF (Mecánica Computacional de Fluidos).

Como complemento, en todo programa de simulación mucha de la información requerida es en forma de tablas para parámetros como, por ejemplo, Propiedades PVT que incluyen disponer de curvas tales como:

- Presión vs Relación Gas en Solución (Rs)
- Presión vs Factor Volumétrico (Bo, Bw,Bg)
- Presión vs Densidad (ro, rg, rw)
- Presión vs Viscosidad (ro, rg, rw)

Debido a esto, es usualmente requerido el uso de métodos avanzados de interpolación como el uso de interpoladores de Lagrange, polinomio de Newton y el lineal, que permiten interpolar los datos que son necesarios en los cálculos de la simulación y la resolución de cada ecuación de Difusividad en cada paso del tiempo, donde se requieren para valores de presiones que no estén exactamente en la tabla proporcionada. Finalmente, es común el uso de método

numérico de resolución de sistemas de ecuaciones lineales, mediante el Método Algoritmo de Thomas, implementado con métodos de relajación lineal (LSOR), además de la aplicación de los métodos de Jacobi, Gauss-Seidel y Gradiente Conjugado, para el modelaje de los yacimientos petrolíferos mediante el uso de la ecuación de Difusividad.

Basado en lo anterior, el proyecto propone el desarrollo de un simulador de yacimientos de petróleo negro convencional con paralelización basada en GPU, desarrollado en java mediante la implementación de las librerías JOCL (JAVA bindings for OpenCL), que permiten aprovechar las bondades de uso de las unidades GPU en el procesamiento de cálculos matemáticos complejos, de una forma rápida que sea accesible y de uso general, tanto para la academia, y como en un futuro, para la industria. Con base a lo anterior, **se evidencia en este proyecto la aplicación tanto de matemáticas y computación avanzadas que son requisitos obligatorios para el trabajo de fin de máster.**

1.2. Planteamiento del trabajo

El proyecto tuvo como finalidad el desarrollo de un software que involucre el uso el procesamiento paralelo GPU para el desarrollo de un simulador de yacimientos Black Oil. El planteamiento fue realizado para mallas de geometría de Bloque Centrado, considerando modelo de fluido tipo Black Oil, además, sin considerar flujo en fractura, solo matricial. El programa esta codificado en JAVA y usa el paquete OpenCL (Open Computing Language) a través de la librería JOCL (JAVA bindings for OpenCL) para el procesamiento paralelo GPU y evidencia la utilidad de esta creciente tecnología, que, de forma más barata y sencilla, permite acelerar aplicaciones que antes solo usaban la tecnología CPU.

1.3. Objetivos

El objetivo general del proyecto es diseñar un simulador de yacimientos de petróleo negro convencional con paralelización GPU basado en JAVA a través del uso de la librería JOCL.

Para llevar a cabo este objetivo general, se plantean los siguientes objetivos específicos:

- Diseñar y describir las interfaces gráficas creadas mediante la librería JAVAFX de las secciones básicas e inputs del simulador propuesto.
- Codificar el módulo del procesamiento CPU secuencial a través de clases JAVA.
- Codificar el módulo procesamiento paralelo GPU a través del uso de las librerías JOCL para implementar la plataforma de cómputo OpenCL.

- Validar los módulos codificados mediante dos casos de validación comparando los resultados obtenidos con el Simulador Comercial Black Oil IMEX de CMG (Computer Modelling Group).

1.4. Estructura del trabajo.

El documento que describe la presente investigación se encuentra distribuido en 8 capítulos los cuales serán brevemente explicados a continuación:

Capítulo 1: Describe la motivación del proyecto, así como el planteamiento del problema, definición de objetivos generales y específicos y la estructura del documento.

Capítulo 2: Se presenta el estado del arte del proyecto donde se mencionan diversas investigaciones asociadas a la tecnología de procesamiento paralelo mediante sistema Heterogéneos GPU-CPU, procesamiento paralelo y conceptos de simulación de yacimientos.

Capítulo 3: En este capítulo se describen algunos simuladores de yacimientos existentes para establecer el estado actual de estas herramientas en el mercado, además se describe el contexto habitual de uso de la simulación de yacimientos y se establece las características esperadas por la aplicación desarrollada en este proyecto.

Capítulo 4: Se detalla la estructura diseñada del aplicativo desarrollado en el proyecto mostrando sus secciones y explicando la información necesaria para ejecución del mismo.

Capítulo 5: Se explica la metodología usada para el desarrollo del proyecto donde se incluye la descripción de las fases de desarrollo del proyecto, así como el tiempo empleado en cada fase presentado en un diagrama de Gantt, la descripción de los participantes con sus roles en el proyecto.

Capítulo 6: En este capítulo se describen las secciones constitutivas de Modulo correspondiente al procesamiento GPU mediante el uso de diagramas de flujo, además se presentan los códigos usados para la configuración de la ejecución del código JOCL de cada sección para implementar la tecnología OpenCL.

Capítulo 7: Corresponde a la descripción del despliegue de la aplicación y la descripción de los dos casos de validación generados para realizar la validación de la aplicación desarrollada en este proyecto al ser comparados los resultados obtenidos por la aplicación desarrollada vs los obtenidos mediante el simulador comercial IMEX de la Compañía CMG (Computer Modelling Group).

Capítulo 8: Corresponde a las conclusiones y líneas futuras.

2. Estado del arte.

Este capítulo describe el estado del arte actual del proyecto, así como aspectos teóricos asociados a OpenCL y Simulación de Yacimientos.

2.1. Estado actual del Procesamiento Computacional Paralelo GPU y del Lenguaje de Programación JAVA

A continuación, se procederá a describir estudios donde se observa la evolución que ha tenido el procesamiento paralelo GPU y el lenguaje de programación JAVA en diversas áreas de estudio:

En primera instancia podemos mencionar diferentes proyectos (Nvidia Corporation, 2018, págs. 2-44), donde el procesamiento GPU ha ayudado a acelerar cálculos computacionales en un amplio rango de industrias, evidenciando aplicaciones en las áreas de Ingeniería Financiera, Oceanografía, Climatología, Data Science, Deep Learning e Machine Learning, Inteligencia y defensa, Manufacturing, CAD and CAE, media y entretenimiento, Medicina, Oil and Gas, Educación superior y Supercomputing entre otros.

Aunado a esto, es importante mencionar que (Levchenko, Anastasia, & Zakirov, 2016) desarrollaron un proyecto el cual describe un nuevo algoritmo para unidades de procesamiento de propósito general (GPGPU[1]) para la resolución de la ecuación de Onda, mediante el método de diferencias finitas, donde evidencian resultados en los cuales, el nuevo algoritmo mejora el desempeño de los cálculos por un factor de 5. También es interesante mencionar que (Kawada, Okubo, Tagawa, Tsuchiya, & Ishizuka, 2015) desarrollaron un estudio que comprende el computo GPU basado en técnicas de alta velocidad de visualización de propagación de ondas sonoras, que muestra el uso del cómputo basado en GPU para mejorar la velocidad de implementación de la técnica de visualización de la solución de la ecuación de onda, donde evidencia el uso de OpenGL y CUDA. Otro estudio similar en esta área había sido desarrollado por (Micikevicius,Paulius;NVIDIA, 2009) , donde hizo uso de la paralelización GPU usando las librerías CUDA de NVIDIA para la paralelización de un algoritmo para aplicación de diferencias finitas 3D para resolver la ecuación de Onda.

[1] GPGPU: Procesamiento de Propósito General GPU

Para mencionar estudios en otras áreas, (Zaza, Awotunde, Fairag, & Al-Mouhamed, 2016), desarrollaron un proyecto basado en CUDA para el desarrollo de un simulador de petróleo negro paralelizado, que muestra el uso la librería CUDA de NVIDIA mediante computo GPU donde evidencia el potencial uso de la tecnología GPU en cálculos de alto desempeño.

También es interesante mencionar que (NIKLAS, 2013) desarrolló un proyecto, donde el procesamiento GPU fue una herramienta capaz de acelerar una aplicación, obteniendo velocidades mayores correspondiente al tradicional algoritmo en CPU al resolver la ecuación de Navier-Stokes para fluidos Incompresibles, aplicándola en el área de la mecánica de los fluidos. Estudios similares fueron desarrollados por (Saeed & Perot, 2012) y (Pina, 2012) donde hicieron uso de las GPU en simulaciones de dinámica computacional de fluidos, usando el método de gradiente conjugado para resolver la ecuación de Possion y, resolver simulaciones basadas en elementos finitos y algoritmos de búsqueda para la indexación espacial para realizar la paralelización del movimiento.

Otro estudio que evidencia cómo ha evolucionado el uso de la tecnología GPU, fue realizado por (Shankar, Kalyanapu, Hansen, & Burian, 2011), donde muestran como modelos de inundación 1D representados con la ecuación Saint Venant, que fueron calculados a través del procesamiento GPU, ganando velocidades de cálculo entre 50 a 135 más que su contraparte CPU. Además (Mohammed, Langguth, Spiga, Baden, & Cai, 2015) desarrollaron un proyecto donde se hace uso eficiente de las plataformas CPU+GPU para el desarrollo de cálculo de alto desempeño de Clústeres donde nuevamente se evidencia la utilidad de esta tecnología.

Por otra parte (Chris, 2010), realizó un resumen de la evolución de la tecnología GPU, desde los años 1980, donde describe la evolución de la misma en las diversas funcionalidades en que han evolucionado las GPU hasta la actualidad, como herramienta de Supercomputo.

Finalmente es importante mencionar que (Mohammed Ali, 2016), desarrolló un proyecto donde se detalla la implementación de procesamiento de alto desempeño (HPC) CPU en JAVA, a través del uso de la concurrencia y, también menciona que JAVA puede tener el mismo desempeño HPC que aplicaciones nativas en C, en casos donde aplique convirtiendo la programación HPC en java como una posibilidad viable. Un estudio similar fue desarrollado previamente por (Launay & Pazat, 2006), donde presentaron un modelo de programación

paralela, entrelazado en un framework es usado como la base de generar programas distribuidos.

Como ha podido evidenciarse, la tecnología de procesamiento Heterogéneo CPU-GPU ha tenido una gran evolución de su uso, desde el simple procesamiento gráfico al procesamiento de alto desempeño, en diversas áreas de estudios en el campo de las ciencias y la ingeniera.

2.2. Teoría de OpenCL

OpenCL es un sistema de software que permite a los programadores escribir un programa portátil capaz de usar todos los recursos disponibles en alguna plataforma heterogénea. En un sistema heterogéneo la plataforma puede incluir CPU de varios núcleos, una o más GPU y otros dispositivos informáticos según los autores (Science, 2012) y (Aaftab, R., G., James, & Dan, 2012).

En principio, OpenCL puede usarse también para programar sistemas homogéneos como los procesadores Intel multi-core, donde un núcleo puede ser un host y los otros pueden proporcionar un rendimiento mejorado mediante el procesamiento paralelo masivo.

Según (Science, 2012) y (Aaftab, R., G., James, & Dan, 2012), la mayoría de los sistemas informáticos paralelos actuales pueden llamarse hardware homogéneo. En sistemas homogéneos, computadoras o procesadores similares están interconectados en una sola computadora capaz de procesamiento paralelo. Los tipos de sistemas paralelos incluyen multiprocesadores de memoria compartida y grupos de computadoras enteras. Los autores consideran que la idea de computación paralela basada en la heterogeneidad, en la cual el sistema tiene dos o más tipos distintos de procesadores, donde la CPU ordinaria, llamada el host, es responsable de administrar las funciones de uno o más procesadores altamente paralelos llamados dispositivos (por ejemplo, dispositivos GPU controlados por un sistema host de CPU). Los dispositivos son responsables del procesamiento altamente paralelo que acelera la computación. Estos sistemas se denominan heterogéneos, porque consisten en tipos muy diferentes de procesadores.

La idea fundamental detrás de OpenCL es su independencia del hardware del proveedor y sistemas operativos. Esta universalidad y portabilidad tiene un precio. El precio es más

esfuerzo de programación y más tiempo de aprendizaje. (Science, 2012) y (Aaftab, R., G., James, & Dan, 2012)

2.3. Simulación de yacimientos

La simulación de yacimientos es el área de la ingeniería de yacimientos que mediante el uso de modelos computarizados es posible predecir el flujo de fluidos (típicamente, petróleo, agua y gas) a través de medios porosos . Esto es realizado a través de uso de la ecuación de difusividad y los métodos numéricos.

La construcción de los modelos computacionales de yacimientos hidrocarburíferos, es realizado por un estudio interdisciplinario que toma en cuenta información geológica, estructural, geofísica, sedimentológica, petrofísica, las propiedades de los fluidos contenidos (petróleo, agua, gas),interacción entre la roca y los fluidos, además del comportamiento dinámico de los pozos, así como la consideración de las capacidades de las infraestructuras para producir los mismos (sistemas),donde también es considerado el uso de tecnologías nuevas para su desarrollo, que mediante el uso de la simulación numérica permiten establecer las actividades sobre el desarrollo del yacimiento en su conjunto, denominado plan de desarrollo.

Tomando en cuenta lo anterior, la principal actividad del ingeniero de yacimientos, es estar involucrado en los cálculos para establecer el desarrollo del mismo, es elaborar un modelo numérico basado en conceptos individuales derivados de los estudios previamente mencionados.

Los modelos, fenómenos y procesos para la producción de yacimientos están soportados mediante leyes físicas que se expresan en forma de ecuaciones matemáticas (usualmente ecuaciones diferenciales), lo cual indican que están caracterizados por ciertas expresiones matemáticas.

Aunado a esto, los grandes avances modernos en el área computacional hacen posible tener en cuenta la heterogeneidad y anisotropía de las propiedades petrofísicas de un yacimiento, así como poder realizar una precisa representación de la estructura y geometría del mismo, los cuales pueden ser representados con un gran detalle.

En el capítulo 3, en el apartado 3.1, podrá ser visualizado un resumen de algunos de los principales simuladores de yacimientos de código abierto y comerciales disponibles en el mercado. A continuación, se procederá a describir estudios donde se observa la evolución que ha tenido la simulación de yacimientos:

En primera instancia, podemos mencionar que (Breitenbach, Thurnau, & Van Poolen, 1968) desarrollaron un programa informático llamado MUFFS, donde hacen hincapié en la versatilidad en el estudio de una amplia gama de problemas relacionados con el flujo inmiscible de una, dos o tres fases en una, dos o tres dimensiones.

(Pope & Nelson, 1978) desarrollaron un simulador unidimensional, composicional, de inyección química para calcular la recuperación de petróleo en función de varias variables importantes del proceso. Las principales relaciones incluidas son el comportamiento de las fases y las tensiones interfaciales en función de las concentraciones de electrolito y surfactante, y la viscosidad del polímero en función de la concentración de electrolito y polímero. Además, (Kazemi, Vestal, & Shank, 1978) desarrollaron un eficiente simulador numérico tridimensional, trifásico y multicomponente de yacimientos para estudiar los yacimientos de petróleo, en los que la transferencia de masa entre fases es importante. Las ecuaciones de flujo tienen un balance de volumen en la fase de agua y un balance molar en las fases de hidrocarburos vapor-líquido.

Aunado a esto (Cheshire, Appleyard, Banks, Crozier, & Holmes, 1980), desarrollaron un simulador llamado PORES, el cual fue un eficiente simulador tridimensional implícito de propósito general para yacimientos de petróleo negro. La eficiencia se logra resolviendo las ecuaciones acopladas mediante un nuevo método secuencial que asegura automáticamente el equilibrio material. En el mismo año, (Youngren, 1980) describen uno de los primeros simuladores de yacimientos de combustión in situ tridimensional y trifásico que modela rigurosamente el flujo de fluidos, la transferencia de calor y la vaporización/condensación. Tiene cinco componentes: agua, oxígeno, aceite no volátil y dos componentes volátiles arbitrarios. Además (Wang, Lake, & Pope, 1981) desarrollaron y aplicaron un simulador composicional de gran escala, bidimensional, de componentes múltiples y multifase para la inyección micelar/polimérica.

Posteriormente (Chen, Wasserman, & Fitzmorris, 1987) , proponen que, debido al petróleo relativamente pesado (8-12 grados API[2]) que se encuentra en muchos yacimientos fracturados naturalmente, los procesos de recuperación térmica podrían ser técnicas de recuperación viables para producir petróleo de estos yacimientos. Para facilitar la simulación de estos procesos, desarrollaron un simulador para modelar los efectos térmicos en yacimientos fracturados naturalmente.

Además,(Durlofsky & Chien, 1993) proponen una técnica de volumen finito mixto de elementos finitos basada en triángulos para la simulación de yacimientos composicionales. La formulación es del tipo IMPES[3] aplicable a sistemas trifásicos, multicomponentes en dos dimensiones. La mayoría de las anteriores formulaciones de elementos finitos mixtos se limitaban a sistemas de dos fases o totalmente miscibles. El enfoque composicional que se presenta aquí amplía estas técnicas anteriores, aunque conserva la estructura general de las formulaciones más limitadas. En el mismo año, (Buchwalter & Miller, 1993) , desarrollaron un nuevo simulador composicional simplificado totalmente implícito. Este simulador es diferente de los modelos existentes porque combina las ecuaciones convencionales del petróleo negro con una ecuación de componente de inyección composicional.

(Kaarstad, Froyen, Bjorstad, & Espedal, 1995) desarrollaron un simulador secuencial de yacimientos tridimensional IMPLICITO de dos fases para sistemas informáticos masivamente paralelos. El algoritmo paralelo se basa en los métodos de descomposición de dominios recientemente desarrollados, El simulador ha sido probado utilizando un procesador 16384 MasPar MP-2. En el mismo año, (Deb, Reddy, Thuren, & Adams, 1995), describen un nuevo simulador de yacimientos tridimensional multifásico de propósito general que utiliza mallas curvilíneas totalmente no estructuradas y emplea una adaptabilidad de malla dinámica basada en una solución de flujo. El simulador se basa en un marco de elementos finitos no convencional denominado elementos finitos de alta potencia (donde h se refiere al tamaño de la malla local y p al orden de la aproximación del polinomio local).

(Hemanth-Kumar & Young, 1996) describen la adaptación de un simulador de yacimientos para cálculos paralelos. El simulador fue diseñado originalmente para procesadores vectoriales. Realiza aproximadamente el 99% de sus cálculos en modo vectorial/paralelo y,

[2] API: American Petroleum Institute
[3] IMPES: Implícito en Presión-Explicito en Saturación

en relación con los cálculos escalares, alcanza velocidades de 65 y 81 para las simulaciones de aceite negro y EOS, respectivamente en el CRAY C-90.

(Wang, y otros, 1999) presentan una ecuación de estado (EOS) composicional paralela totalmente implícita para la simulación de yacimientos a gran escala. Este simulador se desarrolla en el marco llamado IPARS (Integrated Parallel Accurate Reservoir Simulador) y se construye utilizando un tipo de Newton formulación. La EOS de Peng-Robinson se utiliza para los cálculos de comportamiento de la fase de hidrocarburos. Los solucionadores lineales del paquete PETSc (Portable Extensible Toolkit for Scientific Computation) se utilizan para la solución de las ecuaciones lineales subyacentes.

(Crane, Bratvedt, Bratvedt, Childs, & Olufsen, 2000) desarrollaron un nuevo simulador de yacimientos trifásico basado en la extensión del método de la línea de corriente. El proyecto se centra en los nuevos métodos desarrollados para la simulación de la línea de corriente por composición, así como en las ventajas y desventajas de esta estrategia en comparación con los enfoques más tradicionales.

(John, Han, Delshad, Pope, & Sepehrnoori, 2005) proponen que los simuladores de yacimientos composicionales basados en formulaciones de ecuaciones de estado (EOS) típicamente no manejan el modelado del comportamiento de la fase acuosa, y los que están diseñados para modelar procesos químicos típicamente asumen un comportamiento simplificado de la fase de hidrocarburos. Es necesario contar con un simulador de yacimiento único capaz de combinar ambos enfoques para aprovechar las ventajas de los modelos acuosos y de hidrocarburos. El desarrollo y la aplicación de procedimientos implícitos para modelar simultáneamente el comportamiento de las fases de hidrocarburos y acuosa es un proceso complejo. En este proyecto se presenta un enfoque para integrar el modelo de comportamiento de la fase a surfactante en un simulador EOS composicional paralelo totalmente implícito ya existente.

(Han, Stone, Liu, Cook, & Papanastasiou, 2005) proponen que, con el aumento de las fuerzas de carga debido al continuo agotamiento de los yacimientos, se producen desplazamientos irreversibles de roca, que dan lugar a diversos problemas como la compactación de los yacimientos, el hundimiento de la superficie, la fractura de la formación, etc. En la industria petrolera, se han realizado numerosos esfuerzos para incorporar los análisis elastoplásticos

de rocas en simuladores de yacimientos ampliamente disponibles. Sobre la base de las actuales teorías de la mecánica de rocas y los resultados experimentales, se propone un enfoque elastoplástico tridimensional resuelto con el método de los elementos finitos en celdas de cuadrícula en geometría de punto angular, y se demuestra su aplicación a un simulador de yacimientos comerciales.

(Ju, y otros, 2006) proponen que la liberación y la migración de los finos y arenamiento, es un problema universal en la producción de petróleo de los yacimientos de arenisca no consolidadas, lo que puede dar lugar tanto a problemas de arenisca como a profundos efectos en la recuperación del petróleo. Los autores presentan un nuevo modelo matemático tridimensional (3D) a escala de campo, que difiere de los utilizados por el simulador numérico convencional de yacimientos de petróleo, donde presentan tanto las teorías avanzadas de liberación y migración de partículas de arena. El modelo se resuelve mediante un método de diferencia finita y la técnica de línea sucesiva sobre relajación (LSOR). El simulador numérico está escrito en Fortran 90 y VC++.

(Appleyard, Appleyard, Wakefield, & Desitter, 2011) muestran que los recientes avances en las unidades de procesamiento gráfico (GPU) y los entornos de desarrollo asociados adecuados para el modelado científico han generado un gran interés en el ámbito de la computación de alto rendimiento. El artículo investiga las estrategias para incorporar esta nueva tecnología en un simulador de depósito comercial existente. Se demuestra el uso de la GPU para resolver sistemas lineales y se discuten las consideraciones algorítmicas necesarias para explotar el hardware.

(Anciaux-Sedrakian, y otros, 2015) exponen que las supercomputadoras heterogéneas que combinan "procesador gráfico de uso general con muchos núcleos integrados" y CPU de uso general prometen ser la arquitectura principal del futuro, porque ofrecen un excelente rendimiento con un consumo de energía y una ocupación de espacio limitados. Sin embargo, el desarrollo de aplicaciones que logren una escalabilidad "real" sigue siendo objeto de varios campos de investigación. Los autores estudian la posibilidad de aprovechar la potencia de la arquitectura heterogénea para las simulaciones dinámicas de las geociencias para modelos reales a gran escala.

(Mukundakrishnan, y otros, 2015) establecen que hay numerosas características complejas que afectan al comportamiento de declinación a largo plazo de los pozos en los yacimientos de petróleo y gas apretados. Se requiere una simulación numérica con una fina discretización espacial para captar características importantes como: la presencia de fracturas naturales (estimuladas y no estimuladas); cambios en la movilidad y compresibilidad total del fluido; propiedades heterogéneas de la matriz y la fractura (incluida la pérdida de permeabilidad debido a la compactación); transitorios de flujo tempranos; y estimulación no uniforme durante la fractura hidráulica. Los autores presentan esfuerzos, para ayudar a eliminar esta compensación construyendo un simulador de petróleo negro totalmente implícito que combina los recientes avances en los algoritmos de simulación con el alto rendimiento de las GPU. Todas las tareas de cálculo importantes se ejecutan en la GPU, incluyendo la evaluación de propiedades, la construcción y el ensamblaje jacobianos y la solución lineal con CPR-AMG.

(Huang, Shiralkar, Li, & Abbas, 2018) describen un nuevo modelo numérico de EOR químico capaz de simular las inyecciones de surfactantes y polímeros. Presentamos los aspectos más destacados de una implementación de IMPES altamente eficiente y robusta dentro de un simulador de composición de gas-aceite-agua propio y heredado. La sobrecarga computacional adicional, por ejemplo, un cálculo de inundación, es del orden de sólo el 20% para simulaciones a gran escala (modelo de patrón tipo). Presentaron los resultados de rendimiento tanto en modo serie como paralelo (multiprocesador).

Finalmente (Lashgari, Pope, Balhoff, & Tagavifar, 2019) proponen que, en los últimos años se han hecho avances significativos en la recuperación química mejorada del petróleo, incluido el desarrollo de métodos híbridos que combinan surfactantes, polímeros, álcalis, cosolventes, gas y calor en formas novedosas. Se han desarrollado nuevos y mejorados modelos de propiedades químicas y físicas para simular con mayor precisión estos procesos a escala de campo. Presentan modelos mejorados para la permeabilidad relativa, la presión capilar, el efecto de la viscoelasticidad de los polímeros en la saturación de aceite residual, el efecto del pH en la adsorción de surfactantes, la división de los polímeros entre las fases acuosa y de microemulsión, y el efecto del cosolvente en la viscosidad de la microemulsión. Además, presentan varias simulaciones para demostrar cómo se pueden utilizar los modelos para ajustar los datos experimentales.

Como pudo ser observado, la simulación de yacimientos siempre se encuentra en la gran evolución tratando de adaptarse a las nuevas tecnologías disponibles en el mercado.

3. Identificación de Requisitos

3.1. Descripción de algunos simuladores de yacimientos

Para establecer una referencia de estado actual de los simuladores se describirán algunos de los simuladores comerciales y de código abierto más usados. Comenzando por los simuladores de fuente abierta, distinguimos entre:

a) **BOAST**: El simulador *Black Oil Applied Simulation Tool* (Boast), es un paquete de software gratuito para la simulación de yacimientos desarrollado en el Departamento de Energía de EE. UU. Boast es un simulador numérico IMPES (saturación explícita de presión implícita de diferencia finita) que primero encuentra la distribución de presión para un paso de tiempo dado y luego calcula la distribución de saturación para el mismo paso de tiempo isotérmico. (Almengor, y otros, June, 2006).

b) **MRST**: Es la caja de herramientas para simulación de yacimientos de MATLAB (MRST) fue desarrollada por SINTEF *Applied Mathematics* como una caja de herramientas de MATLAB®. MRST consta de dos partes principales: un núcleo que ofrece funcionalidad básica y solucionadores monofásicos y bifásicos, y un conjunto de módulos adicionales que ofrecen modelos, visores y solucionadores más avanzados. (Digital, s.f.).

c) **OPM**: La iniciativa *Open Porous Media* (OPM) proporciona un conjunto de herramientas de código abierto centradas en la simulación del flujo y el transporte de fluidos en medios porosos. (Initiative, s.f.).

En cuanto a los simuladores comerciales, podemos encontrarnos con:

d) ECLIPSE : Desarrollado originalmente por ECL (*Exploration Consultants Limited*) y actualmente propiedad, desarrollado, comercializado y mantenido por SIS (anteriormente conocido como *GeoQuest*), una división de Schlumberger . El nombre ECLIPSE originalmente era un acrónimo de "Programa implícito de ECL para ingeniería de simulación". Los simuladores incluyen, de petróleo negro, composicional, de volumen finito, térmico y otros. Las opciones adicionales incluyen refinamientos locales de la malla, metano de carbón, operaciones de campo de gas, pozos avanzados, acoplamiento de yacimientos y redes de superficie. (Schlumberger Limited, s.f.).

e) INTERSECT: Desarrollado por Schlumberger en conjunto con Chevron es un simulador de alta resolución que permite manejar modelos con un gran número de celdas y pozos. (Schlumberger Limited, s.f.).

f) **ECHELON:** Desarrollado por ***Stone Ridge Technology*****,** un simulador totalmente implícito, el único simulador de yacimientos acelerado de GPU completo para formulaciones de petróleo negro. (ECHELON, s.f.)**.**

g) **RETINA Simulator**: Es un software de simulación de yacimientos de petróleo negro y composicional, totalmente desarrollado en la Compañía de Desarrollo de Tecnología y Soporte de Ingeniería (ESTD). (Engineering Support & Technology Development Company (ESTD), s.f.).

h) **CMG Suite (IMEX, GEM y STARS):** Desarrollado por *Computer Modeling Group,* actualmente ofrece tres simuladores: un simulador de petróleo negro, llamado IMEX, un simulador composicional / no convencional llamado GEM y un simulador de procesos térmicos y avanzados llamado STARS. (Computer Modelling Group L.T.D, s.f.)**.**

i) **Sensor:** Desarrollado por *Coats Engineering,* es un simulador de yacimientos composicional y de petróleo negro desarrollado a principios de la década de 1990 por el Dr. Keith H. Coats, fundador de la industria de simulación de yacimientos comerciales (Engineering, s.f.; Engineering, s.f.).

j) **XXSim:** Es un simulador de yacimientos composicional de propósito general basado en EOS con formulación totalmente implícita. Permite que cualquier componente aparezca y permanezca en cualquier fase fluida (acuosa, oleosa y vapor). Se puede simplificar en los módulos convencionales, tradicionales de petróleo negro, composicional y térmicos. También se puede ampliar a un simulador térmico totalmente basado en EOS[4] (technologycatalogue, s.f.).

k) **Tempest MORE:** De Emerson es un simulador de yacimientos que ofrece aceite negro, opciones de composición y térmicas. (Emerson Electric Co, s.f.).

l) **ExcSim:** Es un simulador de yacimientos de petróleo negro modificado trifásico en 2D totalmente implícito para la plataforma Microsoft Excel. (petromehras, s.f.).

m) **Nexus :** Nexus es un simulador de yacimientos de petróleo y gas desarrollado originalmente como 'Falcon' por Amoco , el Laboratorio Nacional de Los Álamos y *Cray Research* . Actualmente es propiedad, desarrollada, comercializada y mantenida por Landmark Graphics, una línea de servicio de productos de Halliburton . Nexus reemplazará gradualmente VIP, o Desktop VIP, la generación anterior de simulador de Landmark. (Halliburton, s.f.).

n) **tNavigator:** Desarrollado por Rock Flow Dynamics admite simulaciones de petróleo negro, composicional y térmicos para estaciones de trabajo y grupos de computación de alto rendimiento. (Rock Flow Dynamics, s.f.).

[4] EOS: Equations of State.

o) **FlowSim:** Es un simulador de yacimientos de diferencias finitas compositivas trifásicas, tridimensionales, de aceite negro y compuesto con refinamientos locales, doble porosidad, doble permeabilidad y capacidades paralelas. (Plano Research Corporation, s.f.).

p) **ReservoirGrail :** Emplea un método patentado llamado tiempo dinámica volumétrica de equilibrio a los yacimientos para simular durante la recuperación primaria y la recuperación secundaria . (GRAILQUEST CORP, s.f.).

q) **Merlin:** Es un simulador de yacimientos de diferencias finitas trifásico totalmente implícito desarrollado originalmente en el departamento de investigación de Texaco y actualmente utilizado por la Oficina de Gestión de Energía Oceánica y la Oficina de Seguridad y Cumplimiento Ambiental para calcular las tasas de descarga en el peor de los casos y las presiones de estallido/ colapso en zapatos de revestimiento y prevención de reventones. (Gemini Solutions Inc, s.f.).

r) **DeepSim:** Es un simulador de yacimientos de diferencias finitas compositivas de 3 fases totalmente implícito para la plataforma de tabletas y teléfonos Android. (DeepSim, s.f.)

s) **Exodus:** Ofrece un simulador de petróleo negro completo con un módulo de ajuste de histórico asistido que puede variar la porosidad / permeabilidad / estructura / netpay / presión inicial / saturación / profundidades de contacto para que coincida con las tasas observadas / acumulados / presiones de los pozos. (T.T. & Associates Inc., s.f.)

t) **Meera**: Es un simulador de simulación de yacimientos híbrido basado en la tecnología AI-Physics para la planificación de operaciones. (Meera Simulation, s.f.).

3.2. Contexto habitual de uso

Los simuladores de yacimientos usualmente son usados en empresas del sector petróleo y gas y a nivel de la academia con la finalidad de realizar pronósticos de producción que permiten evaluar la factibilidad técnico-económica de diferentes tecnologías, planes de desarrollo y explotación que desean ser implementados en un yacimiento en particular y de esa forma decidir si el proyecto es técnicamente aplicable, y si es financieramente rentable. Los simuladores de yacimientos típicos son capaces de obtener el comportamiento esperado de parámetros tales como:

- Cambios Temporales de la Tasa de Fluidos Producidos (Petróleo, Agua, Gas) a nivel de pozo y campo.
- Cambios Temporales de la Tasa de Fluidos Inyectados (Agua, Gas, etc.) a nivel de pozo y campo.
- Cambios Temporales de la Presión de fondo observada a nivel de pozo y campo.

- Cambios Temporales de los Volúmenes producidos Acumulados (Petróleo, Agua, Gas) a nivel de pozo y campo.
- Cambios Temporales de los Volúmenes inyectados Acumulados (Agua, Gas, etc.) a nivel de pozo y campo.
- Cambios Temporales en la presión promedia del yacimiento.
- Cambios temporales de las saturaciones (petróleo, agua, gas) celda a celda en el yacimiento.
- Cambios temporales de las presiones celda a celda en el yacimiento.
- Modelar procesos de recobro mejorado y procesos especiales. Incluye, cambios temporales de los parámetros específicos asociados a procesos de recobro mejorado y procesos especiales celda a celda en el yacimiento.

El principal inconveniente de acceso a dichas herramientas, es el alto costo asociados a las licencias, además el hecho de no tener acceso a los códigos para que desde el punto de vista académico se permita estudiar la forma como fueron programadas, donde solo existen 4 en el mercado de código abierto. Aunado a esto, la gran mayoría estas herramientas no tienen la flexibilidad de seleccionar los métodos de resolución de sistemas lineales típicos (la mayoría de los programas comerciales solo usa el método de Newton-Raphson). Finalmente, la gran mayoría de los simuladores comerciales no disponen del acceso a procesamiento Heterogéneo GPU-CPU.

La aplicación propuesta en la presente investigación, tendrá las mismas capacidades de cómputo de los simuladores comerciales para modelar recobros convencionales para yacimientos de flujo matricial en modelos de fluido black oil, con la característica distintiva de su capacidad de realizar procesamiento heterogéneo GPU-CPU, y al estar programado en JAVA tendrá la posibilidad de ser multiplataforma. Además, en el futuro, podrá fácilmente ser migrado para pasar de una aplicación de escritorio a aplicación web, así como la plataforma Android y iOS para aplicaciones móviles y Tableta, que estará acelerado por procesamiento GPU.

3.3. Características del simulador propuesto

El simulador propuesto en esta investigación debe ser capaz de proporcionar y ejecutar la siguiente lista de especificaciones y requisitos funcionales que serán mencionados a continuación:

3.3.1. A nivel de datos de entrada (Identificación de requisitos funcionales)

- Solo considerará el régimen de flujo Matricial.
- Considerará procesos isotérmicos (Temperatura Constante).
- La geometría de malla que podrá ser usada, será de tipo block center, sin embargo, permitirá importe de Mallas Corner Point en formato ECLGRD[5], y las convertirá a formato Block Center. Estas podrán también ser generadas dentro de la aplicación.
- Las propiedades petrofísicas podrán ser ingresadas de forma manual capa a capa, o podrán ser importadas en un formato de archivo de texto o en formato ECLGRD para un modelo de heterogeneidad completa.
- Se considerará un modelo de compresibilidad de la roca variable con presión y esta podrá ser importada o generada en la aplicación.
- El modelo de fluido usado será Black Oil considerando las fases petróleo, agua y gas, que podrán ser importadas o generadas mediante correlaciones estadísticas dentro de la aplicación.
- El modelo roca-fluido usado será de dos sistemas bifásicos considerando los modelos Stone I y Stone II (Society of Petroleum Engineers, 2012-2020) para modelar permeabilidades relativas trifásicas. Estos modelos bifásicos podrán ser importados o generados mediante dos correlaciones estadísticas.
- Dispondrá de los modelos de inicialización por equilibrio y también por recurrencia.
- Permitirá el importe de datos de trayectoria en formato de archivo de texto.
- Permitirá generar el modelo de completamiento de forma automática y además permitirá el importe y exporte de los mismos.
- Permitirá el Importe de Datos Históricos de tasa de fluidos producidos e inyectados en formato de archivo de texto.
- A nivel de modelaje de presión a escala de pozo permitirá usar 3 métodos para el método de ponderamiento de la presión promedio en los pozos (1 punto,5 puntos y 9 puntos).
- A nivel de métodos numéricos tendrá la flexibilidad de seleccionar 4 métodos para la resolución de sistemas lineales (Algoritmo de Thomas, Gauss-Seidel, Jacobi, Gradiente Conjugado) que tendrán una optimización por el método de relajación lineal.
- A nivel de métodos numéricos tendrá la flexibilidad de seleccionar 3 métodos para la interpolación (lineal, polinomio de Lagrange, polinomio de Newton).
- Tendrá la flexibilidad de seleccionar 3 módulos de cómputo: procesamiento heterogéneo GPU-CPU, a través de las librerías JOCL para la implementación de

[5] ECLGRD: ECLIPSE GRID Format

OpenCL, procesamiento CPU Unicore y procesamiento CPU paralelo, a través de JAVA Threads. El objetivo principal del proyecto será el procesamiento heterogéneo GPU-CPU y los restantes son complementarios.

3.3.2. A nivel de resultados de salidas (identificación de requisitos funcionales)

La aplicación propuesta deberá ser capaz de arrojar los siguientes resultados:

- Cambios Temporales de la Tasa de Fluidos Producidos (Petróleo, Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de la Tasa de Fluidos Inyectados (Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de la Presión de fondo observada a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de los Volúmenes producidos Acumulados (Petróleo, Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de los Volúmenes inyectados Acumulados (Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales en la presión promedia del yacimiento (Resultados 2D).
- Cambios temporales de las saturaciones (petróleo, agua, gas) celda a celda en el yacimiento (Resultados 3D).
- Cambios temporales de las presiones celda a celda en el yacimiento (Resultados 3D).
- Datos básicos de Geometría (Dimensiones (X, Y, Z) y coordenadas (X, Y, Z) celda a celda en el yacimiento (Resultados 3D).
- Datos básicos de propiedades estáticas (porosidad, permeabilidad (X, Y, Z) y Net to Gross) (Resultados 3D).

3.3.3. A nivel de diseño de la aplicación (Identificación de requisitos funcionales)

Se basará en las siguientes herramientas y especificaciones:

- Se programará en el lenguaje JAVA mediante el IDE ECLIPSE.
- Será una aplicación de escritorio, pero será fácilmente migrable a aplicación Web, y móvil para ser desarrollada en IPad, Tablet, y móviles Android.
- Dispondrá de una interface gráfica representada a través de ventanas que permitirán el ingreso de los datos de entrada.
- Usará la librería JAVAFX y JAVAFX 3D para el diseño de las ventanas y las visualizaciones 2D y 3D.

- Dispondrá de una interface de ventanas que podrá presentar los resultados obtenidos (2D y 3D) dentro de la misma aplicación
- Permitirá el exporte de los resultados 2D en formato Excel (.xls)
- Permitirá el exporte de los resultados 3D en formato ECLGRD (. ECLGRD)
- Arrojará un reporte sumario en formato de archivo de texto de los resultados obtenido a nivel de campo.
- Permitirá el guardado y reapertura de proyecto, así como el importe de resultados de corridas previas
- Realizará el procesamiento heterogéneo GPU-CPU, a través de las librerías JOCL para la implementación de OpenCL. A nivel complementario dispondrá de procesamiento CPU unicore y CPU paralelo a través de JAVA Threads.
- Dispondrá de un Instalador/Des instalador para el sistema operativo Windows, pero podrá ser migrable a plataformas Mac iOS y Linux ya que el lenguaje JAVA tiene la característica de ser multiplataforma.

4. Diseño y descripción de la estructura del simulador de yacimiento Black Oil desarrollado

Las siguientes secciones describirán las interfaces que fueron programadas mediante las librerías JAVAFX que permitirán la generación y ejecución de un caso de simulación para la aplicación propuesta en este proyecto.

La Ilustración 1 presenta la ventana principal de la aplicación desarrollada en esta investigación que se denominó ***JARESIM v2020***, lo cual traduce ***JAVA Reservoir Simulator,*** la cual es una aplicación que permite la simulación de yacimientos en 3 fases, y 3 dimensiones, que puede usar tanto esquema de procesamiento CPU y GPU, mediante el uso de librerías JOCL que permiten la implementación del Lenguaje OpenCL, para el uso de las GPU en el cómputo de alto desempeño.

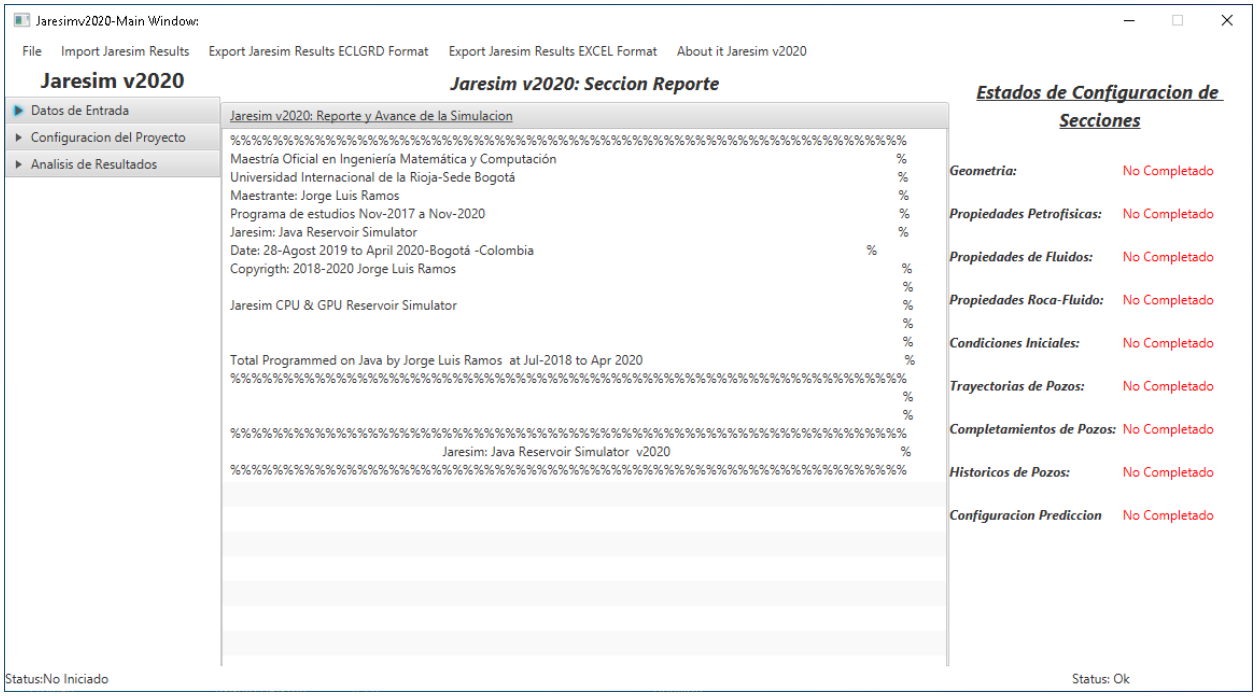


Ilustración 1:Ventana Principal de Simulador propuesto. Fuente: Elaboración propia.

Como puede evidenciarse, la aplicación presenta 3 secciones principales denominadas:

- Datos de Entrada.
- Configuración del Proyecto.
- Análisis de Resultados.

Estas secciones serán descritas a continuación de forma más detallada.

4.1. Datos de Entrada

Esta sección está constituida por subsecciones, a través del cual el simulador podrá adquirir la información necesaria para realizar la simulación, como se menciona a continuación:

- Geometría del Modelo.
- Propiedades Estáticas.
- Propiedades del Fluido.
- Datos de Interacción Roca-Fluido.
- Inicialización.
- Trayectorias.
- Completamientos.
- Históricos.

Como puede evidenciarse en la Ilustración 2, la distribución de la sección de datos de entrada del Modelo.

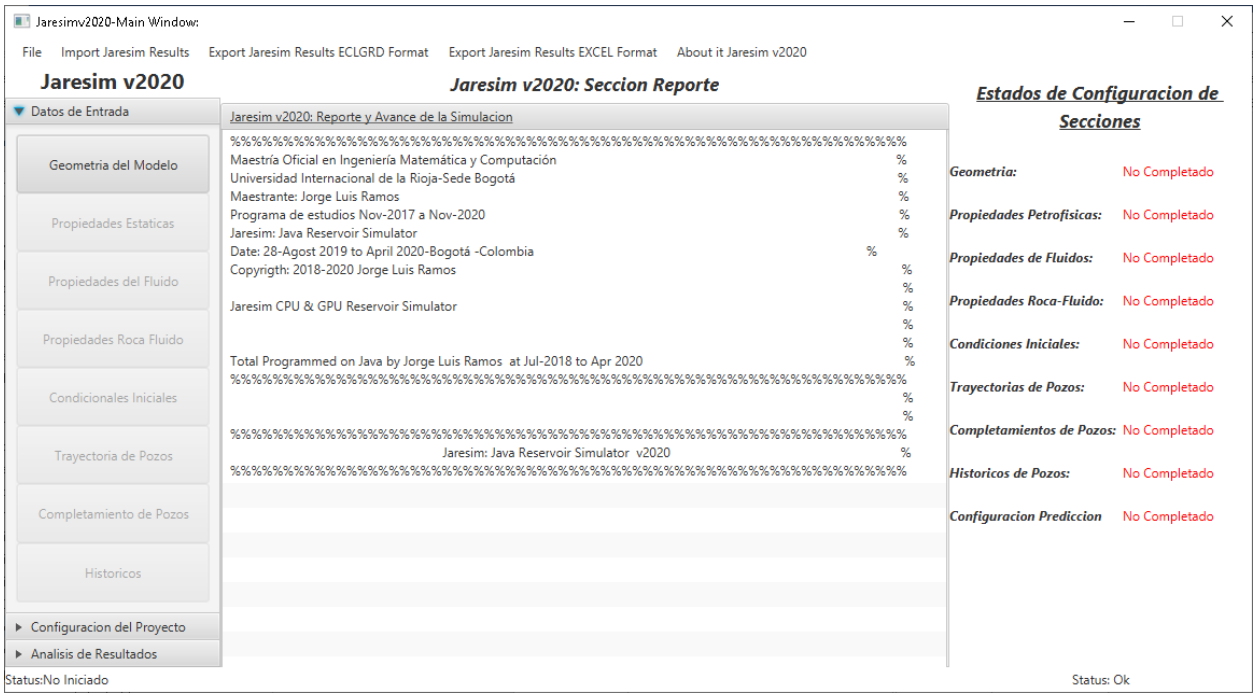


Ilustración 2: Ventana Principal de Simulador propuesto -Sección Datos de entrada Fuente: Elaboración propia.

4.1.1. Geometría del Reservorio y Propiedades Estáticas

A continuación, se realizará una descripción de cada una de las respectivas subsecciones de geometría del modelo y propiedades estáticas:

El primer aspecto de la geometría considerado será el tipo de malla, la malla considerada para el desarrollo de la aplicación es una malla cartesiana ortogonal de geometría de Bloque Centrado (Este punto está desarrollado en el Anexo 10.5), el cual puede ser visto en la Ilustración 3.

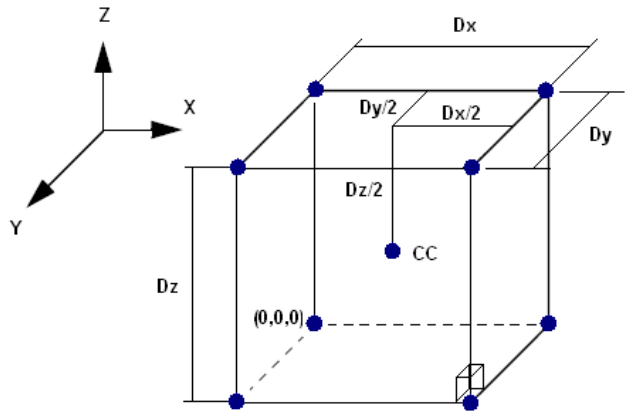


Ilustración 3:Configuración de las celdas de mallas regulares ortogonal. Fuente: (Ramos R, 2008).

El segundo aspecto de la geometría considerado, es las opciones de generación de malla, el programa desarrollado permite dos vías para generar la malla para un caso de simulación:

1. **Opción de generación de malla rectangular**: Para la generación de una malla rectangular, se debe realiza el ingreso de los datos de la malla, se debe disponer de la información observada en la Ilustración 4, donde se observa la ventana de ingresos de datos de geometría de malla. Los parámetros que deben ser ingresados se listan a continuación:
   - Nombre de la Malla.
   - Tamaño de celdas en dirección X, Y, Z.
   - Número de Celdas en Dirección X, Y, Z.
   - Origen del sistema de Coordenadas en dirección X, Y, Z.
   - Se considera que el sistema no tendrá rotación.

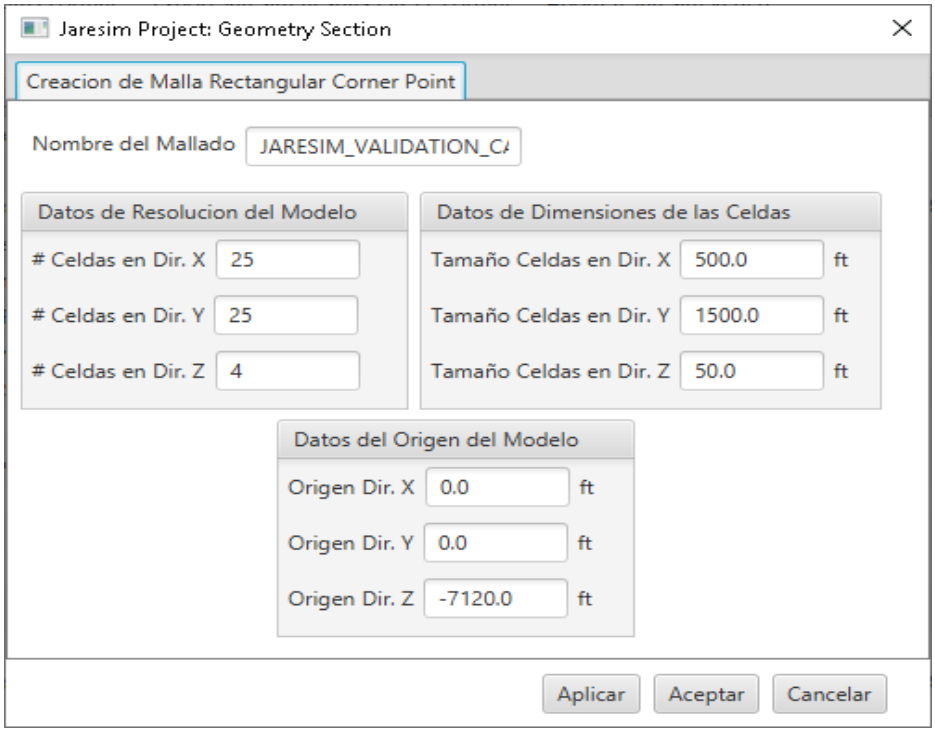


Ilustración 4:Ventana de la Sección Geometría del Modelo-Malla Rectangular Generada. Fuente: Elaboración propia.

A continuación, se muestra la Ilustración 5, donde se puede evidenciar la orientación asumida de la malla para el modelo en la aplicación:

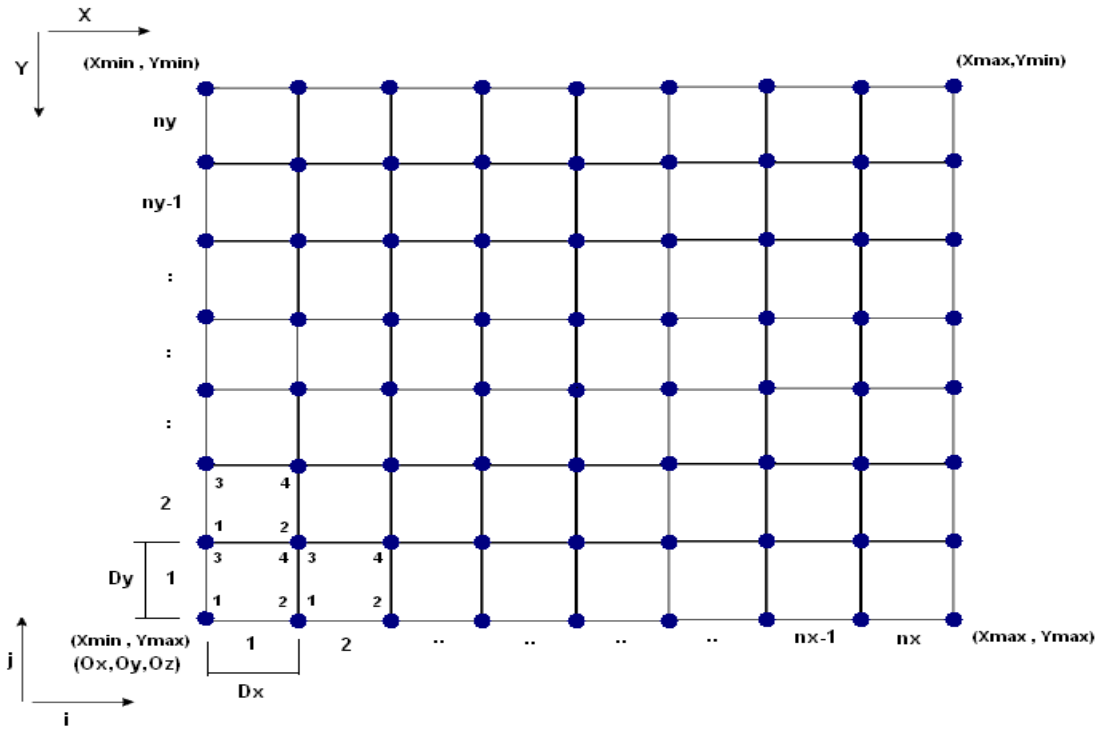


Ilustración 5:Ordenamiento de los Vértices en una Malla Regular Ortogonal (Vista areal). Fuente: (Ramos R, 2008).

2. **Opción de generación mediante importe de malla Corner Point:** Para la generación de una malla mediante importe, se realiza importando la malla en formado ECLGRD de ECLIPSE para mallas de tipo Corner point (Este punto está desarrollado en el Anexo 10.5), el cual puede ser buscado a través de una ventana browser. Una vez seleccionado el archivo, el aplicativo realizará la conversión a su equivalente Bloque Centrado, mostrando los valores promedio de las dimensiones de la malla como se muestra en la Ilustración 6.

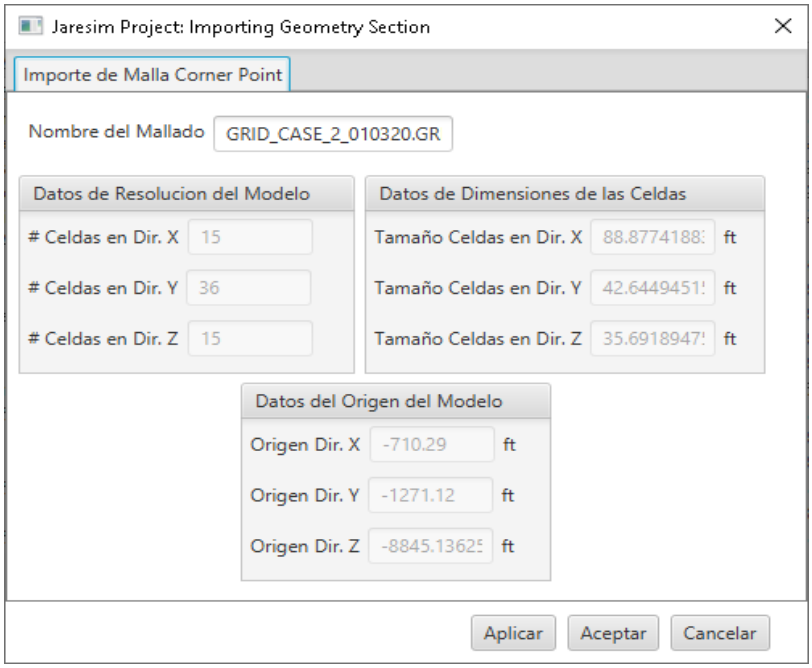


Ilustración 6:Ventana de la Sección Geometría del Modelo-Malla Corner Point Importada. Fuente: Elaboración propia.

4.1.2. Propiedades Estáticas

En esta sección se describirá todas las propiedades petrofísicas necesarias para realizar un proceso de simulación, y al final se mostrarán las diferentes opciones de incorporar estas variables:

La primera propiedad requerida a considerar será la porosidad efectiva, la cual se define como, la fracción de la roca ocupada por poros, y es una medida de la capacidad de almacenamiento de la roca. Existen varios tipos de porosidad, la porosidad total, la porosidad efectiva y la porosidad no efectiva. Para efectos de simulación, la porosidad usada es la efectiva, que hace referencia a la fracción de espacio poral conectado en la roca, la cual es la que genera la capacidad de flujo de fluidos de la misma (Ver Ecuación 1).

$$\emptyset_e = -\frac{Volumen\ Poroso_{conectado}}{Volumen\ de\ Roca\ Total}$$

Ecuación 1:Porosidad Efectiva

La segunda propiedad requerida a considerar en un simulador de yacimientos es, la Permeabilidad Absoluta, la cual de define como la capacidad de un medio poroso de permitir el flujo de un fluido a través del mismo, este parámetro fue descubierto por Henry Darcy en 1856, a través de su ley la cual puede ser vista en la Ecuación 2. Es una propiedad anisotrópica (varía en cada dirección) por lo cual deben ser definido 3 valores del mismo en la dirección de los ejes coordenados X, Y, Z.

$$v_{ox} = -\frac{k_{ox}}{\mu_o} * \frac{\partial P_o}{\partial x}$$

Ecuación 2:Ley de Darcy (1856) para flujo de Fluidos Incompresibles unidimensional

La tercera propiedad requerida corresponde al Net to Gross, el cual se define como la relación entre el espesor neto de arena respecto al espesor total del yacimiento. Este parámetro indica en una columna de roca el porcentaje de roca neta que permite el flujo de fluidos. Un valor de 1 significa que todo el espesor es poroso y permeable, y un valor de 0 indica que todo el espesor es impermeable y no permite el flujo. La ecuación de este parámetro se ilustra a continuación:

$$NTG = \frac{Espesor\ Neto\ de\ Arena\ Porosa\ y\ Permeable}{Espesor\ Total}$$

Ecuación 3:Concepto de Net to Gross (NTG)

La cuarta propiedad requerida corresponde a la Compresibilidad de la Roca, la cual se define como la variación del volumen poral de la roca asociada a un cambio de presión considerando una temperatura constante. Matemáticamente se define como:

$$c_r = \frac{1}{\emptyset} * \frac{\partial \emptyset}{\partial P}_{T=const}$$

Ecuación 4:Compresibilidad isotérmica de la roca.

La importancia de la Compresibilidad de la Roca es que permite conocer la variación del volumen poroso del yacimiento asociado a los cambios de la presión del yacimiento, por medio de la corrección de la porosidad por presión, a través de la siguiente expresión:

$$V_p(P) = V_p(P_i) * c_r(p) * (P_i - P)$$

Ecuación 5:Corrección del Volumen poroso debido al Cambio de Presión del Yacimiento.

Finalmente, serán descritas las Opciones de Ingreso de las Propiedades Petrofísicas para la porosidad, las permeabilidades X, Y, Z, y el NTG pueden ser ingresados mediante 3 vías en la aplicación:

1. Mediante la aplicación, estas pueden puede ser ingresada en la ventana que se observa en la Ilustración 7, como un parámetro que puede cambiar por capa, ingresado de forma manual.

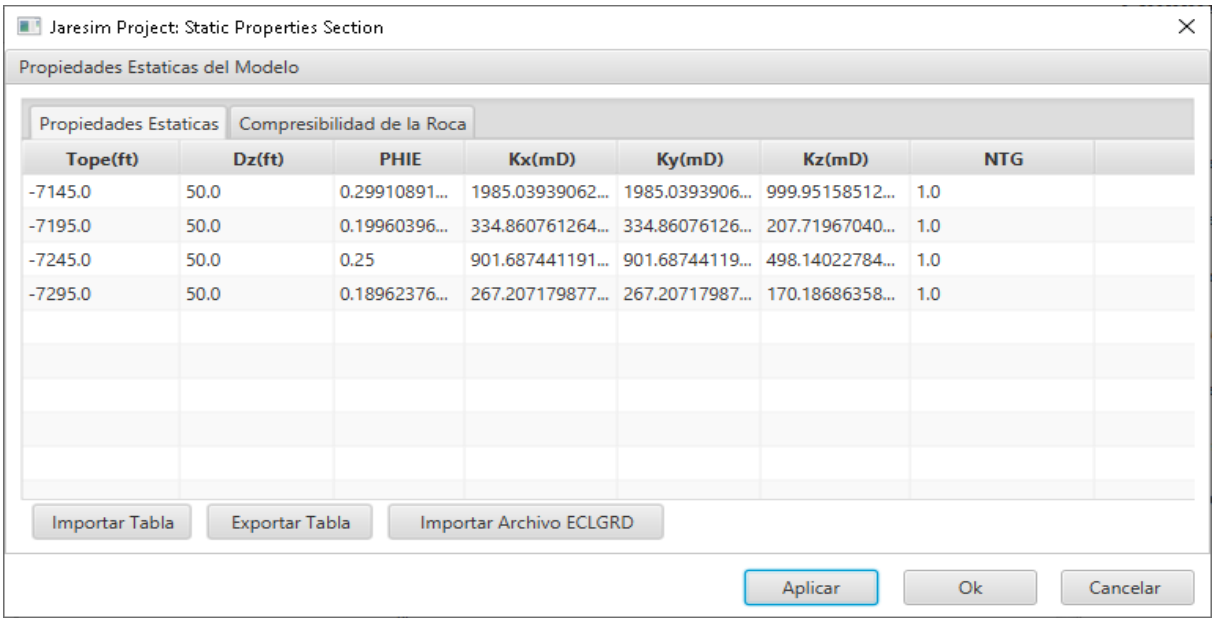


Ilustración 7:Ventana de la Sección Propiedades estáticas del Modelo-Ingreso de Forma Manual. Fuente: Elaboración propia.

2. Mediante importe de archivos de extensión *.jaresim_statictable con una ventana browser de búsqueda del archivo. Este formato puede ser visualizado en el Anexo 10.8.2.

3. Mediante importe de archivos de extensión *. GRDECL con una ventana browser de búsqueda del archivo. Este formato puede ser visualizado en los Anexos 10.8.1 y 10.5.

Una vez importado el archivo, los resultados cargados se evidencian en el formato presentado en la Ilustración 8.

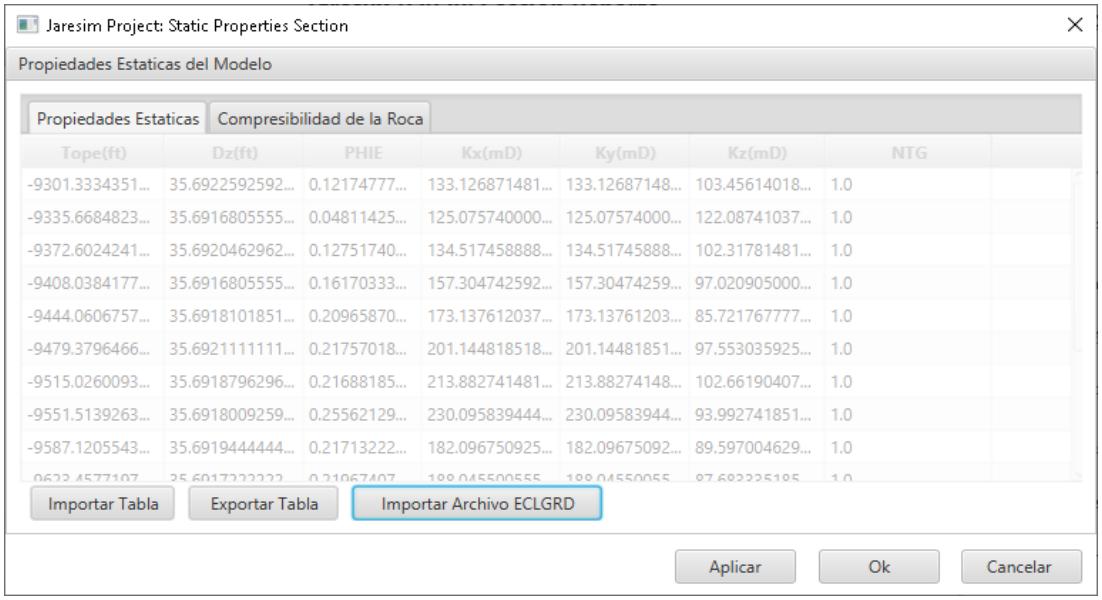


*Ilustración 8: Ventana de propiedades estáticas con propiedades cargadas en formato *. GRDECL. Fuente: Elaboración propia.*

Respecto a la compresibilidad de la roca, esta puede ser ingresada en la ventana presentada en la Ilustración 9 a continuación, como un parámetro que puede cambiar por presión.

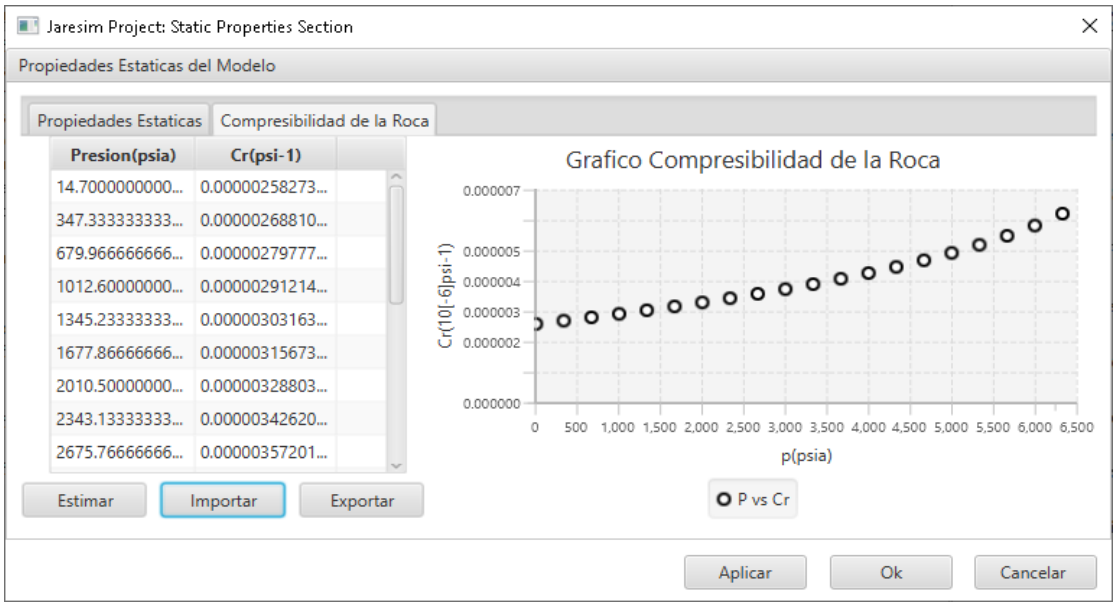


Ilustración 9:Ventana de la Sección Propiedades estáticas del Modelo-Compresibilidad de la roca. Fuente: Elaboración propia.

Además, puede ser ingresada como mediante el importe de un archivo de extensión *.jaresim_crtable mediante una ventana browser de búsqueda del archivo. Este formato puede ser visualizado en el Anexo 10.8.3.

Finalmente, si no se dispone de datos de compresibilidad de la roca, la aplicación dispone de tres modelos de correlaciones numéricas permite la estimación de las propiedades roca a través de correcciones. Las correlaciones disponibles en la aplicación se listan a continuación:

- Modelo en función de tipos de roca:
  - Correlación de Hall:

$$C_f = \frac{1.782 * 10^{-6}}{\emptyset}$$

Ecuación 6:Correlación de Hall para estimación de Compresibilidad. Fuente: (Ahmed, 2006).

  - Correlación de Newman:

$$C_f = \frac{a}{1 + c * b * \emptyset} \quad donde$$
$$a = 97.32 * 10^{-6}$$
$$b = 0.6999993$$
$$c = 79.8181$$

Ecuación 7:Correlación de Hall para estimación de Compresibilidad para areniscas calcáreas. Fuente: (Ahmed, 2006).

$$C_f = \frac{a}{1 + c * b * \emptyset} \quad donde$$
$$a = 0.8535$$
$$b = 1.075$$
$$c = 2.202 * 10^{6}$$

Ecuación 8:Correlación de Hall para estimación de Compresibilidad para Calizas y Dolomitas. Fuente: (Ahmed, 2006).

4.1.3. Modelo de Fluido

Respecto al modelo de fluido, se describirá los elementos más importantes asociados al modelo de fluido Black Oil, el cual fue usado para la definición de la propiedad del fluido del simulador desarrollado.

El modelo de fluido se define como la forma matemática de presentar las propiedades de un fluido en un simulador de yacimientos. En general existe dos principales enfoques, los cuales son: el modelo de fluidos Black Oil y el Modelo de fluidos Composicional. La presente

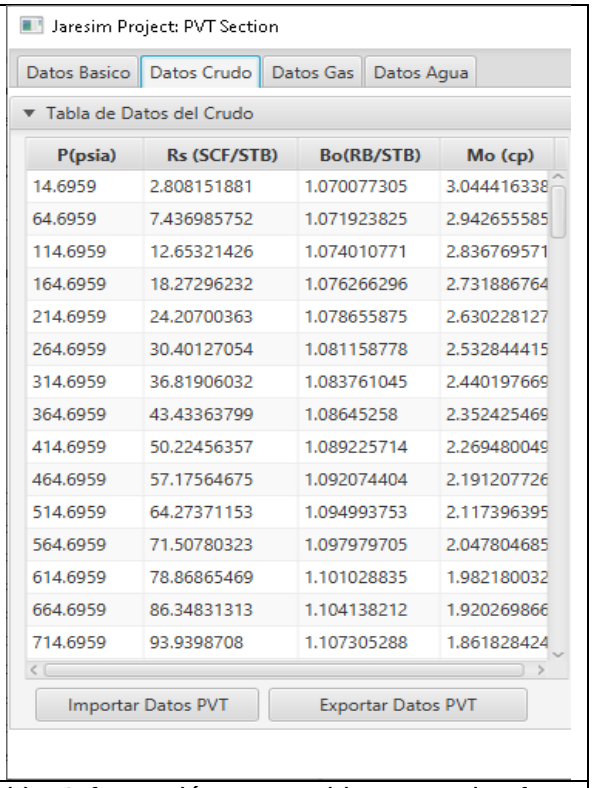
investigación está enfocada en el uso del modelo de fluido Black Oil el cual considera las siguientes premisas:

- Considera tres fases presentes (Petróleo, Agua, Gas).
- Proceso isotérmico.
- Yacimiento a condiciones de Presión y Temperatura.
- Cada fase solo tiene un Componente (Petróleo, Agua, Gas).
- No se consideran los fenómenos de transferencia de masa y/o componentes que pueden existir entre las fases petróleo y gas. No se consideran cambios en la composición de las fases petróleo y gas.
- Para la definición de las propiedades es necesario conocer:

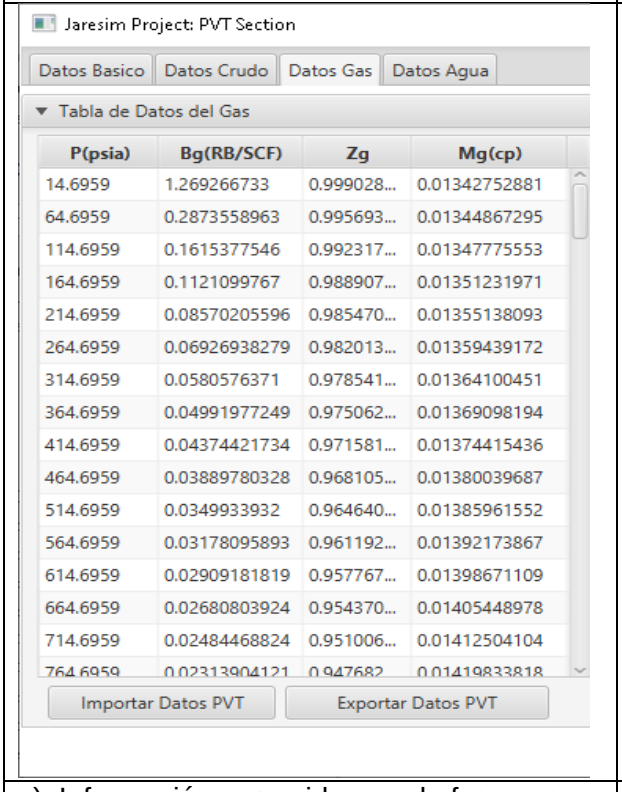
    - Temperatura.
    - Densidad del petróleo, gas y agua.
    - Presión de burbuja (Pb) y Solubilidad del gas en el crudo (Rs).
    - Para la fase petróleo es necesario definir las siguientes tablas: P vs Rs, P vs Bo, P vs Mo.
    - Para la fase gas es necesario definir las siguientes tablas: P vs Z, P vs Bg, P vs Mg.
    - Para la fase agua es necesario definir las siguientes tablas: P vs Rsw, P vs Bw, P vs Mw, donde el agua se considera ligeramente compresible en el modelo usado en este proyecto.

A nivel de la aplicación, las propiedades PVT (Presión-Volumen-Temperatura) pueden ser ingresadas en las ventanas presentes en la Ilustración 10, donde se evidencian diversas fichas que permite el ingreso de los parámetros correspondientes a cada fase. La Ilustración 10.a muestra los Datos Básicos PVT de la Aplicación, la Ilustración 10.b muestra la información requerida para la fase petróleo, la Ilustración 10.c muestra la información requerida para la fase gas, y finalmente, la Ilustración 10.d, muestra la información requerida para la fase agua.

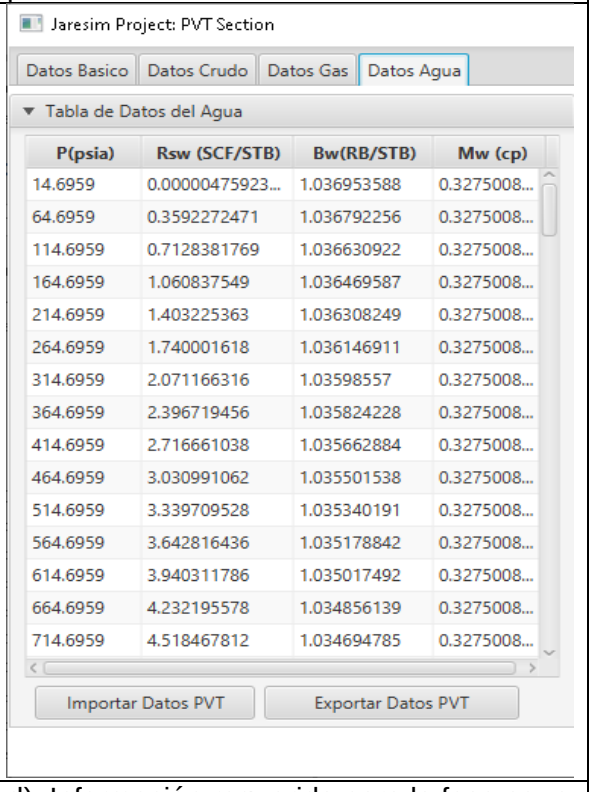
a). Datos Básicos de la Aplicación



b). Información requerida para la fase petróleo



c). Información requerida para la fase gas



d). Información requerida para la fase agua

Ilustración 10:Ventana de la Sección Propiedades PVT-Datos de las fases petróleo, gas y agua. Fuente: Elaboración propia.

La aplicación permite la visualizar las curvas de las propiedades ingresadas, como se presenta a continuación en las siguientes Ilustraciones 11,12 y 13:

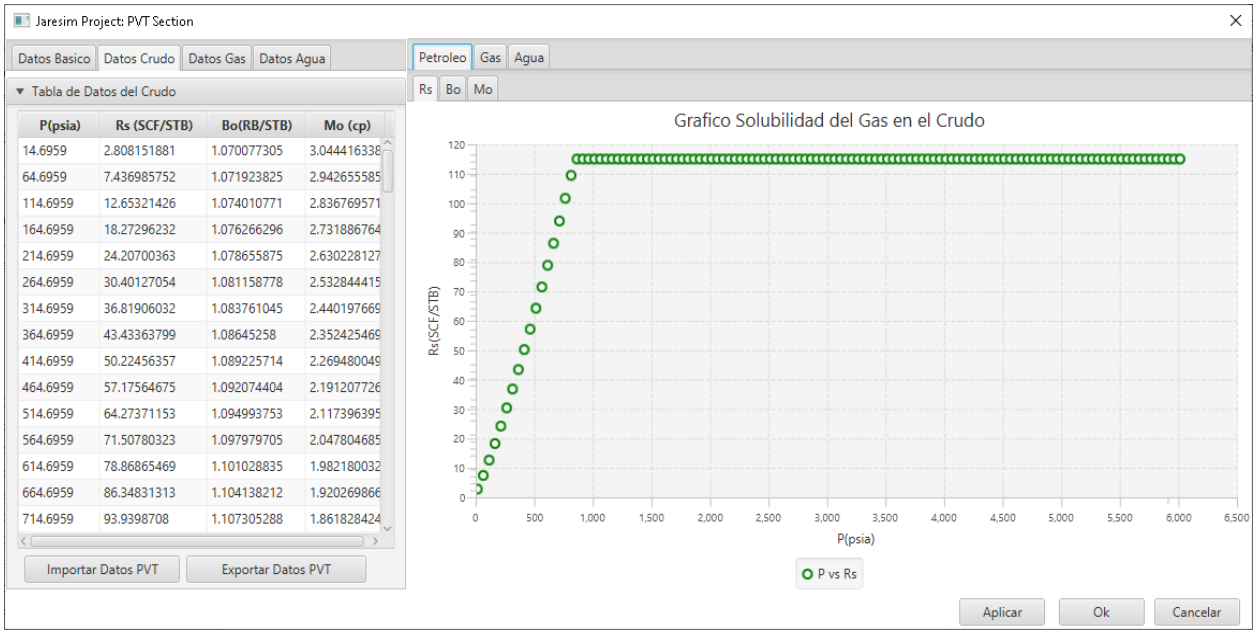


Ilustración 11:Ventana de la Sección Propiedades PVT-Gráficos de la fase Petróleo. Fuente: Elaboración propia.

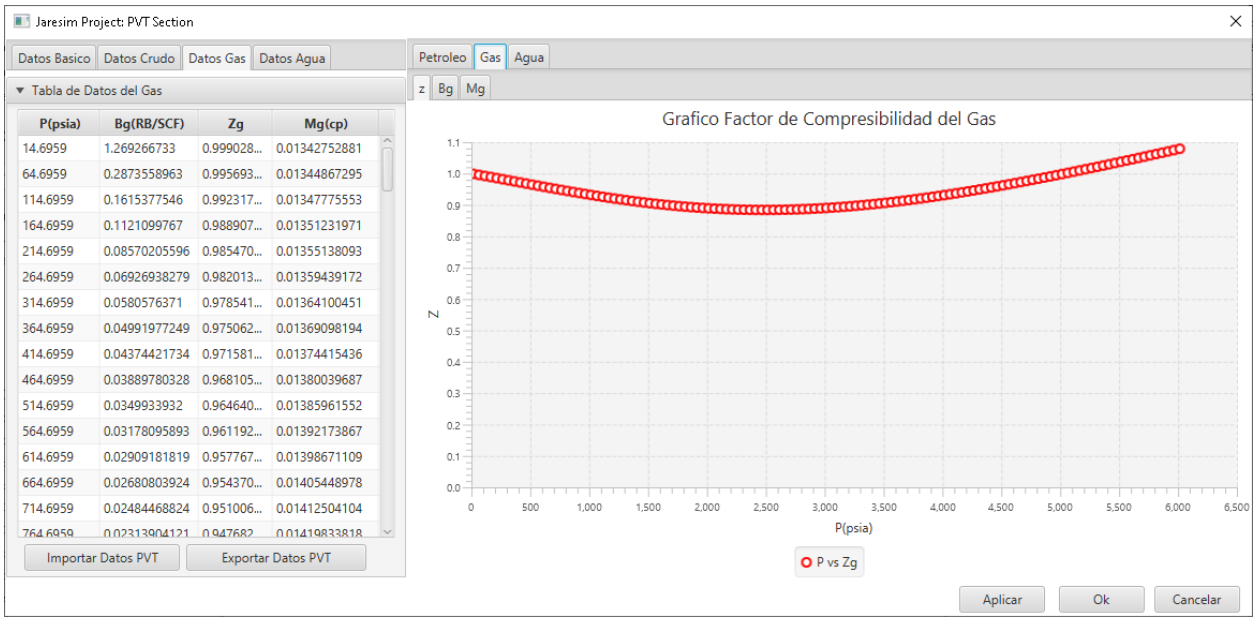


Ilustración 12:Ventana de la Sección Propiedades PVT-Gráficos de la fase Gas. Fuente: Elaboración propia.

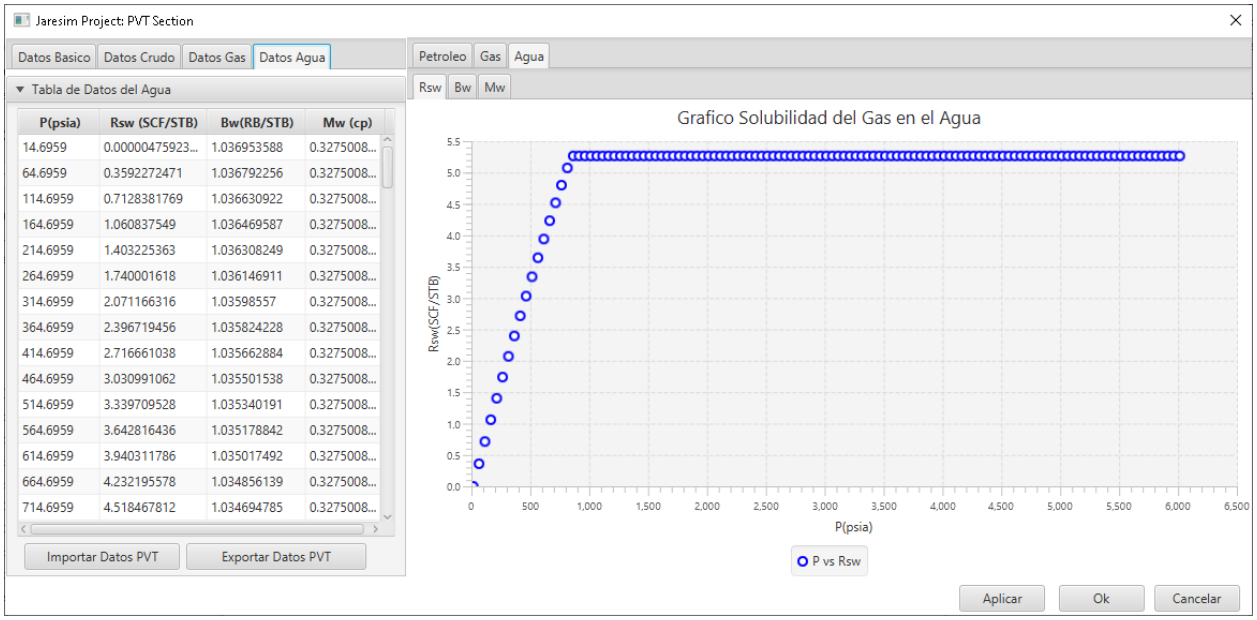


Ilustración 13:Ventana de la Sección Propiedades PVT-Gráficos de la fase Agua. Fuente: Elaboración propia.

Además la aplicación permite la carga del modelo de fluidos a través del importe de un archivo de extensión *.jaresim_pvtoiltable, así como *.jaresim_pvtwatertable y *.jaresim_pvtgastable los cuales pueden ser observados en el Anexo 10.8.4.

Finalmente, si no se dispone de un modelo de fluido, la aplicación tiene disponibles correlaciones numéricas para las fases petróleo agua y gas que permite la estimación de las propiedades PVT. Estas correlaciones que pueden verse en el Anexo 10.8.11.

4.1.4. Interacción Roca-Fluido

A continuación, se describirán los parámetros necesarios para la configuración del modelo roca-fluido del simulador desarrollado en este proyecto:

La primera propiedad requerida correspondiente a la sección roca-fluido es la Humectabilidad, la cual se define como la tendencia que tiene un fluido inmiscible, en presencia de otro fluido inmiscible de adherirse a la superficie de una roca. Esta propiedad se especifica en la aplicación como se observa en la Ilustración 14.

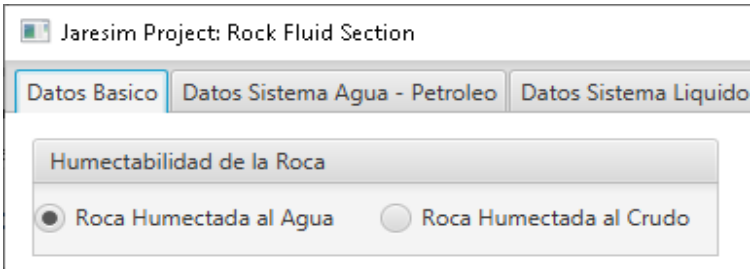


Ilustración 14:Ventana de la Sección Propiedades Roca Fluido- Datos de Humectabilidad. Fuente: Elaboración propia

El segundo aspecto a ser considerado en la sección roca-fluido, corresponde a las permeabilidades efectivas y relativas, primero será definida la permeabilidad efectiva, la cual se define como la capacidad que tiene una fase en particular en fluir a través del medio poroso de una roca, cuando está saturada por más de una fase a una saturación de fase en movimiento. Aunado a esto, se define como permeabilidades relativas, a la relación entre la permeabilidad efectiva a una saturación dada de la fase respecto a la permeabilidad absoluta de la roca. Matemáticamente se define como:

$$k_{r\,fase}(Saturacion) = \frac{k_{efectiva\,fase}\,(Saturacion)}{k}$$

Ecuación 9:Permeabilidad efectiva de fase en una roca.

La importancia de las permeabilidades relativas es que permiten obtener curvas normalizadas de las permeabilidades en función de saturación de la fase que almacena la roca. Estas permiten calcular la movilidad simultánea de todas las fases en movimiento dentro de la roca, respecto a sus correspondientes saturaciones, y esto a su vez permite calcular la velocidad relativa de cada fase respecto a la otra, en función de la saturación de cada fase.

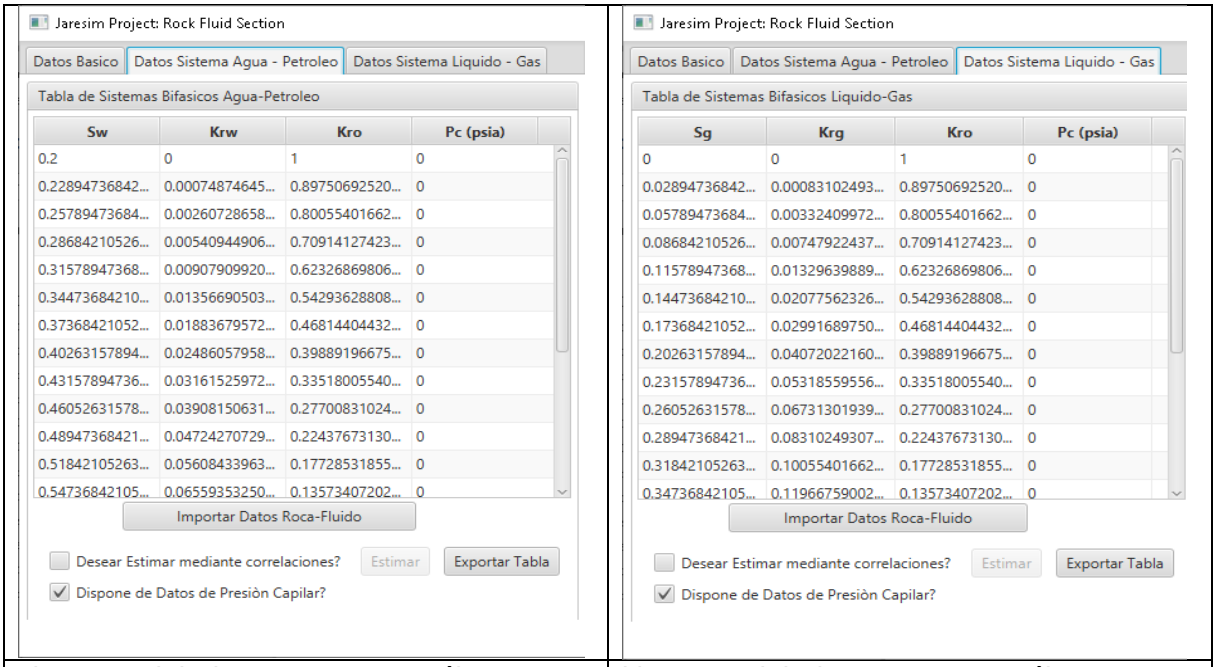
El tercer aspecto a ser considerado corresponde a la presión capilar, esta hace referencia a la diferencia de presión entre la fase no mojante y la fase mojante, matemáticamente se define con la siguiente expresión:

$$p_{c\,ow}(Saturacion) = p_o(Saturacion_o) - p_w(Saturacion_w)$$

Ecuación 10:Presión Capilar.

La principal utilidad de la presión capilar es que permite el cálculo de la distribución inicial de fluidos, cuando el reservorio presenta zonas de transición donde existe flujo simultáneo de más de una fase móvil.

La aplicación desarrollada en este proyecto permite el ingreso de las permeabilidades relativas, considerando las curvas para sistemas bifásicos Agua-Petróleo y Gas-Petróleo como puede ser observado en la Ilustración 15. La Ilustración 15.a muestra los Datos del Sistema Agua-Petróleo, y la Ilustración 15.b muestra la información de Datos del Sistema Gas-Petróleo:



a). Datos del Sistema Agua-Petróleo | b). Datos del Sistema Gas-Petróleo

Ilustración 15:Ventana de la Sección Propiedades Roca Fluido- Datos del Sistema Agua-Petróleo y Sistema Gas-Petróleo. Fuente: Elaboración propia

Debido a que, el flujo de fluidos en el medio poroso corresponde a un flujo simultáneo de las fases petróleo, gas y agua, y donde, las curvas de permeabilidades relativas bifásicas son insuficientes para representar el flujo simultáneo de las mismas. Por este motivo, Stone (Society of Petroleum Engineers, 2012-2020) desarrolló una serie de modelos que permiten mediante el uso de dos sistemas bifásicos, obtener el correspondiente modelo de 3 fases a través de los siguientes modelos:

- Modelo STONE I (1970).
- Modelo STONE II (1973) .

A nivel de la aplicación, el modelo de STONE puede ser especificado tal como se muestra a continuación en la Ilustración 16:

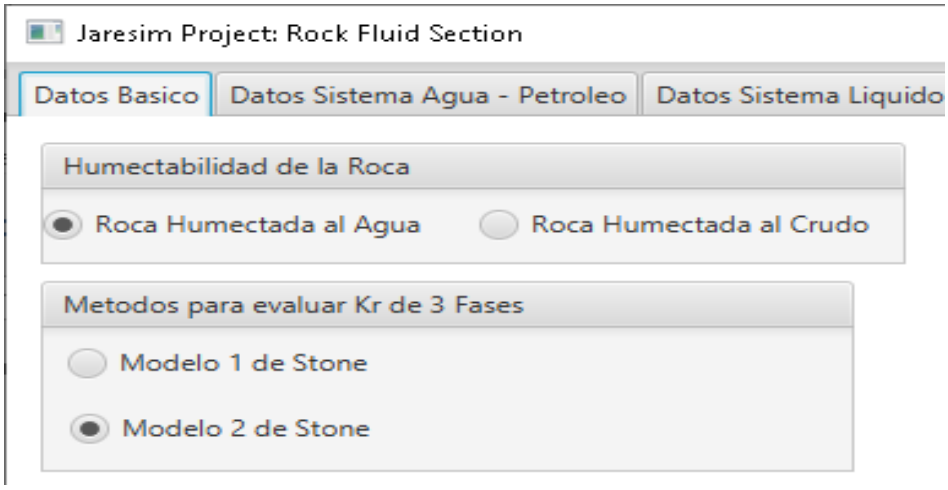


Ilustración 16:Ventana de la Sección Propiedades Roca Fluido- Modelo de STONE. Fuente: Elaboración propia.

Además, la aplicación permite la carga del modelo roca-fluido a través del importe de un archivo de extensión *.jaresim_krgotable y *.jaresim_krowtable los cuales pueden ser observados en el Anexo 10.8.5.

Finalmente, si no se dispone de un modelo roca-fluido, la aplicación dispone de dos modelos de correlaciones numéricas para los sistemas: agua-petróleo y gas-petróleo, permitiendo la estimación de las propiedades roca-fluido a través de correlaciones. Las correlaciones disponibles en la aplicación se listan a continuación:

- Modelo en función de tipos de roca:
  - Para Areniscas.
  - Para Caliza.

- Modelo General Corey (Society of Petroleum Engineers, 2012-2020).

Estas correlaciones pueden ser evidenciadas en el Anexo 10.8.12.

4.1.5. Inicialización del Modelo

La inicialización hace referencia al procedimiento matemático que permite la determinación a t=0 (Tiempo inicial de la corrida) de los parámetros que definen las condicionales iniciales del modelo. Los parámetros que son necesarios para esta definición son los mencionados a continuación:

- Presiones celda a celda en todo el modelo.

- Saturaciones celdas a celda de todo el modelo para las fases petróleo, agua y gas.

Para el proyecto, existen dos tipos de modelo de inicialización como se expone a continuación:

El primer tipo de inicialización disponible corresponde, a Equilibrio Hidrostático, el cual consiste en considerar equilibrio hidrostático donde conociendo los datos de presión al Datum o Profundidad de Referencia , Profundidad del Datum , Profundidad del Contacto Agua-Petróleo y Profundidad del Contacto Gas-Petróleo se puede calcular la distribución de presiones y saturaciones aplicando el concepto de ley de vasos comunicantes y equilibro hidrostático, además, es necesario el uso de los datos de presión capilar para la determinación del gradiente de saturación del modelo.

A nivel de la aplicación, el ingreso de la información para la inicialización por equilibrio se muestra en las Ilustraciones 17 y 18.

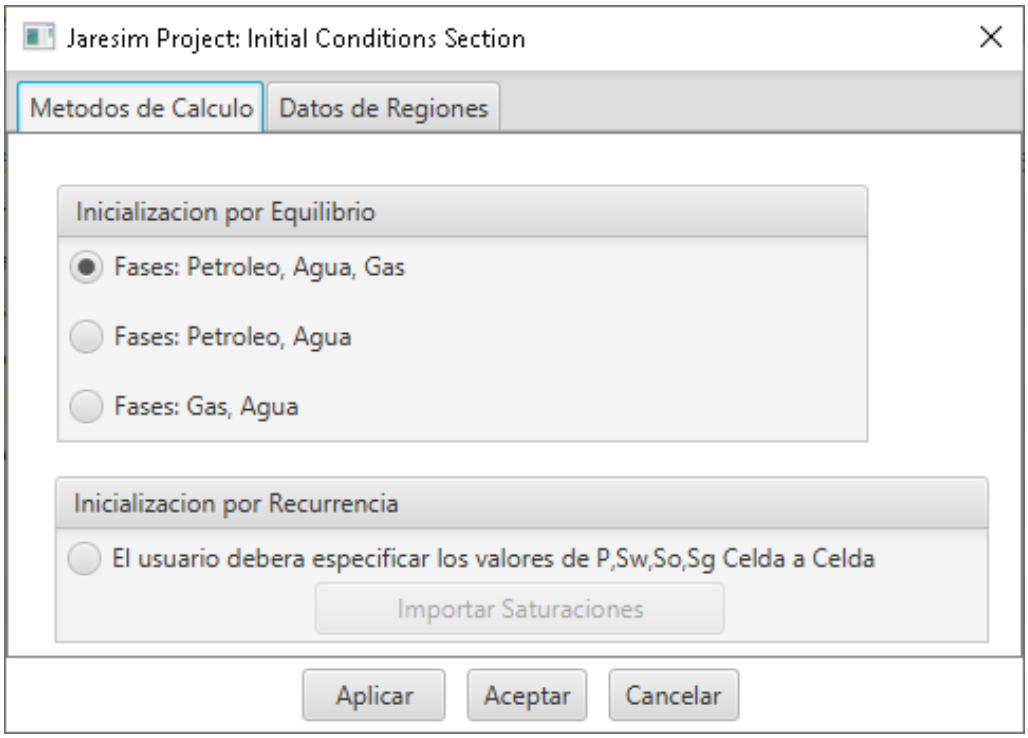


Ilustración 17:Ventana de la Sección Inicialización – Por Equilibrio-Método de Cálculo. Fuente: Elaboración propia.

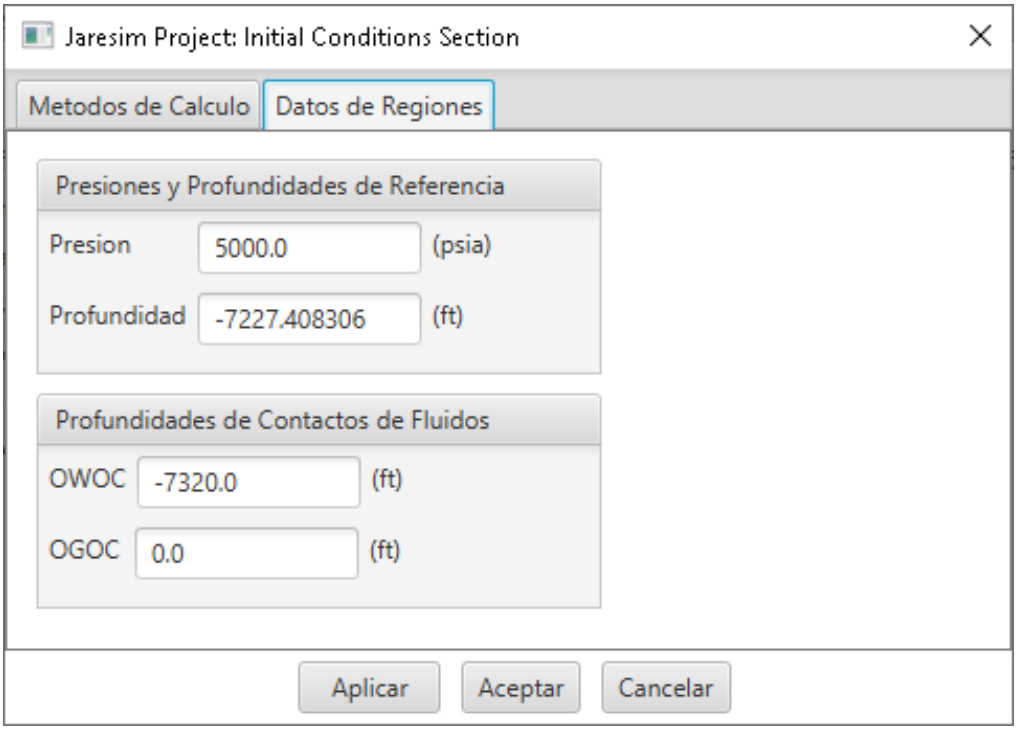


Ilustración 18:Ventana de la Sección Inicialización – Por Equilibrio-Datos de Regiones. Fuente: Elaboración propia.

Como puede ser observado en la ilustración anterior, se requiere definir valor de la profundidad de los contactos presentes en el yacimiento.

El segundo tipo de inicialización considerado en la aplicación se denomina, por Recurrencia o no equilibrio, donde de forma explícita se debe proporcionar al simulador las distribuciones celda a celda de los datos de presión, y las saturaciones de las fases petróleo, agua y gas. Para la ilustración del ingreso de la información para la inicialización por recurrencia se muestra la Ilustración 19.

Para ejecutar esta inicialización se requiere importar un archivo en formato *.jaresim_Properties_3dGrid_RestartProperties el cual puede ser evidenciado en el Anexo 10.8.6.

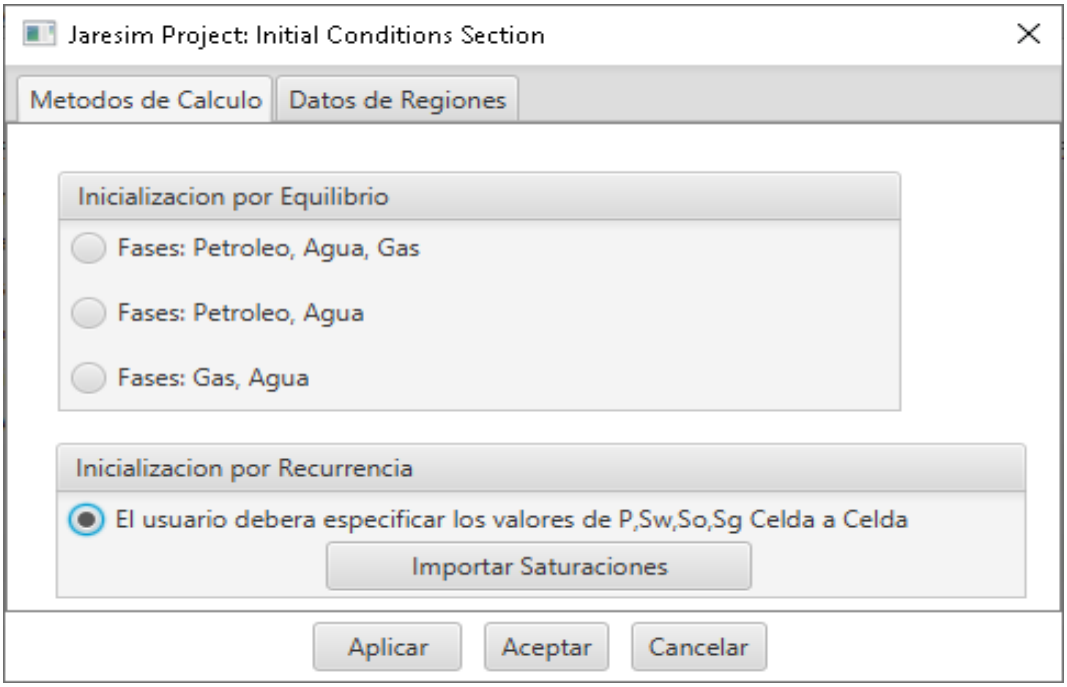


Ilustración 19:Ventana de la Sección Inicialización – Por Recurrencia. Fuente: Elaboración propia.

