Desarrollo de una Herramienta Libre para la Simulación Rigurosa de Columnas de Destilación en Estado Estacionario

UNIVERSIDAD
DE LOS ANDES

Gustavo A. León Díaz

Facultad de Ingeniería, Postgrado de Ingeniería Química

Universidad de Los Andes

Trabajo presentado como requisito parcial para obtener el grado de

Magister Scientiae

Junio 2014

Resumen

Desde el punto de vista de la ingeniería química, se puede decir que La Simulación es una técnica para evaluar en forma rápida un proceso químico con base en una representación del mismo mediante modelos matemáticos para tener un mejor conocimiento de su comportamiento. La solución de estos modelos se lleva a cabo por medio de programas de computadora denominados simuladores de procesos que facilitan la evaluación, control y optimización, especialmente en el caso de procesos complejos tal como los que se encuentran en la industria petrolera. No obstante, los simuladores de proceso utilizados actualmente se adquieren a través de una autorización formal con carácter contractual denominada licencia, teniendo solo el derecho a ejecutar el simulador bajo ciertas condiciones, comúnmente fijadas por el proveedor. Este tipo de licencias implican por lo general restricciones ya sea para su uso, estudio, modificación y/o distribución, por lo que el desarrollo del simulador depende totalmente de la empresa propietaria. Ante esta situación, se identificó DWSIM, un software bajo Licencia Pública General versión 3 (GPL v3, por sus siglas en inglés) para la simulación de procesos químicos, que cuenta con una estructura tecnológica y funcional semejante a los simuladores comerciales existentes. Por ser un programa con licencia libre, a diferencia de los programas propietario, respeta la libertad de los usuarios sobre el simulador y, por tanto, una vez obtenido puede ser usado, copiado, estudiado, modificado y redistribuido libremente. Además DWSIM posee una arquitectura flexible que permite incorporar componentes de terceros utilizando los estándares abiertos para simulación de procesos químicos CAPE-OPEN. Con el soporte de la libertad de modificación de código que brinda la licencia de DWSIM y su compatibilidad con los estándares abiertos CAPE-OPEN. En el presente estudio se logró su adaptación para el modelado de columnas de destilación en estado estacionario. Para su validación se utilizaron tanto datos experimentales disponibles en la literatura como datos operacionales de unidades de destilación de la industria petrolera nacional. En ambos casos los resultados obtenidos se mostraron en concordancia con los datos de validación obteniéndose porcentajes de desviación por debajo del 5 %.

Palabras Clave: Simulación de procesos, Destilación, CAPE-OPEN , Software Libre, DWSIM

A mi esposa *Claudina*, quien es impulso y el pilar principal de mi vida y carrera. Amiga, consejera, compañera, guerrera inseparable. Fuente de sabiduría, calma y consejo en todo momento. quien a pesar de las distancias momentáneas siempre me ha brindado su apoyo constante y amor incondicional.

A mi hermana Ysabel , por ser ejemplo de lucha, voluntad y éxito le hago el honor de este logro.

A mi sobrino Salvador, que Dios lo tenga en su gloria, angelito que encendió la vida de alegría.

Agradecimientos

A mis padres Ligia y Felipe, las personas que me dieron la oportunidad de vivir y que con su gran sabiduría y amor me han llevado a ser lo que soy.

A mi Amada Esposa Claudina, por su amor, paciencia y motivación a o largo de estos años de vida que hemos compartido.

A la Ilustre Universidad de Los Andes, por brindarme la oportunidad de realizar mis estudios de postgrado, para llevar a cabo el presente trabajo especial de grado.

A Daniel Wagner, por su incondicional ayuda y brindarme la oportunidad de formar parte del gran proyecto de DWSIM.

A César Pernalete, que con su gran apoyo, experiencia y consejos, colaboró de manera importante a la elaboración del presente trabajo especial de grado,

A mi primo Javier Rincón, por siempre creer siempre en mi y su excelente apoyo en el camino de ser un mejor programador.

Al personal de las Gerencias de RIRF y RIVC de Intevep Norte 3 Piso 4, por hacerme sentir como uno más del equipo.

A La Profesora Yezabel Rivera, por su orientación, su apoyo incondicional y consideración.

A Mis Compañeros de Postgrado Yanira, Aram, Glenda por su gran apoyo y ayuda con quienes compartí una vida post-universitaria llenas de excelentes y gratas experiencias.

A Marbeth le agradezco enormemente su maravillosa y excelente atención, calidez humana y comprensión desde el inicio de mis estudios de postgrado.