4.1.6. Información de pozos y Datos Recurrentes

En este apartado, se describirán las subsecciones a nivel de datos recurrentes, requeridos para el modelaje de pozo en la aplicación desarrollada en este proyecto:

El primer tipo de información recurrente requerida corresponde a Datos de Trayectoria de Pozos, que comprende la información correspondiente a los datos de trayectorias de los pozos que atraviesan el modelo. La aplicación requiere la carga de la siguiente información:

- Nombre del Pozo.
- Coordenada X del Pozo en pies(ft).
- Coordenada Y del Pozo en pies(ft).
- Coordenada Z del Pozo en pies(ft).
- Coordenada TVD (True vertical Depth) del Pozo en pies(ft).

A nivel de la aplicación, el ingreso de la información para la información de trayectorias de pozo se muestra en la Ilustración 20.

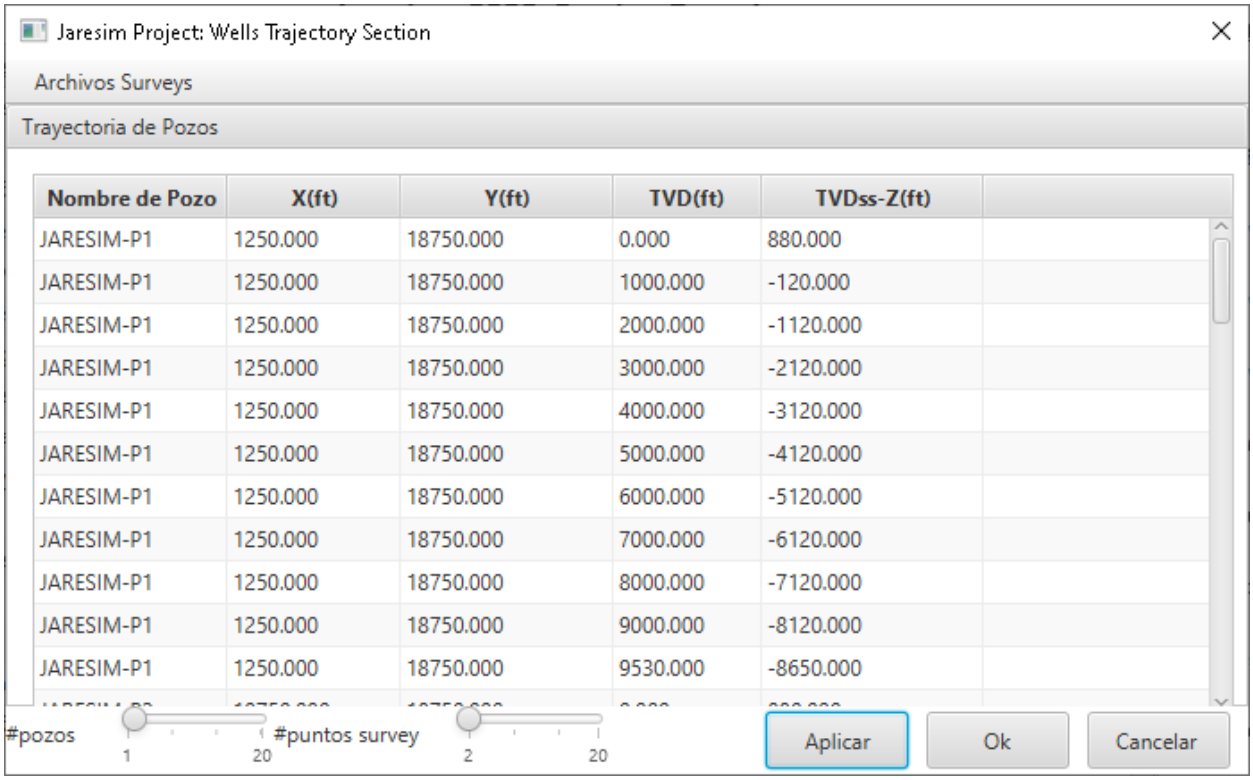


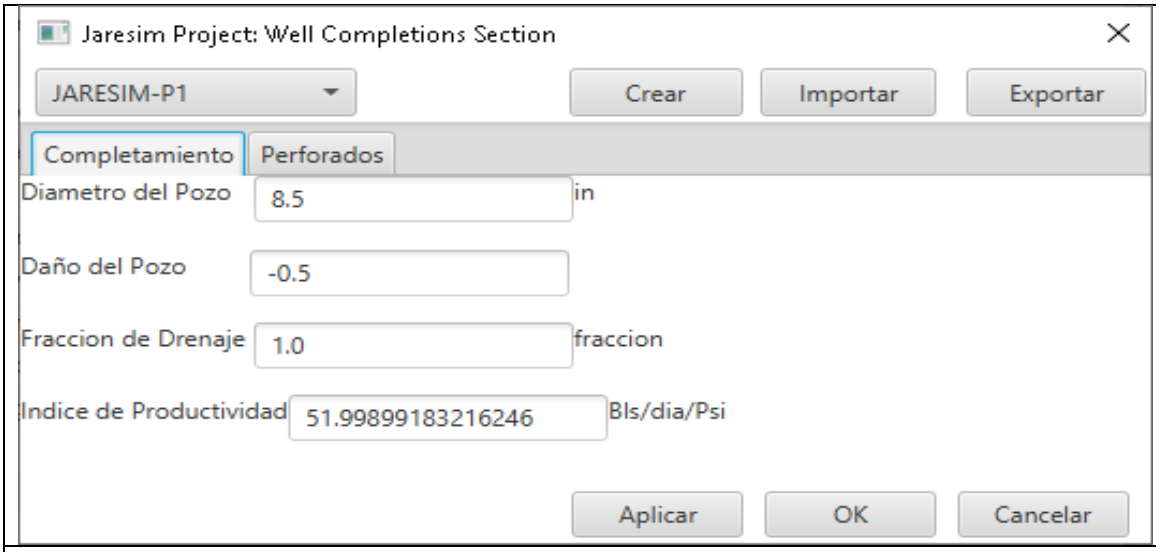
Ilustración 20:Ventana de la Sección de Trayectoria de Pozos – Datos de Trayectorias. Fuente: Elaboración propia.

Además, la aplicación permite el importe de trayectoria de pozos, el cual se realiza mediante la carga de un archivo de extensión *.jaresim_surveytable, que puede ser evidenciado en el Anexo 10.8.7.

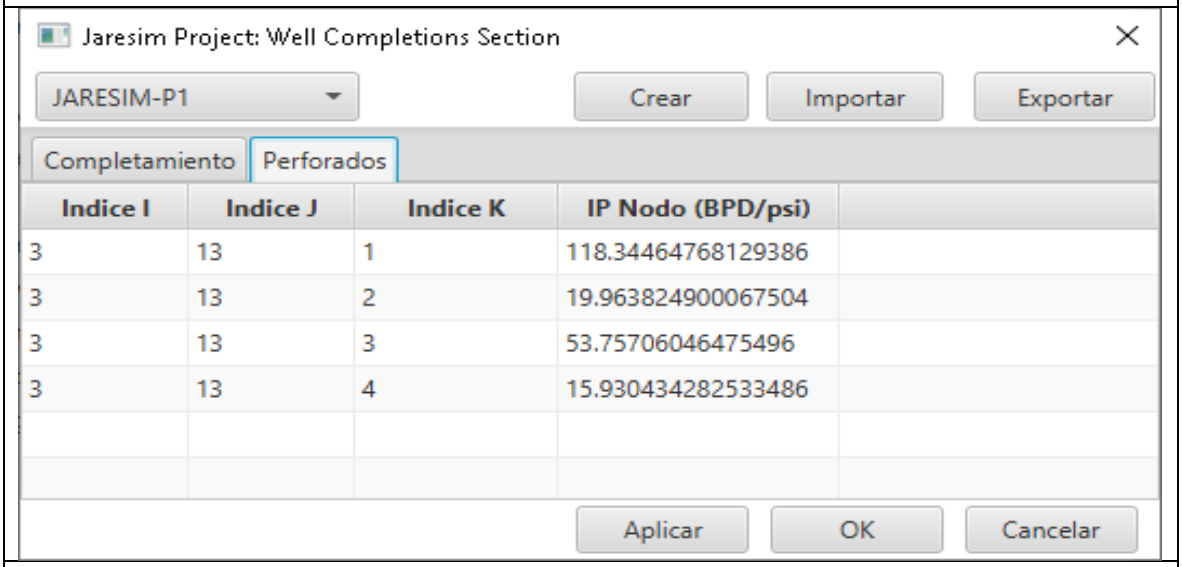
El segundo tipo de información recurrente requerido corresponde a Datos de Completamiento, el cual comprende la información asociada a los datos de los nodos de los pozos que se encuentra comunicados con la malla del modelo. Físicamente, representan el intervalo de cañoneo que los pozos tienen en la arena productora y son los cuales permiten transferir el fluido producido del yacimiento hacia el pozo.

El aplicativo permite calcular automáticamente que celdas son atravesadas por la trayectoria del pozo, mediante un algoritmo de búsqueda de celdas atravesadas por pozo se determinan los índices i,j,k de las celdas de la malla que son atravesadas por un pozo particular. La aplicación requiere la carga de la siguiente información a nivel del pozo para definir el modelo de completamiento (ver Ilustración 21):

- Diámetro del Pozo (rw) (pulg).
- Daño del pozo (S).
- Angulo de drenaje del pozo (wfrac)(°).



a).Datos de Completamiento del Pozo



b). Información de Nodos (Perforados) del pozo

Ilustración 21:Ventana de la Sección Completamiento de pozos-Datos de Completamiento. Fuente: Elaboración propia.

Con base a esta información (ver Ilustración 21.a) y los nodos (ver Ilustración 21.b) automáticamente calculados, esta aplicación calcula el índice de productividad (IP) para cada nodo en cada pozo a través del uso de la Ley de Darcy (1856) de estado Pseudo Estable de flujo radial:

$$q_{o\ xy} = IP_{xy} * (P_i - P_{wf}) \quad donde \quad IP_{xy} = -\frac{\sqrt{k_{ox} * k_{oy}} * \Delta Z * w_{frac}}{\mu_o * B_o} * \frac{1}{(\ln\left(\frac{r_{exy}}{r_w}\right) + S - \frac{3}{4})}$$

Ecuación 11:Cálculo de Índice de Productividad a partir de Ley de Darcy (1856) de flujo radial para Flujo de Fluidos Incompresibles.

Donde Re es el radio de drenaje del pozo en pies (re). Usualmente el radio de drenaje es calculado a través de la ecuación de Peaceman (CHEN & ZHANG, 2008), que se muestra a continuación:

$$r_{exy} = -\frac{0.14 * \sqrt{\sqrt{\frac{k_y}{k_x}} * \Delta x^2 + \sqrt{\frac{k_x}{k_y}} * \Delta y^2}}{0.5 * \left(\frac{k_y}{k_x}\right)^{\frac{1}{4}} * \Delta x^2 + \left(\frac{k_x}{k_y}\right)^{\frac{1}{4}} * \Delta y^2}$$

Ecuación 12:Ecuación de Peaceman para el cálculo del radio de drenaje equivalente para mallas anisotropías para el plano xy. Fuente: (CHEN & ZHANG, 2008).

El tercer tipo de información recurrente requerida corresponde a Datos Históricos, esta comprende la información asociada a los datos históricos de producción e inyección de los pozos. La aplicación requiere al momento de la carga de la información, se defina el tipo de pozo, los cuales pueden ser:

- Pozos Productores.
- Pozos Inyectores.

A continuación, se presenta la Ilustración 22 que muestra dicha definición en la aplicación:

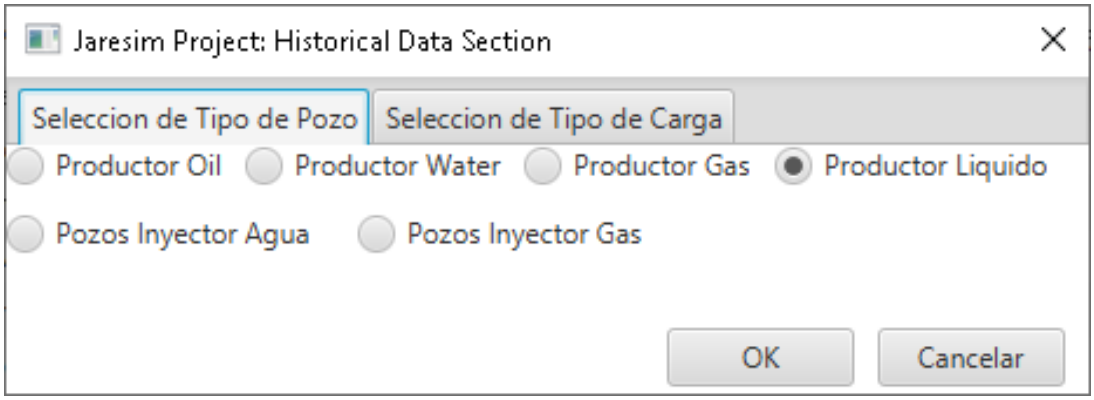


Ilustración 22:Ventana de la Sección Datos Históricos de pozos-Importe de Datos Históricos-Definición de Tipo de Pozo. Fuente: Elaboración propia.

Para los pozos productores, se requiere la carga de la siguiente información:

- Nombre del Pozo.
- Fecha.
- Tasa de Petróleo (BOPD).
- Tasa de Agua (BWPD).
- Tasa de GAS (SCPD).

- Tasa de Liquido (BFPD).
- Presión de Fondo Fluyente (psia).

Al momento de la carga de los datos, debe especificar el control primario a ser usado en la simulación, en función de los datos cargados tal como se muestra en la Ilustración 23.

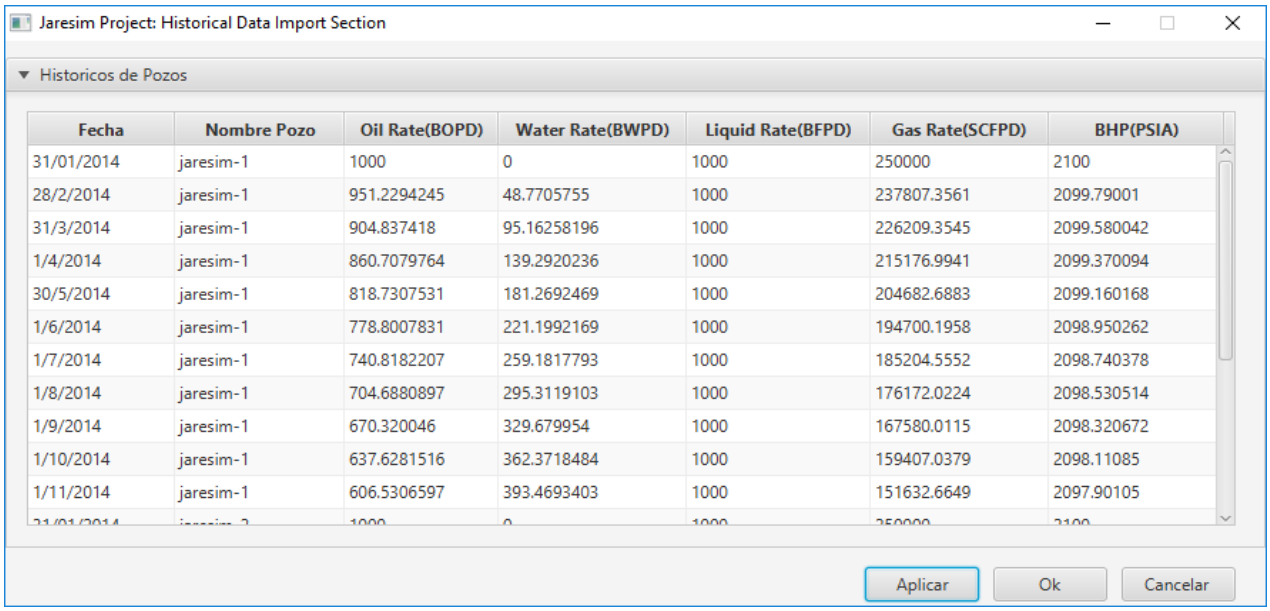


Ilustración 23:Ventana de la Sección Datos Históricos de pozos-Importe de Datos Históricos-Definición del Control primario de producción de simulación de pozos. Fuente: Elaboración propia.

Para el importe de datos históricos de producción de pozos, se debe cargar un archivo de extensión *.jaresim_producersRatetable, en cual puede ser evidenciado en el Anexo 10.8.8.

Para los pozos inyectores, la aplicación requiere la carga de la siguiente información:

- Nombre del Pozo.
- Fecha.
- Tasa de Agua Inyectada (BWIPD).
- Tasa de Gas Inyectado (SCIPD).
- Presión de Fondo de Inyección (psia).

Al momento de la carga de los datos, se debe especificar el control primario que será usado en la simulación, en función de los datos cargados tal como se muestra en la Ilustración 24.

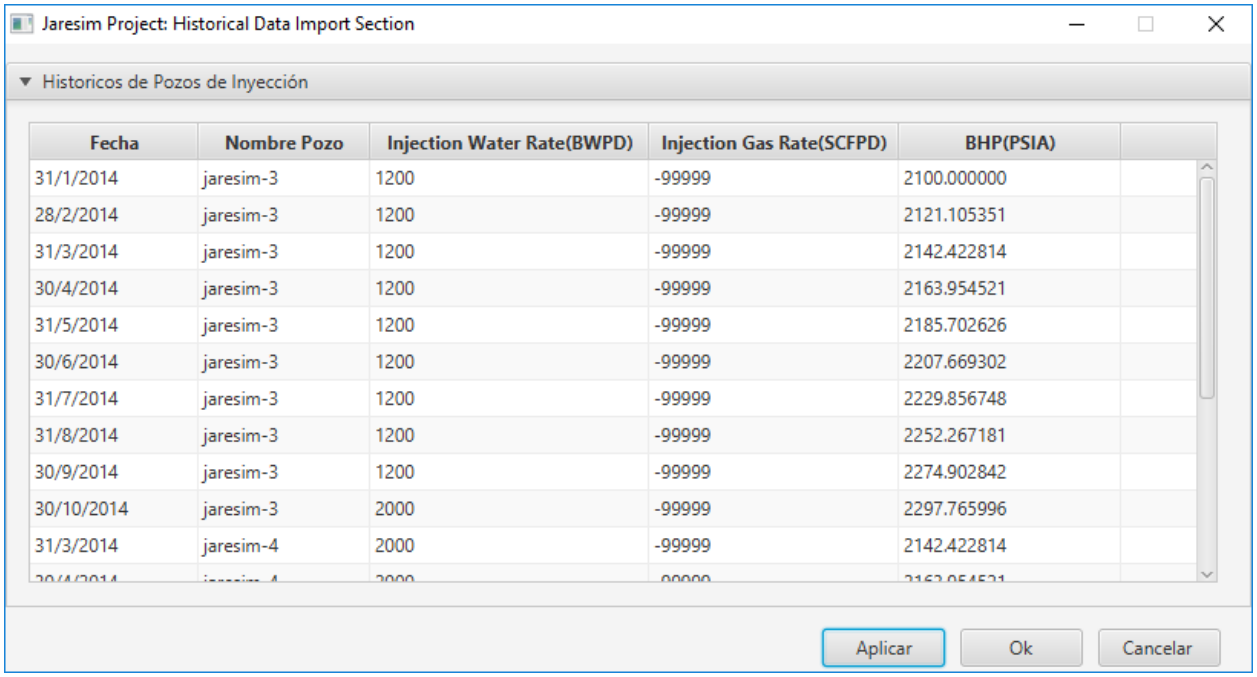


Ilustración 24:Ventana de la Sección Datos Históricos de pozos-Importe de Datos Históricos-Definición del Control primario de inyección de simulación de pozos. Fuente: Elaboración propia.

Para el importe de datos históricos de producción de pozos, se utiliza la carga de un archivo de extensión *.jaresim_injectorsRatetable, el cual puede ser evidenciado en el Anexo 10.8.9.

Es importante mencionar, que el formato de carga de los datos de producción e inyección requiere que todos los datos sean ingresados, sin dejar espacios vacíos, por lo cual cuando en una fecha no exista un valor de algún parámetro, para la definición de valor nulo deberá ser usado el valor -99999.

4.2. Configuración de Caso de Predicción

Esta sección contiene las subsecciones encargadas de:

- Configuración Predicción.
- Métodos Numéricos.
- Corrida.

Lo anterior, puede evidenciarse en la Ilustración 25, donde se observa la distribución de la sección Configuración del Proyecto:

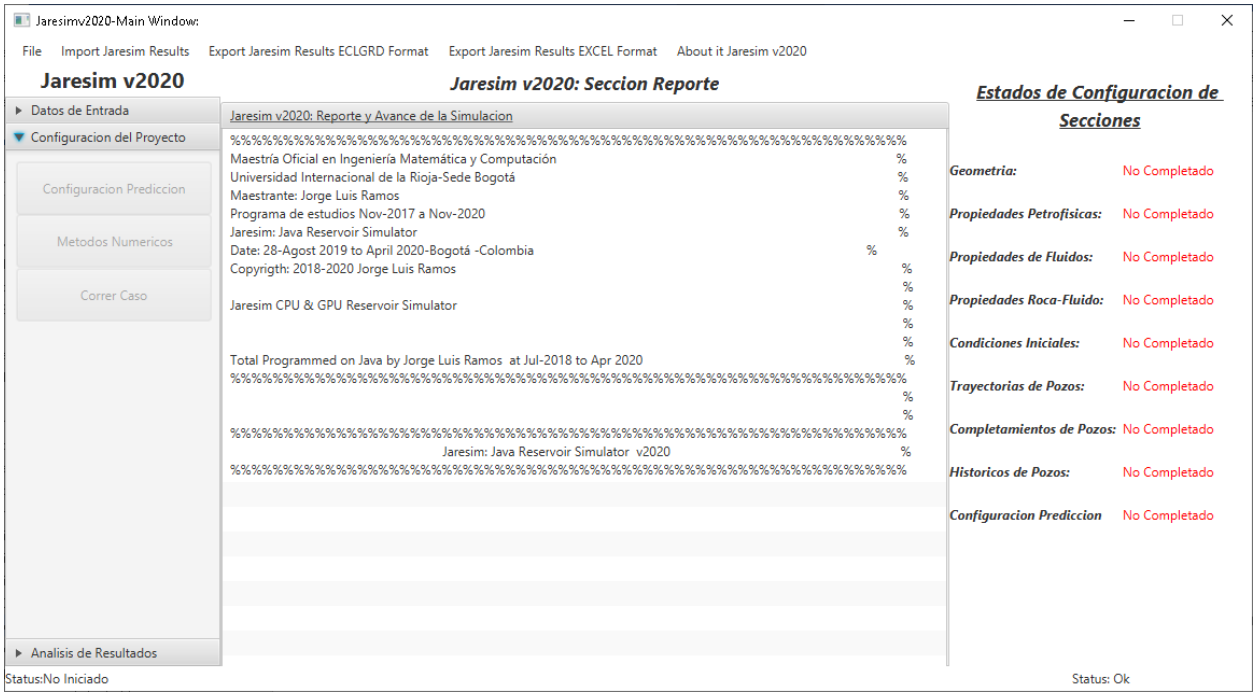


Ilustración 25:Ventana Principal de Simulador propuesto-Sección Configuración del proyecto. Fuente: Elaboración propia.

4.2.1. Configuración de Caso de Predicción

En esta subsección comprende la información de configuración que se quiere establecer para la predicción de los pozos. La aplicación requiere al momento de definir esta información que se establezca el tipo de pozo, los cuales pueden ser:

- Productores.
- Inyectores.

Para la configuración de los pozos productores, debe ser establecido el control primario que corresponde a tasas de Petróleo, Gas, Agua o Líquido, lo cual puede ser observado en la Ilustración 26:

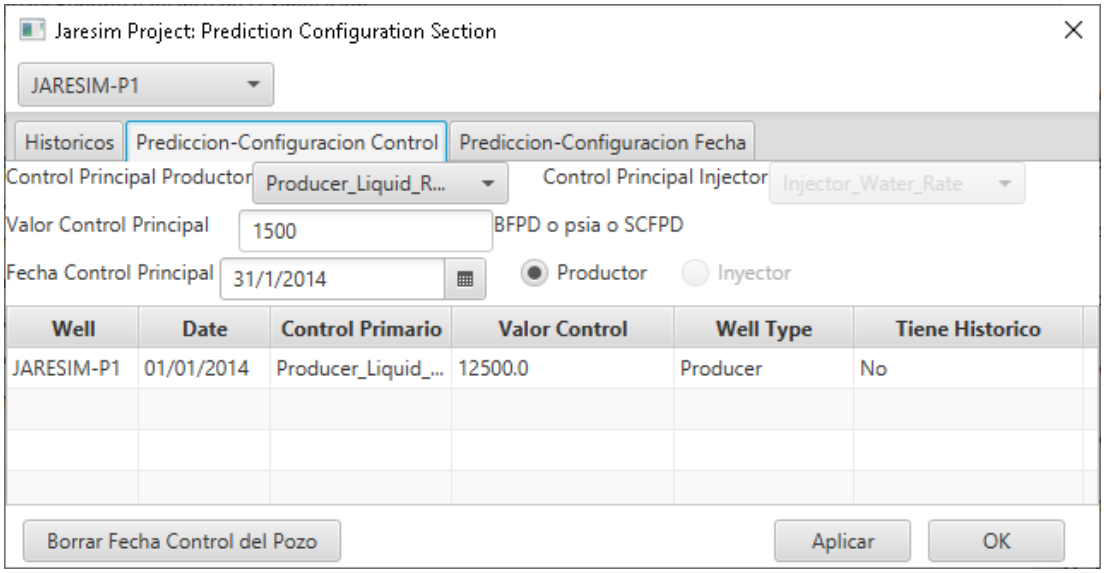


Ilustración 26:Ventana de la Sección Configuración de Predicción-Definición de parámetros para pozos productores. Fuente: Elaboración propia.

Para la configuración de los pozos inyectores, debe ser establecido el control primario que corresponde a tasa de inyección de Gas o Agua, lo cual puede ser observado en la Ilustración 27:

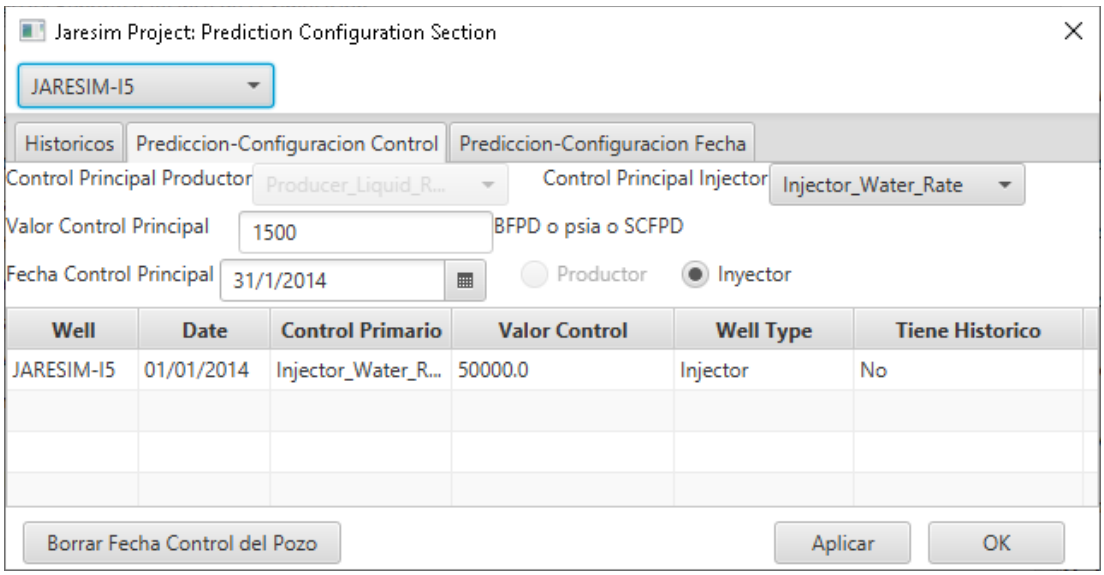


Ilustración 27:Ventana de la Sección Configuración de Predicción-Definición de parámetros para pozos inyectores. Fuente: Elaboración propia.

Adicionalmente, debe ser configurado el horizonte de predicción donde se debe establecer la fecha de inicio y fin de la predicción, así como la frecuencia de la predicción, y el método de cálculos de las presiones a nivel de pozo. En la Ilustración 28 esto puede ser visto:

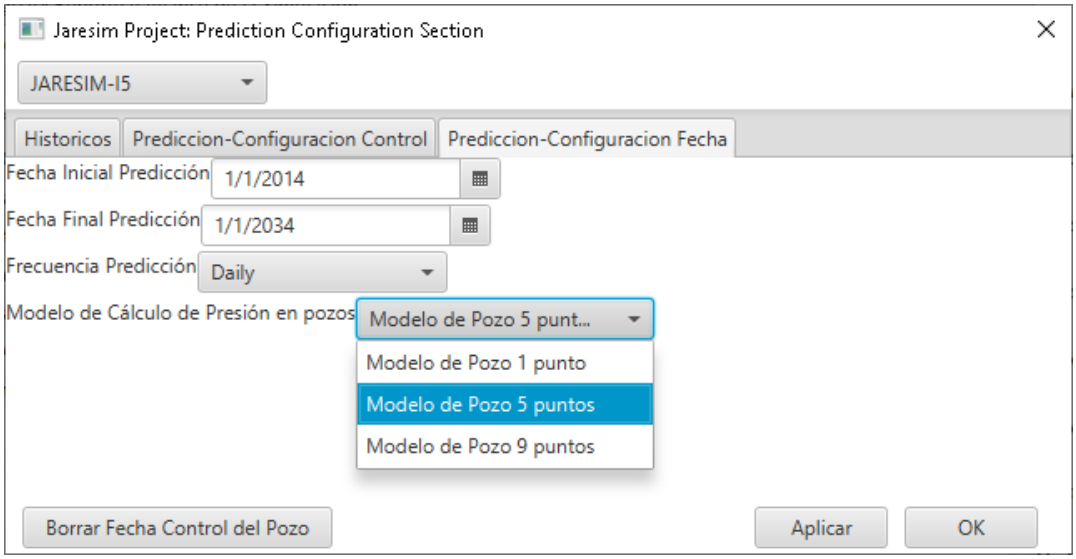


Ilustración 28:Ventana de la Sección Configuración de Predicción- Definición de Horizonte de Predicción del Modelo. Fuente: Elaboración propia.

4.2.2. Métodos Numéricos

En esta investigación, la aplicacion desarrollada permite establecer los parámetros de simulación que controlar la convergencia del mismo . A nivel del aplicativo, los parámetros que deben ser definidos en la configuracion general son los siguiente:

- #Max de Time Steps.
- Factor de Incremento de los Time Steps.
- Factor de Disminución de los Time Steps
- Duración de la Corrida.

Esta informacion puede ser vista en la Ilustración 29.

A nivel de resolución de sistemas lineales,se debe definir son los siguiente datos:

- #Max de Iteraciones.
- Para el método Relajación Lineal deben ser definidos los coeficientes de aceleración inicial del método.
- Se deben establecer los Cambios maximos permisibles en presión y saturación.
- Duración de la Corrida.

Esta informacion puede ser vista en la Ilustración 29.

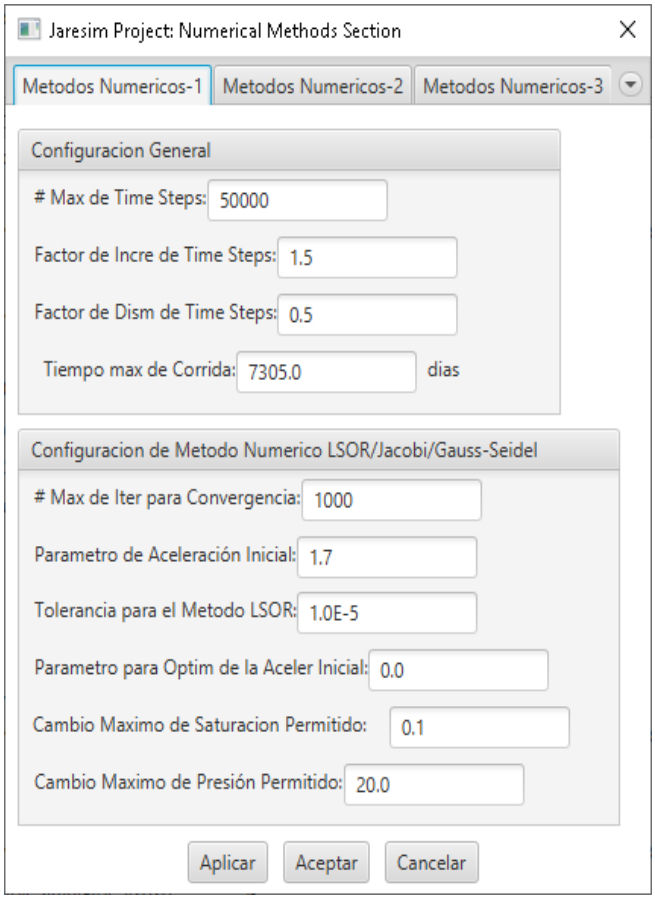


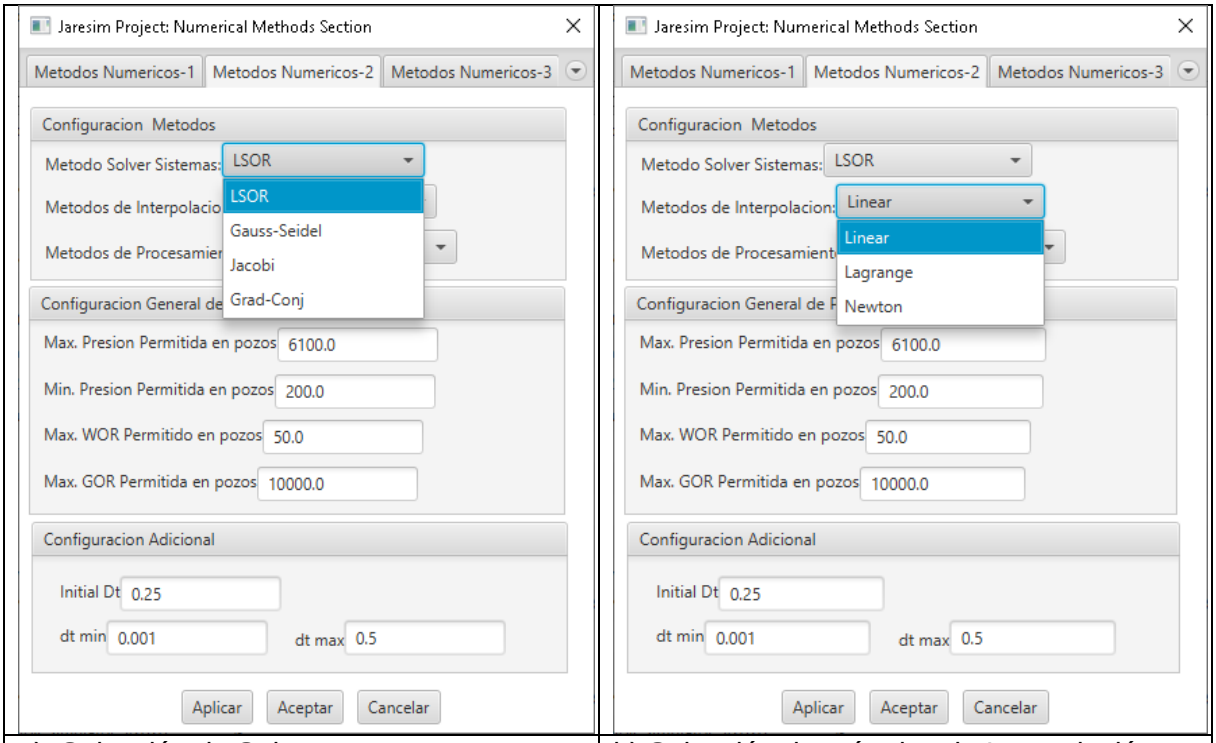
Ilustración 29:Ventana de la Sección Métodos Numéricos- Configuración General. Fuente: Elaboración propia.

A nivel de la configuracion de los métodos de solución, se debe definir lo siguiente:

- Método de Resolución de Sistemas Lineales : Gauss-Seidel, Jacobi, Matriz Tridiagonal y Gradiente Conjugado, en los cuales es posible activar el método de relajacion lineal.
- Los métodos de Interpolacion disponibles son: Lineal, polinomio de Newton y polinomio de Lagrange.

La información anterior puede ser vista en la Ilustración 30, que muestra en la Ilustración 30.a la selección de Solver y la Ilustración 30.b, muestra la selección de método de Interpolación en la aplicación.

Finalmente, el programa desarrollado tuvo como principal objetivo el desarrollo de un módulo de procesamiento Heterogéneo CPU-GPU, a través de uso de las librerías OpenCL. Sin embargo, y de manera complementaria, se desarrollaron dos módulos adicionales, uno a nivel de CPU, y otro módulo, de procesamiento paralelo CPU, para garantizar la mayor flexibilidad en la selección de un núcleo de procesamiento. Esta informacion puede ser vista en la Ilustración 31.



a). Selección de Solver

b).Selección de métodos de Interpolación

Ilustración 30:Ventana de la Sección Métodos Numéricos- Configuración de Métodos de resolución de sistemas lineales e Interpoladores. Fuente: Elaboración propia.

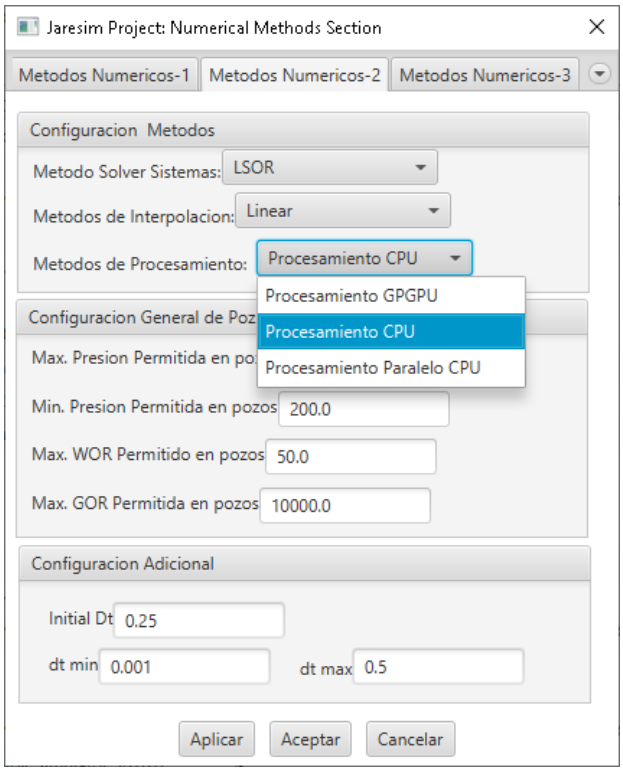


Ilustración 31:Ventana de la Sección Métodos Numéricos- Configuración de Núcleo de procesamiento CPU y Heterogéneo CPU_GPU. Fuente: Elaboración propia.

A nivel del procesamiento Heterogéneo GPU mediante JOCL, la aplicación permite especificar los siguientes ítems:

- Plataforma de procesamiento: Corresponde a la marca de unidad de procesamiento disponible.
- Tipo de Unidad de procesamiento: Se especifica si es CPU, GPU, Acelerador.
- # de Unidad de Procesamiento GPGPU[6]: Se especifica cuáles unidades de procesamiento serán usadas en la simulación, con base al filtro de la plataforma y el tipo de unidad.

La información anterior puede ser vista en la Ilustración 32, que muestra en la Ilustración 32.a la selección de Plataforma , la Ilustración 32.b muestra la selección del tipo de Dispositivo y finalmente, la Ilustración 32.c muestra la selección del Dispositivo en la aplicación. Los algoritmos asociados a los métodos numéricos (Interpolación y Resolución de Sistemas Lineales) podrán ser observados en el Anexo 10.9 de esta investigación.

Adicionalmente, en el capítulo 6 en la sección 6.3, podrá verse una breve descripción de los módulos en que está dividida la clase, diseñada para la ejecución del procesamiento GPU mediante diagramas de flujo. Adicionalmente, podrá verse en el mismo capítulo sección 6.4, la presentación de los códigos JOCL empleados para configurar cada uno de los módulos de la clase usada, para el procesamiento GPU de la aplicación.

[6] GPGPU: Procesamiento de Propósito General GPU

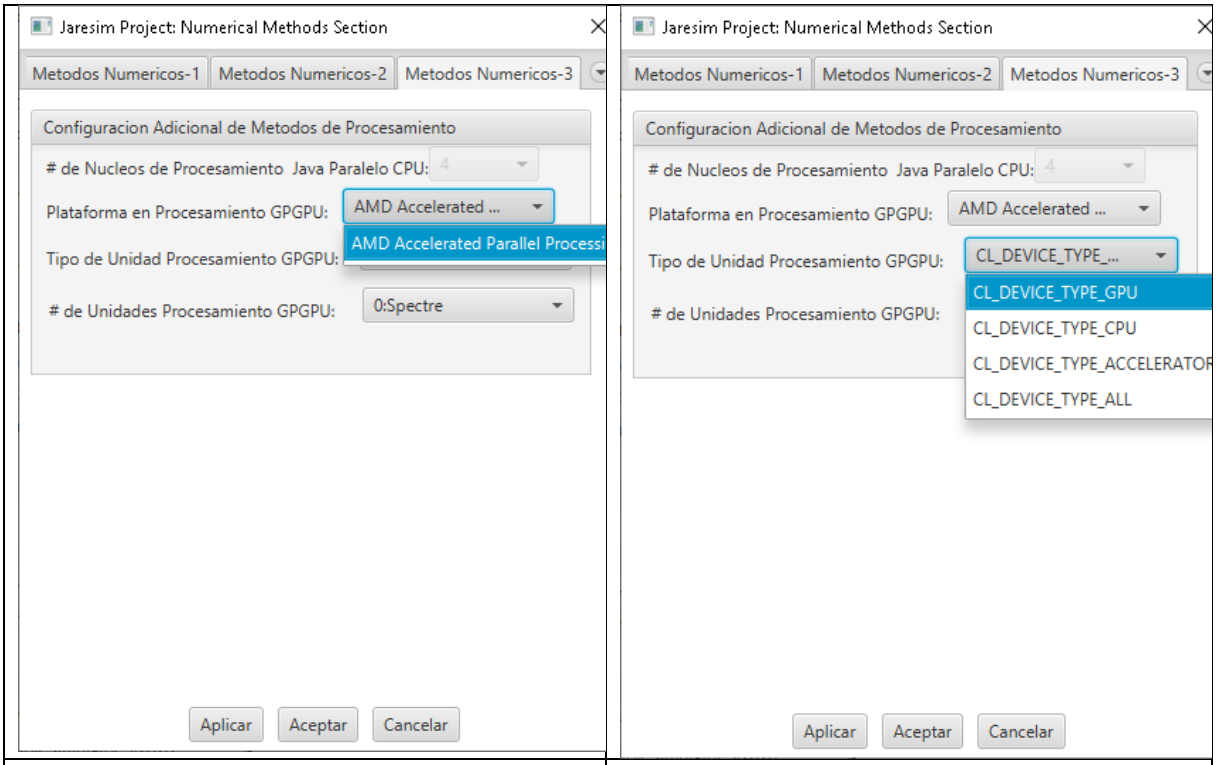


Ilustración 32:Ventana de la Sección Métodos Numéricos- Configuración de Núcleo de procesamiento Heterogéneo CPU_GPU. Fuente: Elaboración propia.

4.2.3. Corrida del Modelo

Finalmente, para ejecutar la simulación del caso se debe presionar el botón Correr Caso el cual puede ser visualizado en la Ilustración 33:

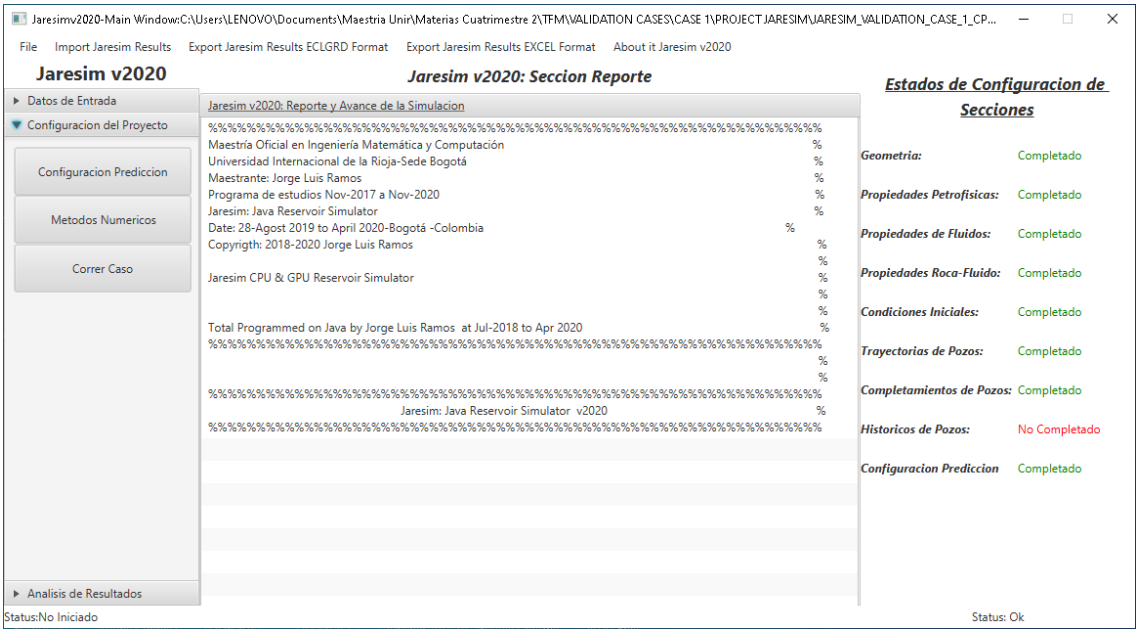


Ilustración 33:Sección Correr Caso. Fuente: Elaboración propia.

4.3. Análisis de Resultados

La sección correspondiente a la visualización de resultados tiene la capacidad de presentar los resultados en formato 2D y 3D como se muestra en la Ilustración 34:

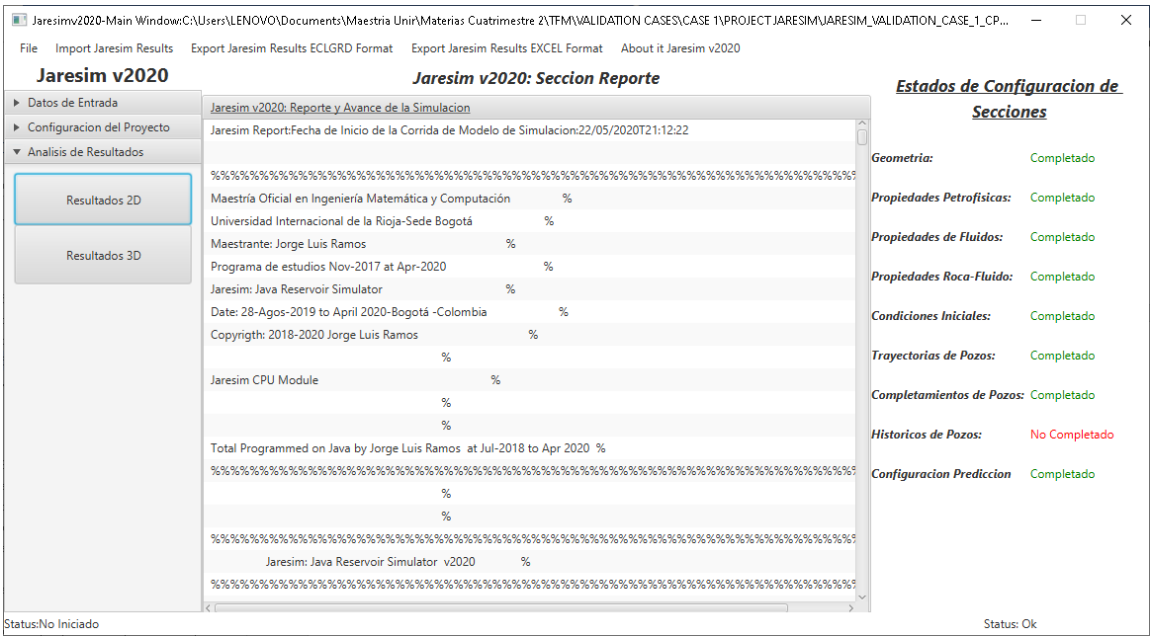


Ilustración 34:Ventana de la Sección Análisis de Resultados 2D-3D. Fuente: Elaboración propia.

La información 2D corresponde a la información relativa:

- Cambios Temporales de la Tasa de Fluidos Producidos (Petróleo, Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de la Tasa de Fluidos Inyectados (Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de la Presión de fondo observada a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de los Volúmenes producidos Acumulados (Petróleo, Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales de los Volúmenes inyectados Acumulados (Agua, Gas) a nivel de pozo y campo (Resultados 2D).
- Cambios Temporales en la presión promedia del yacimiento (Resultados 2D).

Estas curvas podrán ser vistas en las Ilustraciones que van desde la 35 a la 40, presentadas a continuación:

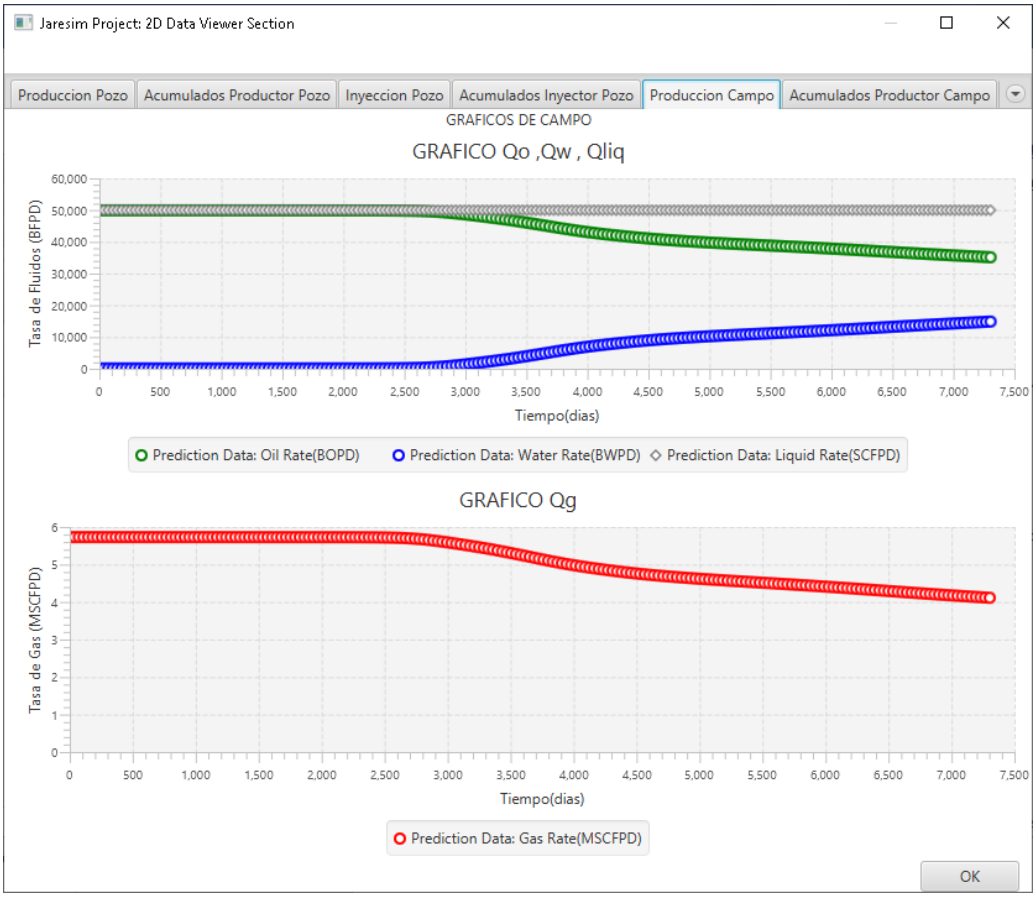


Ilustración 35:Sección Análisis de Resultados 2D: Resultados a Escala de Campo (Tasa). Fuente: Elaboración propia.

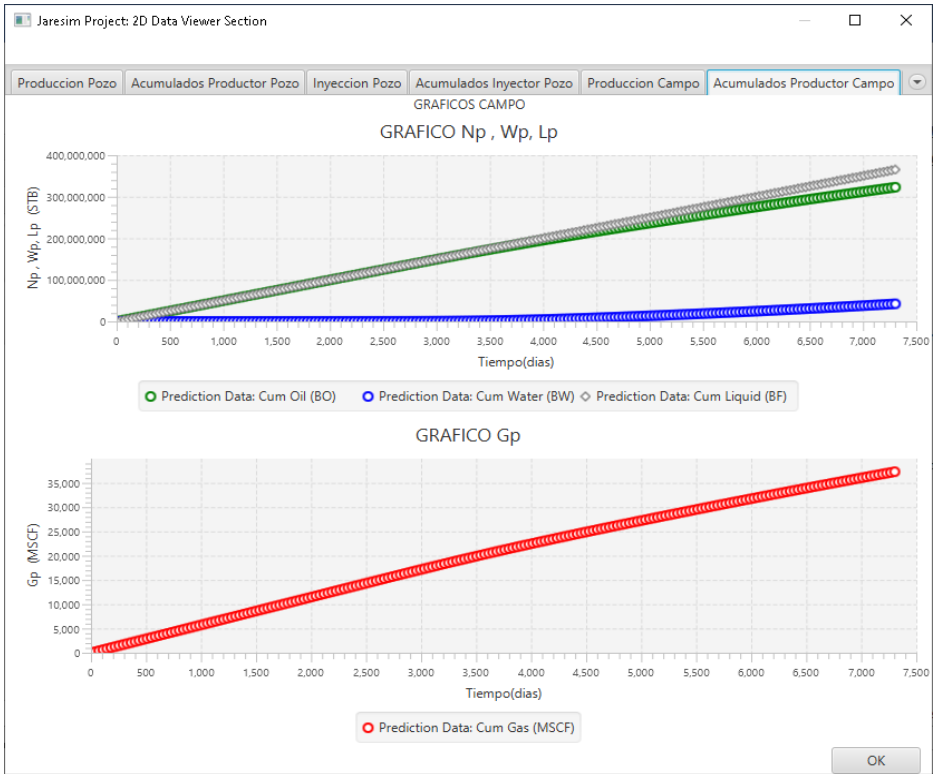


Ilustración 36:Sección Análisis de Resultados 2D: a Escala de Campo (Acumulados). Fuente: Elaboración propia.

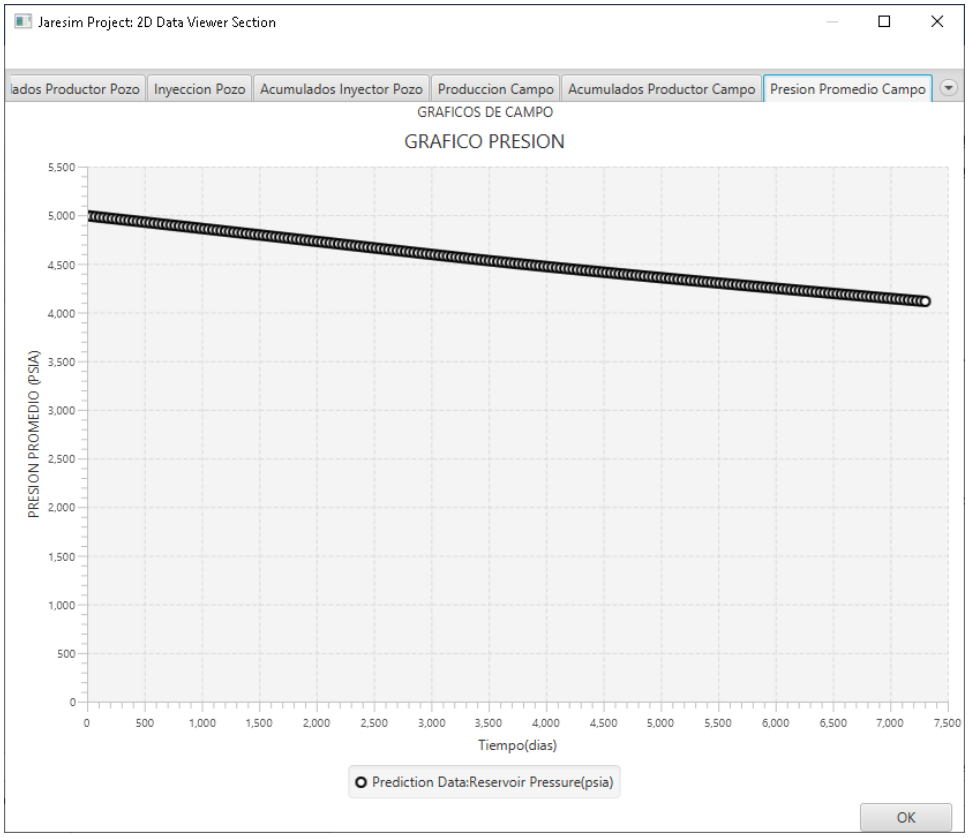


Ilustración 37:Sección Análisis de Resultados 2D: Resultados a Escala de Campo (Presiones). Fuente: Elaboración propia.

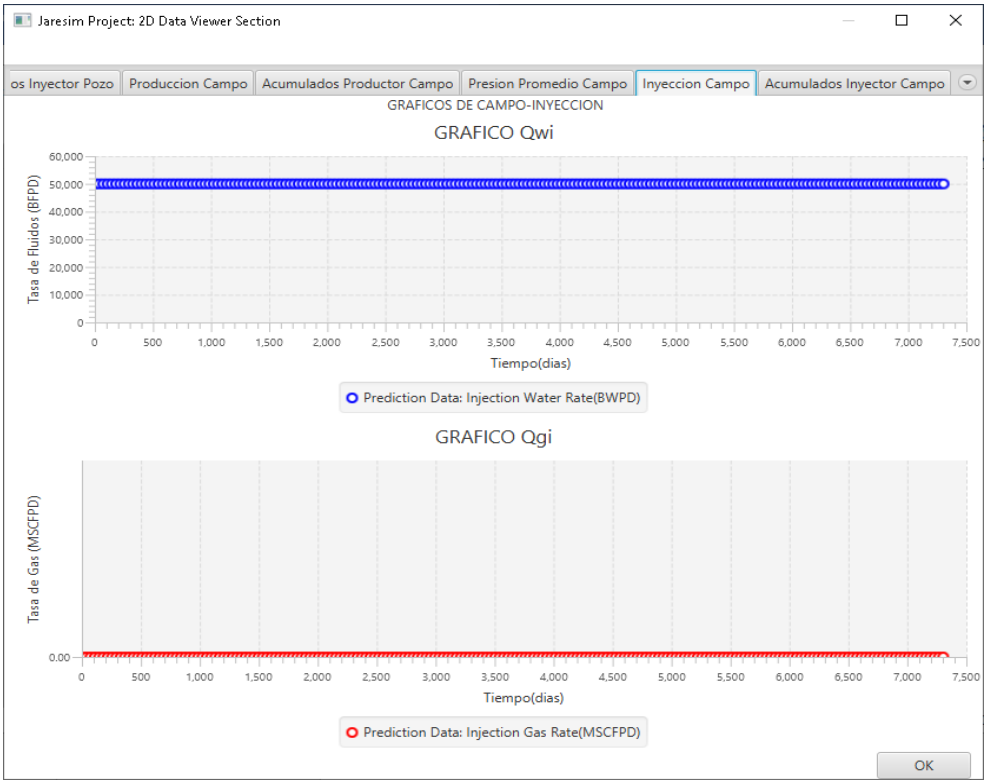


Ilustración 38:Sección Análisis de Resultados 2D: Resultados a Escala de Campo (Tasa de Inyección).Fuente: Elaboración propia.

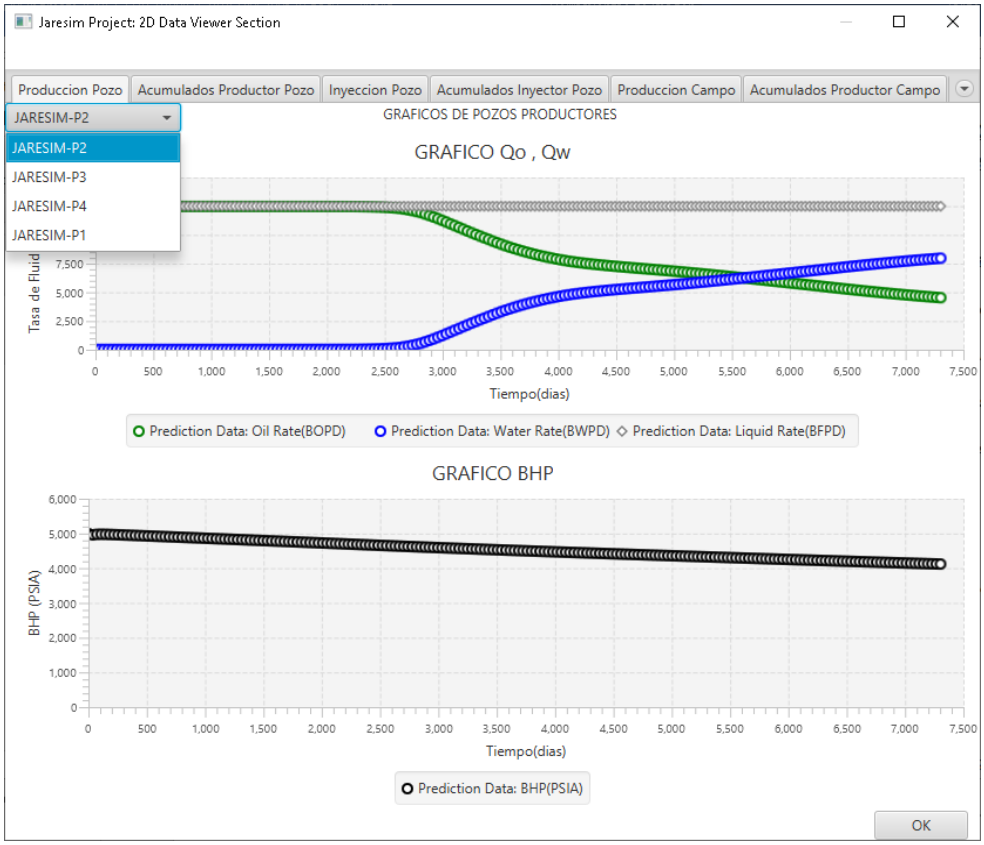


Ilustración 39:Sección Análisis de Resultados 2D: Resultados a Escala de Pozo (Tasa). Fuente: Elaboración propia.

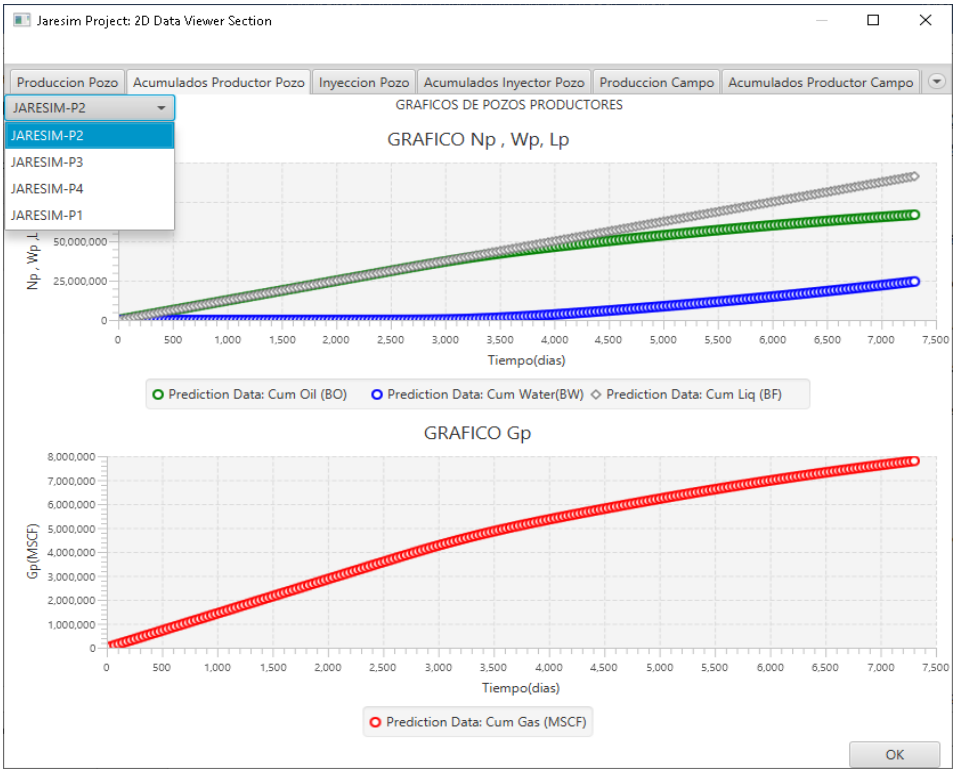


Ilustración 40:Sección Análisis de Resultados 2D: Resultados a Escala de Pozo (Acumulados). Fuente: Elaboración propia.

Adicionalmente, la información 3D que puede ser visualizada, corresponde a la información relativa a:

- Cambios temporales de las saturaciones (Petróleo, Agua, Gas) celda a celda en el yacimiento (Resultados 3D).
- Cambios temporales de las presiones celda a celda en el yacimiento (Resultados 3D).
- Datos básicos de Geometría (Dimensiones (X, Y, Z) y coordenadas (X, Y, Z) celda a celda en el yacimiento (Resultados 3D).
- Datos básicos de propiedades estáticas (porosidad, permeabilidad (X, Y, Z) y Net to Gross) (Resultados 3D).

A continuación, algunas de estas propiedades 3D pueden ser vistas en las Ilustraciones comprendidas de la 41 a la 45, presentadas a continuación:

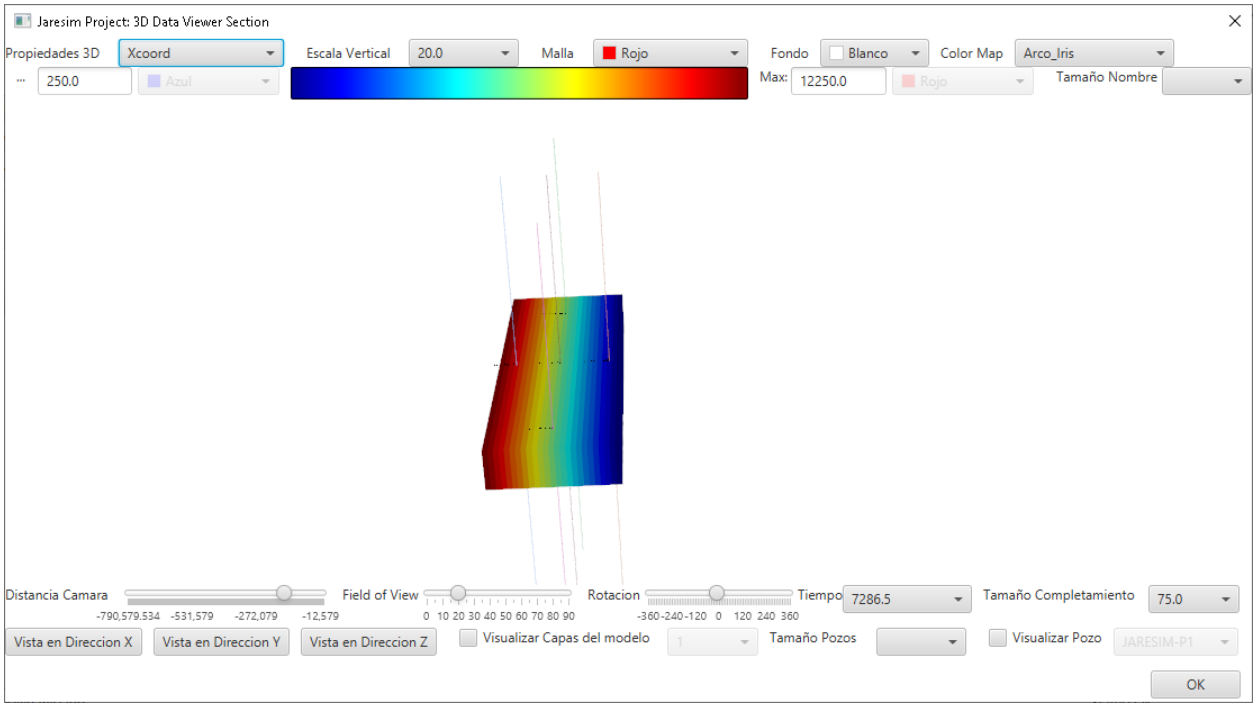


Ilustración 41:Sección Análisis de Resultados 3D: Malla de Coordenada X (Datos de Geometría). Fuente: Elaboración propia.

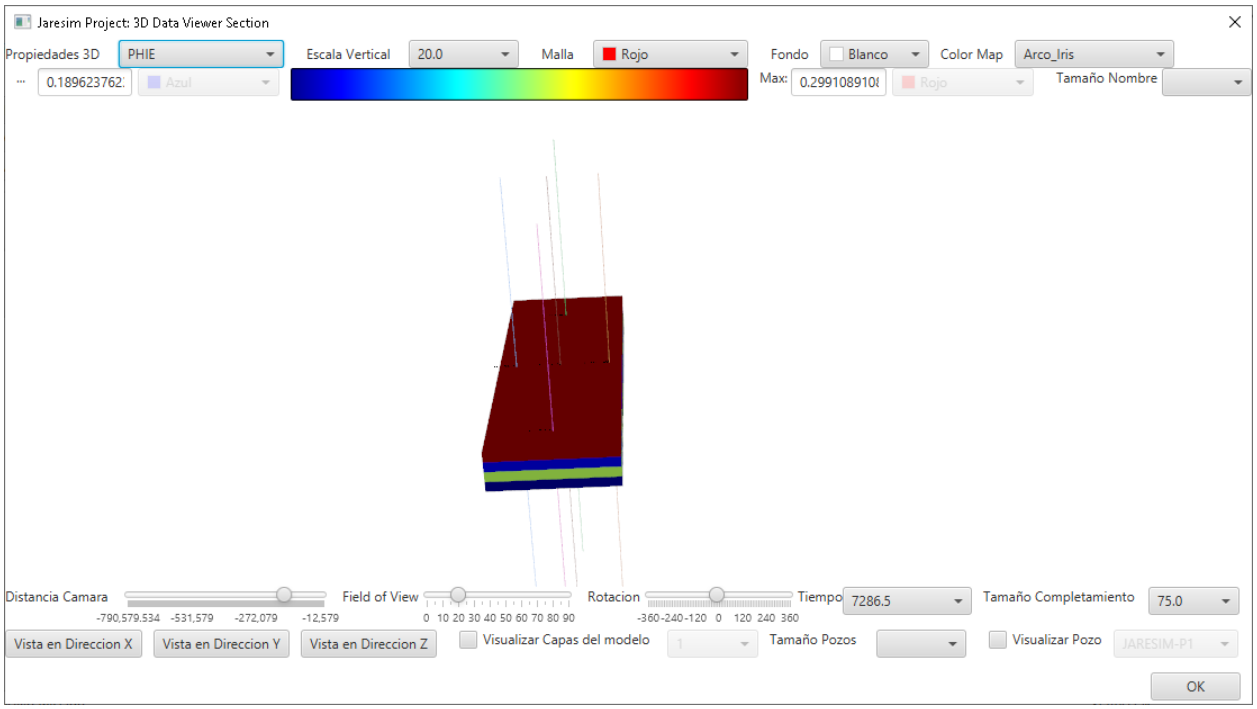


Ilustración 42:Sección Análisis de Resultados 3D: Malla de Porosidad Efectiva (Propiedad estática). Fuente: Elaboración propia.

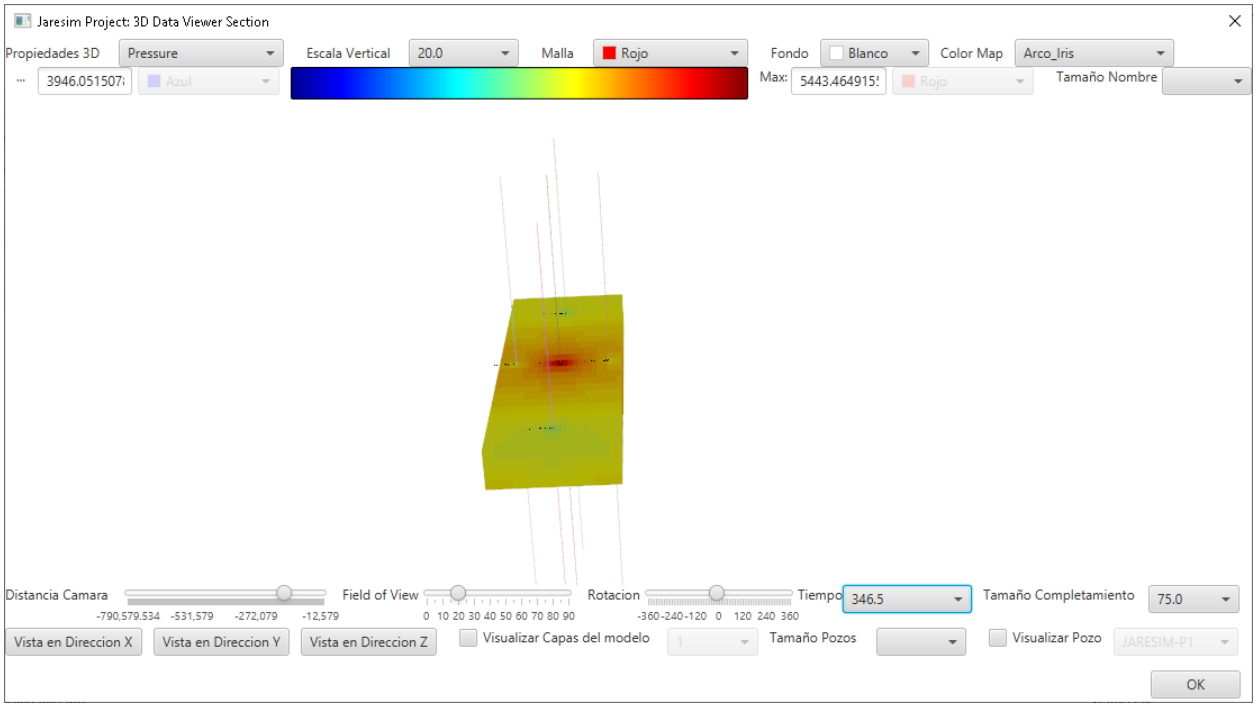


Ilustración 43:Sección Análisis de Resultados 3D: Malla de Presión (A t=346.5 días). Fuente: Elaboración propia.

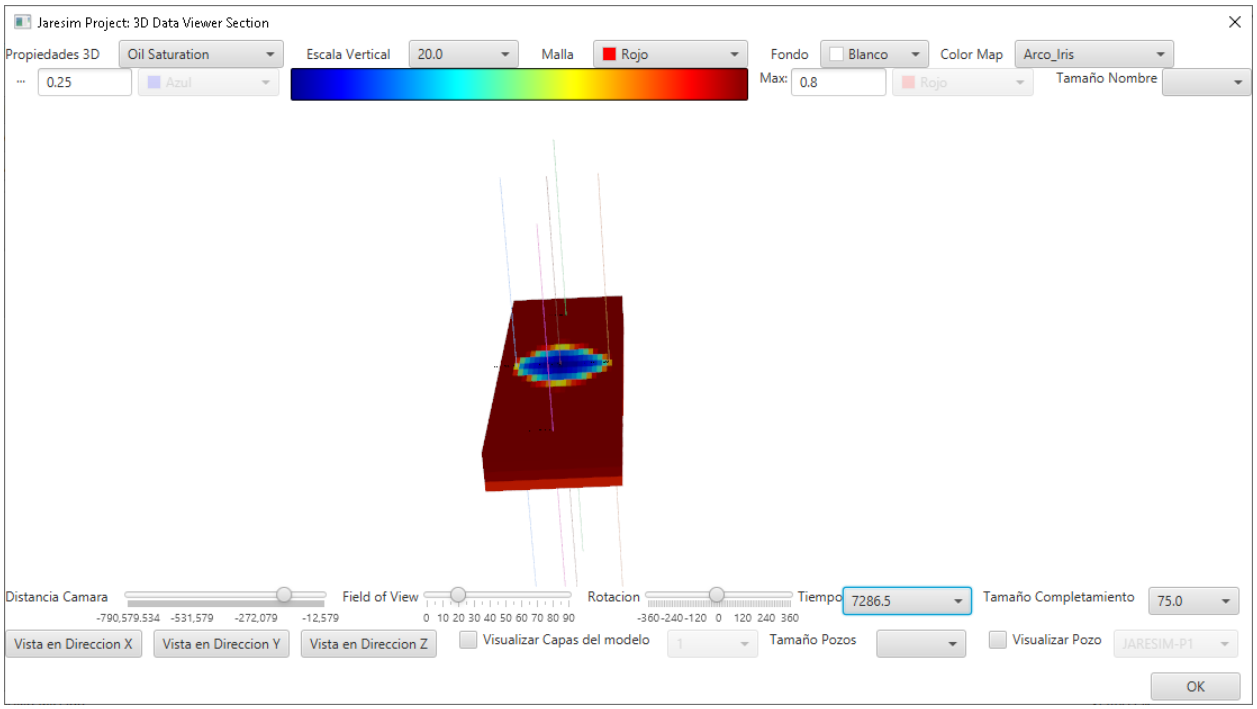


Ilustración 44:Sección Análisis de Resultados 3D: Malla de Saturación de Petróleo (A t=7286.5 días). Fuente: Elaboración propia.

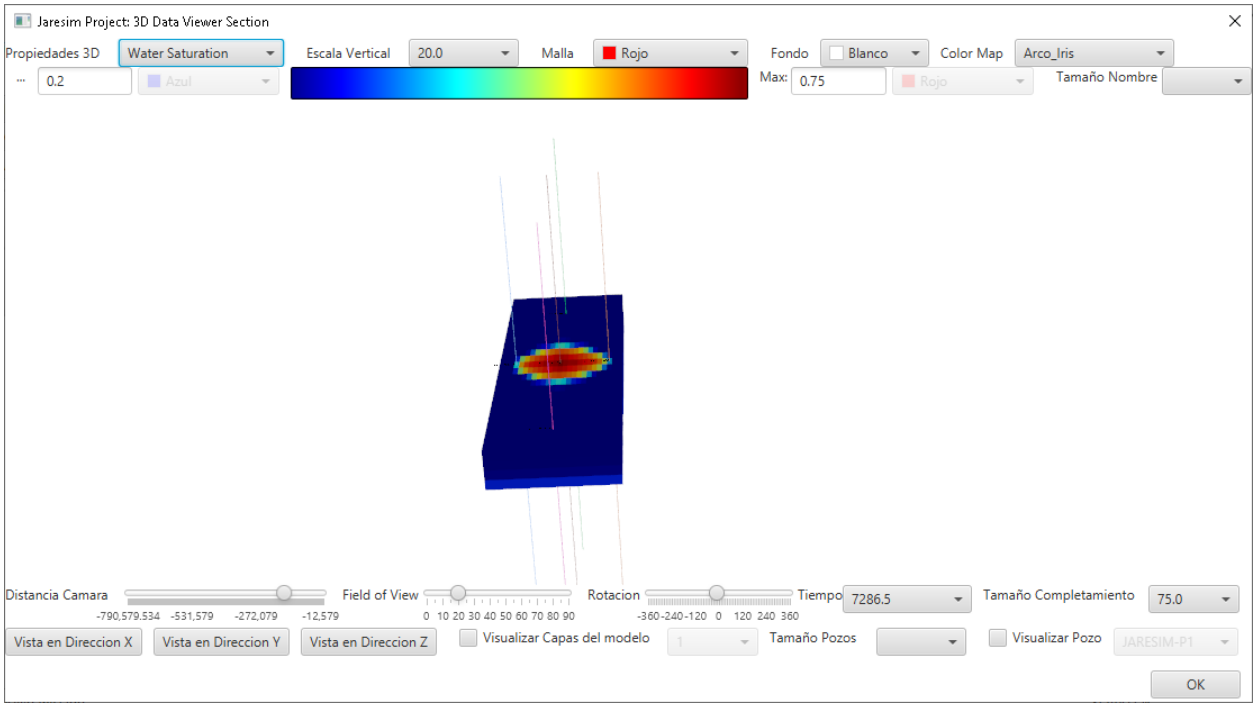


Ilustración 45:Sección Análisis de Resultados 3D: Malla de Saturación de Agua (A t=7286.5 días). Fuente: Elaboración propia.

Como puede verse, la aplicación tiene la capacidad de presentar gráficos típicos 2D y 3D, a partir de los resultados obtenidos de la simulación haciendo que esta sea autocontenida y no requiera de herramientas externas adicionales para el análisis de los resultados.

5. Metodología para el desarrollo del proyecto

5.1. Descripción de la Fases de Desarrollo del Proyecto y Descripción de los Roles de los Participantes en el Desarrollo del Proyecto

Una vez aprobado el proyecto, la primera fase correspondió a la realización de un estado del arte, por medio de la búsqueda bibliográfica como soporte de la investigación, realización de lectura de artículos y libros asociados al procesamiento paralelo GPU mediante las Librerías OpenCL ™.

La segunda fase del proyecto involucró el diseño y la codificación de dos módulos de procesamiento principales, el primero correspondió al módulo del simulador de yacimientos con el procesamiento tradicional CPU. El segundo módulo correspondió, al simulador de yacimientos usando el procesamiento GPU a través de la implementación de la librería JOCL, para implementar la plataforma OpenCL para el procesamiento paralelo de alto desempeño.

Para el desarrollo del Módulo de procesamiento CPU, se usó como referencia el proyecto llamado BOAST NFR (Almengor, y otros, June, 2006) como una versión de un simulador Black Oil, basado en el esquema IMPES[7] para yacimientos naturalmente fracturados. Tomando en cuentas las ecuaciones presentadas en la formulación IMPES en este proyecto, se procedió a realizar la codificación en JAVA para el desarrollo en una estructura similar, pero agregando el uso de 3 métodos de interpolación, el uso de 4 métodos para la resolución de sistemas lineales y el uso del modelo de Stone I y Stone II para flujo trifásico. Posteriormente, como codificación adicional, se procedió a implementar el procesamiento Multi Hilo, mediante las librerías JAVA de procesamiento paralelos generándose un módulo secundario CPU.

Una vez completada la codificación en JAVA del módulo CPU, se procedió con la codificación del Módulo Principal GPU. Para hacer esto, primero se realizó la serialización de esquema planteando en el procesamiento CPU, lo cual significa que el esquema original las variables del módulo CPU, estaba trabajando con matrices multidimensionales tipo $A[nx][ny][nz]$ y estas fueron convertidas a una variable 1D de tipo $B[nx * ny * nz]$ para poder realizar de

[7] IMPES: Implícito en Presión- Explicito en Saturación

forma más sencilla, la asignación de memoria de las Unidades de Procesamiento Gráfico (GPU). Una vez realizada la serialización del código CPU, se procedió a escribir los archivos con el código OpenCL en el Estándar C99 de los módulos del simulador. Además, se realizó la creación de las clases JAVA que permiten realizar la llamada de la librería OpenCL a través de JOCL y poder ejecutar el código del simulador serializado.

En la tercera fase, se realizó el diseño y la posterior codificación de las interfaces o ventanas de la aplicación mediante la librería JAVAFX, que permitió diseñar la interface gráfica de la aplicación desarrollada en este proyecto, a través de una serie de ventanas que permiten el acceso más sencillo de la información para la creación del caso de simulación. Además, dichas ventanas presentan una serie de utilidades adicionales, como el uso de correlaciones para la estimación de datos de compresibilidad de roca, correlaciones para estimación de la información de propiedades PVT de los fluidos (Presión-Volumen-Temperatura), y dos correlaciones para la estimación de los modelos de permeabilidades relativas. Aunado a esto, se diseñaron las ventanas correspondientes a la visualización de los resultados en 2D y 3D.

En la cuarta fase, fue realizada la depuración de todo el ejecutado en código, buscando posibles fallos. Posterior a eso, se procedió a realizar el despliegue de la aplicación realizando la generación del instalador para el sistema operativo Windows de la misma.

La quinta fase correspondió a la fase de validación, donde se realizó la generación de dos casos de validación para la verificación de los resultados, los cuales fueron comparados con los resultados obtenidos en un simulador comercial convencional CPU llamado IMEX de Computer Modellling Group (CMG) (Computer Modelling Group (CMG), 2014-2020).

Finalmente, en la última fase se procedido a la generación de documento soporte de la investigación.

A continuación, la Ilustración 46 muestra el diagrama de Gantt de las fases de desarrollo del proyecto, y la Tabla 1 correspondiente a los participantes en el desarrollo de cada una de las fases del proyecto donde se evidencia su rol participativo:

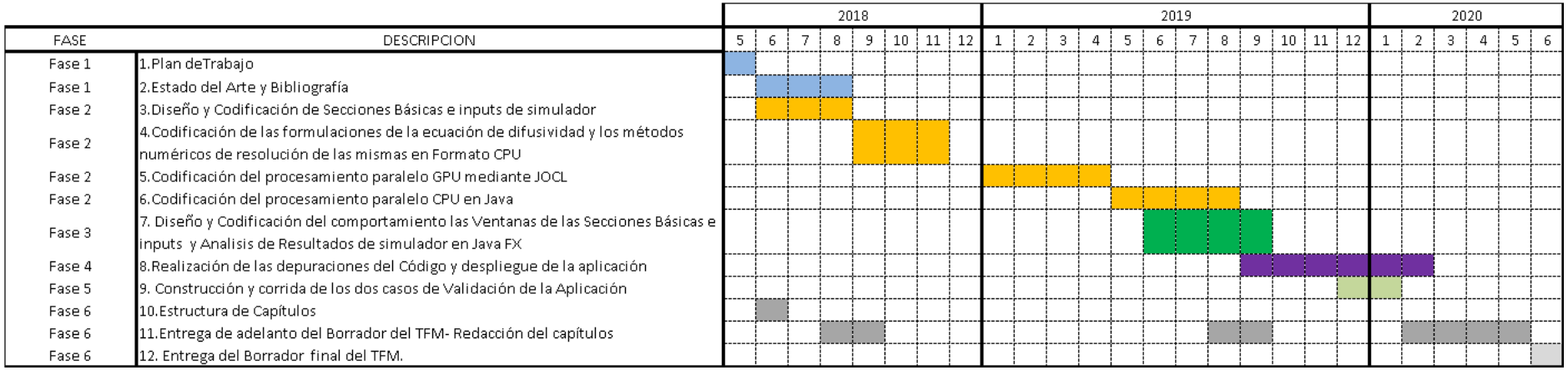


Ilustración 46:Diagrama de Gantt de las fases de desarrollo del proyecto. Fuente: Elaboración propia.

FASE	DESCRIPCION	PARTICIPANTE	ROL
Fase 1	1.Plan de Trabajo	JORGE L. RAMOS	COORDINADOR DE PROYECTO
Fase 1	2.Estado del Arte y Bibliografía	JORGE L. RAMOS	COORDINADOR DE PROYECTO
Fase 2	3.Diseño y Codificación de Secciones Básicas e inputs de simulador	JORGE L. RAMOS	PROGRAMADOR
Fase 2	4.Codificación de las formulaciones de la ecuación de difusividad y los métodos numéricos de resolución de las mismas en Formato CPU	JORGE L. RAMOS	PROGRAMADOR
Fase 2	5.Codificación del procesamiento paralelo GPU mediante JOCL	JORGE L. RAMOS	PROGRAMADOR
Fase 2	6.Codificación del procesamiento paralelo CPU en JAVA	JORGE L. RAMOS	PROGRAMADOR
Fase 3	7. Diseño y Codificación del comportamiento las Ventanas de las Secciones Básicas e inputs y Análisis de Resultados de simulador en JAVAFX	JORGE L. RAMOS	PROGRAMADOR
Fase 4	8.Realización de las depuraciones del Código y despliegue de la aplicación	JORGE L. RAMOS	DEPURADOR
Fase 5	9. Construcción y corrida de los dos casos de Validación de la Aplicación	JORGE L. RAMOS	INGENIERO EXPERTOS EN SIMULACIÓN DE YACIMIENTOS
Fase 6	10.Estructura de Capítulos	JORGE L. RAMOS	REDACTOR DE DOCUMENTO
Fase 6	11.Entrega de adelanto del Borrador del TFM-Redacción de los capítulos	JORGE L. RAMOS	REDACTOR DE DOCUMENTO
Fase 6	12. Entrega del Borrador final del TFM.	JORGE L. RAMOS	REDACTOR DE DOCUMENTO

Tabla 1:Participantes y sus roles de las fases de desarrollo del proyecto. Fuente: Elaboración propia.

5.2. Descripción de la herramienta a usadas para el desarrollo de la aplicación

Para el desarrollo de la aplicación propuesta se utilizaron las siguientes herramientas y paquetes que serán descritos brevemente a continuación:

5.2.1. IDE ECLIPSE

ECLIPSE es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto denomina "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para implementar entornos de desarrollo integrados (del inglés IDE), como el IDE de JAVA llamado JAVA Development Toolkit (JDT), y el compilador (ECJ) que se entrega como parte de ECLIPSE, y que son usados también para desarrollar el mismo ECLIPSE. (Eclipse Foundation, s.f.).

ECLIPSE fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para Visual Age. ECLIPSE es ahora desarrollado por la Fundación ECLIPSE, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. (Eclipse Foundation, s.f.).

Justificación de uso: IDE ECLIPSE

La selección de este paquete de desarrollo se debió en primera instancia a la facilidad de manejo de la misma, así como la vasta bibliografía disponible y las constantes actualizaciones por parte del desarrollador. Aunado a esto, se resalta el hecho de haber seleccionado el lenguaje JAVA como lenguaje de programación, permitiendo que la aplicación desarrollada pueda ser multiplataforma (aplicable a todos los sistemas operativos y a diversos dispositivos).

5.2.2. Oracle JAVAFX

JAVAFX es el conjunto de herramientas de interfaz gráfica de usuario (GUI) de última generación de JAVA que permite a los desarrolladores construir rápidamente aplicaciones amplias y multiplataformas. Construido desde cero, JAVAFX aprovecha modernas GPU a través de gráficos acelerados por hardware, mientras proporciona una programación bien diseñada en interfaces que permiten a los desarrolladores combinar gráficos, animación y controles de la interfaz de usuario. El JAVAFX 8 es una interfaz de programación de aplicaciones (API) en lenguaje JAVA puro. El objetivo de JAVAFX es ser utilizado a través de

muchos tipos de dispositivos, como los dispositivos integrados, teléfonos inteligentes, televisores, ordenadores Tablet, y escritorios. (Carl, Mark, Gerrit, Pereda, & Phillips, 2014).

Aplicaciones JAVAFX

Dado que la biblioteca de JAVAFX está escrita como una API de JAVA, el código de la aplicación de JAVAFX puede hacer referencia a APIs de cualquier biblioteca de JAVA. Por ejemplo, las aplicaciones de JAVAFX pueden utilizar las bibliotecas de API de JAVA para acceder a las capacidades nativas del sistema y conectarse a las aplicaciones de middleware basadas en el servidor. (Oracle, s.f.).

El aspecto de las aplicaciones de JAVAFX puede personalizarse. Las hojas de estilo en cascada (CSS) separan la apariencia y el estilo de la implementación para que los desarrolladores puedan concentrarse en la codificación. Los diseñadores gráficos pueden personalizar fácilmente la apariencia y el estilo de la aplicación a través de las CSS. Si tienes experiencia en diseño web, o si quiere separar la interfaz de usuario (UI) y la lógica del back-end, puedes desarrollar los aspectos de presentación de la UI en el lenguaje de scripts FXML y utilizar el código JAVA para la lógica de la aplicación. Si prefiere diseñar la interfaz de usuario sin escribir código, entonces use el Constructor de Escenas JAVAFX. A medida que diseña la interfaz de usuario, el Constructor de escenas crea un marcado FXML que se puede portar a un Entorno de Desarrollo Integrado (IDE) para que los desarrolladores puedan añadir la lógica de negocio. (Oracle, s.f.).

Justificación de uso: Oracle JAVAFX

La selección de esta librería se debió básicamente, a que esta permite generación de las interfaces gráficas, visualización 2D y 3D, así como la programación de comportamientos de las mismas de una forma rápida y sencilla. Además, este paquete es multiplataforma y puede ser usado para el desarrollo de aplicaciones de escritorio, web, dispositivos móviles y tabletas y puede ser desplegada en diversos sistemas operativos.

5.2.3. OpenCL

OpenCL™ (Lenguaje de Computación Abierto) es un estándar abierto y libre de regalías para la programación paralela y multiplataforma de diversos aceleradores que se encuentran en supercomputadoras, servidores en nube, computadoras personales, dispositivos móviles y plataformas incorporadas. OpenCL mejora enormemente la velocidad y la capacidad de respuesta de un amplio espectro de aplicaciones en numerosas categorías del mercado, entre ellas las herramientas creativas profesionales, el software científico y médico, el

procesamiento de la visión y la capacitación e inferencia de redes neuronales. (khronos.org, s.f.) .

Cómo se programan las aplicaciones OpenCL

OpenCL acelera las aplicaciones descargando su código más intensivo de computación en los procesadores del acelerador o dispositivos. Los desarrolladores de OpenCL utilizan lenguajes de núcleo basados en C o C++ para codificar programas que se pasan a través de un compilador de dispositivos para su ejecución en paralelo en los dispositivos de aceleración. El estándar OpenCL proporciona llamadas a la API para descubrir qué dispositivos de aceleración están disponibles en un sistema, compilar y cargar programas del núcleo a esos procesadores y coordinar su ejecución paralela desde una CPU anfitriona. (khronos.org, s.f.) .(ver Ilustración 47).

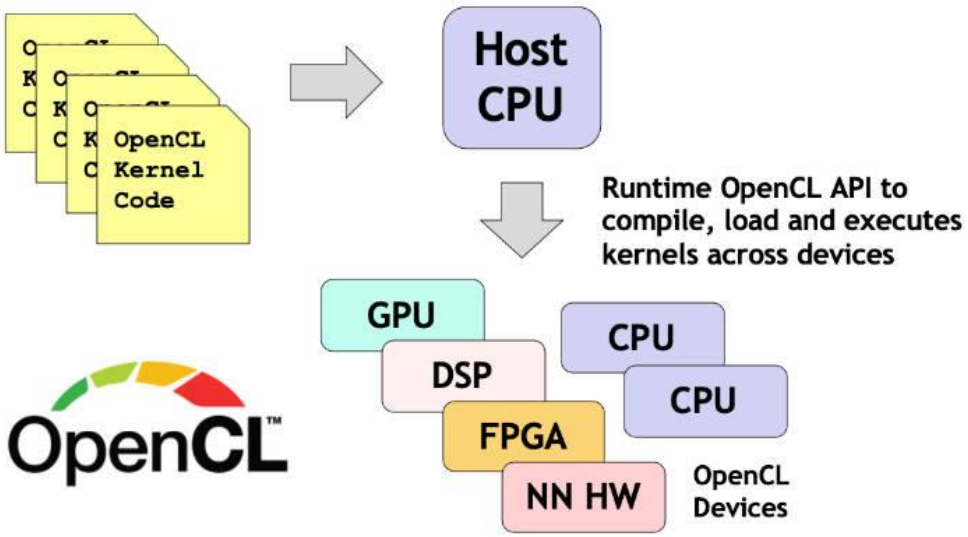


Ilustración 47:Cómo se programan las aplicaciones OpenCL. Fuente: (khronos.org, s.f.)

Cómo se relaciona OpenCL con la familia de estándares de aceleración de Khronos

OpenCL proporciona a la industria la capa de ejecución de procesador más baja "cercana al metal" para acelerar aplicaciones, bibliotecas y motores, y también proporciona un objetivo de generación de código para los compiladores. A diferencia de las API "sólo para la GPU", como Vulkan, OpenCL permite el uso de una amplia gama de aceleradores, incluyendo CPU multinúcleo, GPU, DSP, FPGAs y hardware dedicado, como los motores de inferencia. (khronos.org, s.f.).(ver Ilustración 48).

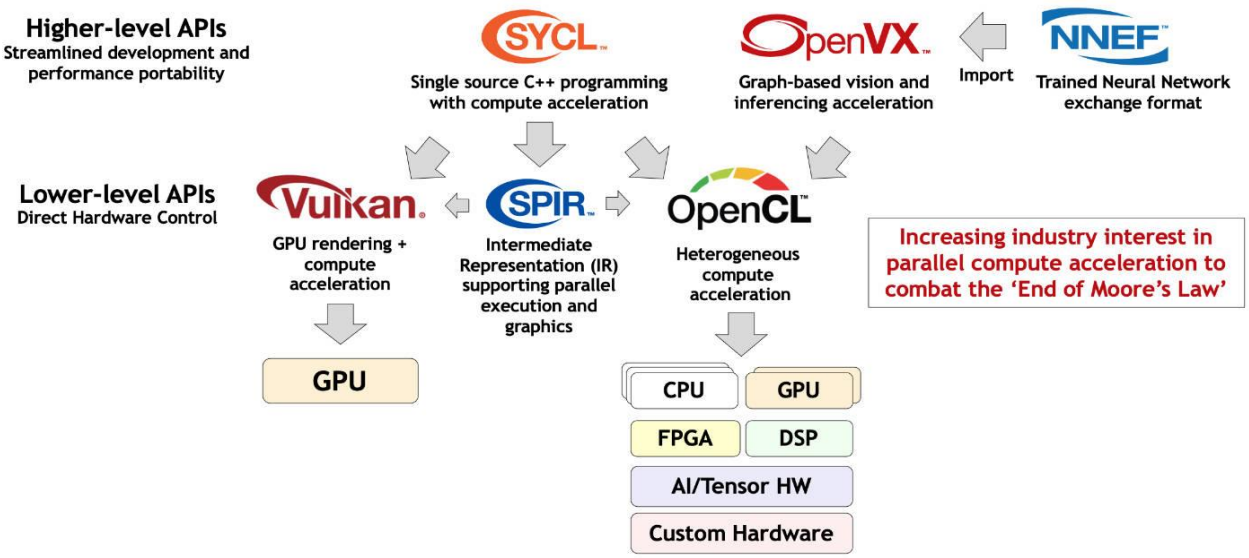


Ilustración 48:Cómo se relaciona OpenCL con la familia de estándares de aceleración de Khronos. Fuente: (khronos.org, s.f.)

Flexibilidad de despliegue

A medida que el panorama de la industria de plataformas y dispositivos se hace más complejo, las herramientas están evolucionando para permitir que las aplicaciones OpenCL se desplieguen en plataformas que no tienen disponibles controladores nativos de OpenCL. Por ejemplo, el compilador clspv de código abierto y el traductor de la API de clvk permiten que las aplicaciones OpenCL se ejecuten en un tiempo de ejecución de Vulkan. Esto proporciona a los desarrolladores de OpenCL una gran flexibilidad en cuanto a dónde y cómo pueden desplegar sus aplicaciones OpenCL. (khronos.org, s.f.).(ver Ilustración 49).

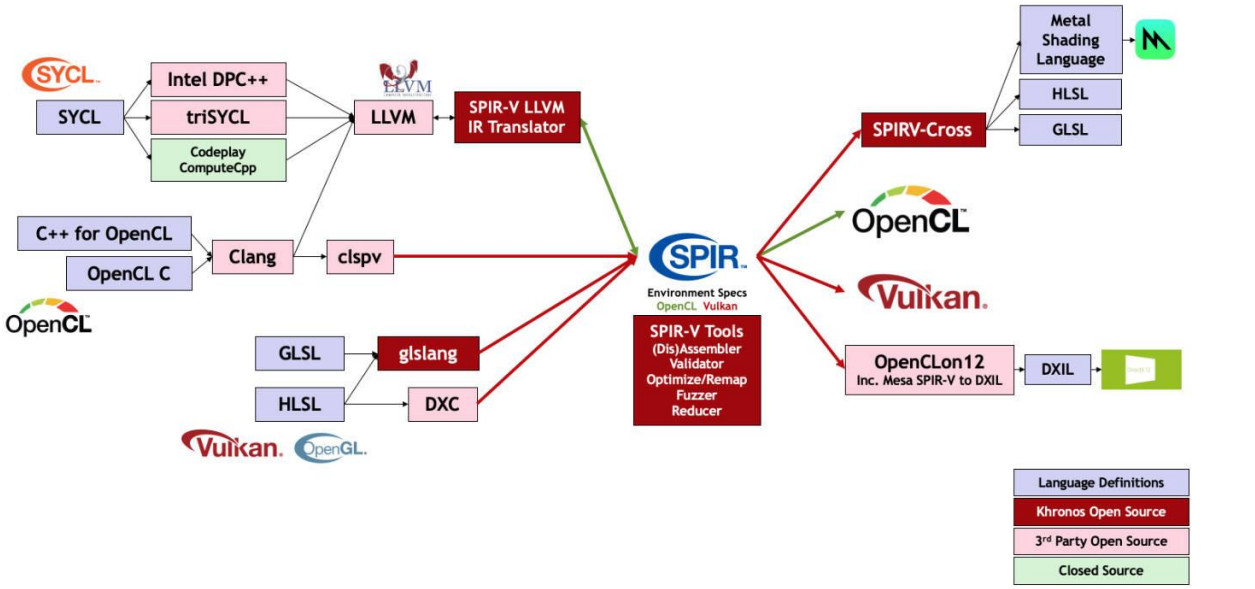


Ilustración 49:Herramientas de software de código abierto permiten que los núcleos de OpenCL se ejecuten sobre múltiples APIs de destino. Fuente: (khronos.org, s.f.)

Modelo de programación de OpenCL

Una aplicación OpenCL se divide en partes, la correspondiente al host y la correspondiente al dispositivo con el código del host, el cual es escrito con un lenguaje de programación general como C o C++ y compilado por un compilador convencional para su ejecución en una CPU del host. (khronos.org, s.f.).

La fase de compilación del dispositivo puede realizarse en línea, es decir, durante la ejecución de una aplicación mediante llamadas especiales a la API. También puede compilarse antes de ejecutar la aplicación en la representación binaria de la máquina o en la representación intermedia portátil especial definida por Khronos llamada SPIR-V. También hay lenguajes y marcos específicos de dominio que pueden compilar a OpenCL ya sea utilizando traducciones de fuente a fuente o generando binarios/SPIR-V, por ejemplo, Halide. (khronos.org, s.f.) ,como puede ser visto en la Ilustración 50.

El código del host de la aplicación suele estar escrito en C o C++, pero también hay disponibles enlaces para otros lenguajes, como Python o JAVA. Los programas del kernel pueden ser escritos en un dialecto de C (OpenCL C) o C++ (C++ para OpenCL), que permite a un desarrollador programar partes intensivas en computación de su aplicación en un programa del kernel. Todas las versiones del lenguaje OpenCL C están basadas en C99. El lenguaje C++ para OpenCL, impulsado por la comunidad, reúne las capacidades de OpenCL y C++17. (khronos.org, s.f.).

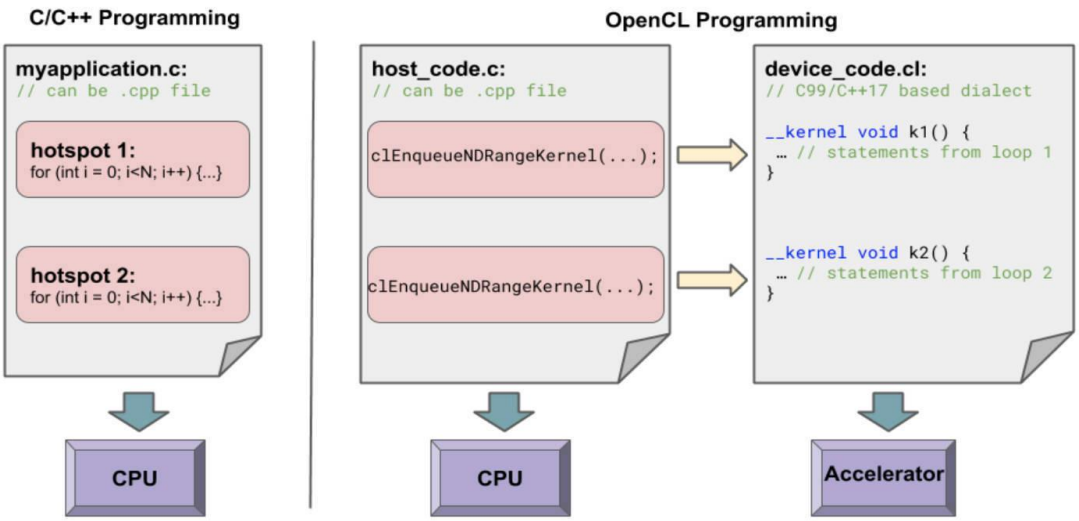


Ilustración 50:Paradigma de programación tradicional vs. OpenCL. Fuente: (khronos.org, s.f.)

C++ para el lenguaje del núcleo de OpenCL

El grupo de trabajo de OpenCL ha pasado del lenguaje original del kernel OpenCL C++ definido por primera vez en OpenCL 2.0 a C++ para OpenCL desarrollado por la comunidad

de código abierto para proporcionar mejores características y compatibilidad con OpenCL C. C++ para OpenCL está soportado por Clang, y permite a los desarrolladores utilizar la mayoría de las características de C++17 en los núcleos de OpenCL. Es en gran parte compatible con OpenCL C 2.0, lo que permite utilizarlo para programar aceleradores con OpenCL 2.0 o superior con controladores conformes que soporten SPIR-V. Su implementación en Clang se puede seguir a través de la página de soporte de OpenCL. (khronos.org, s.f.) .(ver Ilustración 51).

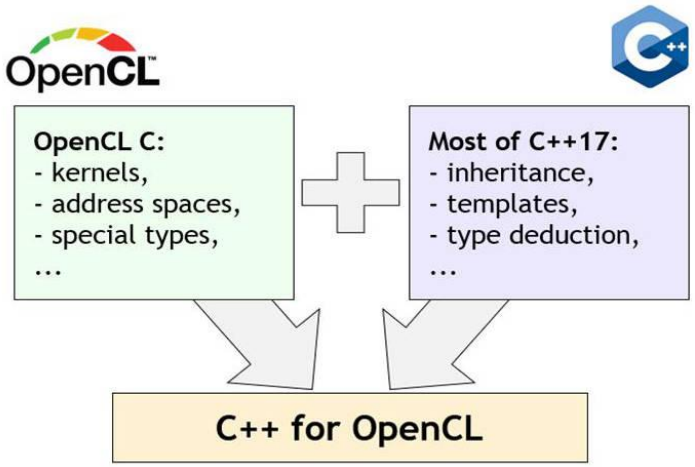


Ilustración 51:C++ para el lenguaje del núcleo de OpenCL. Fuente: (khronos.org, s.f.)

Extensiones del lenguaje del núcleo

Se dispone de algunas extensiones para las normas de lenguaje del núcleo ya publicadas. La lista completa de tales extensiones está documentada aquí. Los compiladores y controladores conformes pueden opcionalmente soportar las extensiones, y, por lo tanto, hay un mecanismo para detectar su soporte en el momento de la compilación. Los desarrolladores deben ser conscientes de que no todas las extensiones pueden ser soportadas en todos los dispositivos. (khronos.org, s.f.).

JOCL

JOCL (JAVA bindings for OpenCL) es una biblioteca que proporciona enlaces JAVA para OpenCL. La API de JOCL es muy similar a la API original de OpenCL. Así que muchos de los recursos online disponibles para OpenCL pueden aplicarse directamente a los programas de JOCL. La mayoría de las implementaciones de OpenCL vienen con un conjunto de muestras, que pueden servir como puntos de partida para los propios programas. Algunas de estas muestras también se han portado a JOCL, y pueden obtenerse en la sección de muestras. (jocl.org, s.f.). La mayor parte de la documentación de la API de JAVADoc de JOCL fue extraída de las páginas de referencia de OpenCL del sitio web de OpenCL de Khronos. (jocl.org, s.f.).

Configuración básica del JOCL

Para la configuración básica del JOCL, se deben seguir los siguientes pasos (jocl.org, s.f.):

- Instalar una implementación de OpenCL. Esta puede ser una de las implementaciones resumidas en la página principal de JOCL.
- Crear un proyecto en tu IDE favorito.
- Añadir los archivos JOCL al proyecto. Esto normalmente significa copiar los archivos apropiados de la sección de descargas en el directorio raíz de tu proyecto.
- Añadir el archivo JAR al classpath.
- Ejecutar una de las muestras.

Justificación de uso: OpenCL y JOCL

La selección de esta librería principalmente se debe a que permite la generación de implementaciones de procesamiento heterogéneo CPU-GPU, haciéndola una herramienta poderosa en la generación de aplicaciones de computación de alto desempeño, la cual es la librería Core para el desarrollo de este proyecto y que permite el procesamiento y computo en las unidades de procesamiento gráfico (GPU). Además, este paquete es multiplataforma y puede ser usado para el desarrollo de aplicaciones de escritorio, web, dispositivos móviles y tabletas y puede ser desplegada en diversos sistemas operativos.

6. Descripción de las clases desarrolladas en la aplicación y del desarrollo de módulo para procesamiento GPU propuestos en el proyecto

6.1. Descripción de las Clases JAVA desarrolladas para las diferentes secciones del Simulador desarrollado en la investigación

La aplicación desarrollada está constituida por cuatro principales paquetes que contiene las clases JAVA, desarrolladas en la aplicación como se presenta a continuación:

1. Paquete **aplication_jaresim:** Este paquete contiene la codificación de las interfaces diseñadas mediante el uso de JAVA FX, así como las clases que controlan el comportamiento de dichas interfaces en las cuales está divida la aplicación.
2. Paquete **CPU_MR_Module_Codes:** Este paquete contiene la codificación de la clase correspondiente al módulo de procesamiento CPU, mediante lenguaje JAVA y los módulos en las cuales está divida dicha clase.
3. Paquete **Parallel_Module_Codes:** Este paquete contiene la codificación de la clase correspondiente al módulo de procesamiento CPU paralelo, mediante lenguaje JAVA mediante el uso de ExecutorService y los módulos en las cuales está divida dicha clase.
4. Paquete **GPU_MR_Module_Codes:** Este paquete contiene la codificación de la clase correspondiente al módulo de procesamiento GPU, mediante lenguaje JAVA por medio de las librerías JOCL para la implementación del paquete OpenCL y los módulos en las cuales está dividida dicha clase.

La Ilustración 52 muestra como están distribuidos estos paquetes dentro del proyecto JAVA desarrollado:

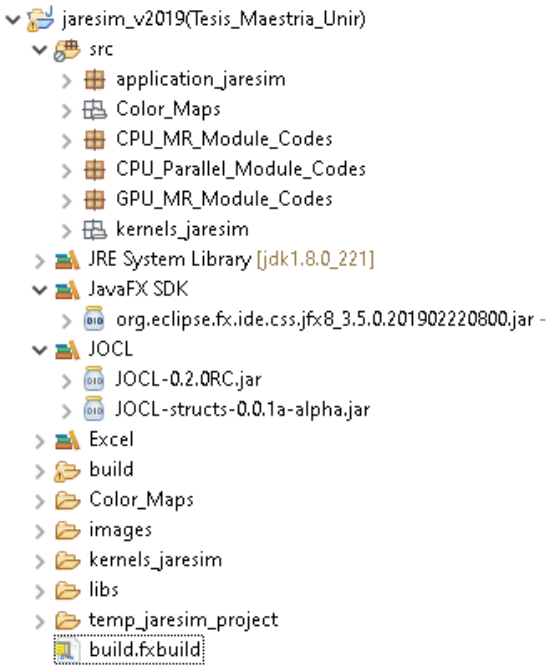


Ilustración 52:Paquetes JAVA desarrollados dentro de la aplicación. Fuente: Elaboración propia.

6.2. Descripción de la Formulación IMPES usada en el Simulador

El planteamiento de la solución de la ecuación de Difusividad para una malla ortogonal viene dado por la siguiente ecuación:

Para la Fase Petróleo:

$$\frac{\partial(\frac{\rho_o * k_{ox}}{\mu_o} * \frac{\partial P_o}{\partial x})}{\partial x} + \frac{\partial(\frac{\rho_o * k_{oy}}{\mu_o} * \frac{\partial P_o}{\partial y})}{\partial y} + \frac{\partial(\frac{\rho_o * k_{oz}}{\mu_o} * \frac{\partial(P_o \pm \rho_o * \Delta z * g)}{\partial z})}{\partial z} = \frac{\partial(\rho_o * \emptyset * S_o)}{\partial t}$$

Para la Fase Agua:

$$\frac{\partial(\frac{\rho_w * k_{wx}}{\mu_w} * \frac{\partial P_w}{\partial x})}{\partial x} + \frac{\partial(\frac{\rho_w * k_{wy}}{\mu_o} * \frac{\partial P_w}{\partial y})}{\partial y} + \frac{\partial(\frac{\rho_w * k_{wz}}{\mu_w} * \frac{\partial(P_w \pm \rho_w * \Delta z * g)}{\partial z})}{\partial z} = \frac{\partial(\rho_w * \emptyset * S_w)}{\partial t}$$

Para la Fase Gas:

$$\frac{\partial(\frac{\rho_g * k_{gx}}{\mu_g} * \frac{\partial P_g}{\partial x})}{\partial x} + \frac{\partial(\frac{\rho_g * k_{gy}}{\mu_g} * \frac{\partial P_g}{\partial y})}{\partial y} + \frac{\partial(\frac{\rho_g * k_{gz}}{\mu_g} * \frac{\partial(P_g \pm \rho_g * \Delta z * g)}{\partial z})}{\partial z} = \frac{\partial(\rho_g * \emptyset * S_g)}{\partial t}$$

Ecuación 13:Ecuación de Difusividad para tres fases en un modelo de fluido Black Oil.

Para el planteamiento de sistema Implícito en Presión y Explicito en Saturación (IMPES), la ecuación de difusividad debe ser reescrita en términos de transmisibilidades las cuales se presenta a continuación:

Sistema Explicito de Saturaciones:

$$\Delta T * \left[M_o^n * \Delta(P_o)_{i.j}^{n+1}\right] + (q_o)_{i,j}^n = \left(\frac{Vr_{i,j} * \emptyset}{B_o * \Delta t}\right) * (S_o{}_{i,j}^{n+1} - S_o{}_{i,j}^{n})$$

$$\Delta T * \left[M_w^n * \Delta(P_w)_{i.j}^{n+1}\right] + (q_w)_{i,j}^n = \left(\frac{Vr_{i,j} * \emptyset}{B_w * \Delta t}\right) * (S_w{}_{i,j}^{n+1} - S_w{}_{i,j}^{n}$$

$$\Delta T * \left[M_g^n * \Delta\left(P_g\right)_{i.j}^{n+1}\right] + \left(q_g\right)_{i,j}^n = \left(\frac{Vr_{i,j} * \emptyset}{B_g * \Delta t}\right) * (S_g{}_{i,j}^{n+1} - S_g{}_{i,j}^{n})$$

Ecuación 14:Sistema IMPES para tres fases en un modelo de fluido Black Oil (Saturaciones). Fuente: (SPE Series Reservoir SImulation, 1990).

Sistema Implícito de Presiones:

$$B_o * \Delta T * \left[M_o^n * \Delta(P_o)_{i.j}^{n+1}\right] + B_w * \Delta T * \left[M_w^n * \Delta(P_w)_{i.j}^{n+1}\right] + B_g * \Delta T * \left[M_g^n * \Delta\left(P_g\right)_{i.j}^{n+1}\right]$$
$$+ (B_o * q_o)_{i,j}^n + (B_w * q_w)_{i,j}^n + \left(B_g * q_g\right)_{i,j}^n = 0$$

Ecuación 15:Sistema IMPES para tres fases en un modelo de fluido Black Oil (Presiones). Fuente: (SPE Series Reservoir SImulation, 1990).

En general, para el diseño y la generación de un simulador se sigue el siguiente flujo de trabajo presentado en la Ilustración 53, según (Islam, Moussavizadegan, Mustafiz, & Abou-Kassem, 2010):

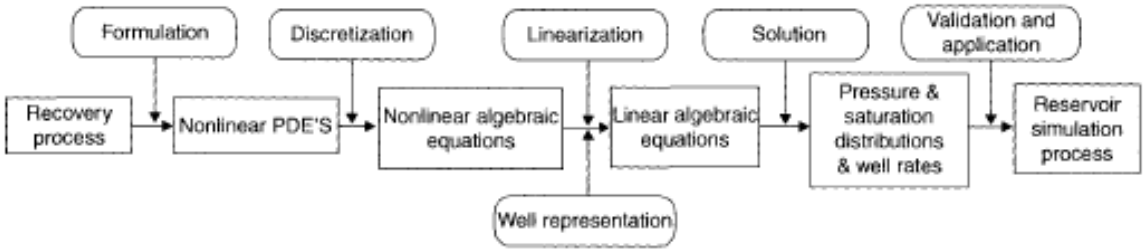


Ilustración 53:Mayores Pasos para desarrollar simuladores de Yacimientos. Fuente: (Islam, Moussavizadegan, Mustafiz, & Abou-Kassem, 2010).

6.3. Descripción de los diagramas de flujo de las secciones del simulador desarrollado en la investigación

La ilustración de los diagramas de flujo que muestran los códigos desarrollados para el módulo GPU de la aplicación desarrollada, se encuentran ubicados en el Anexo 10.6 de este documento debido a su extensión.

6.4. Descripción de la configuración JOCL para procedimiento CPU-GPU del simulador desarrollado en esta investigación

Para la implementación del procesamiento GPU se procedió a activar la librería JOCL, que permite el uso de las librerías OpenCL para procesamiento paralelo CPU-GPU. Los códigos JOCL usados para realizar esto, se muestran en el Anexo 10.7 de este documento debido a su extensión.

7. Casos de validación de la aplicación.

En el presente capitulo, se describirán 2 casos generados para realizar la validación de la aplicación desarrollada en esta investigación, los cuales se presentan a continuación:

7.1. Caso de validación #1

Se describirá la información asociada al caso de validación #1 correspondiente a una malla rectangular con propiedades heterogéneas por capa que incluye 5 pozos dentro del modelo para un periodo de predicción de 25 años. Posteriormente, se podrán observar los resultados obtenidos de esta evaluación realizando una comparación de los resultados del simulador desarrollado en este proyecto y el simulador Comercial IMEX de la Compañía CMG (Computer Modelling Group (CMG), 2014-2020):

7.1.1. Datos de Entrada

La información usada para construir el caso de validación #1 de la aplicación desarrollada en este proyecto, se encuentran ubicados en el Anexo 10.1 de este documento debido a su extensión.

7.1.2. Resultados Obtenidos

Los resultados obtenidos para el simulador propuesto y el simulador comercial usado como referencia pueden ser vistos en detalle en el Anexo 10.3 presente en este proyecto. Aunado a esto, se muestra a continuación las ilustraciones 54 a la 58 de los gráficos comparativos de los resultados obtenidos del simulador propuesto en esta investigación (Jaresim), y el Simulador comercial IMEX de la Compañía CMG con la finalidad de verificar la calidad de pronóstico del simulador desarrollado en este proyecto:

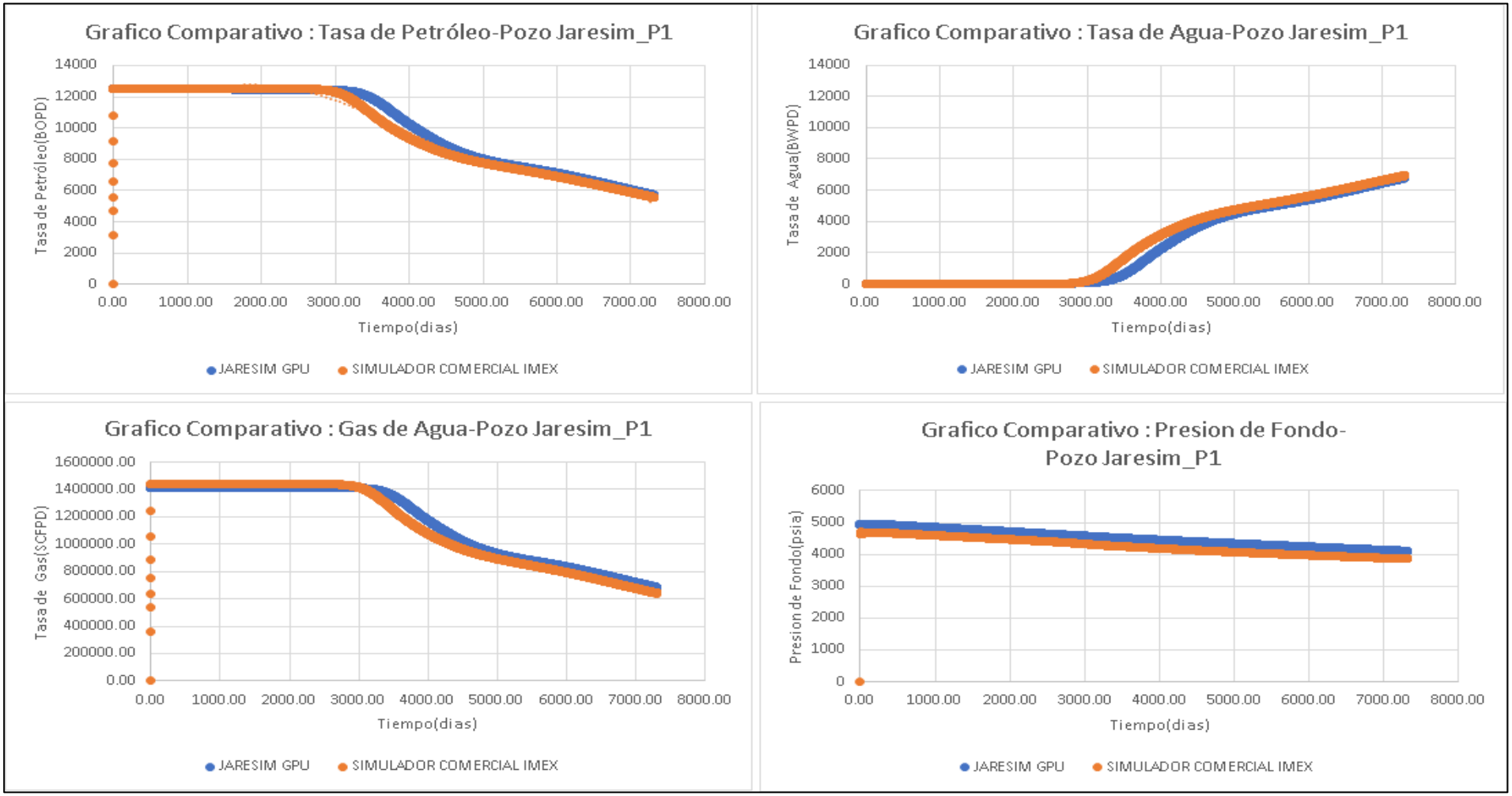


Ilustración 54:Gráficos comparativos a nivel para el Pozo P1 de los resultados obtenidos en el caso de Validación para: Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

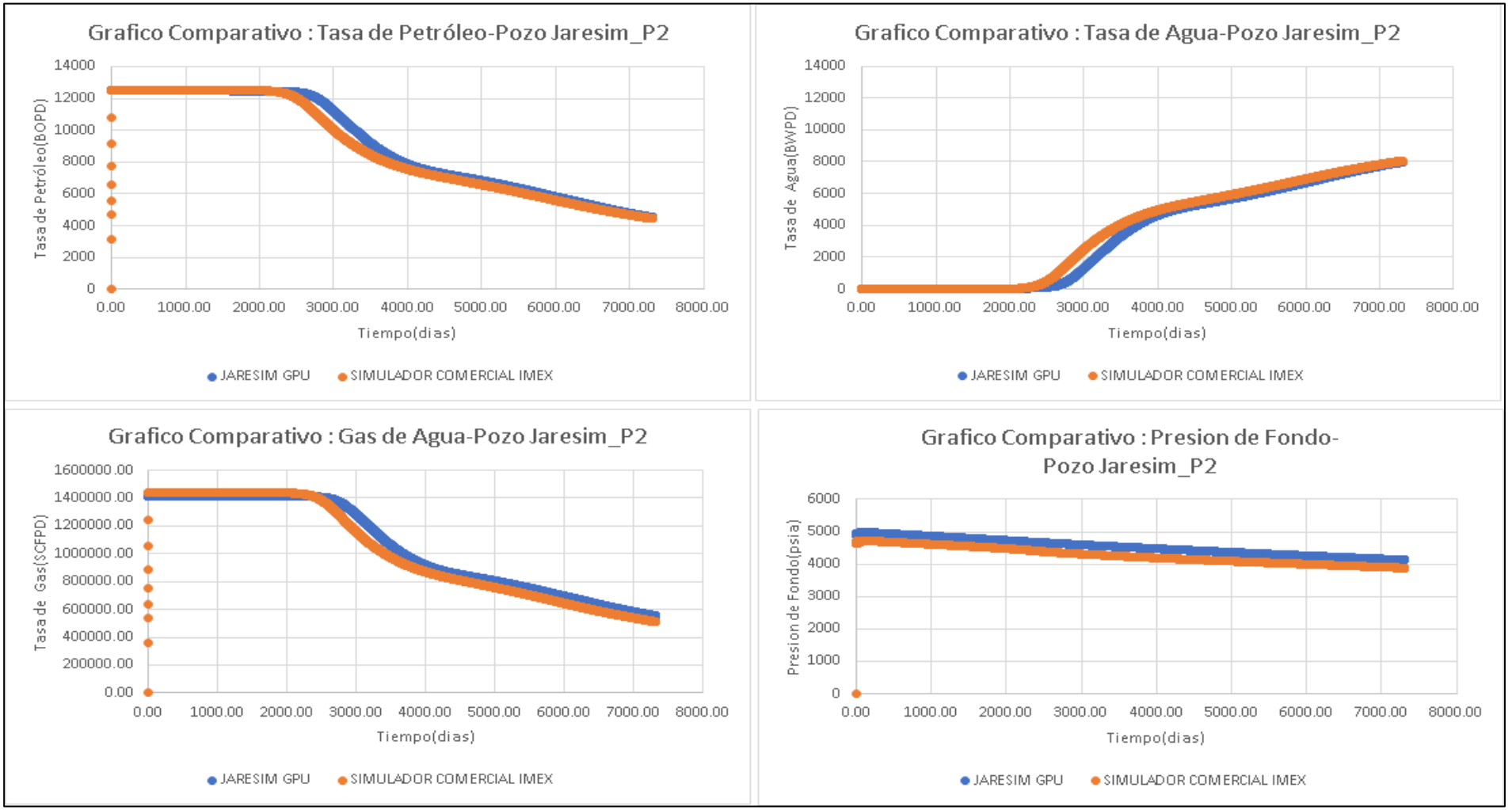


Ilustración 55:Gráficos comparativos a nivel para el Pozo P2 de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

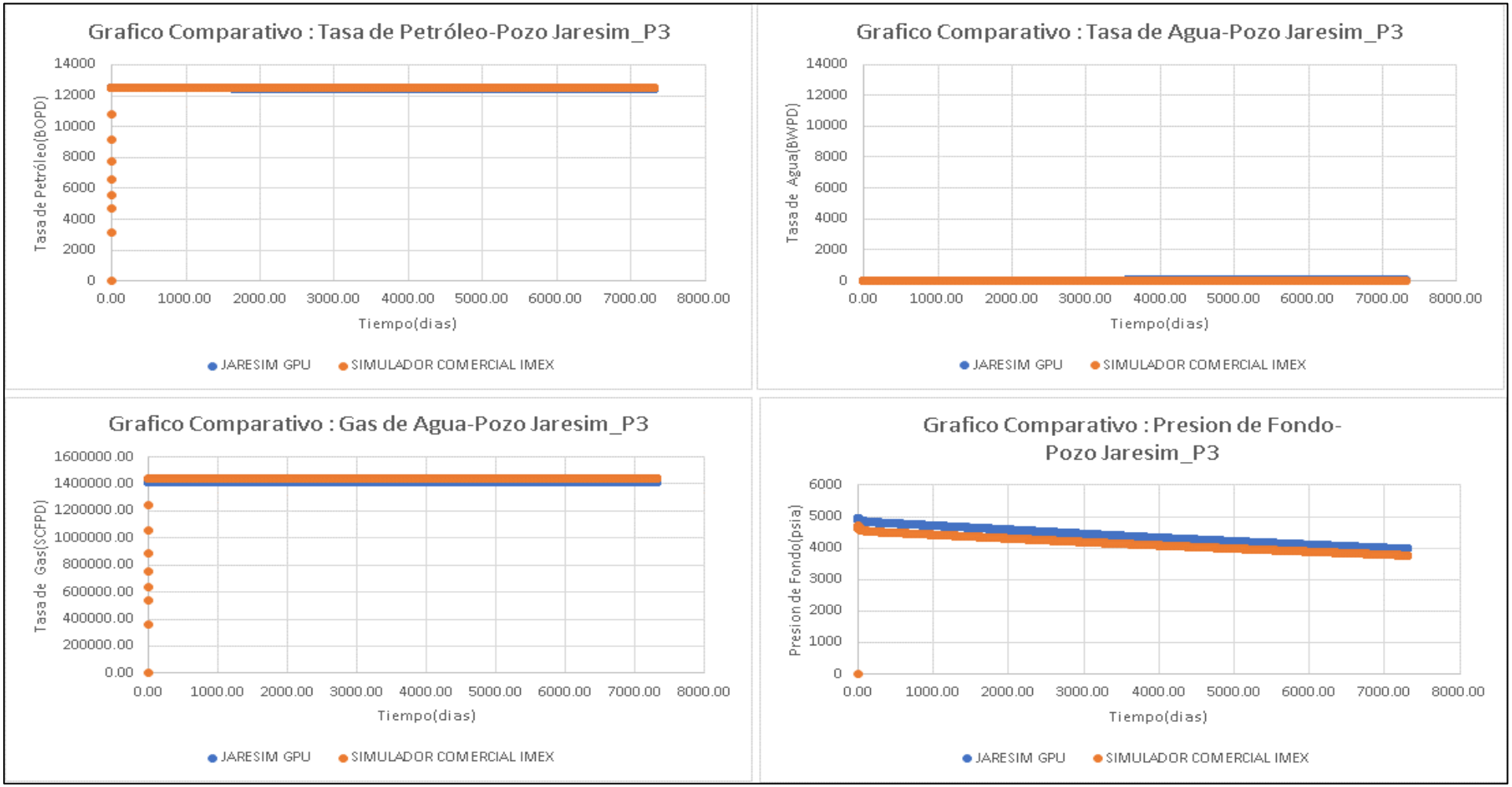


Ilustración 56:Gráficos comparativos a nivel para el Pozo P4 de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

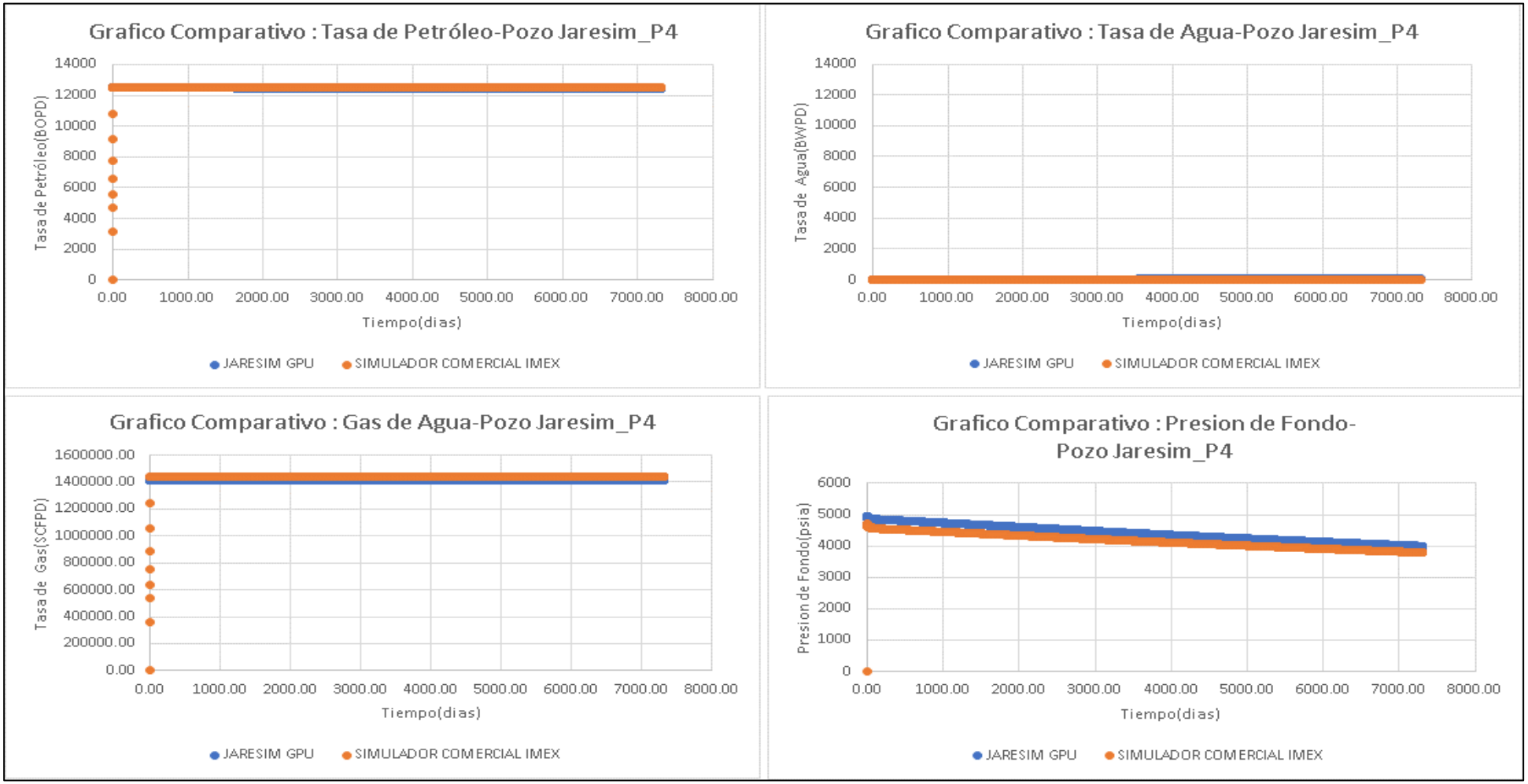


Ilustración 57:Gráficos comparativos a nivel para el Pozo P4 de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

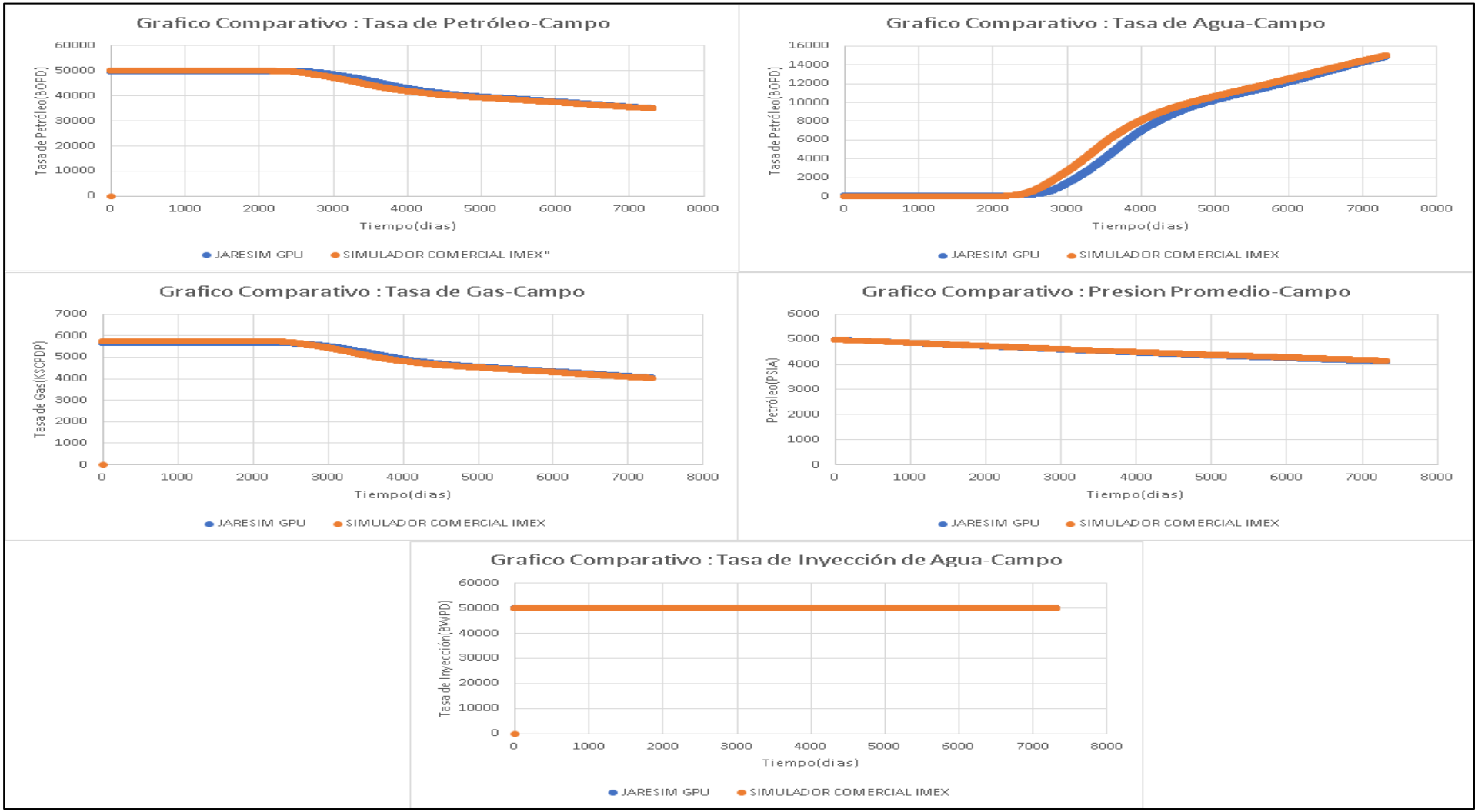


Ilustración 58:Gráficos comparativos a nivel de campo de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

Las ilustraciones presentadas evidencian que el simulador propuesto en este proyecto (puntos azules de las ilustraciones) tiene la capacidad de reproducir con precisión la tendencia de los resultados obtenidos en el programa comercial (puntos naranjas de las ilustraciones), al comparar sus resultados de las tasas de petróleo (gráfico superior-izquierdo), agua(gráfico superior-derecho), gas (gráfico inferior-izquierdo), y presión (gráfico inferior-derecho), a nivel de los pozos P1, P2, P3, P4 del modelo, así como los resultados a nivel de campo.

A continuación, se muestra la Tabla 2 comparativa de los acumulados obtenidos por el simulador propuesto por esta investigación, respecto al simulador comercial de referencia:

TABLA RESUMEN VALORES ACUMULADOR Y PRESION AL FINAL DE LA CORRIDA DEL CASO 1

CASO	SIMULADOR	FECHA	Np(BO)	Wp(BW)	Gp(KSCF)	Wi(BW)	Presion(psia)
CASO 1	JARESIM GPU	1/1/2034	322829981.2	42395018.8	36819496.7	365225000.0	4157.4
CASO 1	IMEX-CMG	1/1/2034	319784736.0	45465272.0	36774780.9	365250016.0	4157.3

Tabla 2:Tabla resumen de los valores acumulados a nivel de Campo para caso de Validación #1: para el Simulador desarrollado en la Investigación (JARESIM GPU) y Simulador Comercial (IMEX-CMG). Fuente: Elaboración propia.

Adicionalmente, se presenta la Tabla 3 comparativa del % de las diferencias de los acumulados obtenidos por el simulador propuesto por esta investigación, respecto al simulador comercial de referencia:

TABLA RESUMEN DIFERENCIAS ASOCIADO AL FINAL DE LA CORRIDA DEL CASO 1 PARA EL SIMULADOR JARESIM

CASO	SIMULADOR	FECHA	Np Diferencia (%)	Wp Diferencia (%)	Gp Diferencia (%)	Wi Diferencia (%)	Presion Diferencia (%)
CASO 1	JARESIM GPU	1/1/2034	0.952%	-6.753%	0.122%	-0.007%	0.002%

Tabla 3:Tabla resumen del porcentaje de desviación de los valores acumulados a nivel de Campo para caso de Validación #1: para el Simulador desarrollado en la Investigación (JARESIM GPU) respecto al simulador Comercial (IMEX-CMG). Fuente: Elaboración propia.

Con base a estos resultados, es posible concluir que para este caso el simulador propuesto en este proyecto usando procesamiento GPU fue capaz de reproducir los resultados obtenidos por un simulador comercial siguiendo a tendencia de los resultados obtenidos, mostrando desviaciones menores a 6.8% a nivel de los acumulados y 0.002% a nivel de la presión.

7.2. Caso de validación #2

Se describirá la información asociada al caso de validación #2 correspondiente a una malla importada en formato ECLGRD[8] con propiedades totalmente heterogéneas, que incluye 5 pozos dentro del modelo para un periodo de predicción de 10 años. Posteriormente, se podrán observar los resultados obtenidos de esta evaluación realizando una comparación de los resultados del simulador desarrollado en este proyecto, y el simulador Comercial IMEX de la Compañía CMG (Computer Modelling Group (CMG), 2014-2020):

7.2.1. Datos de Entrada

La información usada para construir el caso de validación #2 de la aplicación desarrollada en este proyecto, se encuentran ubicados en el Anexo 10.2 de este documento debido a su extensión.

7.2.2. Resultados Obtenidos

Los resultados obtenidos para el simulador propuesto y el simulador comercial usado como referencia, pueden ser vistos en detalle en el Anexo 10.4 presente en este proyecto. Aunado a esto, se muestra a continuación las ilustraciones de la 59 a la 63 de los gráficos comparativos de los resultados obtenidos del simulador propuesto en esta investigación (Jaresim), y el Simulador comercial IMEX de la Compañía CMG con la finalidad de verificar la calidad de pronóstico del simulador desarrollado:

[8] ECLGRD: ECLIPSE GRID Format

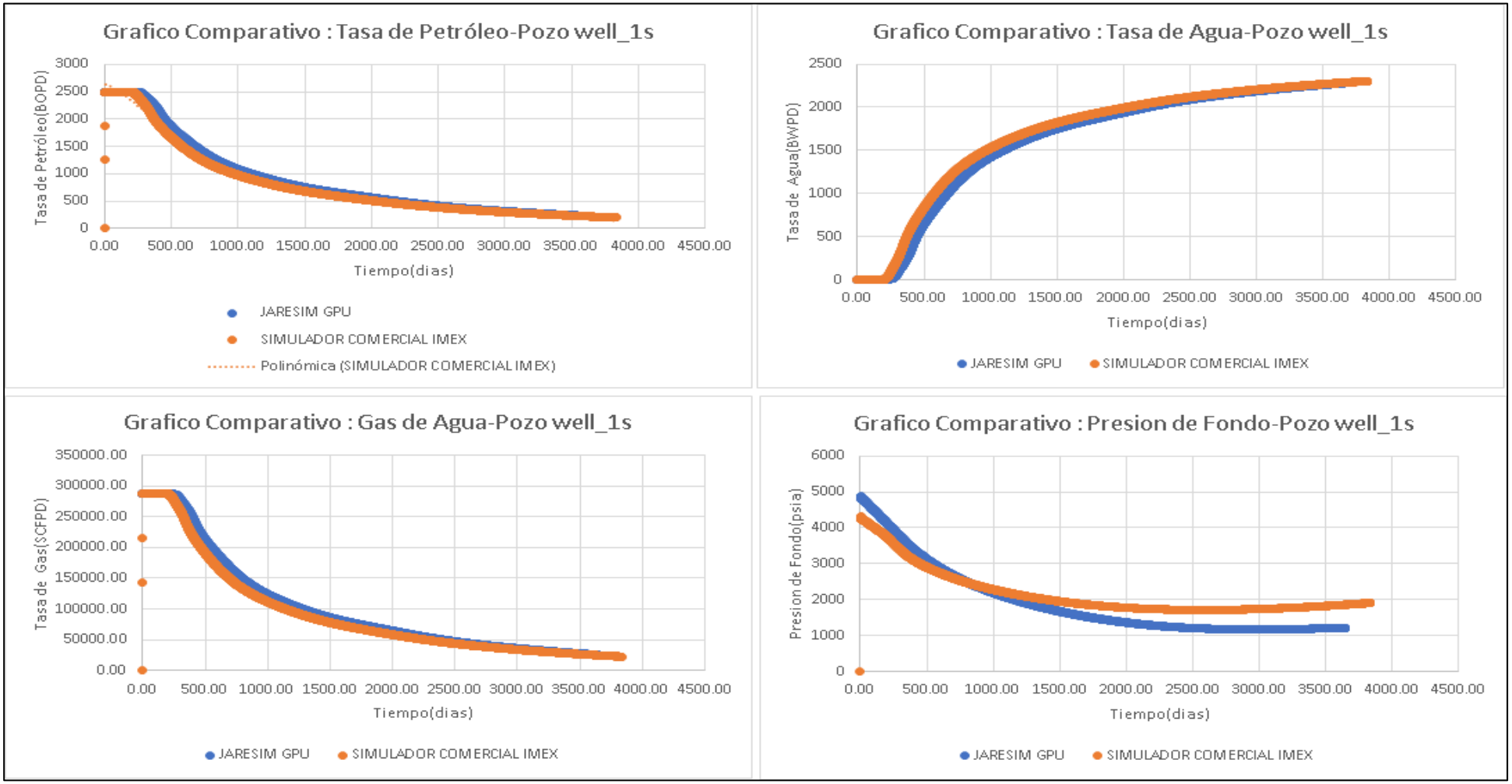


Ilustración 59:Gráficos comparativos a nivel para el Pozo well_1s de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

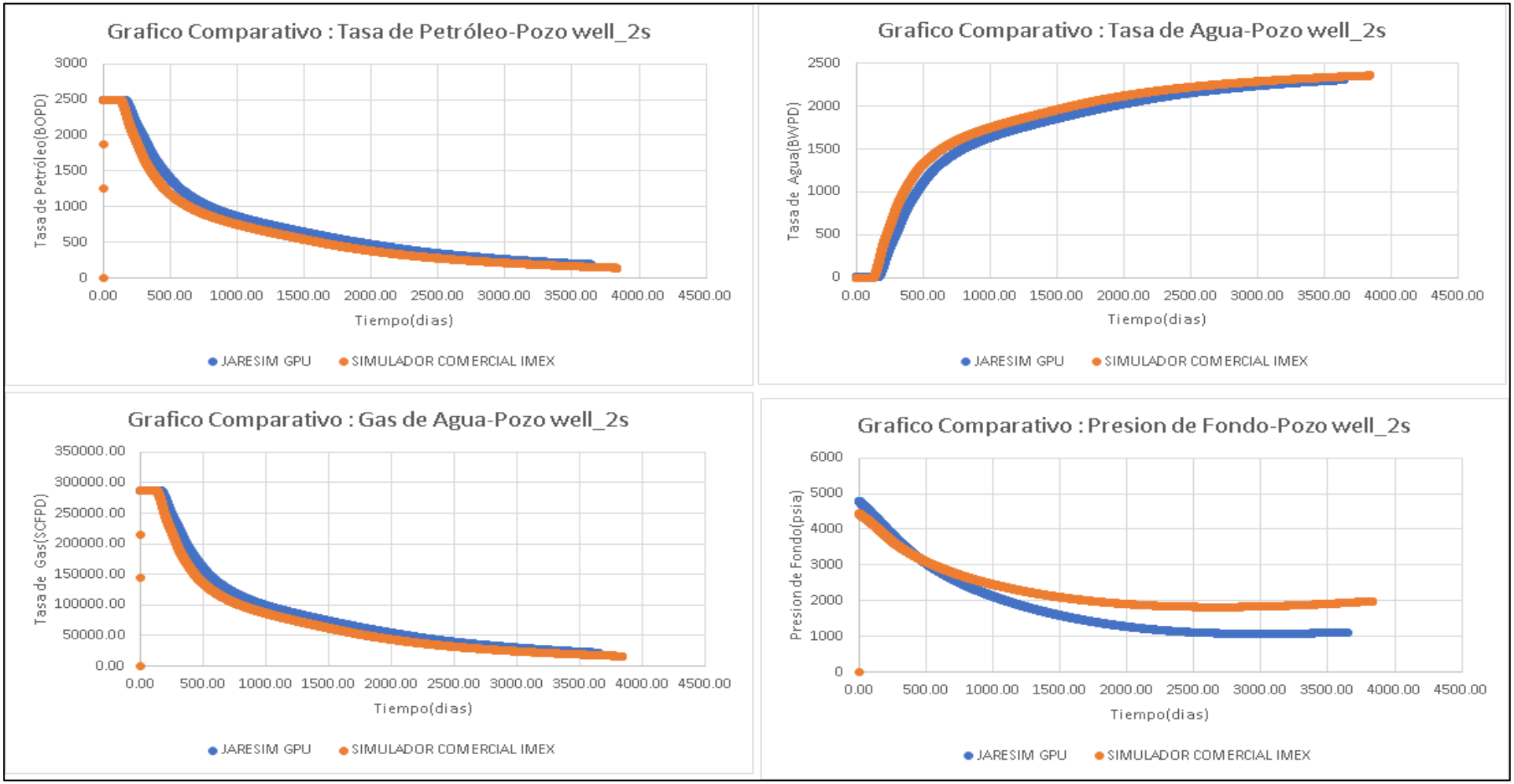


Ilustración 60:Gráficos comparativos a nivel para el Pozo well_2s de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

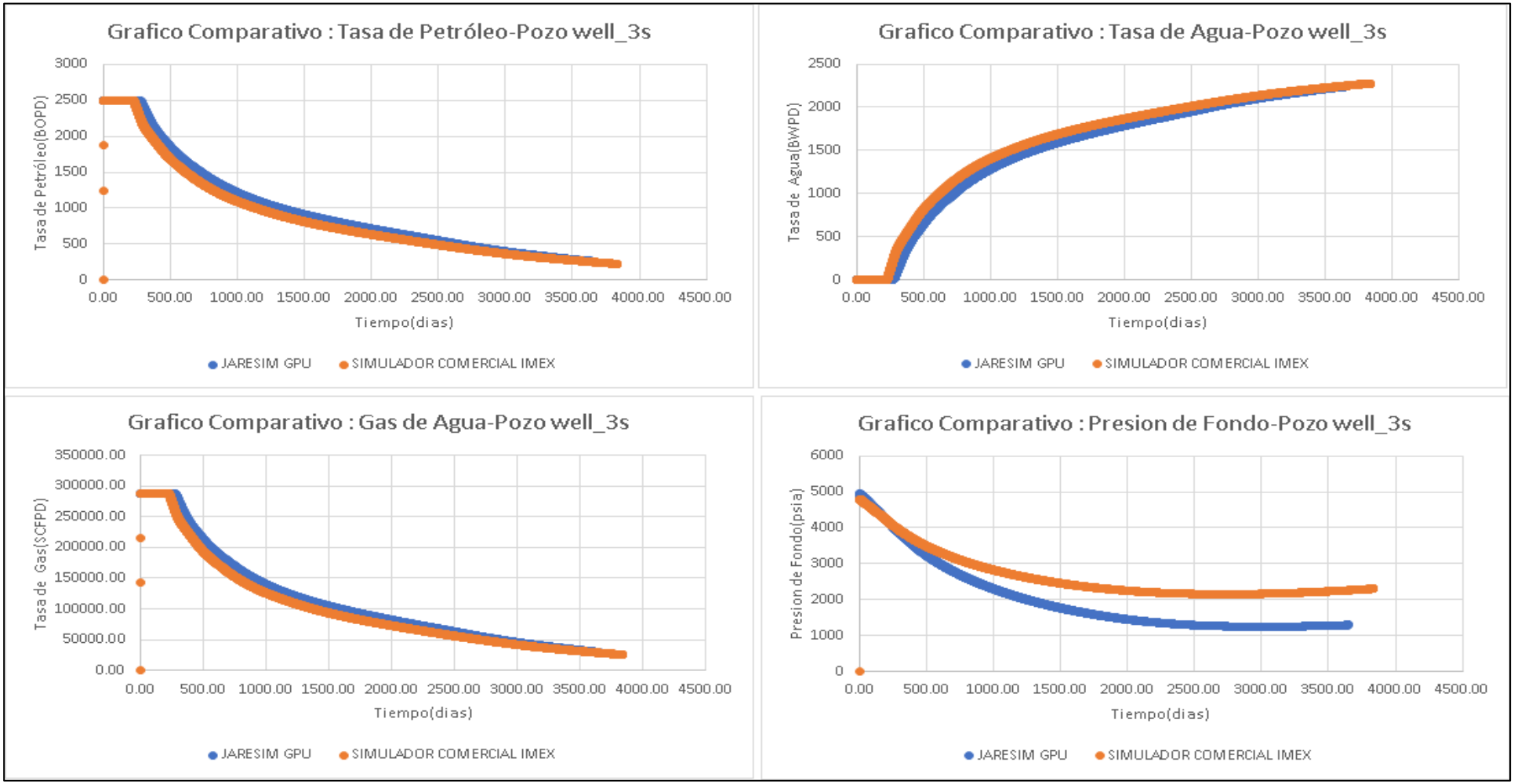


Ilustración 61:Gráficos comparativos a nivel para el Pozo well_3s de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

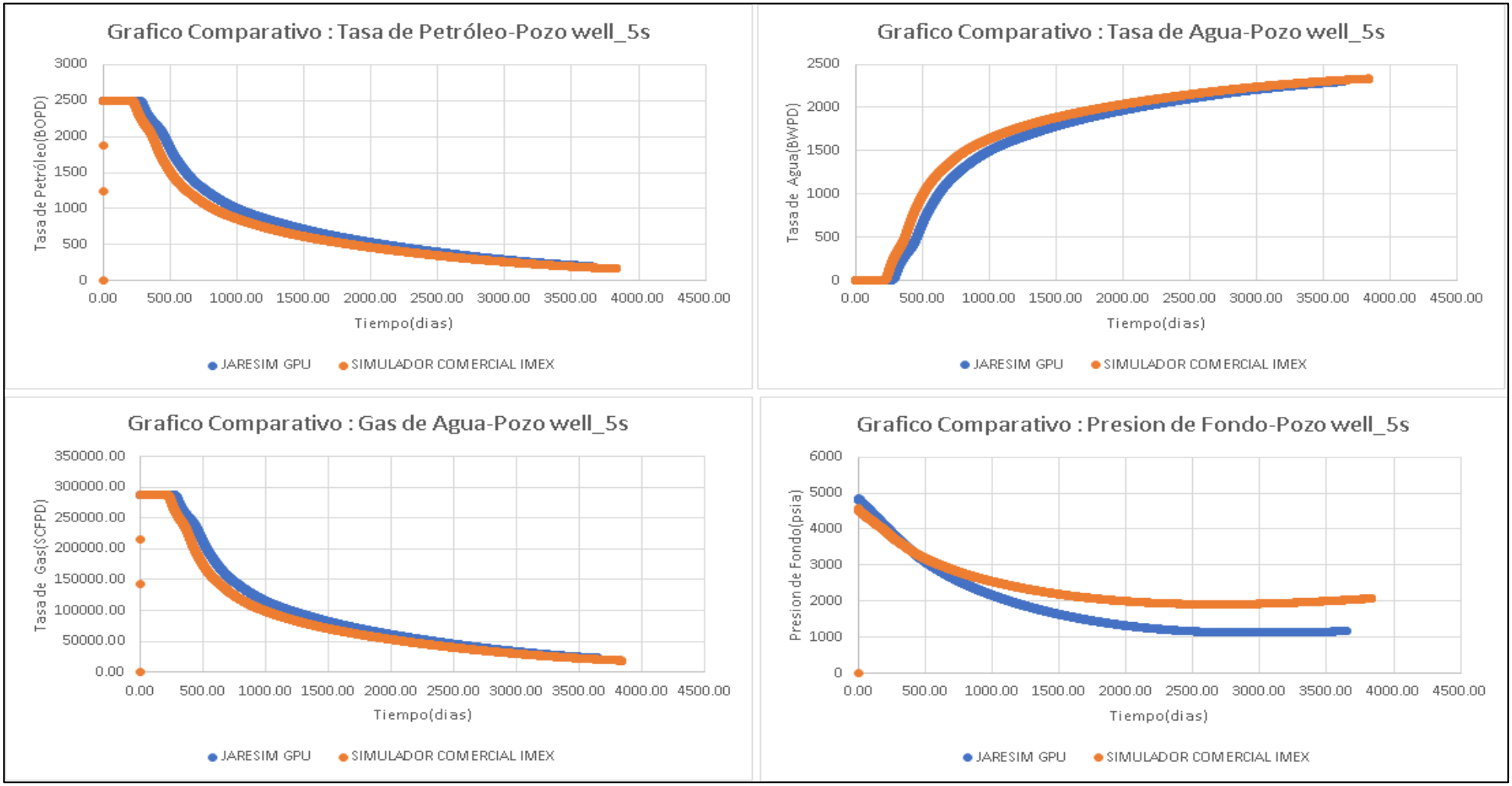


Ilustración 62:Gráficos comparativos a nivel para el Pozo well_5s de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

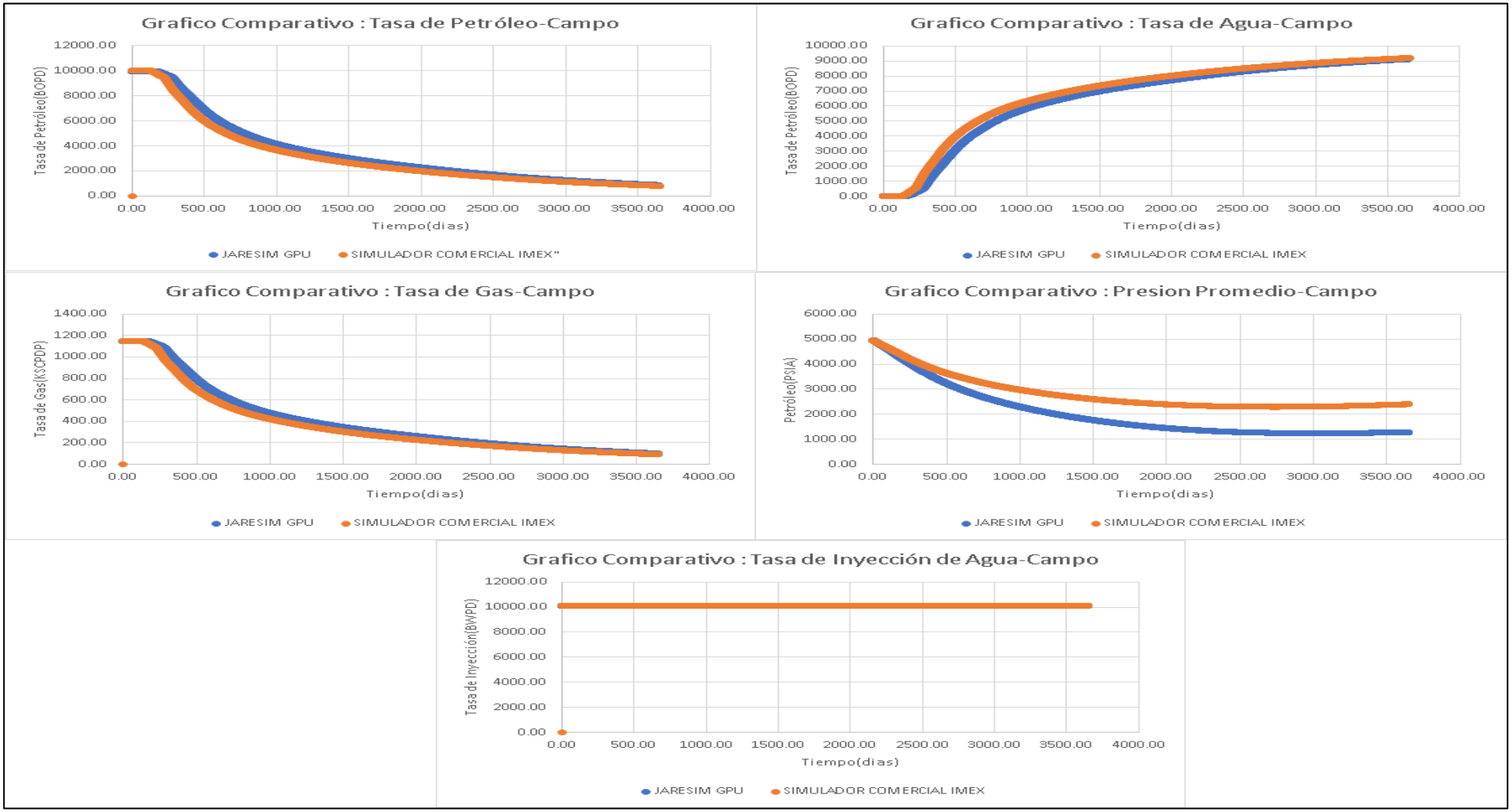


Ilustración 63:Gráficos comparativos a nivel de campo de los resultados obtenidos en el caso de Validación para: ·Simulador desarrollado en la Investigación (puntos azules) y Simulador Comercial IMEX de la Compañía CMG (puntos naranjas). Fuente: Elaboración propia.

Las ilustraciones presentadas evidencian que el simulador propuesto en este proyecto (puntos azules de las ilustraciones) tiene la capacidad de reproducir con precisión la tendencia de los resultados obtenidos en el programa comercial (puntos naranjas de las ilustraciones), al comparar sus resultados de las tasas de petróleo (gráfico superior-izquierdo), agua(gráfico superior-derecho), gas (gráfico inferior-izquierdo), y presión (gráfico inferior-derecho), a nivel de los pozos well_1s, well_2s, well_3s, well_5s del modelo, así como los resultados obtenidos a nivel de campo.

A continuación, se muestra la Tabla 4 comparativa de los acumulados obtenidos por el Simulador propuesto por esta investigación respecto al simulador comercial de referencia:

TABLA RESUMEN VALORES ACUMULADOR Y PRESION AL FINAL DE LA CORRIDA DEL CASO 2

CASO	SIMULADOR	FECHA	Np(BO)	Wp(BW)	Gp(KSCF)	Wi(BW)	Presion(psia)
CASO 2	JARESIM GPU	1/3/2030	12663695.3	23771304.7	1456286.1	36799350.0	1292.0
CASO 2	IMEX-CMG	1/3/2030	11446881.0	25033120.0	1316379.8	36844800.0	2405.9

Tabla 4:Tabla resumen de los valores acumulados a nivel de Campo para caso de Validación #2: para el Simulador desarrollado en la Investigación (JARESIM GPU) y Simulador Comercial (IMEX-CMG). Fuente: Elaboración propia.

Adicionalmente, se presenta la Tabla 5 comparativa del % de las diferencias de los acumulados obtenidos por el simulador propuesto por esta investigación, respecto al simulador comercial de referencia:

TABLA RESUMEN DIFERENCIAS ASOCIADO AL FINAL DE LA CORRIDA DEL CASO 2 PARA EL SIMULADOR JARESIM

CASO	SIMULADOR	FECHA	Np Diferencia (%)	Wp Diferencia (%)	Gp Diferencia (%)	Wi Diferencia (%)	Presion Diferencia (%)
CASO 2	JARESIM GPU	1/3/2030	10.63%	-5.04%	10.63%	-0.12%	-46.30%

Tabla 5:Tabla resumen del porcentaje de desviación de los valores acumulados a nivel de Campo para caso de Validación #2: para el Simulador desarrollado en la Investigación (JARESIM GPU) respecto al simulador Comercial (IMEX-CMG). Fuente: Elaboración propia.

Con base a estos resultados, es posible concluir para este caso que el simulador propuesto en este proyecto, usando el procesamiento GPU, fue capaz de reproducir los resultados obtenidos por un simulador comercial siguiendo la tendencia de los resultados obtenidos, mostrando desviaciones menores a 10.63% a nivel de los acumulados, y 46% a nivel de la presión, considerando el hecho de usar un modelo totalmente heterogéneo.

Finalmente, los resultados comparativos entre el procesamiento CPU y GPU en la aplicación desarrollada, no se muestran debido a la extensión del documento. Sin embargo, los módulos CPU y CPU paralelo producen exactamente los mismos resultados presentados para ambos casos en esta sección, donde se validó el Módulo GPU al comparar sus resultados con el simulador comercial IMEX.

8. Conclusiones y líneas futuras.

8.1. Conclusiones

- Fue posible implementar OpenCL a través de uso de la librería JOCL para la realización de un simulador de yacimientos de petróleo negro, la cual es una herramienta de cálculo de alto desempleo en JAVA.
- Se desarrolló un módulo de procesamiento GPU como objetivo principal, sin embargo, de forma complementaria fueron desarrollados un módulo CPU base y un módulo CPU con procesamiento Paralelo, haciendo que el programa propuesto en este proyecto pueda tener una gran flexibilidad en la selección de los módulos de procesamiento.
- Se desarrolló una interface gráfica que permite el fácil ingreso de los datos necesarios para la generación del caso de simulación, adicionalmente se implementaron ciertas correlaciones a nivel de propiedades de fluidos y de roca, que permite tener estos parámetros en caso que no se dispongan.
- Se observó durante el desarrollo del proyecto que métodos de resolución de sistemas lineales usados en este proyecto (Gauss-Seidel, Jacobi, Matriz Tridiagonal y Gradiente Conjugado) a nivel del código GPU, presentaron mayor complejidad en el desarrollo de algoritmos paralelizables que le resto de los módulos, haciendo que no fuera posible su implementación en el Módulo GPU, dentro de los plazos de desarrollo del proyecto. Haciendo que la aplicación GPU sea más bien un módulo Heterogéneo GPU-CPU, más que un módulo puramente GPU.
- Se generaron dos casos de validación donde el primer caso considerado corresponde a una malla de geometría rectangular creada con propiedades constantes por capa, y un segundo caso con una malla con geometría Corner Point importada con propiedades totalmente heterogéneas.
- Para los casos de validación propuestos se realizó una corrida con el módulo GPU propuesto por el programa desarrollado por esta investigación, y una segunda corrida usando el simulador comercial IMEX de CMG como caso de referencia para realizar la validación de los resultados obtenidos.
- Los resultados del caso #1 de validación de la aplicación, mostró que la aplicación desarrollada fue capaz de reproducir los resultados obtenidos por el simulador comercial, siguiendo la tendencia de los resultados obtenidos, mostrando

desviaciones menores a 6.8% a nivel de los acumulados, y 0.002% a nivel de la presión.

- Los resultados del caso #2 de validación de la aplicación, mostró que la aplicación desarrollada fue capaz de reproducir los resultados obtenidos por un simulador comercial, siguiendo la tendencia de los resultados obtenidos, mostrando desviaciones menores a 10.63% a nivel de los acumulados, y 46% a nivel de la presión, considerando el hecho de usar un modelo totalmente heterogéneo.
- Con base a los resultados de ambos casos generados, es posible concluir que la aplicación desarrollada en este proyecto presenta la capacidad de reproducir resultados de un simulador comercial, garantizando posterior aplicabilidad a escala académica y comercial.
- Para garantizar la precisión de los resultados GPU fue necesario usar variables de tipo *double* en vez de *float.*

8.2. Líneas futuras de trabajo

- Se recomienda estudiar la viabilidad de implementación de este tipo de procesos en el diseño de simuladores numéricos de todas de la ingeniería, y evaluar el impacto de su uso en los cálculos generados.
- Se recomienda optimizar los módulos de solución de sistema lineales para una ejecución mucho más rápida.
- Se recomienda desarrollar con base a los códigos desarrollando, aplicaciones orientadas a dispositivos Android, Tablets (Tabletas) y IPad.

9. Referencias.

Eclipse Foundation. (s.f.). *Eclipse Foundation*. Recuperado el 30 de 05 de 2020, de Eclipse Foundation: https://www.eclipse.org/org/

Plano Research Corporation. (s.f.). *planoresearch*. Recuperado el 30 de 05 de 2020, de planoresearch: https://www.planoresearch.com/flowsim-studio-detail

Aaftab, M., R., G. B., G., M. T., James, F., & Dan, G. (2012). OpenCL Programming Guide. En *OpenCL Programming Guide* (pág. 648). Addison-Wesley. doi:ISBN-13: 978-0-321-74964-2; ISBN-10: 0-321-74964-2

Ahmed, T. (2006). *Reservoir Engineering HandBook* (Third Edition ed.). Elservier. doi:ISBN 13: 978-0-7506-7972-5

Almengor, J. G., Penuela, G., Wiggins, M. L., Browm, R., Civan, D., & Hughes, R. (June, 2006). *BOAST 98-MC:Probabilistic Simulation Module for BOAST 98.*

Anciaux-Sedrakian, A., Eaton, J., Gratien, J., Guignon, T., Havé, P., Preux, C., & Ricois, O. (23 de 02 de 2015). Will GPGPUs be Finally a Credible Solution for Industrial Reservoir Simulators? *Society of Petroleum Engineers*. doi:10.2118/173223-MS

Appleyard, J., Appleyard, J., Wakefield, M., & Desitter, A. (01 de 01 de 2011). Accelerating Reservoir Simulators using GPU Technology. *Society of Petroleum Engineers*. doi:10.2118/141402-MS

Breitenbach, E. A., Thurnau, D. H., & Van Poolen, H. K. (01 de 01 de 1968). Immiscible Fluid Flow Simulator. *Society of Petroleum Engineers*. doi:10.2118/2019-MS

Buchwalter, J. L., & Miller, C. A. (01 de 01 de 1993). A New Simplified Compositional Simulator. *Society of Petroleum Engineers*. doi:10.2118/25858-MS

Carl, D., Mark, H., Gerrit, G., Pereda, J., & Phillips, S. M. (2014). *JavaFX 8 Introduction by Example* (Second Edition ed.). Apress. doi:ISBN 978-1-4302-6461-3

Chen, W. H., Wasserman, M. L., & Fitzmorris, R. E. (01 de 01 de 1987). A Thermal Simulator for Naturally Fractured Reservoirs. *Society of Petroleum Engineers*. doi:10.2118/16008-MS

CHEN, Z., & ZHANG, Y. (30 de agust de 2008). WELL FLOW MODELS FOR VARIOUS NUMERICAL METHODS. *INTERNATIONAL JOURNAL OF NUMERICAL ANALYSIS AND MODELING*, 375-388. Obtenido de http://www.math.ualberta.ca/ijnam/Volume-6-2009/No-3-09/2009-03-02.pdf

Cheshire, I. M., Appleyard, J. R., Banks, D., Crozier, R. J., & Holmes, J. A. (01 de 01 de 1980). An Efficient Fully Implicit Simulator. *Society of Petroleum Engineers*. doi:10.2118/179-1980-MS

Chris, M. (2010). History and Evolution of GPU Architecture. *A Paper Survey*, 7. Obtenido de https://pdfs.semanticscholar.org/2479/80e834f1c8f684d85067402f950930e6af91.pdf?_ga=2.256915147.1974529909.1568145038-2095758839.1568145038

Computer Modelling Group (CMG). (2014-2020). *CMG-IMEX:Simulador Trifasico Black Oil Tridimensional.* Recuperado el 28 de 05 de 2020

Computer Modelling Group L.T.D. (s.f.). *cmg*. Recuperado el 30 de 05 de 2020, de cmg: https://www.cmgl.ca/

Crane, M., Bratvedt, F., Bratvedt, K., Childs, P., & Olufsen, R. (01 de 01 de 2000). A Fully Compositional Streamline Simulator. *Society of Petroleum Engineers*. doi:10.2118/63156-MS

Deb, M. K., Reddy, M. P., Thuren, J. B., & Adams, W. T. (01 de 01 de 1995). A New Generation Solution Adaptive Reservoir Simulator. *Society of Petroleum Engineers*. doi:10.2118/30720-MS

DeepSim. (s.f.). *DeepSim*. Recuperado el 30 de 05 de 2020, de DeepSim: https://deepsim.stupendous.org/

Digital, C. G. (s.f.). *The MATLAB Reservoir Simulation Toolbox (MRST)*. Recuperado el 31 de 05 de 2020, de The MATLAB Reservoir Simulation Toolbox (MRST): https://www.sintef.no/MRST

Durlofsky, L. J., & Chien, M. C. (01 de 01 de 1993). Development of a Mixed Finite-Element-Based Compositional Reservoir Simulator. *Society of Petroleum Engineers*. doi:10.2118/25253-MS

ECHELON, t. w. (s.f.). *ECHELON, the world's fastest reservoir simulator*. Recuperado el 30 de 05 de 2020, de ECHELON, the world's fastest reservoir simulator: https://www.stoneridgetechnology.com/

Emerson Electric Co. (s.f.). *Tempest MORE*. Recuperado el 30 de 05 de 2020, de Tempest MORE: https://www.emerson.com/en-us/catalog/roxar-tempest-more

Engineering Support & Technology Development Company (ESTD). (s.f.). *RETINA Simulation*. Recuperado el 30 de 05 de 2020, de RETINA Simulation: http://en.tafahomco.ir/Main.html

Engineering, C. (s.f.). *SENSOR-System for Efficient Numerical Simulation of Oil Recovery*. Recuperado el 30 de 05 de 2020, de SENSOR-System for Efficient Numerical Simulation of Oil Recovery: https://www.coatsengineering.com/

Gemini Solutions Inc. (s.f.). *Merlin Simulator*. Recuperado el 30 de 05 de 2020, de Merlin Simulator: http://www.geminisi.com/index.php/merlin

GRAILQUEST CORP. (s.f.). *reservoirgrail*. Recuperado el 30 de 05 de 2020, de reservoirgrail: http://reservoirgrail.com/

Halliburton. (s.f.). *landmark.solutions*. Obtenido de landmark.solutions: https://www.landmark.solutions/Nexus-Reservoir-Simulation

Han, G., Stone, T., Liu, Q., Cook, J., & Papanastasiou, P. (01 de 01 de 2005). 3D Elastoplastic FEM Modelling in a Reservoir Simulator. *Society of Petroleum Engineers*. doi:10.2118/91891-MS

Hemanth-Kumar, K., & Young, L. C. (01 de 03 de 1996). Parallel Reservoir Simulator Computations. *Society of Petroleum Engineers*. doi:10.2118/29104-PA

Huang, C. K., Shiralkar, G. S., Li, G., & Abbas, S. (14 de 04 de 2018). A New Chemical EOR Surfactant Polymer Flood Model. *Society of Petroleum Engineers*. doi:10.2118/190254-MS

Initiative, T. O. (s.f.). *The Open Porous Media Initiative*. Recuperado el 30 de 05 de 2020, de The Open Porous Media Initiative: https://opm-project.org/?page_id=19

Islam, M. R., Moussavizadegan, S., Mustafiz, S., & Abou-Kassem, J. (2010). *Advanced Petroleum Reservoir Simulation.* Scrivener Publishing LLC;John Wiley & Sons,.

jocl.org. (s.f.). Recuperado el 28 de 05 de 2020, de Java bindings for OpenCL: http://www.jocl.org/

John, A., Han, C., Delshad, M., Pope, G. A., & Sepehrnoori, K. (01 de 06 de 2005). A New Generation Chemical Flooding Simulator. *Society of Petroleum Engineers*. doi:10.2118/89436-PA

Jordan Russell. (1997-2019). Inno Setup version 6.0.3. *Inno Setup version 6.0.3*.

Ju, B., Dai, S., Qiu, X., Wu, H., Li, S., & Zhang, M. (01 de 01 de 2006). A Novel 3D Field-Scale Reservoir Numerical Simulator for predicting the Fines Migration and Production Performance. *Society of Petroleum Engineers*. doi:10.2118/99797-MS

Kaarstad, T., Froyen, J., Bjorstad, P., & Espedal, M. (01 de 01 de 1995). A Massively Parallel Reservoir Simulator. *Society of Petroleum Engineers*. doi:10.2118/29139-MS

Kawada, N., Okubo, K., Tagawa, N., Tsuchiya, T., & Ishizuka, T. (6 de August de 2015). GPU-computing-based high-speed visualization techniques. *The Acoustical Society of Japan for sound wave propagation using permeable multi-cross-section contours, 36*(3), 3. doi:doi:10.1250

Kazemi, H., Vestal, C. R., & Shank, D. G. (01 de 10 de 1978). An Efficient Multicomponent Numerical Simulator. *Society of Petroleum Engineers*. doi:10.2118/6890-PA

khronos.org. (s.f.). *OPEN CL*. (T. K. Inc., Editor) Recuperado el 28 de 05 de 2020, de OPEN STANDARD FOR PARALLEL PROGRAMMING OF HETEROGENEOUS SYSTEMS: https://www.khronos.org/opencl/

Lashgari, H., Pope, G., Balhoff, M., & Tagavifar, M. (29 de 03 de 2019). New and Improved Physical Property Models for Chemical Flooding Simulators. *Society of Petroleum Engineers*. doi:10.2118/193930-MS

Launay, P., & Pazat, J.-L. (24 de May de 2006). A Framework for Parallel Programming in Java. *A Framework for Parallel Programming in Java*. Lecture Notes in Computer Science. doi:DOI: 10.1007

Levchenko, V., Anastasia, P., & Zakirov, A. Z. (August de 2016). DiamondTorre Algorithm for High-Performance. (D. T. Tsahalis, Ed.) *computation, Computation 2016, 4, 29*, 1-15. doi:doi:10.3390

Meera Simulation. (s.f.). *Meera Simulation*. Recuperado el 30 de 05 de 2020, de Meera Simulation: https://meerasimulation.com/

Micikevicius,Paulius;NVIDIA. (March de 2009). 3D Finite Difference Computation on GPUs using CUDA. *GPGPU2;NVIDIA*, 1-6. Obtenido de https://developer.download.nvidia.com/CUDA/CUDA_Zone/papers/gpu_3dfd_rev.pdf

Mohammed Ali, M. (2016). Parallel Computing in Java. St. Cloud State University. Obtenido de https://repository.stcloudstate.edu/csit_etds/9/

Mohammed, S., Langguth, J., Spiga, F., Baden, S., & Cai, X. (2015). CPU+GPU Programming of Stencil Computations for Resource-Efficient Use of GPU Clusters. *IEEE 18th International Conference on Computational Science and Engineering, 18*, 10. doi:DOI 10.1109

Mukundakrishnan, K., Esler, K., Dembeck, D., Natoli, V., Shumway, J., Zhang, Y., & … Meng, H. (23 de 02 de 2015). Accelerating Tight Reservoir Workflows With GPUs. *Society of Petroleum Engineers*. doi:10.2118/173246-MS

NIKLAS, K. (August de 2013). An Incompressible Navier-Stokes Equations Solver on the GPU Using CUDA. *An Incompressible Navier-Stokes Equations Solver on the GPU Using CUDA*. Goteborg,, Sweden: Chalmers University of Technology University of Gothenburg Department of Computer Science and Engineering. Obtenido de http://publications.lib.chalmers.se/records/fulltext/185027/185027.pdf

Nvidia Corporation. (2018). *GPU-ACCELERATED APPLICATIONS* (Vol. AGO). POPULAR GPU-ACCELERATED APPLICATIONS CATALOG. Obtenido de https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/tesla-product-literature/gpu-applications-catalog.pdf

Oracle. (s.f.). *JavaFX: Getting Started with JavaFX*. Recuperado el 28 de 05 de 2020, de https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm#JFXST784

petromehras. (s.f.). *petromehras*. Obtenido de https://www.petromehras.com/petroleum-software-directory/reservoir-simulation-software/dynamic-simulation-software/excsim

Pina, C. A. (2012). Simulacion de fluidos en Open CL. *Disertation project, 1/1*. Arquitectura de Computadores (AC). Obtenido de

https://upcommons.upc.edu/bitstream/handle/2099.1/15699/79635.pdf?sequence=1&isAllowed=yy

Pope, G. A., & Nelson, R. C. (01 de 10 de 1978). A Chemical Flooding Compositional Simulator. *Society of Petroleum Engineers*. doi:10.2118/6725-PA

Ramos R, J. L. (08 de 04 de 2008). PROGRAMA COMPUTARIZADO PARA LA GENERACIÓN DE DISTRIBUCIONES ESPACIALES DE POROSIDAD A TRAVÉS DE LA SIMULACIÓN SECUENCIAL GAUSSIANA. *PROGRAMA COMPUTARIZADO PARA LA GENERACIÓN DE DISTRIBUCIONES ESPACIALES DE POROSIDAD A TRAVÉS DE LA SIMULACIÓN SECUENCIAL GAUSSIANA*. Maracaibo, Zulia, Venezuela.

Rock Flow Dynamics. (s.f.). *rfdyn*. Recuperado el 30 de 05 de 2020, de rfdyn: https://rfdyn.com/tnavigator/

Saeed, A. K., & Perot, B. (May de 2012). Computational Fluid Dynamics Simulations. 11. doi:DOI: 10.1109/MCSE.2011.117

Schlumberger Limited. (s.f.). *ECLIPSE Industry-Reference Reservoir Simulator*. Recuperado el 30 de 05 de 2020, de ECLIPSE Industry-Reference Reservoir Simulator: https://www.software.slb.com/products/eclipse

Schlumberger Limited. (s.f.). *INTERSECT High-Resolution Reservoir Simulator*. Recuperado el 30 de 05 de 2020, de INTERSECT High-Resolution Reservoir Simulator: https://www.software.slb.com/products/intersect?gclid=EAIaIQobChMI2Kjhy9GO6gIVCYizCh3M5gxoEAAYAiAAEgKYx_D_BwE

Science, E. (2012). *Advances in Parallel Computing* (Vol. 21). (P. D.-E. Science., Ed.) Volumes 1–21 published by Elsevier Science. doi:ISSN 1879-808X

Shankar, S., Kalyanapu, A. J., Hansen, C. D., & Burian, S. J. (2011). A GPU-based Flood Simulation Framework. Obtenido de http://www.cs.utah.edu/~sshankar/saahpc/abstract.pdf ; https://www.semanticscholar.org/paper/A-GPU-based-Flood-Simulation-Framework-Shankar-Kalyanapu/31923105b1a3b668c9dbdafe6203edddae6bf26a

Society of Petroleum Engineers. (2012-2020). *PetroWiki-Relative_permeability_models*. Recuperado el 01 de 01 de 2020, de PetroWiki-Relative_permeability_models: https://petrowiki.org/Relative_permeability_models

SPE Series Reservoir SImulation. (1990). En C. Mattax, & R. Dalton, *SPE Series Reservoir SImulation* (Vol. 13, págs. 140-150). SPE Series-Henry L. Doherty Series.

T.T. & Associates Inc. (s.f.). *EXODUS*. Recuperado el 30 de 05 de 2020, de EXODUS: http://www.petrostudies.com/exodus.htm

technologycatalogue. (s.f.). *XXSim Reservoir Simulator*. Obtenido de XXSim Reservoir Simulator: https://www.technologycatalogue.com/product_service/xxsim-reservoir-simulator

Wang, B., Lake, L. W., & Pope, G. A. (01 de 01 de 1981). Development and Application of a Streamline Micellar/Polymer Simulator. *Society of Petroleum Engineers*. doi:10.2118/10290-MS

Wang, P., Balay, S., Sepehrnoori, K., Wheeler, J., Abate, J., Smith, B., & Pope, G. A. (01 de 01 de 1999). A Fully Implicit Parallel EOS Compositional Simulator for Large Scale Reservoir Simulation. *Society of Petroleum Engineers*. doi:10.2118/51885-MS

Youngren, G. K. (01 de 02 de 1980). Development and Application of an In-Situ Combustion Reservoir Simulator. *Society of Petroleum Engineers*. doi:10.2118/7545-PA

Zaza, A., Awotunde, A. A., Fairag, F. A., & Al-Mouhamed, M. A. (2 de January de 2016). A CUDA based parallel multi-phase oil reservoir simulator. (C. P. Communications, Ed.) *206*, 2-16. Obtenido de https://doi.org/10.1016/j.cpc.2016.04.010

10. Anexos.

10.1. Caso de validación #1

Se describirá la información asociada al caso de validación #1, correspondiente a una malla rectangular con propiedades heterogéneas por capa, que incluye 5 pozos dentro del modelo para un periodo de predicción de 25 años. Posteriormente, se podrán observar los resultados obtenidos de esta evaluación, realizando una comparación de los resultados del simulador desarrollado en este proyecto y el simulador Comercial IMEX de la Compañía CMG:

10.1.1. Datos de Entrada

A continuación, se presenta la información usada para construir el caso de validación #1 de la aplicación desarrollada en este proyecto:

Datos de Geometría de Modelo

La información relativa a la geometría del modelo de validación usado, se resume a continuación:

Parámetro	Valor	Unidad
Dimensión de Celdas X	500	ft
Dimensión de Celda Y	1500	ft
Dimensión de Celda Z	5	ft
Origen X	0	ft
Origen Y	0	ft
Origen Z	-7120	ft
Número de Celdas X	25	
Número de Celdas Y	25	
Número de Celdas Z	4	
Tipo de Malla	Creada Rectangular-Corner Point	

Tabla 6:Tabla resume de Datos de Geometría del caso de Validación #1. Fuente: Elaboración propia.

A continuación, se muestra la representación 3D de la malla generada para la realización del caso de validación:

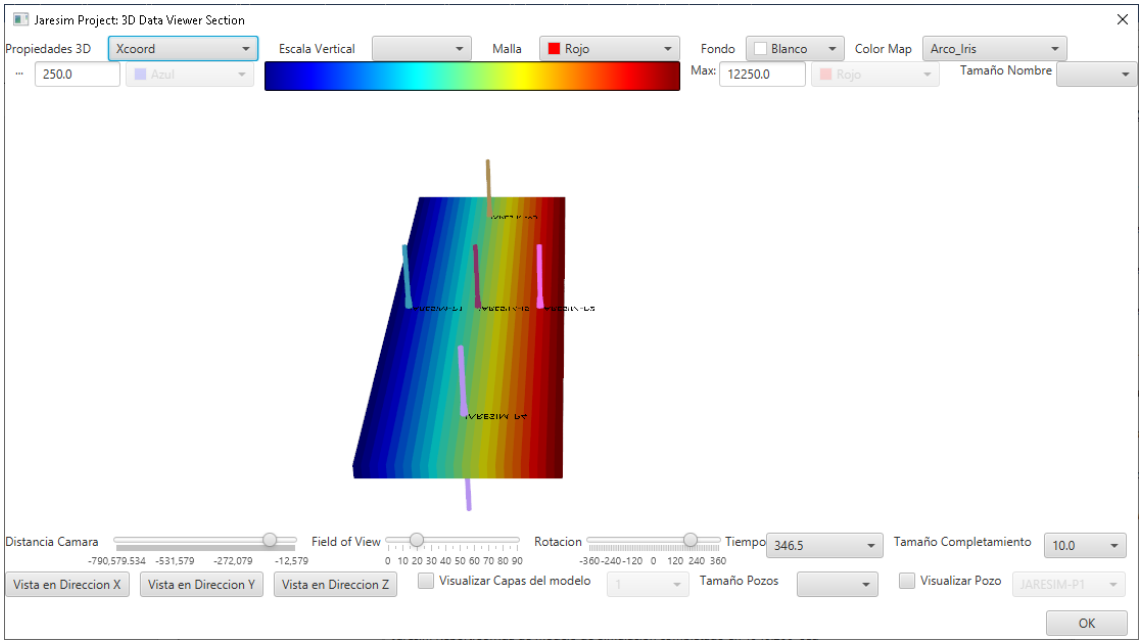


Ilustración 64:Visualización 3D de la Geometría del Modelo del caso de Validación#1 mediante la Aplicación desarrollado en este proyecto. Fuente: Elaboración propia.

Datos de Propiedades Petrofísicas

La información relativa a las propiedades petrofísicas del modelo de validación usado, se resume en la tabla mostrada a continuación:

CAPA	TOPE(ft)	PHIE	GROSS(ft)	NTG	Kx(mD)	Ky(mD)	Kz(mD)
1	-7122.5	0.299	5	1	4985.040	4985.040	4985.040
2	-7127.5	0.200	5	1	2340.860	2340.860	2340.860
3	-7132.5	0.250	5	1	3901.690	3901.690	3901.690
4	-7137.5	0.290	5	1	1670.210	1670.210	1670.210

Tabla 7:Tabla resume de Datos de Propiedades Estáticas del caso de Validación #1. Fuente: Elaboración propia.

Donde:

- PHIE: Es la porosidad Efectiva en fracción.
- GROSS: Es el espesor total promedio por celda en pies (ft).
- NTG: Es el Net to gross en fracción.
- Kx: Es la permeabilidad Absoluta en Dirección X en miliDarcies.
- Ky: Es la permeabilidad Absoluta en Dirección Y en miliDarcies.
- Kz: Es la permeabilidad Absoluta en Dirección Z en miliDarcies.

A continuación, se muestra la representación 3D de la propiedad Porosidad, usada para la realización del caso de validación:

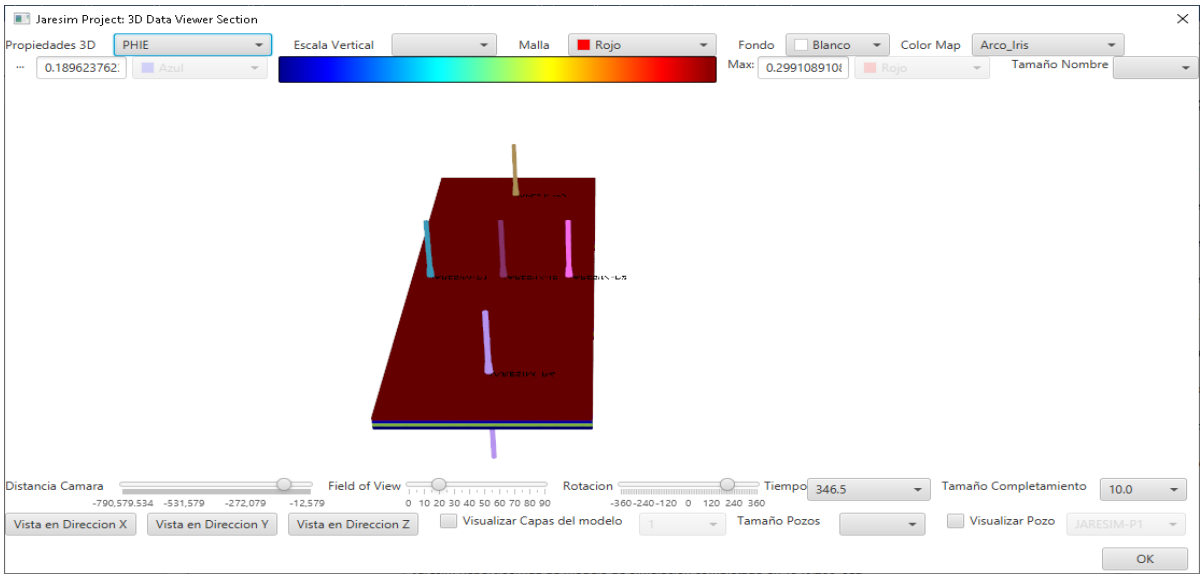


Ilustración 65:Visualización 3D de la Porosidad usada del Modelo del caso de Validación#1 mediante la Aplicación desarrollado en este proyecto. Fuente: Elaboración propia.

Posteriormente, se presenta la información relativa a compresibilidad de la roca y cambio del volumen poroso, en función de la presión del yacimiento, usado en el modelo de validación que se resume en la tabla mostrada a continuación:

P(psia)	Cr (psi-1)
14.7	4.342E-06
150	4.396E-06
250	4.437E-06
500	4.542E-06
750	4.651E-06
1000	4.765E-06
1500	5.006E-06
2000	5.268E-06
2500	5.556E-06
3000	5.874E-06
3500	6.228E-06
4000	6.626E-06
4500	7.080E-06
5000	7.607E-06
5500	8.229E-06
6000	8.984E-06

Tabla 8:Tabla resume de Datos de Compresibilidad de Roca del caso de Validación #1. Fuente: Elaboración propia.

Finalmente, se presenta la curva presión vs compresibilidad de la roca, usada para la realización del caso de validación:

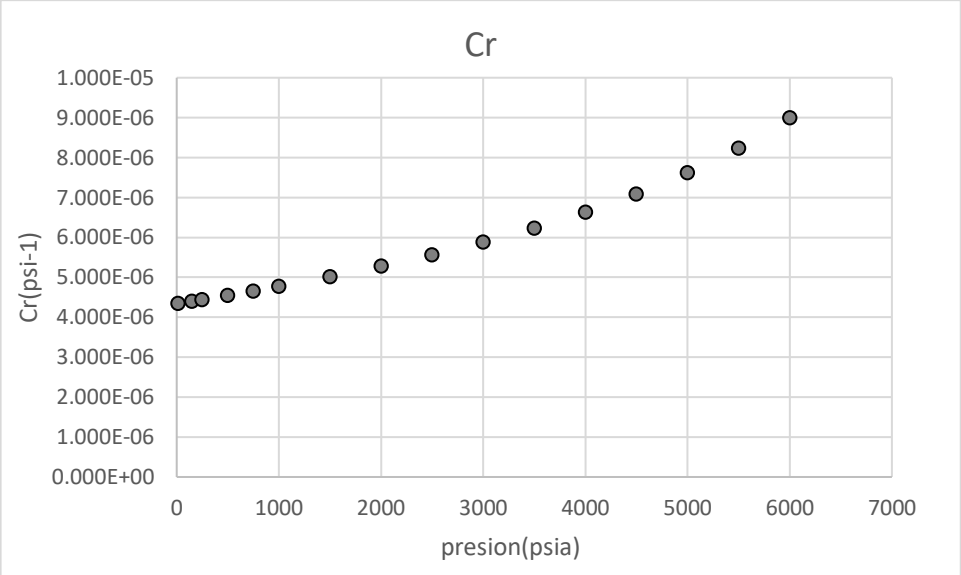


Ilustración 66:Gráfico de Curva P vs Cr de las propiedades Petrofísicas del caso de Validación#1. Fuente: Elaboración propia.

Datos de Propiedades PVT

Inicialmente, presenta la información relativa a la data PVT del modelo de fluido, usado en el modelo de validación que se resume en las tablas mostradas a continuación:

Parámetro	Valor	Unidad
MODELO DE FLUIDO	BLACK OIL	
PROCESO ISOTERMICO	SI	
FASES PRESENTES	OIL-GAS-WATER	
API	28.0000	°
Gas Gravity	0.6800	
Rsb	114.9990	SCF/STB
Pb	850.0000	psia
T yac	199.1665	°F
Den Oil	55.3580	lb/ft3
Den Water	62.3397	lb/ft3
Den Gas	15.5167	lb/ft3

Tabla 9:Tabla resume de Datos PVT del caso de Validación #1. Fuente: Elaboración propia.

Pressure (psia)	Z ()	Bg (cf/scf)	Mug (cp)	Rs (cf/bbl)	Bo (B/STB)	Muo (cp)	Rsw (cf/bbl)	Bw (B/STB)	Muw (cp)
14.70	0.9990	1.26927	0.01343	2.8082	1.0701	3.0444	0.0000	1.0370	0.3275
64.70	0.9957	0.28736	0.01345	7.4370	1.0719	2.9427	0.3592	1.0368	0.3275
114.70	0.9923	0.16154	0.01348	12.6532	1.0740	2.8368	0.7128	1.0366	0.3275
214.70	0.9855	0.08570	0.01355	24.2070	1.0787	2.6302	1.4032	1.0363	0.3275
314.70	0.9785	0.05806	0.01364	36.8191	1.0838	2.4402	2.0712	1.0360	0.3275
414.70	0.9716	0.04374	0.01374	50.2246	1.0892	2.2695	2.7167	1.0357	0.3275
514.70	0.9646	0.03499	0.01386	64.2737	1.0950	2.1174	3.3397	1.0353	0.3275
614.70	0.9578	0.02909	0.01399	78.8687	1.1010	1.9822	3.9403	1.0350	0.3275
814.70	0.9444	0.02164	0.01427	109.4351	1.1138	1.7544	5.0742	1.0344	0.3275
864.70	0.9412	0.02032	0.01435	114.9990	1.1161	1.7214	5.2650	1.0342	0.3275
914.70	0.9380	0.01915	0.01443	114.9990	1.1158	1.7292	5.2650	1.0341	0.3275
964.70	0.9349	0.01809	0.01452	114.9990	1.1156	1.7375	5.2650	1.0339	0.3275
1014.70	0.9318	0.01715	0.01461	114.9990	1.1153	1.7462	5.2650	1.0337	0.3275
2014.70	0.8892	0.00824	0.01687	114.9990	1.1123	2.0105	5.2650	1.0307	0.3275
3014.70	0.8903	0.00551	0.01993	114.9990	1.1105	2.4121	5.2650	1.0278	0.3275
4014.70	0.9313	0.00433	0.02325	114.9990	1.1092	2.9306	5.2650	1.0251	0.3275
5014.70	0.9984	0.00372	0.02640	114.9990	1.1082	3.5527	5.2650	1.0225	0.3275
5514.70	1.0376	0.00351	0.02787	114.9990	1.1077	3.8976	5.2650	1.0213	0.3275
6014.70	1.0789	0.00335	0.02928	114.9990	1.1073	4.2616	5.2650	1.0202	0.3275

Tabla 10:Continuación Tabla resume de Datos PVT del caso de Validación #1. Fuente: Elaboración propia.

A continuación, se muestran las curvas correspondientes a la fase petróleo:

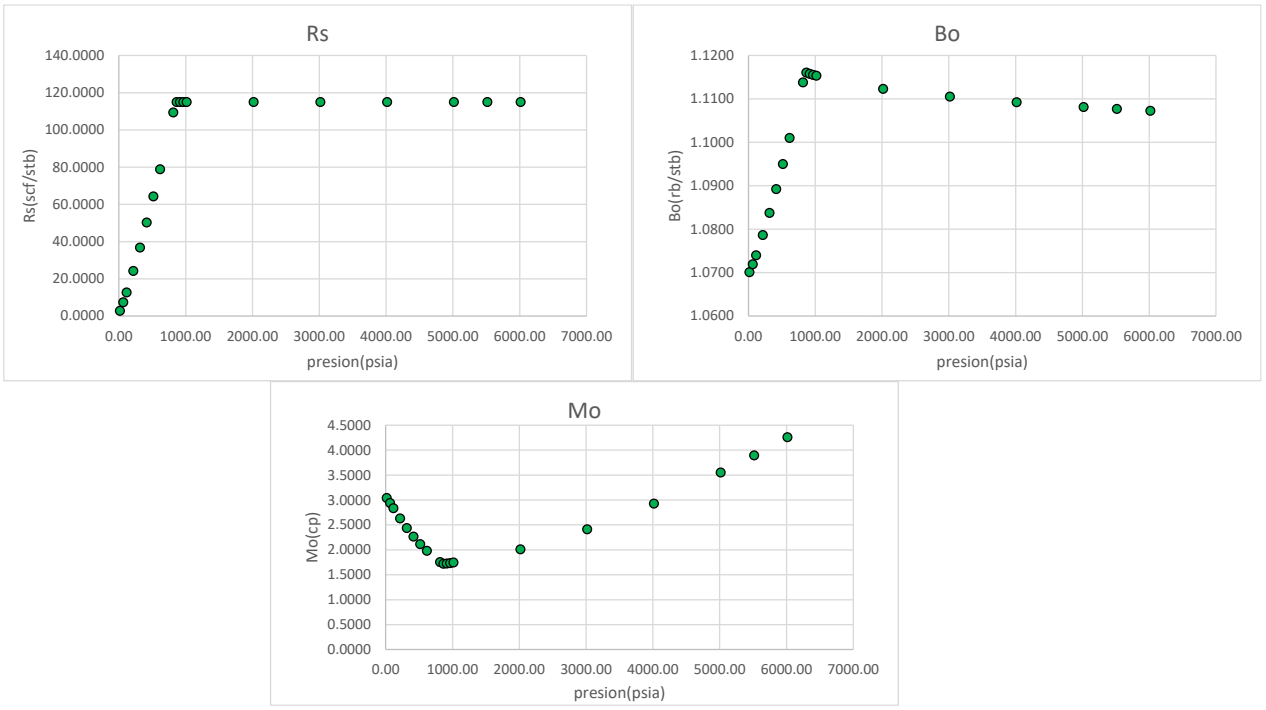


Ilustración 67:Gráfico de Curvas Rs, Bo y Mo de los Datos PVT del caso de Validación #1. Fuente: Elaboración propia.

Posteriormente, se muestran las curvas correspondientes a la fase gas:

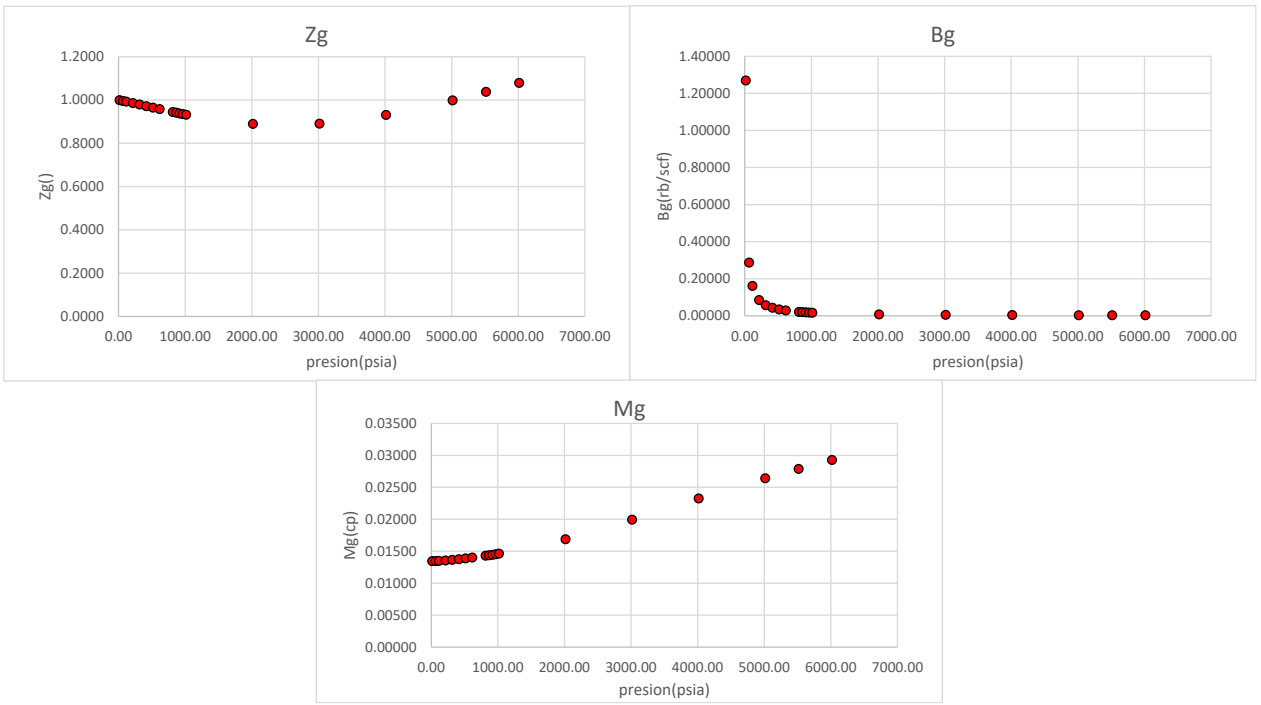


Ilustración 68:Gráfico de Curvas Z, Bg y Mg de los Datos PVT del caso de Validación #1. Fuente: Elaboración propia.