A Luís, Anggy, Deivis y Jorge por ser parte significativa en mi estadía en Los Téques, y por haber hecho el papel de familia, gracias por su apoyo y amistad.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Los procesos químicos en la industria petrolera, como tecnologías destinadas al procesamiento del petróleo y sus derivados, poseen un grado de complejidad alto en comparación con los procesos de otras industrias químicas (Luque & Vega, 2005). Los procesos en la industria petrolera se han diseñado de acuerdo a las características de los crudos disponibles en el mercado y las especificaciones de calidad de sus productos. En cuanto a las propiedades de los crudos, es bien conocido el decrecimiento de la disponibilidad mundial de petróleo convencional, trayendo como consecuencia la necesidad de considerar crudos con mayor viscosidad, densidad, heteroátomos y compuestos de alto peso molecular. Por esta razón se hace indispensable la utilización de nuevos procesos de tratamiento y transformación para satisfacer la demanda y cumplir con los estándares de calidad de producto (PDVSA, 2005; MathPro, 2011).

Debido a los nuevos desafíos que se presentan con este tipo de crudo, especialmente en el depósito, transporte y procesamiento, a causa de sus propiedades fisicoquímicas (Speight, 2008), la disponibilidad de herramientas que faciliten el diseño, evaluación y optimización de estos procesos es imperativa. Una de estas herramientas son los simuladores de procesos químicos, los cuales ahorran considerablemente el tiempo en cálculos prolongados y repetitivos, considerando así a la simulación de procesos una actividad fundamental en la ingeniería de procesos químicos (Dimian, 2003). Un caso de estudio concreto y típico en la industria petrolera, es el modelado y simulación de columnas de destilación, teniendo en cuenta a la destilación como una de las operaciones más importantes en la industria petrolera, tanto en el acondicionamiento previo de los crudos como en la separación de los productos obtenidos en refinerías.

Los simuladores de uso general en ingeniería de procesos, cuentan con las funcionalidades y características necesarias para el análisis de columnas de destilación mediante la configuración de diagramas de flujo de proceso y las operaciones unitarias que las conforman (Finlayson, 2006). No obstante, los simuladores utilizados actualmente en la in-

dustria petrolera nacional se adquieren a través de una autorización formal con carácter contractual denominada licencia mediante un pago recurrente, teniendo solo el derecho a ejecutar el *software* bajo ciertas condiciones, comúnmente fijadas por el proveedor. Este tipo de licencias establecen por lo general restricciones de uso, estudio, modificación o de distribución (Abella et al., 2003).

En este sentido al presentarse dificultades técnicas en relación con el modelado y simulación de columnas de destilación en función de nuevos requerimientos a nivel nacional e internacional, y dada las restricciones mencionadas, se incurre en la dependencia hacia fabricante del *software* a responder por una solución, lo que constituye un obstáculo a la industria para profundizar en los aspectos técnicos de la herramienta, limitando la incorporación de funcionalidades o mejoras orientadas a la representación más exacta de esta operación unitaria.

En este trabajo se propone desarrollar una herramienta para la simulación rigurosa de columnas de destilación en estado estacionario a partir de un *software* para la simulación de procesos químicos denominado DWSIM, aprovechando principalmente, su característica de código abierto bajo la Licencia Pública General de GNU y en segundo lugar, su compatibilidad con los estándares de interfaz CAPE-OPEN, un conjunto disponible de estándares que permite la comunicación entre componentes de *software* de ingeniería química.

1.1. La Simulación en la Ingeniería Química

Partiendo del concepto de simulación desde el punto de vista de la ingeniería química, los sistemas considerados son los procesos físicoquímicos, en este sentido se define la simulación como una técnica para evaluar en forma rápida un proceso químico con base en una representación del mismo mediante modelos matemáticos para tener un mejor conocimiento de su comportamiento. La solución de estos modelos se lleva a cabo por medio de programas de computadora debido al número de variables involucradas y la no linealidad de estos modelos, las cuales se reflejan en las ecuaciones de balance de materia y energía (Martínez, 2000; Braunschweig & Gani, 2002).

En los últimos años debido a los avances en la tecnología de la información, la simulación de procesos ha tenido un impacto significativo en la ingeniería química, tanto que ésta ha llegado a ser una técnica de apoyo imprescindible para la solución adecuada de los problemas en la ingeniería de procesos. De esta forma se creó un nuevo paradigma donde la simulación está involucrada en todas las etapas del ciclo de vida de un proceso, desde los experimentos de laboratorio preliminares, durante el escalado a diferentes niveles, hasta finalmente el diseño del proceso y operación de una planta. La Figura 1.1 ilustra este

nuevo enfoque colocando a la simulación en el centro de las tres actividades principales de ingeniería de procesos.