Finalmente, se muestran las curvas correspondientes a la fase agua:

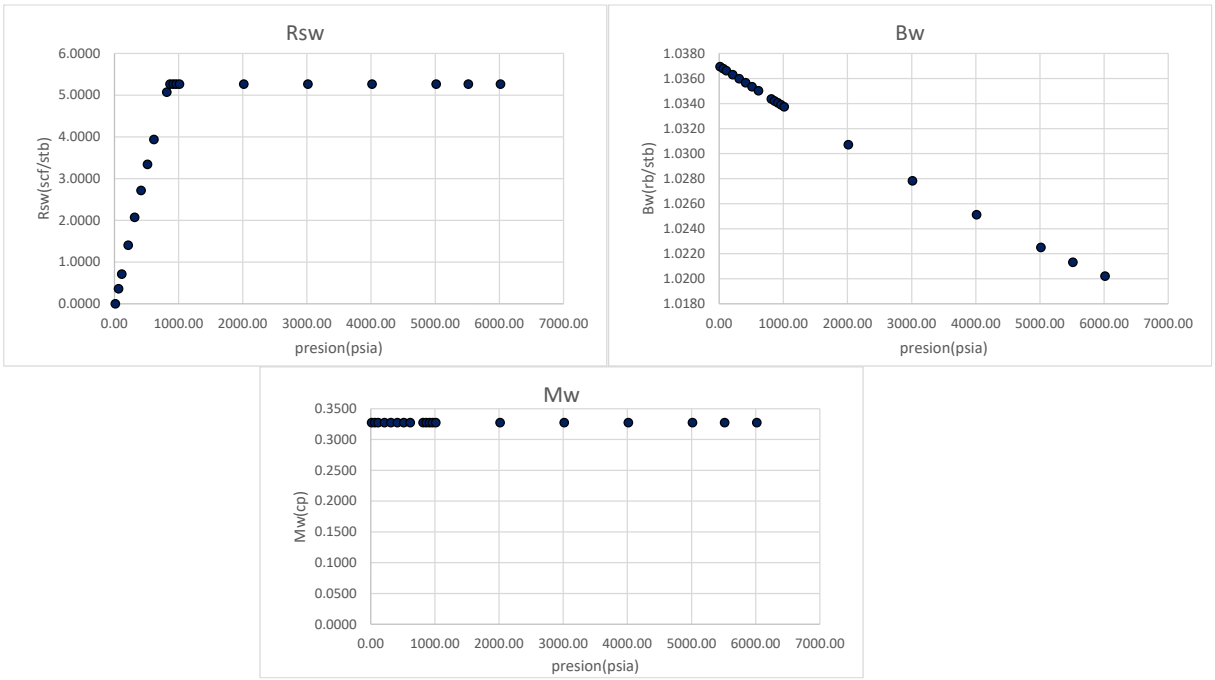


Ilustración 69:Gráfico de Curvas Rsw, Bw y Mw de los Datos PVT del caso de Validación #1. Fuente: Elaboración propia.

Datos de Propiedades Roca Fluido

Se presenta la información relativa a los datos de interacción Roca-Fluido, en el modelo de validación que se resumen en las tablas mostradas a continuación:

Parámetro	Valor	Unidad
Tipo de Roca	Arenisca Limpia	
Humectabilidad de la Roca	Agua	
Tipo de Desplazamiento	Imbibición	
Modelo 3 Fases	STONE II	°

Tabla 11:Tabla resume de Datos Roca Fluido del caso de Validación #1. Fuente: Elaboración propia.

Sw	Krw	Kro	Pcow(psi)
0.20	0.00000	1.00000	0.0000
0.23	0.00075	0.89751	0.0000
0.26	0.00261	0.80055	0.0000
0.29	0.00541	0.70914	0.0000
0.32	0.00908	0.62327	0.0000
0.34	0.01357	0.54294	0.0000
0.37	0.01884	0.46814	0.0000
0.40	0.02486	0.39889	0.0000
0.43	0.03162	0.33518	0.0000
0.46	0.03908	0.27701	0.0000
0.49	0.04724	0.22438	0.0000
0.52	0.05608	0.17729	0.0000
0.55	0.06559	0.13573	0.0000
0.58	0.07576	0.09972	0.0000
0.61	0.08657	0.06925	0.0000
0.63	0.09802	0.04432	0.0000
0.66	0.11009	0.02493	0.0000
0.69	0.12278	0.01108	0.0000
0.72	0.13609	0.00277	0.0000
0.75	0.15000	0.00000	0.0000

Sg	Krg	Kro	Pcgo(psi)
0.00	0.00000	1.00000	0.0000
0.03	0.00083	0.89751	0.0000
0.06	0.00332	0.80055	0.0000
0.09	0.00748	0.70914	0.0000
0.12	0.01330	0.62327	0.0000
0.14	0.02078	0.54294	0.0000
0.17	0.02992	0.46814	0.0000
0.20	0.04072	0.39889	0.0000
0.23	0.05319	0.33518	0.0000
0.26	0.06731	0.27701	0.0000
0.29	0.08310	0.22438	0.0000
0.32	0.10055	0.17729	0.0000
0.35	0.11967	0.13573	0.0000
0.38	0.14044	0.09972	0.0000
0.41	0.16288	0.06925	0.0000
0.43	0.18698	0.04432	0.0000
0.46	0.21274	0.02493	0.0000
0.49	0.24017	0.01108	0.0000
0.52	0.26925	0.00277	0.0000
0.55	0.30000	0.00000	0.0000

Tabla 12:Tablas de Datos Roca Fluido para un Sistema Agua Petróleo (Izq.) y un Sistema gas Petróleo (Der.) del caso de Validación #1. Fuente: Elaboración propia.

Posteriormente, se muestran las curvas correspondientes a los gráficos de permeabilidad relativas y presión capilar para un sistema Agua-Petróleo:

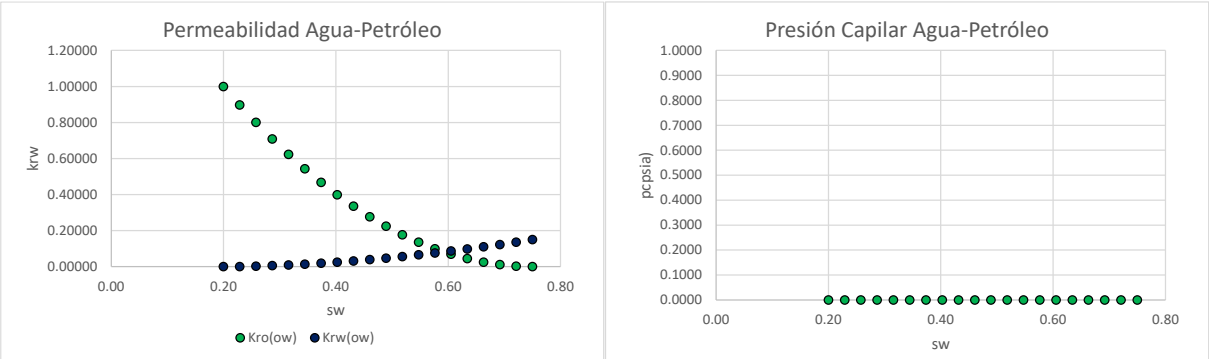


Ilustración 70:Gráfico de Curvas Kro, Krw y Pcow para un sistema Agua Petróleo de los Datos Roca Fluido del caso de Validación #1. Fuente: Elaboración propia.

Finalmente, se muestran las curvas correspondientes a los gráficos de permeabilidad relativas y presión capilar para un sistema Gas-Petróleo:

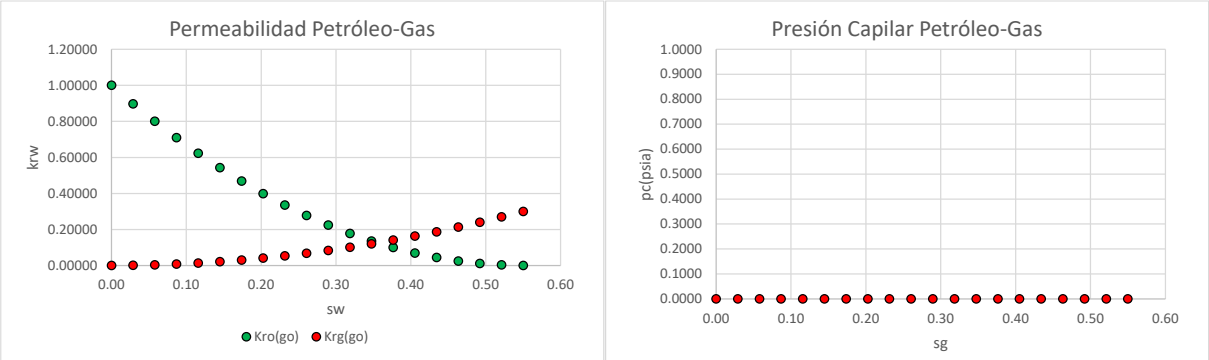


Ilustración 71:Gráfico de Curvas Kro, Krg y Pcog para un sistema Gas Petróleo de los Datos Roca Fluido del caso de Validación #1. Fuente: Elaboración propia.

Condiciones Iniciales

A continuación, se presenta la información relativa a los datos de inicialización del modelo de validación, que se resumen en las tablas mostrada a continuación:

Parámetro	Valor	Unidad
TIPO DE INIALIZACION	EQUILIBRIO	
FASES INICIALES PRESENTES	OIL-WATER-GAS	
DATUM-TVDSS	-7130.7408	ft
OWOC	-7320.0000	ft
OGOC	0.0000	ft
Presión al Datum	5000.0000	psia

Tabla 13:Tabla resume de Datos de Condiciones Iniciales del caso de Validación #1. Fuente: Elaboración propia.

Datos recurrentes

La información de datos recurrente a ser considerada, corresponde a los datos de desviaciones, completamientos y controles de producción, los cuales son mostrados a continuación:

- Pozos

Se presenta la información relativa a los datos de las trayectorias de los pozos en el modelo de validación #1, correspondientes a 5 pozos usados en el modelo:

TABLA DE TRAYECTORIAS					
WELL NAME	X(ft)	Y(ft)	MD(ft)	TVD(ft)	TVDSS(ft)
JARESIM-P1	1250.0000	18750.00000	0.00000	0.0000	880.00
JARESIM-P1	1250.0000	18750.00000	2000.00000	2000.0000	-1120.00
JARESIM-P1	1250.0000	18750.00000	4000.00000	4000.0000	-3120.00
JARESIM-P1	1250.0000	18750.00000	6000.00000	6000.0000	-5120.00
JARESIM-P1	1250.0000	18750.00000	8000.00000	8000.0000	-7120.00
JARESIM-P1	1250.0000	18750.00000	9000.00000	9000.0000	-8120.00
JARESIM-P2	10750.0000	18750.00000	0.00000	0.0000	880.00
JARESIM-P2	10750.0000	18750.00000	2000.00000	2000.0000	-1120.00
JARESIM-P2	10750.0000	18750.00000	4000.00000	4000.0000	-3120.00
JARESIM-P2	10750.0000	18750.00000	6000.00000	6000.0000	-5120.00
JARESIM-P2	10750.0000	18750.00000	8000.00000	8000.0000	-7120.00
JARESIM-P2	10750.0000	18750.00000	9000.00000	9000.0000	-8120.00
JARESIM-P3	6250.0000	33750.00000	0.00000	0.0000	880.00
JARESIM-P3	6250.0000	33750.00000	2000.00000	2000.0000	-1120.00
JARESIM-P3	6250.0000	33750.00000	4000.00000	4000.0000	-3120.00
JARESIM-P3	6250.0000	33750.00000	6000.00000	6000.0000	-5120.00
JARESIM-P3	6250.0000	33750.00000	8000.00000	8000.0000	-7120.00
JARESIM-P3	6250.0000	33750.00000	9000.00000	9000.0000	-8120.00
JARESIM-P4	6250.0000	5250.00000	0.00000	0.0000	880.00
JARESIM-P4	6250.0000	5250.00000	2000.00000	2000.0000	-1120.00
JARESIM-P4	6250.0000	5250.00000	4000.00000	4000.0000	-3120.00
JARESIM-P4	6250.0000	5250.00000	5000.00000	5000.0000	-4120.00
JARESIM-P4	6250.0000	5250.00000	7000.00000	7000.0000	-6120.00
JARESIM-P4	6250.0000	5250.00000	9000.00000	9000.0000	-8120.00
JARESIM-I5	6250.0000	18750.00000	0.00000	0.0000	880.00
JARESIM-I5	6250.0000	18750.00000	2000.00000	2000.0000	-1120.00
JARESIM-I5	6250.0000	18750.00000	4000.00000	4000.0000	-3120.00
JARESIM-I5	6250.0000	18750.00000	6000.00000	6000.0000	-5120.00
JARESIM-I5	6250.0000	18750.00000	8000.00000	8000.0000	-7120.00
JARESIM-I5	6250.0000	18750.00000	9000.00000	9000.0000	-8120.00

Tabla 14:Tablas de Datos de trayectorias de los pozos usados en el caso de Validación #1. Fuente: Elaboración propia.

- Datos de Completamiento

A continuación, se presenta la información relativa a los datos de completamiento de los pozos del modelo de validación, como se muestra en la siguiente tabla, correspondiente a 5 pozos usados en el modelo:

1.TABLA DE DATOS DE COMPLETAMIENTO					
POZO	FECHA	DAÑO	WFRAC	DIAMETRO(PULG)	DIAMETRO (FT)
JARESIM-P1	1/1/2014	-2.5	1.0	8.5	0.3542
JARESIM-P2	1/1/2014	-2.5	1.0	8.5	0.3542
JARESIM-P3	1/1/2014	-2.5	1.0	8.5	0.3542
JARESIM-P4	1/1/2014	-2.5	1.0	8.5	0.3542
JARESIM-I5	1/1/2014	-2.5	1.0	8.5	0.3542

Tabla 15:Tablas de Datos de Completamiento de los pozos en el caso de Validación #1. Fuente: Elaboración propia.

- Controles de Producción, Inyección y Predicción

Se presenta la información relativa a los datos de producción, inyección y predicción de los 5 pozos del modelo de validación, como se muestra en la siguiente tabla a continuación respectivamente:

0.TABLA RESUMEN CONTROLES DE PRODUCCIÓN			
POZO	FECHA	LIQUID_RATE(BFPD)	BHP MIN(PSIA)
JARESIM-P1	1/1/2014	12500.0	250.0
JARESIM-P2	1/1/2014	12500.0	250.0
JARESIM-P3	1/1/2014	12500.0	250.0
JARESIM-P4	1/1/2014	12500.0	250.0

Tabla 16:Tablas de Datos de Controles de Producción de los pozos en el caso de Validación #1. Fuente: Elaboración propia.

1.TABLA RESUMEN CONTROLES DE INYECCIÓN			
POZO	FECHA	WATE RATE(BWPD)	BHP MAX(PSIA)
JARESIM-I5	1/1/2014	51000.0	8100.0

Tabla 17:Tablas de Datos de Controles de Inyección de los pozos en el caso de Validación #1. Fuente: Elaboración propia.

Adicionalmente, se presenta el resumen del periodo de evaluación usado para la generación del caso de validación, correspondiente a las siguientes fechas:

2.TABLA RESUMEN CONTROLES PREDICCION		
FECHA INICIO	FECHA FIN	FRECUENCIA
1/1/2014	1/1/2034	DIARIA

Tabla 18:Tablas de Datos de Controles Predicción en el caso de Validación #1. Fuente: Elaboración propia.

Datos de Configuración de la Sección de métodos numéricos

Finalmente, se presenta la información relativa a la configuración de la sección de métodos numéricos del modelo de validación, como se muestra en la siguiente tabla a continuación respectivamente:

Parámetro	Valor	Unidad
Paso de Tiempo Mínimo (dtmin)	0.00001	días
Paso de Tiempo Máximo (dtmax)	0.50000	días
# Máximo de Pasos de Tiempo ()	30000.00000	
Tolerancia	0.00001	%
# Máximo de Iteraciones	1000.00000	
Máximo Cambio de Presión permitido	20.00000	psia
Máximo Cambio de Saturación permitido	10.000%	%
Método de Interpolación	Lineal	
Método de Resolución de Sistemas Lineales	Algoritmo de Thomas/Newton	

Tabla 19:Tablas de Datos resumen de la Configuración de la Sección de métodos numéricos en el caso de Validación #1. Fuente: Elaboración propia.

10.2. Caso de validación #2

En esta sección, se describirá la información asociada al caso de validación # 2, correspondiente a una malla importada en formato ECLGRD con propiedades totalmente heterogéneas, que incluye 5 pozos dentro del modelo para un periodo de predicción de 10 años. Posteriormente, se podrán observar los resultados obtenidos de esta evaluación realizando una comparación de los resultados del simulador desarrollado en este proyecto y el simulador Comercial IMEX de la Compañía CMG:

10.2.1. Datos de Entrada

Se presenta a continuación la información usada para construir el caso de validación #2 de la aplicación desarrollada en este proyecto:

Datos de Geometría de Modelo

La información relativa a la geometría del modelo de validación usado, se resume en la siguiente tabla:

Parámetro	Valor	Unidad
Dimensión de Celdas X	88.88	ft
Dimensión de Celda Y	53.98	ft
Dimensión de Celda Z	35.69	ft
Origen X	-710.29	ft
Origen Y	-1271.12	ft
Origen Z	-8845.14	ft
Número de Celdas X	15	
Número de Celdas Y	36	
Número de Celdas Z	15	
Tipo de Malla	Importada -Corner Point	

Tabla 20:Tabla resume de Datos de Geometría del caso de Validación #2. Fuente: Elaboración propia.

Posteriormente, se muestra la representación 3D de la malla usada para la realización del caso de validación:

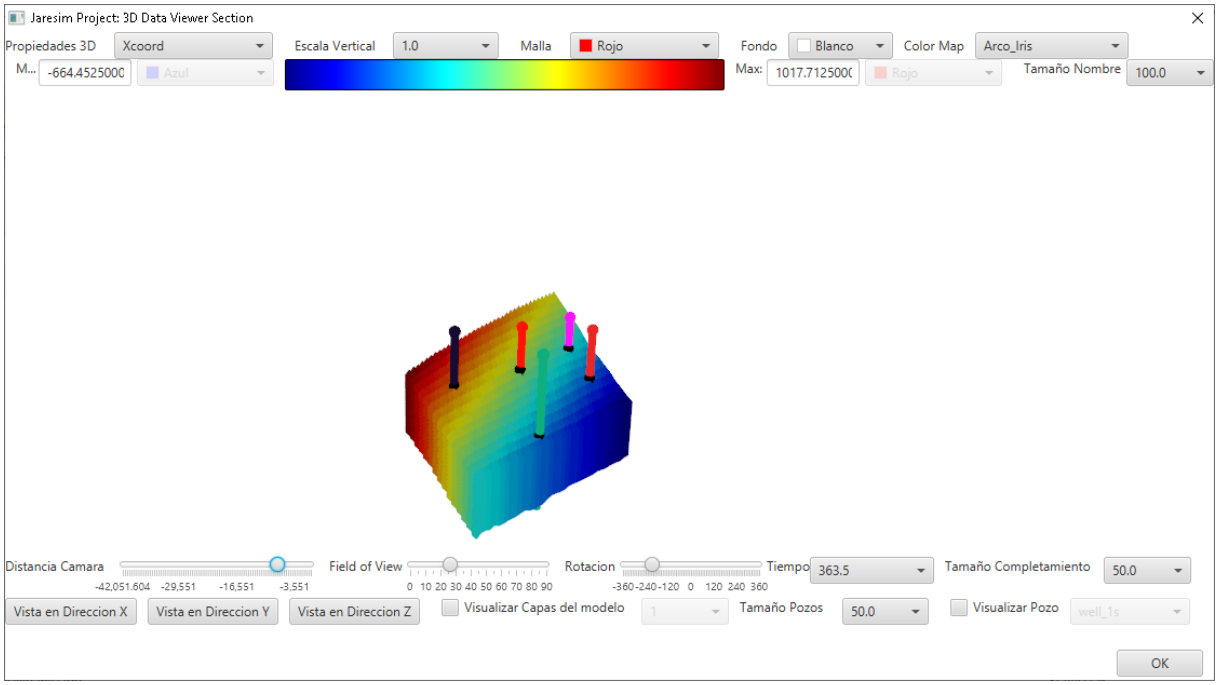


Ilustración 72:Visualización 3D de la Geometría del Modelo del caso de Validación#2 mediante la aplicación desarrollado en este proyecto. Fuente: Elaboración propia.

Datos de Propiedades Petrofísicas

La información relativa a las propiedades petrofísicas del modelo de validación en un modelo totalmente heterogéneo, importado en formato GRDECL donde las propiedades importadas se listan a continuación:

- PHIE: Es la porosidad Efectiva en fracción.
- GROSS: Es el espesor total promedio por celda en pies (ft).
- NTG: Es el Net to gross en fracción.
- Kx: Es la permeabilidad Absoluta en Dirección X en miliDarcies.
- Ky: Es la permeabilidad Absoluta en Dirección Y en miliDarcies.
- Kz: Es la permeabilidad Absoluta en Dirección Z en miliDarcies.

A continuación, se muestra la representación 3D de la propiedad Porosidad, Permeabilidad en X, Permeabilidad en Y, y Permeabilidad en Z usadas para la realización del caso de validación:

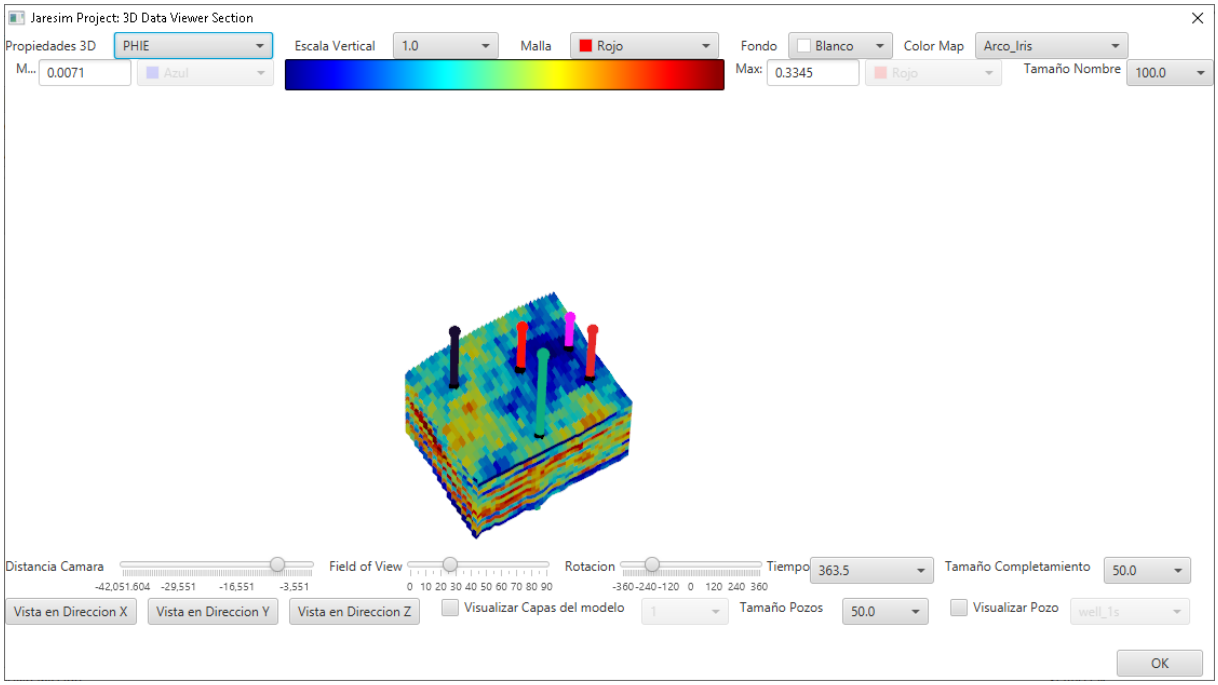


Ilustración 73:Visualización 3D de la Porosidad usada del Modelo del caso de Validación#2 mediante la aplicación desarrollado en este proyecto. Fuente: Elaboración propia.

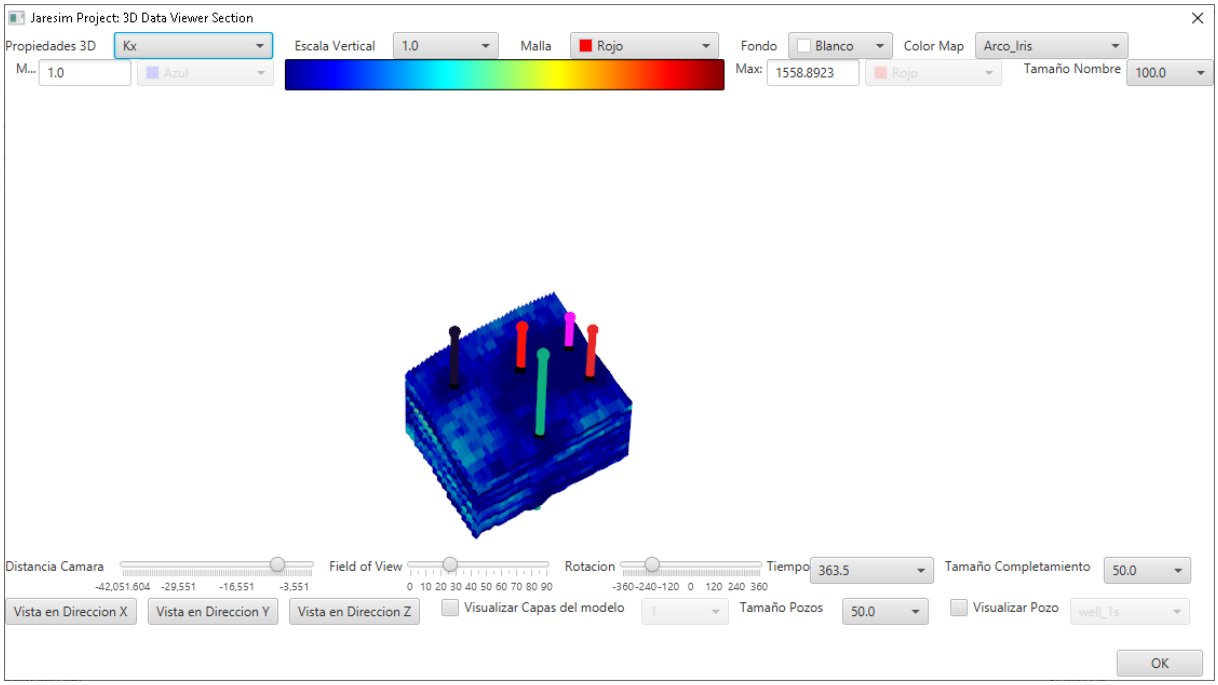


Ilustración 74:Visualización 3D de la Permeabilidad en X usada del Modelo del caso de Validación#2 mediante la aplicación desarrollado en este proyecto. Fuente: Elaboración propia.

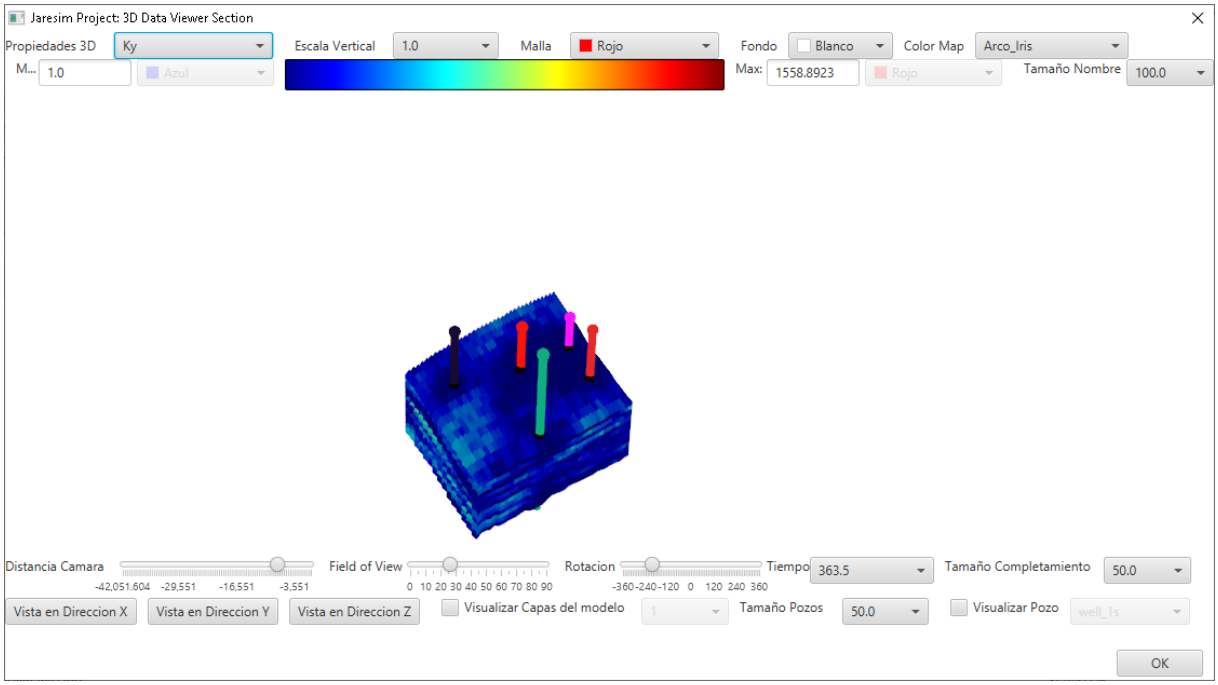


Ilustración 75:Visualización 3D de la Permeabilidad en Y usada del Modelo del caso de Validación#2 mediante la aplicación desarrollado en este proyecto. Fuente: Elaboración propia.

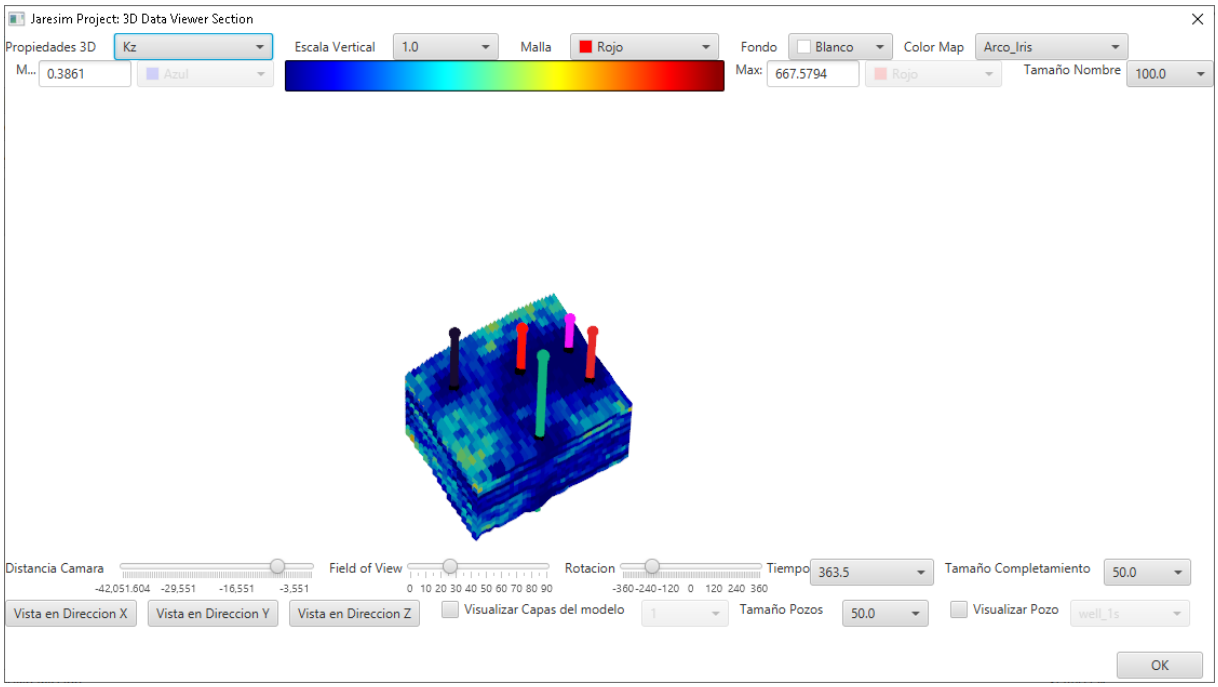


Ilustración 76:Visualización 3D de la Permeabilidad en Y usada del Modelo del caso de Validación#2 mediante la aplicación desarrollado en este proyecto. Fuente: Elaboración propia.

Aunado a lo anterior, se presenta la información relativa a compresibilidad de la roca y cambio del volumen poroso en función de la presión del yacimiento, usado en el modelo de validación que se resume en la tabla mostrada a continuación:

P(psia)	Cr (psi-1)
14.7	4.342E-06
150	4.396E-06
250	4.437E-06
500	4.542E-06
750	4.651E-06
1000	4.765E-06
1500	5.006E-06
2000	5.268E-06
2500	5.556E-06
3000	5.874E-06
3500	6.228E-06
4000	6.626E-06
4500	7.080E-06
5000	7.607E-06
5500	8.229E-06
6000	8.984E-06

Tabla 21:Tabla resume de Datos de Compresibilidad de Roca del caso de Validación #2. Fuente: Elaboración propia.

Finalmente, se presenta la curva presión vs compresibilidad de la roca, usada para la realización del caso de validación:

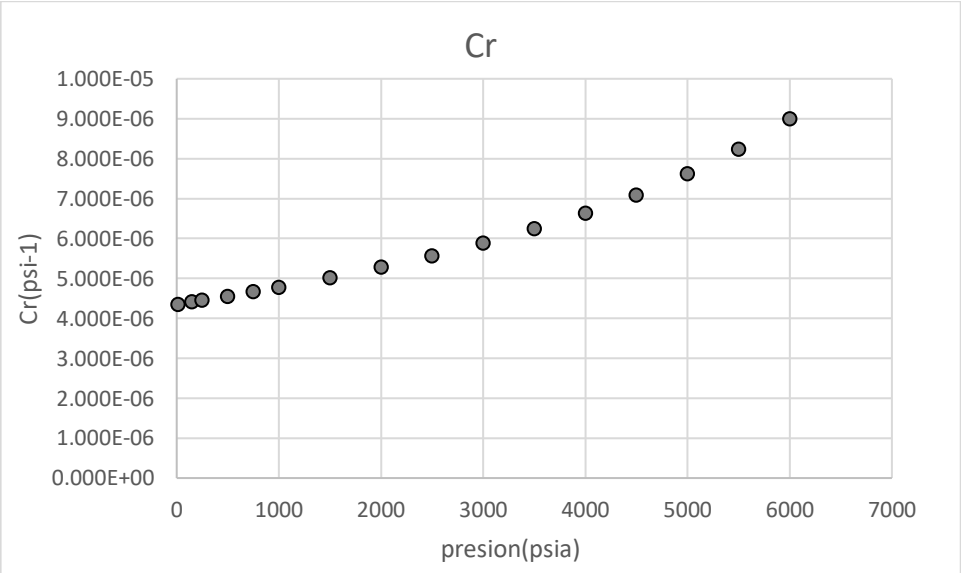


Ilustración 77:Gráfico de Curva P vs Cr de las propiedades Petrofísicas del caso de Validación#2. Fuente: Elaboración propia.

Datos de Propiedades PVT

La presente información es relativa a la data PVT del modelo de fluido, usado en el modelo de validación que se resume en las tablas mostradas a continuación:

Parámetro	Valor	Unidad
MODELO DE FLUIDO	BLACK OIL	
PROCESO ISOTERMICO	SI	
FASES PRESENTES	OIL-GAS-WATER	
API	28.0000	°
Gas Gravity	0.6800	
Rsb	114.9990	SCF/STB
Pb	850.0000	psia
T yac	199.1665	°F
Den Oil	55.3580	lb/ft3
Den Water	62.3397	lb/ft3
Den Gas	15.5167	lb/ft3

Tabla 22:Tabla resume de Datos PVT del caso de Validación #2. Fuente: Elaboración propia.

Pressure (psia)	Z ()	Bg (cf/scf)	Mug (cp)	Rs (cf/bbl)	Bo (B/STB)	Muo (cp)	Rsw (cf/bbl)	Bw (B/STB)	Muw (cp)
14.70	0.9990	1.26927	0.01343	2.8082	1.0701	3.0444	0.0000	1.0370	0.3275
64.70	0.9957	0.28736	0.01345	7.4370	1.0719	2.9427	0.3592	1.0368	0.3275
114.70	0.9923	0.16154	0.01348	12.6532	1.0740	2.8368	0.7128	1.0366	0.3275
214.70	0.9855	0.08570	0.01355	24.2070	1.0787	2.6302	1.4032	1.0363	0.3275
314.70	0.9785	0.05806	0.01364	36.8191	1.0838	2.4402	2.0712	1.0360	0.3275
414.70	0.9716	0.04374	0.01374	50.2246	1.0892	2.2695	2.7167	1.0357	0.3275
514.70	0.9646	0.03499	0.01386	64.2737	1.0950	2.1174	3.3397	1.0353	0.3275
614.70	0.9578	0.02909	0.01399	78.8687	1.1010	1.9822	3.9403	1.0350	0.3275
814.70	0.9444	0.02164	0.01427	109.4351	1.1138	1.7544	5.0742	1.0344	0.3275
864.70	0.9412	0.02032	0.01435	114.9990	1.1161	1.7214	5.2650	1.0342	0.3275
914.70	0.9380	0.01915	0.01443	114.9990	1.1158	1.7292	5.2650	1.0341	0.3275
964.70	0.9349	0.01809	0.01452	114.9990	1.1156	1.7375	5.2650	1.0339	0.3275
1014.70	0.9318	0.01715	0.01461	114.9990	1.1153	1.7462	5.2650	1.0337	0.3275
2014.70	0.8892	0.00824	0.01687	114.9990	1.1123	2.0105	5.2650	1.0307	0.3275
3014.70	0.8903	0.00551	0.01993	114.9990	1.1105	2.4121	5.2650	1.0278	0.3275
4014.70	0.9313	0.00433	0.02325	114.9990	1.1092	2.9306	5.2650	1.0251	0.3275
5014.70	0.9984	0.00372	0.02640	114.9990	1.1082	3.5527	5.2650	1.0225	0.3275
5514.70	1.0376	0.00351	0.02787	114.9990	1.1077	3.8976	5.2650	1.0213	0.3275
6014.70	1.0789	0.00335	0.02928	114.9990	1.1073	4.2616	5.2650	1.0202	0.3275

Tabla 23:Continuación Tabla resume de Datos PVT del caso de Validación #2. Fuente: Elaboración propia.

A continuación, se muestran las curvas correspondientes a la fase petróleo:

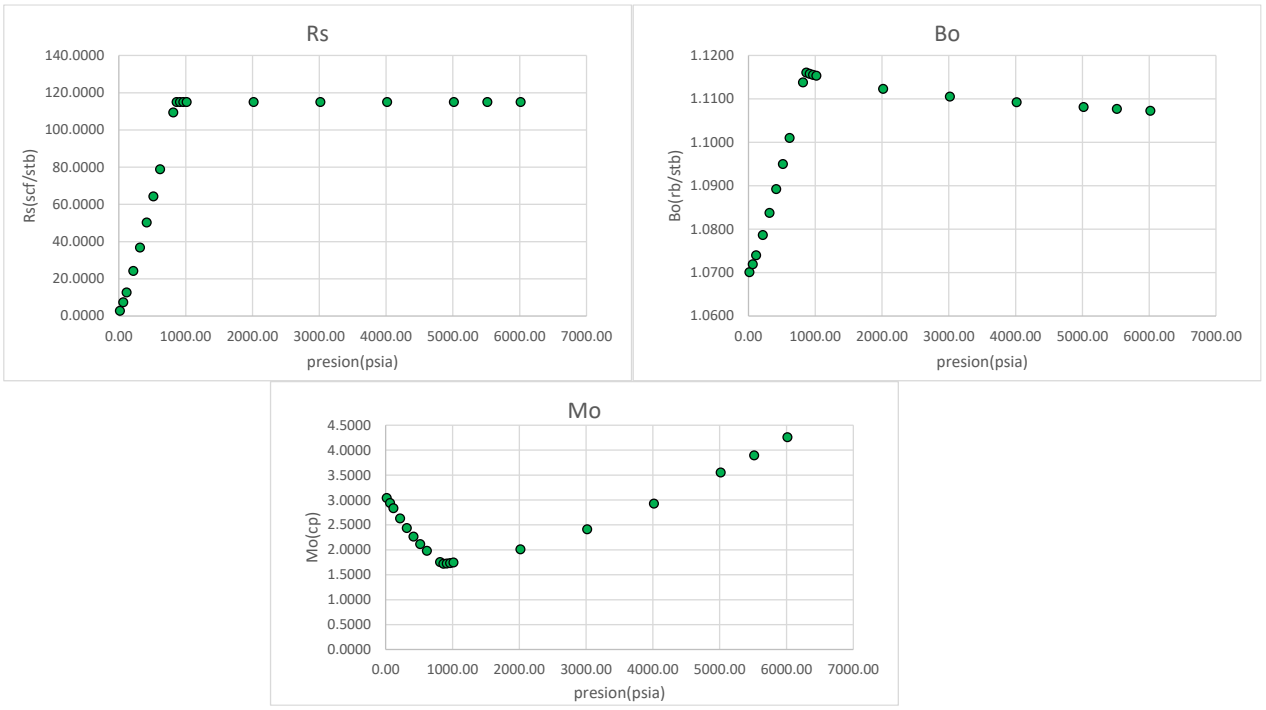


Ilustración 78:Gráfico de Curvas Rs, Bo y Mo de los Datos PVT del caso de Validación #2. Fuente: Elaboración propia.

Posteriormente, se muestran las curvas correspondientes a la fase gas:

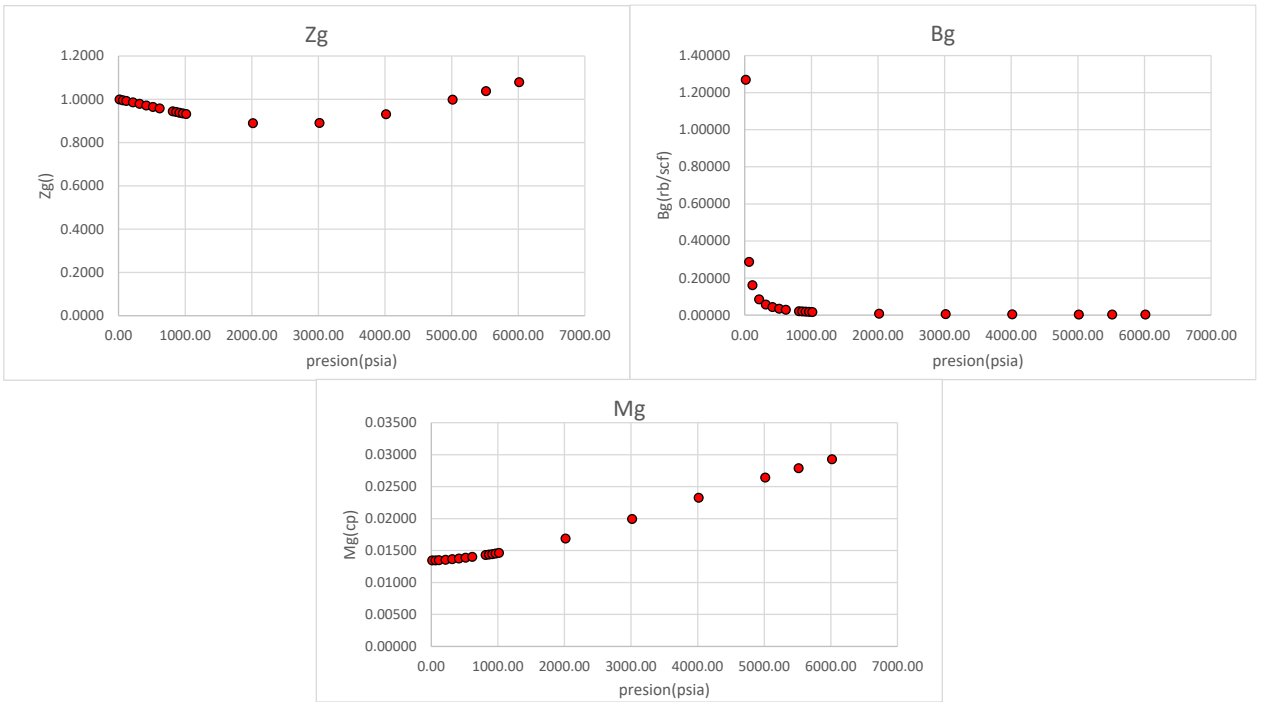


Ilustración 79:Gráfico de Curvas Z, Bg y Mg de los Datos PVT del caso de Validación #2. Fuente: Elaboración propia.

Finalmente, se muestran las curvas correspondientes a la fase agua:

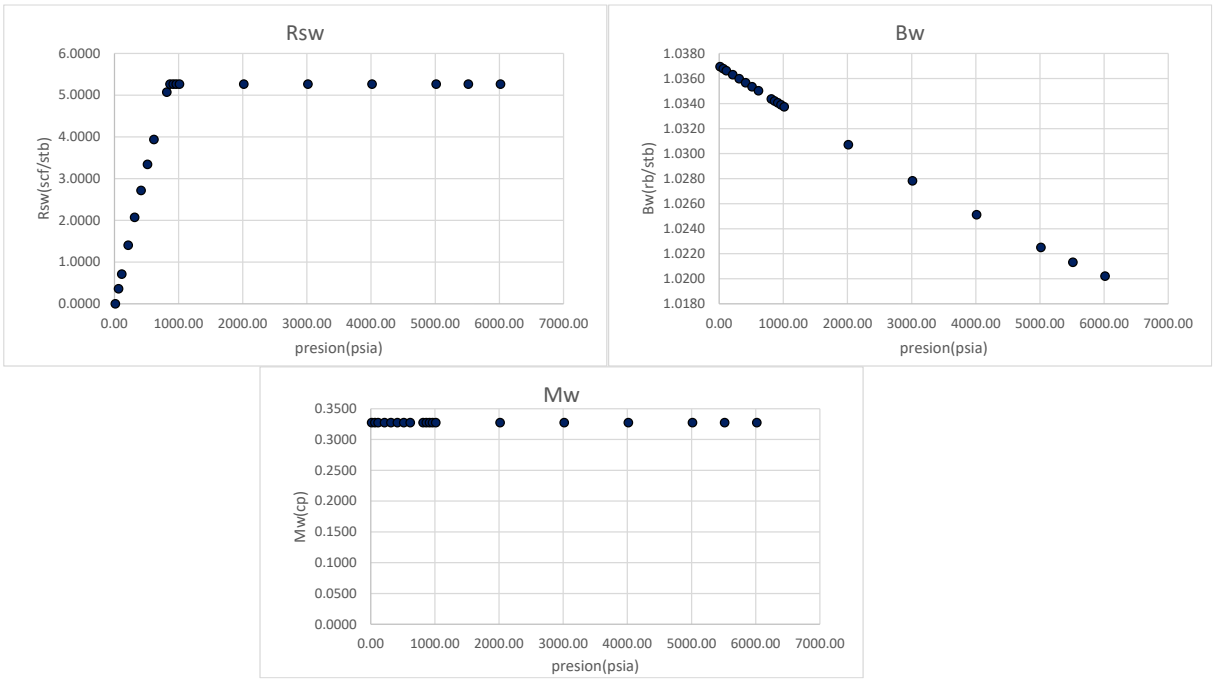


Ilustración 80:Gráfico de Curvas Rsw, Bw y Mw de los Datos PVT del caso de Validación #2. Fuente: Elaboración propia.

Datos de Propiedades Roca Fluido

Se presenta la información relativa a los datos de interacción Roca-Fluido, en el modelo de validación que se resumen en las tablas mostradas a continuación:

Parámetro	Valor	Unidad
Tipo de Roca	Arenisca Limpia	
Humectabilidad de la Roca	Agua	
Tipo de Desplazamiento	Imbibición	
Modelo 3 Fases	STONE II	°

Tabla 24:Tabla resume de Datos Roca Fluido del caso de Validación #2. Fuente: Elaboración propia.

Sw	Krw	Kro	Pcow(psi)
0.20	0.00000	1.00000	0.0000
0.23	0.00075	0.89751	0.0000
0.26	0.00261	0.80055	0.0000
0.29	0.00541	0.70914	0.0000
0.32	0.00908	0.62327	0.0000
0.34	0.01357	0.54294	0.0000
0.37	0.01884	0.46814	0.0000
0.40	0.02486	0.39889	0.0000
0.43	0.03162	0.33518	0.0000
0.46	0.03908	0.27701	0.0000
0.49	0.04724	0.22438	0.0000
0.52	0.05608	0.17729	0.0000
0.55	0.06559	0.13573	0.0000
0.58	0.07576	0.09972	0.0000
0.61	0.08657	0.06925	0.0000
0.63	0.09802	0.04432	0.0000
0.66	0.11009	0.02493	0.0000
0.69	0.12278	0.01108	0.0000
0.72	0.13609	0.00277	0.0000
0.75	0.15000	0.00000	0.0000

Sg	Krg	Kro	Pcgo(psi)
0.00	0.00000	1.00000	0.0000
0.03	0.00083	0.89751	0.0000
0.06	0.00332	0.80055	0.0000
0.09	0.00748	0.70914	0.0000
0.12	0.01330	0.62327	0.0000
0.14	0.02078	0.54294	0.0000
0.17	0.02992	0.46814	0.0000
0.20	0.04072	0.39889	0.0000
0.23	0.05319	0.33518	0.0000
0.26	0.06731	0.27701	0.0000
0.29	0.08310	0.22438	0.0000
0.32	0.10055	0.17729	0.0000
0.35	0.11967	0.13573	0.0000
0.38	0.14044	0.09972	0.0000
0.41	0.16288	0.06925	0.0000
0.43	0.18698	0.04432	0.0000
0.46	0.21274	0.02493	0.0000
0.49	0.24017	0.01108	0.0000
0.52	0.26925	0.00277	0.0000
0.55	0.30000	0.00000	0.0000

Tabla 25:Tablas de Datos Roca Fluido para un Sistema Agua Petróleo (Izq.) y un Sistema gas Petróleo (Der.) del caso de Validación #1. Fuente: Elaboración propia.

A continuación, se muestran las curvas correspondientes a los gráficos de permeabilidad relativas y presión capilar para un sistema Agua-Petróleo:

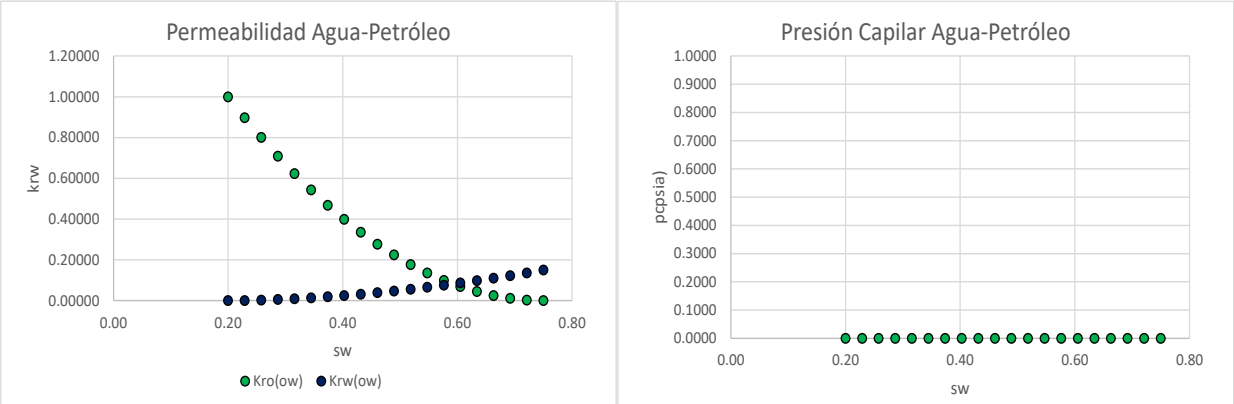


Ilustración 81:Gráfico de Curvas Kro, Krw y Pcow para un sistema Agua Petróleo de los Datos Roca Fluido del caso de Validación #1. Fuente: Elaboración propia.

Finalmente, se muestran las curvas correspondientes a los gráficos de permeabilidad relativas y presión capilar para un sistema Gas-Petróleo:

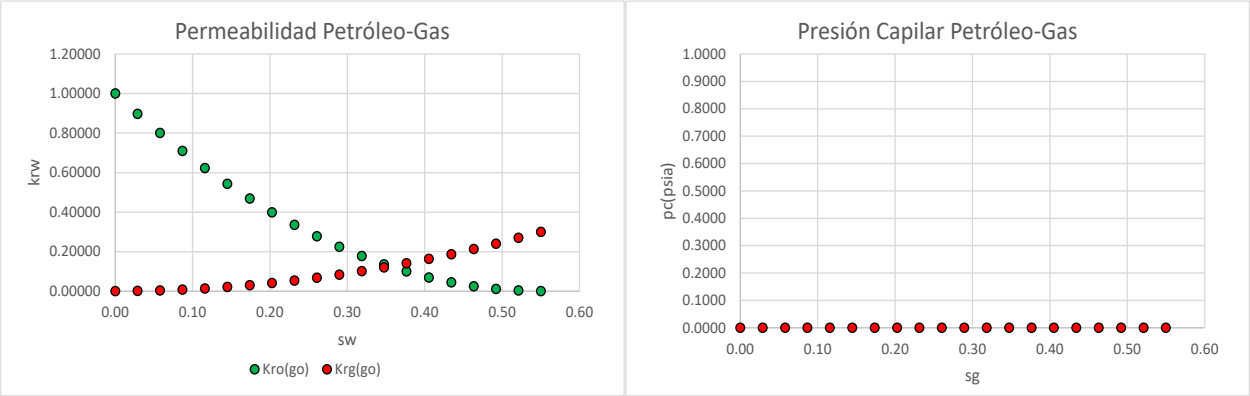


Ilustración 82:Gráfico de Curvas Kro, Krg y Pcog para un sistema Gas Petróleo de los Datos Roca Fluido del caso de Validación #2. Fuente: Elaboración propia.

Condiciones Iniciales

Posteriormente, se presenta la información relativa a los datos de inicialización del modelo de validación que se resumen en la tabla mostrada a continuación:

Parámetro	Valor	Unidad
TIPO DE INIALIZACION	EQUILIBRIO	
FASES INICIALES PRESENTES	OIL-WATER-GAS	
DATUM-TVDSS	-10000.0000	ft
OWOC	-11000.0000	ft
OGOC	0.0000	ft
Presión al Datum	5100.0000	psia

Tabla 26:Tabla resume de Datos de Condiciones Iniciales del caso de Validación #2. Fuente: Elaboración propia.

Datos recurrentes

La información de datos recurrente a ser considerada corresponde a los datos de desviaciones, completamientos y controles de producción, los cuales son mostrados a continuación:

- **Pozos**

Se presenta la información relativa a los datos de las trayectorias de los pozos del modelo de validación #2, correspondientes a 5 pozos usados en el modelo:

1.TABLA DE TRAYECTORIAS					
POZO	X(ft)	Y(ft)	MD(ft)	TVD(ft)	TVDSS(ft)
well_1s	-213.2300	-582.20000	9529.61000	9529.6100	-9529.61
well_1s	-213.2300	-582.20000	10221.07000	10221.0700	-10221.07
well_2s	-4.2800	-1433.93000	9114.56000	9114.5600	-9114.56
well_2s	-4.2800	-1433.93000	10221.07000	10221.0700	-10221.07
well_3s	614.3200	-597.03000	9307.06000	9307.0600	-9307.06
well_3s	614.3200	-597.03000	10221.07000	10221.0700	-10221.07
well_4s	217.2600	-997.93000	9213.36000	9213.3600	-9213.36
well_4s	217.2600	-997.93000	10221.07000	10221.0700	-10221.07
well_5s	-297.3100	-1279.55000	9267.05000	9267.0500	-9267.05
well_5s	-297.3100	-1279.55000	10221.07000	10221.0700	-10221.07

Tabla 27:Tablas de Datos de trayectorias de los pozos usados en el caso de Validación #2. Fuente: Elaboración propia.

- **Datos de Completamiento**

Posteriormente, se presenta la información relativa a los datos de completamiento de los pozos del modelo de validación como se muestra en la siguiente tabla a continuación corresponde a los 5 pozos usados en el modelo:

1.TABLA DE DATOS DE COMPLETAMIENTO					
POZO	FECHA	DAÑO	WFRAC	DIAMETRO(PULG)	DIAMETRO (FT)
well-1s	5/3/2020	-0.5	1.0	8.5	0.3542
well-2s	5/3/2020	-0.5	1.0	8.5	0.3542
well-3s	5/3/2020	-0.5	1.0	8.5	0.3542
well-4s	5/3/2020	-0.5	1.0	8.5	0.3542
well-5s	5/3/2020	-0.5	1.0	8.5	0.3542

Tabla 28:Tablas de Datos de Completamiento de los pozos en el caso de Validación #2. Fuente: Elaboración propia.

- **Controles de Producción, Inyección y Predicción**

Aunado a lo anterior, se presenta la información relativa a los datos de producción, inyección y predicción de los 5 pozos del modelo de validación, como se muestra en la siguiente tabla respectivamente:

0.TABLA RESUMEN CONTROLES DE PRODUCCIÓN			
POZO	FECHA	LIQUID_RATE(BFPD)	BHP MIN(PSIA)
well-1s	5/3/2020	2500.0	250.0
well-2s	5/3/2020	2500.0	250.0
well-3s	5/3/2020	2500.0	250.0
well-5s	5/3/2020	2500.0	250.0

Tabla 29:Tablas de Datos de Controles de Producción de los pozos en el caso de Validación #2. Fuente: Elaboración propia.

1.TABLA RESUMEN CONTROLES DE INYECCIÓN			
POZO	FECHA	WATE RATE(BWPD)	BHP MAX(PSIA)
well-4s	5/3/2020	10100.0	8100.0

Tabla 30:Tablas de Datos de Controles de Inyección de los pozos en el caso de Validación #2. Fuente: Elaboración propia.

Adicionalmente, se presenta el resumen del periodo de evaluación usado para la generación del caso de validación correspondiente a las siguientes fechas:

2.TABLA RESUMEN CONTROLES PREDICCION		
FECHA INICIO	FECHA FIN	FRECUENCIA
5/3/2020	1/3/2020	DIARIA

Tabla 31Tablas de Datos de Controles Predicción en el caso de Validación #2. Fuente: Elaboración propia.

Datos de Configuración de la Sección de métodos numéricos

Finalmente, se presenta la información relativa a la configuración de la sección de métodos numéricos del modelo de validación, como se muestra en la siguiente tabla a continuación respectivamente:

Parámetro	Valor	Unidad
Paso de Tiempo Mínimo (dtmin)	0.00001	días
Paso de Tiempo Máximo (dtmax)	0.50000	días
# Máximo de Pasos de Tiempo ()	30000.00000	
Tolerancia	0.00100	%
# Máximo de Iteraciones	1000.00000	
Máximo Cambio de Presión permitido	20.00000	psia
Máximo Cambio de Saturación permitido	10%	%
Método de Interpolación	Lineal	
Método de Resolución de Sistemas Lineales	Algoritmo de Thomas/Newton	

Tabla 32:Tablas de Datos resumen de la Configuración de la Sección de métodos numéricos en el caso de Validación #2. Fuente: Elaboración propia.

10.3. Resultados Obtenidos para el Caso de Validación #1

A continuación, se muestran los resultados obtenidos para el software comercial IMEX de la compañía CMG, y los resultados generados por el simulador propuesto por esta investigación:

10.3.1. Caso Simulador Comercial IMEX

Las ilustraciones presentadas a continuación muestran los resultados correspondientes a las curvas de producción de Petróleo, Agua y Gas de los 4 pozos productores, así como la tasa de Inyección, producción de Petróleo, Agua, Gas y Presión promedio a escala de Campo para caso de validación #1, generados en el simulador Comercial IMEX de la Compañía Computer Modeling Group:

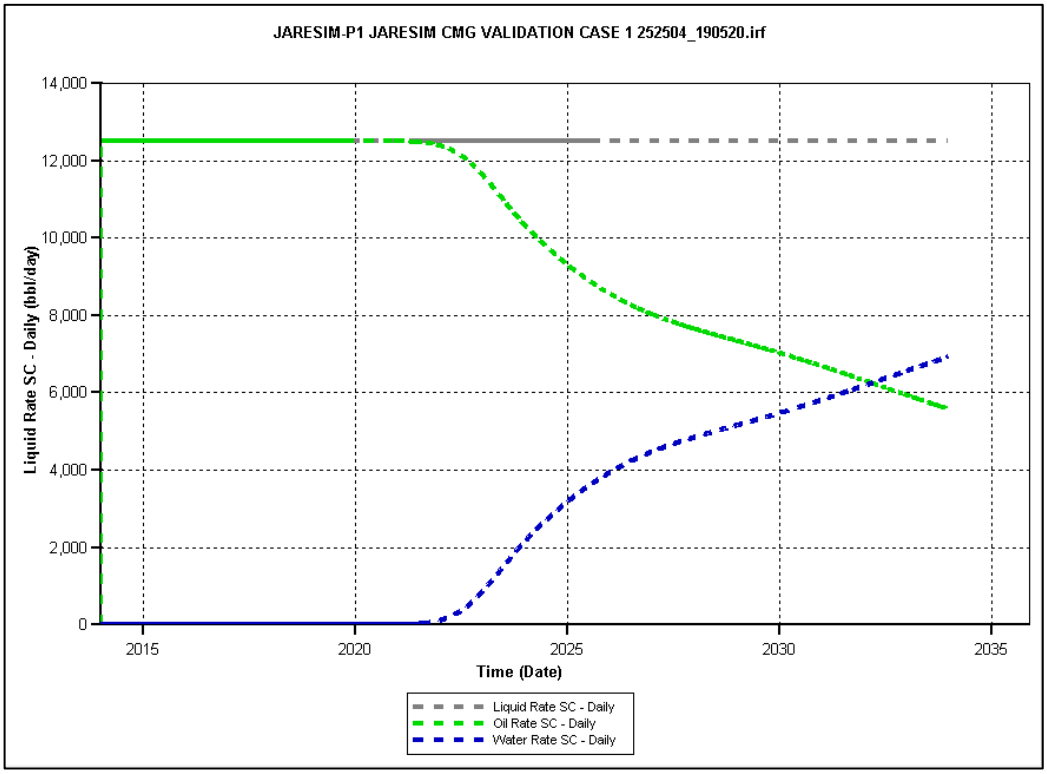


Ilustración 83:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P1 del caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

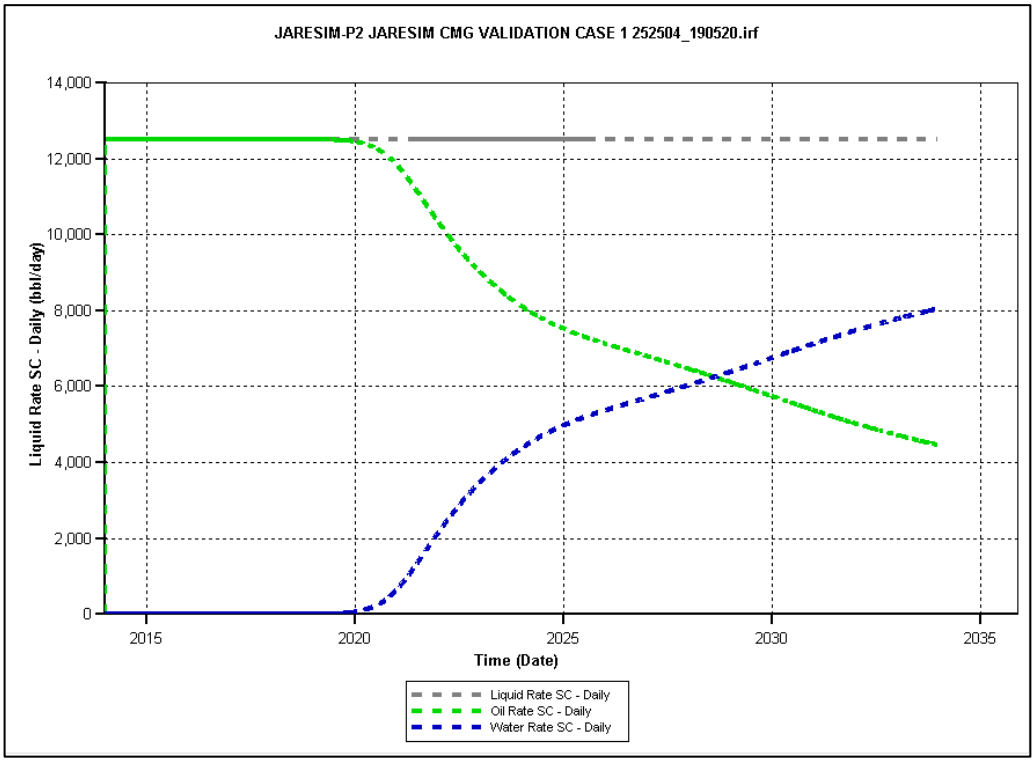


Ilustración 84:Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P2 del caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

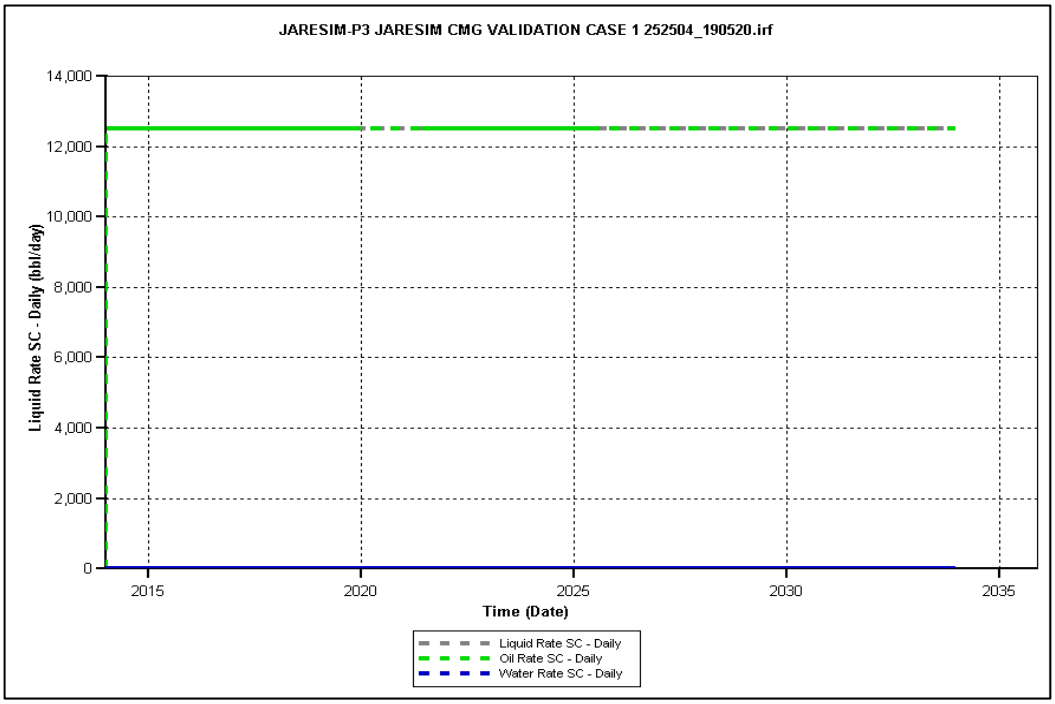


Ilustración 85: Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P3 del caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

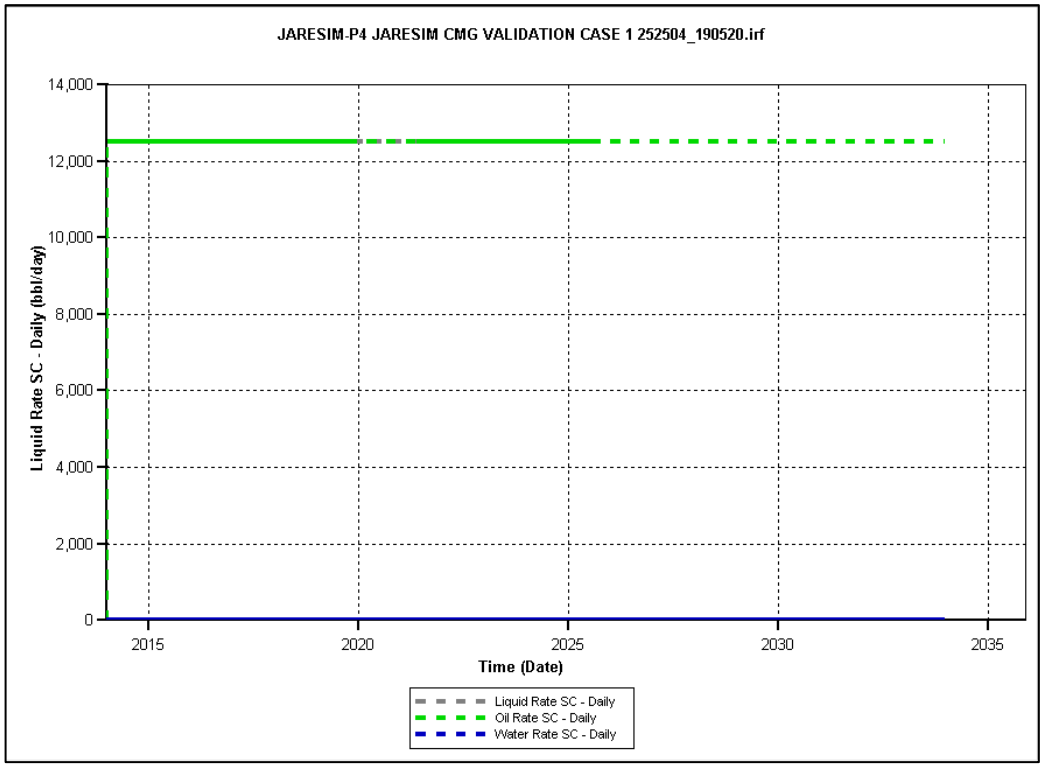


Ilustración 86:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P4 del caso de Validación #1 mediante el simulador Comercial IMEX. . Fuente: CMG IMEX 2019.

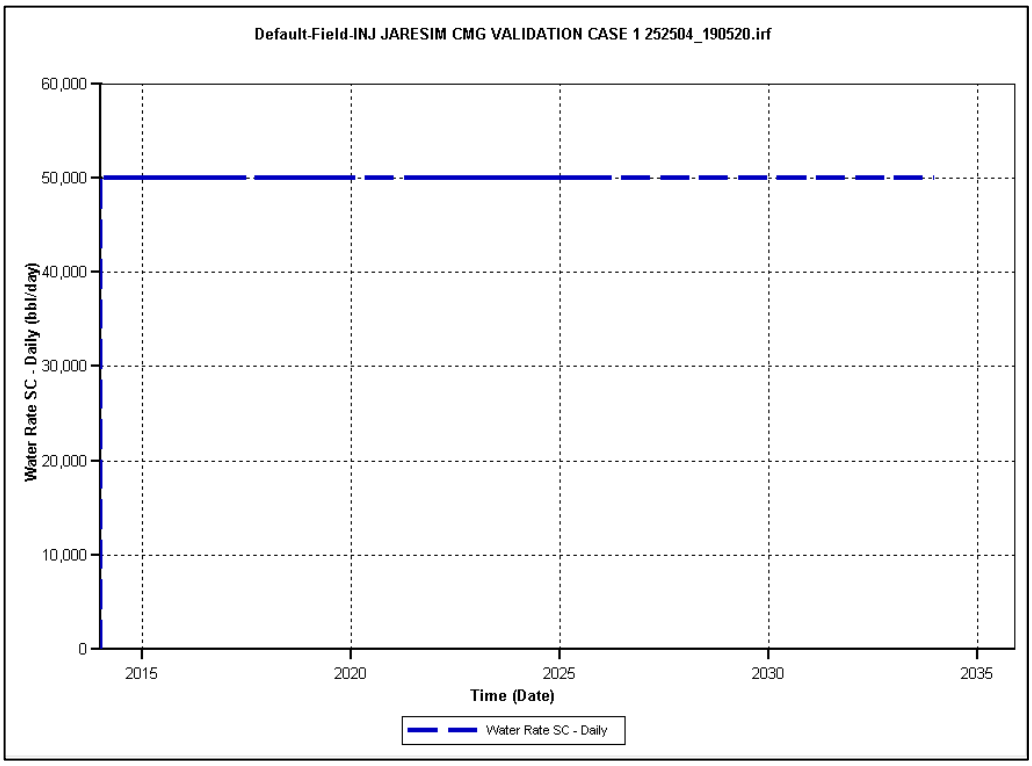


Ilustración 87:Gráfico de Curva de Tasa de Inyección de Agua (Azul) a nivel del Campo para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

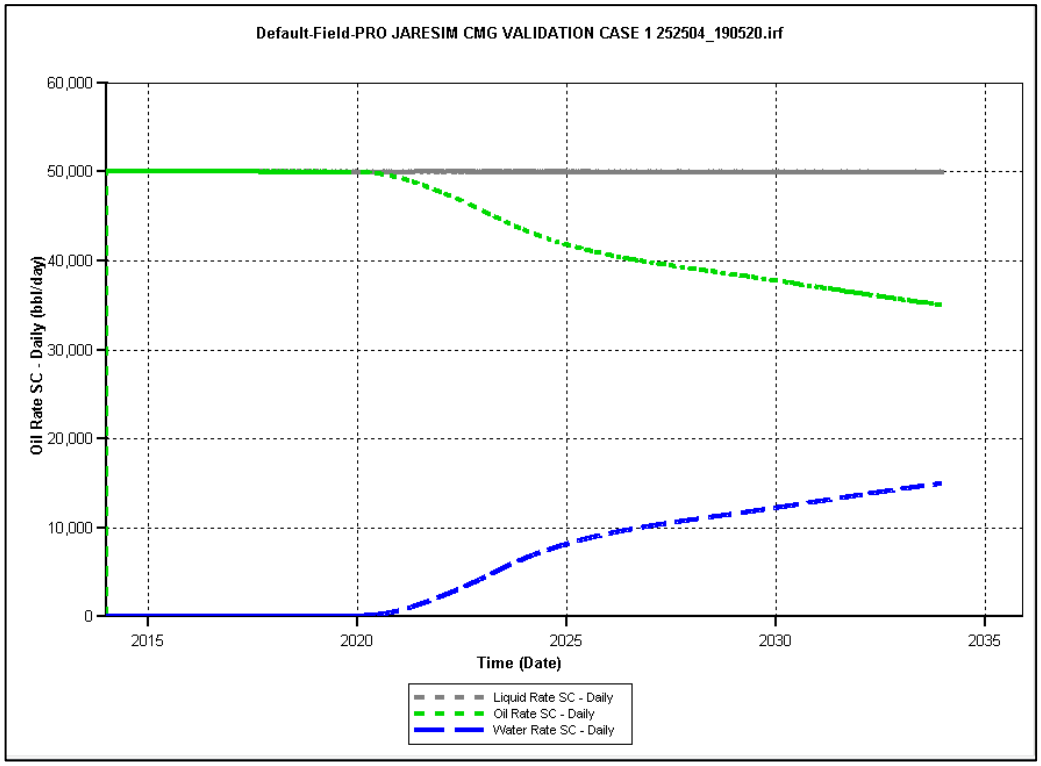


Ilustración 88:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) a escala de Campo para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

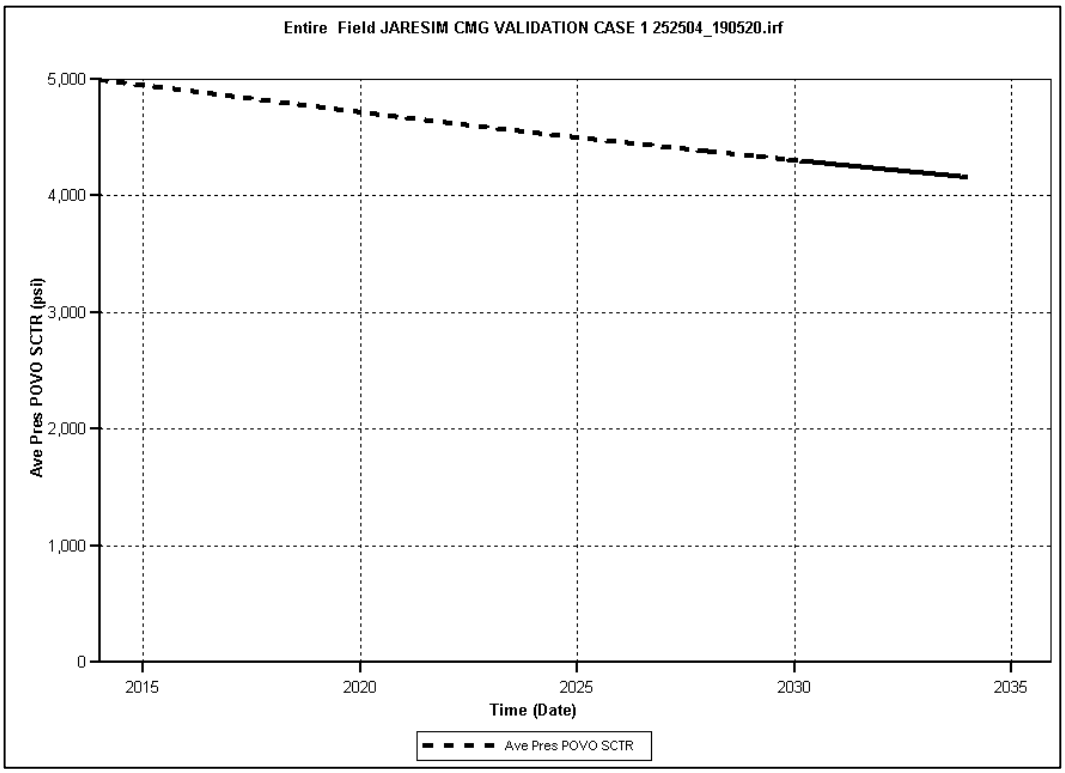


Ilustración 89:Gráfico de presión promedio (Negro) a escala de Campo para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

A continuación, se muestra vista 3D de la variable saturación de agua para los siguientes estados temporales:

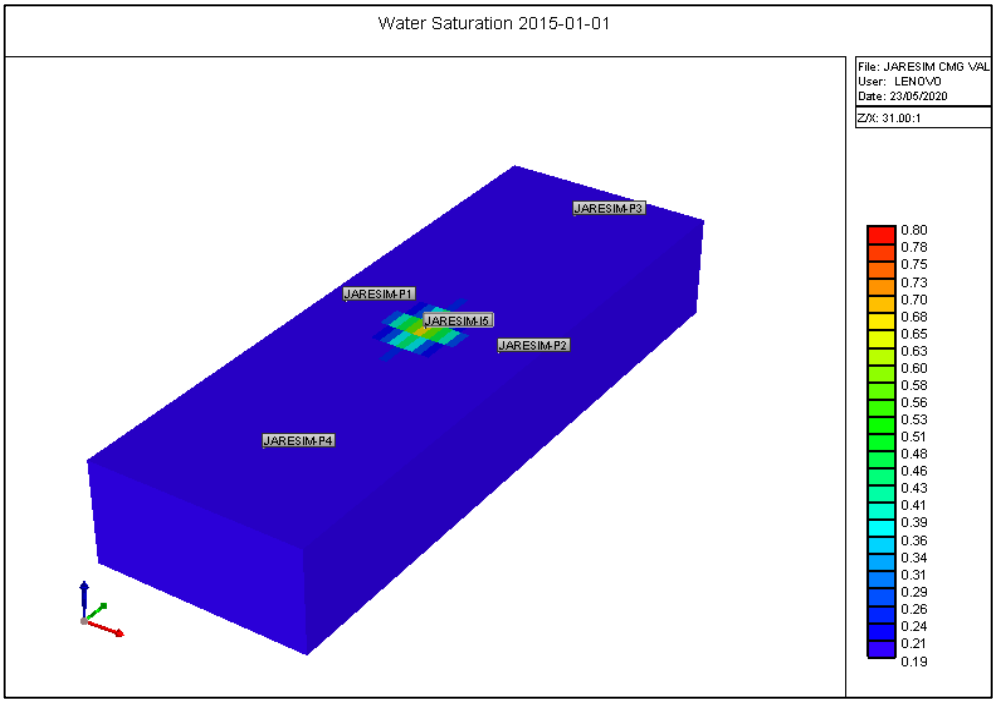


Ilustración 90:Gráfico de Distribución 3D de Saturación de Agua a los 365 días de Simulación para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

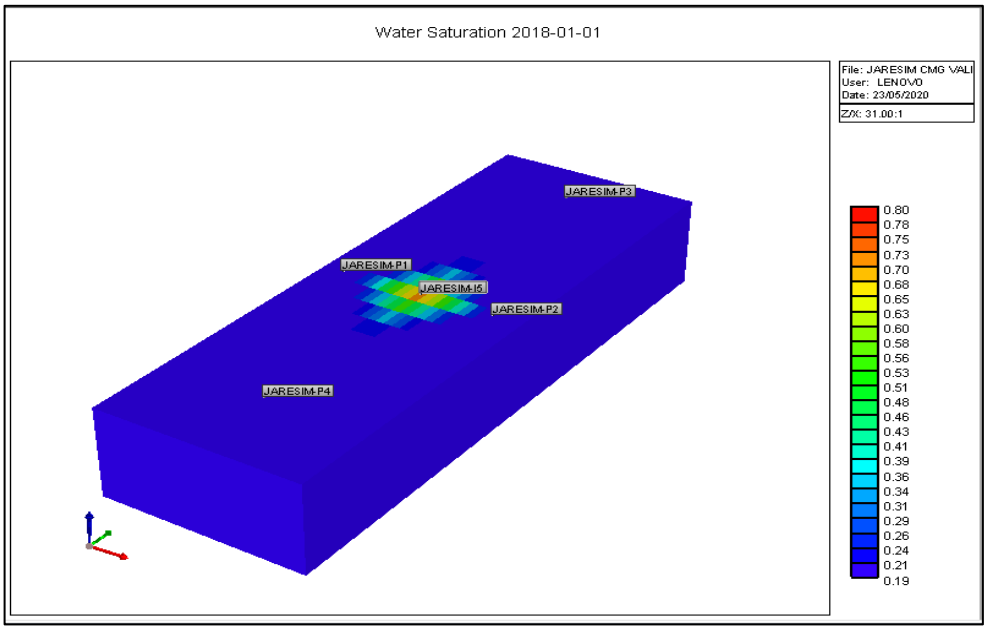


Ilustración 91:Gráfico de Distribución 3D de Saturación de Agua a los 1460 días de Simulación para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

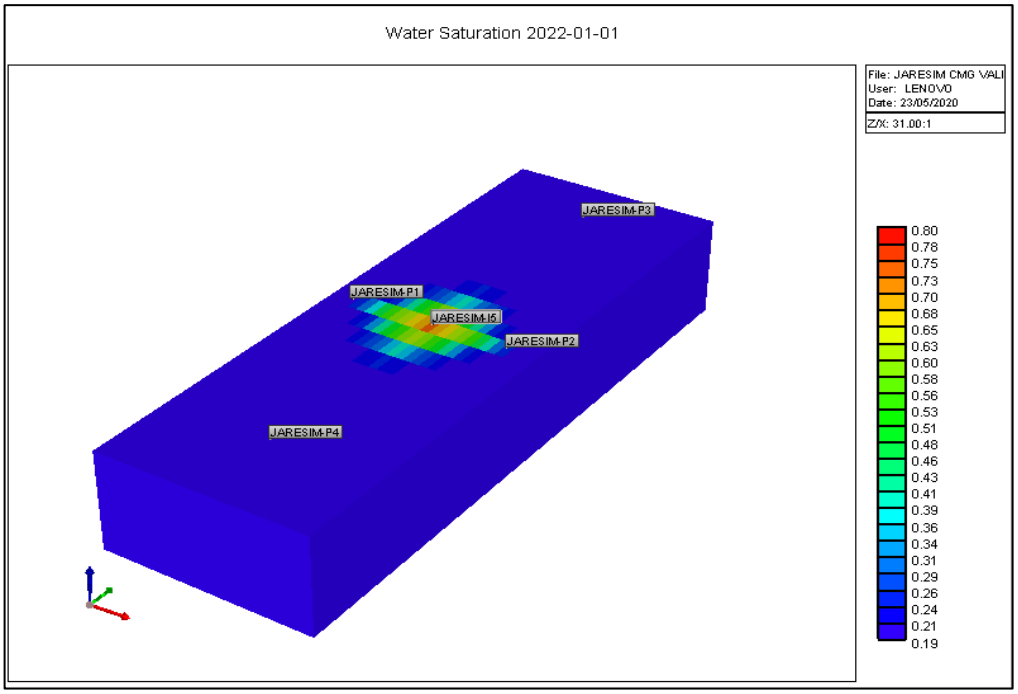


Ilustración 92:Gráfico de Distribución 3D de Saturación de Agua a los 2920 días de Simulación para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

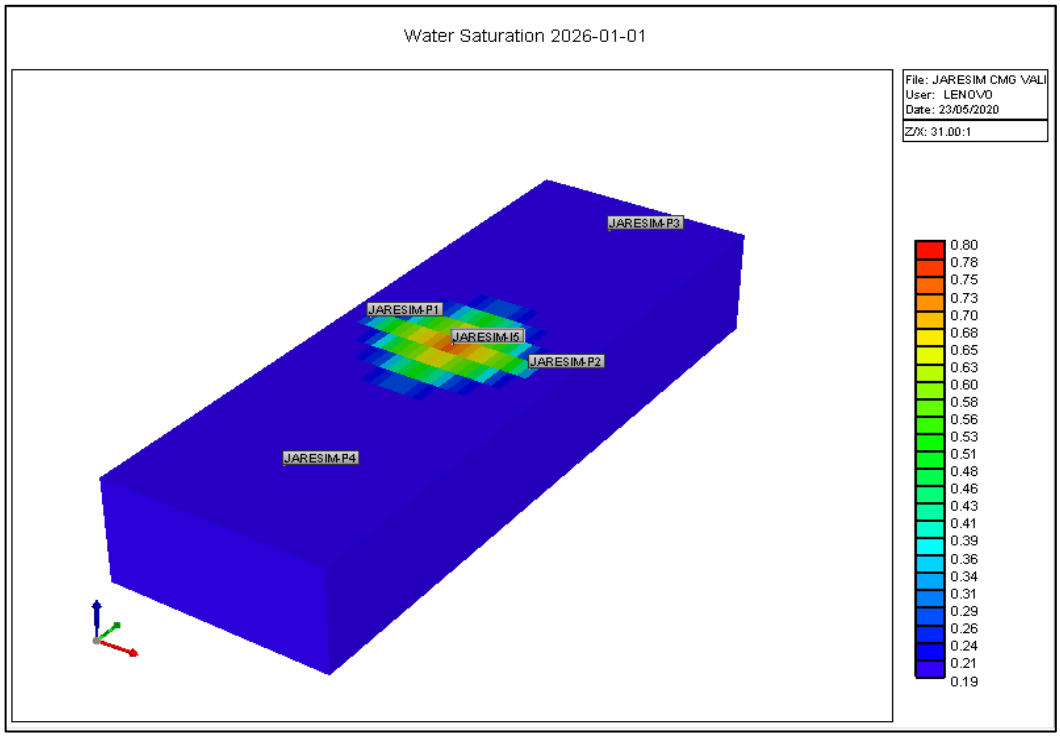


Ilustración 93:Gráfico de Distribución 3D de Saturación de Agua a los 4380 días de Simulación para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

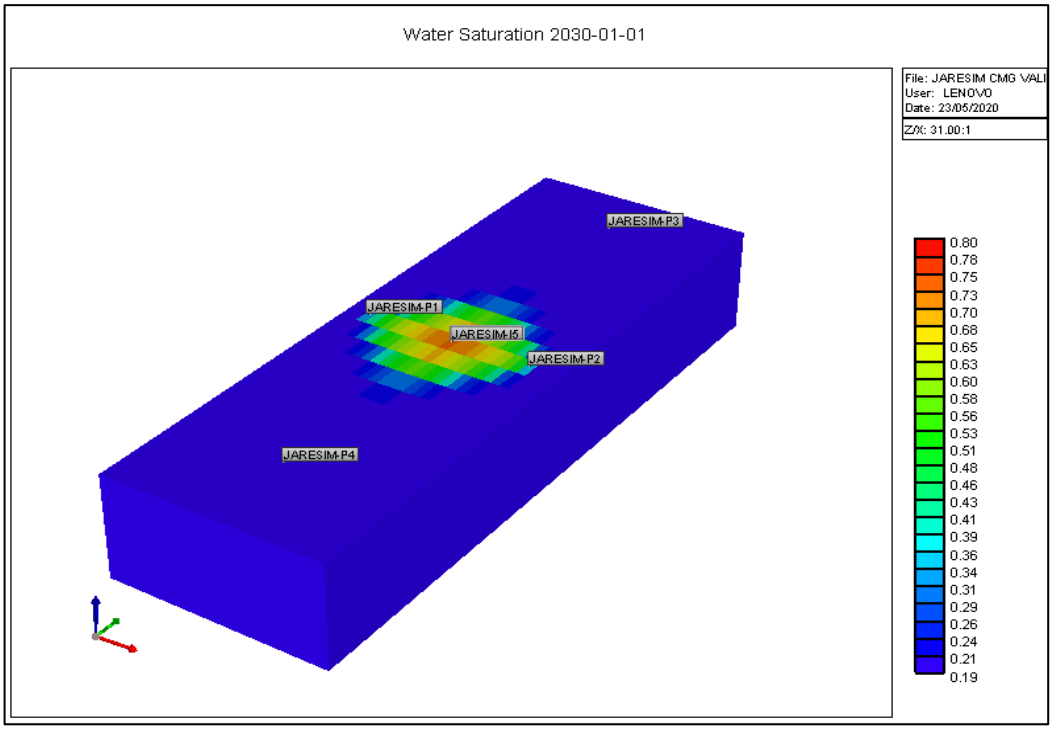


Ilustración 94:Gráfico de Distribución 3D de Saturación de Agua a los 5840 días de Simulación para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

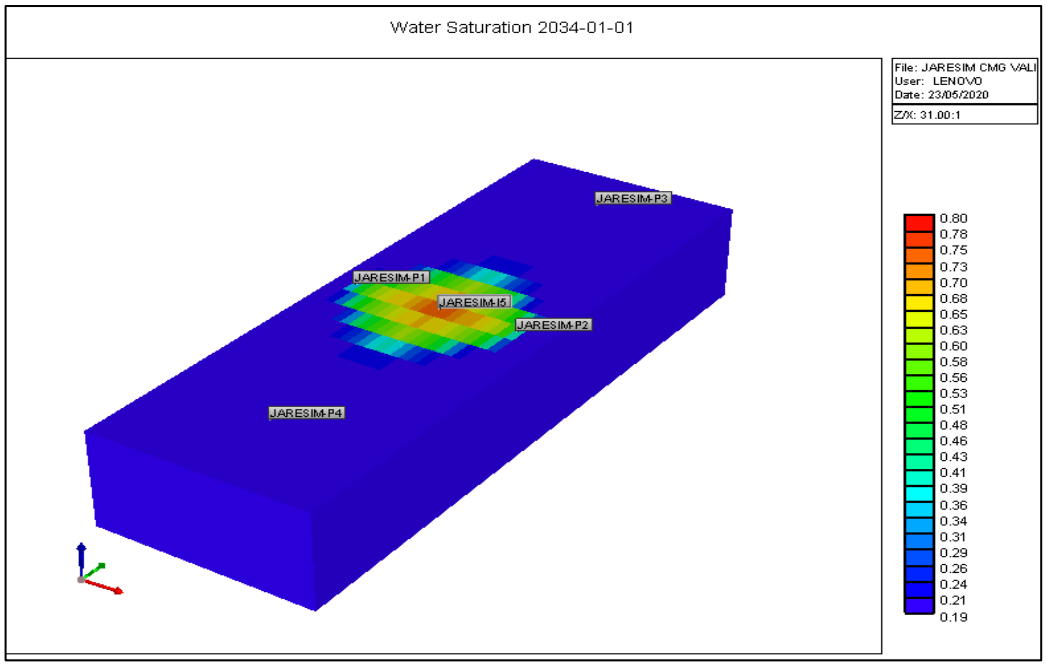


Ilustración 95:Gráfico de Distribución 3D de Saturación de Agua a los 7300 días de Simulación para el caso de Validación #1 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

Los resultados previamente mostrados, serán usados como elemento referencial para la validación del programa de simulación propuesto en esta investigación.

10.3.2. Caso Simulador Propuesto-Modulo GPU

Las ilustraciones presentadas a continuación muestras los resultados correspondientes a las curvas de producción de Petróleo, Agua y Gas de los 4 pozos productores, así como la tasa de Inyección, producción de Petróleo, Agua, Gas y Presión promedio a escala de Campo para caso de validación #1, generados en el simulador desarrollado en este proyecto de investigación:

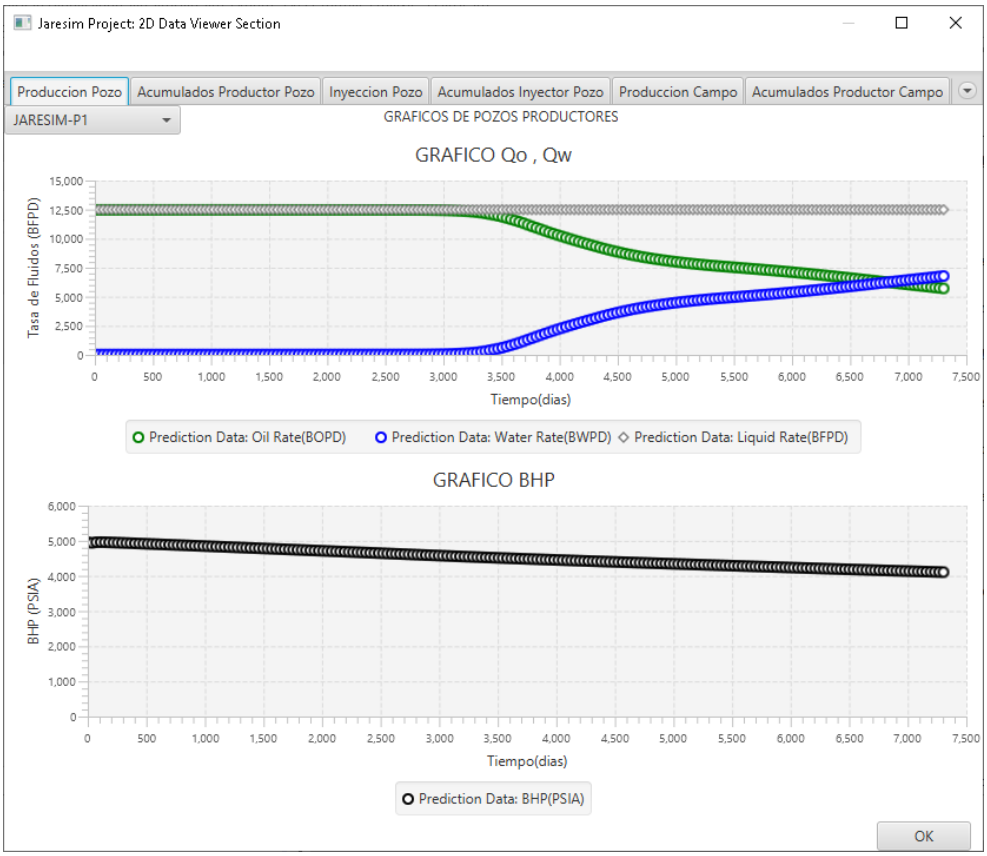


Ilustración 96:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P1 del caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

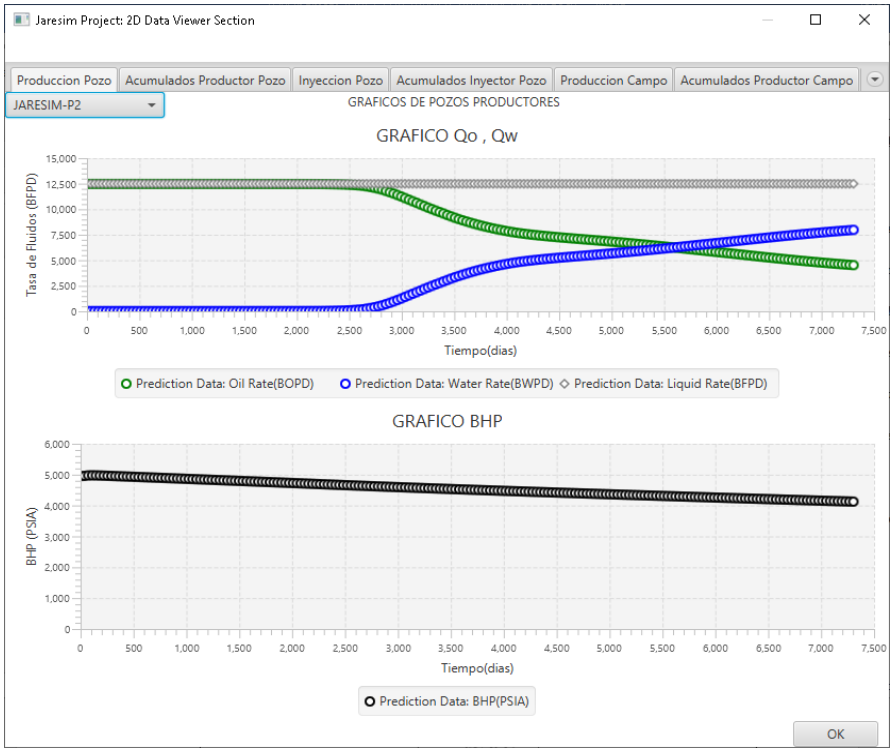


Ilustración 97:Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P2 del caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

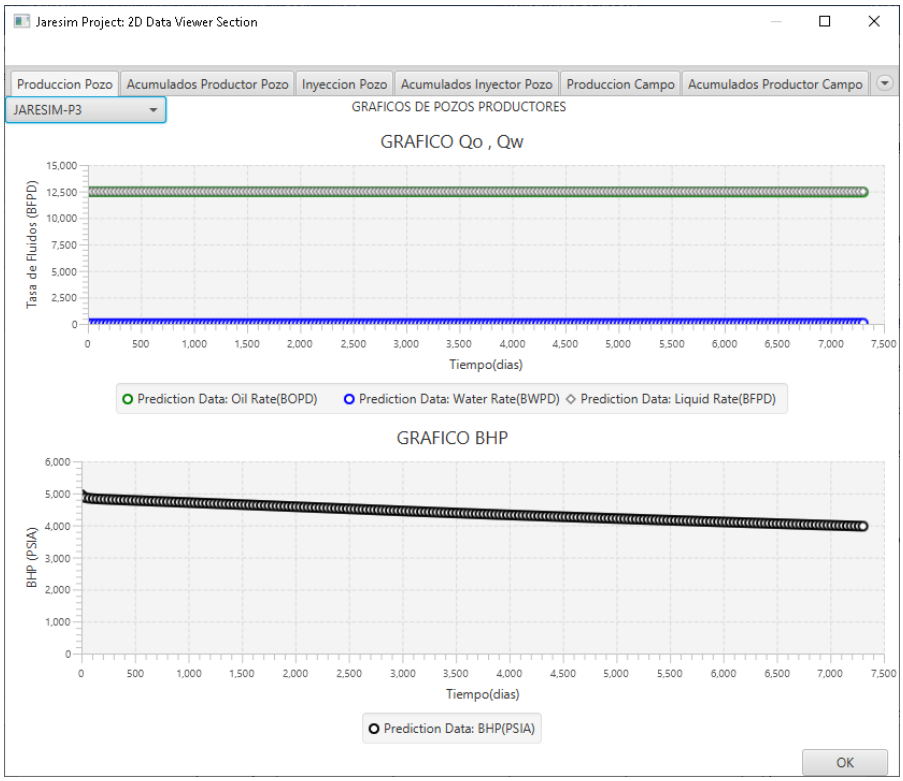


Ilustración 98:Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P3 del caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

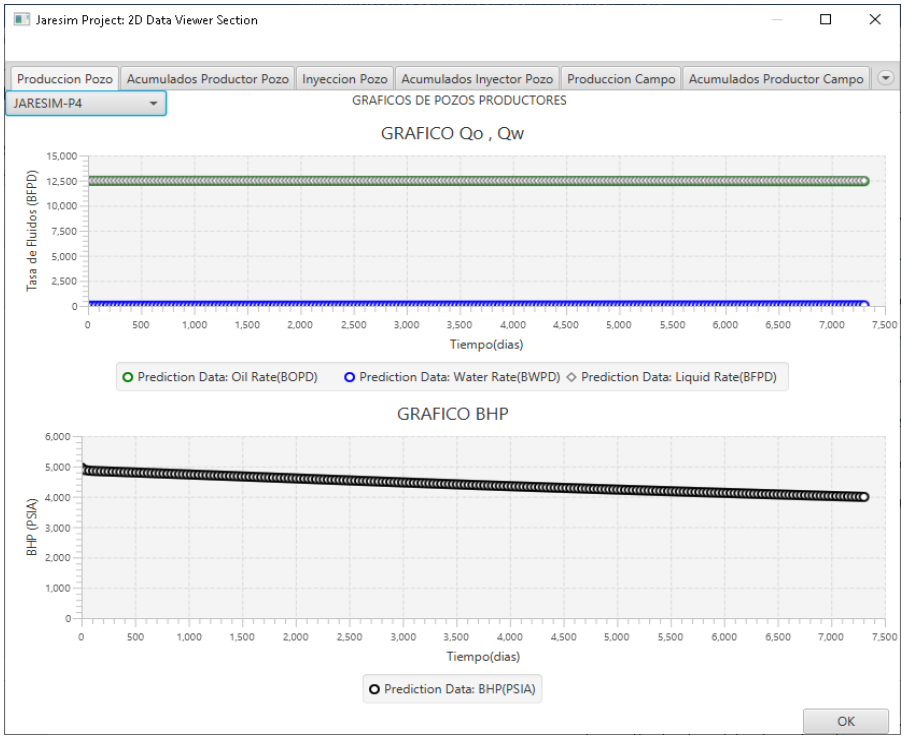


Ilustración 99:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo P4 del caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

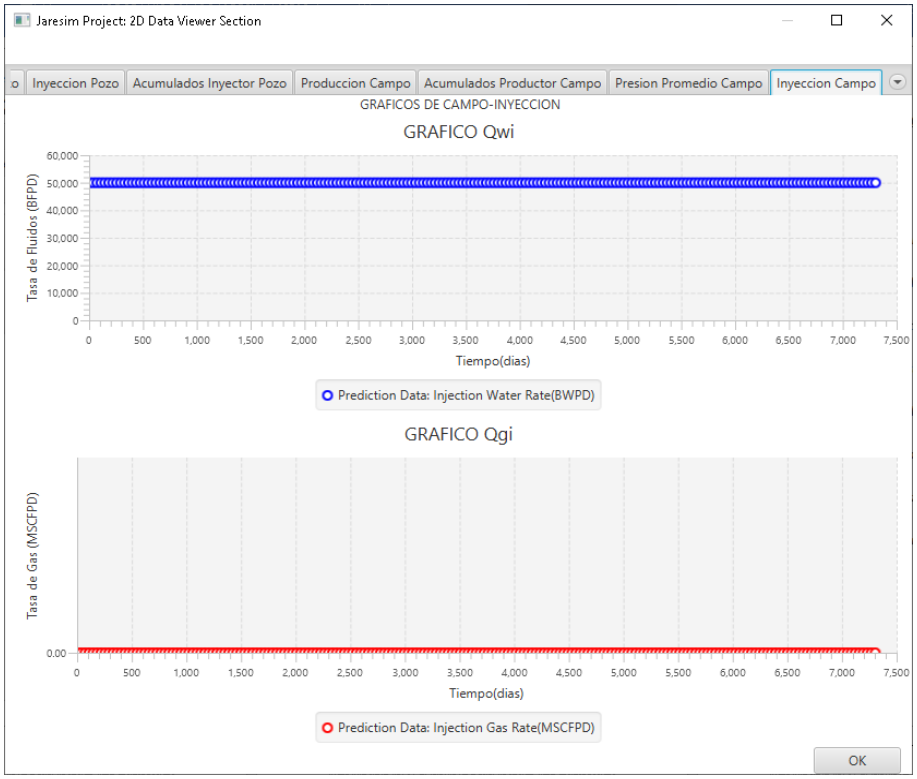


Ilustración 100:Gráfico de Curva de Tasa de Inyección de Agua (Azul) a nivel del Campo para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

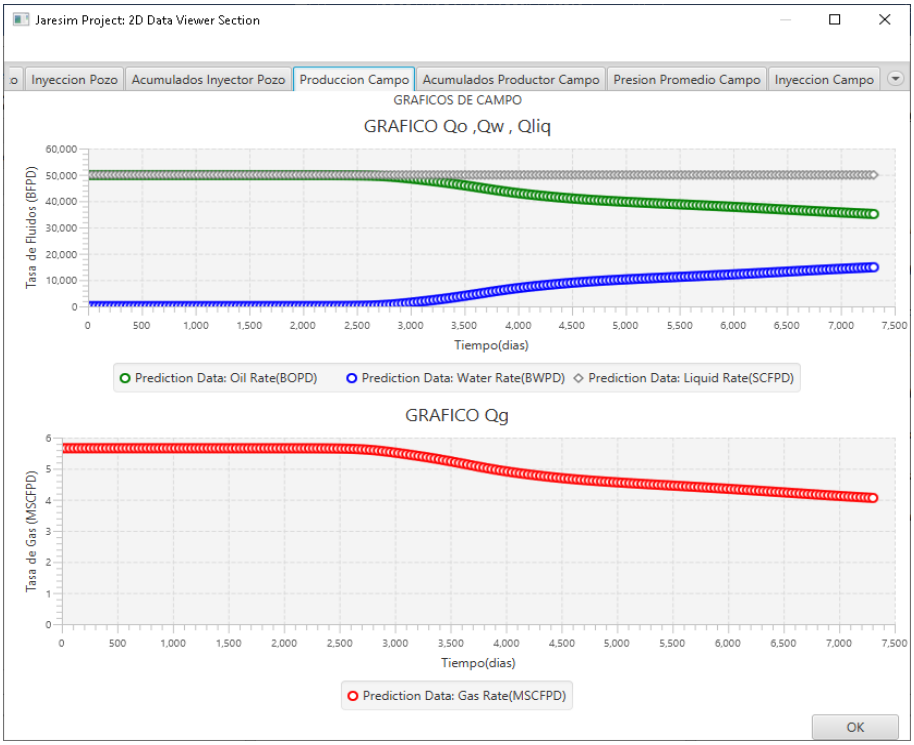


Ilustración 101:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) a escala de Campo para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

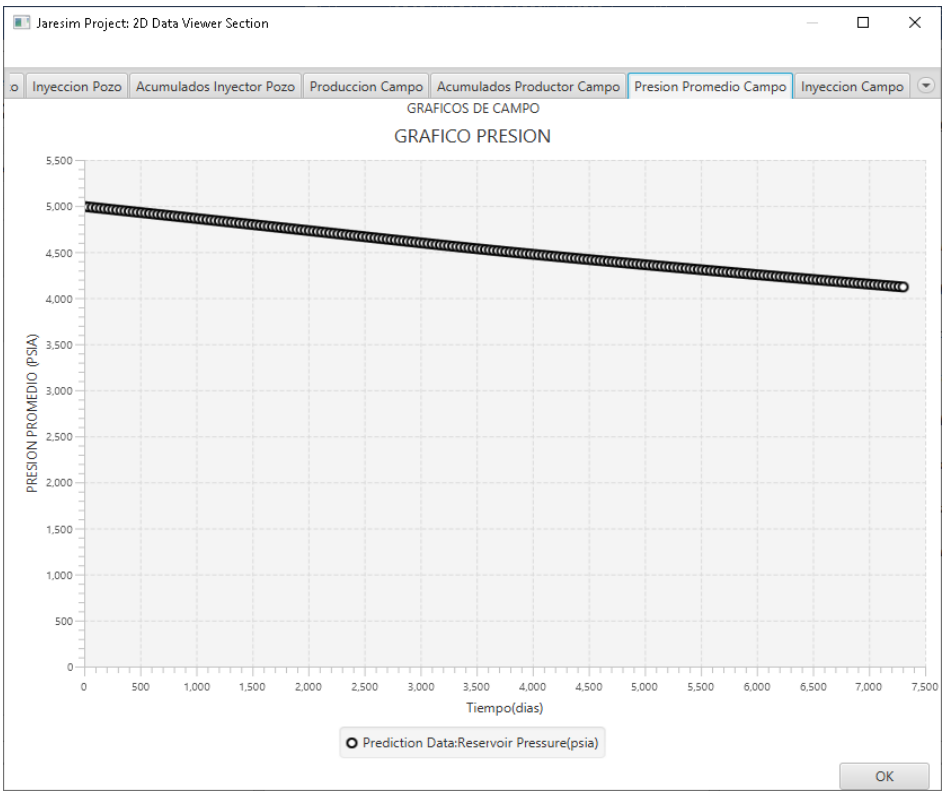


Ilustración 102:Gráfico de presión promedio (Negro) a escala de Campo para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

A continuación, se muestra la vista 3D de la variable saturación de agua para los siguientes estados temporales:

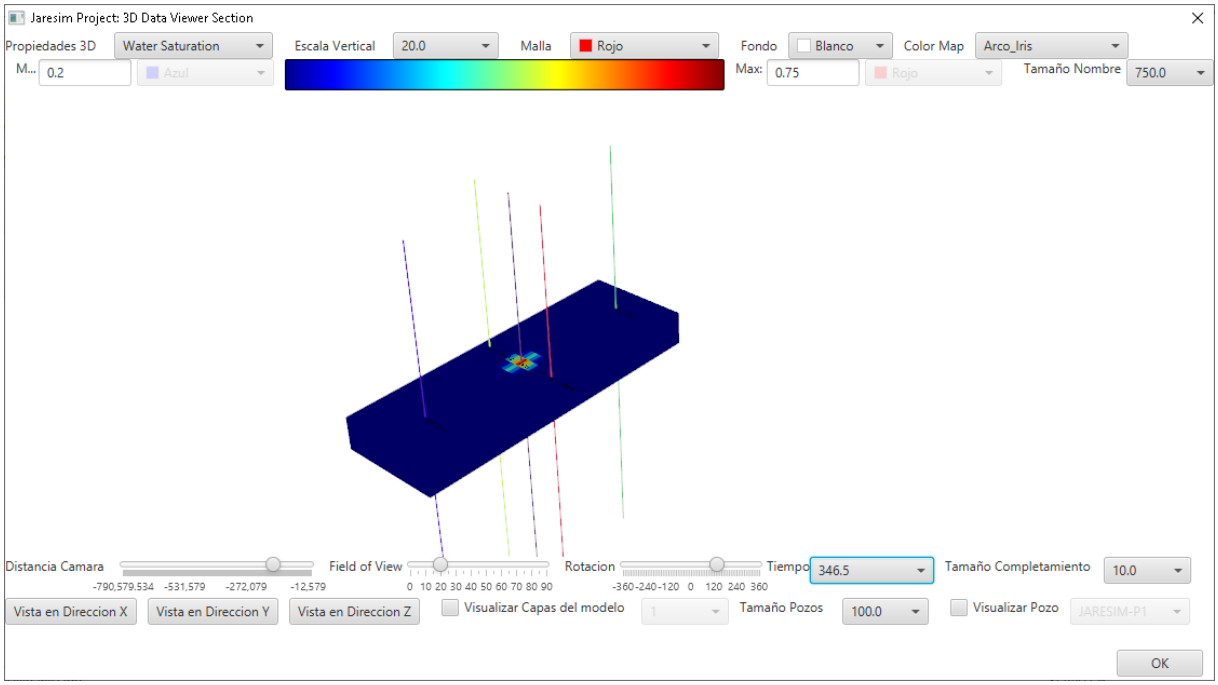


Ilustración 103:Gráfico de Distribución 3D de Saturación de Agua a los 347 días de Simulación para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

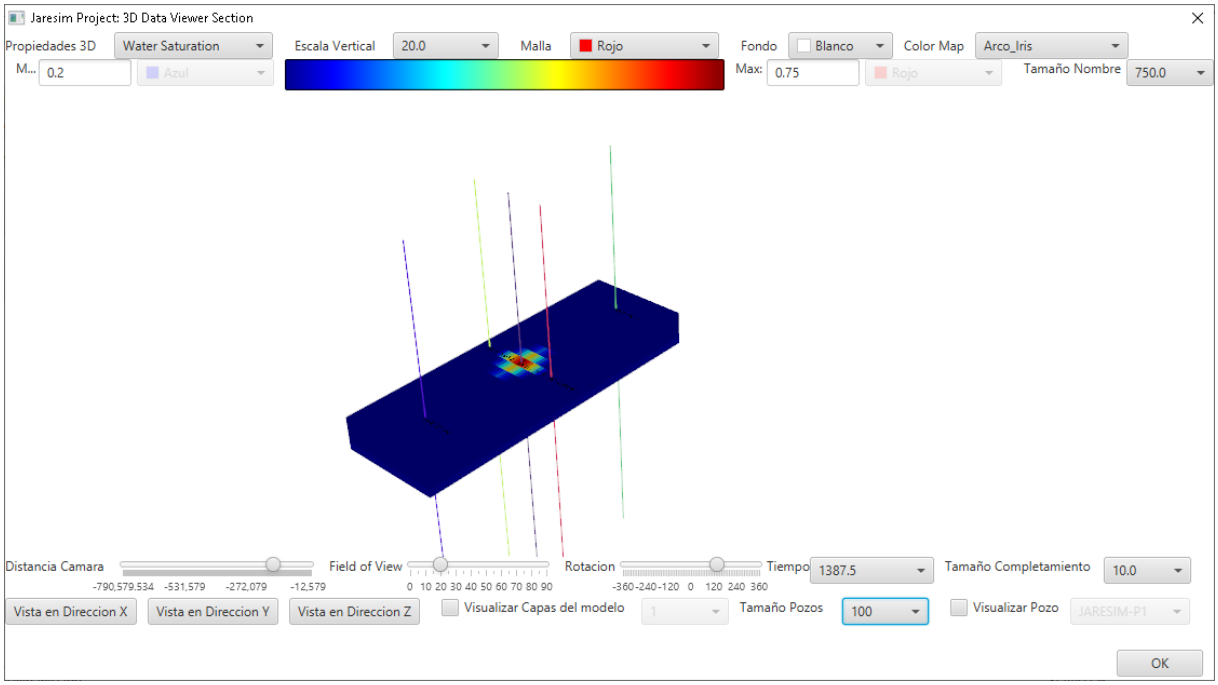


Ilustración 104:Gráfico de Distribución 3D de Saturación de Agua a los 1388 días de Simulación para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

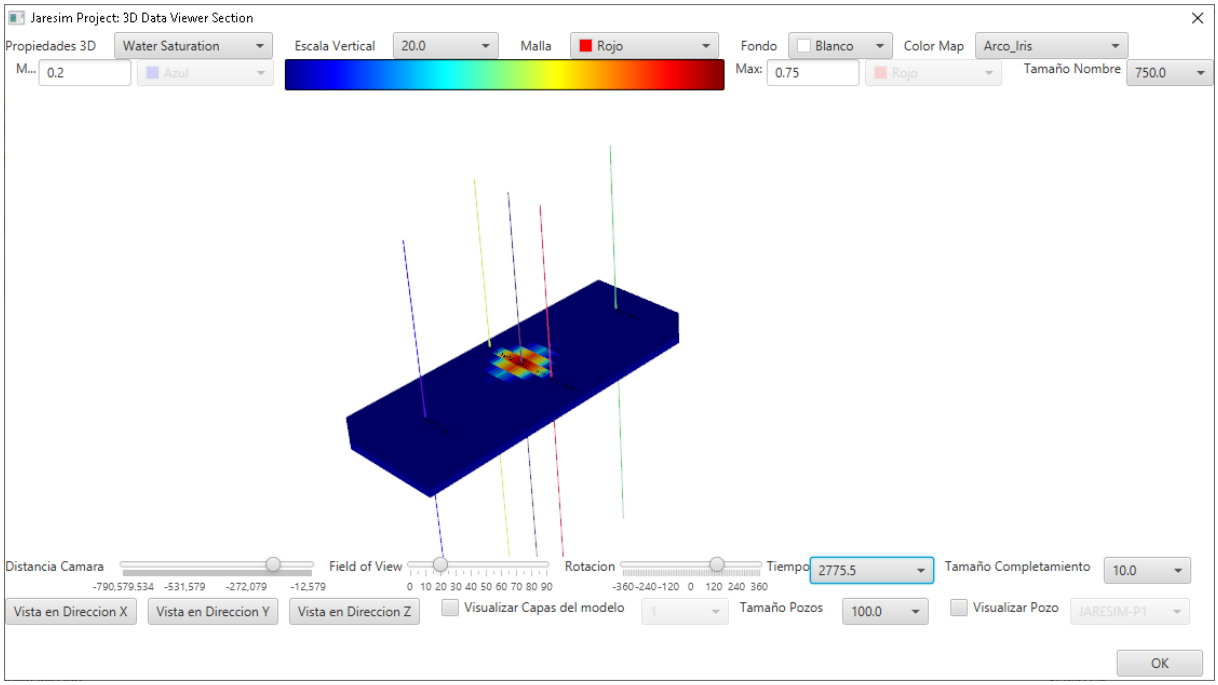


Ilustración 105:Gráfico de Distribución 3D de Saturación de Agua a los 2776 días de Simulación para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

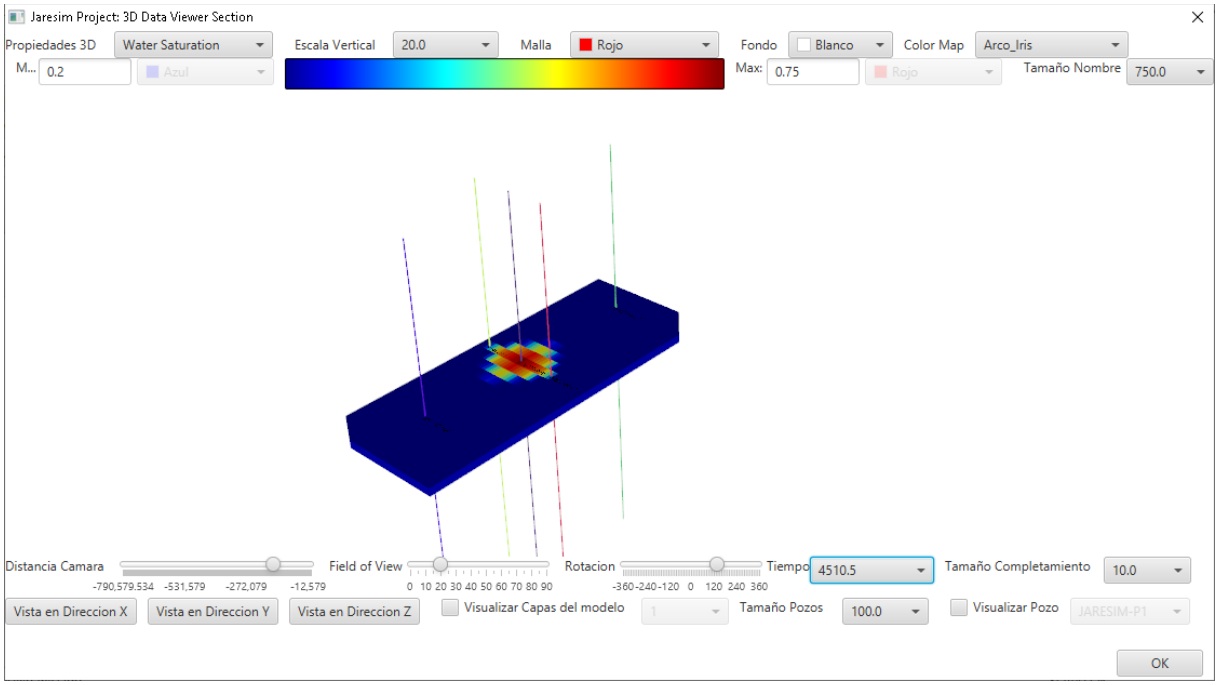


Ilustración 106:Gráfico de Distribución 3D de Saturación de Agua a los 4511 días de Simulación para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

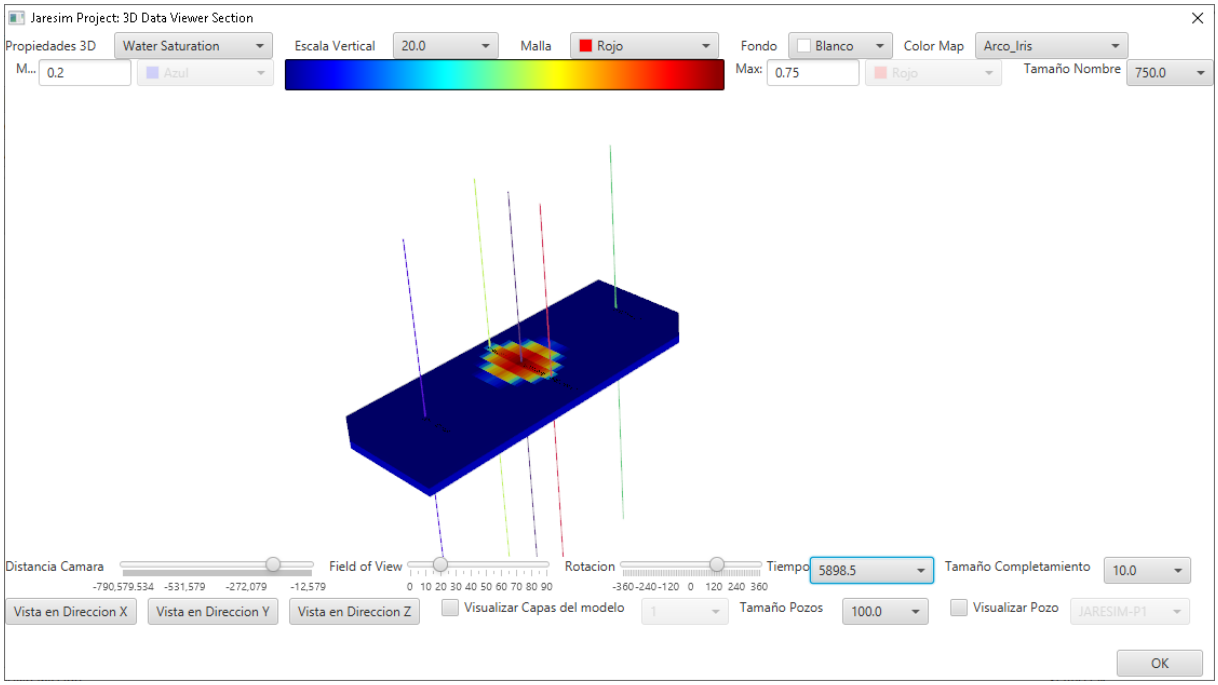


Ilustración 107:Gráfico de Distribución 3D de Saturación de Agua a los 5899 días de Simulación para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

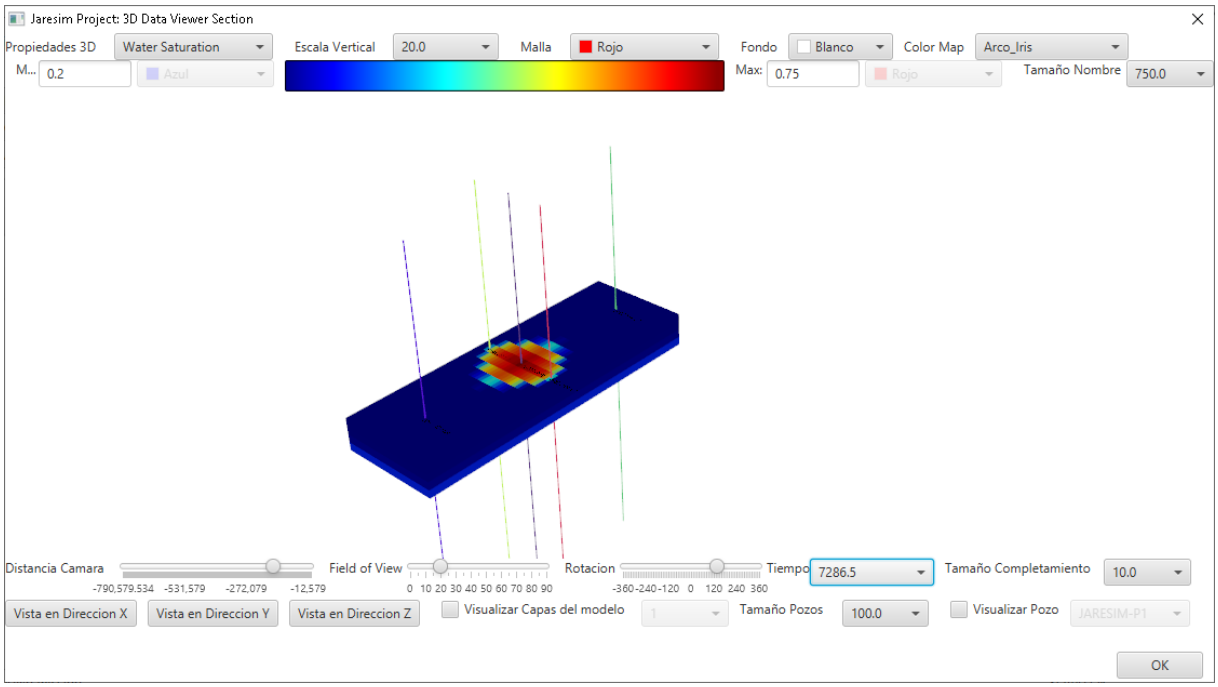


Ilustración 108:Gráfico de Distribución 3D de Saturación de Agua a los 7287 días de Simulación para el caso de Validación #1 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

Los resultados previamente mostrados, permiten ver que la aplicación es capaz de evidenciar los resultados que usualmente son presentados en un simulador típico comercial.

10.4. Resultados Obtenidos para el Caso de Validación #2

A continuación, se muestran los resultados obtenidos para el software comercial IMEX de la compañía CMG y los resultados generados por el simulador propuesto por esta investigación:

10.4.1. Caso Simulador Comercial IMEX

Las ilustraciones presentadas a continuación muestras los resultados correspondientes a las curvas de producción de Petróleo, Agua y Gas de los 4 pozos productores, así como la tasa de Inyección, producción de Petróleo, Agua, Gas y Presión promedio a escala de Campo para caso de validación #2 generados en el simulador Comercial IMEX de la Compañía Computer Modeling Group:

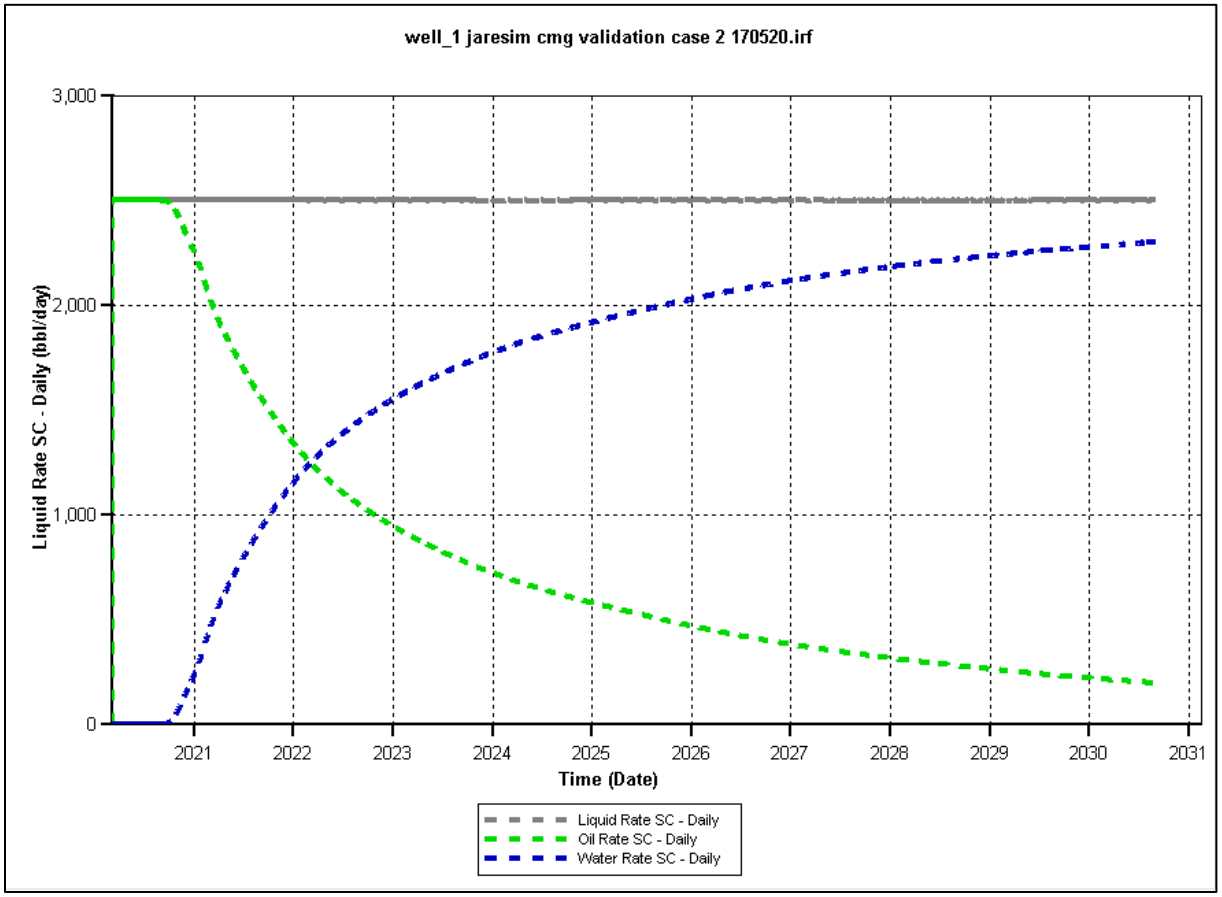


Ilustración 109:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_1s del caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

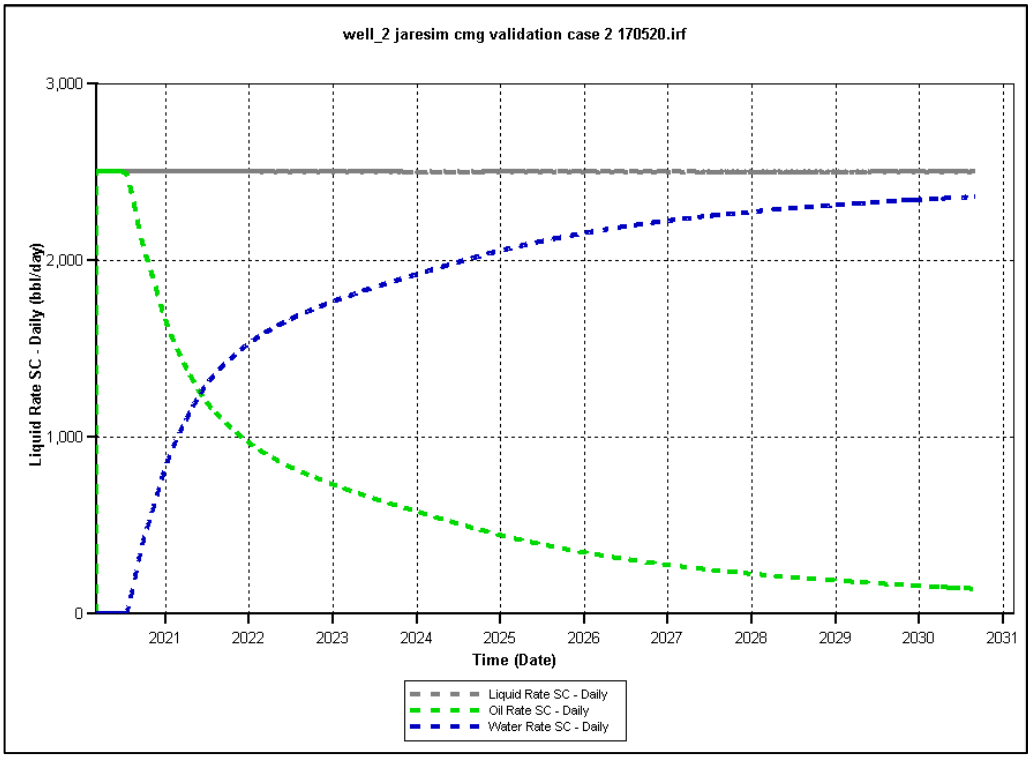


Ilustración 110:Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_2s del caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

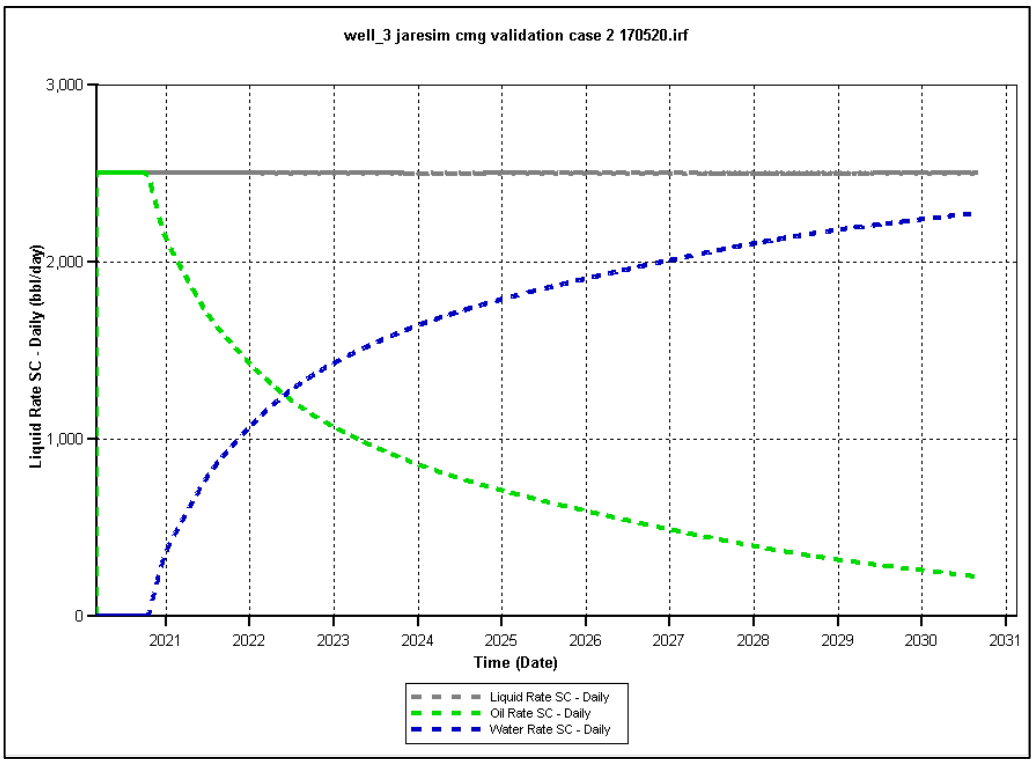


Ilustración 111:Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_3s del caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

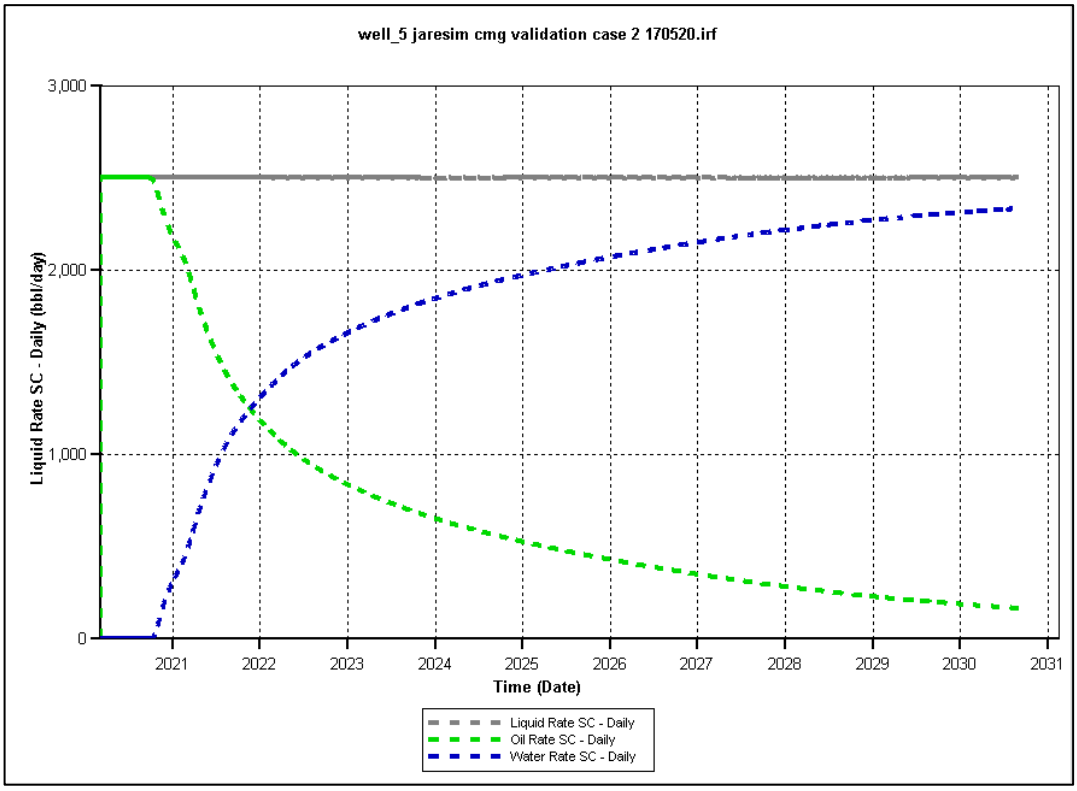


Ilustración 112:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_5s del caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

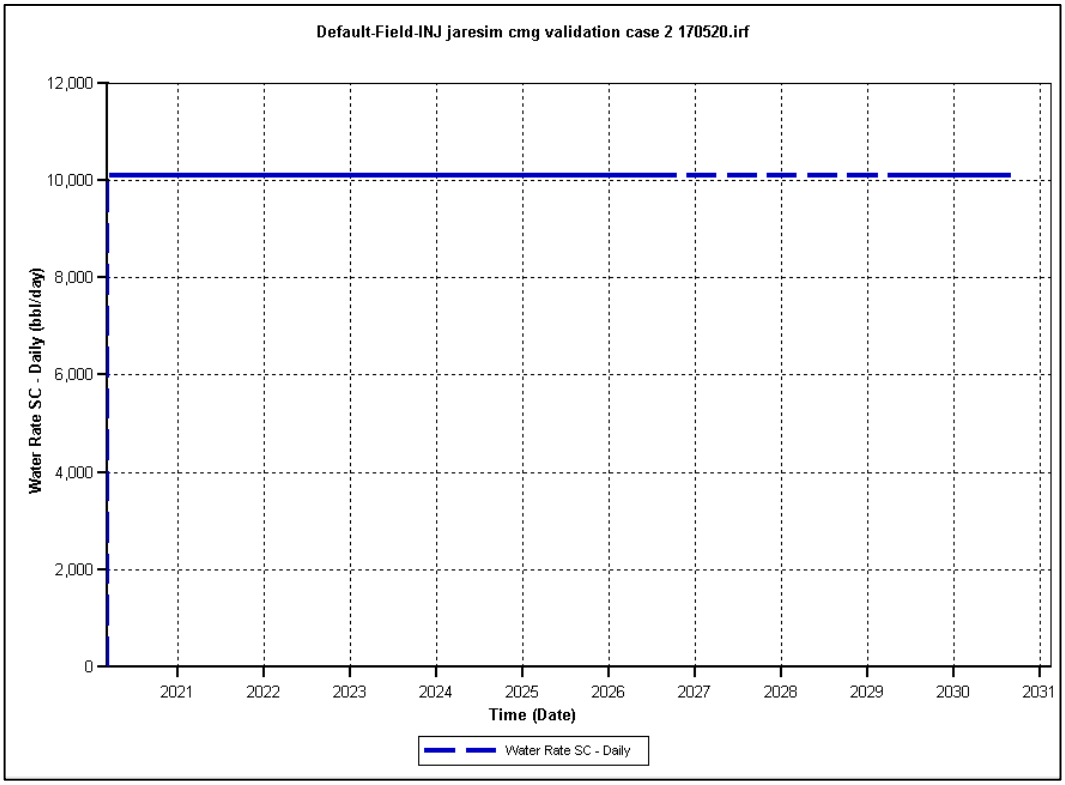


Ilustración 113:Gráfico de Curva de Tasa de Inyección de Agua (Azul) a nivel del Campo para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

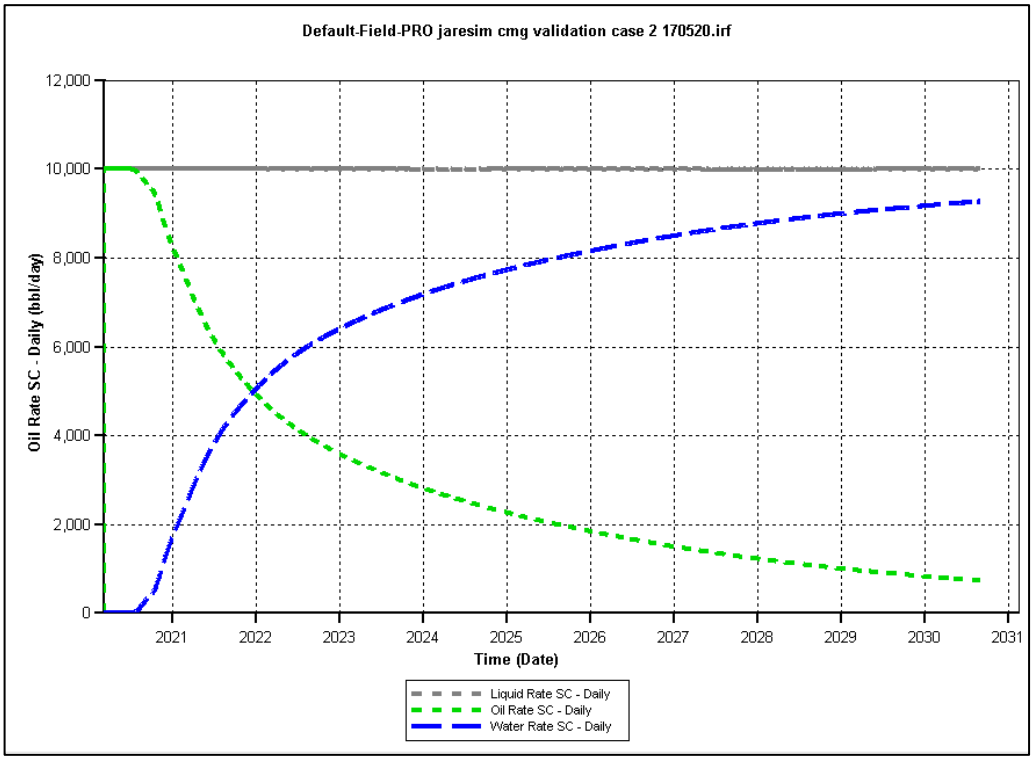


Ilustración 114:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) a escala de Campo para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

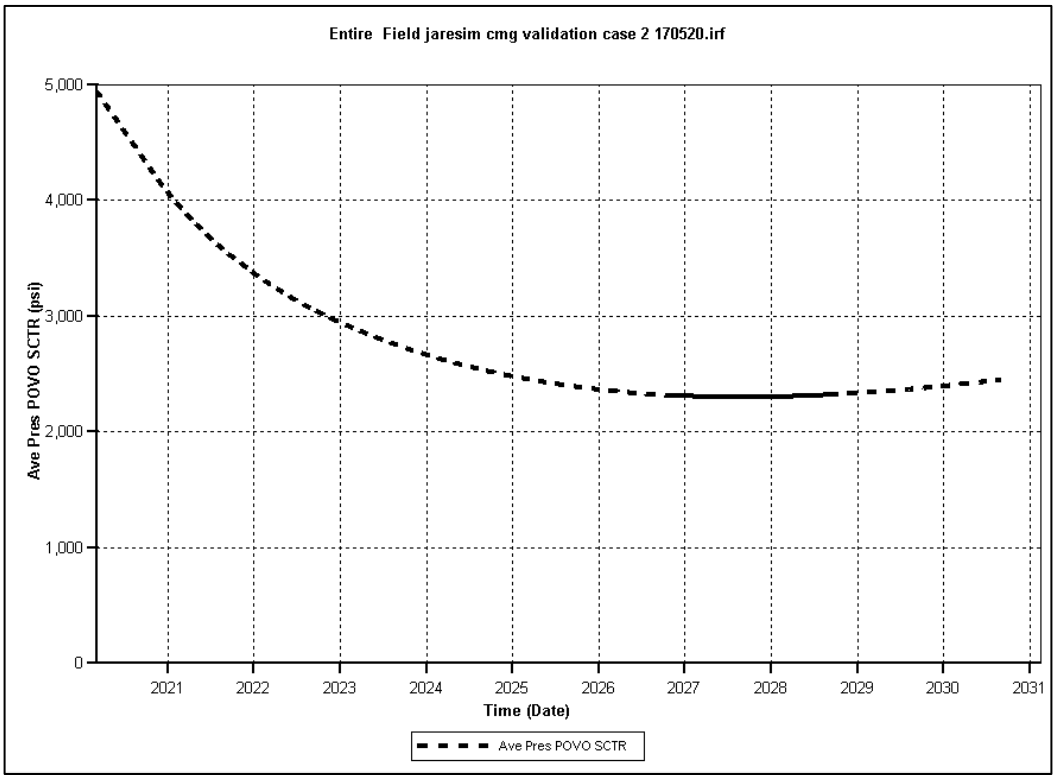


Ilustración 115:Gráfico de presión promedio (Negro) a escala de Campo para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

A continuación, se muestra la vista 3D de la variable saturación de agua para los siguientes estados temporales:

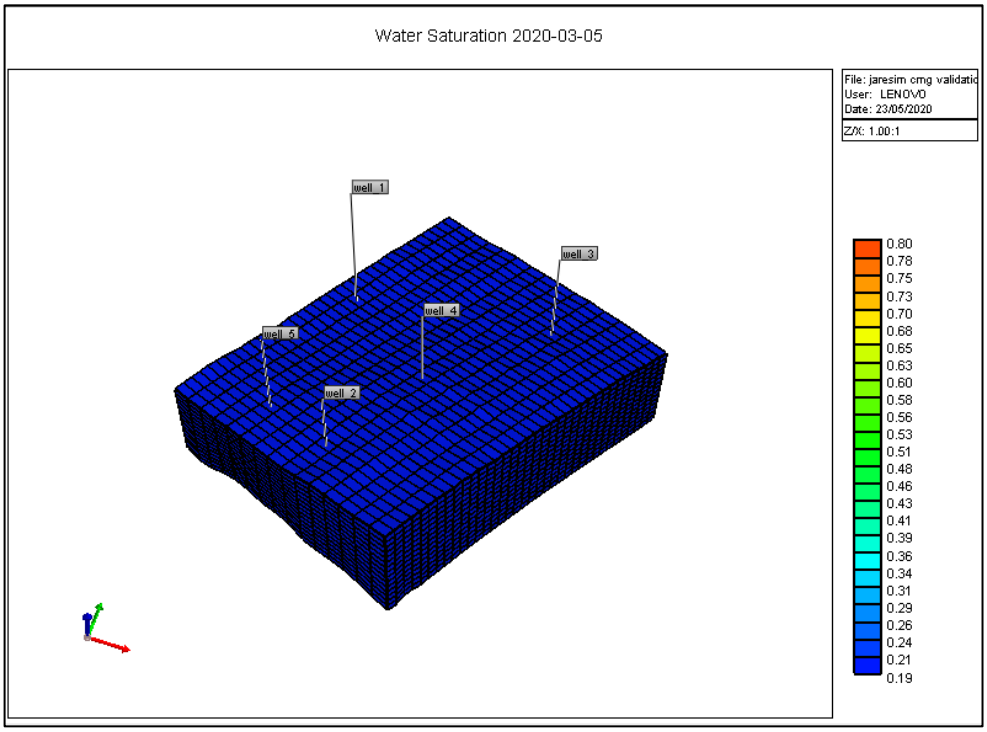


Ilustración 116:Gráfico de Distribución 3D de Saturación de Agua a los 365 días de Simulación para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

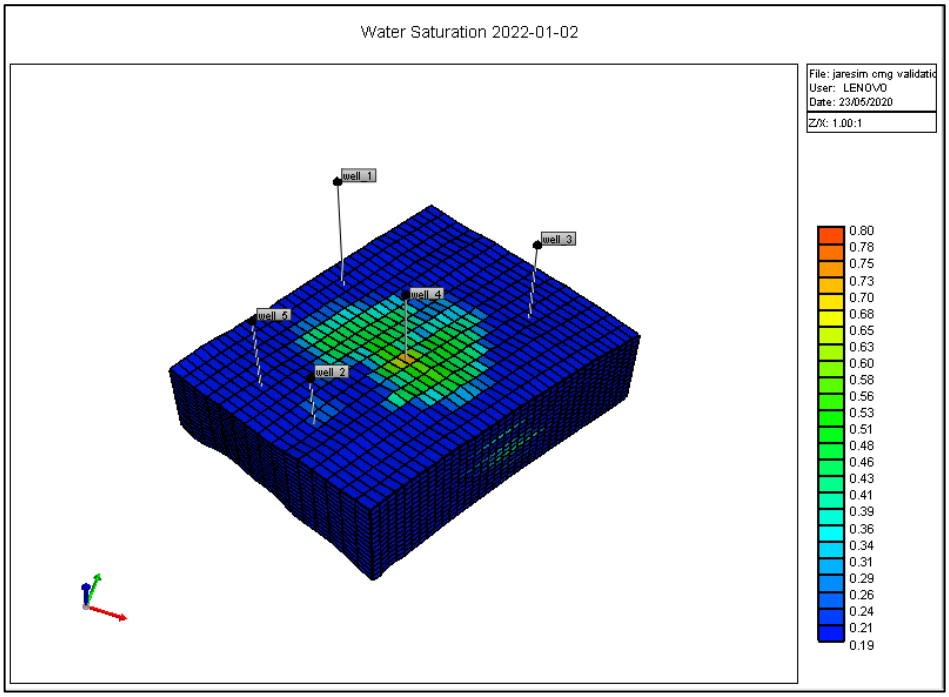


Ilustración 117:Gráfico de Distribución 3D de Saturación de Agua a los 760 días de Simulación para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

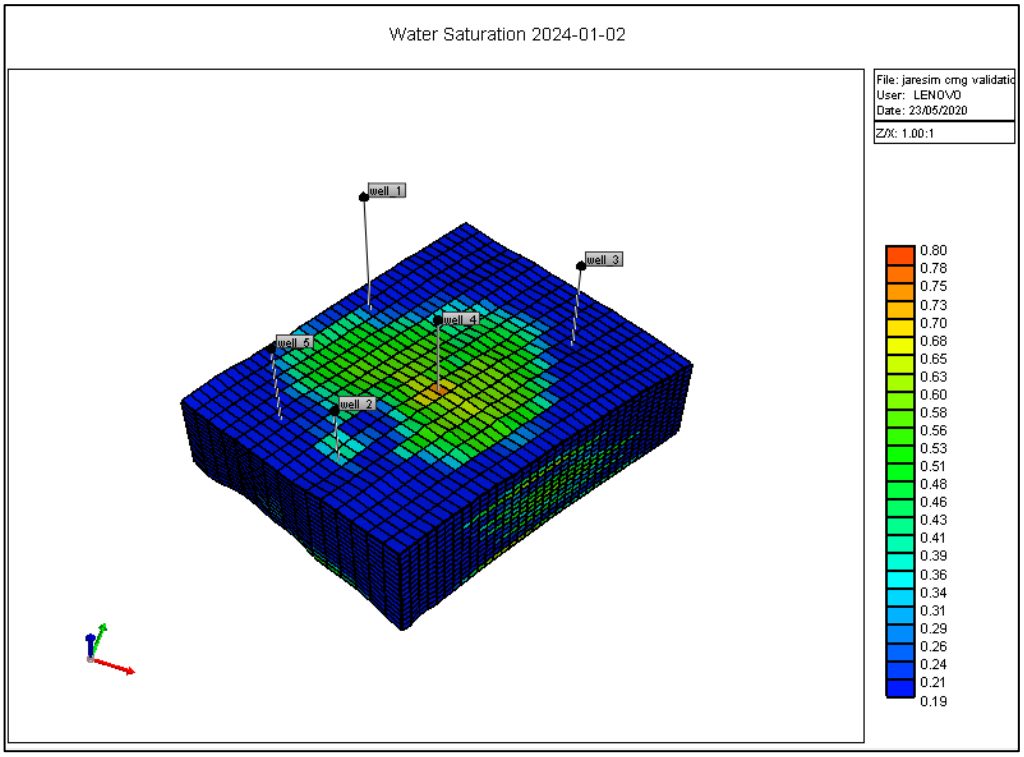


Ilustración 118:Gráfico de Distribución 3D de Saturación de Agua a los 2920 días de Simulación para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

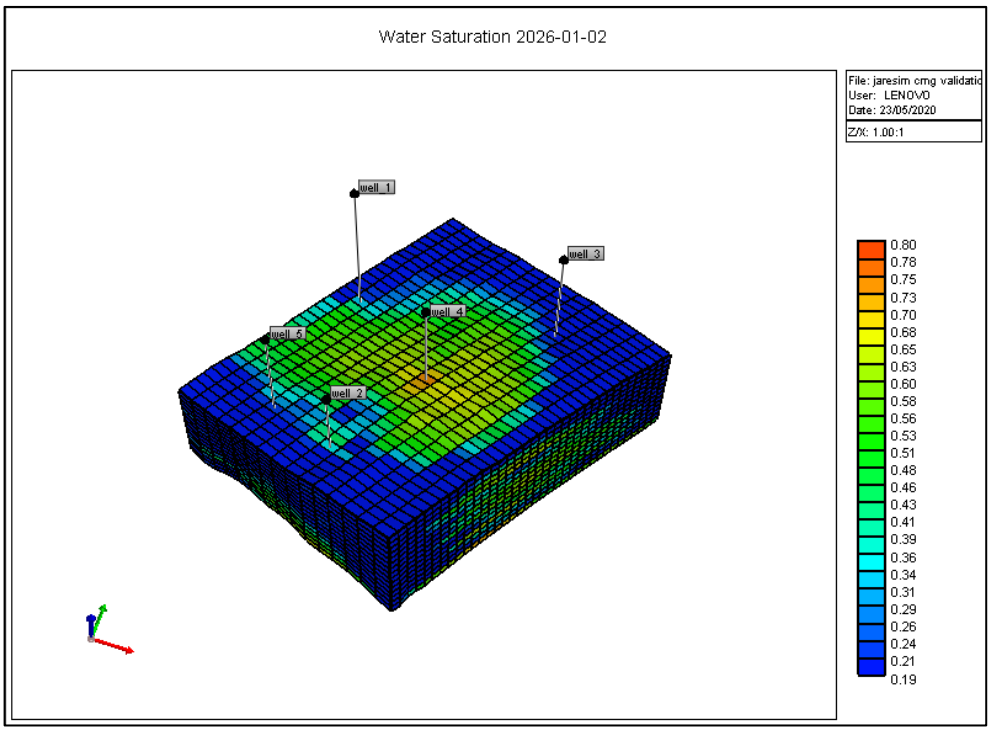


Ilustración 119:Gráfico de Distribución 3D de Saturación de Agua a los 4380 días de Simulación para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

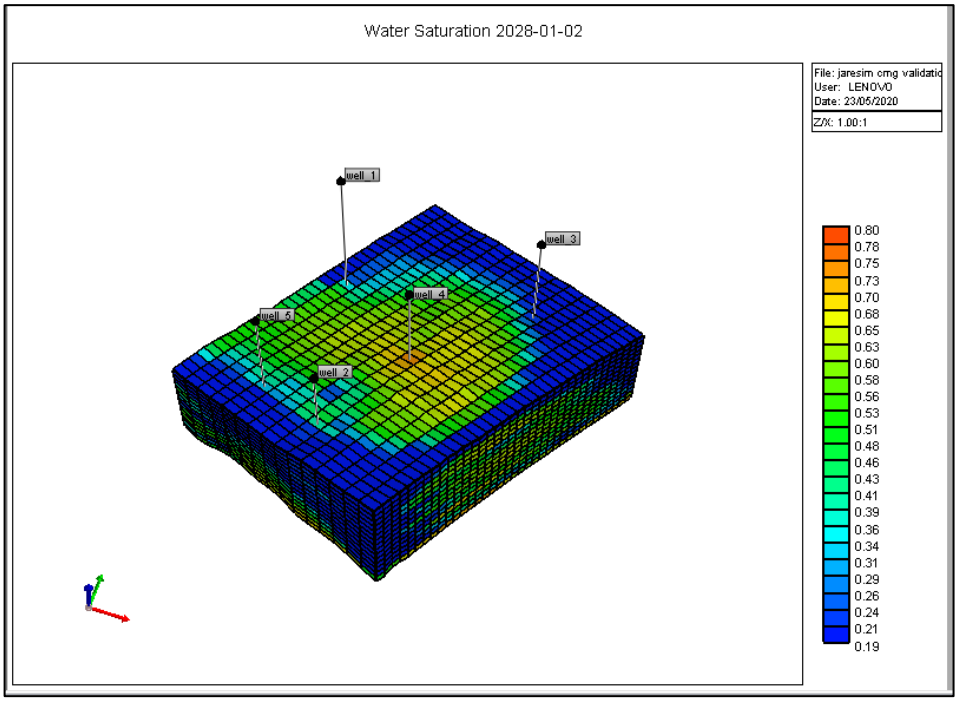


Ilustración 120:Gráfico de Distribución 3D de Saturación de Agua a los 5840 días de Simulación para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

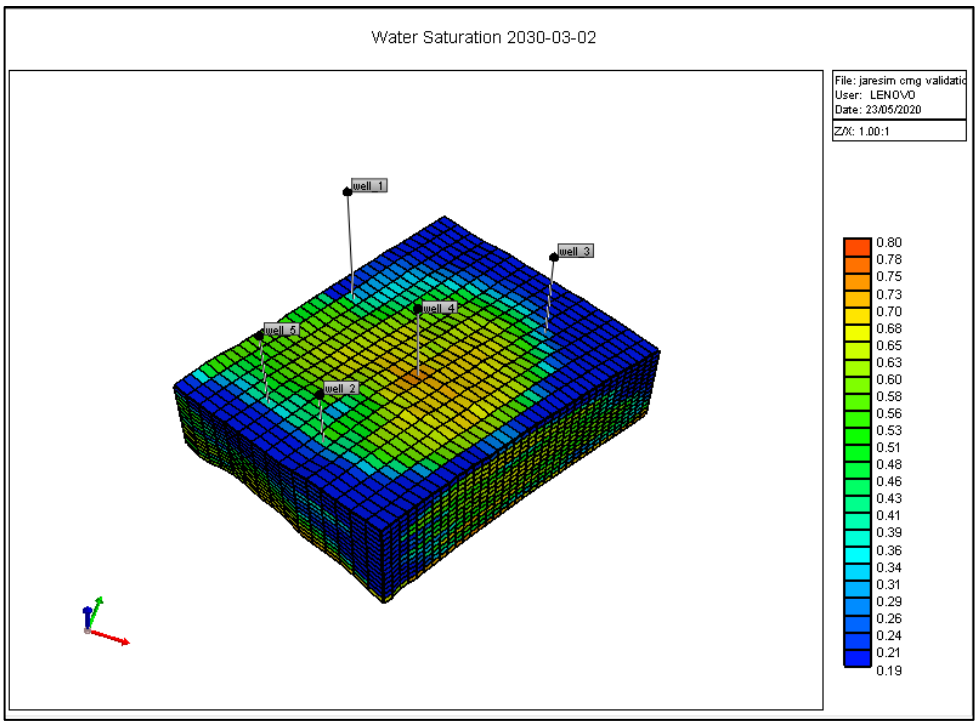


Ilustración 121:Gráfico de Distribución 3D de Saturación de Agua a los 7300 días de Simulación para el caso de Validación #2 mediante el simulador Comercial IMEX. Fuente: CMG IMEX 2019.

Los resultados previamente mostrados, serán usados como elemento referencial para la validación del programa de simulación propuesto en esta investigación.

10.4.2. Caso Simulador Propuesto-Modulo GPU

Las ilustraciones presentadas a continuación muestran los resultados correspondientes a las curvas de producción de Petróleo, Agua y Gas de los 4 pozos productores, así como la tasa de Inyección, producción de Petróleo, Agua, Gas y Presión promedio a escala de Campo para caso de validación #2, generados en el simulador desarrollado en este proyecto de investigación:

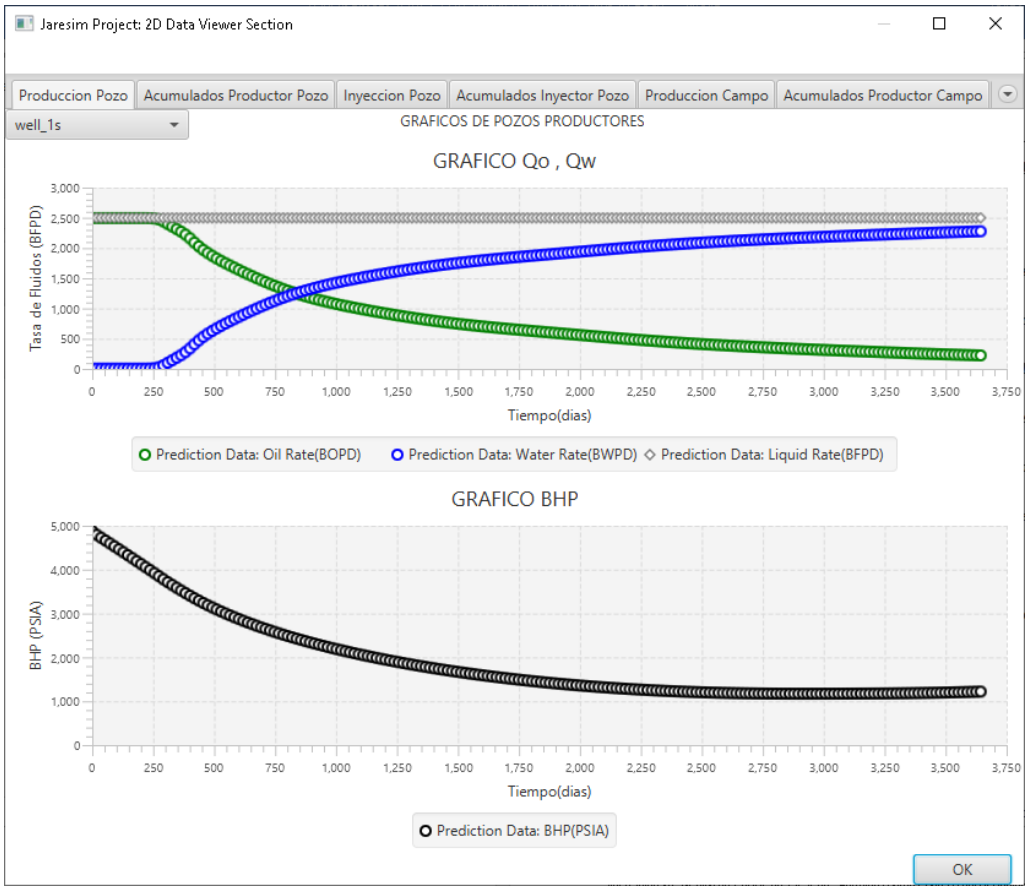


Ilustración 122:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_1s del caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

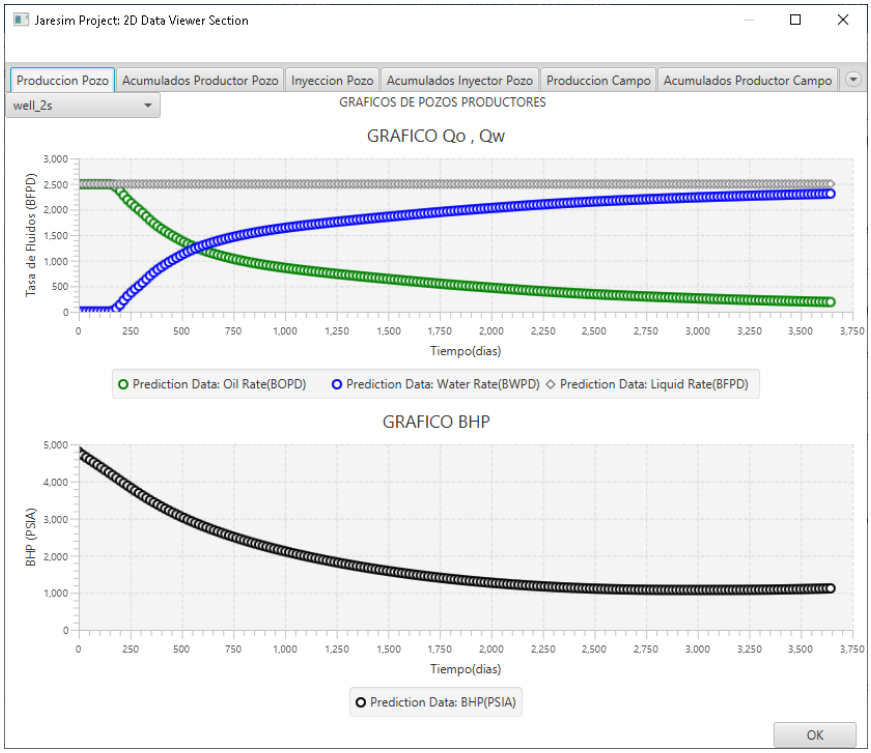


Ilustración 123:Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_2s del caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

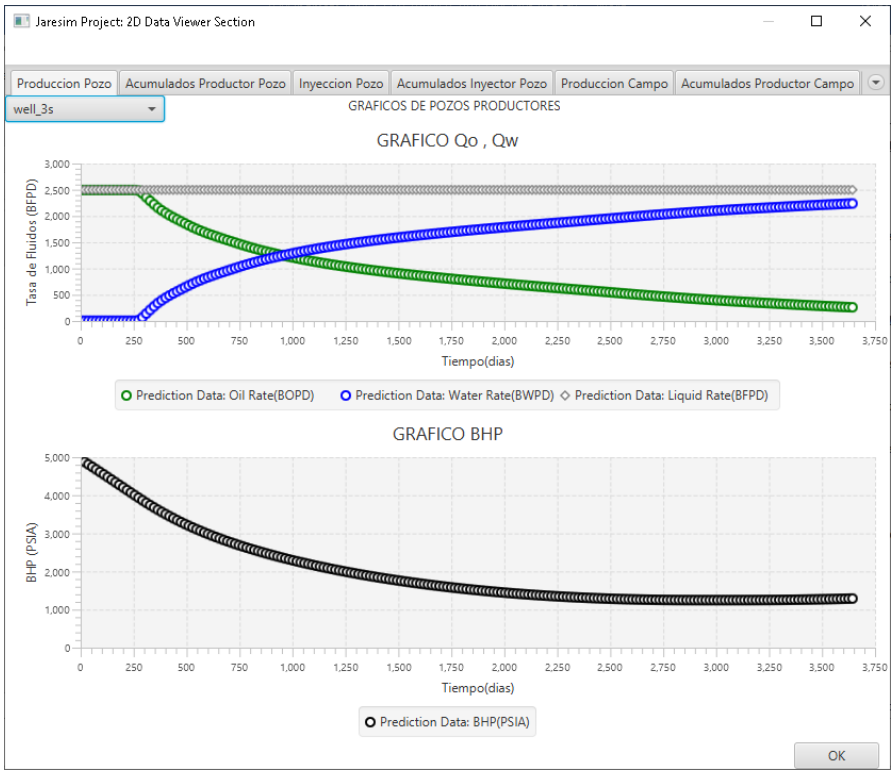


Ilustración 124:Gráfico de Curvas de Tasas de_ Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_3s del caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

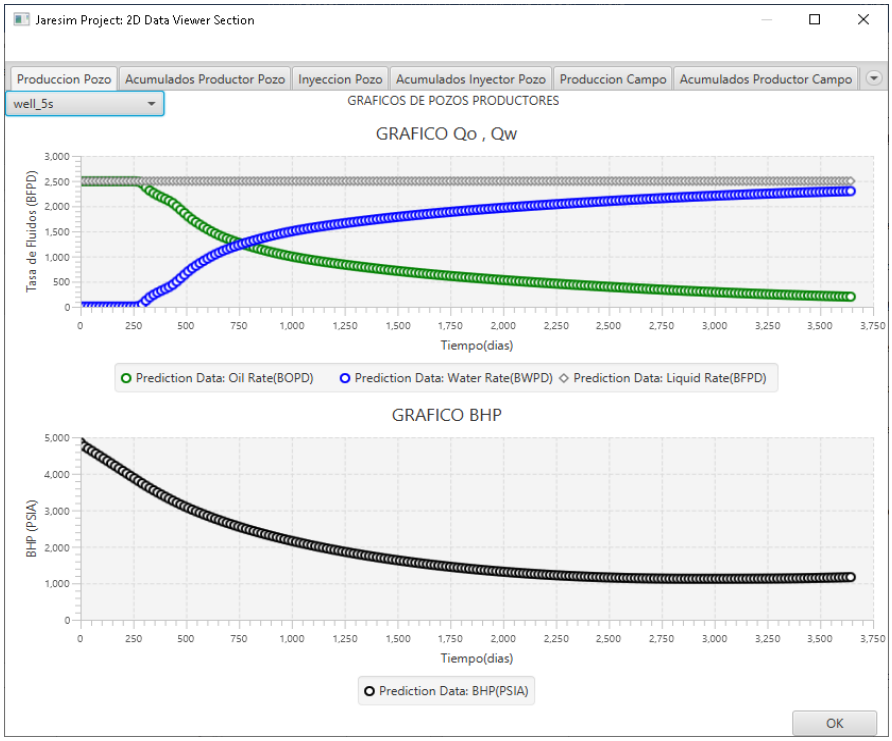


Ilustración 125:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) del pozo well_5s del caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

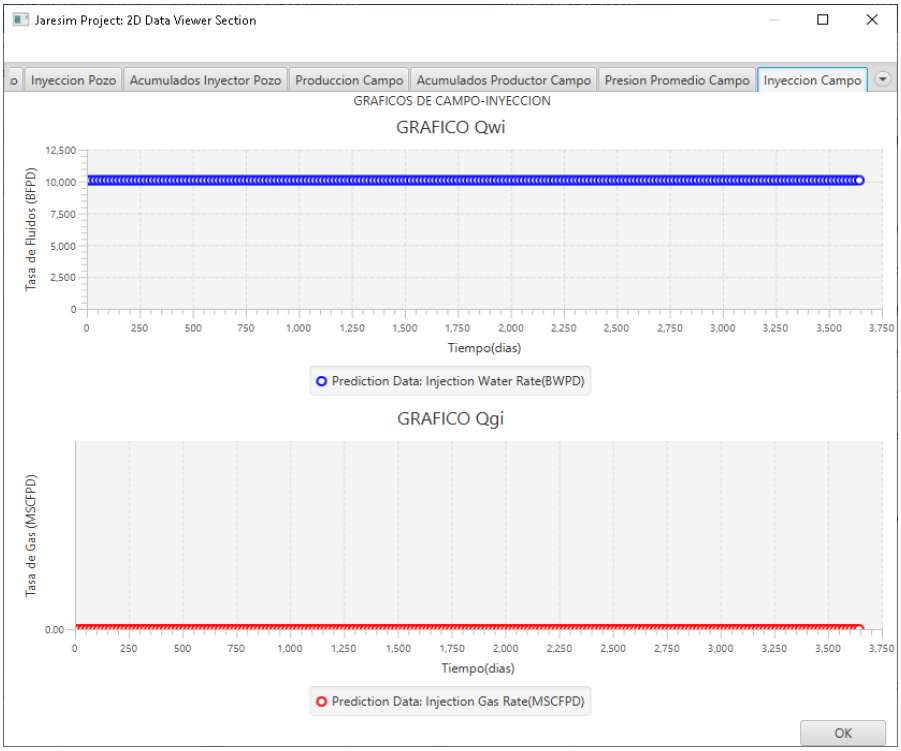


Ilustración 126:Gráfico de Curva de Tasa de Inyección de Agua (Azul) a nivel del Campo para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

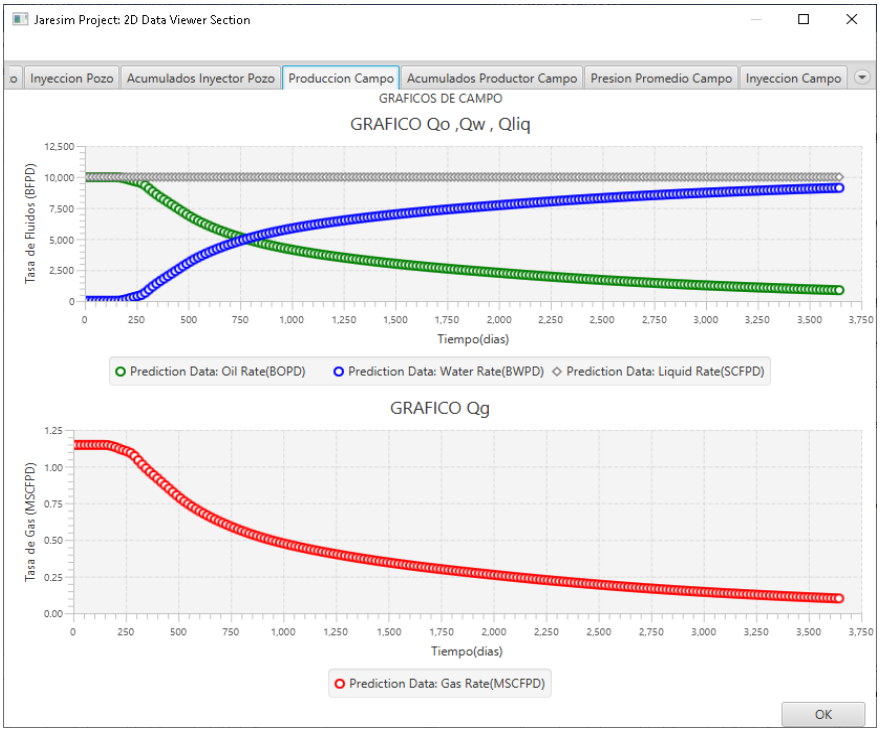


Ilustración 127:Gráfico de Curvas de Tasas de Petróleo (Verde), Agua (Azul) y Liquido (Gris) a escala de Campo para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

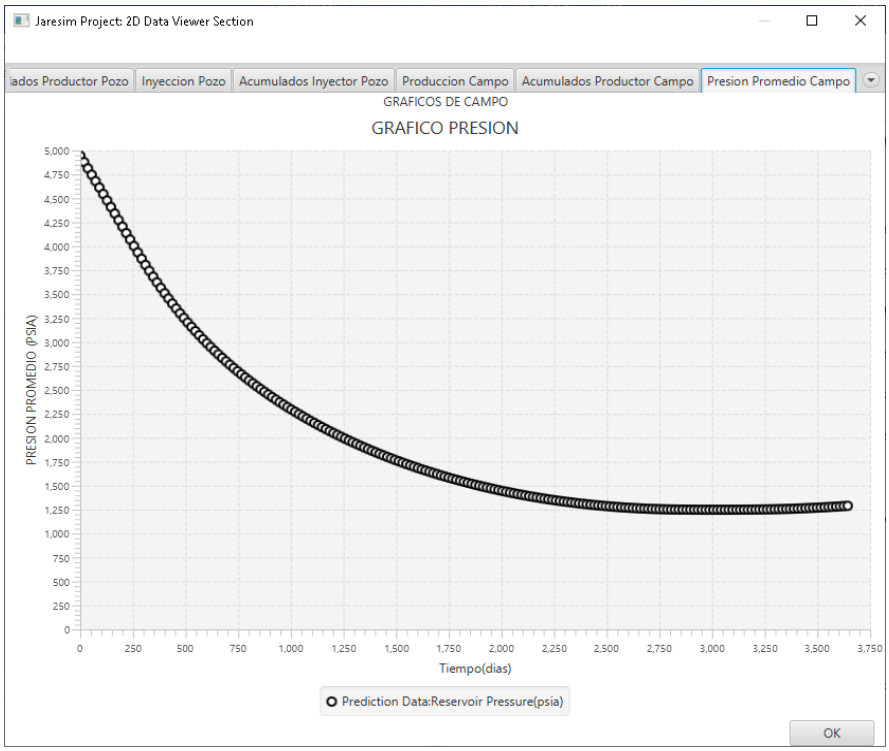


Ilustración 128:Gráfico de presión promedio (Negro) a escala de Campo para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

A continuación, se muestra la vista 3D de la variable saturación de agua para los siguientes estados temporales:

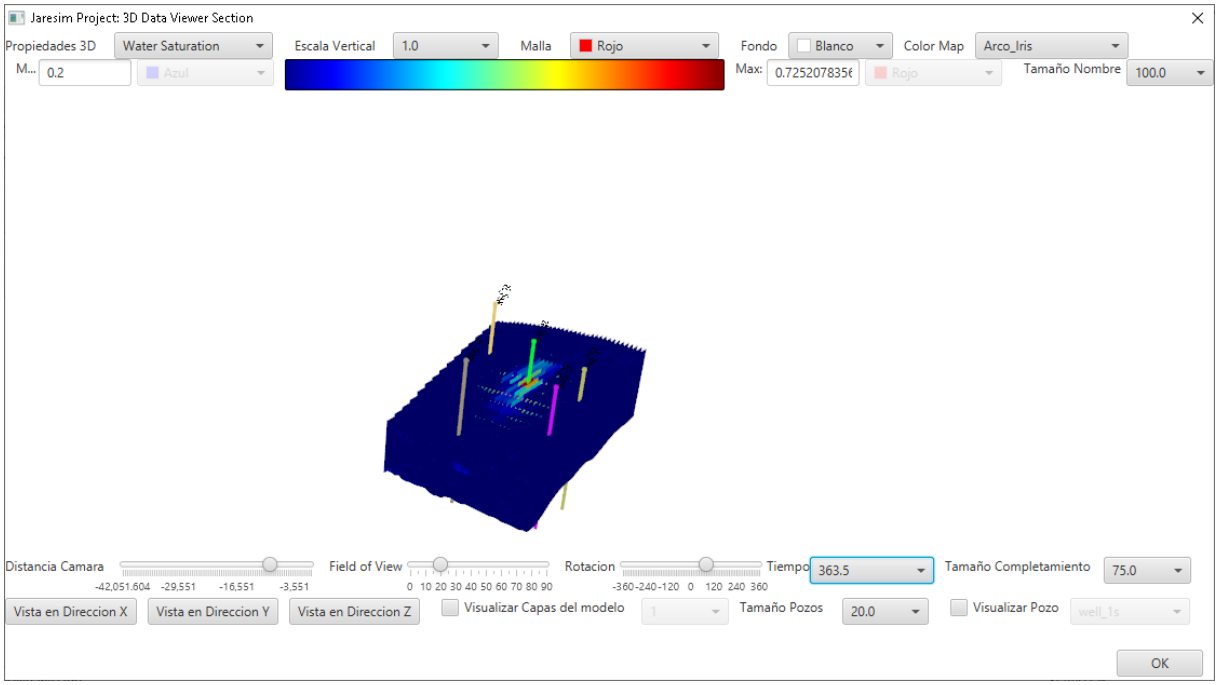


Ilustración 129:Gráfico de Distribución 3D de Saturación de Agua a los 364 días de Simulación para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

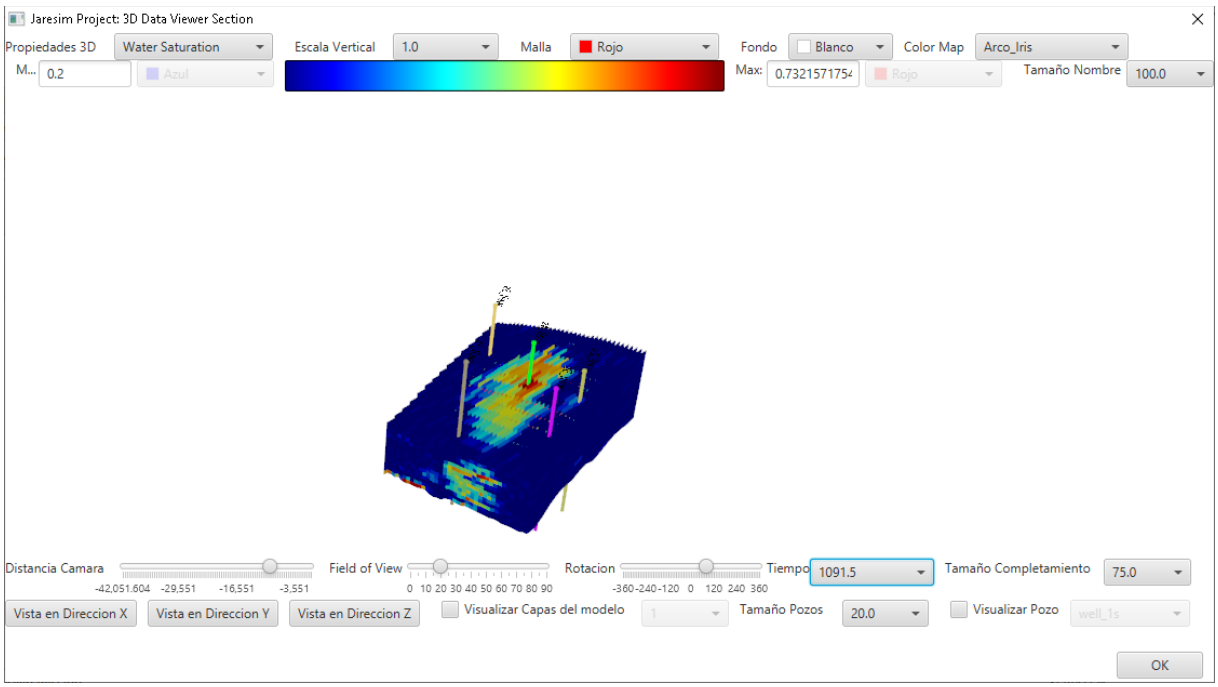


Ilustración 130:Gráfico de Distribución 3D de Saturación de Agua a los 1092 días de Simulación para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

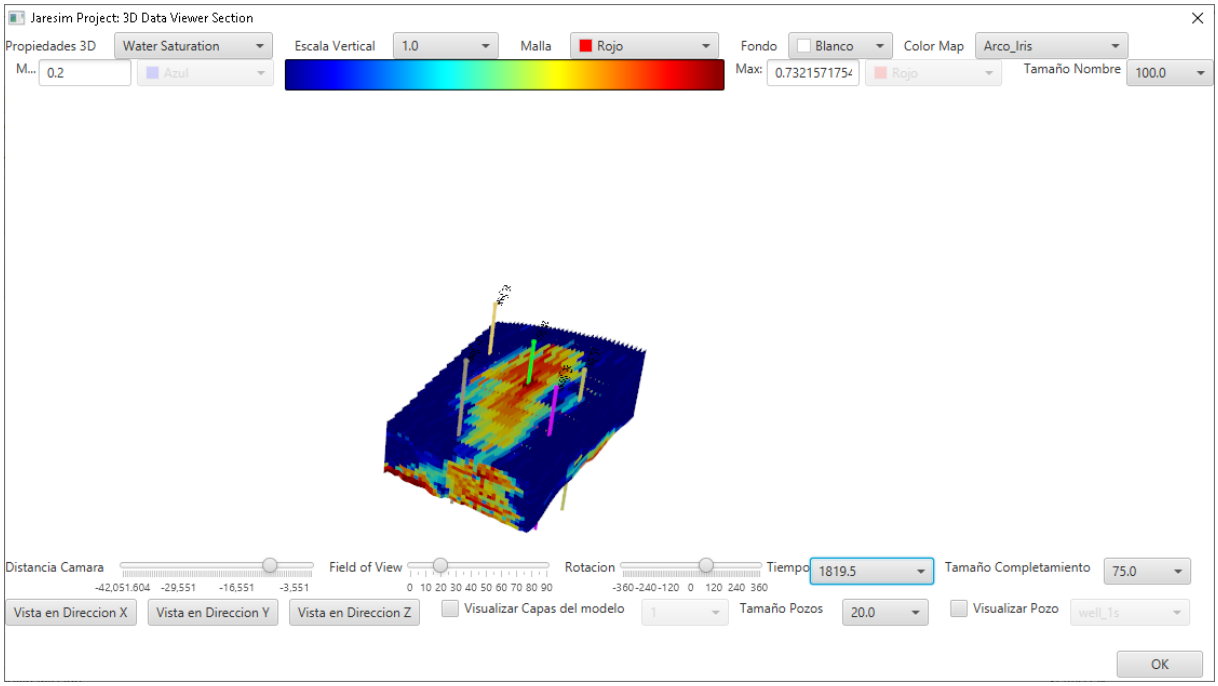


Ilustración 131:Gráfico de Distribución 3D de Saturación de Agua a los 1820 días de Simulación para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

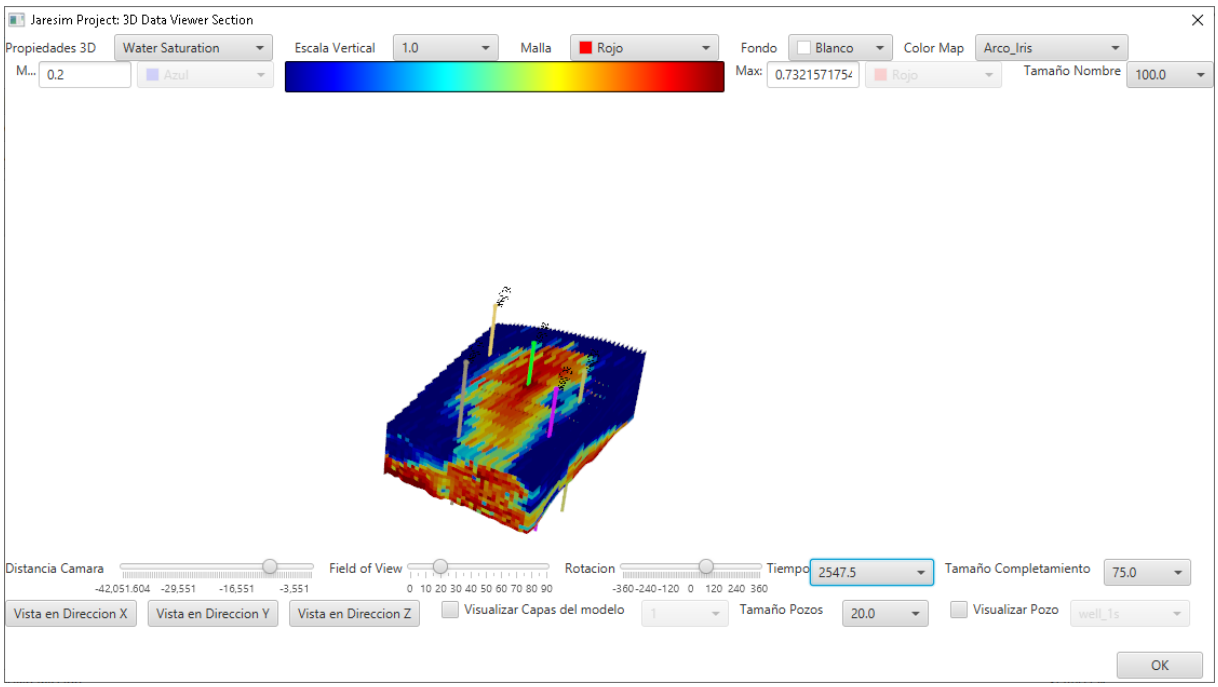


Ilustración 132:Gráfico de Distribución 3D de Saturación de Agua a los 2548 días de Simulación para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

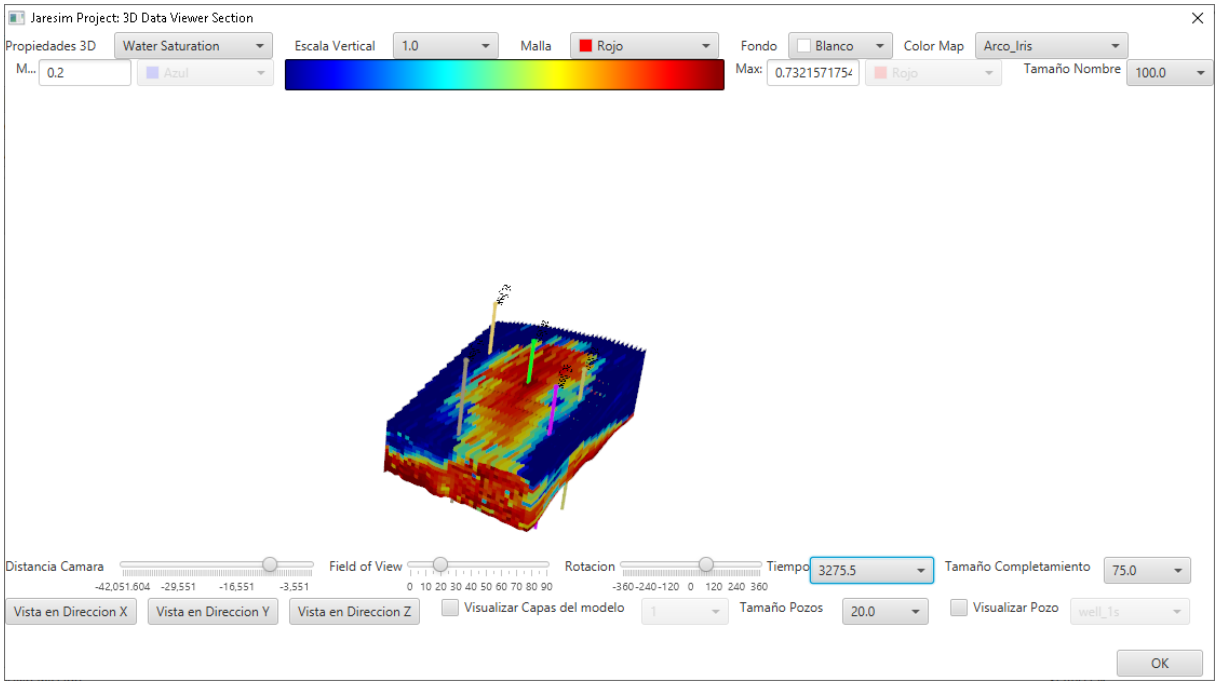


Ilustración 133:Gráfico de Distribución 3D de Saturación de Agua a los 3276 días de Simulación para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

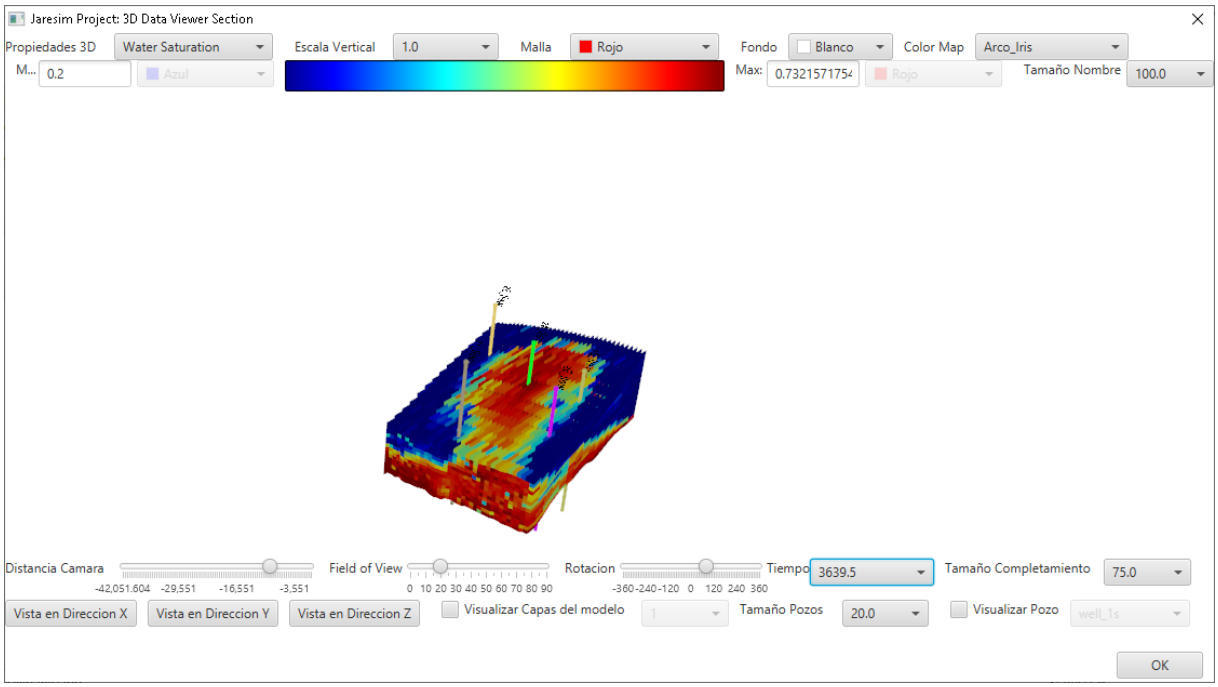


Ilustración 134:Gráfico de Distribución 3D de Saturación de Agua a los 3640 días de Simulación para el caso de Validación #2 mediante el simulador desarrollado en esta investigación. Fuente: Elaboración propia.

Los resultados previamente mostrados, permiten ver que la aplicación es capaz de evidenciar los resultados que usualmente son presentados en un simulador típico comercial.

10.5. Definición de Malla de Simulación Numérica

Es una aproximación numérica que consiste en dividir en un número finito de Unidades de Volumen Discretas (Celdas) un yacimiento, lo cual permite modelar las estructuras y propiedades presentes en el mismo (Ramos R, 2008).

10.5.1. Tipos de Mallas de Simulación Numérica

Dos tipos de mallas son usadas generalmente, y estas son (Ramos R, 2008):

- Mallas regulares: Tienen espaciamiento uniforme en la dirección X, Y.
- Mallas irregulares: Tiene espaciamiento no uniforme en la dirección X, Y.

De acuerdo a la construcción de la malla, ésta puede ser definida bajo distintas geometrías tales como Corner Point (Punto de esquina), Block Centered (Bloque centrado) en el caso de mallas estructuradas, y mallas PEBI (Perpendicular Bisector) en el caso de mallas un-estructuradas.

Las mallas con geometría bloque centrado requieren para cada celda un valor para el tope, al igual que el tamaño de la celda en dirección X, Y, Z. Los parámetros calculados por las distintas ecuaciones otorgan valores referentes al centro de la celda o bloque (Ramos R, 2008).

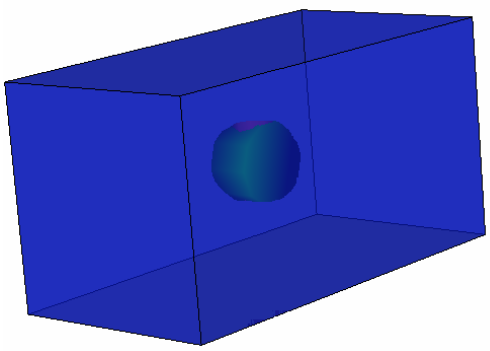


Ilustración 135:Celda de geometría Block Centered. Fuente: (Ramos R, 2008).

Algunas representaciones que muestran la manera en que la geometría de bloque centrada trabaja, son las siguientes:

- Definición de las conexiones vecinas, en las cuales considera como celdas contiguas aquellas cuyos índices sean consecutivos y no necesariamente se encuentren físicamente en contacto.

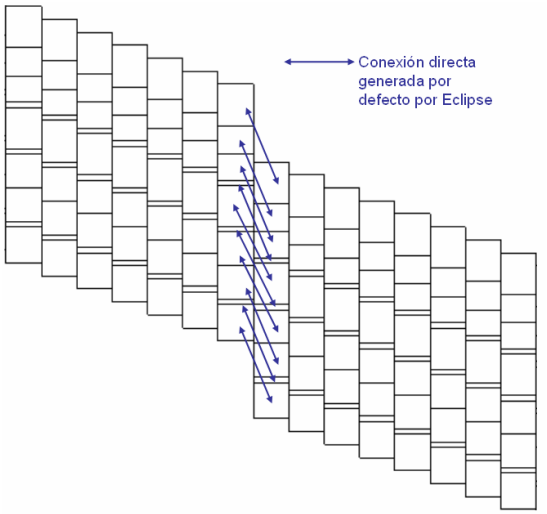


Ilustración 136:Definición de Conexiones No Vecinas para bloque centrado. (Ramos R, 2008).

- Ejemplo de malla con geometría bloque centrado.

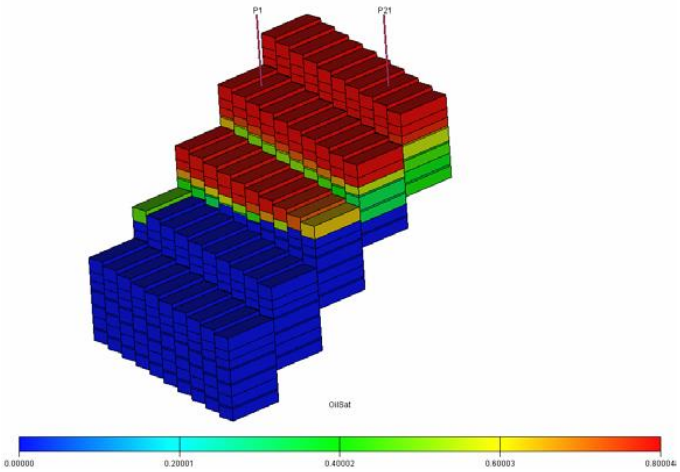


Ilustración 137:Malla con geometría bloque centrado. Fuente: (Ramos R, 2008).

La geometría Corner Point (punto de esquina) está basada en líneas de coordenadas y en profundidades en las que se encuentran las esquinas cada celda. Las coordenadas x, y, z de un punto arriba y un punto debajo de la malla define una línea coordenada o pilar, las celdas son definidas por la unión de las esquinas de las mismas y la elevación es definida con respecto a las líneas coordenadas (Ramos R, 2008):

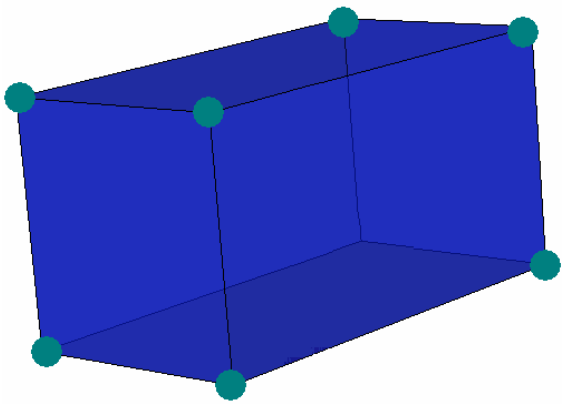


Ilustración 138:Malla con geometría Corner point. Fuente: (Ramos R, 2008).

A continuación, se ilustra la manera en que trabajan las conexiones vecinas de este tipo de geometría, donde se considera el flujo entre celdas físicamente vecinas, es decir entre las celdas cuyas caras están en contacto, esto es:

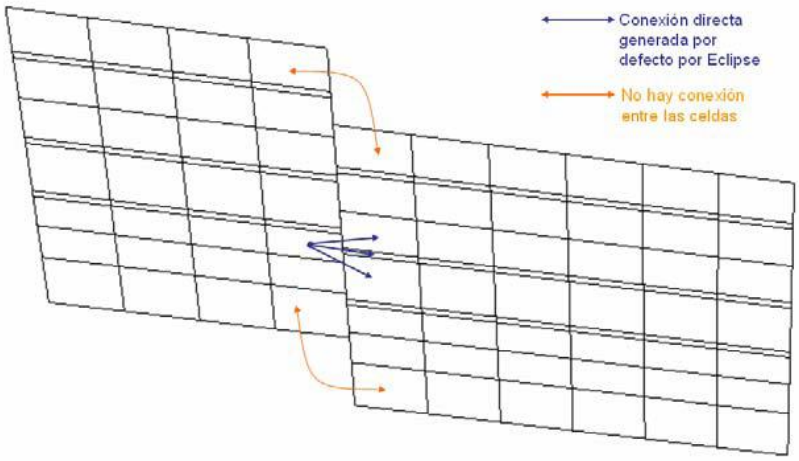


Ilustración 139:Definición de conexiones no vecinas para corner point. Fuente: (Ramos R, 2008).

La siguiente figura, ilustra la definición del mismo modelo mostrado anteriormente con geometría bloque centrado, y aquí se aprecia una figura más suavizada que permite un modelado de la estructura del yacimiento más preciso:

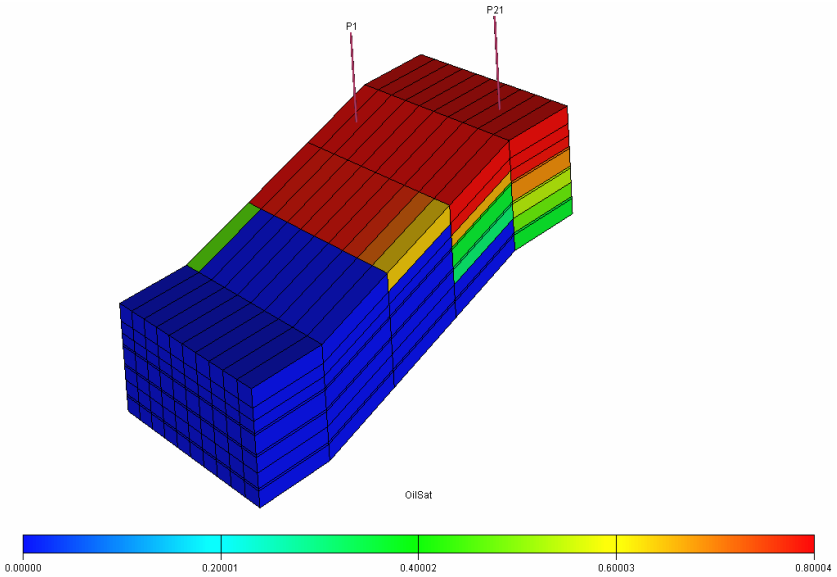


Ilustración 140: Modelo de malla con geometría corner point. Fuente: (Ramos R, 2008).

10.5.2. Componentes Principales de una Malla 3D de Tipo Corner Point.

Los componentes principales de una malla corner point son los siguientes (Ramos R, 2008):

- Líneas de coordenadas.
- Tubos de coordenada (Que tienen líneas coordinadas como sus bordes).
- Bloques de malla (Obtenidos cortando los tubos de coordenada con los horizontes).

Las líneas de coordenadas son líneas paralelas, verticales o inclinadas y rectas dibujadas en dirección hacia abajo de las Esquinas de los cuadriláteros en una malla areal estructurada:

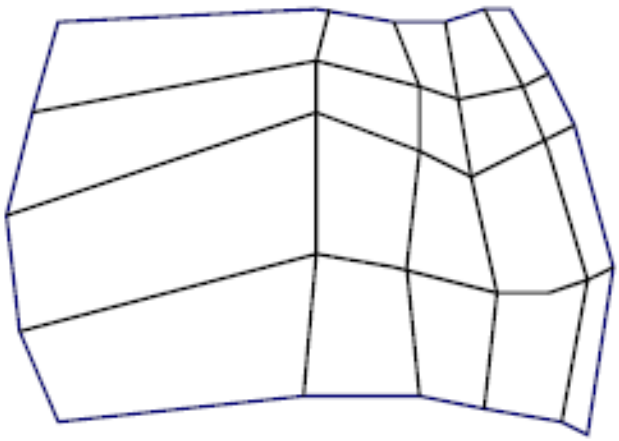


Ilustración 141:Ejemplo de una malla estructurada de cuadriláteros. Fuente: (Ramos R, 2008).

Los tubos de coordenadas son una extensión vertical de los cuadriláteros de una malla areal estructurada:

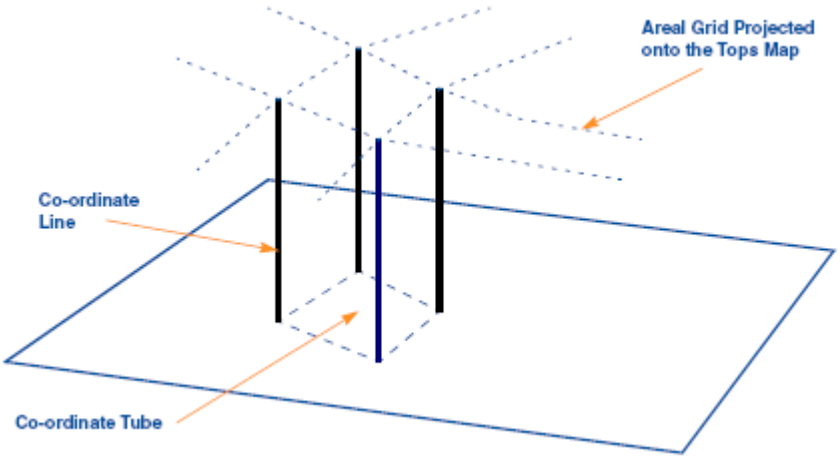


Ilustración 142: Tubos de coordenadas con líneas de coordenadas como paredes. Fuente: (Ramos R, 2008).

Los Bloques de Malla son tubos de coordenada que son cortados por horizontes y son definidos por los cuatro puntos de las esquinas del bloque de malla. Si hay n+1 horizontes que cortan cada tubo entonces hay n bloques de malla en dicho tubo de coordenada.

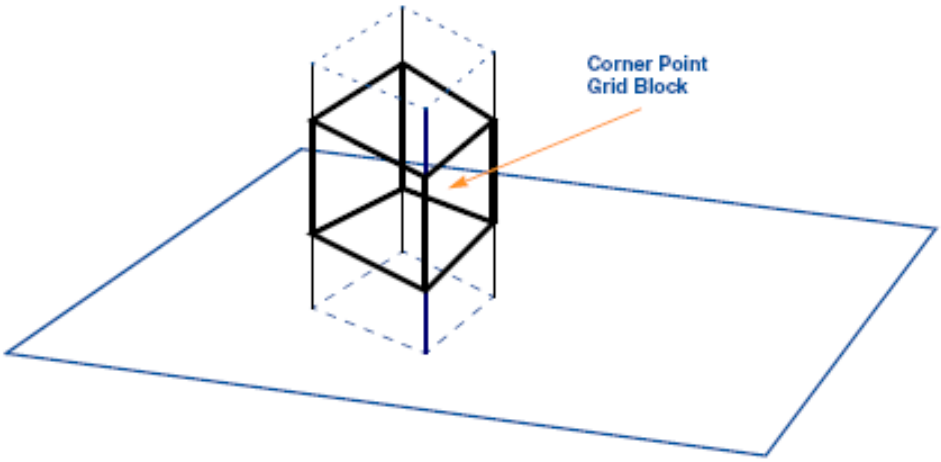


Ilustración 143:Bloques de malla de geometría corner point. Fuente: (Ramos R, 2008).

10.5.3. Formato ASCII para Mallas 3D ECLIPSE™ de Tipo Corner Point.

La malla de Eclipse™ en formatos ASCII (*. GRDECL, (ASCII) (*.*)) está formado por algunas palabras claves (Keywords) y valores. Los Keywords, deben ser escritos en mayúsculas y la secuencia de valores dados para un keyword debe ser seguido por una barra oblicua o slash (/). Como ejemplo, se describe lo siguiente (Ramos R, 2008):

SPECGRID
Nx Ny Nz 1 F /

En esta línea se lee con los primeros tres números:

1. El Número de Celdas en la dirección X.
2. El Número de Celdas en la dirección Y.
3. El Número de Celdas en la dirección Z.

Para efectos de esta investigación se considerará que el archivo solo contendrá información de un yacimiento, el sistema de unidades es en inglés y el sistema de coordenadas será cartesiano donde los sentidos de los ejes ortogonales propuestos para el proyecto se observan en la siguiente figura:

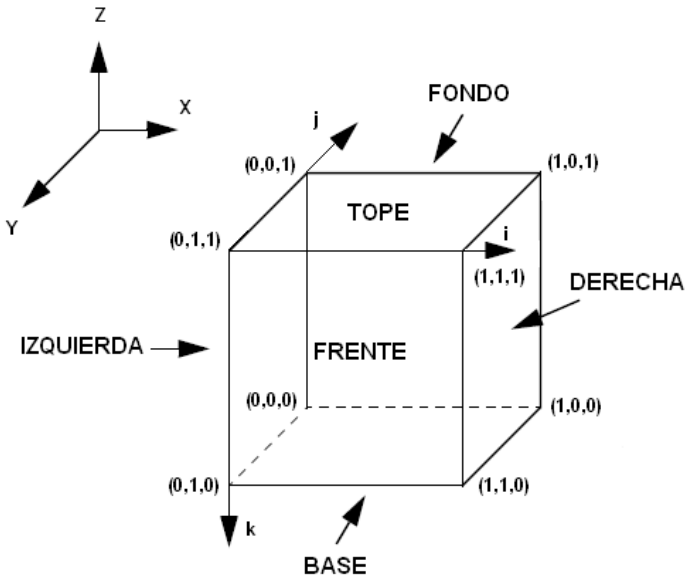


Ilustración 144:Sistema de Referencia utilizado. Fuente: (Ramos R, 2008).

Considerando lo anteriormente descrito, se continúa cargando el resto de la información de encabezamiento, y se buscar el siguiente Keyword denominado COORD. Este keyword, indica que el próximo bloque de datos contiene la información sobre el esqueleto de la malla. Los datos empiezan en la esquina izquierda superior de la misma, y cada línea contiene valores para XYZ mínimos y máximos (ver Ilustración 145). El bloque de datos termina con un Slash (/). (Ramos R, 2008)

COORD
X1 Y1 Z1 Z2 Y2 Z2
. . . . . .

El número total de líneas de coordenadas viene dado por:$(n_x + 1) * (n_y + 1)$, donde los pilares de índices (i, j) serán los mismos para todas las celdas correspondientes a los índices mencionados, en las diferentes estratificaciones o capas del modelo:

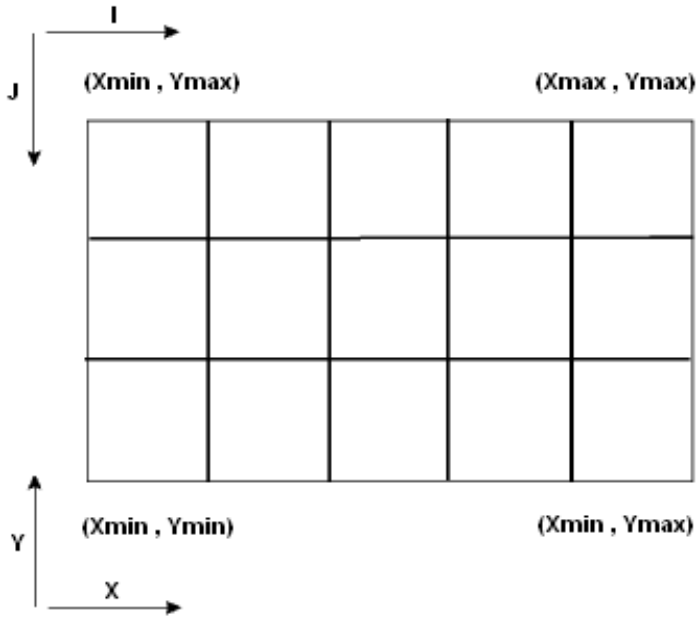


Ilustración 145:Ordenamiento de los datos del keyword COORD. Fuente: (Ramos R, 2008)

ZCORN constituye el siguiente keyword a ser leído. Este bloque de datos consta de coordenadas de Z, que da el Inter capeado o la estratificación de la malla. Cada capa es leída desde arriba hacia abajo. Si las coordenadas en la dirección Z son diferentes, son considerados dos horizontes

```
 Y
   ^
   |
3.0+    1-----2-----3-----4      COORD node numbering
   |    |1   2|3   4|5   6|      ZCORN node numbering
   |    |7   8|9  10|11 12|      ZCORN node numbering
2.0+    5-----6-----7-----8      COORD node numbering
   |    |13 14|15 16|17 18|      ZCORN node numbering
   |    |19 20|21 22|23 24|      ZCORN node numbering
1.0+    9----10----11----12      COORD node numbering
   |
   |
   +----+-----+-----+-----+---> X
0.0    1.0   2.0   3.0   4.0
```

*Ilustración 146:Formato de almacenamiento de los Keywords ZCORN y COORD en archivos *. GRDECL. Fuente: (Ramos R, 2008).*

There are (nx+1)*(ny+1) = COORD 12 nodes

COORD

```
1 3 0   1 3 0
2 3 0   2 3 0
3 3 0   3 3 0
4 3 0   4 3 0

1 2 0   1 2 0
2 2 0   2 2 0
3 2 0   3 2 0
4 2 0   4 2 0

1 1 0   1 1 0
2 1 0   2 1 0
3 1 0   3 1 0
4 1 0   4 1 0
```

ZCORN

```
1   2 2   3 3   4

5   6 6   7 7   8
5   6 6   7 7   8

9  10 10 11 11 12

2   3 3   4 4   5

6   7 7   8 8   9
6   7 7   8 8   9

10 11 11 12 12 13
```

Ilustración 147: Ejemplos del Formato de salvado de datos de los Keywords COORD y ZCORN. Fuente: (Ramos R, 2008).

El siguiente keyword leído de un archivo de Eclipse™ es ACTNUM. Este comando es usado para describir celdas activas e inactivas, 1 es usado para indicar las celdas activas, 0 es usado para indicar las celdas inactivas. Si el archivo no contiene la información de ACTNUM, se considerará que todas las celdas leídas del archivo Eclipse™ se encuentran activas (Ramos R, 2008):

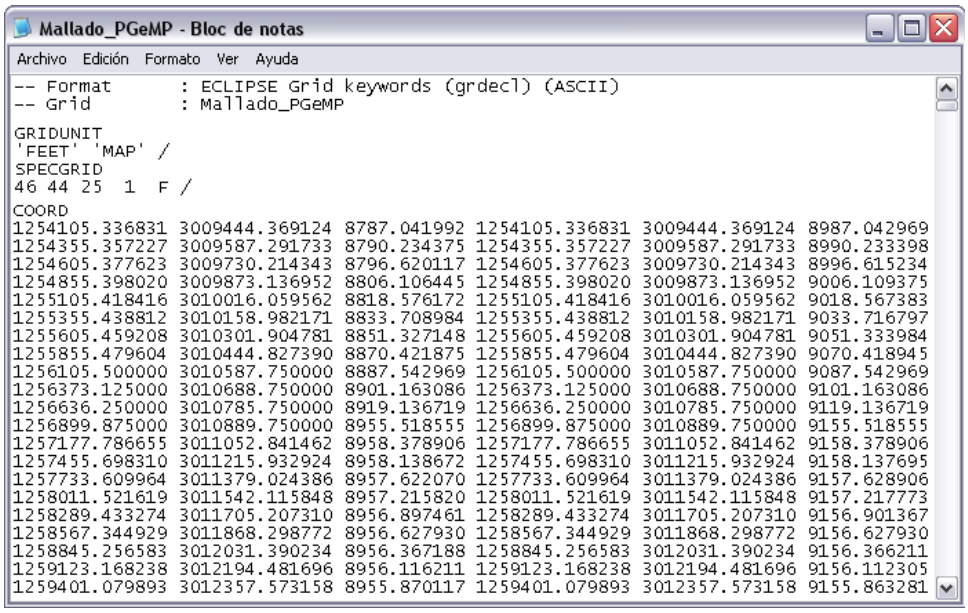


Ilustración 148:Archivo de Formato de malla 3D de Eclipse™. Fuente: (Ramos R, 2008).

El archivo de malla de Eclipse™ también puede contener modelos 3D de propiedades. El proceso de lectura del modelo 3D de una propiedad en el archivo, es el usado para los datos del esqueleto de malla 3D, pero la extensión usada típicamente del archivo de propiedades de Eclipse™ es properties (ASCII) (*.*). Los keywords utilizados en el proyecto son los siguientes (Ramos R, 2008):

- SWAT: Saturación de Agua.
- SGAS: Saturación de Gas.
- SOIL: Saturación de Petróleo.
- PORO: Porosidad.
- PERMX: Permeabilidad en la dirección X.
- PERMY: Permeabilidad en la dirección Y.
- PERMZ: Permeabilidad en la dirección Z.
- NTG: Net to Gross.

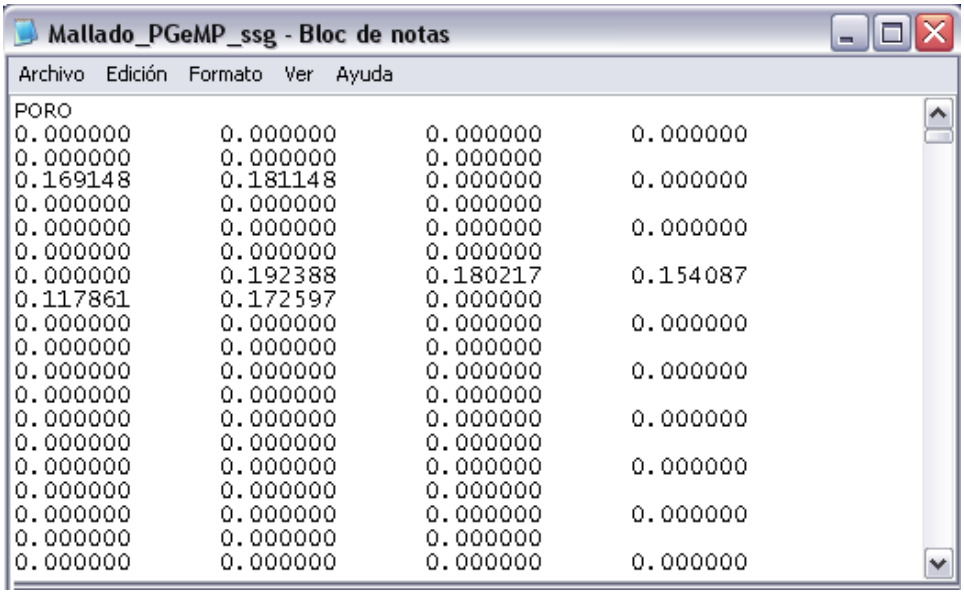


Ilustración 149: Archivo de formato ASCII de propiedades de Eclipse™. Fuente: (Ramos R, 2008).

10.6. Descripción de los diagramas de flujo de las secciones del Simulador desarrollado en la investigación

El módulo GPU del simulador desarrollado en esta investigación, tomó como referencia el proyecto llamado BOAST NFR (Almengor, y otros, June, 2006) como una versión de un simulador Black Oil, basado en el esquema IMPES para yacimientos naturalmente fracturados.

El programa desarrollado en esta investigación, corresponde a un simulador de Petróleo Negro Trifásico con formulación IMPES para régimen de flujo matricial el cual se codifico en JAVA realizando la activación de la librería JOCL, que permite el uso de las librerías OpenCL para procesamiento paralelo CPU-GPU. El simulador desarrollado tiene las capacidades especiales:

1. Dispone de 3 métodos de interpolación (Lineal, Polinomio de Lagrange, Polinomio de Newton).
2. Dispone de 4 métodos para la resolución de sistemas lineales (Algoritmo de Thomas, Método de Jacobi, Método de Gauss Seidel y Método de Gradiente Conjugado, los cuales tienen la capacidad de usar el algoritmo de sobre relajación lineal).
3. Dispone de Métodos de STONE I y STONE II para el modelaje de permeabilidades relativas trifásicas mediante dos sistemas bifásicos.
4. Dispone de 3 modelos de modelaje de presión de pozos (1 ,5 y 9 puntos).

El módulo está distribuido en una serie de sub módulos los cuales son descritos a continuación mediante el uso de diagramas de flujo, en los cuales está basado los códigos OpenCL de la aplicación:

10.6.1. Módulo Principal Jaresim GPU:

En la siguiente ilustración se puede evidenciar el diagrama de flujo principal de la clase GPU:

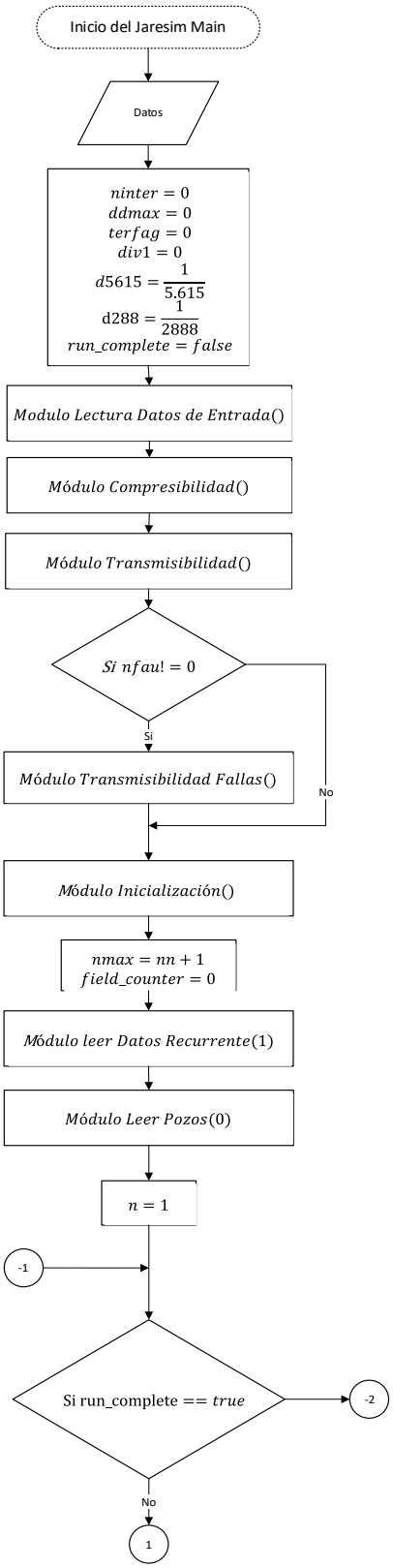


Ilustración 150:Diagrama de flujo de la Sección General. Fuente: Elaboración propia.

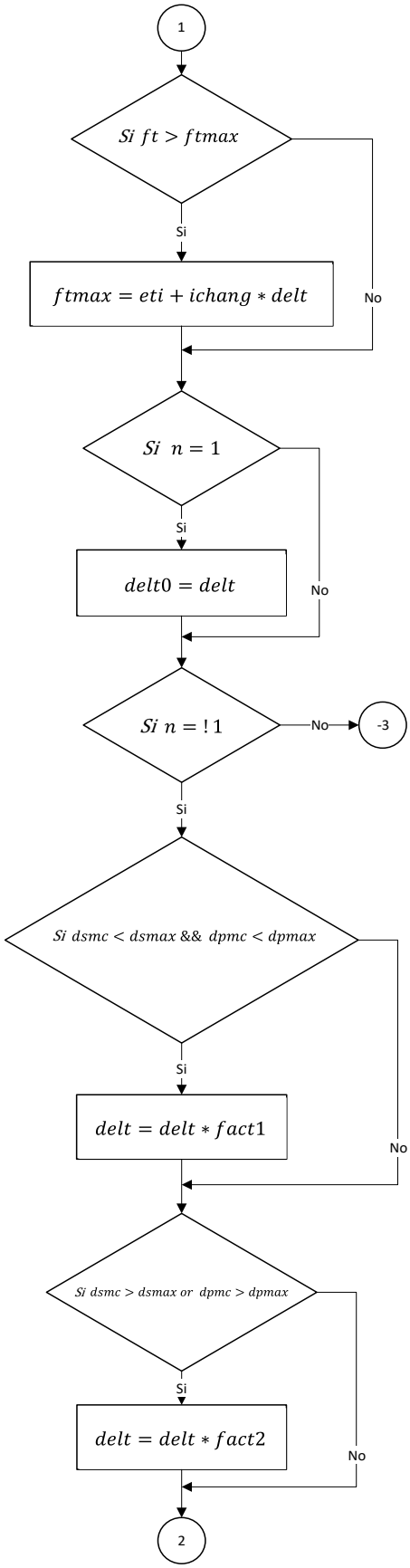


Ilustración 151:Diagrama de flujo de la Sección General-Continuación. Fuente: Elaboración propia.

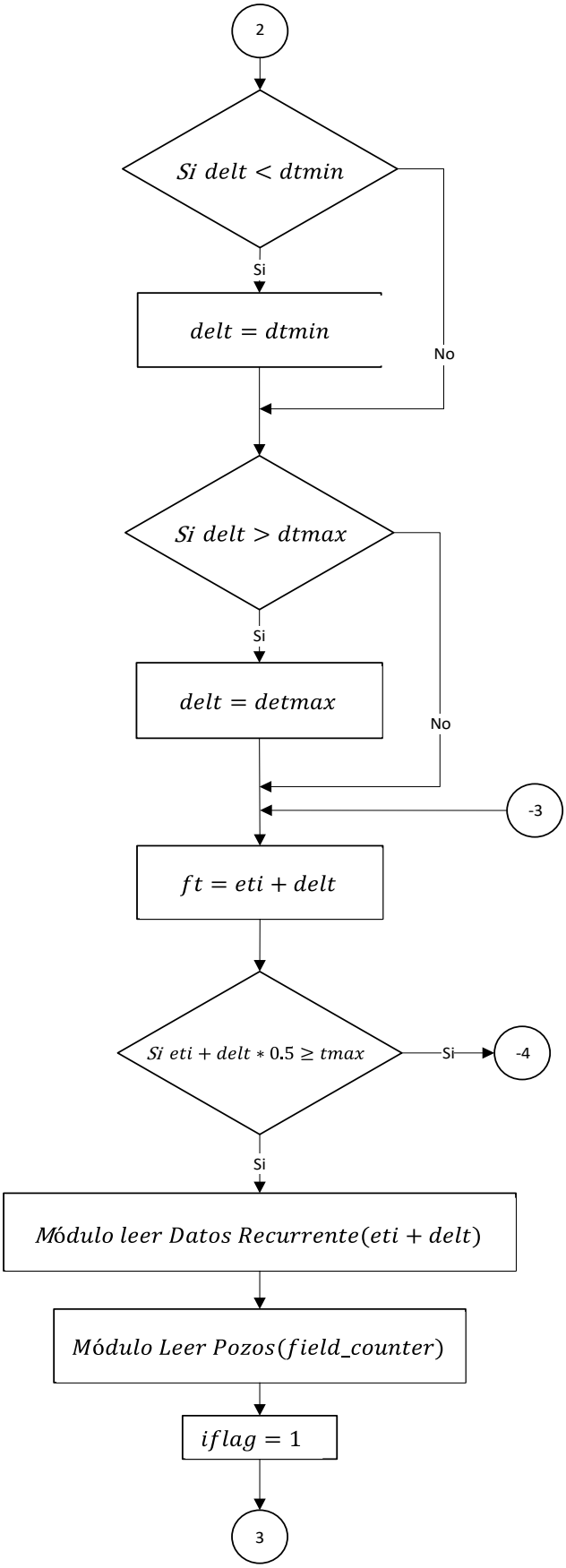


Ilustración 152:Diagrama de flujo de la Sección General-Continuación. Fuente: Elaboración propia.

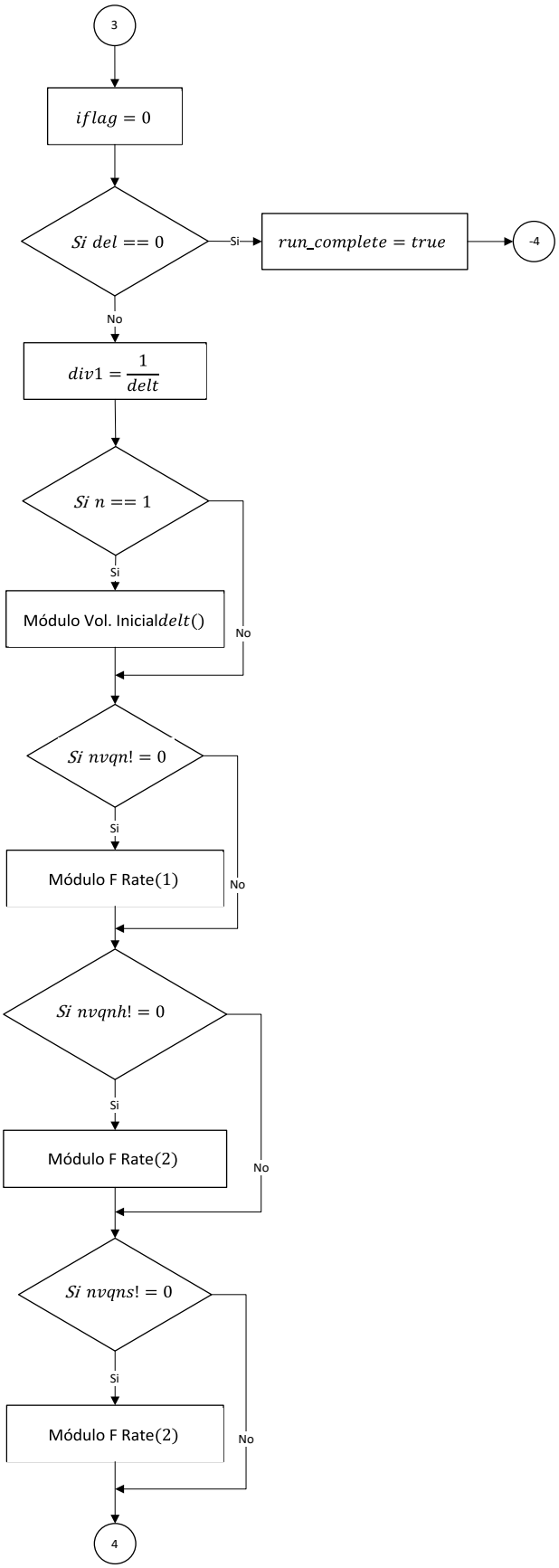


Ilustración 153:Diagrama de flujo de la Sección General-Continuación. Fuente: Elaboración propia.

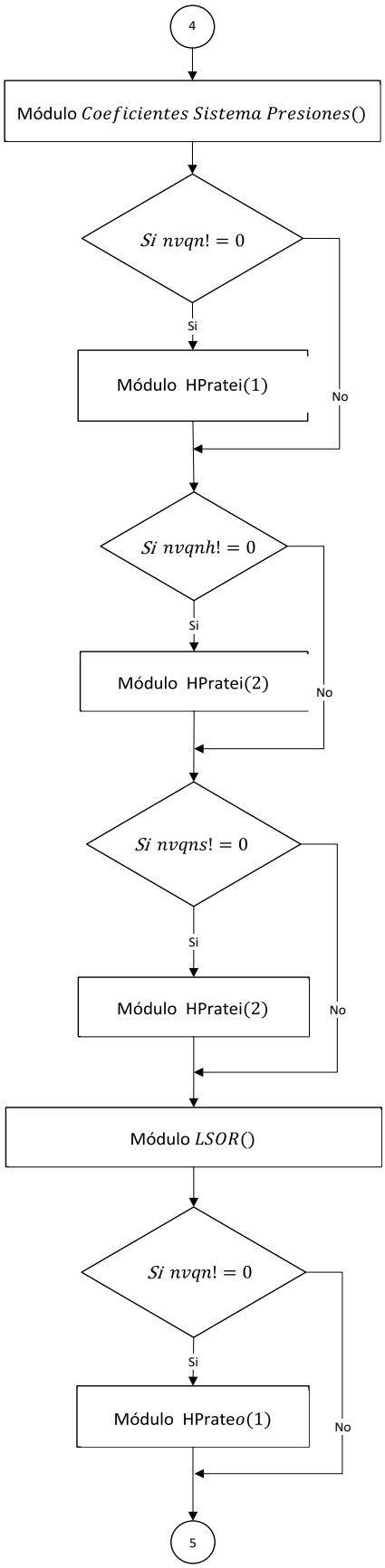


Ilustración 154:Diagrama de flujo de la Sección General-Continuación. Fuente: Elaboración propia.

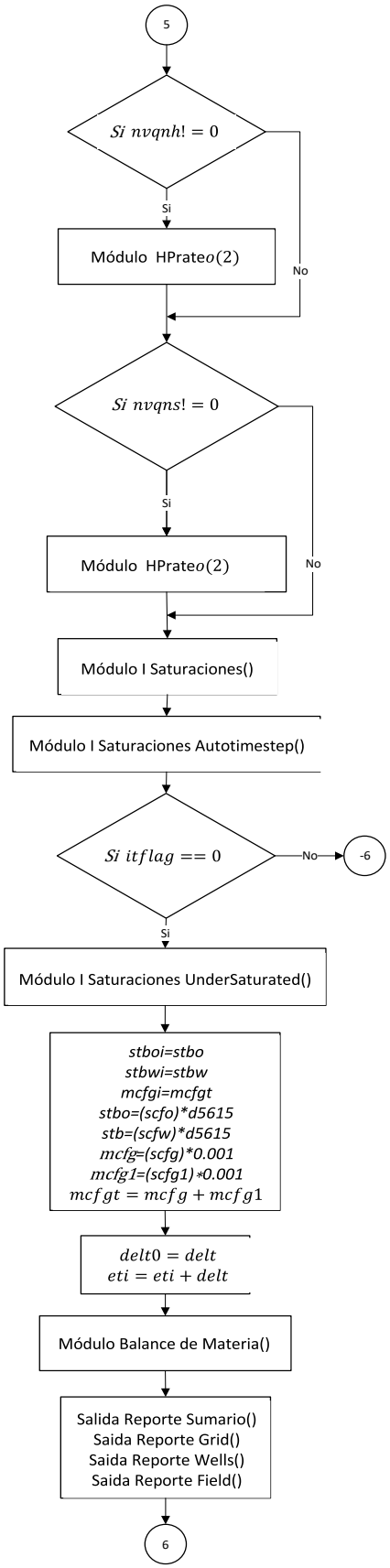


Ilustración 155:Diagrama de flujo de la Sección General-Continuación. Fuente: Elaboración propia.

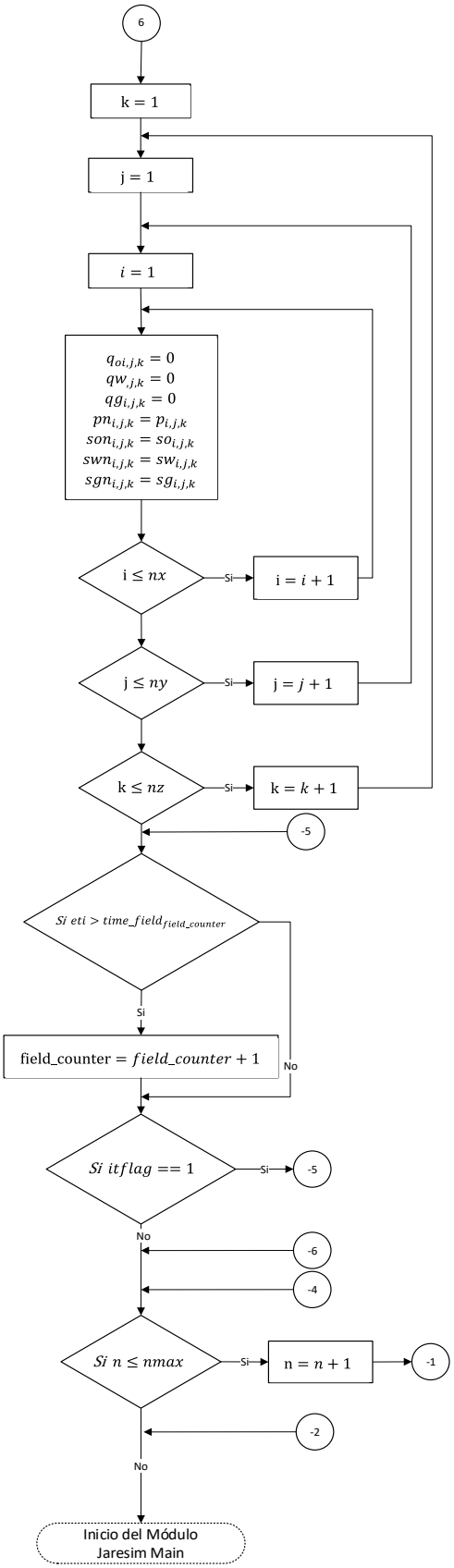


Ilustración 156:Diagrama de flujo de la Sección General-Continuación. Fuente: Elaboración propia.

10.6.2. Módulo de Cálculos de Compresibilidad:

En la siguiente ilustración se puede evidenciar el diagrama de flujo del módulo de Compresibilidad:

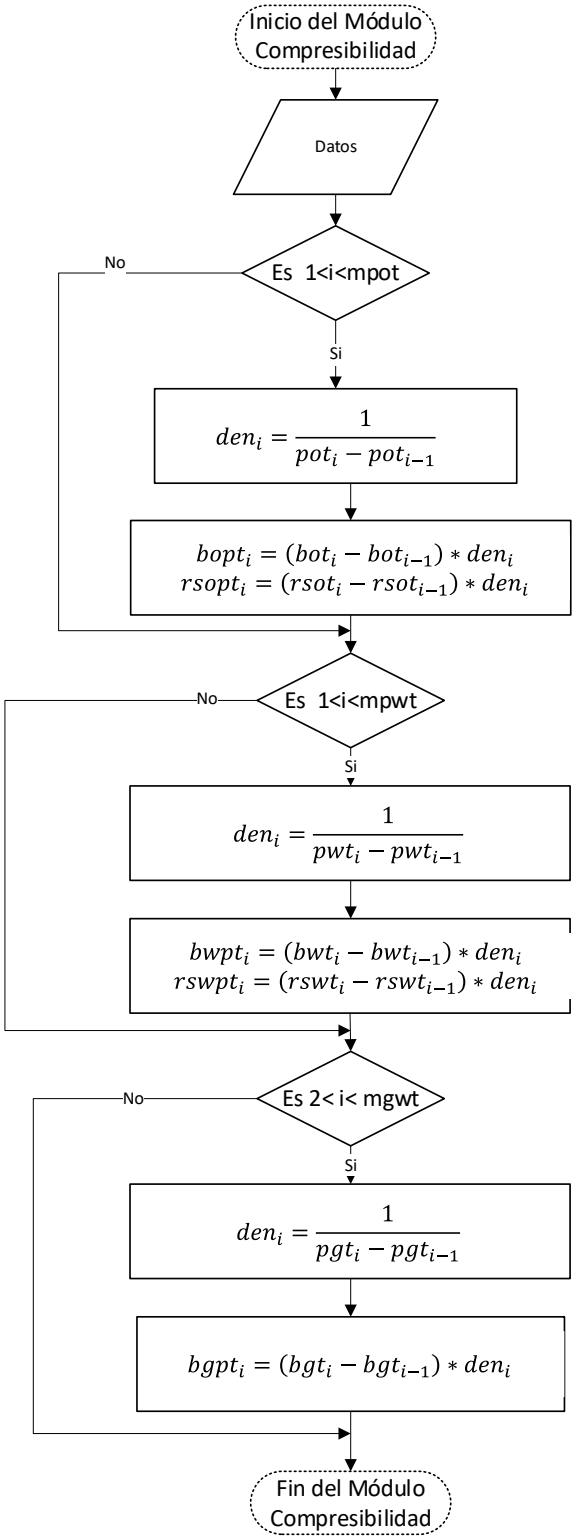


Ilustración 157:Diagrama de Flujo Sección Compresibilidad. Fuente: Elaboración propia.

10.6.3. Módulo de Cálculo de Transmisibilidad

La siguiente ilustración muestra el diagrama de flujo del módulo de los cálculos de transmisibilidad del modelo:

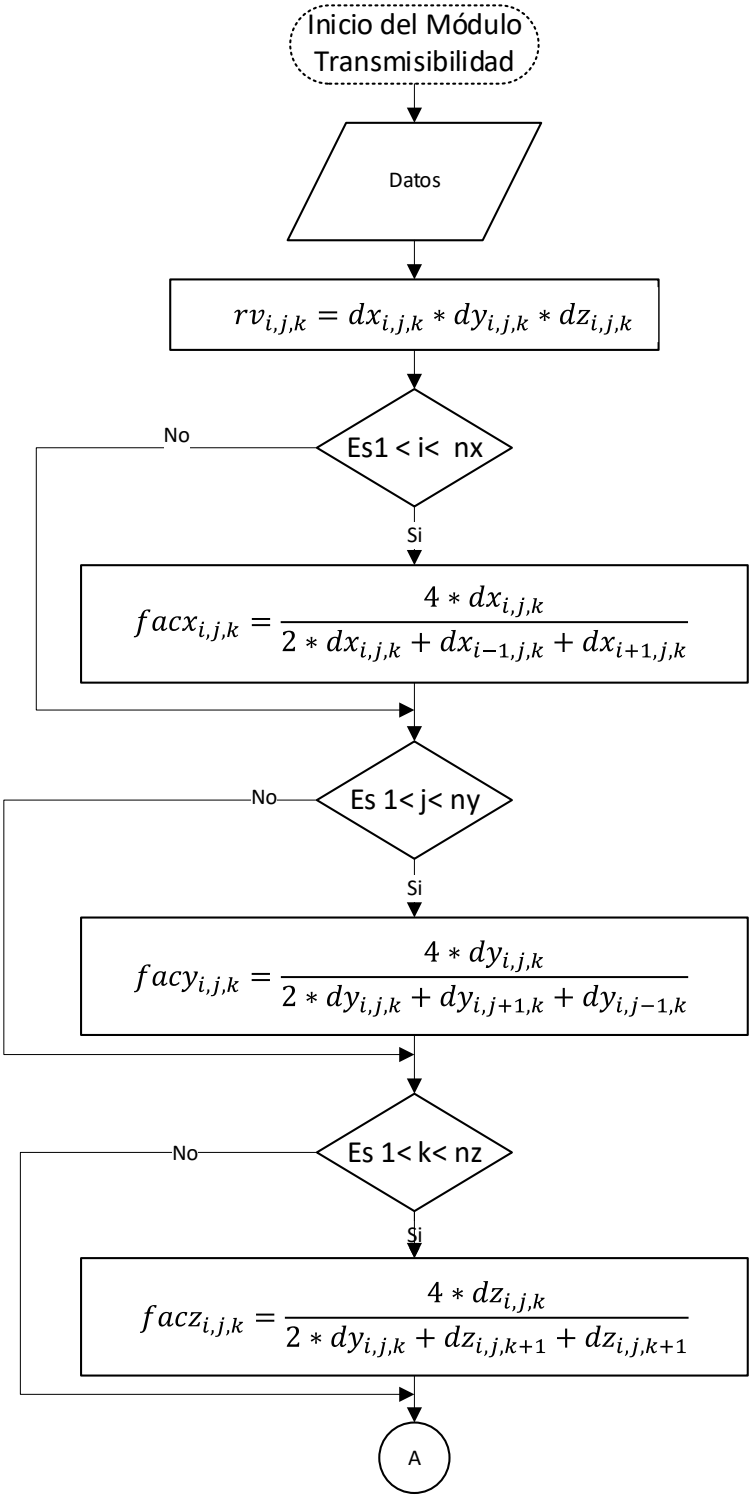


Ilustración 158:Diagrama de flujo de la Sección Transmisibilidad. Fuente: Elaboración propia.

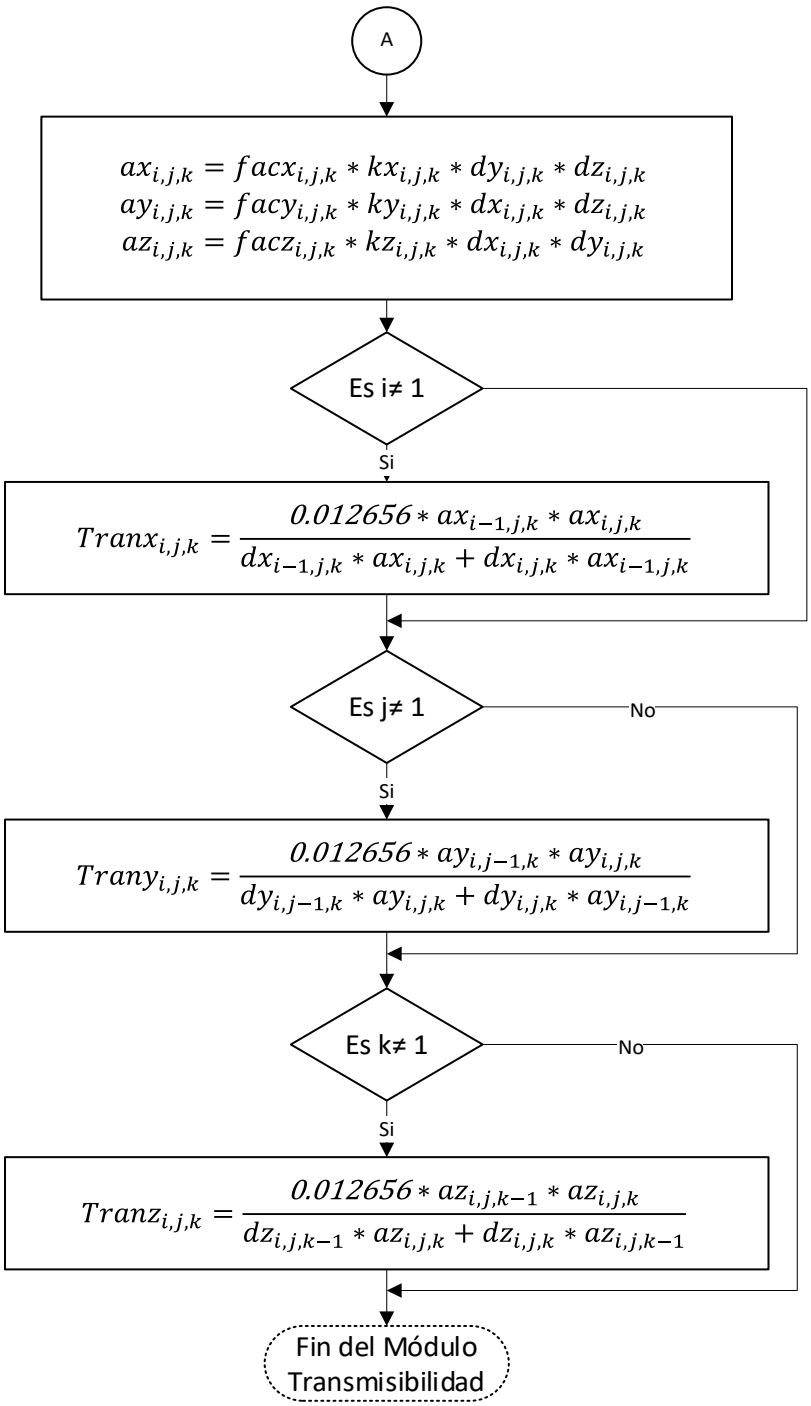


Ilustración 159:Continuación del Diagrama de Flujo de la Sección Transmisibilidad. Fuente: Elaboración propia.

10.6.4. Módulo de Inicialización:

En las siguientes ilustraciones se pueden evidenciar los diagramas de flujo del módulo de Inicialización:

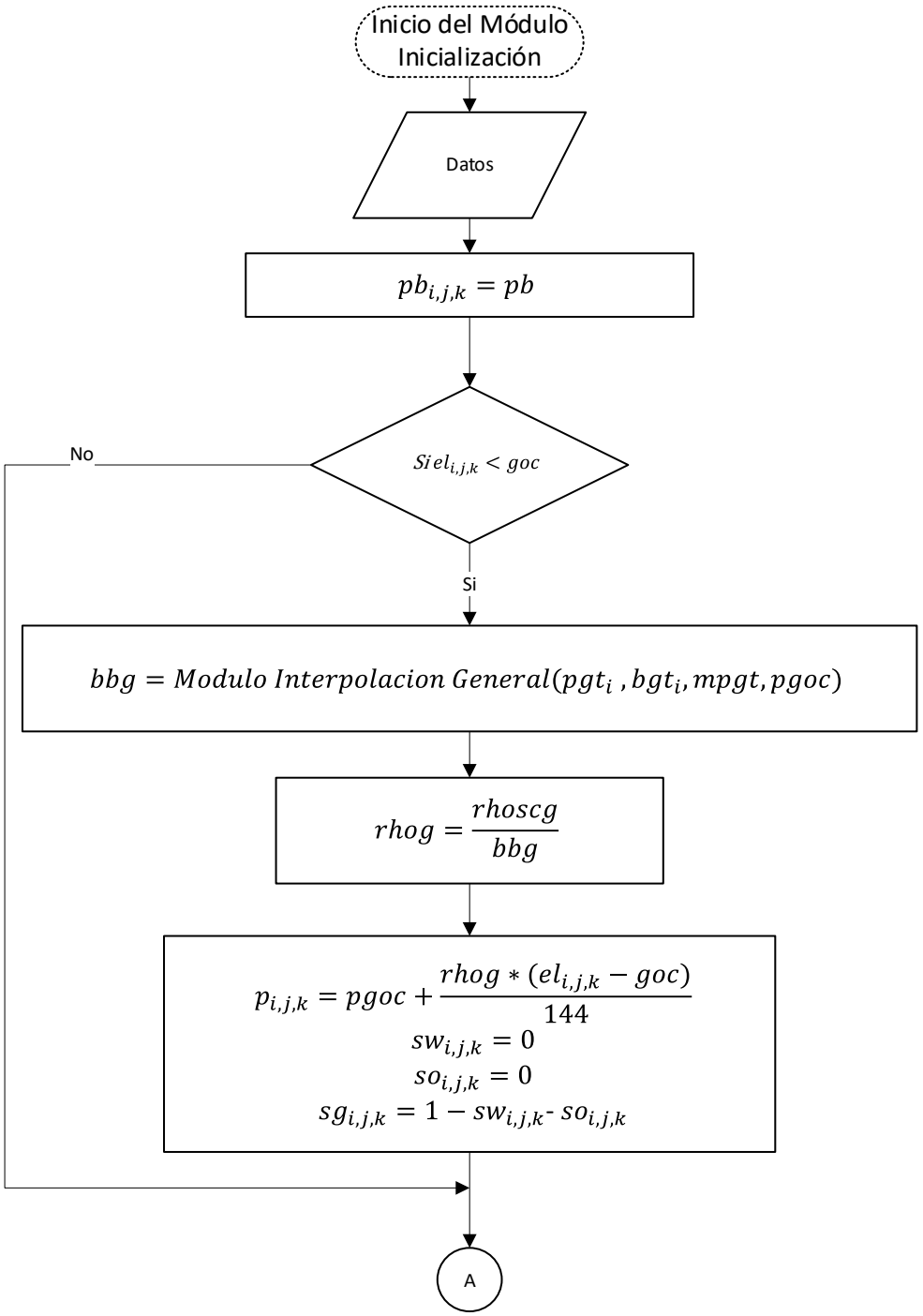


Ilustración 160:Diagrama de Flujo de Sección Inicialización. Fuente: Elaboración propia.

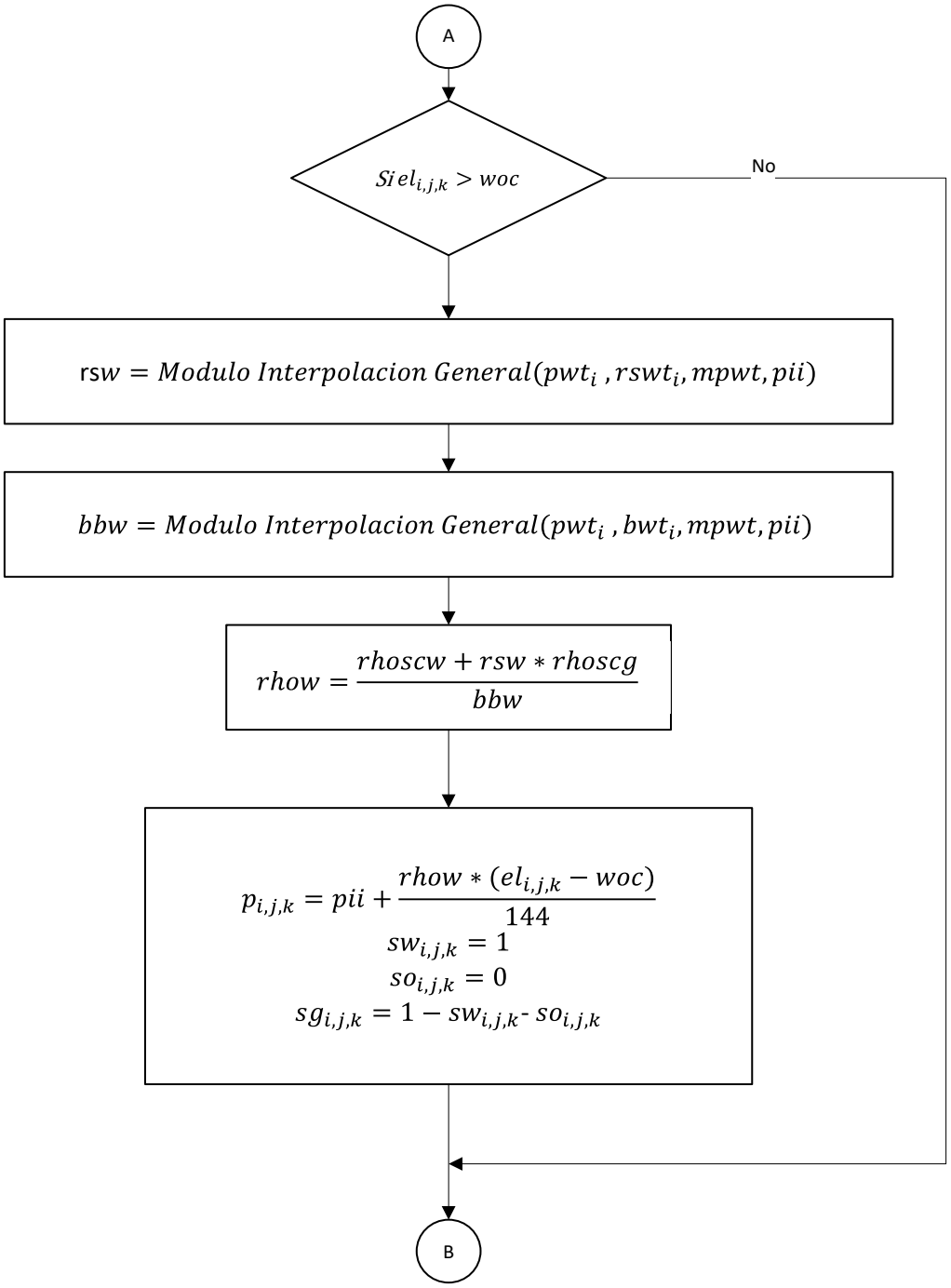


Ilustración 161:Continuación del Diagrama de Flujo de Sección Inicialización. Fuente: Elaboración propia.

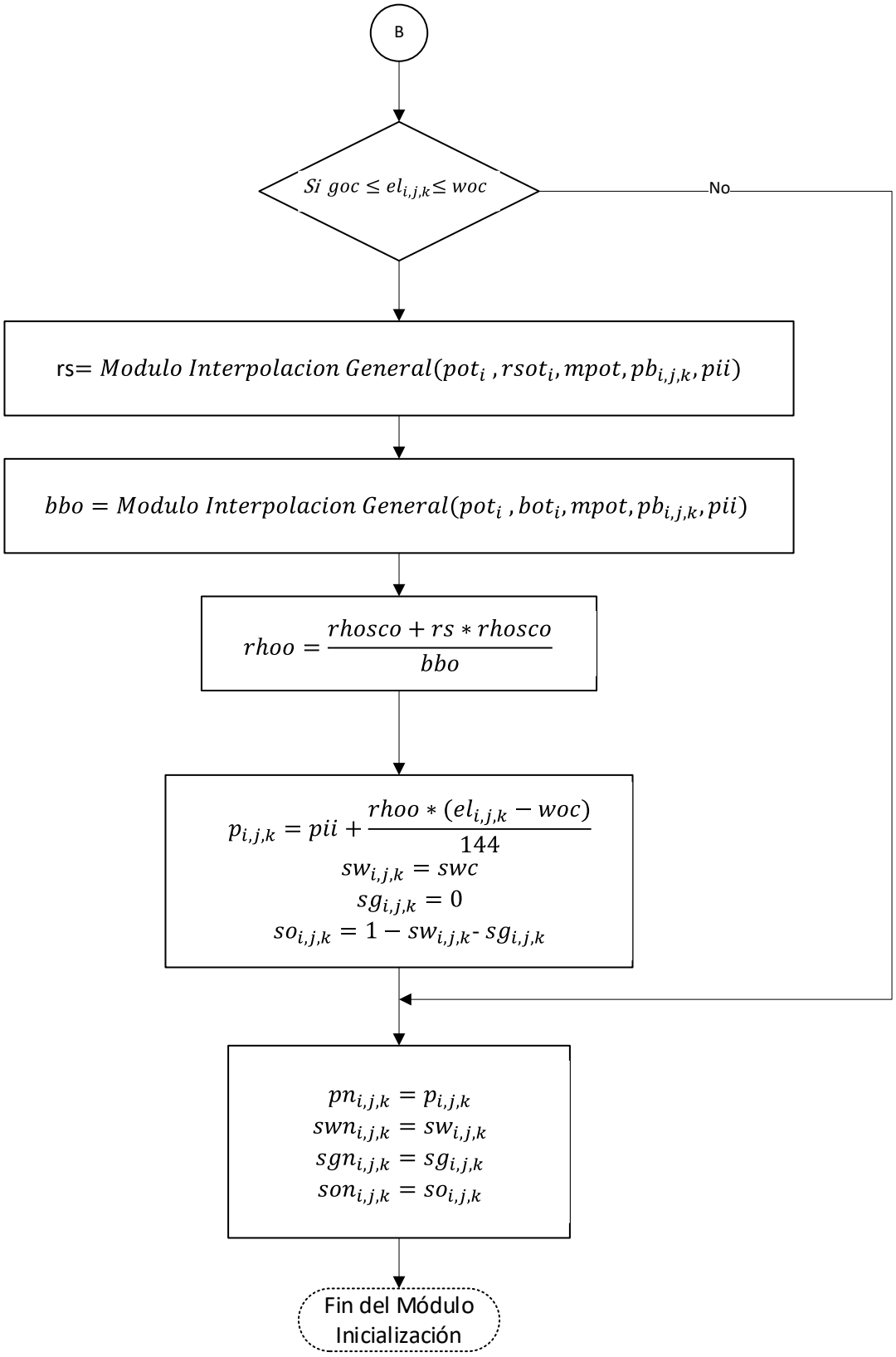


Ilustración 162:Continuación del Diagrama de Flujo de Sección Inicialización. Fuente: Elaboración propia.

10.6.5. Módulo de Cálculo de Volúmenes In Situ Originales

En las siguientes ilustraciones se pueden evidenciar los diagramas de flujo del módulo de Volúmenes in Situ:

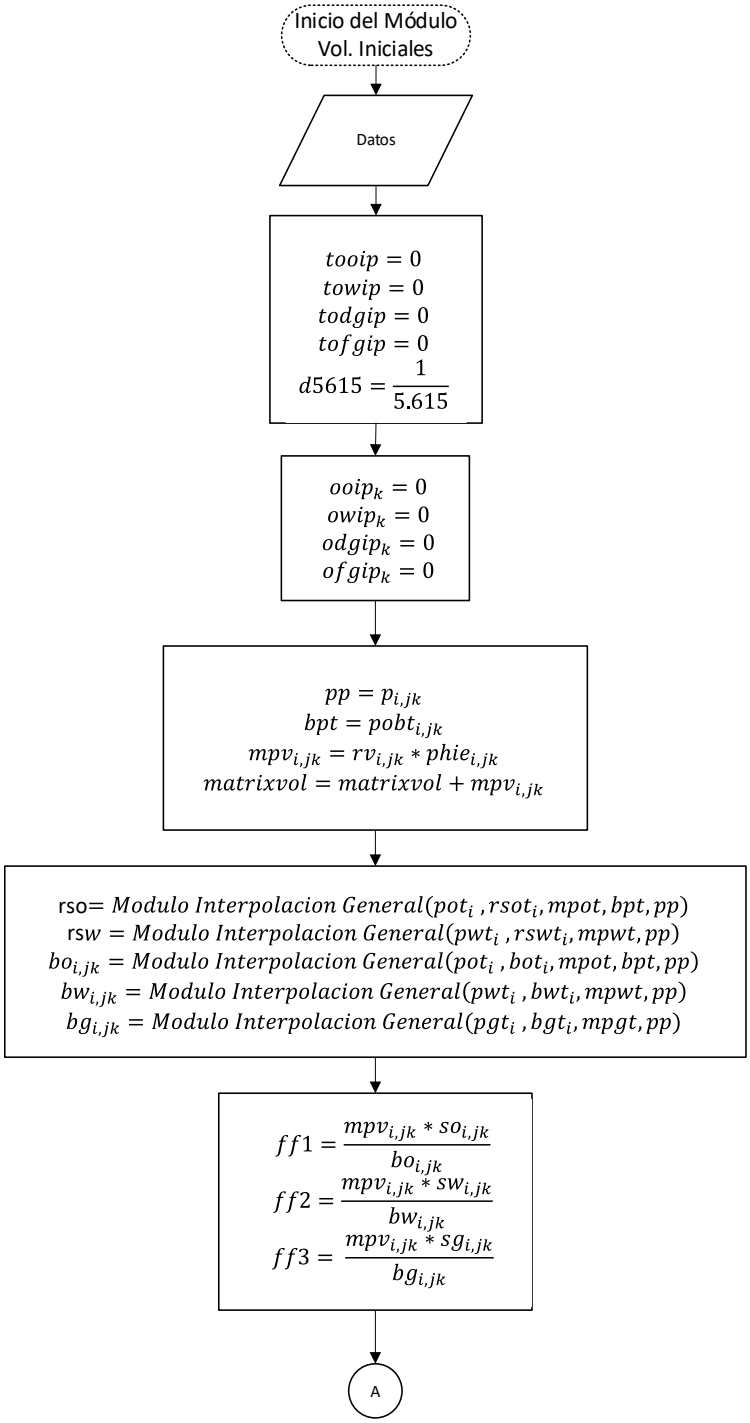


Ilustración 163:Diagrama de Flujo de la Sección Volúmenes in Situ. Fuente: Elaboración propia.

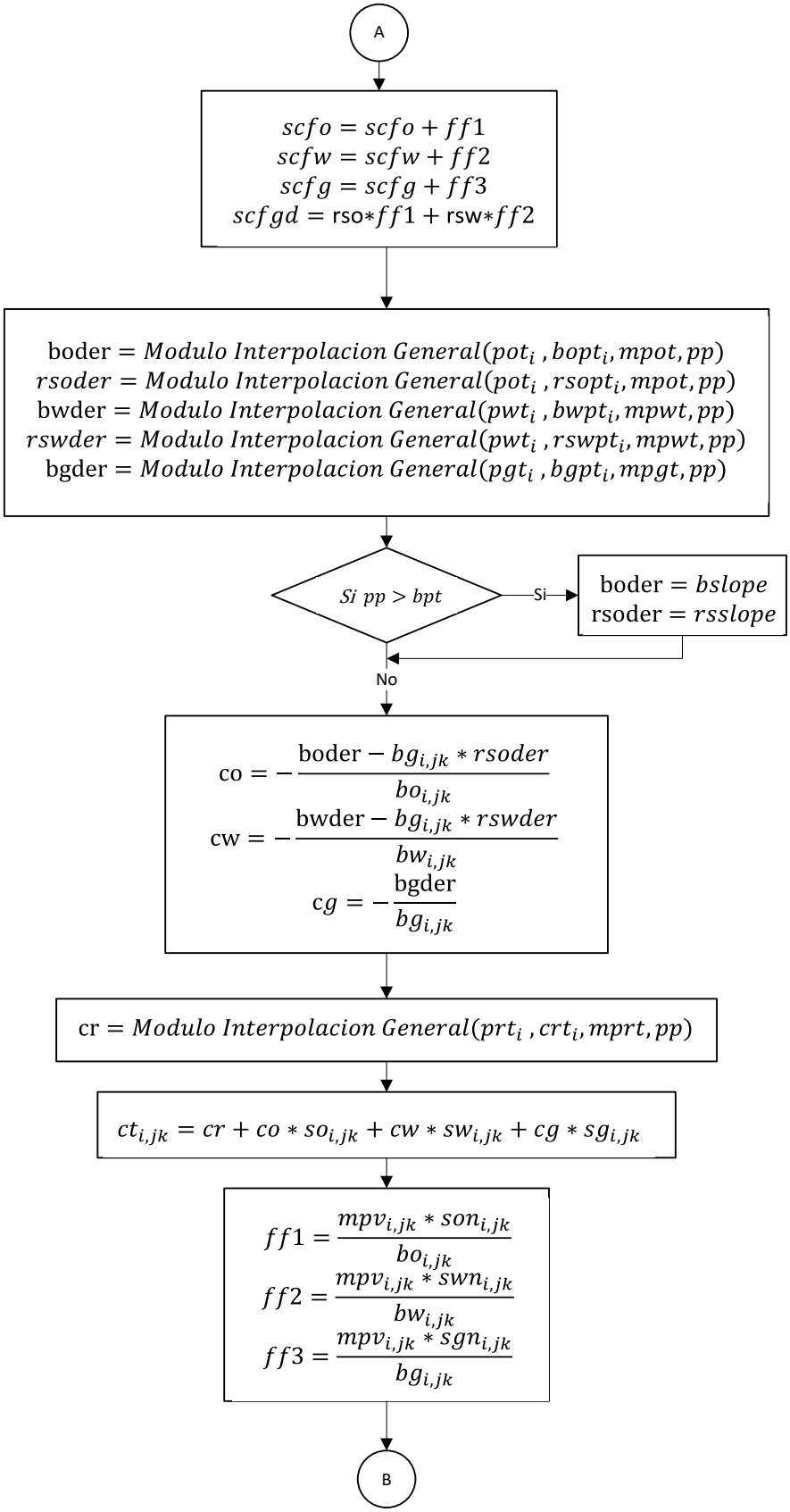


Ilustración 164:Continuación del Diagrama de Flujo de la Sección Volúmenes in Situ. Fuente: Elaboración propia.

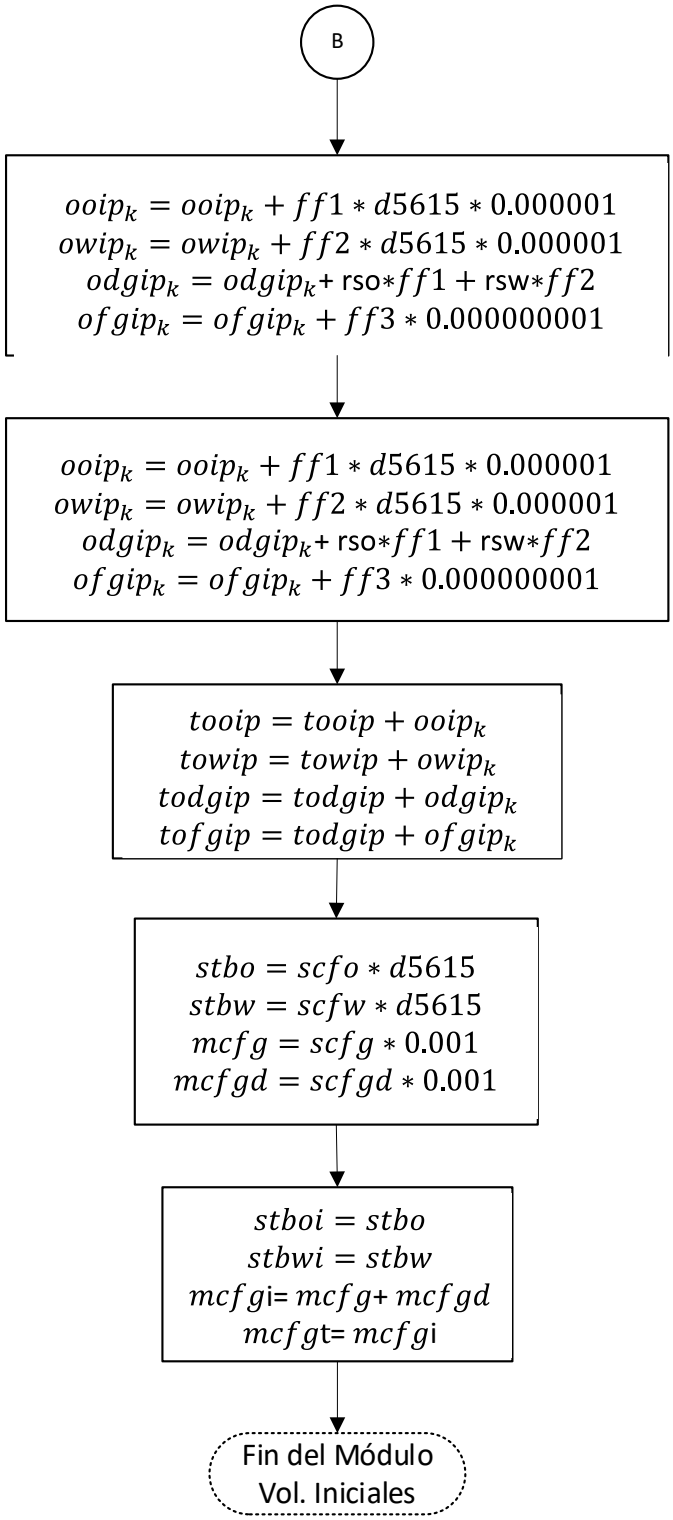


Ilustración 165:Continuación del Diagrama de Flujo de la Sección Volúmenes in Situ. Fuente: Elaboración propia.

10.6.6. Módulo de Cálculo de Balance de Materia

En las siguientes ilustraciones se pueden evidenciar los diagramas de flujo del módulo de Balance de Materia:

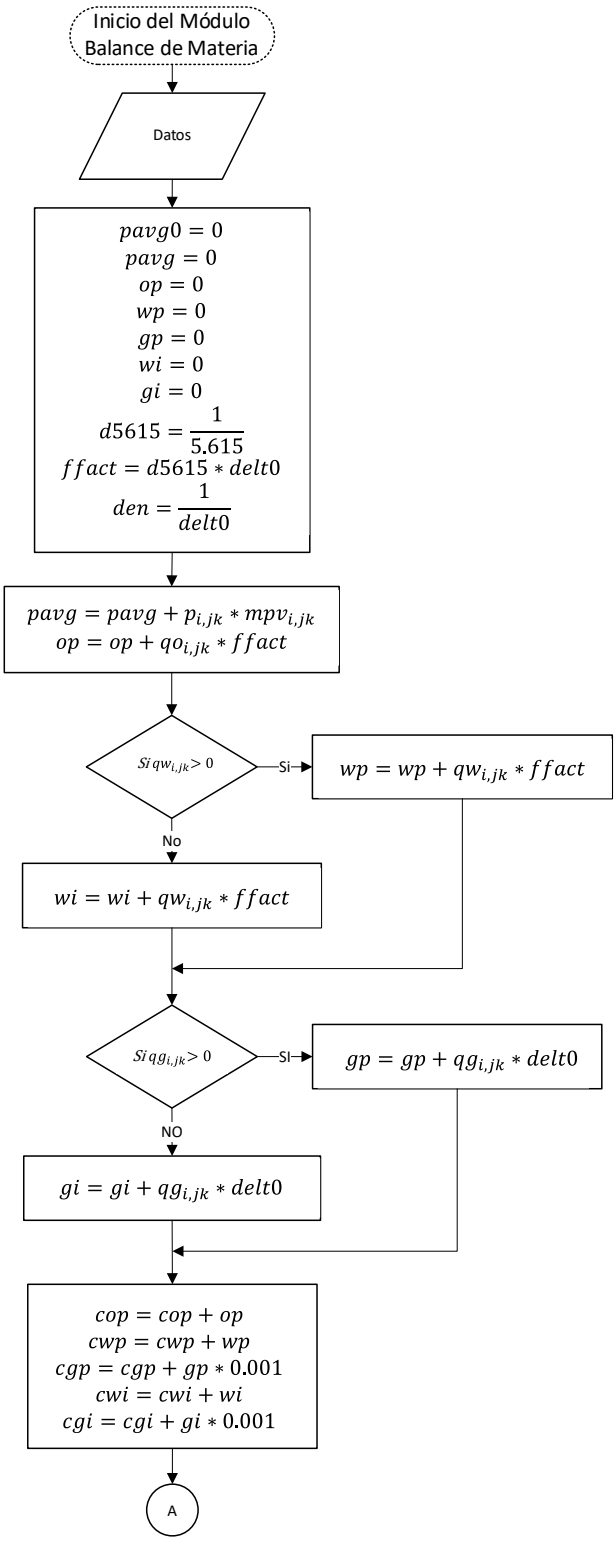


Ilustración 166:Diagrama de Flujo de la Sección Balance de Materia. Fuente: Elaboración propia.

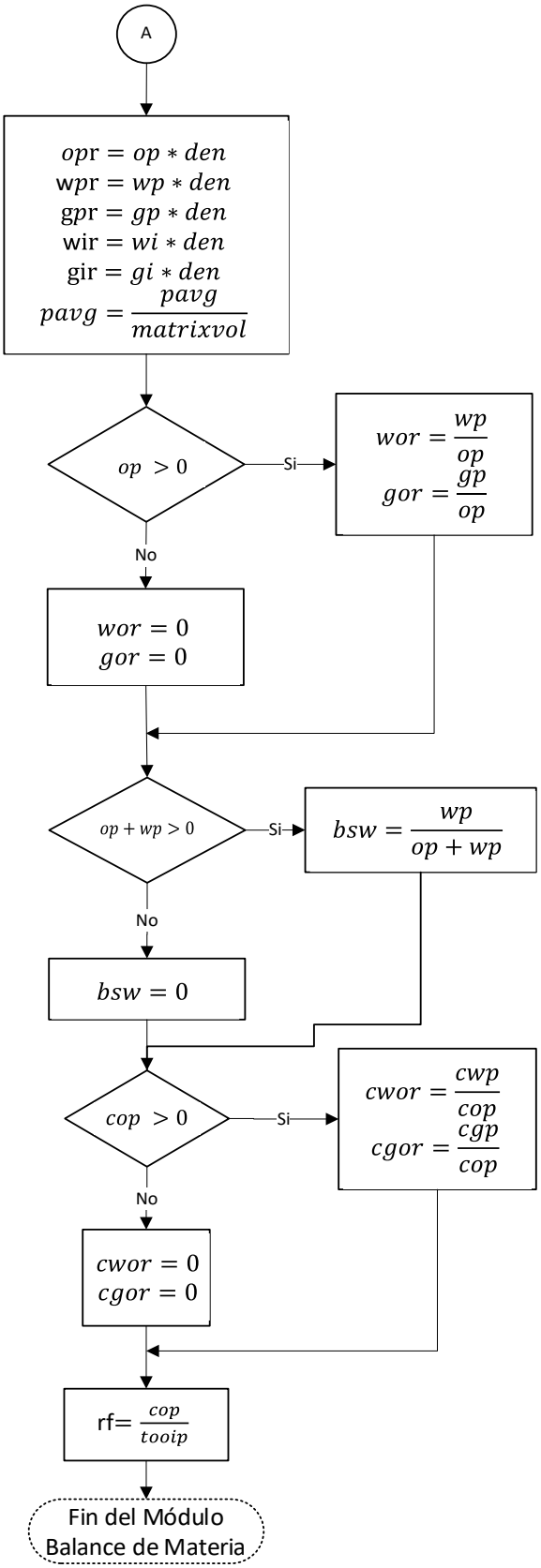


Ilustración 167:Continuación de diagrama de Flujo de la Sección Balance de Material. Fuente: Elaboración propia.

10.6.7. Módulo de Cálculo de Tasas asociada a los cálculos de la malla

En las siguientes ilustraciones pueden ser evidenciados los diagramas de flujo del módulo de Tasas asociadas a los cálculos de la malla:

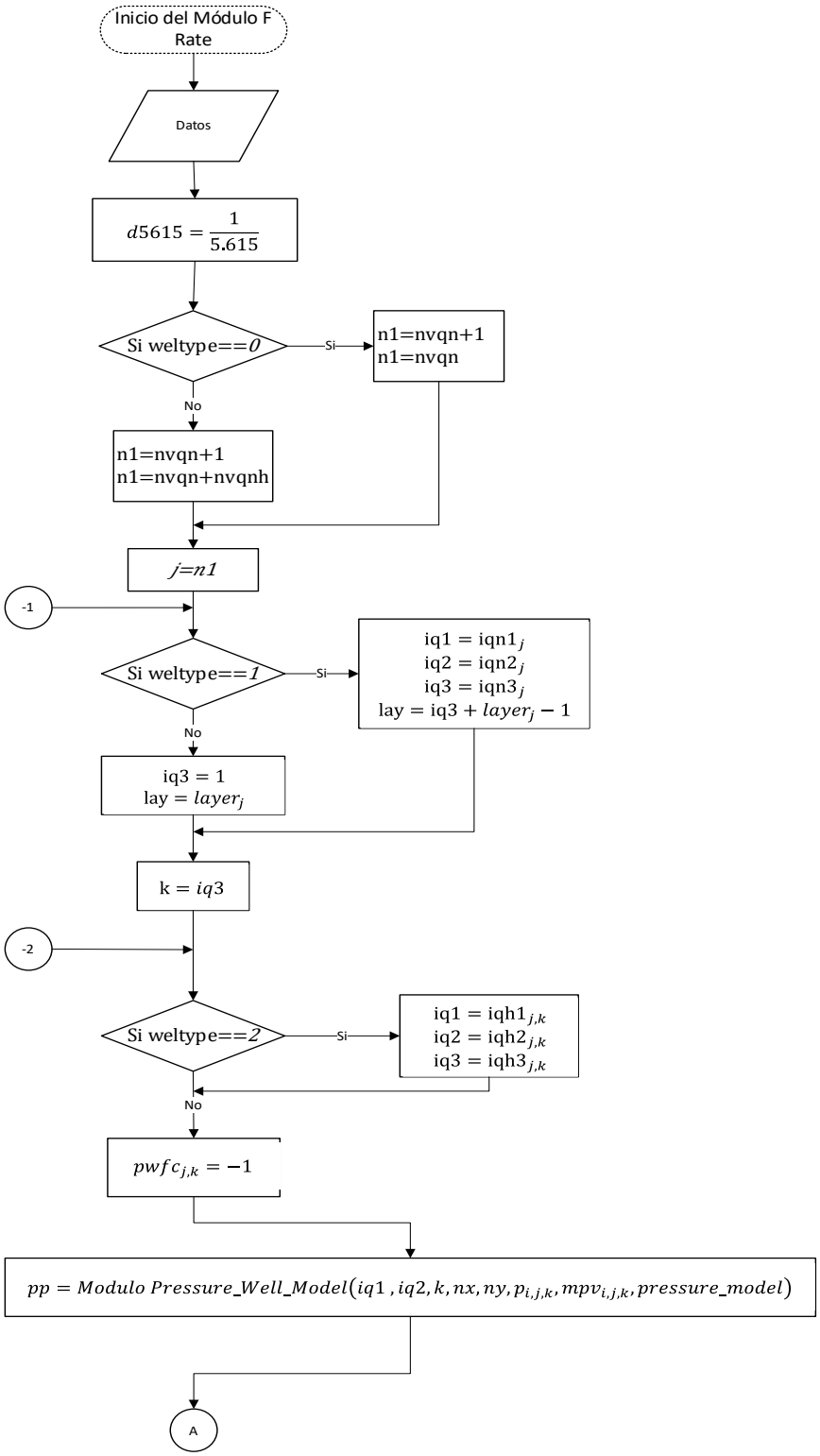


Ilustración 168:Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

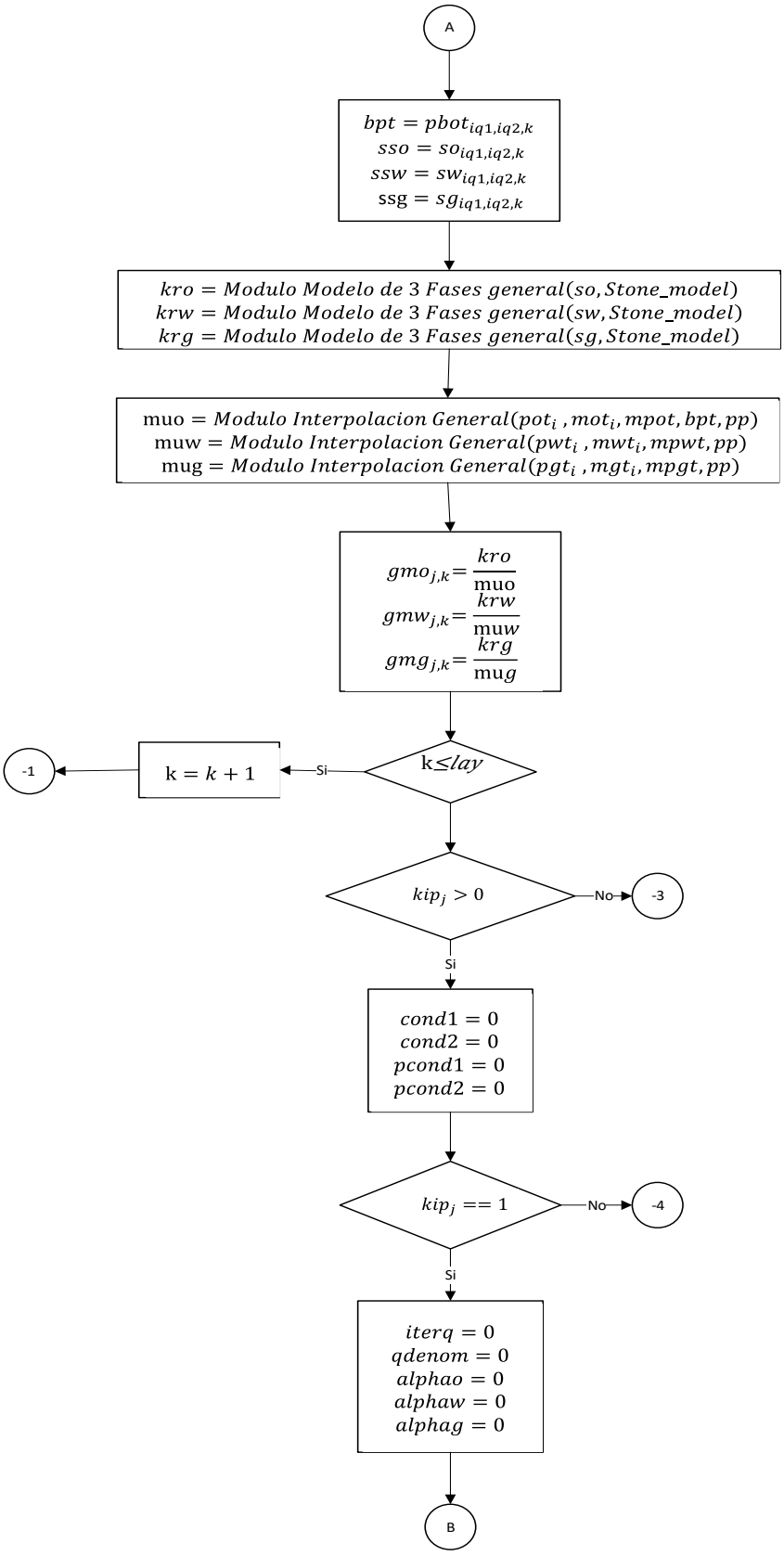


Ilustración 169:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

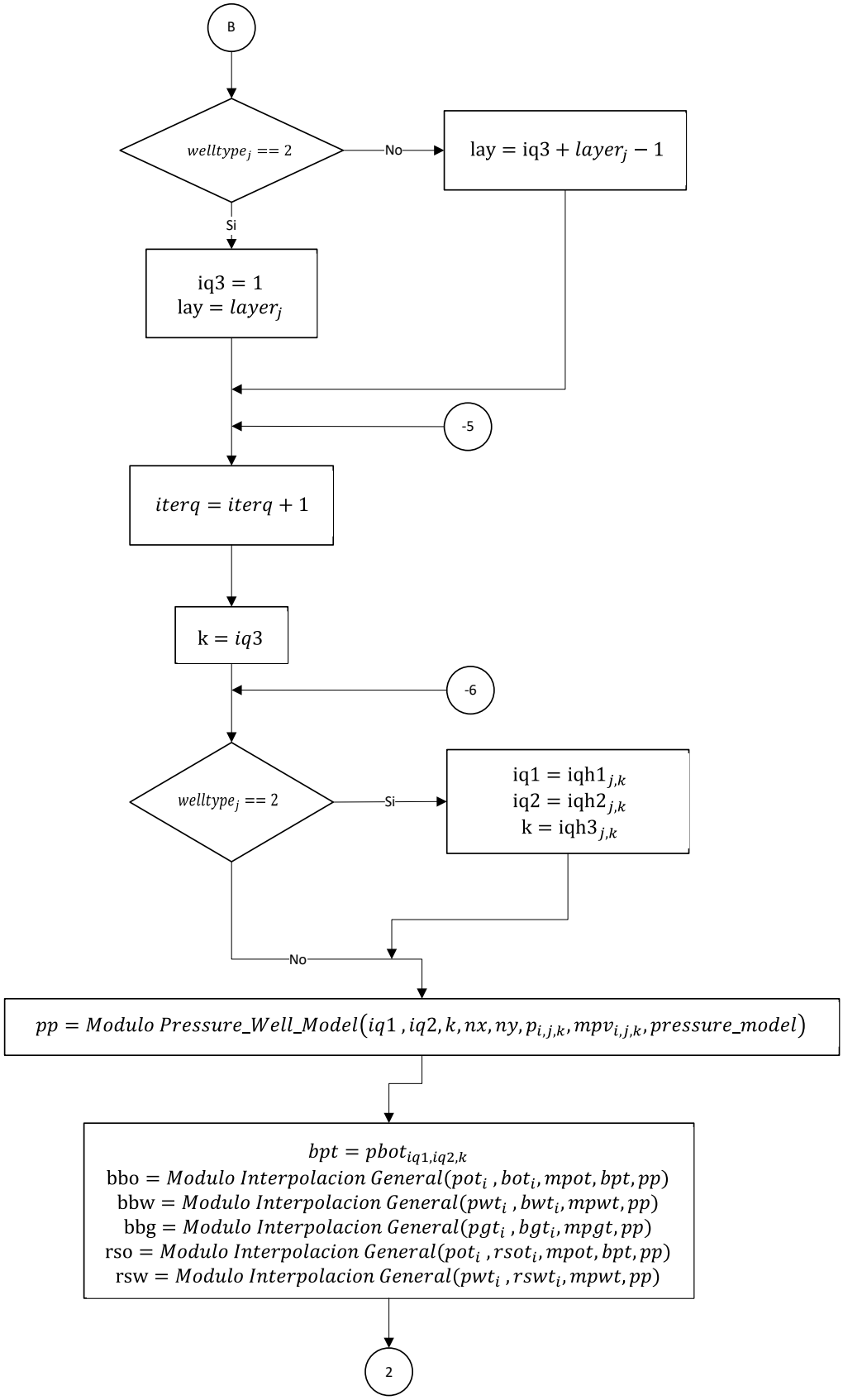


Ilustración 170:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

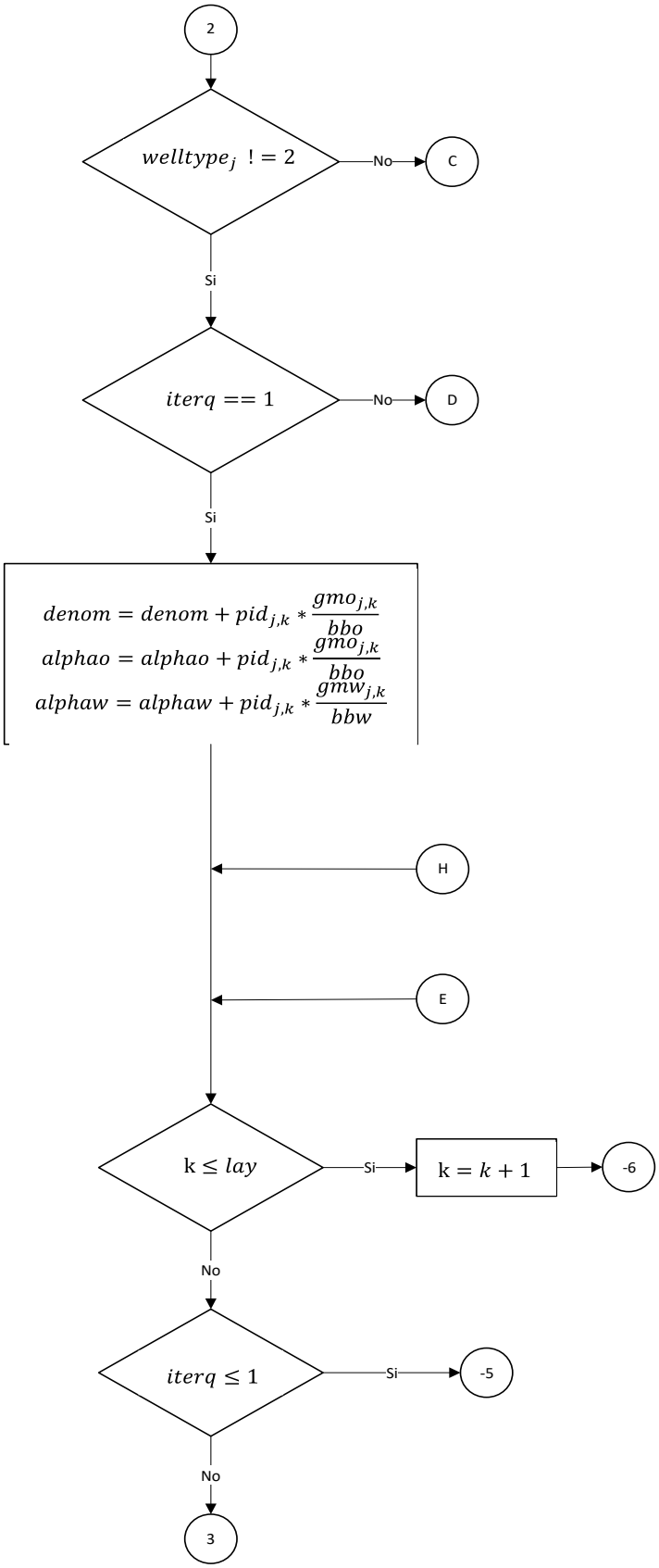


Ilustración 171:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

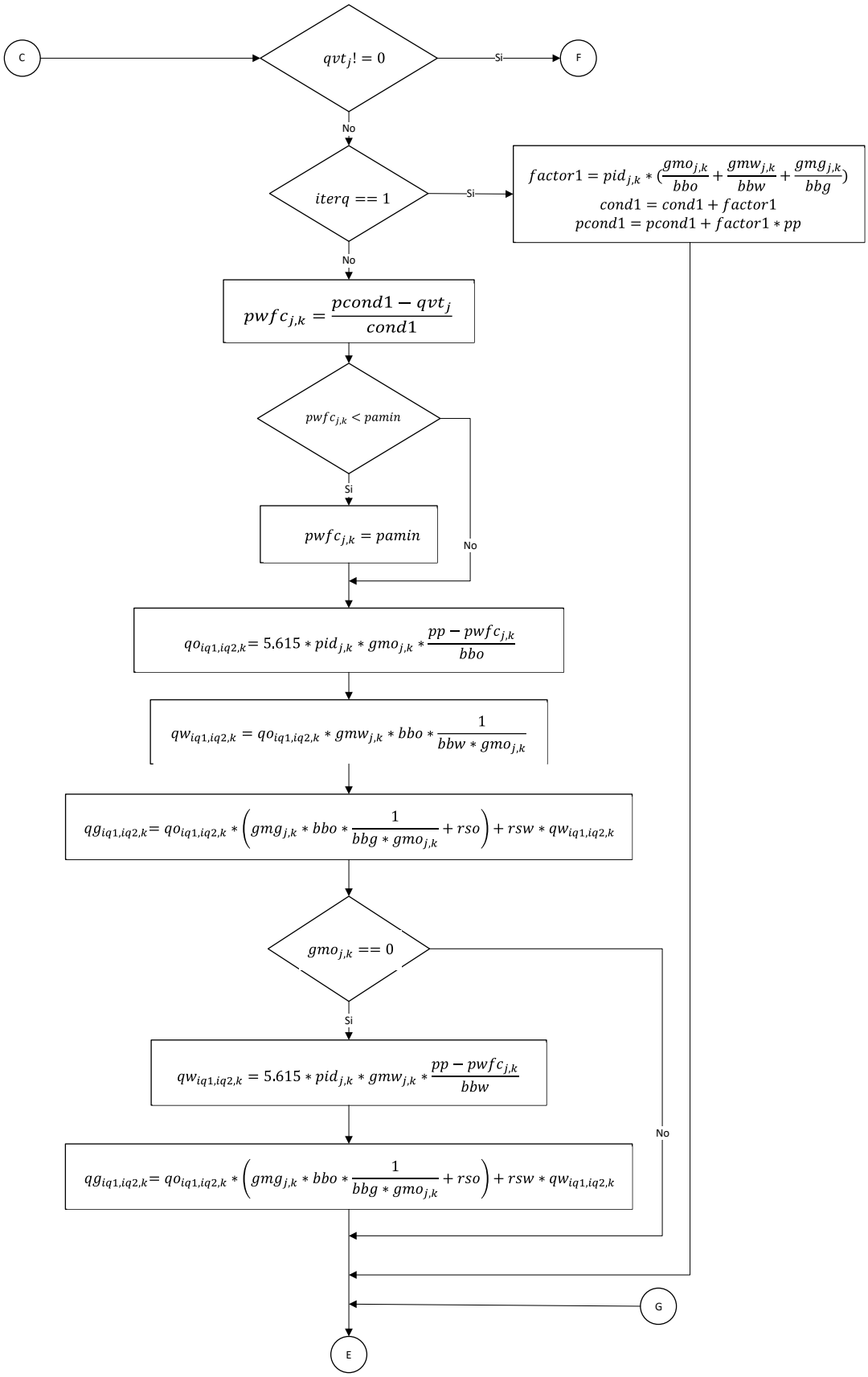


Ilustración 172:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

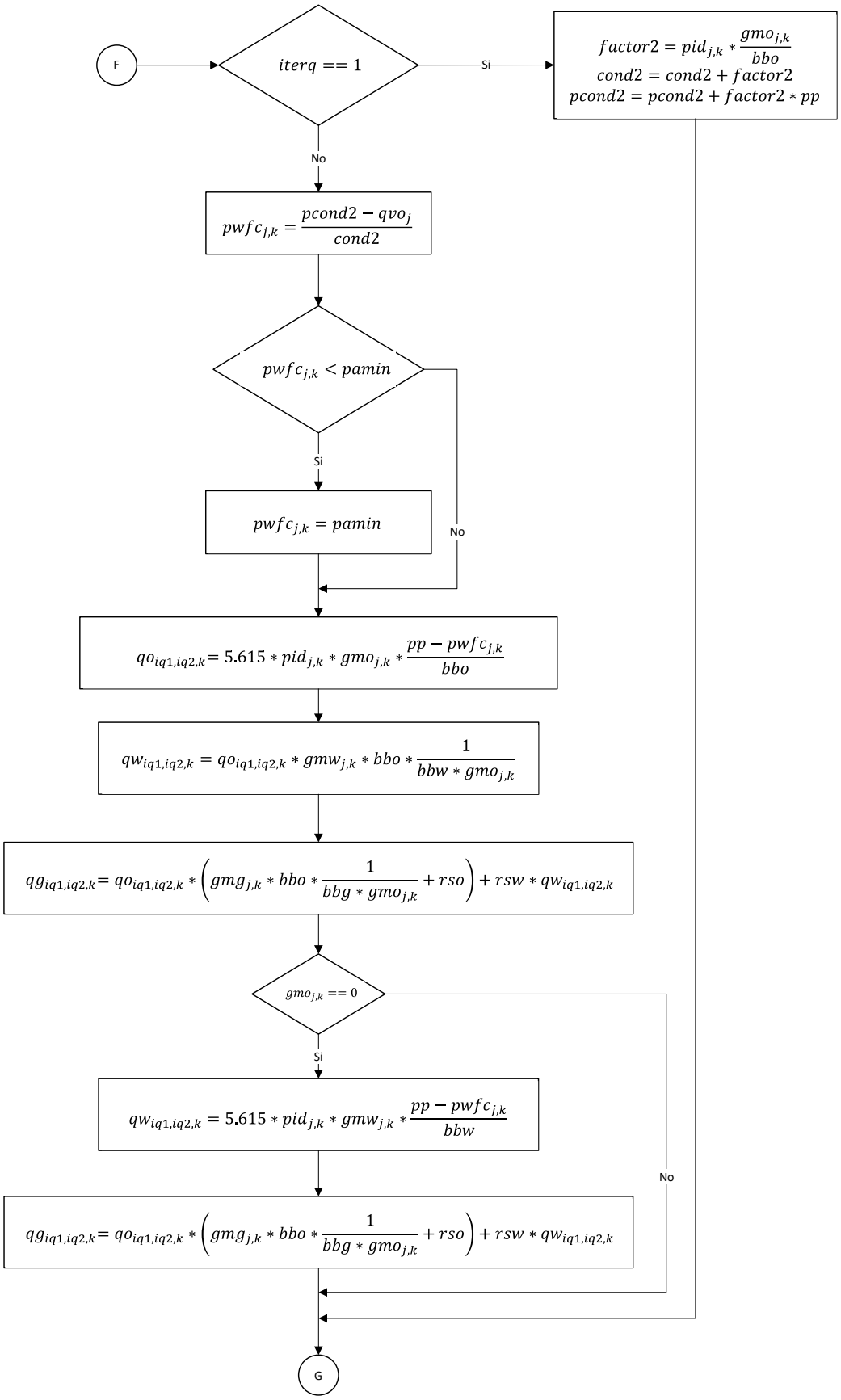


Ilustración 173:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

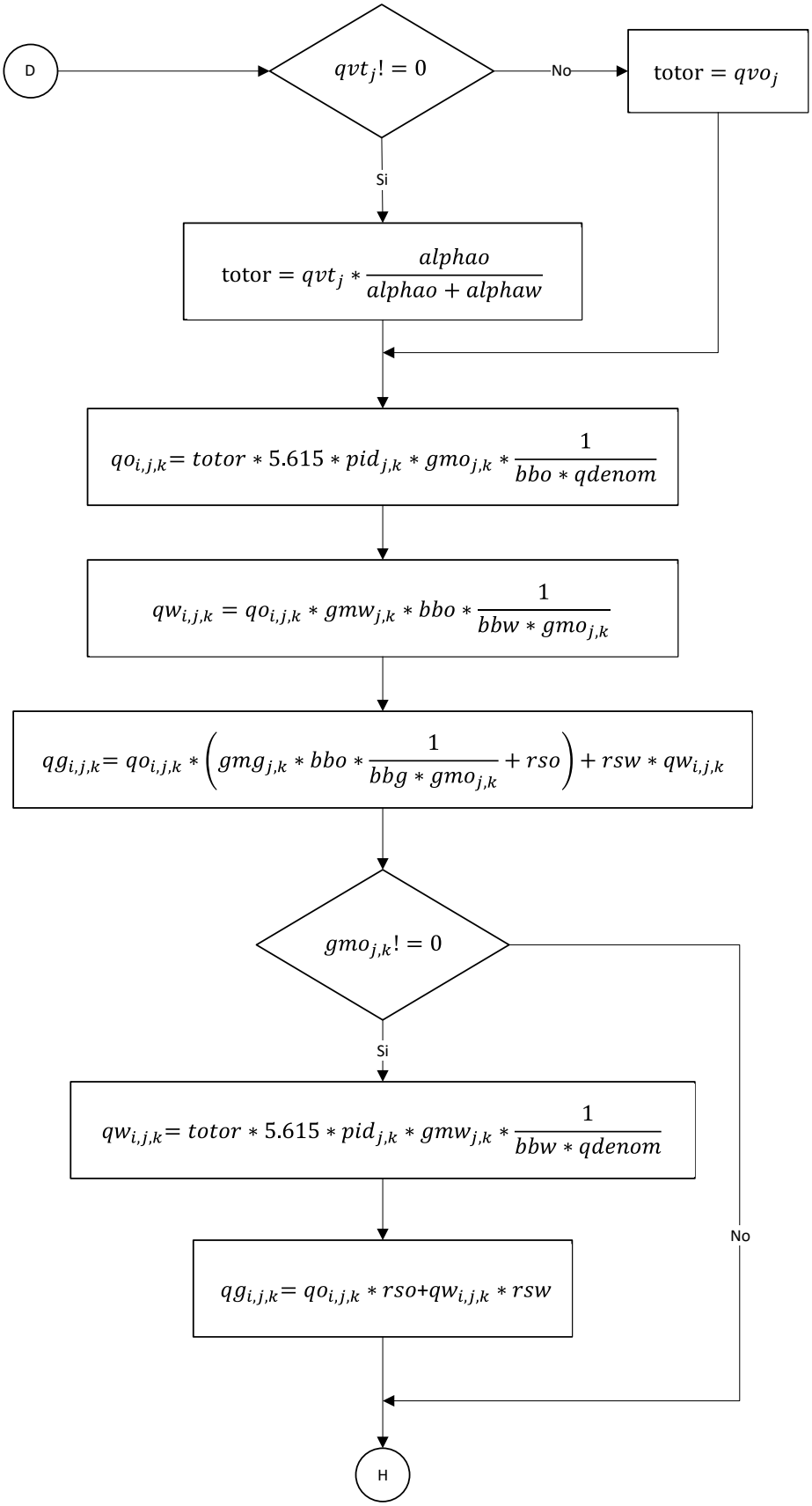


Ilustración 174:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

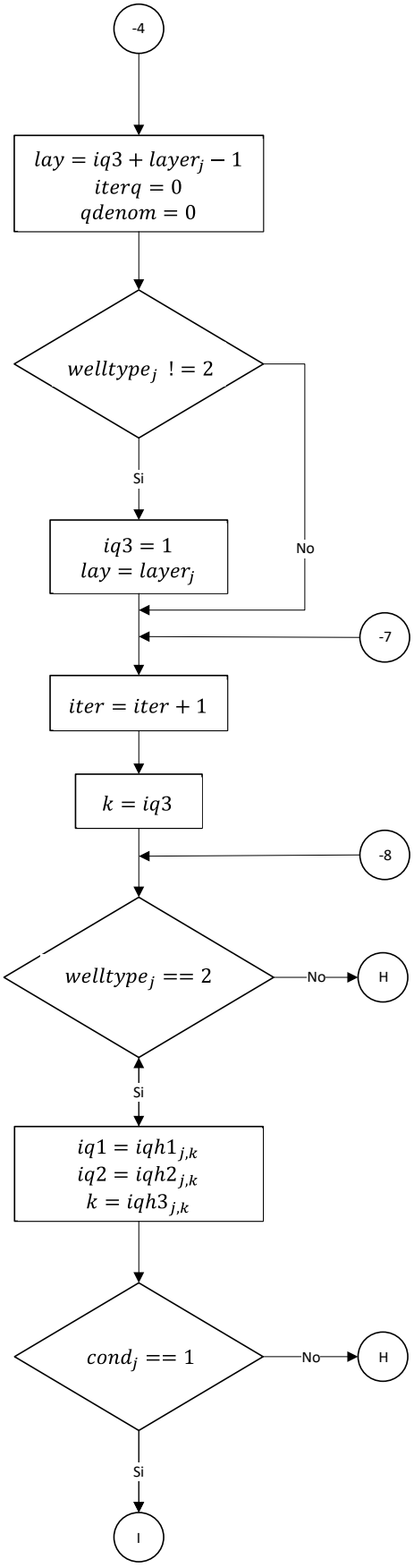


Ilustración 175:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

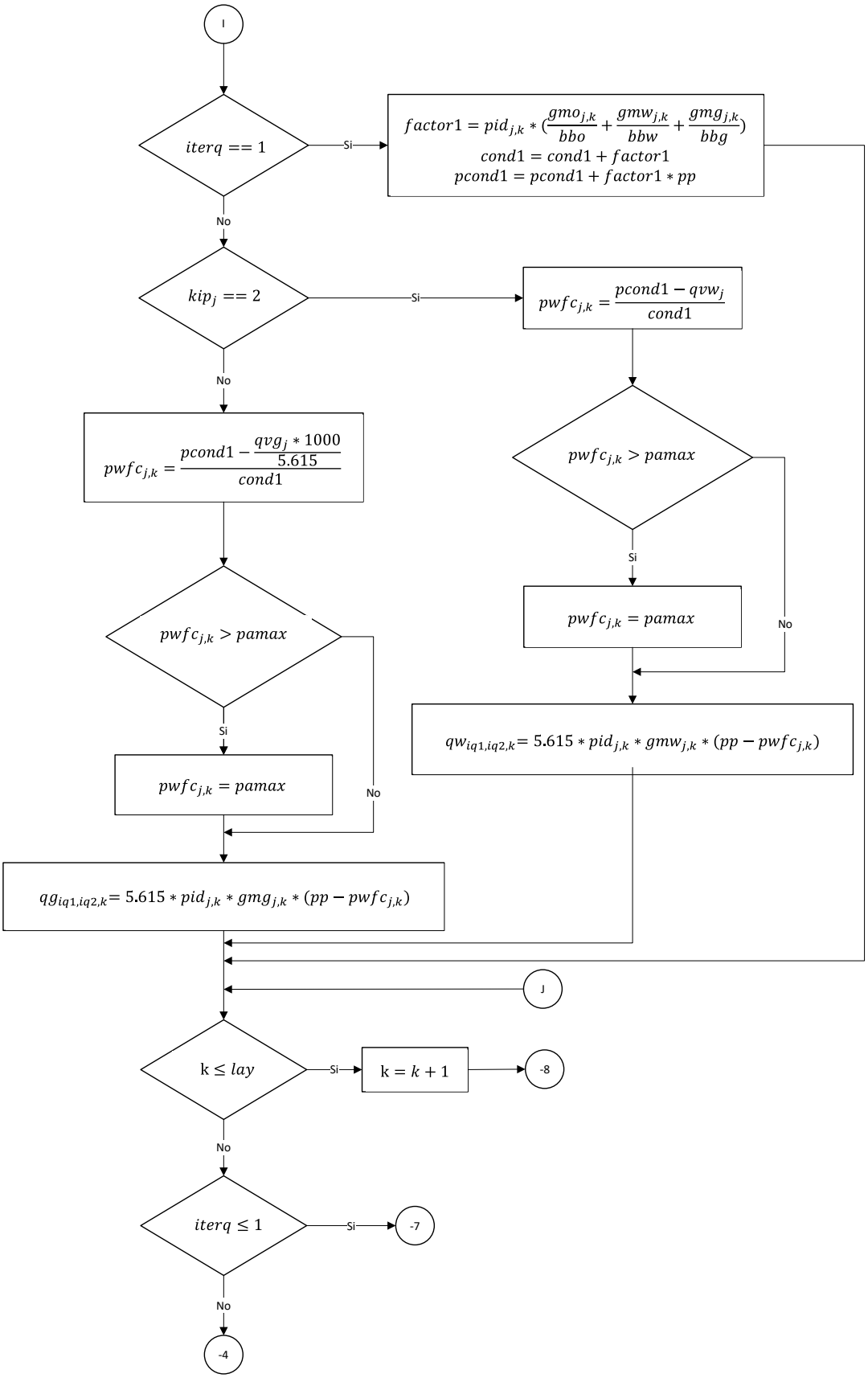


Ilustración 176:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

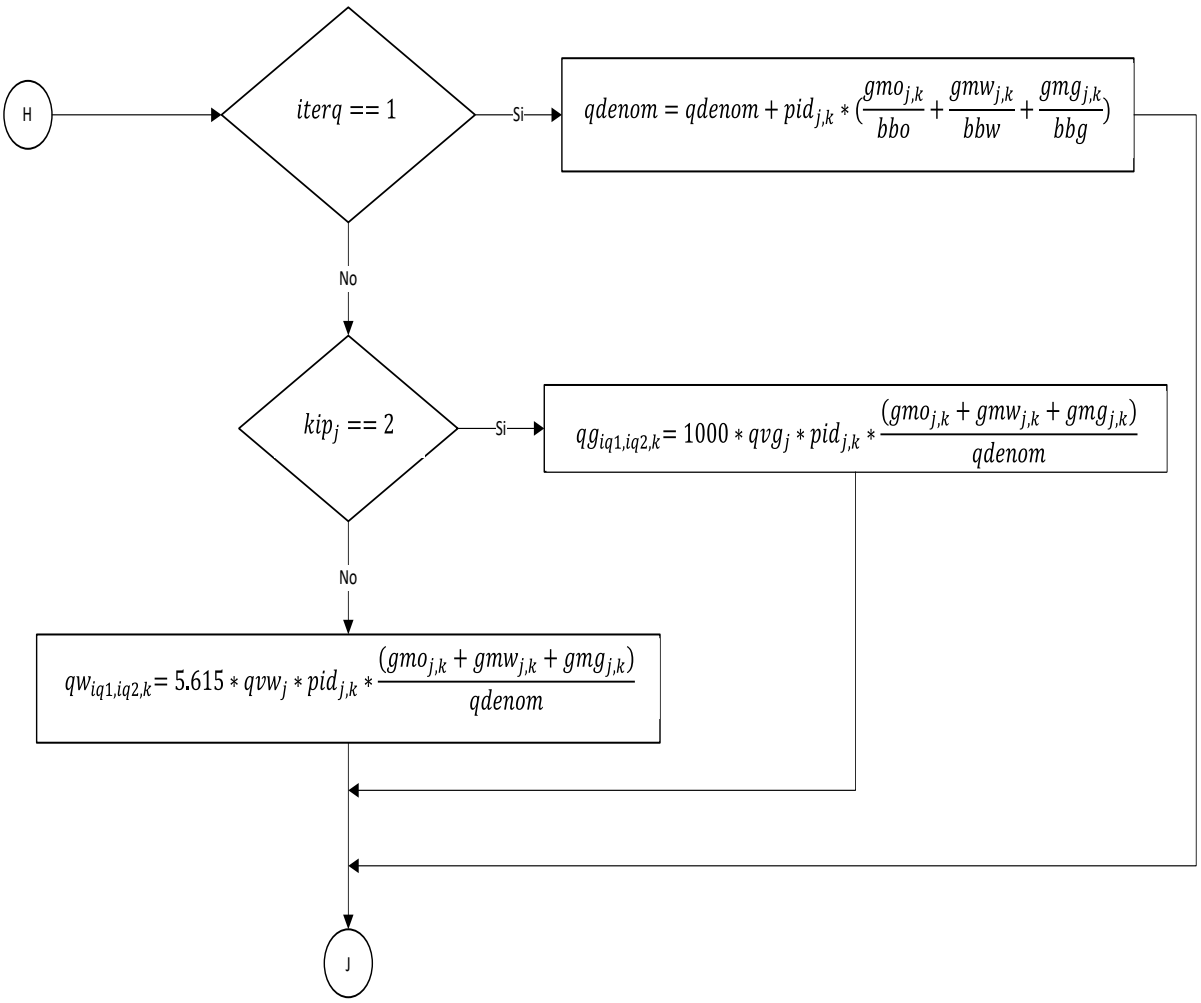


Ilustración 177:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

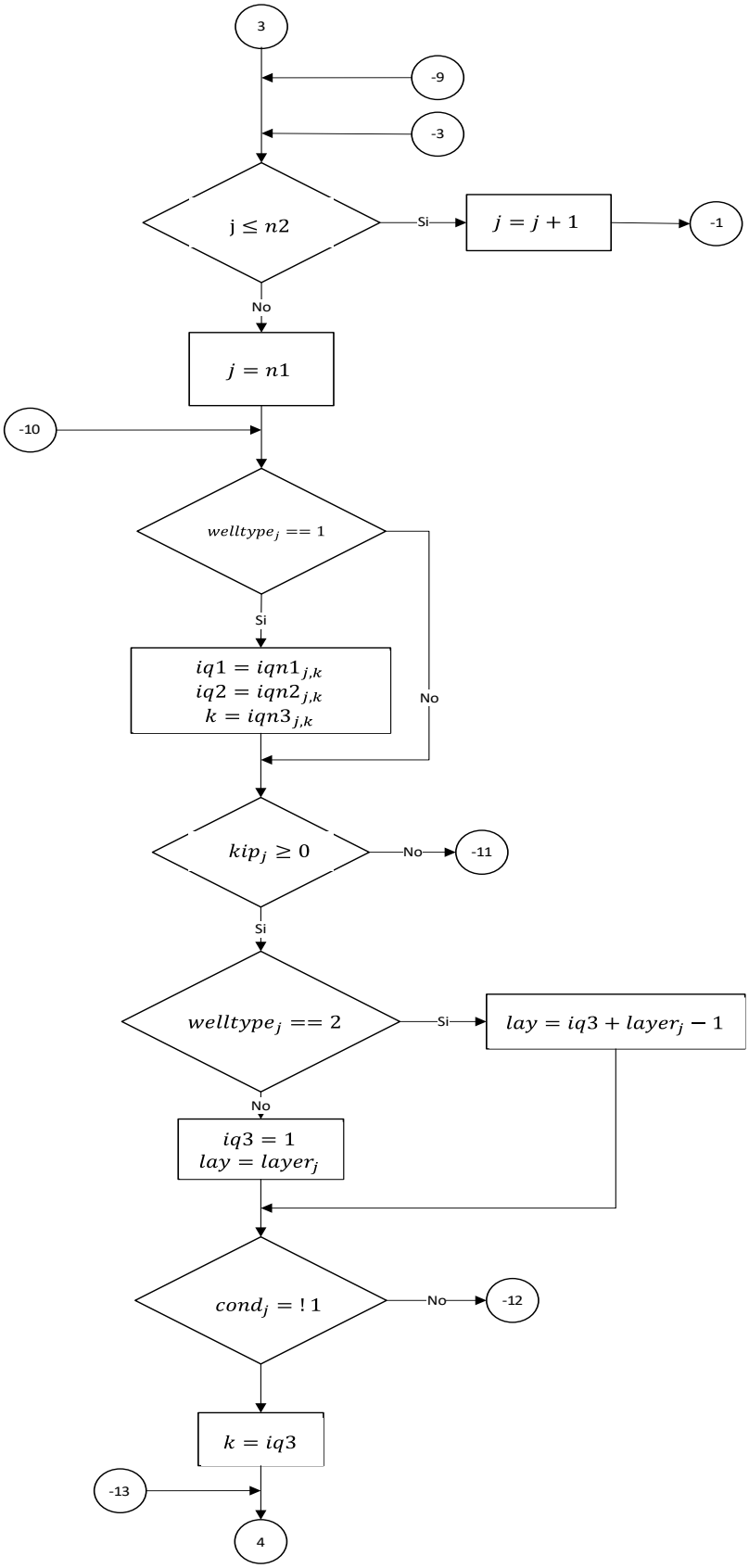


Ilustración 178:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

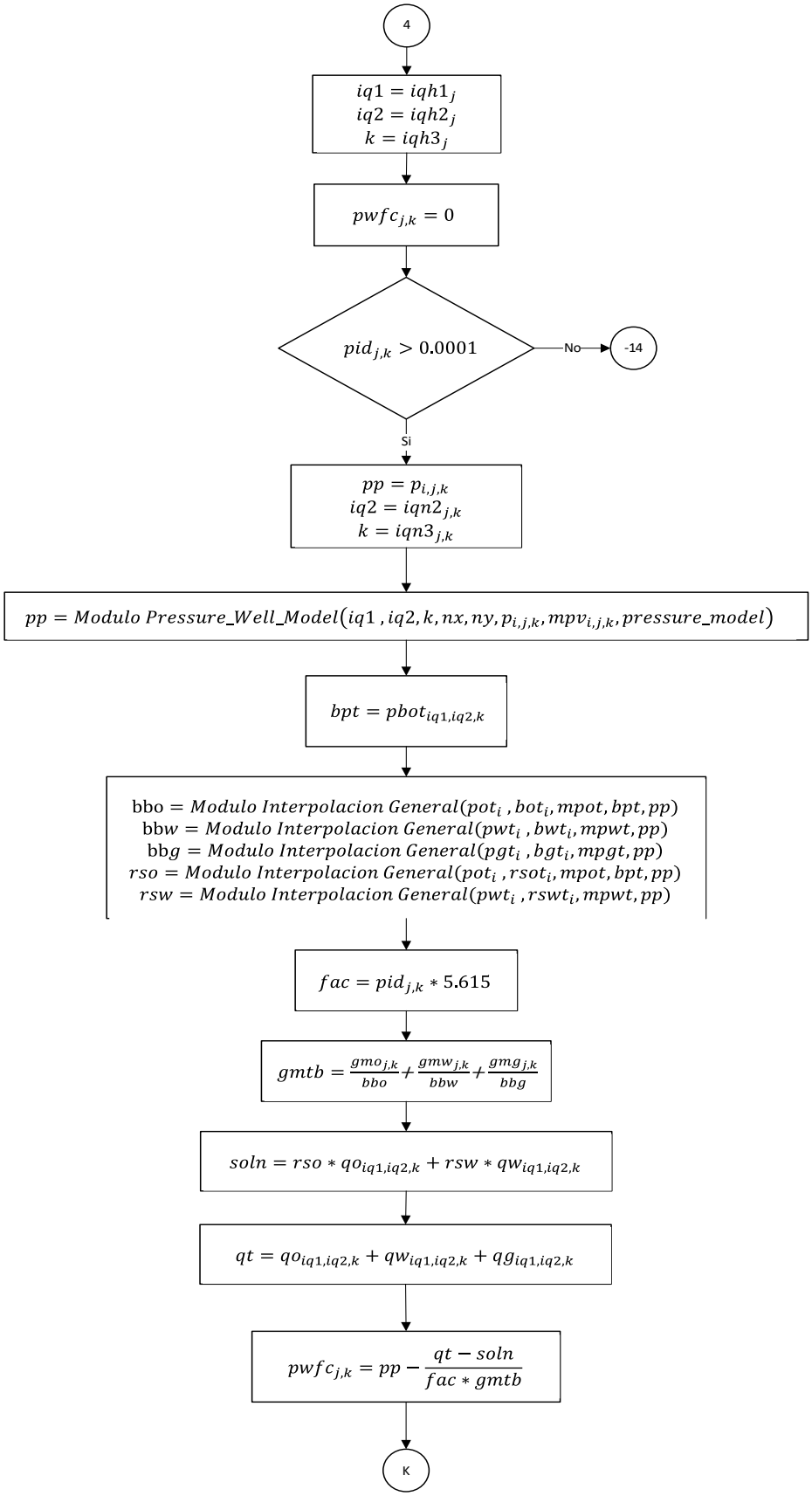


Ilustración 179:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

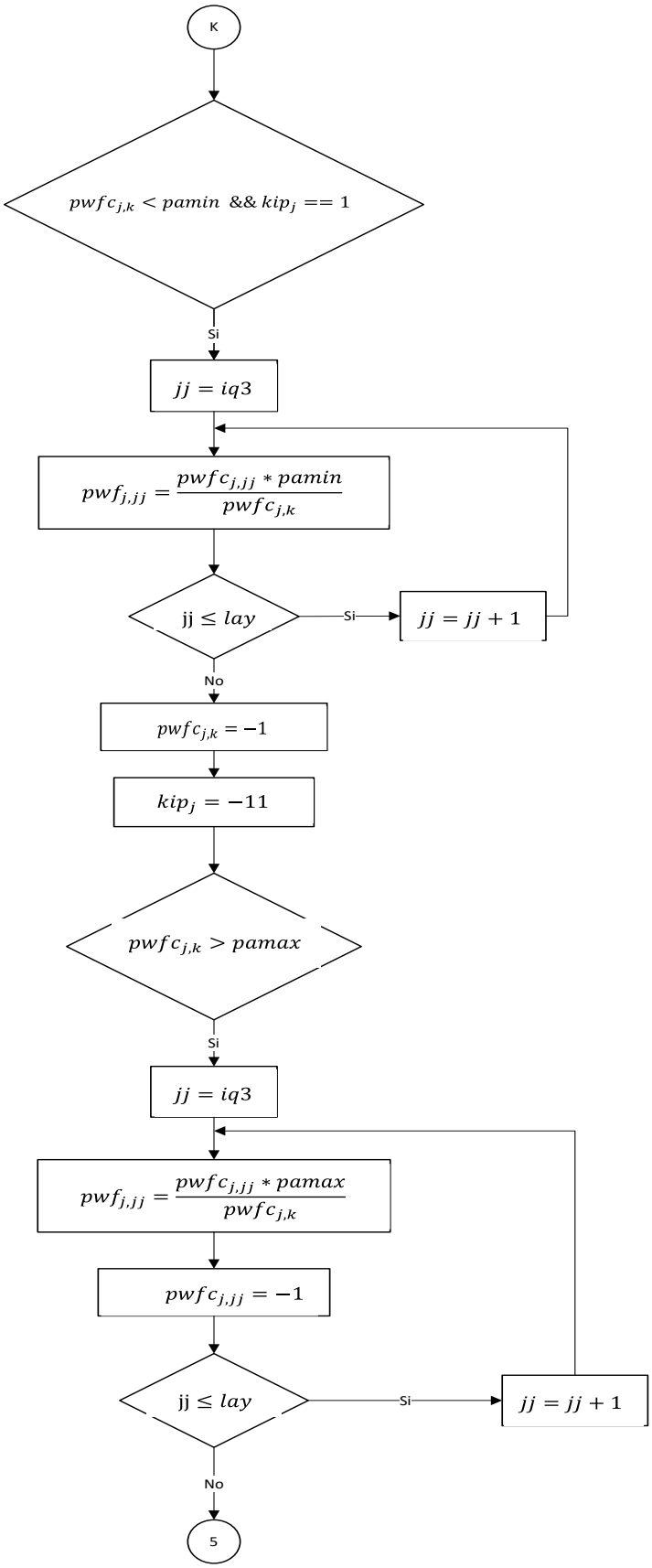


Ilustración 180:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

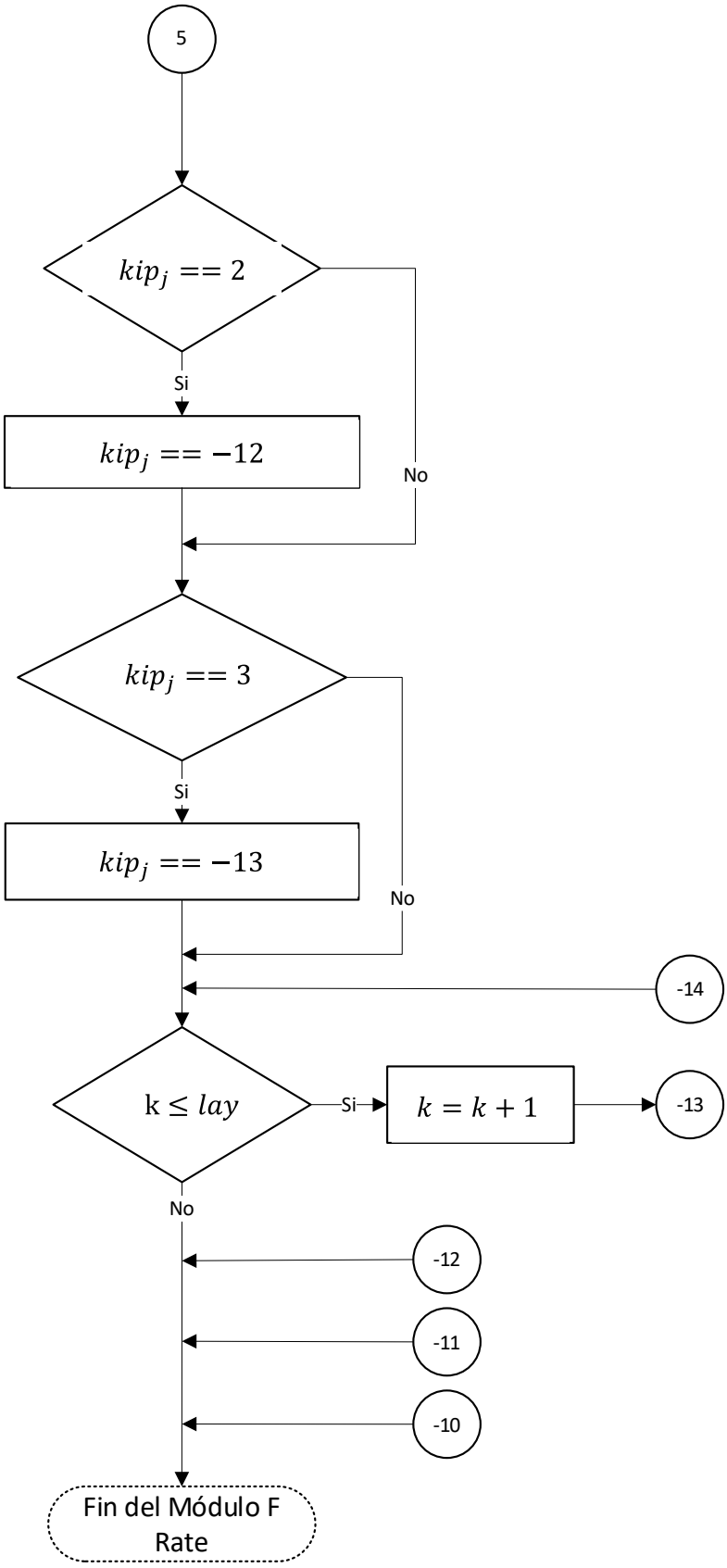


Ilustración 181:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

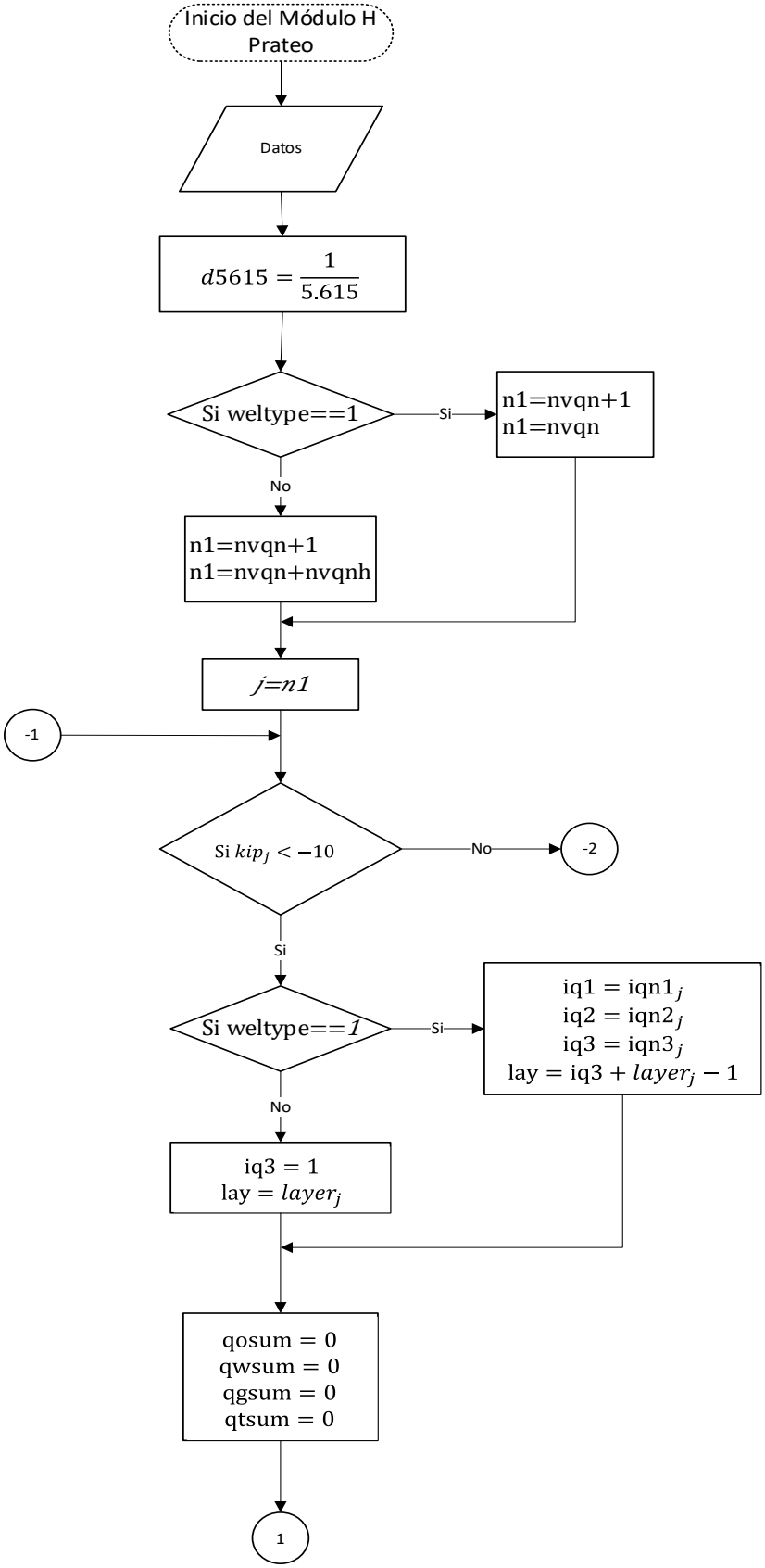


Ilustración 182:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

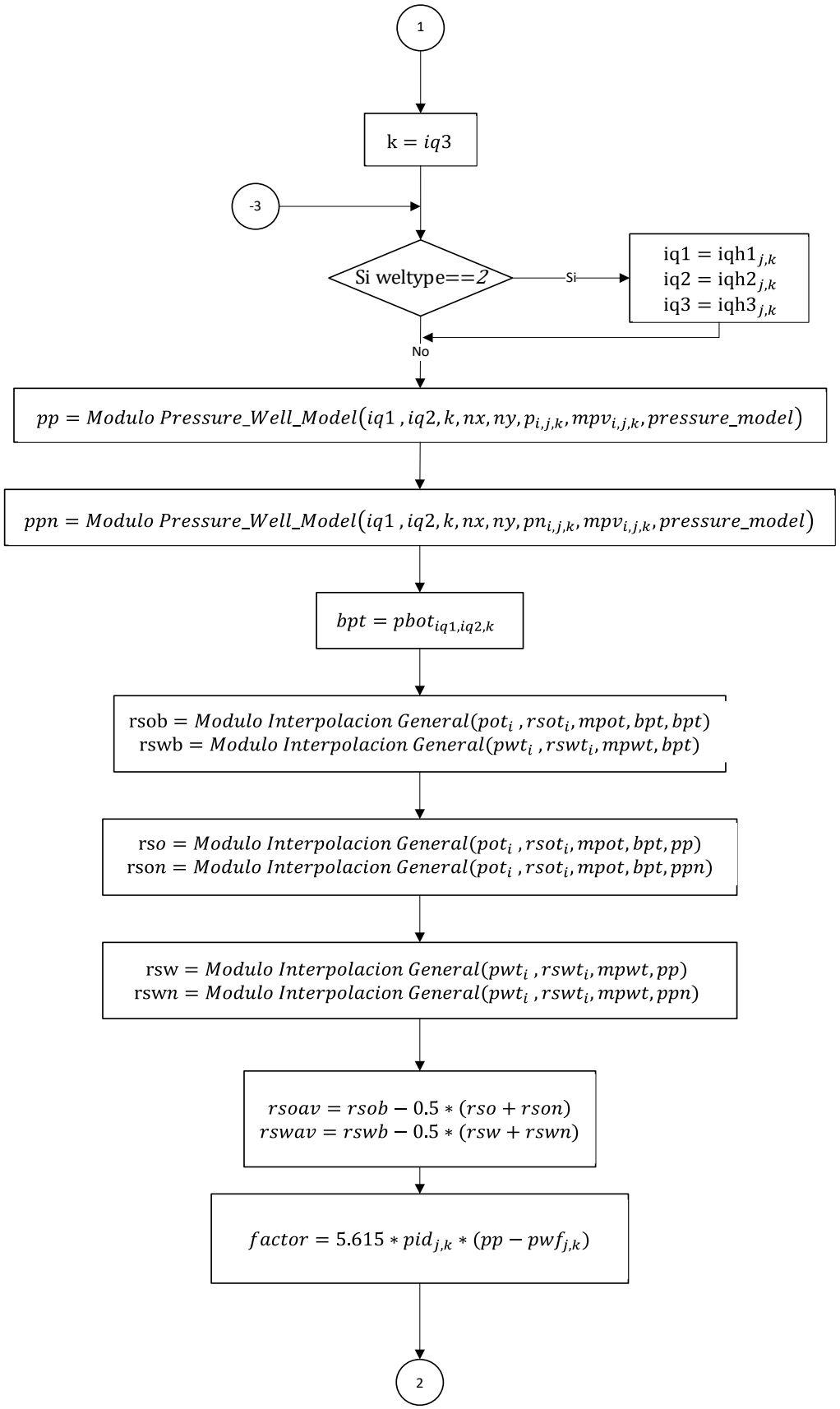


Ilustración 183:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

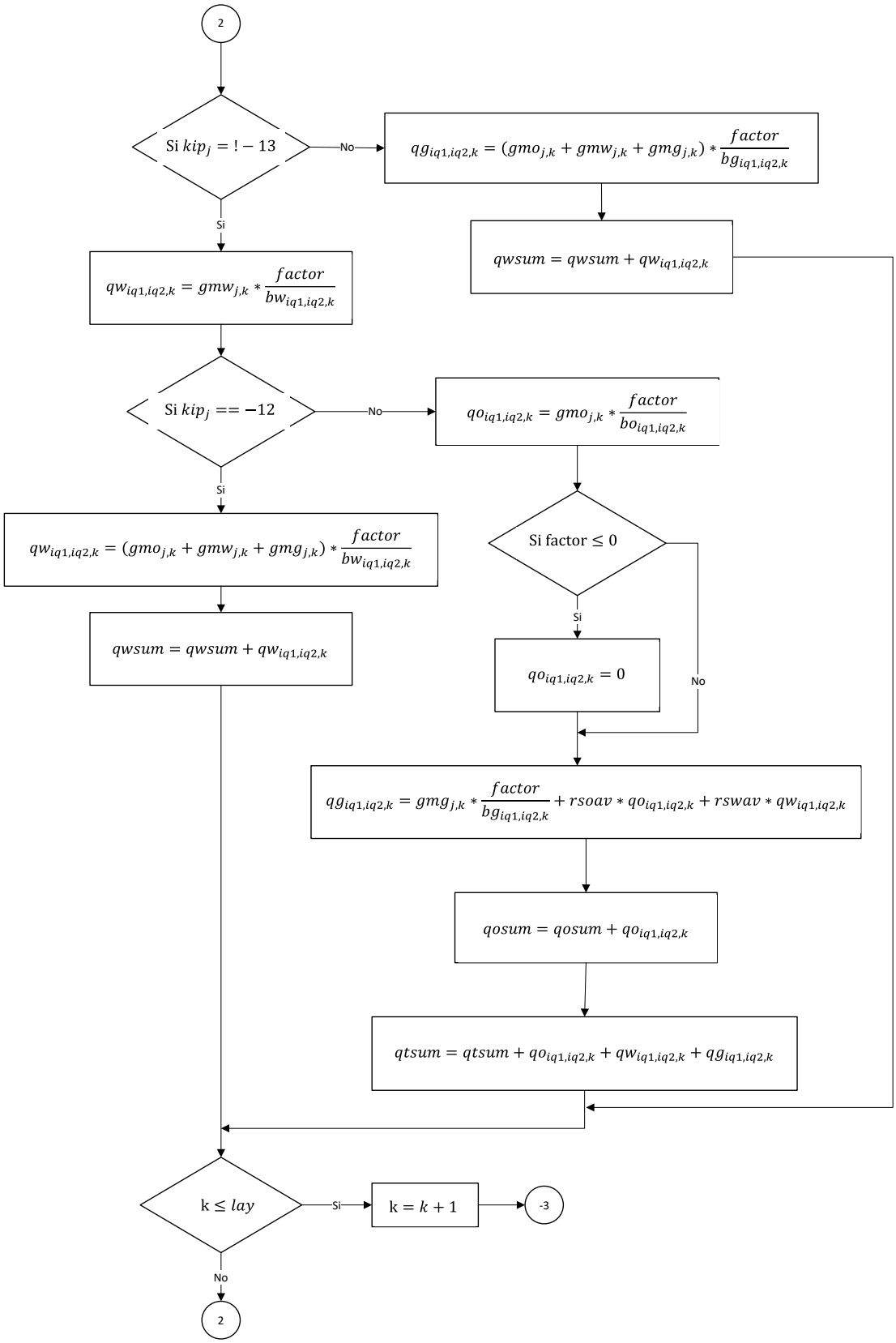


Ilustración 184:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

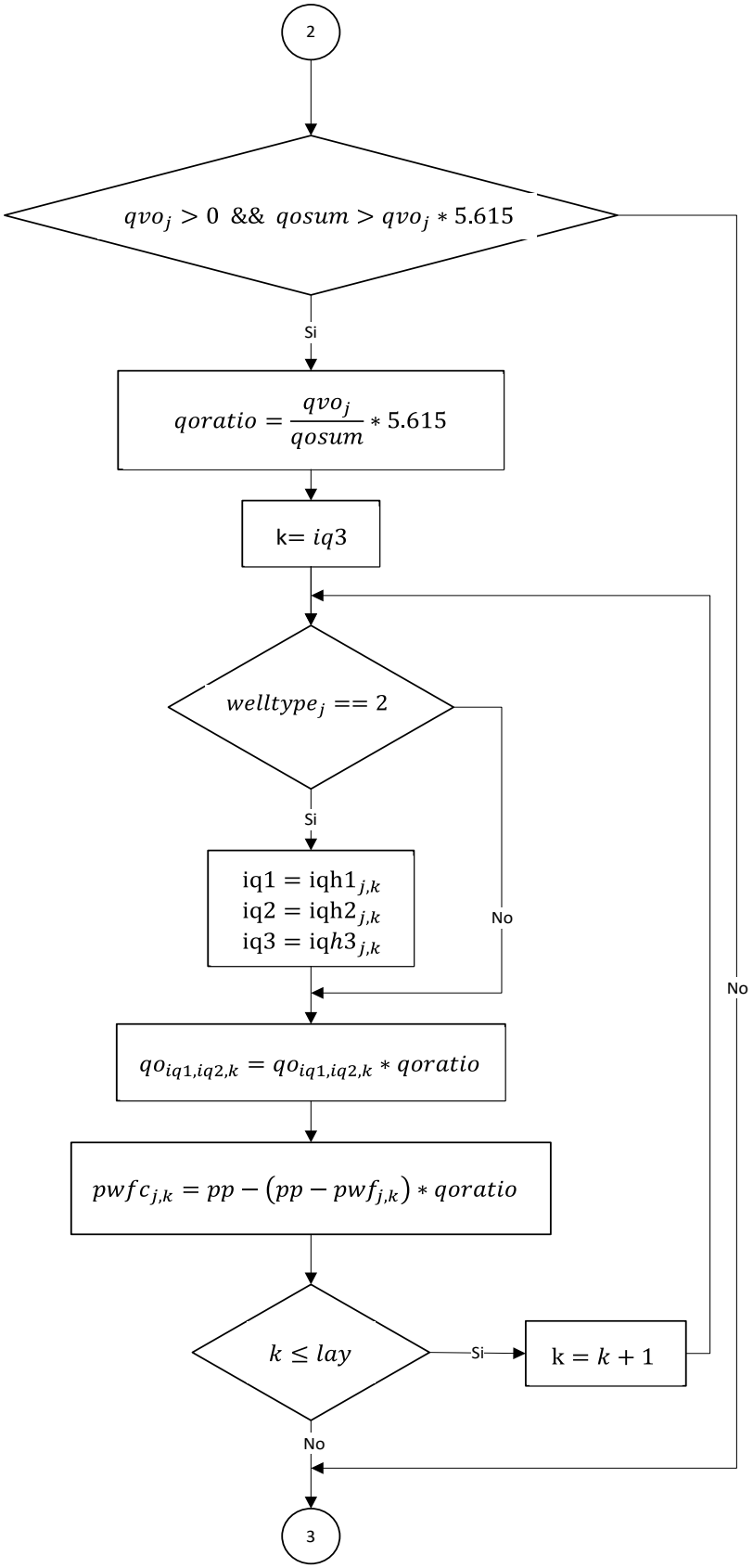


Ilustración 185:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

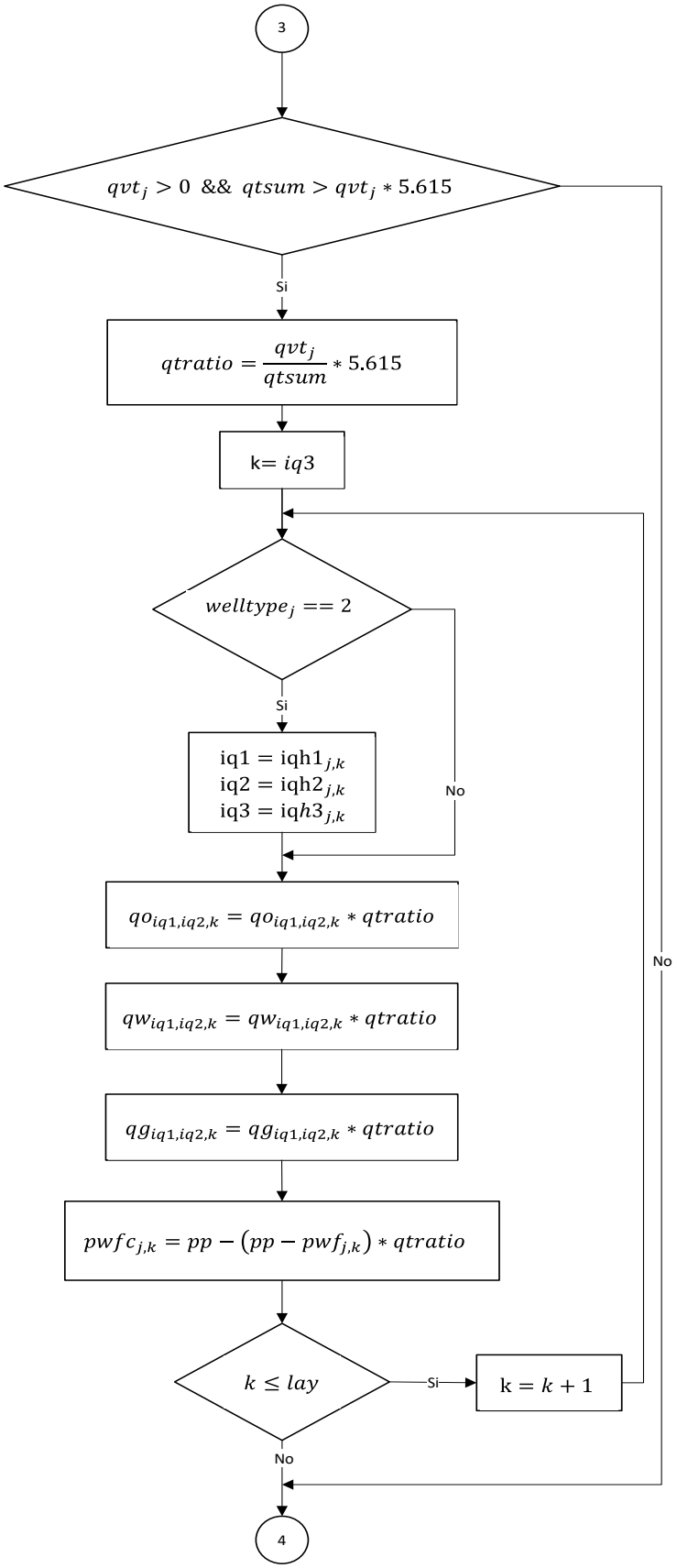


Ilustración 186:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

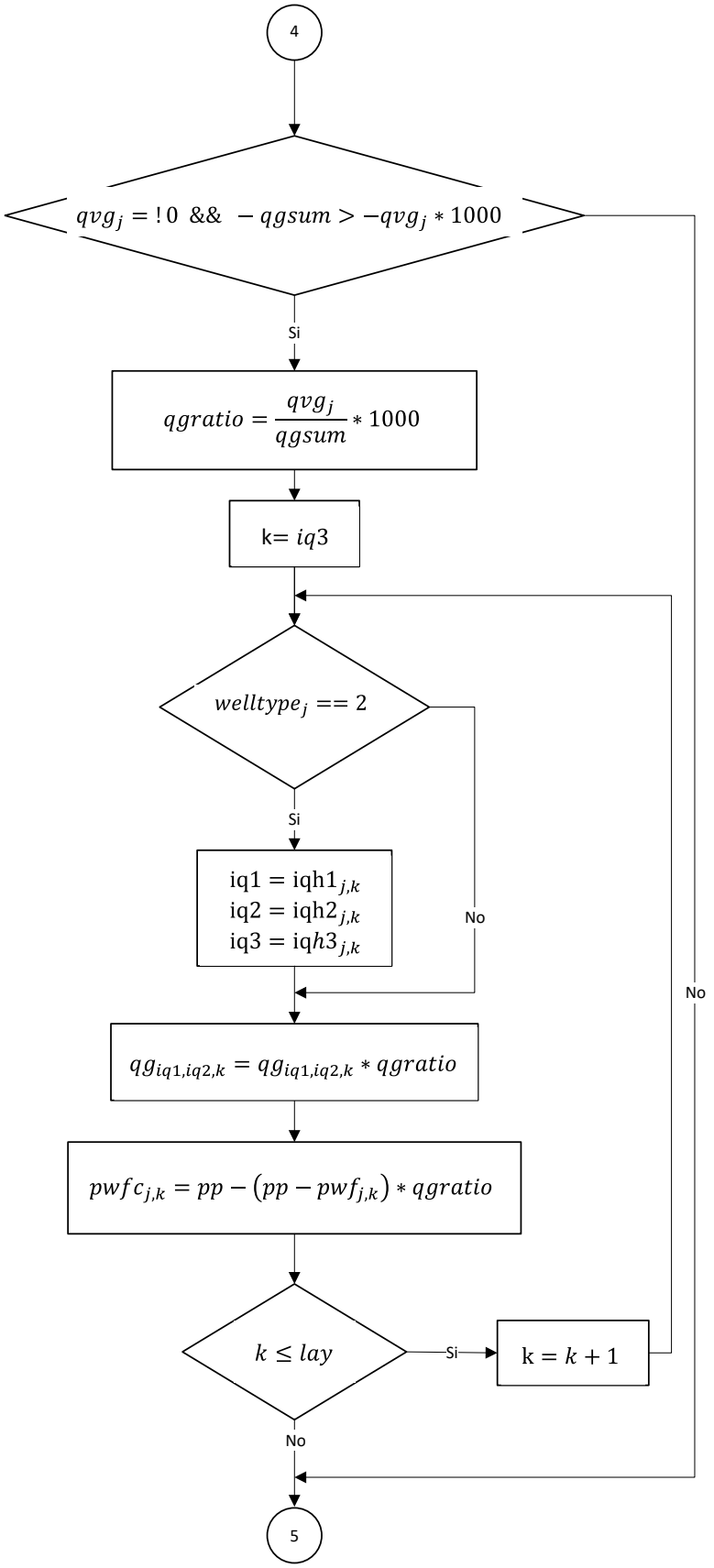


Ilustración 187:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

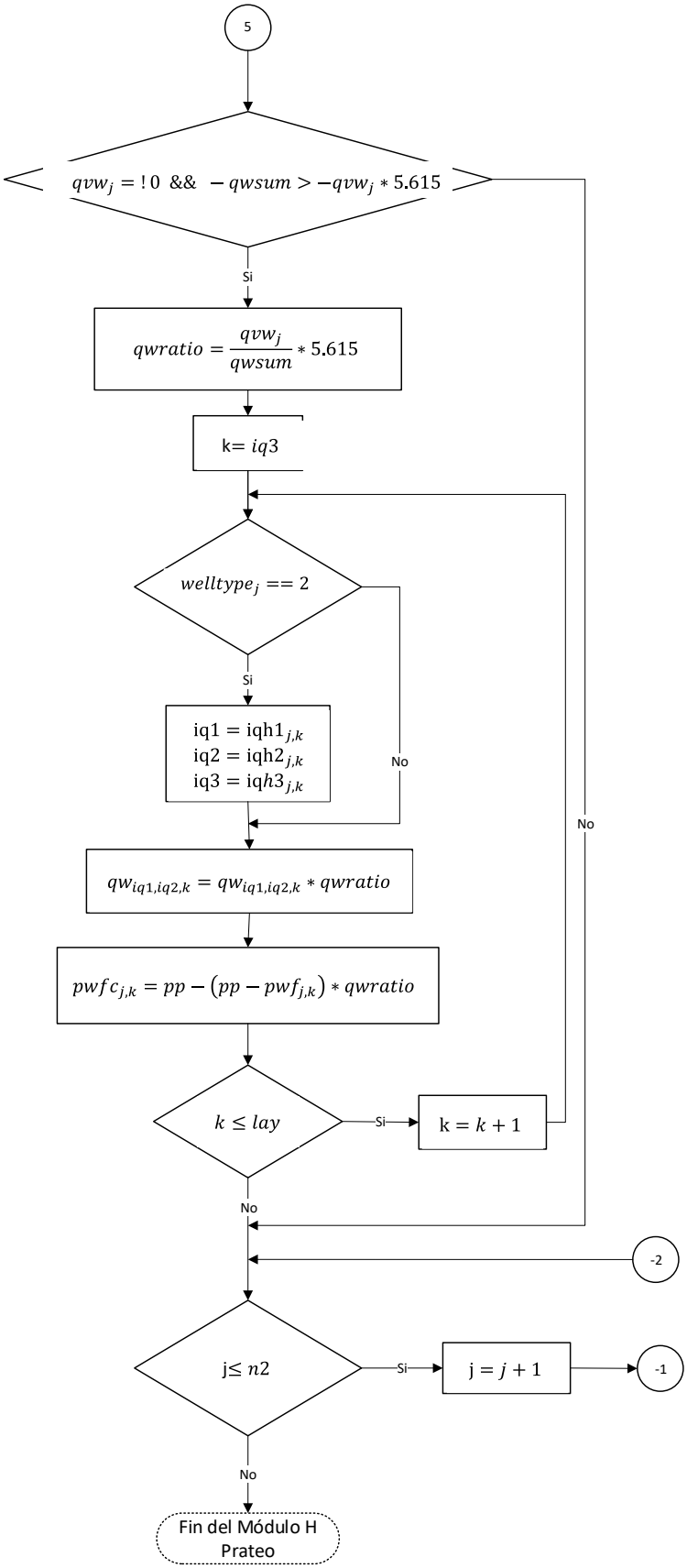


Ilustración 188:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

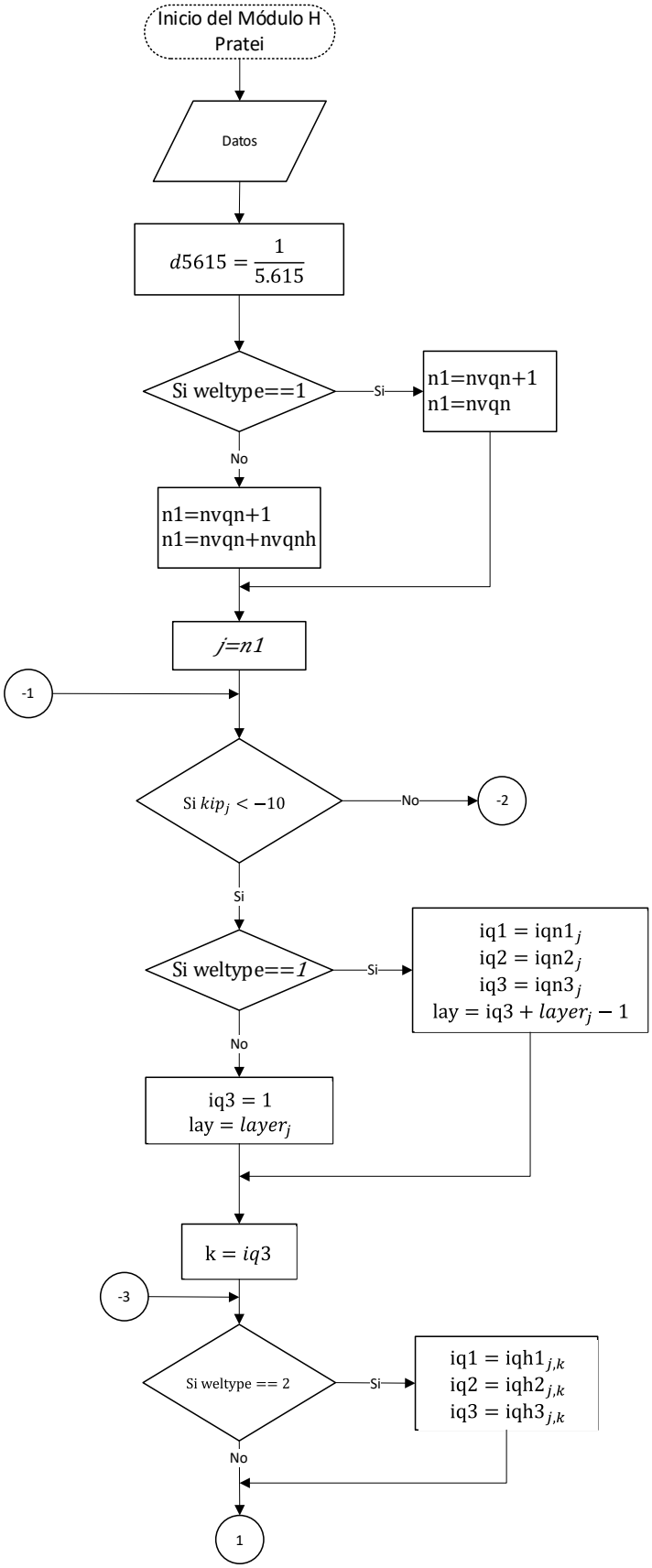


Ilustración 189:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

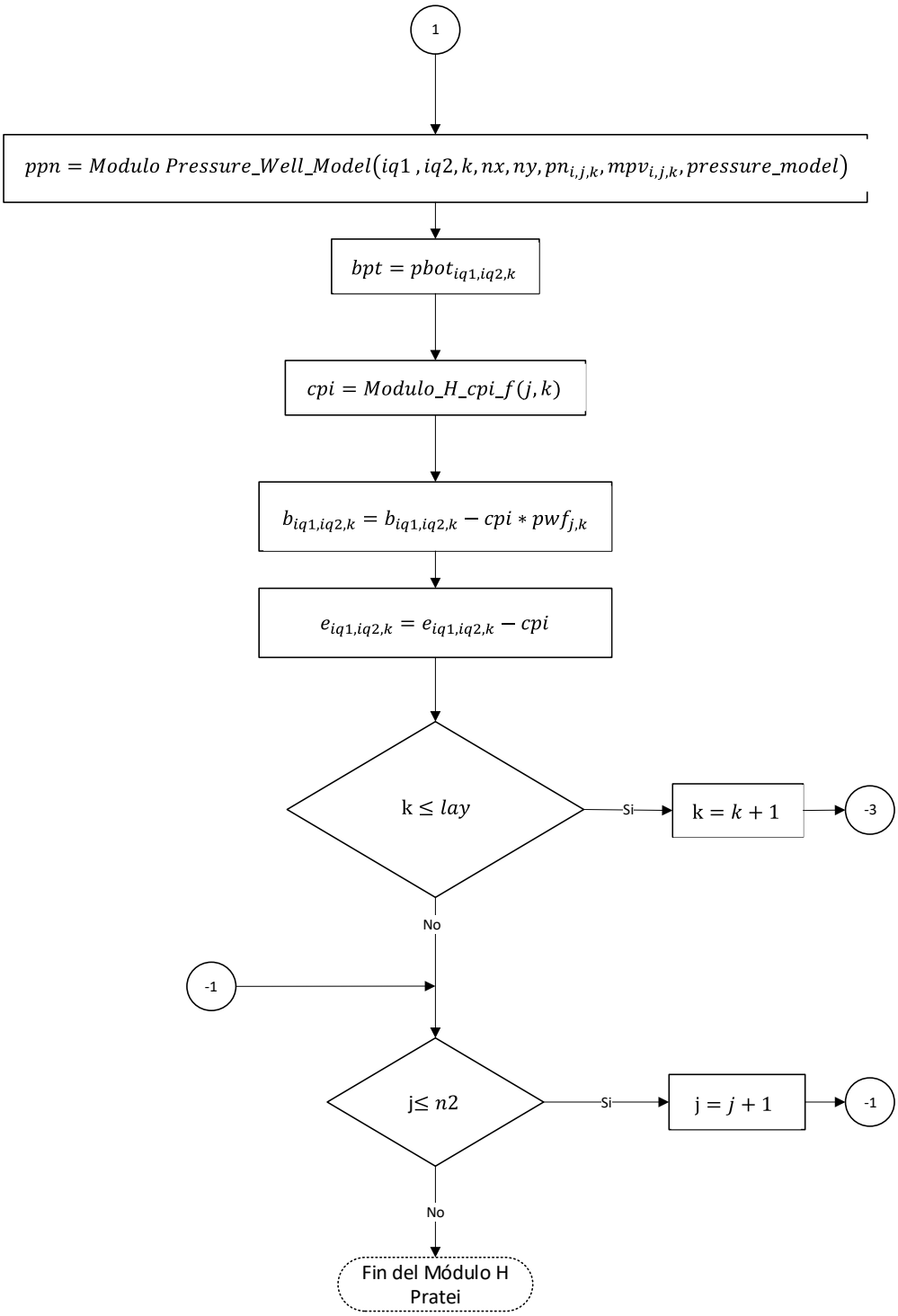


Ilustración 190:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

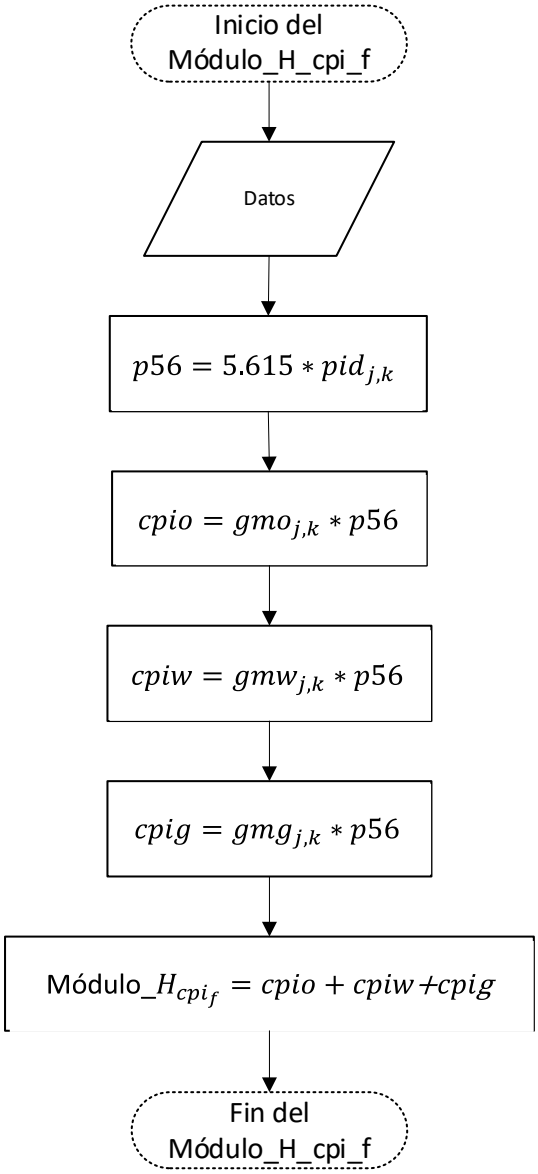


Ilustración 191:Continuación-Sección Estimación de Tasa asociadas a los Cálculos de la malla. Fuente: Elaboración propia.

10.6.8. Módulos Calculo de Saturaciones Explícitas

En las siguientes ilustraciones pueden ser evidenciados los diagramas de flujo de los submódulos asociados a la implementación del Módulo de Saturaciones Explícitas:

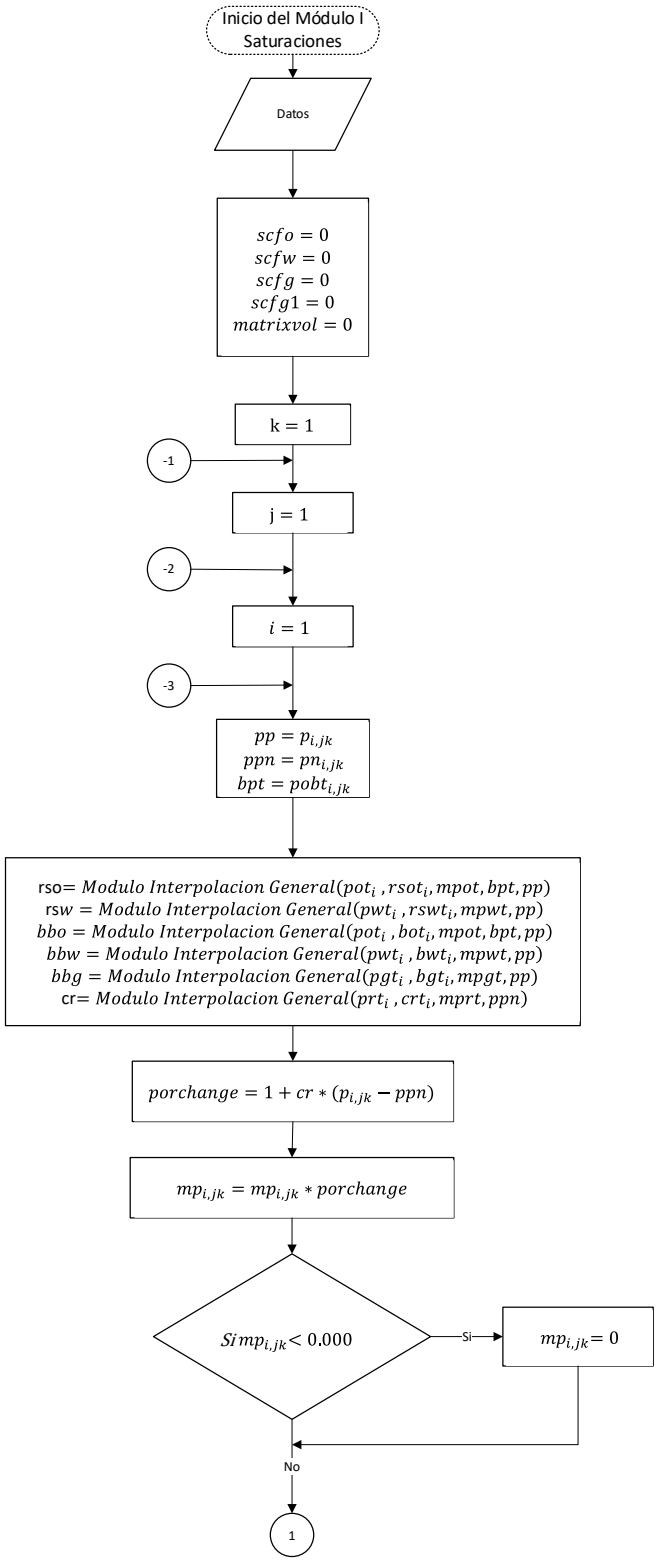


Ilustración 192:Sección solvers de estimación Saturaciones Explicitas. Fuente: Elaboración propia.

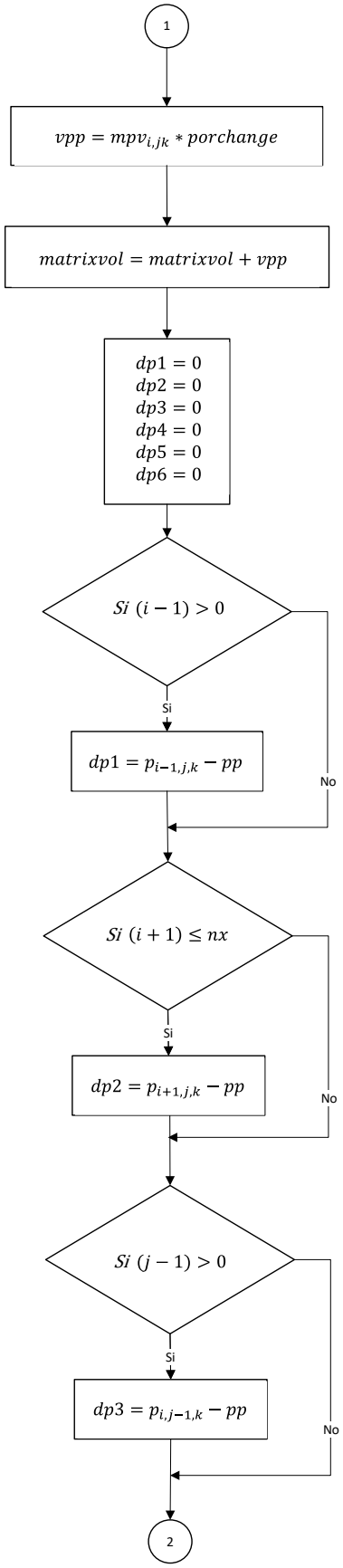


Ilustración 193:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

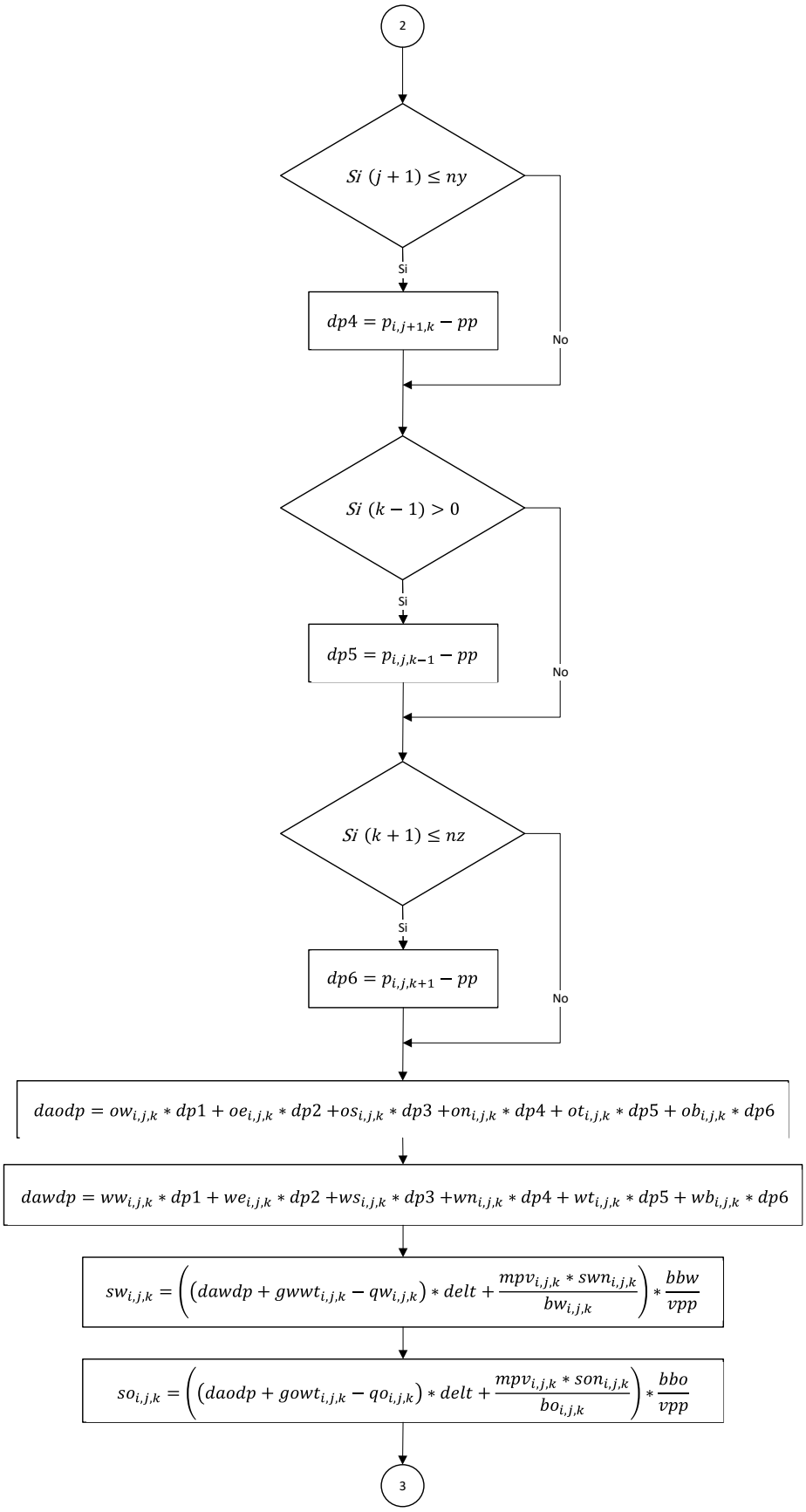


Ilustración 194:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

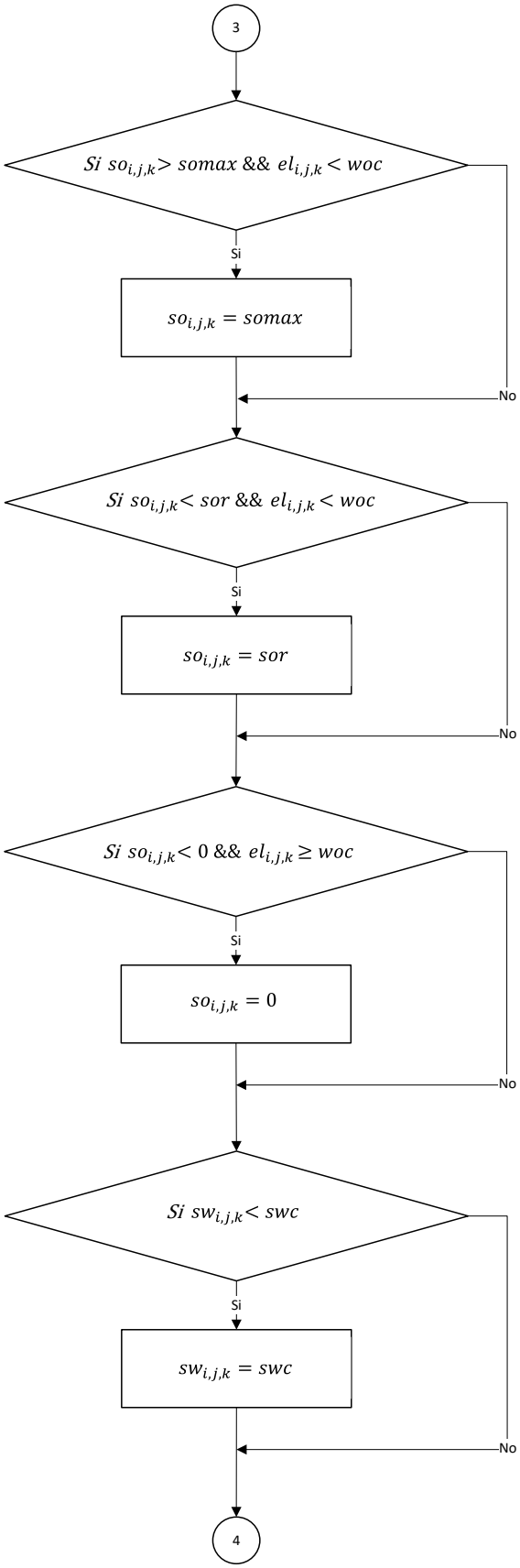


Ilustración 195:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

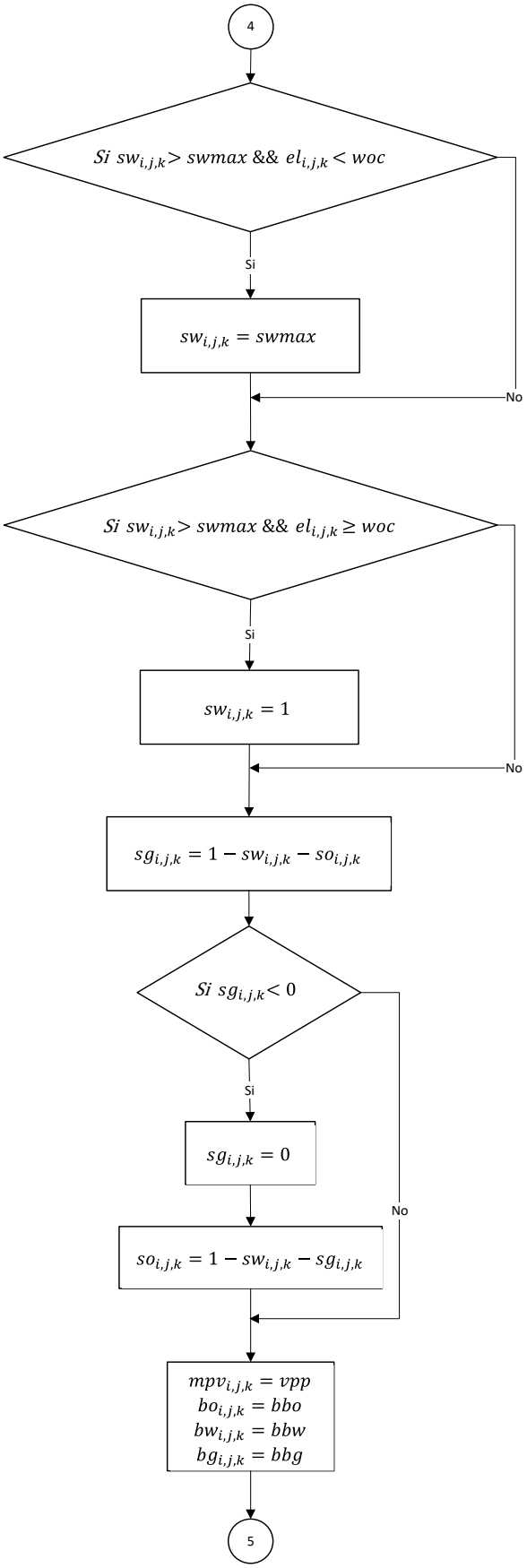


Ilustración 196:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

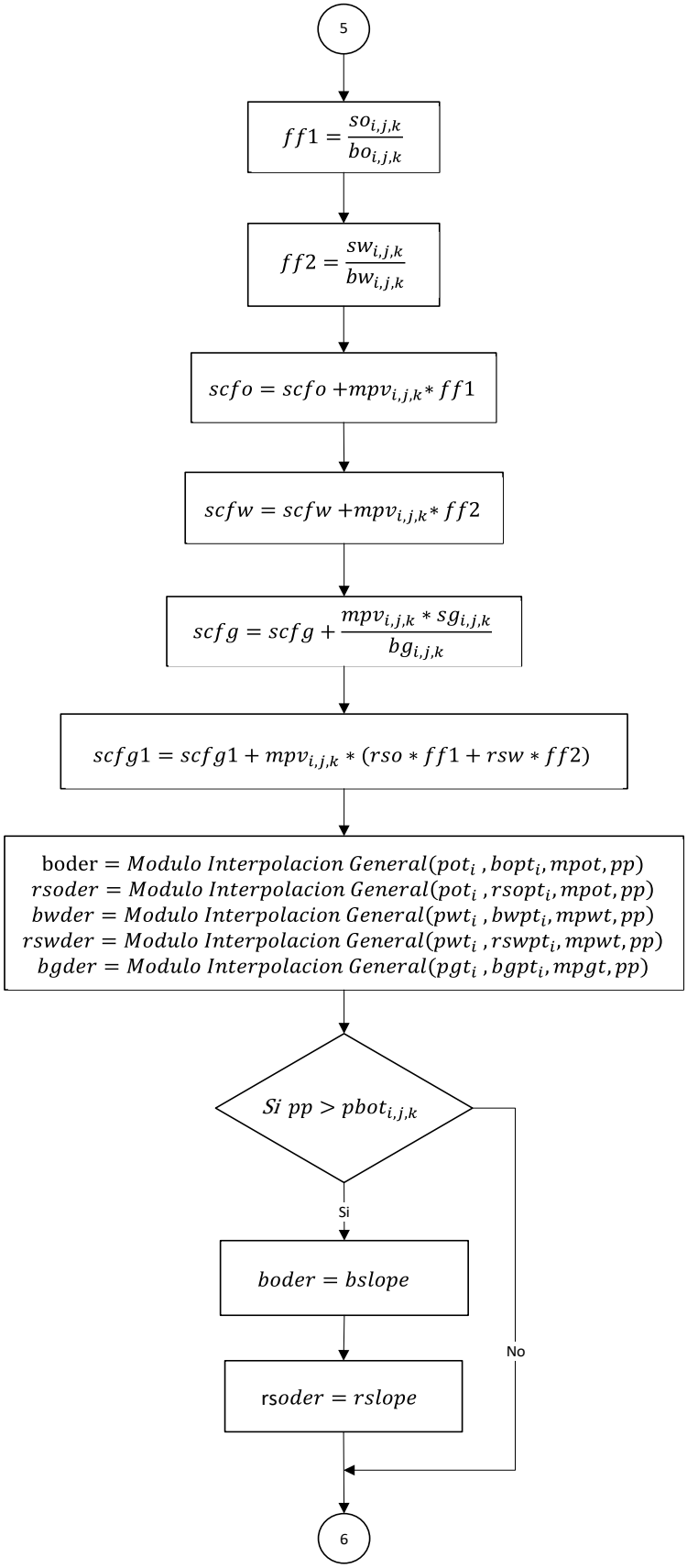


Ilustración 197:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

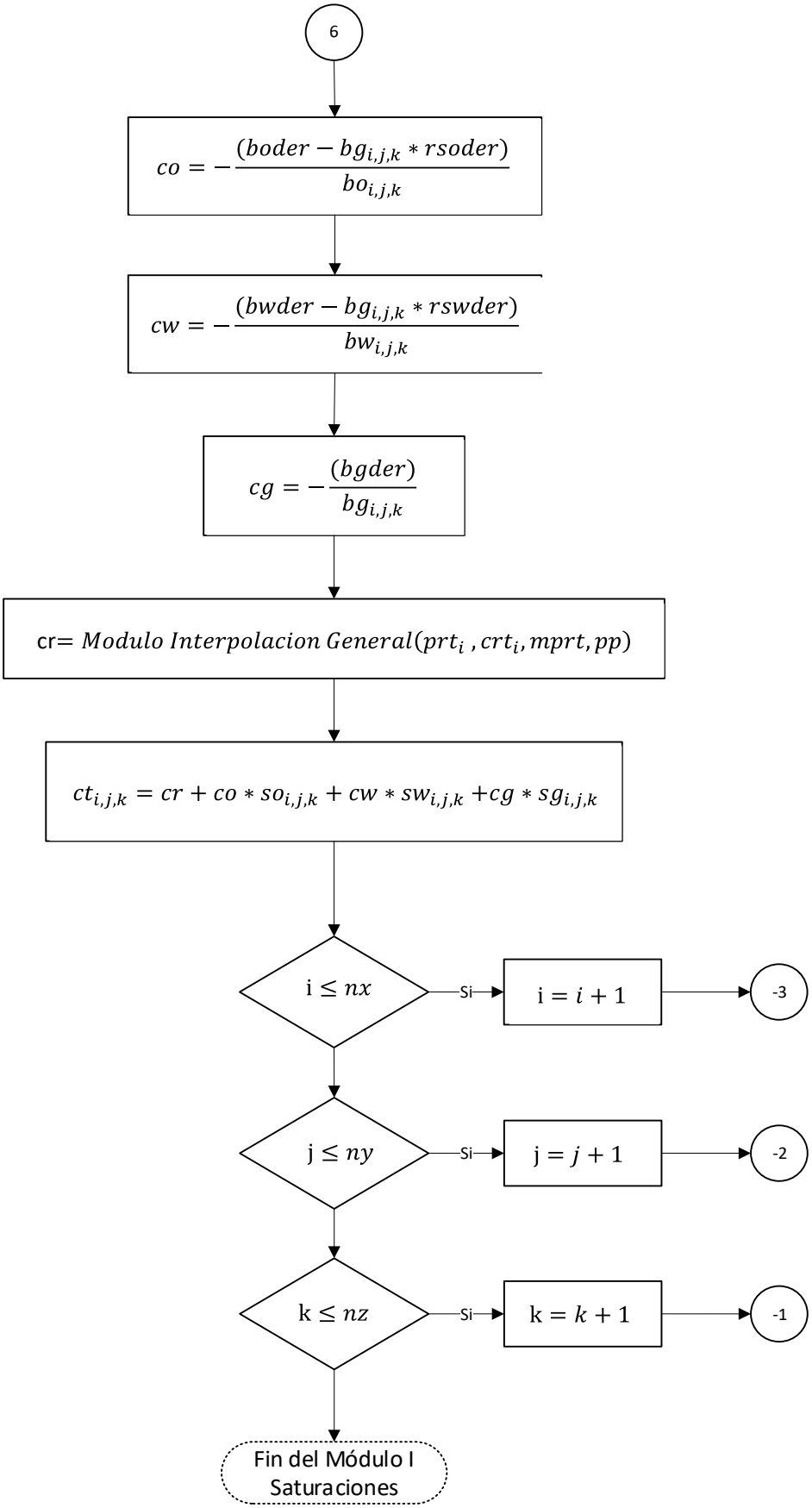


Ilustración 198:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

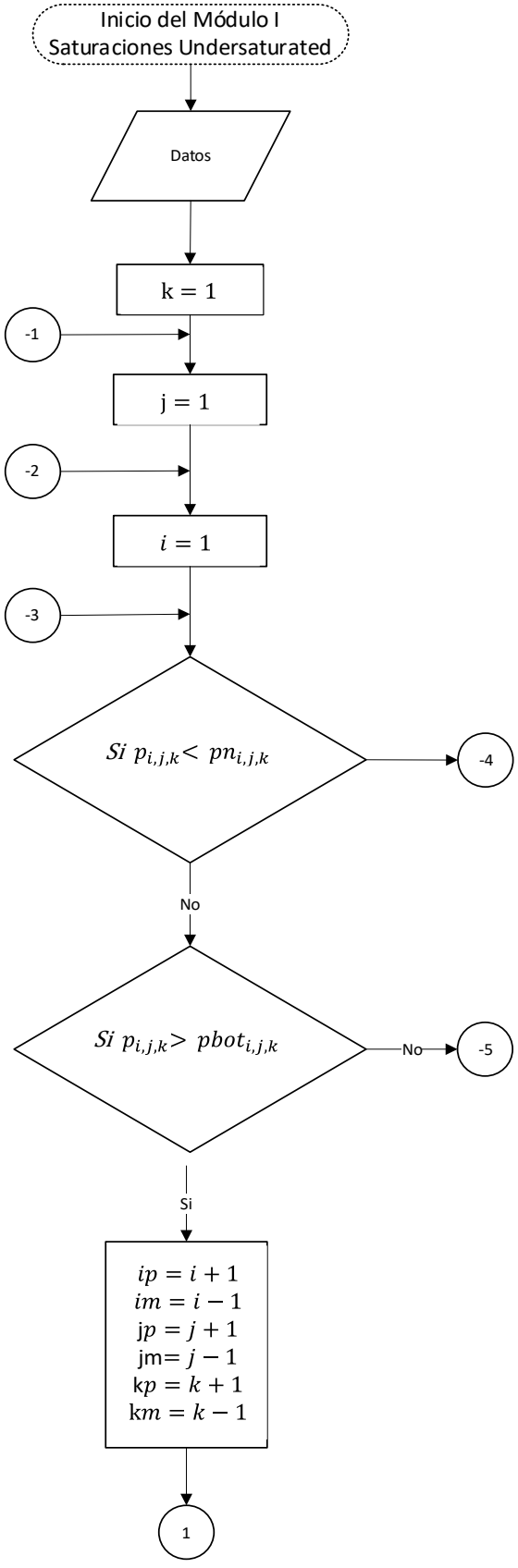


Ilustración 199:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

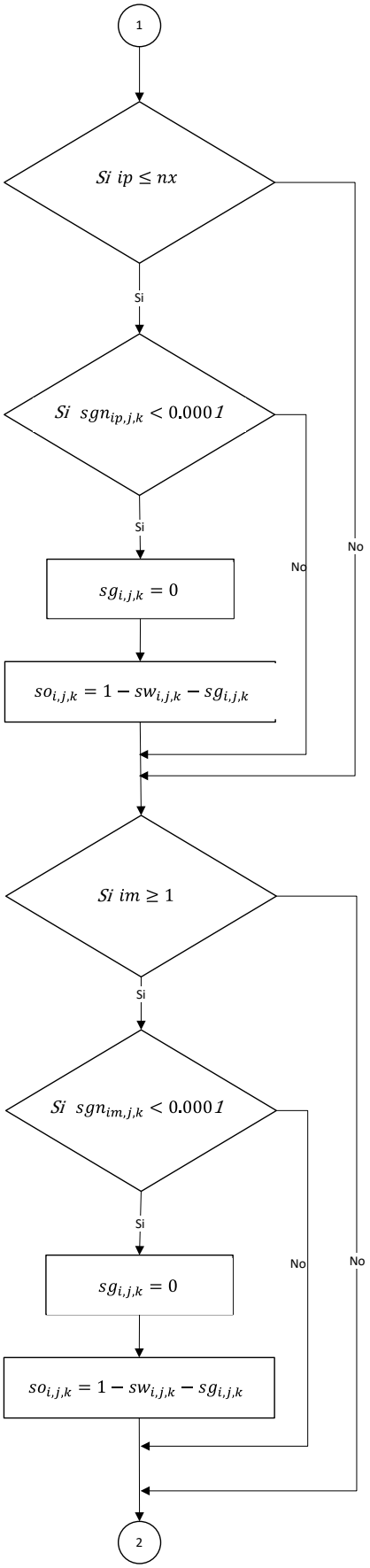


Ilustración 200:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

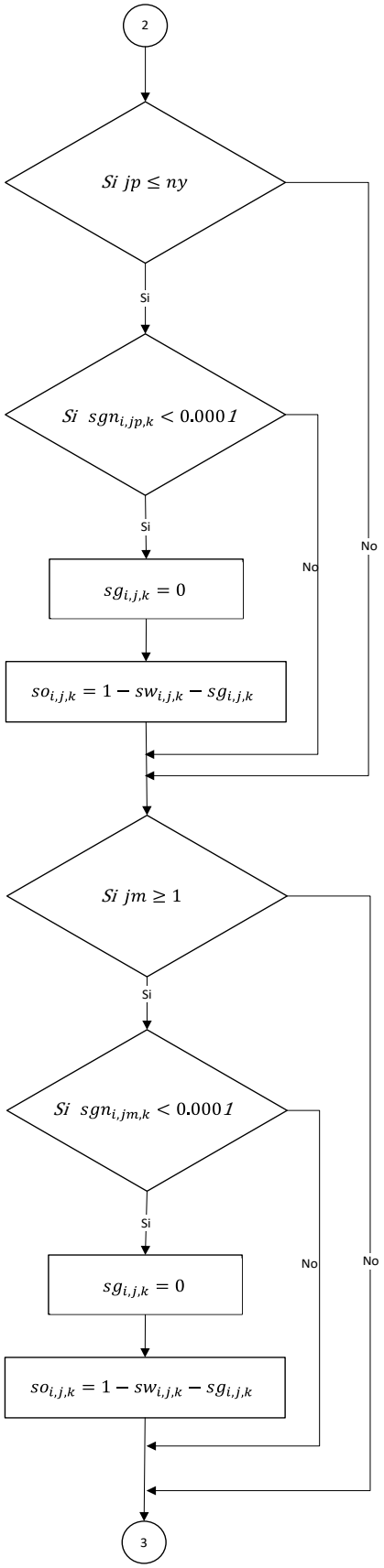


Ilustración 201:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

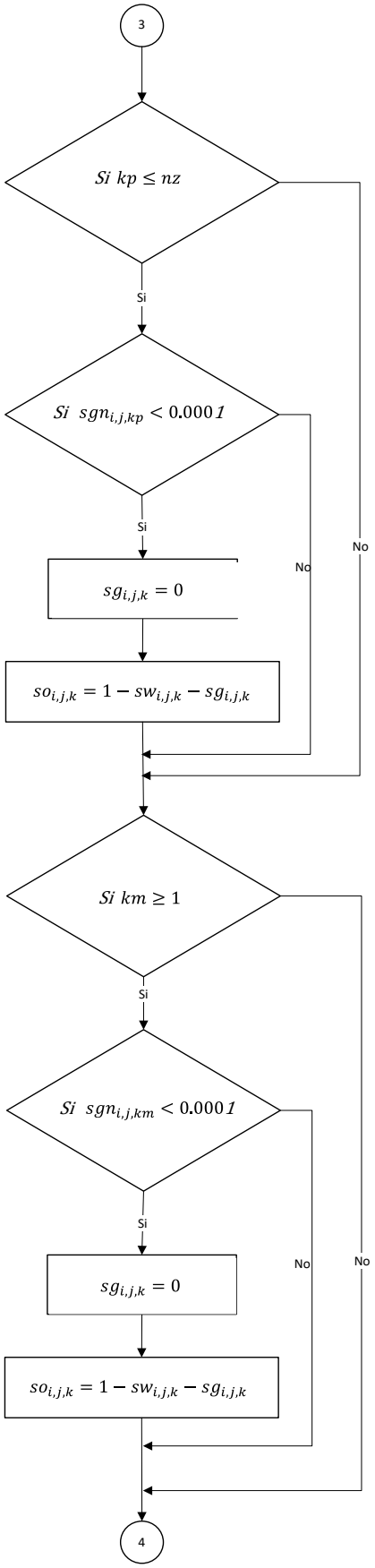


Ilustración 202:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

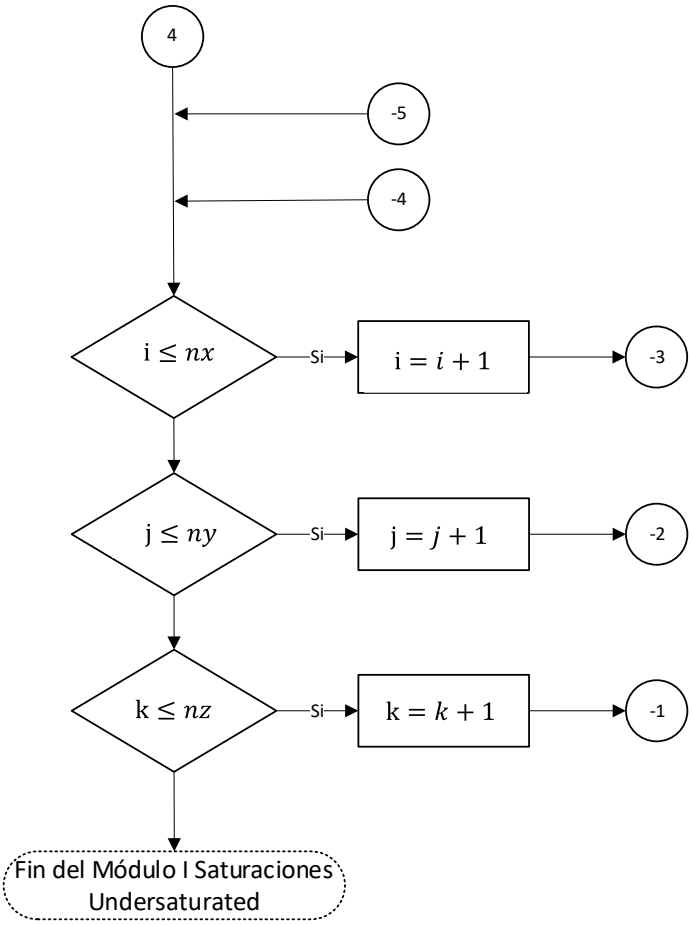


Ilustración 203:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

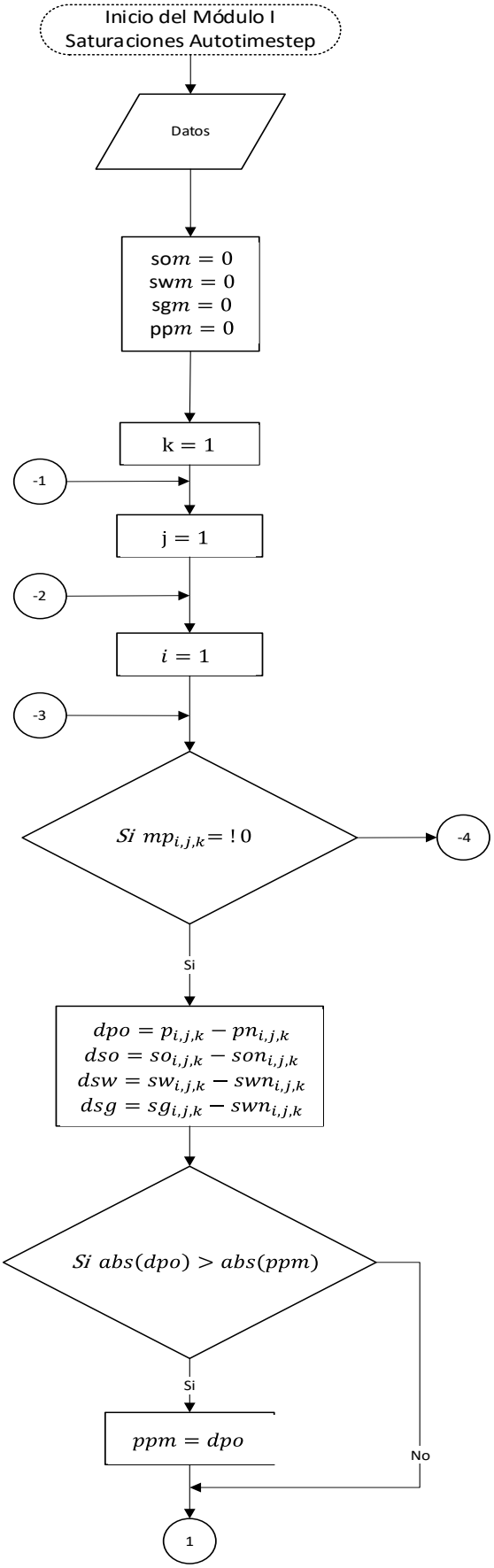


Ilustración 204:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

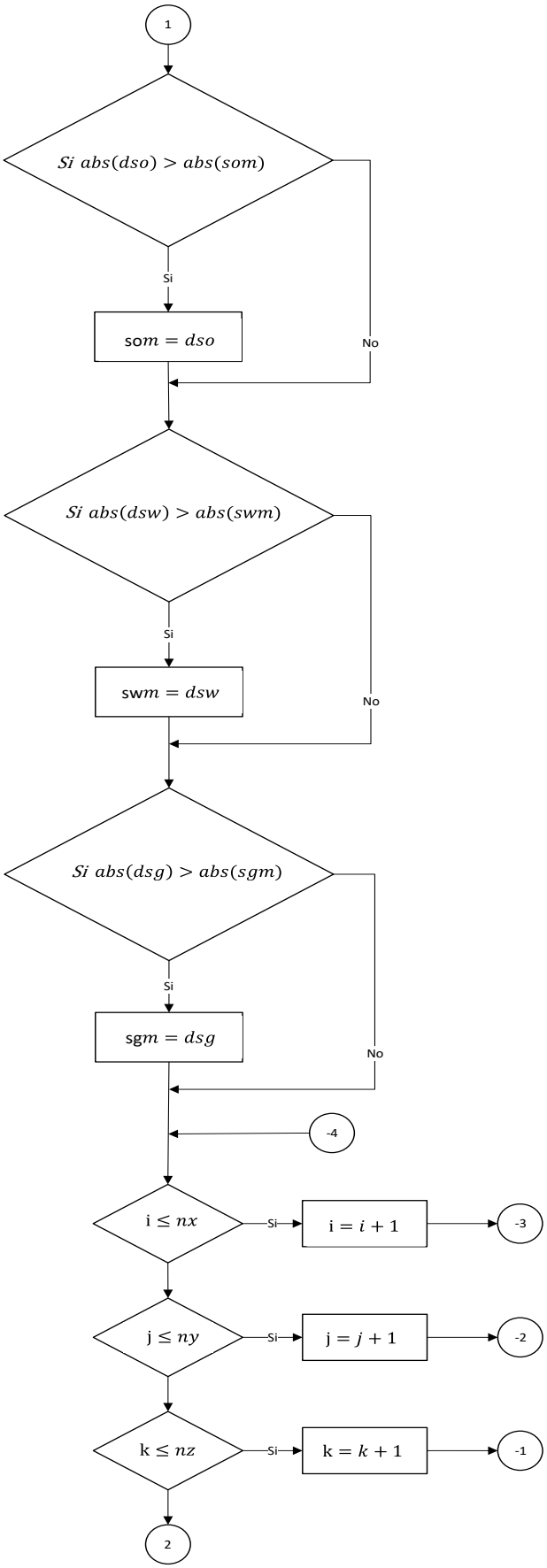


Ilustración 205:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

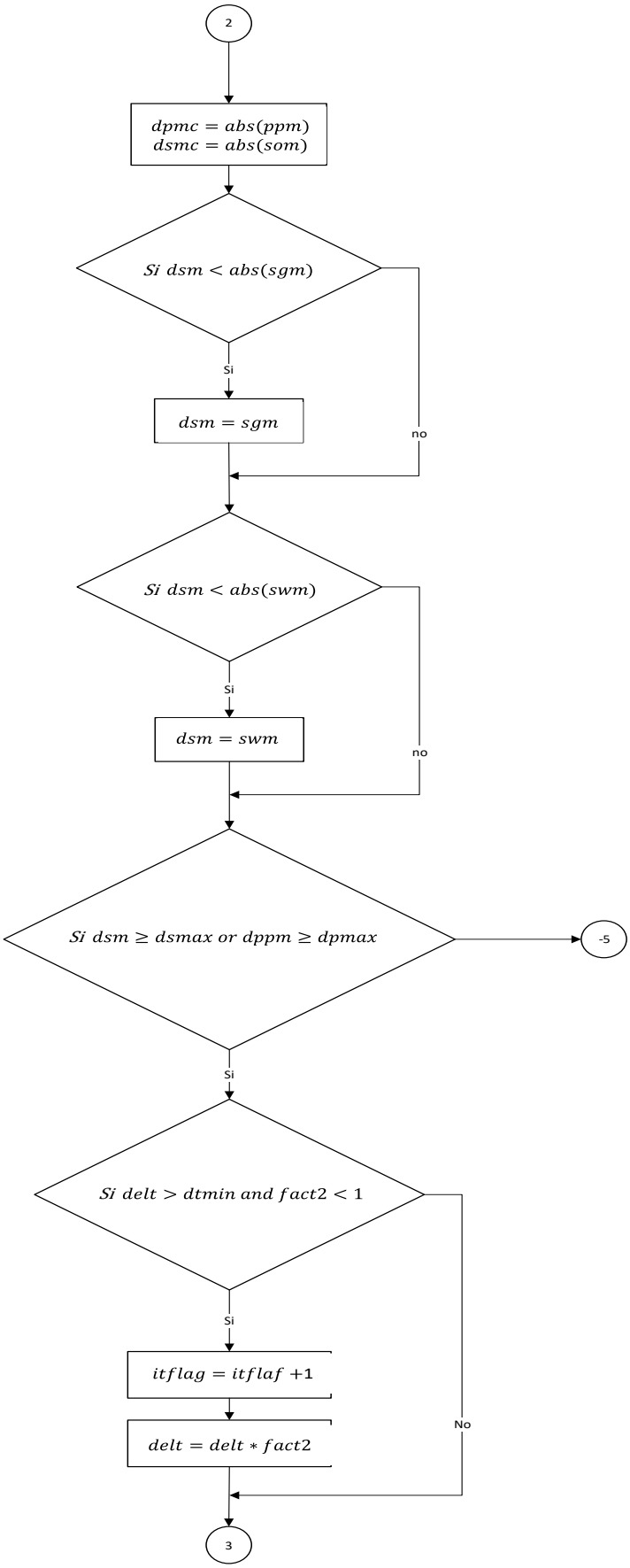


Ilustración 206:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

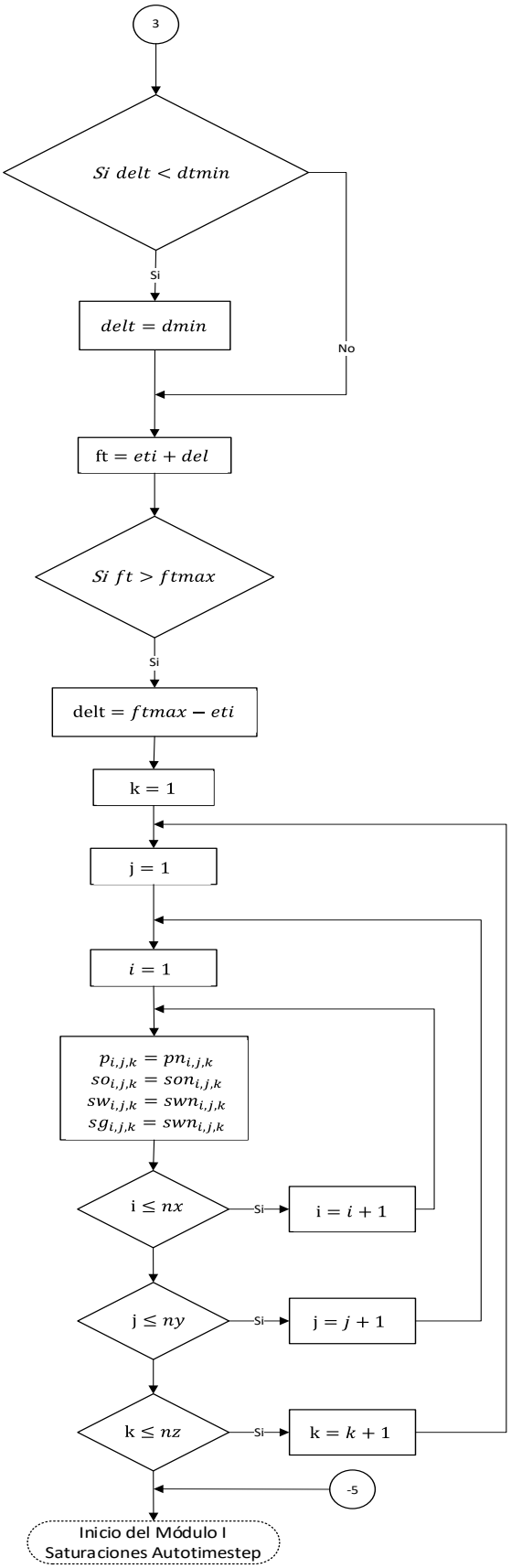


Ilustración 207:Continuación-Sección solvers de estimación Saturaciones Explicitas Fuente: Elaboración propia.

10.7. Descripción de la Configuración JOCL para procedimiento CPU-GPU del Simulador desarrollado en esta investigación

Para el despliegue del procesamiento GPU se procedió a activar la librería JOCL, que permite el uso de las librerías OpenCL para procesamiento paralelo CPU-GPU. Los códigos JOCL usados para realizar esto se muestran a continuación con algunos ejemplos de las secciones:

10.7.1. Módulo de Inicialización de las Librerías OpenCL

Para la Implementación del procesamiento GPU, primero debe realizarse la inicialización de estas librerías en JAVA, lo cual comprende la identificación de la plataforma de cómputo, número de dispositivos de cómputo disponible y versión OpenCL disponible en el computador. En las siguientes ilustraciones se muestra cómo se lleva a cabo esta inicialización:

```
private static void Jaresim_GPU_MR_JOCL_Section_initCL(Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var )
{

     System.out.println("Jaresim Report:Corriendo la Inicializacion del Modulo OPEN CL JOCL para procesamiento GPU");
     if(Jaresim_Report_ID!= null)Jaresim_Report_ID.getItems().add("Jaresim Report:Corriendo la Inicializacion del Modulo OPEN CL JOCL para procesamiento GPU");

     double start_run=System.currentTimeMillis();

    // The platform and device type that will be used
    final int platformIndex = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.platformIndex;
    final long deviceType=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.deviceType;

    // Enable exceptions and subsequently omit error checks in this sample
    CL.setExceptionsEnabled(true);

    // Obtain the number of platforms
    int numPlatformsArray[] = new int[1];
    clGetPlatformIDs(0, null, numPlatformsArray);
    int numPlatforms = numPlatformsArray[0];

    // Obtain a platform ID
    cl_platform_id platforms[] = new cl_platform_id[numPlatforms];
    clGetPlatformIDs(platforms.length, platforms, null);
    cl_platform_id platform = platforms[platformIndex];

    System.out.println("Jaresim Report: OPENCL usando Plataforma "+ Jaresim_GPU_MR_JOCL_Section_getPlatformInfoString(platform, CL.CL_PLATFORM_NAME));
    if(Jaresim_Report_ID!= null)Jaresim_Report_ID.getItems().add("Jaresim Report: OPENCL usando Plataforma "+ Jaresim_GPU_MR_JOCL_Section_getPlatformInfoString(platform, CL.CL

    // Initialize the context properties
    cl_context_properties contextProperties = new cl_context_properties();
    contextProperties.addProperty(CL_CONTEXT_PLATFORM, platform);

    // Obtain the number of devices for the platform
    int numDevicesArray[] = new int[1];
    clGetDeviceIDs(platform, deviceType, 0, null, numDevicesArray);
    numDevices = numDevicesArray[0];

    // Obtain the all device IDs
    cl_device_id all_Devices_available[] = new cl_device_id[numDevices];
    cl_device_id allDevices[] = new cl_device_id[Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.numDevices];
    clGetDeviceIDs(platform, deviceType, numDevices, all_Devices_available, null);
```

Ilustración 208:Configuración de Inicialización Modulo OpenCL en JAVA. Fuente: Elaboración propia.

```
int index_dev=0;

for (int i=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.min_device;i<=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.max_device;i++)
{
    allDevices[index_dev]=all_Devices_available[i];
    index_dev++;
}

numDevices= Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.numDevices;
all_Devices_available=null;

// Create a context for the devices

context= clCreateContext(
    contextProperties, numDevices, allDevices,
    null, null, null);

// Find the first device that supports OpenCL >1.2
int counter=0;

for (cl_device_id currentDevice : allDevices)
{
    String deviceName = Jaresim_GPU_MR_JOCL_Section_getString(currentDevice, CL_DEVICE_NAME);
    double version = Jaresim_GPU_MR_JOCL_Section_getOpenCLVersion(currentDevice);

    counter=counter+1;

    if (version <= 2)
    {
        System.out.println("Jaresim Report:Usando Dispositivo  "+
            deviceName+", version "+version);
        device = currentDevice;
        if(Jaresim_Report_ID!= null)Jaresim_Report_ID.getItems().add("Jaresim Report:Usando Dispositivo  "+
                deviceName+", version "+version +"\n");

    }
    else
    {
        System.out.println("Jaresim Report: Dispositivo Omitido  "+
            deviceName+", version "+version);
        if(Jaresim_Report_ID!= null)Jaresim_Report_ID.getItems().add("Jaresim Report: Dispositivo Omitido  "+
                deviceName+", version "+version);
    }

}
```

Ilustración 209:Continuación-Configuración de Inicialización Modulo OpenCL en JAVA. Fuente: Elaboración propia.

```
if (numDevices == 0)
{
    System.out.println("Jaresim Report: No OpenCL 2.0 capable device found");
    if(Jaresim_Report_IO!= null)Jaresim_Report_IO.getItems().add("Jaresim Report: No OpenCL 2.0 capable device found");
    System.exit(1);
}


//*****************************************************************
// Read the kernel files and set up the OpenCL program

Path currentRelativePath = Paths.get("");
String Path = currentRelativePath.toAbsolutePath().toString();

String Jaresim_GPU_MR_Section_B_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_B_Section.cl");
String Jaresim_GPU_MR_Section_D_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_D_Section.cl");
String Jaresim_GPU_MR_Section_E_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_E_Section.cl");
String Jaresim_GPU_MR_Section_J_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_J_Section.cl");
String Jaresim_GPU_MR_Section_I_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_I_Section.cl");
String Jaresim_GPU_MR_Section_F_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_F_Section.cl");
String Jaresim_GPU_MR_Section_G_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_G_Section.cl");
String Jaresim_GPU_MR_Section_H_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_H_Section.cl");
String Jaresim_GPU_MR_Section_P_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_P_Section.cl");
String Jaresim_GPU_MR_Section_O_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_O_Section.cl");
String Jaresim_GPU_MR_Section_R_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_R_Section.cl");
String Jaresim_GPU_MR_Section_T_file = Jaresim_GPU_MR_JOCL_Section_readFile(Path+"/kernels_jaresim/Jaresim_GPU_MR_T_Section.cl");

System.out.println("Jaresim Report:Lectura de archivos contentivos del Kernel Completado");
if(Jaresim_Report_IO!= null)Jaresim_Report_IO.getItems().add("Jaresim Report:Lectura de archivos contentivos del Kernel Completado");

// Create the program from the source code

commandQueue = new cl_command_queue [numDevices];
events = new cl_event[numDevices];

kernel_Section_B=new cl_kernel [3];
kernel_Section_D=new cl_kernel [3];
kernel_Section_E=new cl_kernel [1];
kernel_Section_J=new cl_kernel [4];
kernel_Section_I=new cl_kernel [7];
kernel_Section_F=new cl_kernel [1];
kernel_Section_G=new cl_kernel [1];
kernel_Section_H=new cl_kernel [2];
kernel_Section_P=new cl_kernel [3];
kernel_Section_T=new cl_kernel [1];
```

```
program_Jaresim_GPU_MR= clCreateProgramWithSource(context, 12,
    new String[]{
                 Jaresim_GPU_MR_Section_R_file,
                 Jaresim_GPU_MR_Section_O_file,
                 Jaresim_GPU_MR_Section_D_file,
                 Jaresim_GPU_MR_Section_E_file,
                 Jaresim_GPU_MR_Section_J_file,
                 Jaresim_GPU_MR_Section_I_file,
                 Jaresim_GPU_MR_Section_F_file,
                 Jaresim_GPU_MR_Section_G_file,
                 Jaresim_GPU_MR_Section_H_file,
                 Jaresim_GPU_MR_Section_P_file,
                 Jaresim_GPU_MR_Section_T_file,
                 Jaresim_GPU_MR_Section_B_file
                 }, null, null);
clBuildProgram( program_Jaresim_GPU_MR, 0, null,null, null, null);

cl_queue_properties properties = new cl_queue_properties();

System.out.println("Jaresim Report:Creacion de Kernel Completado");
if(Jaresim_Report_IO!= null)Jaresim_Report_IO.getItems().add("Jaresim Report:Creacion de Kernel Completado");

for (int i=0 ;i<numDevices;i++)
{

//*****************************************************************
// Create the command queue

    commandQueue[i] = clCreateCommandQueueWithProperties(
    context, allDevices[i], properties, null);

}

// Create all kernels for the All Sections for the reservoir

kernel_Section_B[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_B_Inputs_Data_Leer_Datos_Recurrentes", null);
kernel_Section_B[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_B_Inputs_Data_Leer_Pozos", null);
kernel_Section_B[2] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_B_Inputs_Data_Field_Time", null);

// Section D for jaresim reservoir: Transmisibillity Section

kernel_Section_D[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_D_Transmissibility", null);
kernel_Section_D[2] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_D_Faults", null);
kernel_Section_D[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_D_Slopes_for_compressibilities", null);
```

Ilustración 210:Continuación-Configuración de Inicialización Modulo OpenCL en JAVA. Fuente: Elaboración propia.

```
program_Jaresim_GPU_MR= clCreateProgramWithSource(context, 12,
    new String[]{
                Jaresim_GPU_MR_Section_R_file,
                Jaresim_GPU_MR_Section_O_file,
                Jaresim_GPU_MR_Section_D_file,
                Jaresim_GPU_MR_Section_E_file,
                Jaresim_GPU_MR_Section_J_file,
                Jaresim_GPU_MR_Section_I_file,
                Jaresim_GPU_MR_Section_F_file,
                Jaresim_GPU_MR_Section_G_file,
                Jaresim_GPU_MR_Section_H_file,
                Jaresim_GPU_MR_Section_P_file,
                Jaresim_GPU_MR_Section_T_file,
                Jaresim_GPU_MR_Section_B_file
                }, null, null);
clBuildProgram( program_Jaresim_GPU_MR, 0, null,null, null, null);

cl_queue_properties properties = new cl_queue_properties();

System.out.println("Jaresim Report:Creacion de Kernel Completado");
if(Jaresim_Report_ID!= null)Jaresim_Report_ID.getItems().add("Jaresim Report:Creacion de Kernel Completado");

for (int i=0 ;i<numDevices;i++)
{

//*********************************************************************
// Create the command queue

    commandQueue[i] = clCreateCommandQueueWithProperties(
    context, allDevices[i], properties, null);

}

// Create all kernels for the All Sections for the reservoir

kernel_Section_B[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_B_Inputs_Data_Leer_Datos_Recurrentes", null);
kernel_Section_B[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_B_Inputs_Data_Leer_Pozos", null);
kernel_Section_B[2] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_B_Inputs_Data_Field_Time", null);

// Section D for jaresim reservoir: Transmisibillity Section

kernel_Section_D[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_D_Transmissibility", null);
kernel_Section_D[2] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_D_Faults", null);
kernel_Section_D[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_D_Slopes_for_compressibilities", null);

 // Section E for jaresim reservoir: Initialization Section

 kernel_Section_E[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_E_Initialization", null);

 // Section J for jaresim reservoir: Volumetrics Section

 kernel_Section_J[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_J_Modulo_Volumenes_Iniciales", null);
 kernel_Section_J[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_J_Modulo_Balance_de_Materia", null);
 kernel_Section_J[2] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_J_Modulo_Balance_de_Materia_Update", null);
 kernel_Section_J[3] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_J_Modulo_Volumenes_Iniciales_Update", null);

 // Section I for jaresim reservoir: Saturations Calculations Section

 kernel_Section_I[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_I_Modulo_Saturacion_Represurizacion", null);
 kernel_Section_I[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_I_Modulo_Saturacion_Auto_time_step_Part_I", null);
 kernel_Section_I[2] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_I_Modulo_Saturacion_Undersaturated_grid", null);
 kernel_Section_I[3] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_I_Modulo_Saturacion_Saturaciones_Part_I", null);
 kernel_Section_I[4] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_I_Modulo_Saturacion_Auto_time_step_Part_II", null);
 kernel_Section_I[5] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_I_Modulo_Saturacion_Saturaciones_Part_II", null);
 kernel_Section_I[6] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_I_Modulo_Saturacion_Auto_time_step_Part_III", null);

 // Section F for jaresim reservoir:QRate Section

 kernel_Section_F[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_F_Qrate", null);

 // Section G for jaresim reservoir: Coefficients Section

 kernel_Section_G[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_G_Coefficientes", null);

 // Section H for jaresim reservoir: Pratei-Prateo Section

 kernel_Section_H[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_H_Pratei", null);
 kernel_Section_H[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_H_Prateo", null);

 // Section P for jaresim reservoir: LSOR Section

 kernel_Section_P[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_P_LSOR", null);
 kernel_Section_P[1] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_P_LSOR_Part_II", null);
 kernel_Section_P[2] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_P_LSOR_Part_III", null);

 // Section T for jaresim reservoir: Actualization Section

 kernel_Section_T[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_T_Modulo_Actualizacion", null);
```

Ilustración 211:Continuación-Configuración de Inicialización Modulo OpenCL en JAVA. Fuente: Elaboración propia.

```
    // Section T for jaresim reservoir: Actualization Section

    kernel_Section_T[0] = clCreateKernel( program_Jaresim_GPU_MR, "Jaresim_GPU_MR_T_Modulo_Actualizacion", null);


    System.out.println("Jaresim Report:Creacion del Kernel GPU Completado");
    if(Jaresim_Report_ID!= null)Jaresim_Report_ID.getItems().add("Jaresim Report:Creacion del Kernel GPU Completado");

    double stop_run=System.currentTimeMillis();

    System.out.println("Jaresim Report:Inicializacion del Modulo JOCL para procesamiento GPU completado en "+ (start_run-stop_run)/1000 + " seg:");
    if(Jaresim_Report_ID!= null)Jaresim_Report_ID.getItems().add("Jaresim Report:Inicializacion del Modulo JOCL para procesamiento GPU completado en

}
```

Ilustración 212:Continuación-Configuración de Inicialización Modulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.2. Módulo de Cálculos de Compresibilidad

En la siguiente ilustración puede ser evidenciado el código JAVA para la llamada del Archivo *.CL que contienen el código paralelizado del Módulo de Compresibilidad:

```
public void Jaresim_GPU_MR_D_Slopes_for_compressibilities(
    Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
    // Gas PVT Properties
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pgt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_mugt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgpt[],
    // Oil PVT Properties
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pot[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muot[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bot[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsot[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bopt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsopt[],
    // Water PVT Properties
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pwt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muwt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwpt[],
    double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswpt[]
)
{

       // Asignacion de los Argumentos de la para el calculo de la compresibilidad

        // Set the work-item dimensions
        long globalWorkSize[] = new long[1];
        globalWorkSize[0]=Jaresim_Input_eLements[14]+Jaresim_Input_eLements[18]+Jaresim_Input_eLements[29];

        for (int w=0;w<numDevices;w++)
        {

         Jaresim_Input_Buffer[0][w] = Buffers.aLLocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);

         Buffers.writeToBuffer(Jaresim_Input_Buffer[0][w] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);

         input_Mems_GPU[0][w] = clCreateBuffer(context, CL_MEM_READ_WRITE  | CL_MEM_COPY_HOST_PTR,structSize_jaresim[0]

         clSetKernelArg(kernel_Section_D[0], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));

         clSetKernelArg(kernel_Section_D[0], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[14][w]));
         clSetKernelArg(kernel_Section_D[0], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[15][w]));
         clSetKernelArg(kernel_Section_D[0], 3, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[16][w]));
         clSetKernelArg(kernel_Section_D[0], 4, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[17][w]));
```

Ilustración 213:Configuración de la sección Compresibilidad implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
            clSetKernelArg(kernel_Section_D[0], 5, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[18][w]));
            clSetKernelArg(kernel_Section_D[0], 6, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[19][w]));
            clSetKernelArg(kernel_Section_D[0], 7, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[20][w]));
            clSetKernelArg(kernel_Section_D[0], 8, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[21][w]));
            clSetKernelArg(kernel_Section_D[0], 9, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[22][w]));
            clSetKernelArg(kernel_Section_D[0], 10, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[23][w]));

            clSetKernelArg(kernel_Section_D[0], 11, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[29][w]));
            clSetKernelArg(kernel_Section_D[0], 12, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[30][w]));
            clSetKernelArg(kernel_Section_D[0], 13, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[31][w]));
            clSetKernelArg(kernel_Section_D[0], 14, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[32][w]));
            clSetKernelArg(kernel_Section_D[0], 15, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[33][w]));
            clSetKernelArg(kernel_Section_D[0], 16, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[34][w]));

        // Execute event
            if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

        // Execute the kernel
        clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_D[0], 1, null,globalWorkSize,null, 0, null, events[w]);

        }

        if (numDevices>1)
        {
        // Wait until the work is finished on all command queues
            clWaitForEvents(events.length, events);
        }


    }
```

Ilustración 214:Continuación-Configuración de la sección Compresibilidad implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.3. Módulo de Cálculo de Transmisibilidad

En la siguiente ilustración se puede evidenciar el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de Transmisibilidad:

```
public void Jaresim_GPU_MR_D_Section_Transmisibility(
  Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
  Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[],
  Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Transm Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[]
 )
{
    // System.out.println("Iniciando la Corrida de los Datos de Transmisibilidad de la Seccion D");

     double start_run=System.currentTimeMillis();

    //  Set the work-item dimensions
    long globalWorkSize[] = new long[2];
    globalWorkSize[0]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx+1;
    globalWorkSize[1]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny+1;

    for (int w=0;w<numDevices;w++)
    {

      //Jaresim_Input_Buffer[0][w] = Buffers.allocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
      Jaresim_Input_Buffer[5][w] = Buffers.allocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D);
      Jaresim_Input_Buffer[6][w] = Buffers.allocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var);

      //  Buffers.writeToBuffer(Jaresim_Input_Buffer[0] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
      Buffers.writeToBuffer(Jaresim_Input_Buffer[5][w] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D);
      Buffers.writeToBuffer(Jaresim_Input_Buffer[6][w] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var);

      //    input_Mems_GPU[0] = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_USE_HOST_PTR,structSize_jaresim[0]*Jaresim_Input_elements[0
      input_Mems_GPU[5][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,structSize_jaresim[5]*Jaresim_Input_elements[5],
      input_Mems_GPU[6][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,structSize_jaresim[6]*Jaresim_Input_elements[6],

      //  Asignacion de los Argumentos de la para el calculo de la transmiibilidad
      clSetKernelArg(kernel_Section_D[1], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
      clSetKernelArg(kernel_Section_D[1], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
      clSetKernelArg(kernel_Section_D[1], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[6][w]));

      // Execute event
      if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

      //  Execute the kernel
      clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_D[1], 2, null,globalWorkSize,null, 0, null, events[w]);

    }

    // Wait until the work is finished on all command queues
    if (numDevices>1) clWaitForEvents(events.length, events);
```

Ilustración 215: configuración de la sección Transmisibilidad implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
    // Wait until the work is finished on all command queues
    if (numDevices>1) clWaitForEvents(events.length, events);

    for (int w=0;w<numDevices;w++)
    {
      //  Read back the data from to memory object to the transmissibility buffer
      clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[0][w], CL_TRUE, 0, structSize_jaresim[0]*Jaresim_Input_elements[0], Pointe
      clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[5][w], CL_TRUE, 0, structSize_jaresim[5]*Jaresim_Input_elements[5], Pointe
      clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[6][w], CL_TRUE, 0, structSize_jaresim[6]*Jaresim_Input_elements[6], Pointe

      //  Read the data from the output jaresim data buffer back into the output transmissibility section
      Jaresim_Input_Buffer[0][w].rewind();
      Jaresim_Input_Buffer[5][w].rewind();
      Jaresim_Input_Buffer[6][w].rewind();
      Buffers.readFromBuffer(Jaresim_Input_Buffer[0][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
      Buffers.readFromBuffer(Jaresim_Input_Buffer[5][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D);
      Buffers.readFromBuffer(Jaresim_Input_Buffer[6][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var);

    }

   double stop_run=System.currentTimeMillis();

    System.out.println("Corrida de los Datos de Transmisibilidad de la Seccion D completado en seg:"+ (stop_run-start_run)/1000);

}
```

Ilustración 216:Continuación-configuración de la sección Transmisibilidad implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.4. Módulo de Cálculo de Inicialización

En la siguiente ilustración puede ser evidenciado el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de Inicialización:

```
public void Jaresim_GPU_MR_E_Initialization(
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Run Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[],
        // Gas PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_mugt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgpt[],
        // Oil PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bopt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsopt[],
        // Water PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwpt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswpt[]
    )
    {

         double start_run=System.currentTimeMillis();

         // Set the work-item dimensions
         long globalWorkSize[] = new long[3];
         globalWorkSize[0]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx+1;
         globalWorkSize[1]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny+1;
         globalWorkSize[2]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+1;
```

Ilustración 217:Configuración de la sección Inicialización implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
    long localWorkSize[] = new long[2];
    localWorkSize[0]=12;
    localWorkSize[1]=12;

   for (int w=0;w<numDevices;w++)
   {

     // Actualizacion del Buffer de los Argumentos de la para el calculo de la Inicializacion
     Jaresim_Input_Buffer[7][w] = Buffers.allocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run);

     Buffers.writeToBuffer(Jaresim_Input_Buffer[7][w] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run);

     input_Mems_GPU[7][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR,structSize_jaresim[7]*Jar

     // Asignacion de los Argumentos de la para el calculo de la Inicializacion
    clSetKernelArg(kernel_Section_E[0], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
    clSetKernelArg(kernel_Section_E[0], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
    clSetKernelArg(kernel_Section_E[0], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));

    clSetKernelArg(kernel_Section_E[0], 3, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[14][w]));
    clSetKernelArg(kernel_Section_E[0], 4, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[15][w]));
    clSetKernelArg(kernel_Section_E[0], 5, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[16][w]));
    clSetKernelArg(kernel_Section_E[0], 6, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[17][w]));

    clSetKernelArg(kernel_Section_E[0], 7, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[18][w]));
    clSetKernelArg(kernel_Section_E[0], 8, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[19][w]));
    clSetKernelArg(kernel_Section_E[0], 9, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[20][w]));
    clSetKernelArg(kernel_Section_E[0], 10, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[21][w]));
    clSetKernelArg(kernel_Section_E[0], 11, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[22][w]));
    clSetKernelArg(kernel_Section_E[0], 12, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[23][w]));

    clSetKernelArg(kernel_Section_E[0], 13, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[29][w]));
    clSetKernelArg(kernel_Section_E[0], 14, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[30][w]));
    clSetKernelArg(kernel_Section_E[0], 15, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[31][w]));
    clSetKernelArg(kernel_Section_E[0], 16, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[32][w]));
    clSetKernelArg(kernel_Section_E[0], 17, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[33][w]));
    clSetKernelArg(kernel_Section_E[0], 18, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[34][w]));

   // Execute event
    if (numDevices>1 && events[w]==null)events[w] = new cl_event();

   // Execute the kernel
   clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_E[0], 3, null,globalWorkSize, null, 0, null, events[w]);
  }

   // Wait until the work is finished on all command queues
    if (numDevices>1) clWaitForEvents(events.length, events);

    for (int w=0;w<numDevices;w++)
    {

      // Read back the data from to memory object to the initialization buffer
      clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[0][w], CL_TRUE, 0, structSize_jaresim[0]*Jaresim_Input_elements[0],
      clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[5][w], CL_TRUE, 0, structSize_jaresim[5]*Jaresim_Input_elements[5],
      clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[7][w], CL_TRUE, 0, structSize_jaresim[7]*Jaresim_Input_elements[7],

      // Read the data from the output jaresim data buffer back into the output initialization  section
      Jaresim_Input_Buffer[0][w].rewind();
      Jaresim_Input_Buffer[5][w].rewind();
      Jaresim_Input_Buffer[7][w].rewind();

      Buffers.readFromBuffer(Jaresim_Input_Buffer[0][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
      Buffers.readFromBuffer(Jaresim_Input_Buffer[5][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D);
      Buffers.readFromBuffer(Jaresim_Input_Buffer[7][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run);

    }

}
```

Ilustración 218:Continuación-Configuración de la sección Inicialización implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.5. Módulo de Cálculo de Tasas asociada a los cálculos de la malla

En la siguiente ilustración puede ser evidenciado el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de Cálculo de Tasas asociadas a los cálculos de malla:

```
public void Jaresim_GPU_MR_F_Qrate(int iweltype,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_Run_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_Run_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Run Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[],
        // Gas PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_mugt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgpt[],
        // Oil PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bopt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsopt[],
        // Water PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwpt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswpt[],
        // Matrix Kr and Pc properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_sat[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pcowt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pcgot[],
        // Well Data Parameters
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_nw Datos_Jaresim_GPU_MR_Proyecto_JOCL_nw_var[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_nw_nl Datos_Jaresim_GPU_MR_Proyecto_JOCL_nw_nl_var[]
    )
    {

    // Set the work-item dimensions
    long globalWorkSize[] = new long[1];
    globalWorkSize[0]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nw+1;

     for (int w=0;w<numDevices;w++)
     {

          Jaresim_Input_Buffer[0][w] = Buffers.allocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
     Buffers.writeToBuffer(Jaresim_Input_Buffer[0][w] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
     if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.eti>=0)  input_Mems_GPU[0][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_USE_HOST_PTR,

    // Asignacion de los Argumentos de La Funcion FRate de la Seccion F
     clSetKernelArg(kernel_Section_F[0], 0, Sizeof.cl_int, Pointer.to(new int[]{ iweltype }));
     clSetKernelArg(kernel_Section_F[0], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
     clSetKernelArg(kernel_Section_F[0], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[36][w]));
     clSetKernelArg(kernel_Section_F[0], 3, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
     clSetKernelArg(kernel_Section_F[0], 4, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));

     clSetKernelArg(kernel_Section_F[0], 5, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[14][w]));
     clSetKernelArg(kernel_Section_F[0], 6, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[15][w]));
     clSetKernelArg(kernel_Section_F[0], 7, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[16][w]));
     clSetKernelArg(kernel_Section_F[0], 8, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[17][w]));

     clSetKernelArg(kernel_Section_F[0], 9, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[18][w]));
     clSetKernelArg(kernel_Section_F[0], 10, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[19][w]));
     clSetKernelArg(kernel_Section_F[0], 11, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[20][w]));
     clSetKernelArg(kernel_Section_F[0], 12, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[21][w]));
     clSetKernelArg(kernel_Section_F[0], 13, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[22][w]));
     clSetKernelArg(kernel_Section_F[0], 14, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[23][w]));

     clSetKernelArg(kernel_Section_F[0], 15, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[29][w]));
     clSetKernelArg(kernel_Section_F[0], 16, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[30][w]));
     clSetKernelArg(kernel_Section_F[0], 17, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[31][w]));
     clSetKernelArg(kernel_Section_F[0], 18, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[32][w]));
     clSetKernelArg(kernel_Section_F[0], 19, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[33][w]));
     clSetKernelArg(kernel_Section_F[0], 20, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[34][w]));

     clSetKernelArg(kernel_Section_F[0], 21, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[8][w]));
     clSetKernelArg(kernel_Section_F[0], 22, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[9][w]));
     clSetKernelArg(kernel_Section_F[0], 23, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[10][w]));
     clSetKernelArg(kernel_Section_F[0], 24, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[11][w]));
     clSetKernelArg(kernel_Section_F[0], 25, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[12][w]));
     clSetKernelArg(kernel_Section_F[0], 26, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[13][w]));

     clSetKernelArg(kernel_Section_F[0], 27, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[1][w]));
     clSetKernelArg(kernel_Section_F[0], 28, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[2][w]));

     // Execute event
     if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

     // Execute the kernel
     clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_F[0], 1, null,globalWorkSize, null, 0, null, events[w]);

}
```

Ilustración 219:Configuración de la sección Cálculo de Tasas asociada a los cálculos de la malla implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.6. Módulo de Cálculo de Saturaciones Explícitas

En la siguiente ilustración se presenta el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de Cálculo de Coeficientes del Sistema Explícito de Saturaciones:

```
public void Jaresim_GPU_MR_I_Modulo_Saturacion_Saturaciones(
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_Run_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_Run_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Run Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Transm Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[],
        // Gas PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_mugt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgpt[],
        // Oil PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bopt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsopt[],
        // Water PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwpt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswpt[],
        //Rock Compressibility data
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_prt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_crt[],
        // Matrix Kr and Pc properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_sat[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pcowt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pcgot[]
    )
    {

       // Set the work-item dimensions
    long globalWorkSize[] = new long[3];
    globalWorkSize[0]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx+1;
    globalWorkSize[1]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny+1;
    globalWorkSize[2]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+1;

   for (int w=0;w<numDevices;w++)
   {

    // Asignacion de los Argumentos de la para el calculo de la Saturacion_Saturaciones de la Secciòn Saturaciòn
     clSetKernelArg(kernel_Section_I[3], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
     clSetKernelArg(kernel_Section_I[3], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[36][w]));
     clSetKernelArg(kernel_Section_I[3], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
     clSetKernelArg(kernel_Section_I[3], 3, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));
     clSetKernelArg(kernel_Section_I[3], 4, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[6][w]));

     clSetKernelArg(kernel_Section_I[3], 5, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[14][w]));
     clSetKernelArg(kernel_Section_I[3], 6, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[15][w]));
     clSetKernelArg(kernel_Section_I[3], 7, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[16][w]));
     clSetKernelArg(kernel_Section_I[3], 8, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[17][w]));

     clSetKernelArg(kernel_Section_I[3], 9, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[18][w]));
     clSetKernelArg(kernel_Section_I[3], 10, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[19][w]));
     clSetKernelArg(kernel_Section_I[3], 11, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[20][w]));
     clSetKernelArg(kernel_Section_I[3], 12, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[21][w]));
     clSetKernelArg(kernel_Section_I[3], 13, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[22][w]));
     clSetKernelArg(kernel_Section_I[3], 14, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[23][w]));

     clSetKernelArg(kernel_Section_I[3], 15, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[29][w]));
     clSetKernelArg(kernel_Section_I[3], 16, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[30][w]));
     clSetKernelArg(kernel_Section_I[3], 17, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[31][w]));
     clSetKernelArg(kernel_Section_I[3], 18, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[32][w]));
     clSetKernelArg(kernel_Section_I[3], 19, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[33][w]));
     clSetKernelArg(kernel_Section_I[3], 20, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[34][w]));

     clSetKernelArg(kernel_Section_I[3], 21, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[27][w]));
     clSetKernelArg(kernel_Section_I[3], 22, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[28][w]));

     clSetKernelArg(kernel_Section_I[3], 23, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[8][w]));
     clSetKernelArg(kernel_Section_I[3], 24, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[9][w]));
     clSetKernelArg(kernel_Section_I[3], 25, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[10][w]));
     clSetKernelArg(kernel_Section_I[3], 26, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[11][w]));
     clSetKernelArg(kernel_Section_I[3], 27, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[12][w]));
     clSetKernelArg(kernel_Section_I[3], 28, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[13][w]));

     // Execute event
     if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

     // Execute the kernel
     clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_I[3], 3, null,globalWorkSize,  null, 0, null, events[w]);
}
```

Ilustración 220:Configuración de la sección Cálculo de Coeficientes del Sistema Explicito de Saturaciones implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
// Wait until the work is finished on all command queues
 if (numDevices>1) clWaitForEvents(events.length, events);

// Set the work-item dimensions
long local_Work_Size[] = new long[1];
local_Work_Size[0]=1;

for (int w=0;w<numDevices;w++)
{
 // Asignacion de los Argumentos de la para el calculo de la Saturacion_Saturaciones Part II de la Secciòn Saturaciòn
  clSetKernelArg(kernel_Section_I[5], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
  clSetKernelArg(kernel_Section_I[5], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[36][w]));
  clSetKernelArg(kernel_Section_I[5], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
  clSetKernelArg(kernel_Section_I[5], 3, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));

  clSetKernelArg(kernel_Section_I[5], 4, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[18][w]));
  clSetKernelArg(kernel_Section_I[5], 5, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[21][w]));

  clSetKernelArg(kernel_Section_I[5], 6, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[29][w]));
  clSetKernelArg(kernel_Section_I[5], 7, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[32][w]));

  // Execute event
  if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

  // Execute the kernel
  clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_I[5], 1, null,local_Work_Size, null, 0, null, events[w]);
 }

// Wait until the work is finished on all command queues
 if (numDevices>1) clWaitForEvents(events.length, events);

}
```

Ilustración 221:Continuación- Configuración de la sección Cálculo de Coeficientes del Sistema Explicito de Saturaciones implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.7. Módulo de Cálculo de ajustes automáticos de time step

En la siguiente ilustración se presenta el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de ajuste automático del time step del Sistema Explícito de Saturaciones:

```
public void Jaresim_GPU_MR_I_Modulo_Saturacion_Auto_time_step(
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_Run_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_Run_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Run Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[]
     )
    {

    // Set the work-item dimensions
    long globalWorkSize[] = new long[3];
    globalWorkSize[0]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx+1;
    globalWorkSize[1]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny+1;
    globalWorkSize[2]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+1;

    long localWorkSize[] = new long[2];
    localWorkSize[0]=12;
    localWorkSize[1]=12;

    long local_Work_Size[] = new long[1];
    local_Work_Size[0]=1;

    for (int w=0;w<numDevices;w++)
    {

      // Asignacion de los Argumentos de la para el calculo de Saturacion_Auto_time_step Part 1
      clSetKernelArg(kernel_Section_I[1], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
      clSetKernelArg(kernel_Section_I[1], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[36][w]));
      clSetKernelArg(kernel_Section_I[1], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
      clSetKernelArg(kernel_Section_I[1], 3, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));

      // Execute event
      if (numDevices>1 && events[w]==null)  events[w] = new cl_event();

      // Execute the kernel
      clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_I[1], 1, null,local_Work_Size, null, 0, null, events[w]);
     }

    // Wait until the work is finished on all command queues
    if (numDevices>1)   clWaitForEvents(events.length, events);

    for (int w=0;w<numDevices;w++)
    {
       // Read back the data from to memory object to the initialization buffer
```

Ilustración 222:Configuración de la sección cálculo de ajustes automáticos de time implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
for (int w=0;w<numDevices;w++)
{
   // Read back the data from to memory object to the initialization buffer
   clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[0][w] , CL_TRUE, 0, structSize_jaresim[0]*Jaresim_Input_eLements[0]  , Pointer.to(Jaresim_Input_Buffer[

   // Read the data from the output jaresim data buffer back into the output initialization  section
   Jaresim_Input_Buffer[0][w].rewind();

   Buffers.readFromBuffer(Jaresim_Input_Buffer[0][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);

}


   if(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.dsmc >= Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.dsmax || Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.dpmc >= Datos_Jare
   {
      if((Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.delt) > Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.dtmin && Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.fact2 < 1)
      {

       for (int w=0;w<numDevices;w++)
       {
         // Asignacion de los Argumentos de la para el calculo de Saturacion_Auto_time_step Part III
         clSetKernelArg(kernel_Section_I[6], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));

         // Execute event
         if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

         // Execute the kernel
         clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_I[6], 1, null,local_Work_Size, null, 0, null, events[w]);
       }

       // Wait until the work is finished on all command queues
        if (numDevices>1) clWaitForEvents(events.length, events);

       for (int w=0;w<numDevices;w++)
       {
         // Asignacion de los Argumentos de la para el calculo de Saturacion_Auto_time_step Part II
         clSetKernelArg(kernel_Section_I[4], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
         clSetKernelArg(kernel_Section_I[4], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
         clSetKernelArg(kernel_Section_I[4], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));

         if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

         // Execute the kernel
         clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_I[4], 3, null,globalWorkSize, null, 0, null, events[w]);
       }

       // Wait until the work is finished on all command queues
        if (numDevices>1) clWaitForEvents(events.length, events);

       for (int w=0;w<numDevices;w++)
       {
         // Asignacion de los Argumentos de la para el calculo de Saturacion_Auto_time_step Part II
         clSetKernelArg(kernel_Section_I[4], 0, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
         clSetKernelArg(kernel_Section_I[4], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
         clSetKernelArg(kernel_Section_I[4], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));

         if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

         // Execute the kernel
         clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_I[4], 3, null,globalWorkSize, null, 0, null, events[w]);
       }

       // Wait until the work is finished on all command queues
       if (numDevices>1)  clWaitForEvents(events.length, events);

       for (int w=0;w<numDevices;w++)
       {                                                                                                                                                  :in
         // Read back the data from to memory object to the initialization buffer
         clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[0][w] , CL_TRUE, 0, structSize_jaresim[0]*Jaresim_Input_eLements[0]  , Pointer.to(Jaresim

         // Read the data from the output jaresim data buffer back into the output initialization  section
         Jaresim_Input_Buffer[0][w].rewind();

        Buffers.readFromBuffer(Jaresim_Input_Buffer[0][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);

       }

      }
    }

}
```

Ilustración 223: Continuación-Configuración de la sección cálculo de ajustes automáticos de time implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.8. Módulo de Cálculo de Presiones Implícitas

En la siguiente ilustración se presenta el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de Cálculo del Sistema Implícito de Presiones:

```
public void Jaresim_GPU_MR_O_LSOR_General(int n,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Run Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Transm Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[],

        double Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_beta[],
        double Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_gamma[],
        double Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_w[],

        double  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_um[],
        double  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_azl[],
        double  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_bzl[],
        double  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_czl[],
        double  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dzl[],
        double  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_uzl[],
        double  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dm[]

        )
{
    // '====================================================================
    // 'This subroutine solves a linear system of three-dimensional finite-
    // 'difference equations.
    // '     ( Linear Successive OverRelaxation method )
    // '====================================================================

     for (int w=0;w<numDevices;w++)
     {
        // Read back the data from to memory object to the initialization buffer
       clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[0][w] , CL_TRUE, 0, structSize_jaresim[0]*Jaresim_Input_elements[0]  , Pointer.to(Jaresim_Inpu
       clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[5][w] , CL_TRUE, 0, structSize_jaresim[5]*Jaresim_Input_elements[5]  , Pointer.to(Jaresim_Inpu
       clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[6][w] , CL_TRUE, 0, structSize_jaresim[6]*Jaresim_Input_elements[6]  , Pointer.to(Jaresim_Inpu
       clEnqueueReadBuffer(commandQueue[w], input_Mems_GPU[7][w] , CL_TRUE, 0, structSize_jaresim[7]*Jaresim_Input_elements[7]  , Pointer.to(Jaresim_Inpu

       // Read the data from the output jaresim data buffer back into the output actualization section
       Jaresim_Input_Buffer[0][w].rewind();
       Jaresim_Input_Buffer[5][w].rewind();
       Jaresim_Input_Buffer[6][w].rewind();
       Jaresim_Input_Buffer[7][w].rewind();

       Buffers.readFromBuffer(Jaresim_Input_Buffer[0][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
       Buffers.readFromBuffer(Jaresim_Input_Buffer[5][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D);
       Buffers.readFromBuffer(Jaresim_Input_Buffer[6][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var);
       Buffers.readFromBuffer(Jaresim_Input_Buffer[7][w],Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run);

     }
    int gen_index=0;

    double rho1=0;
    double theta = 0;
    double tw=0;
    double dmax0=0;
    double theta0=0;
    int jp=0;
    int jm=0;
    int km=0;
    int kp=0;

    int gen_jm_index=0;
    int gen_jp_index=0;
    int gen_km_index=0;
    int gen_kp_index=0;

    double gslsor=0;
    double arg=0;
    double Delta=0;
    double om=0;

    double p_jm=0;
    double p_jp=0;
    double p_km=0;
    double p_kp=0;

    Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.niter = 0;
    Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ddmax = 1;

   while (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ddmax > Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.tol && Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.niter < Da
   {

       tw = 1 - Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.omega;
       dmax0 = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ddmax;
       theta0 = theta;

       if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.niter >= Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.miter)
       {
           System.out.println("jaresim report: max iteration number exceed!!!!");
           System.out.println("niter:"+Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.niter);
           System.out.println("miter:"+Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.miter);
           System.out.println("tol:"+String.format("%.12f",Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.tol));
           System.out.println("ddmax:"+String.format("%.12f",Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ddmax));
       }
```

Ilustración 224:Configuración de la sección Cálculo del Sistema Implícito de Presiones implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.niter = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.niter + 1;
Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ddmax = 0;

for (int k=1;k<=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz;k++)
{
    for (int j=1;j<=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny;j++)
    {

        for (int i=1;i<=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx;i++)
        {

              gslsor=0;
              arg=0;
              Delta=0;
              p_jm=0;
              p_jp=0;
              p_km=0;
              p_kp=0;
              gen_jm_index=0;
              gen_jp_index=0;
              gen_km_index=0;
              gen_kp_index=0;

              gen_index=(k)+(j-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+(i-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny*Datos_Jaresim_GPU_MR
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_um[gen_index] = 0;
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_uzl[gen_index] = 0;
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_azl[gen_index] = 0;
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_bzl[gen_index] = 0;
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_czl[gen_index] = 0;
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dzl[gen_index] = 0;

            if(Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[gen_index].mp!= 0)
            {

              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_um[gen_index] = Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_index].p;
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_uzl[gen_index] =Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_azl[gen_index] = Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[gen_inde
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_bzl[gen_index] = Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[gen_inde
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_czl[gen_index] = Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[gen_inde
              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dzl[gen_index] = Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[gen_inde

               if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny!= 1 )
               {
                  jm = j - 1;
                  jp = j + 1;
                  if (j == 1 ) jm = 1;
                  if (j == Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny) jp = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny;


               if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny!= 1 )
               {
                  jm = j - 1;
                  jp = j + 1;
                  if (j == 1 ) jm = 1;
                  if (j == Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny) jp = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny;

                  gen_jm_index=(k)+(jm-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+(i-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny*Datos_Jaresi
                  gen_jp_index=(k)+(jp-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+(i-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny*Datos_Jaresi

                  if (jm<1)
                  {
                    p_jm=0;
                  }
                  else
                  {
                    p_jm=Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_jm_index].p;
                  }

                  if (jp>Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny)
                  {
                    p_jp=0;
                  }
                   else
                  {
                    p_jp=Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_jp_index].p;
                  }

                  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dzl[gen_index] = Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D
               }

               if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz!=1)
               {
                  km = k - 1;
                  kp = k + 1;
                  if (k == 1) km = 1;
                  if (k == Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz) kp = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz;

                  gen_km_index=(km)+(j-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+(i-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny*Datos_Jaresi
                  gen_kp_index=(kp)+(j-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+(i-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny*Datos_Jaresi


                  if(km<1)
                  {
                    p_km=0;
                  }
```

Ilustración 225: Continuación-Configuración de la sección Cálculo del Sistema Implícito de Presiones implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
                        else
                        {
                         p_km=Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_km_index].p;
                        }


                        if(kp>Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz)
                        {
                          p_kp=0;
                        }
                        else
                        {
                          p_kp=Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_kp_index].p;
                        }

                        Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dzl[gen_index] = Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D
                    }


                }
            }

              Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_uzl=Jaresim_GPU_MR_O_Solver_Linear_Systems_module(Datos_Jaresim_GPU_MR_Pro
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz,
                      j,
                      k,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.miter,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.tol,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_azl,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_bzl,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_czl,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dzl,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_uzl,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_beta,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_gamma,
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_w
                      );

              for (int i = 1 ;i<=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx;i++)
              {

                  gen_index=(k)+(j-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+(i-1)*Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny*Datos_Jaresim_GPU


                  if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[gen_index].mp==0)
                  {
                        gslsor=Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_um[gen_index];
                  }
                  else
                  {
                        gslsor = Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_uzl[gen_index];
                  }


                  if(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.omega!=0)
                  {
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_index].p = tw * Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_um[g
                  }
                  else
                  {
                      Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_index].p = gslsor;
                  }

                  arg = Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[gen_index].p - Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_um[gen_
                  Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dm[gen_index] = Math.abs(arg);

                  if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_Equation_System3D_Var_dm[gen_index] > Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ddmax ) Dat

               }

          }
       }

       if(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.tol1 != 0)
       {
           theta = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ddmax / dmax0;
           Delta = theta - theta0;
           arg = Math.abs(Delta);

           if(arg < Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.tol1)
           {
               om = Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.omega - 1;
               rho1 = (double) ((theta + om) * (theta + om) / (theta *Math.pow(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.omega,2) ));
               if (rho1 < 1)
               {
                   arg = 1 - rho1;
                   Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.omega = (double) (2 / (1 + Math.pow(arg,0.5)));
               }
           }
       }

      }

      for (int w=0;w<numDevices;w++)
      {

        // Actualizacion del Buffer de los Argumentos
        Jaresim_Input_Buffer[0][w]= Buffers.allocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
        Jaresim_Input_Buffer[7][w] = Buffers.allocateBuffer(Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run);

        Buffers.writeToBuffer(Jaresim_Input_Buffer[0][w] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_var);
        Buffers.writeToBuffer(Jaresim_Input_Buffer[7][w] ,Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run);

        if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.eti==0) input_Mems_GPU[0][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR | CL_MEM
        if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.eti==0) input_Mems_GPU[7][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | CL_MEM_COPY_HOST_PTR | CL_MEM
        if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.eti>=0)  input_Mems_GPU[0][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | /*CL_MEM_COPY_HOST_PTR | CL_
        if (Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.eti>=0)  input_Mems_GPU[7][w] = clCreateBuffer(context, CL_MEM_READ_WRITE | /*CL_MEM_COPY_HOST_PTR | CL_

      }

  }
```

Ilustración 226: Continuación-Configuración de la sección Cálculo del Sistema Implícito de Presiones implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.9. Módulo de Solución de Sistemas Lineales

En la siguiente ilustración puede ser evidenciado el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de resolución de sistemas Lineales:

```
public double [] Jaresim_GPU_MR_O_Solver_Linear_Systems_module(int solver_method,int n,int ny,int nz,int j_index, int k_index, int iter_max,double tol_so
        double beta[],
        double gamma[],
        double w[]
        )
{

    double xo[]=null;

    if (solver_method==0)      // Thomas Tridiagonal Solver
    {
        xo=Jaresim_GPU_MR_O_thomas_Alg(n,ny,nz,j_index,k_index,  a, bi, c, d, x , beta,gamma, w);
    }
    else if ( solver_method==1)  // Gauss-Seidel Solver
    {
        xo=Jaresim_GPU_MR_O_Gauss_Seidel(n,ny,nz,j_index,k_index,iter_max,tol_sol, a, bi, c, d, x);
    }
    else if (solver_method==2) // Jacobi Solver
    {
        xo=Jaresim_GPU_MR_O_Jacobi(n,ny,nz,j_index,k_index,iter_max,tol_sol, a, bi, c, d, x);
    }
    else if (solver_method==3) // Conjugate Gradient Solver
    {
        xo=Jaresim_GPU_MR_O_Conjugate_Gradient(n,ny,nz,j_index,k_index,iter_max,tol_sol, a, bi, c, d, x);
    }

    return xo;

 }
```

Ilustración 227:Configuración de la sección solución de sistemas lineales implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

A continuación, se muestra los códigos JAVA de los solvers de sistemas lineales mediante los métodos Thomas, Gauss-Seidel, Jacobi y Gradiente Conjugado:

```
public double [] Jaresim_GPU_MR_O_thomas_Alg(int n,int ny,int nz,int j_index, int k_index, double a[], double bi[], double c[], double d[], double x[],
        double beta[],
        double gamma[],
        double w[]

        )
{
    //*********************************************************************
    //     This program solves the tridiagonal system generated by the
    //     system of n equations by thomas algoritm:
    //
    //      a(i) * u(i - 1) + bi(i) * u(i) + c(i) * u(i + 1) = d(i)
    //
    //*********************************************************************
    int nm=0;
    int ip=0;
    int ima=0;
    int jk_index=0;
    int jk_n_index=0;
    int ima_jk_index=0;
    int ip_jk_index=0;

    ip=1;
    ip_jk_index=(k_index)+(j_index-1)*nz+(ip-1)*ny*nz;

    beta[ip_jk_index] = bi[ip_jk_index] ;

    if(beta[ip_jk_index]==0)
    {
        gamma[ip_jk_index]  = d[ip_jk_index]  / (bi[ip_jk_index]+0.0000000001) ;
    }
    else
    {
       gamma[ip_jk_index]  = d[ip_jk_index]  / bi[ip_jk_index] ;
    }

    nm = n - 1;

    //'-----------------------------------------------Compute forward solution

    for(int i=1;i<=nm;i++)
    {

        jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;


       if(beta[ip_jk_index]==0)
       {
          w[jk_index] = c[jk_index] / (beta[jk_index]+0.0000000001);
       }
       else
       {
           w[jk_index] = c[jk_index] / beta[jk_index];;
       }

       ip = i + 1;
       ip_jk_index=(k_index)+(j_index-1)*nz+(ip-1)*ny*nz;
       beta[ip_jk_index] = bi[ip_jk_index] - a[ip_jk_index] * w[jk_index];
    }

   ima=0;

   for(int i=2;i<=n;i++)
   {
       ima = i - 1;
       jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
       ima_jk_index=(k_index)+(j_index-1)*nz+(ima-1)*ny*nz;


       if(beta[jk_index]==0)
       {
          gamma[jk_index] = (d[jk_index] - a[jk_index] * gamma[ima_jk_index]) / (beta[jk_index]+0.0000000001);
       }
       else
       {
           gamma[jk_index] = (d[jk_index] - a[jk_index] * gamma[ima_jk_index]) / beta[jk_index];
       }


   }

    //'-----------------------------------------------Compute back solution


    jk_n_index=(k_index)+(j_index-1)*nz+(n-1)*ny*nz;
    x[jk_n_index] = gamma[jk_n_index];
    ima=0;

    for (int j=1;j<=nm;j++)
    {
        ima = n - j;
        ip = ima + 1;

        ima_jk_index=(k_index)+(j_index-1)*nz+(ima-1)*ny*nz;
        ip_jk_index= (k_index)+(j_index-1)*nz+(ip-1)*ny*nz;

        x[ima_jk_index] = gamma[ima_jk_index] - w[ima_jk_index] * x[ip_jk_index];
    }

    //'--------------------------------------------------------------------

    return x;

}
```

Ilustración 228: Continuación-Configuración de la sección solución de sistemas lineales implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
public double [] Jaresim_GPU_MR_O_Gauss_Seidel(int n,int ny,int nz,int j_index,int k_index,int iter_max,double tol_sol, double a[], double bi[], double c
{
    //**********************************************************************
    //     This program solves the tridiagonal system generated by the
    //     system of n equations by gauss seidel method:
    //
    //     a(i) * u(i - 1) + bi(i) * u(i) + c(i) * u(i + 1) = d(i)
    //
    //**********************************************************************

    int jk_index=0;
    //**********************a[i][j] matrix derivation*************************

    for(int i=1;i<=n;i++)
    {

        jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
        xold[i]=x[jk_index];

        for(int j=1;j<=n;j++)
        {
            ai[i][j]=0;
        }

        ai[i][i]=bi[jk_index];
        if (i>1) ai[i][i-1]= a[jk_index];
        if (i<n) ai[i][i+1]= c[jk_index];
    }

    double sum=0;

    for(int i=1;i<=n;i++)
    {
        sum=ai[i][i];
        jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;

        for(int j=1;j<=n;j++)
        {
            ai[i][j]=ai[i][j]/sum;
        }

        d[jk_index]=d[jk_index]/sum;

    }

    int iter=0;
    double error=1;
    double error=1;

    while (iter<iter_max && error>tol_sol)
    {

      sum=0;

      for(int i=1;i<=n;i++)
      {

          jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          sum=d[jk_index];

          for(int j=1;j<=n;j++)
          {
             int j_k_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
             if(i!=j) sum+=-ai[i][j]*x[j_k_index];
          }

          x[jk_index]=sum;

      }

      jk_index=(k_index)+(j_index-1)*nz+(1-1)*ny*nz;
      error=Math.abs(x[jk_index]-xold[1]);

      for(int i=2;i<=n;i++)
      {
          jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          if (error<Math.abs(x[jk_index]-xold[i])) error=Math.abs(x[jk_index]-xold[i]);
      }

      iter+=1;

      for(int i=1;i<=n;i++)
      {
          jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          xold[i]=x[jk_index];
      }

    }

    //'-------------------------------------------------------------------------

    return x;

}
```

Ilustración 229: Continuación-Configuración de la sección solución de sistemas lineales implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
public double [] Jaresim_GPU_MR_O_Jacobi(int n,int ny,int nz,int j_index,int k_index,int iter_max,double tol_sol, double a[], double bi[], double c[],
{
    //*********************************************************************
    //     This program solves the tridiagonal system generated by the
    //     system of n equations by Jacobi method:
    //
    //     a(i) * u(i - 1) + bi(i) * u(i) + c(i) * u(i + 1) = d(i)
    //
    //*********************************************************************
    int jk_index=0;
    //**********************a[i][j] matrix derivation**********************

    for(int i=1;i<=n;i++)
    {
        jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
        xold[i]=x[jk_index];

        for(int j=1;j<=n;j++)
        {
            ai[i][j]=0;
        }

        ai[i][i]=bi[jk_index];
        if (i>1) ai[i][i-1]= a[jk_index];
        if (i<n) ai[i][i+1]= c[jk_index];
    }

    double sum=0;

    for(int i=1;i<=n;i++)
    {
        sum=ai[i][i];
        jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;

        for(int j=1;j<=n;j++)
        {
            ai[i][j]=ai[i][j]/sum;
        }

        d[jk_index]=d[jk_index]/sum;

    }

    int iter=0;
    double error=1;


    while (iter<iter_max && error>tol_sol)
    {

      sum=0;

      for(int i=1;i<=n;i++)
      {
          jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          sum=d[jk_index];

          for(int j=1;j<=n;j++)
          {
            if(i!=j) sum+=-ai[i][j]*xold[j];
          }

          x[jk_index]=sum;

      }

      jk_index=(k_index)+(j_index-1)*nz+(1-1)*ny*nz;
      error=Math.abs(x[jk_index]-xold[1]);

      for(int i=2;i<=n;i++)
      {
          jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          if (error<Math.abs(x[jk_index]-xold[i])) error=Math.abs(x[jk_index]-xold[i]);
      }

      iter+=1;

      for(int i=1;i<=n;i++)
      {
          jk_index=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          xold[i]=x[jk_index];
      }

    }

    //'-------------------------------------------------------------------------

    return x;

}
```

Ilustración 230: Continuación-Configuración de la sección solución de sistemas lineales implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

```
public double [] Jaresim_GPU_MR_O_Conjugate_Gradient(int n,int ny,int nz,int j_index,int k_index,int iter_max,double tol_sol, double a[], double bi[], do
{
    //*********************************************************************
    //     This code solves the tridiagonal system generated by the
    //     system of n equations by Conjugate Gradiente method:
    //
    //      a[i] * x[i - 1] + bi(i) * x[i] + c[i] * x[i + 1] = d[i]
    //
    //*********************************************************************

    int i_ind=0;
    int j_ind=0;
    //**************************a[i][j] matrix derivation*******************

    for(int i=1;i<=n;i++)
    {
        i_ind=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;

        for(int j=1;j<=n;j++)
        {
            ai[i][j]=0;
        }

        ai[i][i]=bi[i_ind];
        if (i>1) ai[i][i-1]= a[i_ind];
        if (i<n) ai[i][i+1]= c[i_ind];
    }

    for(int i=1;i<=n;i++)
    {

        i_ind=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
        this.w[i]=0;

        for(int j=1;j<=n;j++)
        {
            j_ind=(k_index)+(j_index-1)*nz+(j-1)*ny*nz;
            this.w[i]+=this.ai[i][j]*x[j_ind];
        }

        this.r[i]=d[i_ind]-this.w[i];
        this.p[i]=this.r[i];

        if (this.r[i]==0)
        {
            this.r[i]=0;
            this.p[i]=0;
         }

    }

    double error=0;

    for(int i=1;i<=n;i++)
    {

        if (error<Math.abs(this.r[i])) error=Math.abs(this.r[i]);
    }


    int iter=0;

    while (iter<iter_max && error>tol_sol)
    {

      for(int i=1;i<=n;i++)
      {

          i_ind=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          this.w[i]=0;

          for(int j=1;j<=n;j++)
          {
              this.w[i]+=this.ai[i][j]*this.p[j];
          }

          if (this.w[i]*this.p[i]!=0)
          {
              this.alpha[i]=(this.r[i]*this.r[i])/(this.w[i]*this.p[i]);
          }
          else
          {
              this.alpha[i]=0;
          }

          this.xn[i]=x[i_ind]+this.alpha[i]*this.p[i];
          this.rn[i]=this.r[i]-this.alpha[i]*this.w[i];


          if(this.r[i]*this.r[i]>0)
          {
              this.beta[i]=(this.rn[i]*this.rn[i])/(this.r[i]*this.r[i]);
          }
          else
          {
              this.beta[i]=0;
          }

          this.pn[i]=this.rn[i]+this.beta[i]*this.p[i];

         if (i==1) error=Math.abs(this.rn[i]);
         if (error<Math.abs(this.rn[i])) error=Math.abs(this.rn[i]);

      }

      for(int i=1;i<=n;i++)
      {
          i_ind=(k_index)+(j_index-1)*nz+(i-1)*ny*nz;
          this.p[i]=this.pn[i];
          x[i_ind]=this.xn[i];
          this.r[i]=this.rn[i];
      }

      iter+=1;

    }

    //*********************************************************************

    return x;

}
```

Ilustración 231: Continuación-Configuración de la sección solución de sistemas lineales implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.7.10. Módulo de Cálculo de Coeficientes de Presiones Implícitas

En la siguiente ilustración puede ser evidenciado el código JAVA para la llamada del Archivo *.CL que contiene el código paralelizado del Módulo de Cálculo de Coeficientes del Sistema Implícito de Presiones:

```
public void Jaresim_GPU_MR_G_Coefficientes( int n,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_Run_var Datos_Jaresim_GPU_MR_Proyecto_JOCL_Run_var,
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Run Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Run[],
        Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_3D_Transm Datos_Jaresim_GPU_MR_Proyecto_JOCL_3D_Trans_var[],
        // Gas PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_mugt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bgpt[],
        // Oil PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bopt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rsopt[],
        // Water PVT Properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_muwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_bwpt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_rswpt[],
        //Rock Compressibility data
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_prt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_crt[],
        // Matrix Kr and Pc properties
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_sat[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krot[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krwt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_krgt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pcowt[],
        double Jaresim_Estructura_Datos_Secciones_GPU_from_CPU_1D_pcgot[]
    )
    {

    // Set the work-item dimensions
    long globalWorkSize[] = new long[3];
    globalWorkSize[0]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nx+1;
    globalWorkSize[1]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.ny+1;
    globalWorkSize[2]=Datos_Jaresim_GPU_MR_Proyecto_JOCL_var.nz+1;

for (int w=0;w<numDevices;w++)
{

    // Asignacion de los Argumentos de la para el calculo de Coeficientes de la Seccion G
     clSetKernelArg(kernel_Section_G[0], 0, Sizeof.cl_int, Pointer.to(new int[]{ n }));
     clSetKernelArg(kernel_Section_G[0], 1, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[0][w]));
     clSetKernelArg(kernel_Section_G[0], 2, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[36][w]));
     clSetKernelArg(kernel_Section_G[0], 3, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[5][w]));
     clSetKernelArg(kernel_Section_G[0], 4, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[7][w]));
     clSetKernelArg(kernel_Section_G[0], 5, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[6][w]));

     clSetKernelArg(kernel_Section_G[0], 6, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[14][w]));
     clSetKernelArg(kernel_Section_G[0], 7, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[15][w]));
     clSetKernelArg(kernel_Section_G[0], 8, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[16][w]));
     clSetKernelArg(kernel_Section_G[0], 9, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[17][w]));

     clSetKernelArg(kernel_Section_G[0], 10, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[18][w]));
     clSetKernelArg(kernel_Section_G[0], 11, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[19][w]));
     clSetKernelArg(kernel_Section_G[0], 12, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[20][w]));
     clSetKernelArg(kernel_Section_G[0], 13, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[21][w]));
     clSetKernelArg(kernel_Section_G[0], 14, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[22][w]));
     clSetKernelArg(kernel_Section_G[0], 15, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[23][w]));

     clSetKernelArg(kernel_Section_G[0], 16, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[29][w]));
     clSetKernelArg(kernel_Section_G[0], 17, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[30][w]));
     clSetKernelArg(kernel_Section_G[0], 18, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[31][w]));
     clSetKernelArg(kernel_Section_G[0], 19, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[32][w]));
     clSetKernelArg(kernel_Section_G[0], 20, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[33][w]));
     clSetKernelArg(kernel_Section_G[0], 21, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[34][w]));

     clSetKernelArg(kernel_Section_G[0], 22, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[27][w]));
     clSetKernelArg(kernel_Section_G[0], 23, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[28][w]));

     clSetKernelArg(kernel_Section_G[0], 24, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[8][w]));
     clSetKernelArg(kernel_Section_G[0], 25, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[9][w]));
     clSetKernelArg(kernel_Section_G[0], 26, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[10][w]));
     clSetKernelArg(kernel_Section_G[0], 27, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[11][w]));
     clSetKernelArg(kernel_Section_G[0], 28, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[12][w]));
     clSetKernelArg(kernel_Section_G[0], 29, Sizeof.cl_mem, Pointer.to(input_Mems_GPU[13][w]));

          // Execute event
          if (numDevices>1 && events[w]==null)   events[w] = new cl_event();

          // Execute the kernel
          clEnqueueNDRangeKernel(commandQueue[w], kernel_Section_G[0], 3, null,globalWorkSize,  null, 0, null, events[w]);
     }

     // Wait until the work is finished on all command queues
     if (numDevices>1)   clWaitForEvents(events.length, events);


     }
```

Ilustración 232: Configuración de la sección Cálculo de Coeficientes del Sistema Implícitos de Presiones implementando el Módulo OpenCL en JAVA. Fuente: Elaboración propia.

10.8. Formatos de importe de archivo admitidos por la aplicación desarrollada en esta investigación y ventanas de correlaciones disponibles

Para la carga de información requerida en el proceso de simulación, la aplicación desarrollada admite una serie de formatos que serán presentados a continuación para cada una de las secciones del simulador, así como las ventanas de las correlaciones disponibles.

10.8.1. Formatos de Importe tipo *.ECLGRD para propiedades estáticas

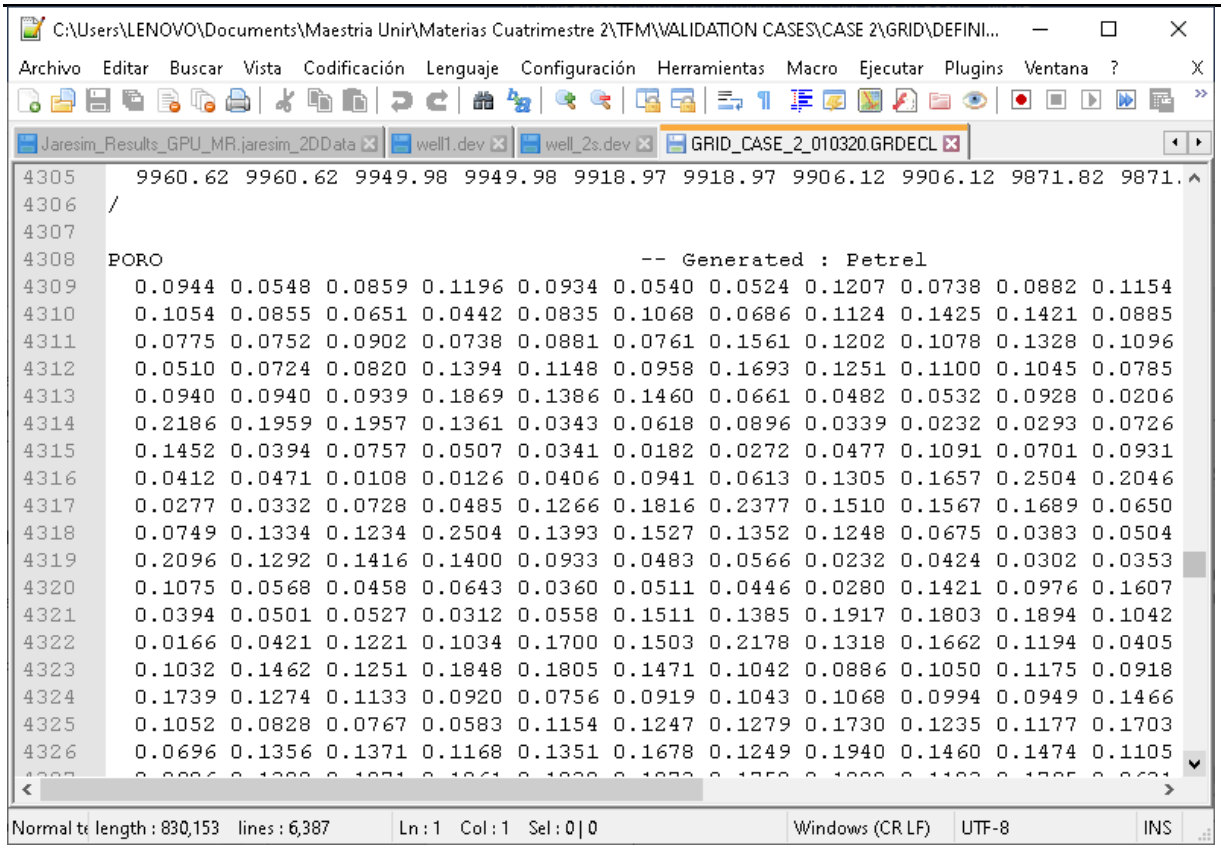


Ilustración 233:Ventana de la Sección Propiedades estáticas del Modelo-Porosidad Efectiva. Fuente: Elaboración propia.

10.8.2. Formatos de Importe tipo *.jaresim_statictable para propiedades estáticas

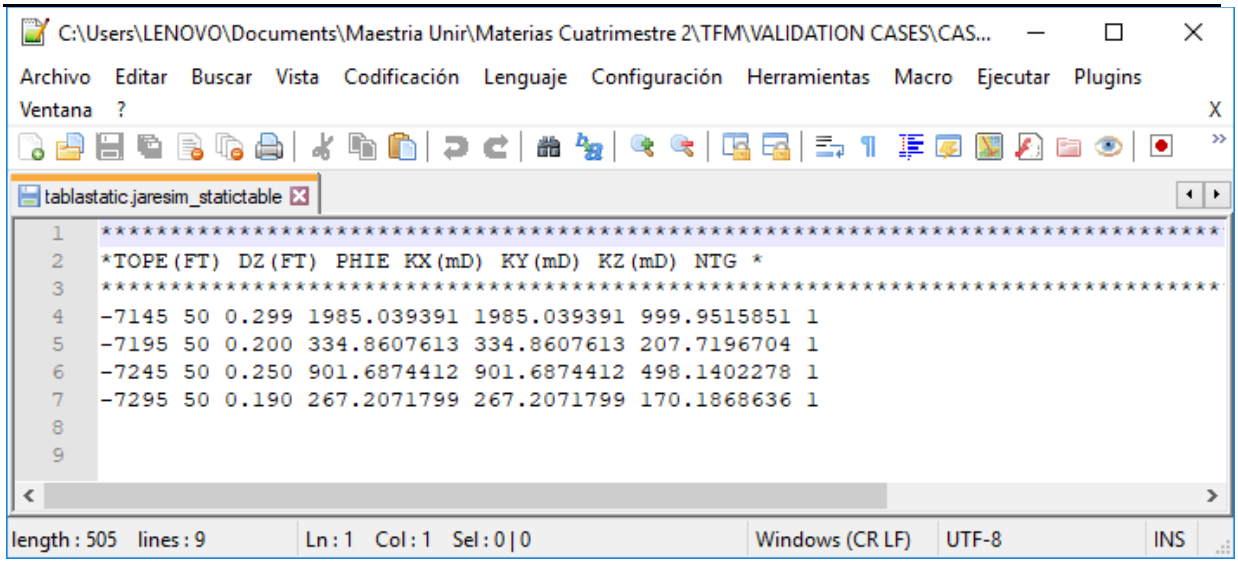


*Ilustración 234: Archivo tipo en formato *.jaresim_statictable. Fuente: Elaboración propia.*

10.8.3. Formatos de Importe tipo *.jaresim_crtable para propiedades estáticas-Compresibilidad

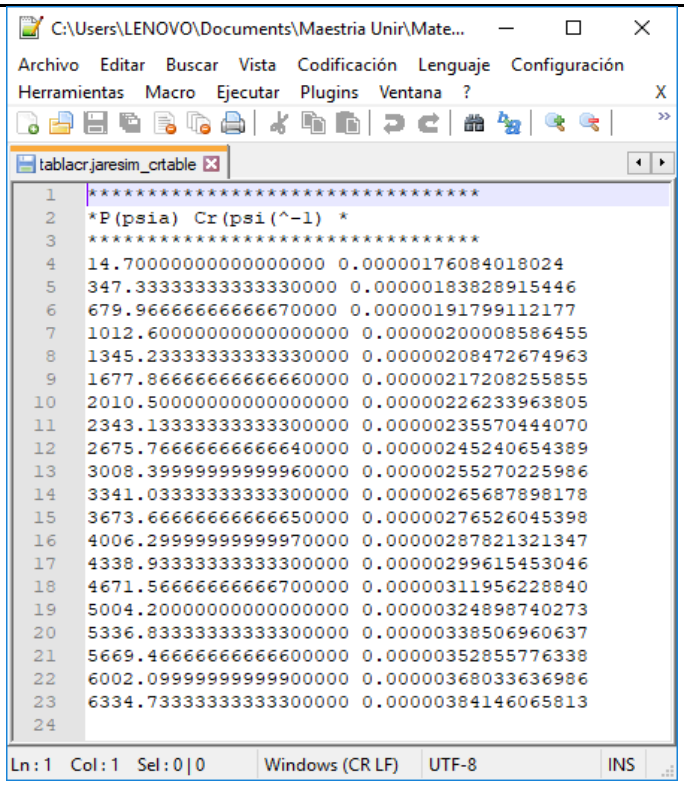
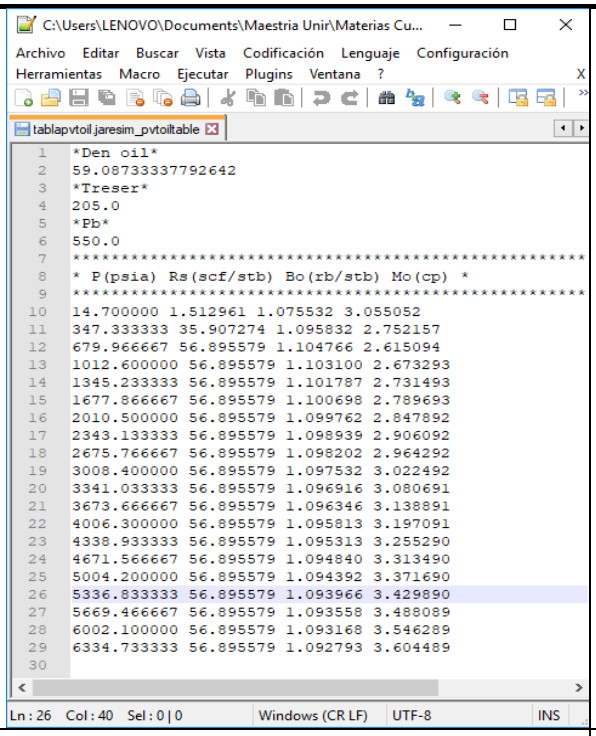


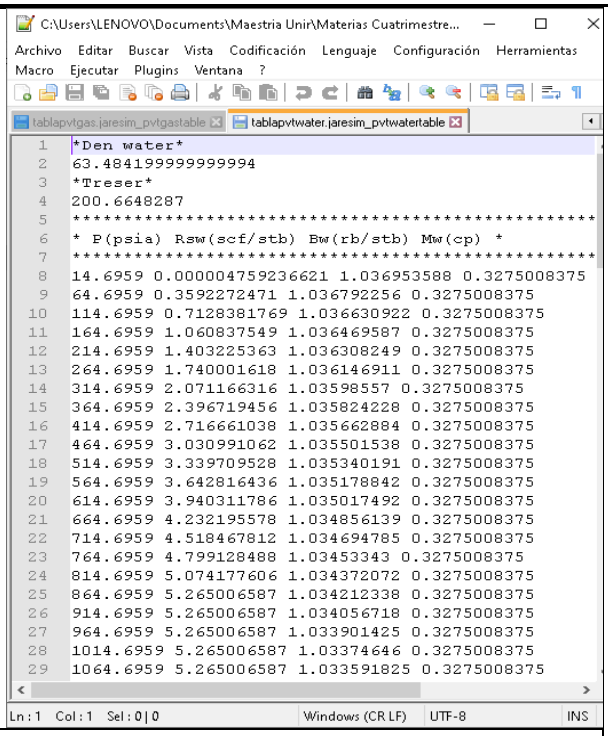
Ilustración 235:Ventana de la Sección Propiedades estáticas del Modelo-Importe Compresibilidad de la roca-Archivo Tipo. Fuente: Elaboración propia.

10.8.4. Formatos de Importe tipo *.jaresim_pvtoiltable, *.jaresim_pvtwatertable y *.jaresim_pvtgastable para propiedades del fluido



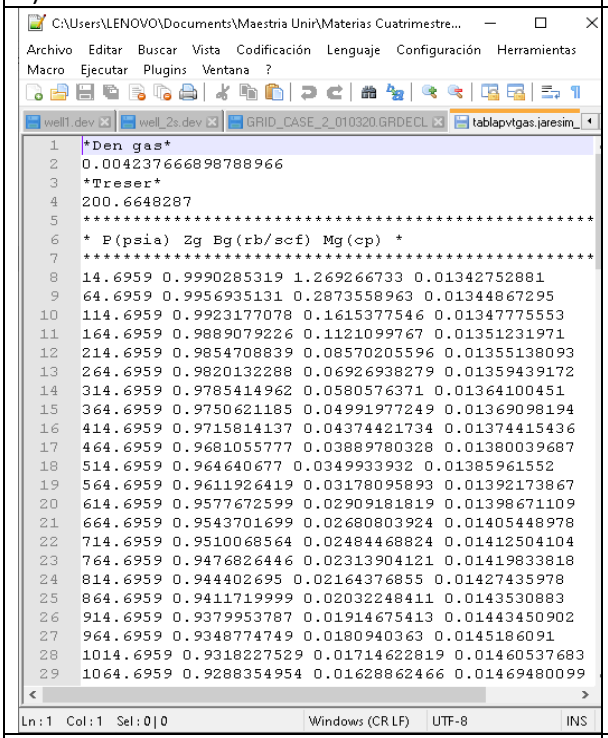
```
*Den oil*
59.08733337792642
*Treser*
205.0
*Pb*
550.0
**************************************************
* P(psia) Rs(scf/stb) Bo(rb/stb) Mo(cp) *
**************************************************
14.700000 1.512961 1.075532 3.055052
347.333333 35.907274 1.095832 2.752157
679.966667 56.895579 1.104766 2.615094
1012.600000 56.895579 1.103100 2.673293
1345.233333 56.895579 1.101787 2.731493
1677.866667 56.895579 1.100698 2.789693
2010.500000 56.895579 1.099762 2.847892
2343.133333 56.895579 1.098939 2.906092
2675.766667 56.895579 1.098202 2.964292
3008.400000 56.895579 1.097532 3.022492
3341.033333 56.895579 1.096916 3.080691
3673.666667 56.895579 1.096346 3.138891
4006.300000 56.895579 1.095813 3.197091
4338.933333 56.895579 1.095313 3.255290
4671.566667 56.895579 1.094840 3.313490
5004.200000 56.895579 1.094392 3.371690
5336.833333 56.895579 1.093966 3.429890
5669.466667 56.895579 1.093558 3.488089
6002.100000 56.895579 1.093168 3.546289
6334.733333 56.895579 1.092793 3.604489
```

a). Datos de la fase Petróleo



```
*Den water*
63.484199999999994
*Treser*
200.6648287
**************************************************
* P(psia) Rsw(scf/stb) Bw(rb/stb) Mw(cp) *
**************************************************
14.6959 0.000004759236621 1.036953588 0.3275008375
64.6959 0.3592272471 1.036792256 0.3275008375
114.6959 0.7128381769 1.036630922 0.3275008375
164.6959 1.060837549 1.036469587 0.3275008375
214.6959 1.403225363 1.036308249 0.3275008375
264.6959 1.740001618 1.036146911 0.3275008375
314.6959 2.071166316 1.03598557 0.3275008375
364.6959 2.396719456 1.035824228 0.3275008375
414.6959 2.716661038 1.035662884 0.3275008375
464.6959 3.030991062 1.035501538 0.3275008375
514.6959 3.339709528 1.035340191 0.3275008375
564.6959 3.642816436 1.035178842 0.3275008375
614.6959 3.940311786 1.035017492 0.3275008375
664.6959 4.232195578 1.034856139 0.3275008375
714.6959 4.518467812 1.034694785 0.3275008375
764.6959 4.799128488 1.03453343 0.3275008375
814.6959 5.074177606 1.034372072 0.3275008375
864.6959 5.265006587 1.034212338 0.3275008375
914.6959 5.265006587 1.034056718 0.3275008375
964.6959 5.265006587 1.033901425 0.3275008375
1014.6959 5.265006587 1.03374646 0.3275008375
1064.6959 5.265006587 1.033591825 0.3275008375
```

b). Datos de la fase Agua

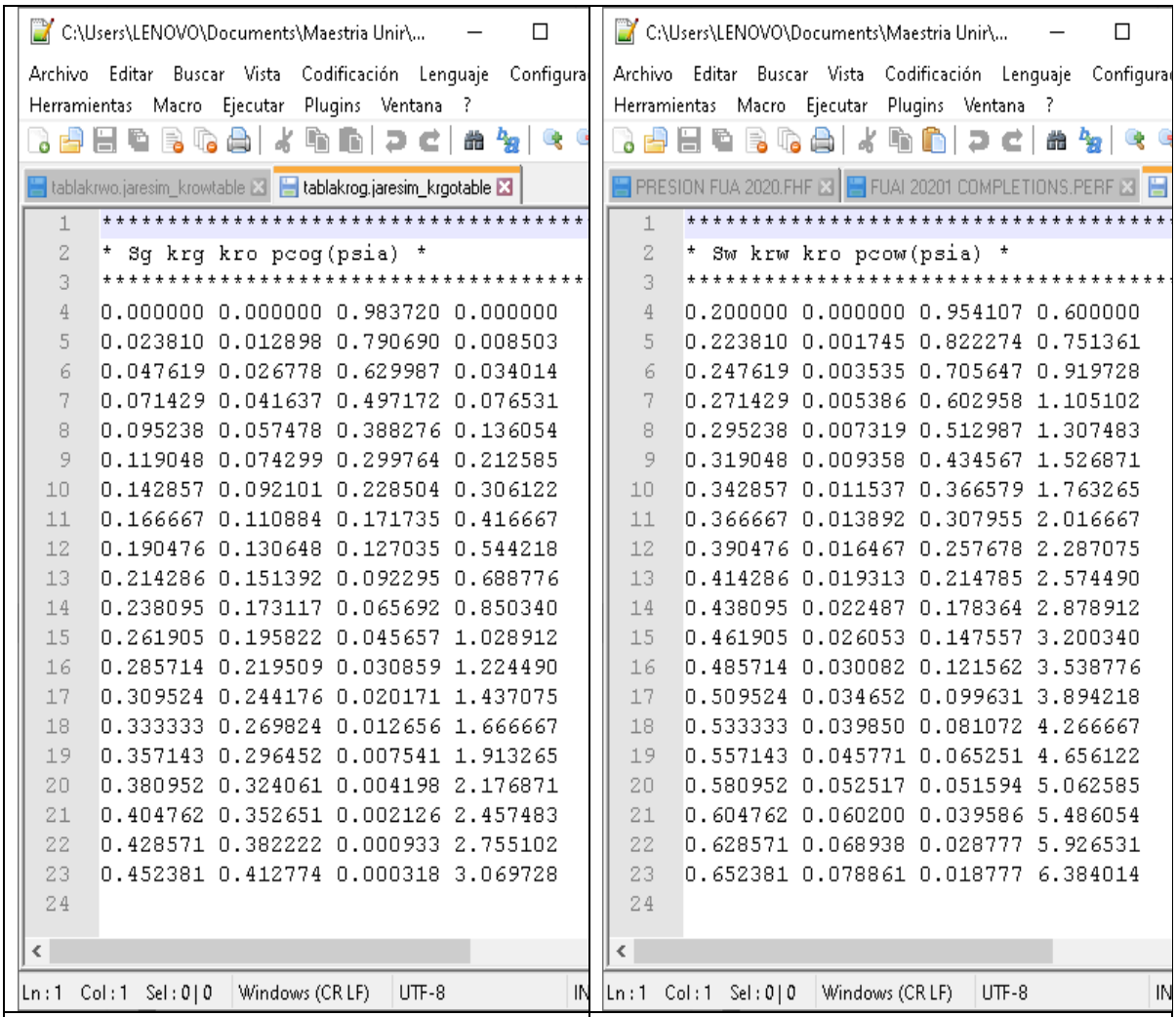


```
*Den gas*
0.004237666898788966
*Treser*
200.6648287
**************************************************
* P(psia) Zg Bg(rb/scf) Mg(cp) *
**************************************************
14.6959 0.9990285319 1.269266733 0.01342752881
64.6959 0.9956935131 0.2873558963 0.01344867295
114.6959 0.9923177078 0.1615377546 0.01347775553
164.6959 0.9889079226 0.1121099767 0.01351231971
214.6959 0.9854708839 0.08570205596 0.01355138093
264.6959 0.9820132288 0.06926938279 0.01359439172
314.6959 0.9785414962 0.0580576371 0.01364100451
364.6959 0.9750621185 0.04991977249 0.01369098194
414.6959 0.9715814137 0.04374421734 0.01374415436
464.6959 0.9681055777 0.03889780328 0.01380039687
514.6959 0.964640677 0.0349933932 0.01385961552
564.6959 0.9611926419 0.03178095893 0.01392173867
614.6959 0.9577672599 0.02909181819 0.01398671109
664.6959 0.9543701699 0.02680803924 0.01405448978
714.6959 0.9510068564 0.02484468824 0.01412504104
764.6959 0.9476826446 0.02313904121 0.01419833818
814.6959 0.944402695 0.02164376855 0.01427435978
864.6959 0.9411719999 0.02032248411 0.0143530883
914.6959 0.9379953787 0.01914675413 0.01443450902
964.6959 0.9348774749 0.0180940363 0.0145186091
1014.6959 0.9318227529 0.01714622819 0.01460537683
1064.6959 0.9288354954 0.01628862466 0.01469480099
```

c). Datos de la fase Gas

Ilustración 236:Ventana de la Sección Propiedades PVT-Importe del Modelo de Fluido. Fuente: Elaboración propia.

10.8.5. Formatos de Importe tipo *.jaresim_krgotable y *.jaresim_krowtable para propiedades roca-fluido



a).Sistema Gas-Petróleo | b).Sistema Agua-Petróleo

Ilustración 237:Ventana de la Sección Propiedades Roca Fluido-Importe del Modelo de Roca-Fluido. Fuente: Elaboración propia.

10.8.6. Formatos de Importe tipo *.jaresim_Properties_3dGrid_RestartProperties para inicialización recurrencia

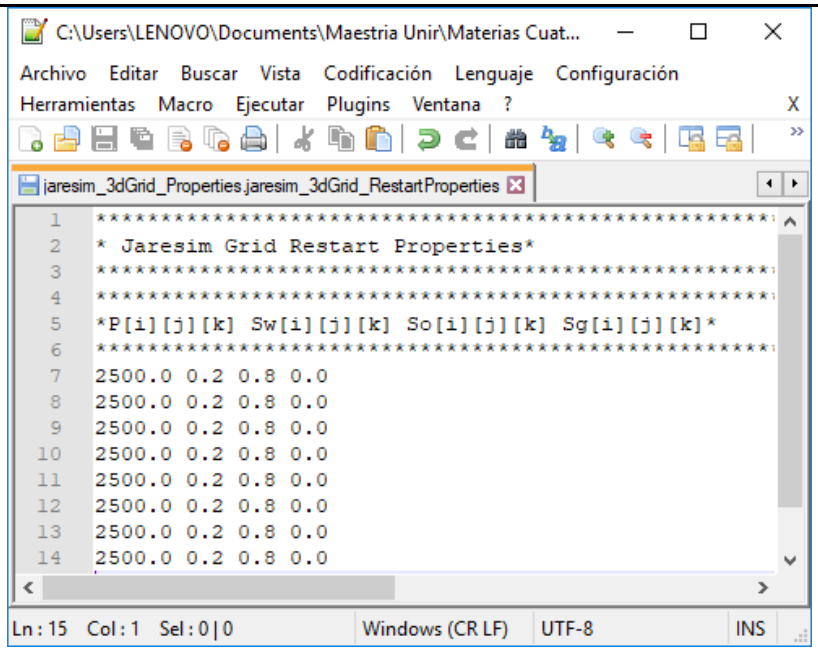


Ilustración 238:Ventana de la Sección Inicialización–Formato archivo carga inicialización recurrencia. Fuente: Elaboración propia.

10.8.7. Formatos de Importe tipo *.jaresim_surveytable para trayectorias

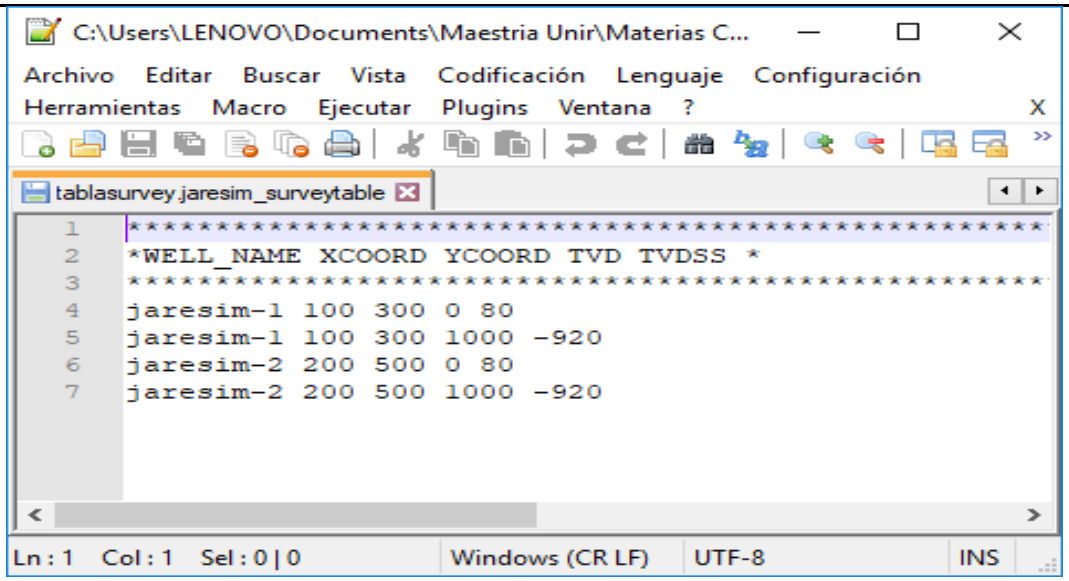


*Ilustración 239:Ventana de la Sección de Trayectoria de Pozos- formato archivo *jaresim_surveytable para el importe de Datos de Trayectorias. Fuente: Elaboración propia.*

10.8.8. Formatos de Importe tipo *.jaresim_producersRatetable para Históricos de Producción

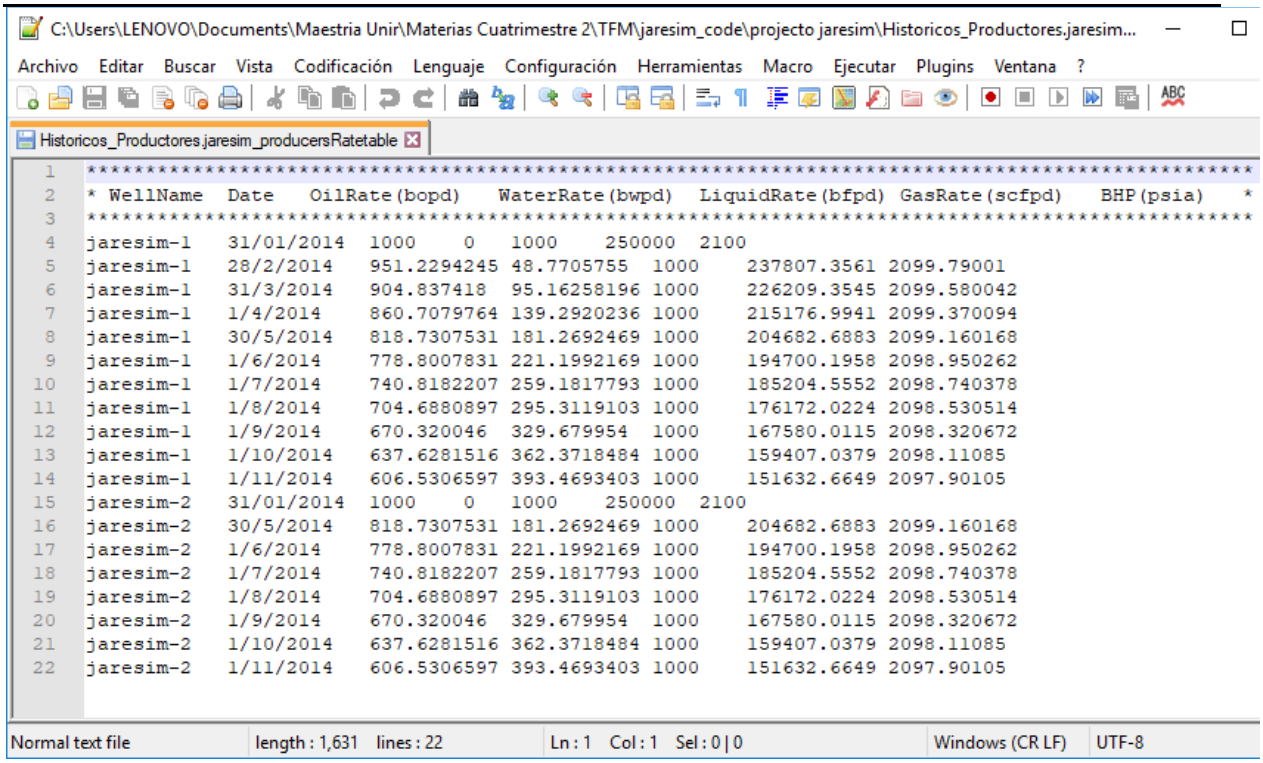


*Ilustración 240:Ventana de la Sección Datos Históricos de pozos-Importe de Datos Históricos-formato *.jaresim_producersRatetable para el importe datos de producción de simulación de pozos. Fuente: Elaboración propia.*

10.8.9. Formatos de Importe tipo *.jaresim_injectorsRatetable para Históricos de Inyección

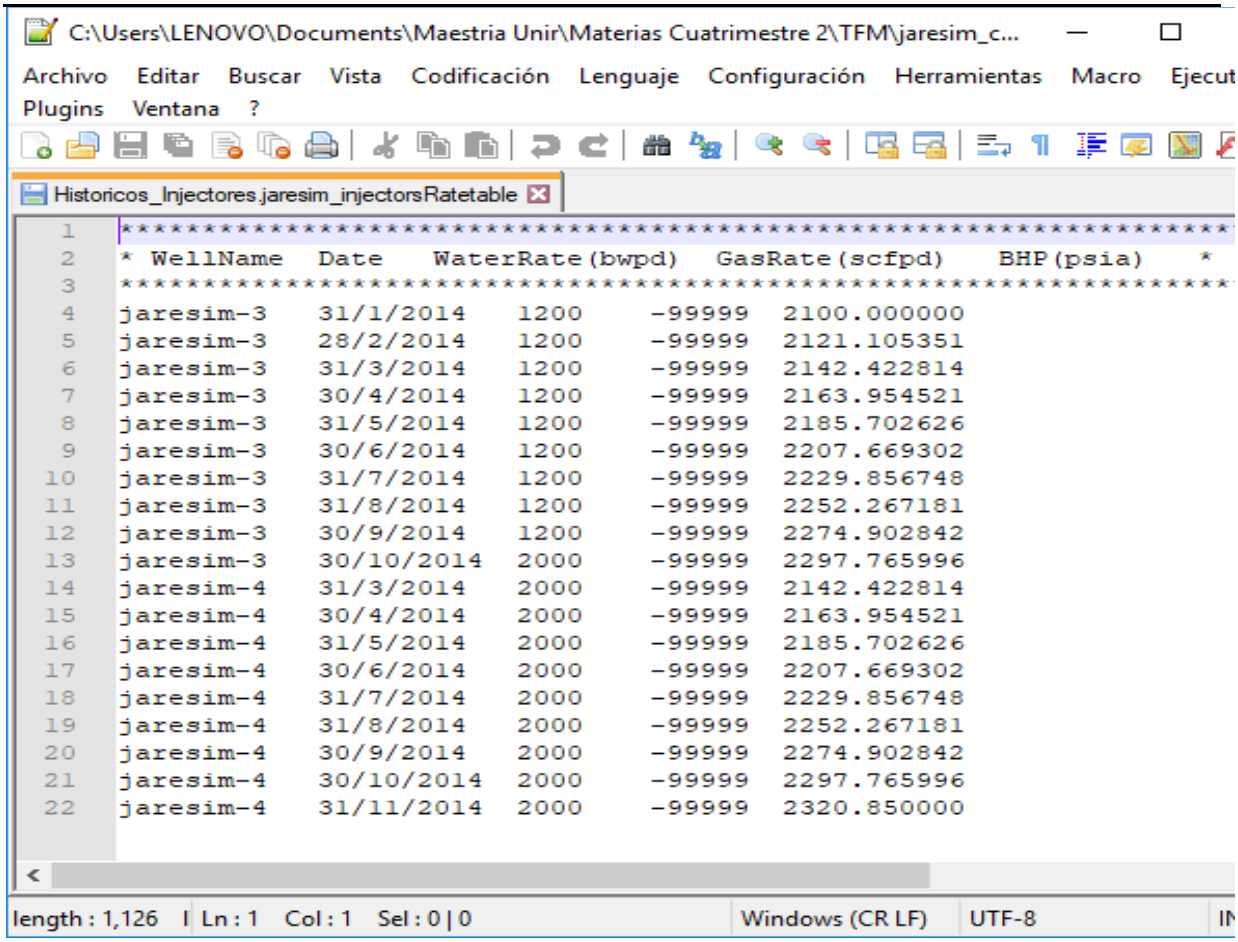


*Ilustración 241:Ventana de la Sección Datos Históricos de pozos-Importe de Datos Históricos-formato *.jaresim_injectorsRatetable para el importe datos de producción de simulación de pozos. Fuente: Elaboración propia.*

10.8.10. Ventana de Correlaciones-Compresibilidad

Estas correlaciones pueden ser evidenciadas en la aplicación como se muestra en las siguientes ilustraciones:

Ilustración 242:Ventana de la Sección Propiedades estáticas del Modelo–Correlaciones de Compresibilidad de la roca. Fuente: Elaboración propia.

10.8.11. Ventana de Correlaciones-Propiedades PVT

Finalmente, si no se dispone de un modelo de fluido, la aplicación dispone de correlaciones numéricas para las fases petróleo, agua y gas, que permite la estimación de las propiedades PVT a través de correcciones. Estas correlaciones pueden ser evidenciadas en la siguiente ilustración:

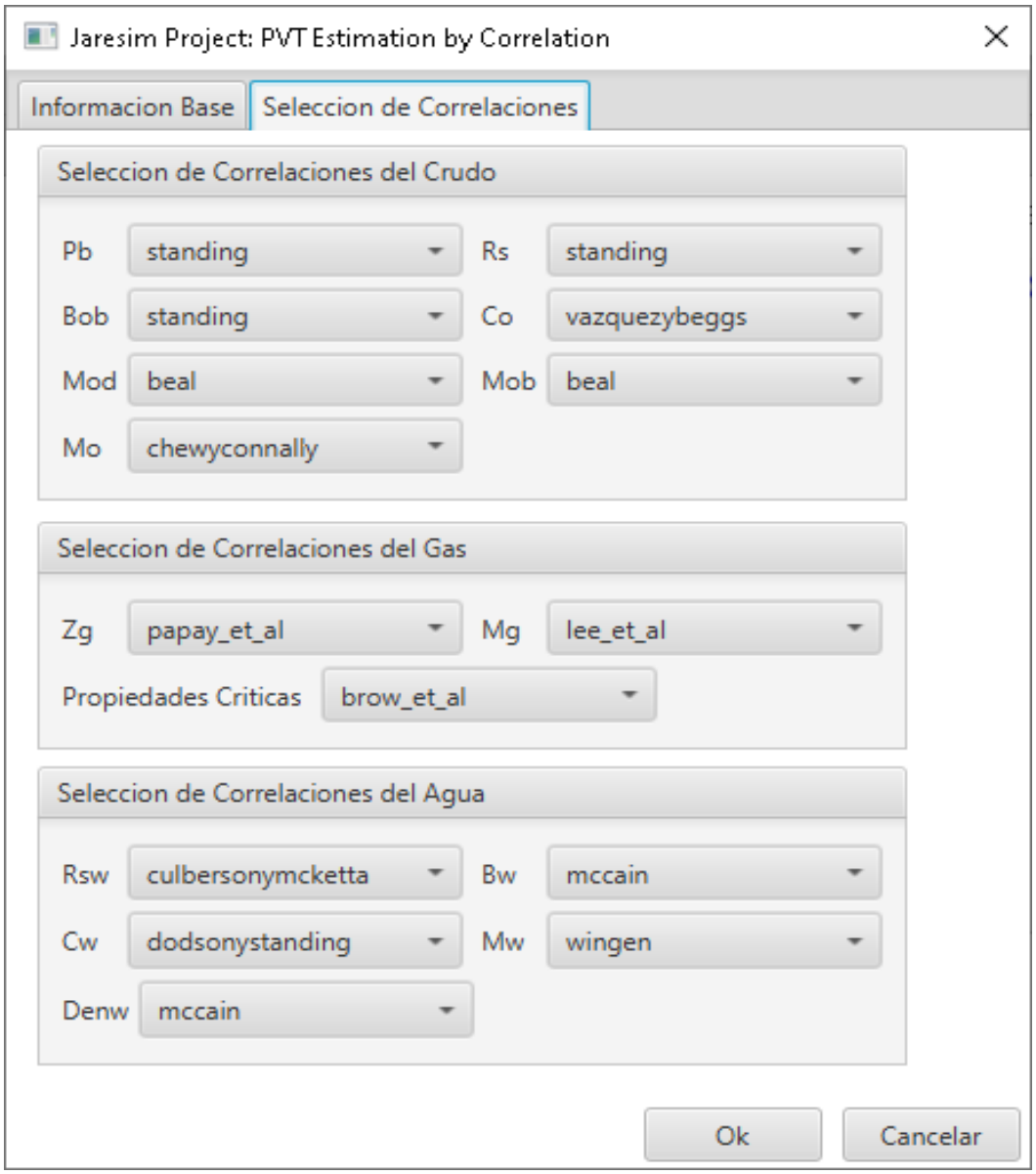
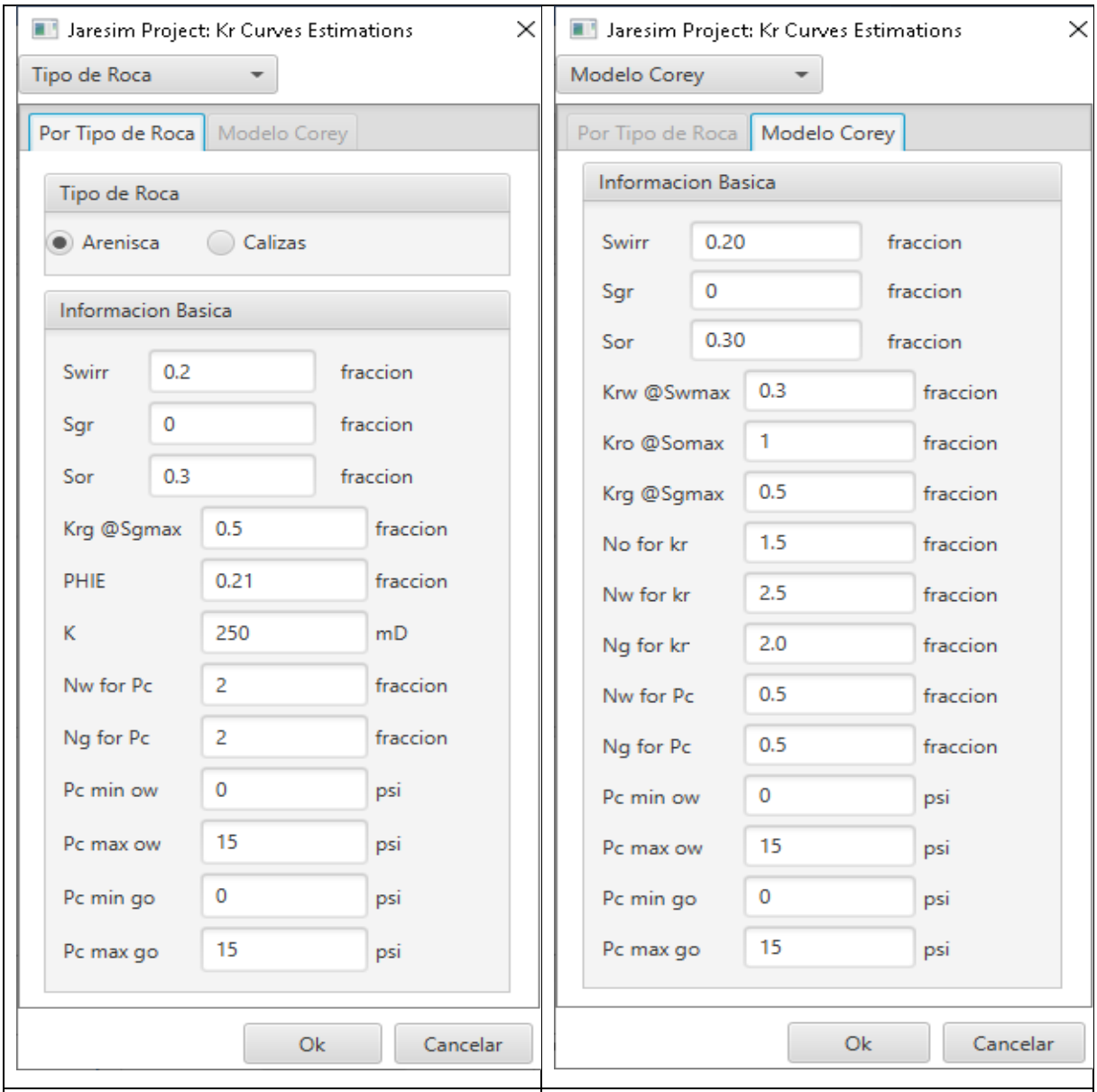


Ilustración 243:Ventana de la Sección Propiedades PVT-Correlaciones Fase Petróleo, Gas y Agua. Fuente: Elaboración propia.

10.8.12. Ventana de Correlaciones-Propiedades Roca-Fluido

Estas correlaciones pueden ser evidenciadas en las siguientes ilustraciones:



a).Datos del Modelo por tipo de roca | b).Datos del Modelo Corey

Ilustración 244:Ventana de la Sección Propiedades Roca Fluido – Izquierda: Modelo en función de tipos de roca-Derecha: Modelo General Corey. Fuente: Elaboración propia.

10.9. Algoritmos usados a nivel de métodos numéricos

A continuación, se muestra los algoritmos asociados a los métodos de interpolación y resolución de sistemas lineales

10.9.1. Algoritmo de para interpolación lineal

$if\ (xo \geq x_n)$

$\quad yo = y_n$

$else$

$\quad i = 2$

$\quad while\ (xo \geq x_n)$

$\qquad i = i + 1$

$\qquad if(i > n)$

$\qquad\qquad i = n$

$\qquad\qquad break$

$\qquad end\ if$

$\quad end\ do$

$end\ if$

$$yo = y_{i-1} + (xo - x_{i-1}) * \frac{y_i - y_{i-1}}{x_i - x_{i-1}}$$

10.9.2. Algoritmo de para interpolación por polinomio de Lagrange para polinomio de grado 2

$if\ (xo \geq x_n)$
$yo = y_n$
$else$
$i = 2$
$while\ (xo \geq x_n)$
$i = i + 1$
$if(i > n)$
$i = n$
$break$
$end\ if$
$end\ do$
$end\ if$

$if\ (n \geq 3)$
$il = i - 2$
$iu = i$
$if\ (il < 1)$
$il = 1$
$iu = il + 2$
$end\ if$
$for\ (j = il, 2,3 \ldots . iu)$
$prod = y_j$
$for\ (k = il, 2,3 \ldots . iu)$
$if\ (j = !\, k)\ prod = prod * \frac{xo - x_k}{x_j - x_k}$
$end\ for$
$yo = yo + prod$
$end\ for$
$else$
$for\ (j = 1,2,3)$
$prod = y_j$
$for\ (k = 1,2,3)$
$if\ (j = !\, k)\ prod = prod * \frac{xo - x_k}{x_j - x_k}$
$end\ for$
$yo = yo + prod$
$end\ for$
$end\ if$

10.9.3. Algoritmo de para interpolación por polinomio de Newton para polinomio de grado 2

$if\ (xo \geq x_n)$
$yo = y_n$
$else$
$i = 2$
$while\ (xo \geq x_n)$
$i = i + 1$
$if(i > n)$
$i = n$
$break$
$end\ if$
$end\ do$
$end\ if$

$if\ (n \geq 3)$
$il = i - 2$
$iu = i$
$if\ (il < 1)$
$il = 1$
$iu = il + 2$
$end\ if$

$ii = 0$
$order = 2$
$for\ (j = il, 2,3 \ldots . iu)$
$ii = ii + 1$
$xint_{ii} = x_j$
$yint_{ii} = y_j$
$end\ for$

$for\ (j = 1, \ldots . order)$
$fdd_{1,j} = y_{intj}$
$end\ for$
$for\ (k = 2, \ldots . order + 1)$
$for\ (j = 2, \ldots . order + 1 - k)$
$fdd_{k,j} = \frac{fdd_{k-1,j+1} - fdd_{k-1,j}}{xint_{k+j} - xint_i}$
$end\ for$
$end\ for$
$yo = fdd_{1,1}$
$for\ (k = 2, \ldots . order + 1)$
$prod = 1$
$for\ (j = 1, \ldots . k - 1)$
$pro = pro * (xo - xint_j)$
$end\ for$
$yo = yo + fdd_{k,1} * pro$
$end\ for$
$else$
$for\ (j = 1, \ldots . n)$
$fdd_{1,j} = y_j$
$end\ for$

$for\ (k = 2, \ldots . n)$
$for\ (j = 2, \ldots . n - k)$
$fdd_{k,j} = \frac{fdd_{k-1,j+1} - fdd_{k-1,j}}{xint_{k+j} - xint_i}$
$end\ for$
$end\ for$
$yo = fdd_{1,1}$
$for\ (k = 2, \ldots . order + 1)$
$prod = 1$
$for\ (j = 1, \ldots . k - 1)$
$pro = pro * (xo - xint_j)$
$end\ for$
$yo = yo + fdd_{k,1} * pro$
$end\ for$
$end\ if$

10.9.4. Algoritmo de Thomas para resolución de una Matriz Tridiagonal Cuadrada

$$beta_1 = bi_1$$

$$gamma_1 = \frac{d_1}{bi_1}$$

$for\ (i = 1,2,3 \ldots . n-1)$

$$w_i = \frac{c_i}{beta_i}$$

$$beta_{i+1} = bi_{i+1} - a_{i+1} * w_i$$

$end\ for$

$for\ (i = 1,2,3 \ldots . n)$

$$gamma_i = \frac{d_i - a_i * gamma_{i-1}}{beta_i}$$

$end\ for$

$$x_n = gamma_n$$

$for\ (i = 1,2,3 \ldots . n-1)$

$$x_{n-i} = gamma_{n-i} - w_{n-i} * x_{n-i+1}$$

$end\ for$

10.9.5. Algoritmo de Jacobi para resolución de una Matriz Cuadrada

$for\ (i = 1,2,3 \ldots . n)$

$xold_i = x_i$

$ai_{i,i} = bi_i$

$if\ (i > 1)\ ai_{i,i-1} = a_i$

$if\ (i < n)\ ai_{i,i+1} = c_i$

$end\ for$

$for\ (i = 1,2,3 \ldots . n)$

$for\ (j = 1,2,3 \ldots . n)$

$ai_{i,j} = \frac{ai_{i,j}}{ai_{i,i}}$

$end\ for$

$d_i = \frac{d_i}{ai_{i,i}}$

$end\ for$

$iter = 0$

$error = 1$

$while\ (iter < iter_{max}\ \ y\ error > tol)$

$for\ (i = 1,2,3 \ldots . n)$

$sum = d_i$

$for\ (j = 1,2,3 \ldots . n)$

$if\ (i = !\, j)\ sum = sum - ai_{i,j} * xold_j$

$end\ for$

$x_i = sum$

$end\ for$

$error = abs(\ x_1 - xold_1)$

$for\ (j = 2,3,4 \ldots . n)$

$if\ (error < abs(\ x_i - xold_i))\ error$

$= abs(\ x_i - xold_i)$

$end\ for$

$iter = iter + 1$

$for\ (j = 1,2,3 \ldots . n)$

$xold_i = x_i$

$end\ for$

$end\ do$

10.9.6. Algoritmo de Gauss-Seidel para resolución de una Matriz Cuadrada

$for\ (i = 1,2,3\ldots.n)$
$\quad xold_i = x_i$
$\quad ai_{i,i} = bi_i$
$\quad if\ (i > 1)\ ai_{i,i-1} = a_i$
$\quad if\ (i < n)\ ai_{i,i+1} = c_i$
$end\ for$

$for\ (i = 1,2,3\ldots.n)$
$\quad for\ (j = 1,2,3\ldots.n)$
$\quad\quad ai_{i,j} = \frac{ai_{i,j}}{ai_{i,i}}$
$\quad end\ for$
$\quad d_i = \frac{d_i}{ai_{i,i}}$
$end\ for$

$iter = 0$
$error = 1$

$while\ (iter < iter_{max}\ \ y\ error > tol)$
$\quad for\ (i = 1,2,3\ldots.n)$
$\quad\quad sum = d_i$
$\quad\quad for\ (j = 1,2,3\ldots.n)$
$\quad\quad\quad if\ (i = !\,j)\ sum = sum - ai_{i,j} * x_j$
$\quad\quad end\ for$
$\quad\quad x_i = sum$
$\quad end\ for$

$error = abs(\ x_1 - xold_1)$

$for\ (j = 2,3,4\ldots.n)$
$\quad if\ (error < abs(\ x_i - xold_i))\ error = abs(\ x_i - xold_i)$
$end\ for$

$iter = iter + 1$

$for\ (j = 1,2,3\ldots.n)$
$\quad xold_i = x_i$
$end\ for$

$end\ do$