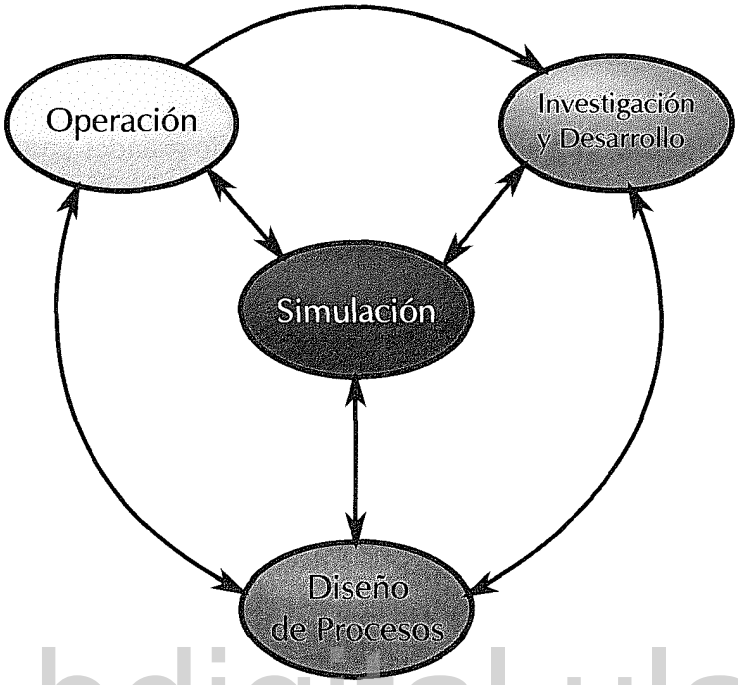


Figura 1.1: El nuevo paradigma de la Ingeniería de Procesos: La simulación como actividad principal en investigación y desarrollo, diseño y operación
Fuente: Dimian (2003)

En el área de la investigación y desarrollo, en particular en el campo de la termodinámica. Los modelos incorporados en paquetes de simulación sacan partido de la investigación experimental. Por ejemplo, por un lado se pueden utilizar experimentos de Equilibrio Vapor-Líquido (EVL) en el laboratorio que proveen datos para la obtención de valores de los parámetros en formas matemáticas para la construcción de un modelo termodinámico. Así, los modelos resultantes se pueden usar en diversas aplicaciones, incluyendo cálculos de equilibrios químicos de reacción, que son importantes para el diseño de reactor químico y cálculos de equilibrio de fases, que se emplean en la destilación, extracción con disolventes y la cristalización (O'Connell & Haile, 2005). Por otra parte, las mediciones EVL industriales se pueden usar para calibrar los modelos termodinámicos incorporados en un simulador cuando la información experimental es limitada o no se encuentre disponible(Dimian, 2003).

Dentro del marco de diseño, se presentan los retos de diseñar los procesos con mayor eficiencia, flexibilidad y capacidad de respuesta a la dinámica del mercado. En este sentido,

la simulación de procesos trae contribuciones significativas en el el desarrollo de nuevas tecnologías sostenibles con el objetivo de minimizar el uso de energía, materias primas, así como también la minimización de residuos y contaminantes y la mejora continua de las tecnologías existentes, por renovación y eliminación de cuellos de botella a nivel operacional (Martínez, 2000).

En el ámbito operacional, la llegada de la optimización en tiempo real y la fabricación integrada computarizada en la década de los noventa abrió grandes oportunidades para la aplicación de la simulación directamente en el proceso de fabricación. Además del control de proceso basado en modelos, podemos mencionar el mantenimiento preventivo de equipo de vigilancia sistemática del rendimiento del equipo. La integración de la fabricación con la cadena de suministro sólo se puede lograr mediante la creación de un sistema informático complejo, en el que la simulación juega un papel central (Braunschweig & Gani, 2002; Pernalete, 2013).

Aunado a esto es muy grande la variedad de aplicaciones de los simuladores de procesos, estos anteriormente estaban destinados a la ingeniería de procesos, ahora son manejados en ramas de ingeniería ambiental, seguridad industrial y distintos procesos de manufactura de productos cotidianos. Además, se emplean tanto para el estudio de la factibilidad técnica y económica en el desarrollo de un proyecto, así como también en la toma crítica de decisiones cuando se prueban diferentes opciones de procesos y condiciones de operación (Dimian, 2003; Martínez, 2000).

1.2. El Simulador de Procesos

La simulación de procesos químicos como técnica requiere de un programa de *software* denominado simulador de procesos. El simulador se puede definir como una aplicación de computador basadas en diagramas de flujo de proceso, vinculadas al cálculo de sus balances de materia y energía, partiendo de la descripción sistemática de sus corrientes de materia, energía y operaciones unitarias, con el objetivo de diseñar una nueva planta o evaluar una ya existente para mejorar su rendimiento (Martínez, 2000).

1.2.1. Reseña Histórica

Según Zeigler et al. (2000) el comienzo de los simuladores se encuentra en las universidades antes de los años 60. Sin embargo, los simuladores de procesos químicos surgieron a mediados de los años 60 con la comercialización de un *software* genérico para simular columnas de destilación en estado estacionario por parte de la compañía estadounidense *Simulation Science* el cual usaron para desarrollar *PROCESS*. Años más tarde *ChemShare*

presentó *DESIGN 2*, un programa para simular procesos de gas y petróleo. En ese momento la expansión de la industria de refinación y petroquímica motivó la llegada de los paquetes de computación (Dimian, 2003).

La crisis mundial de petróleo en 1973 estimuló el interés en la simulación de procesos con materias primas alternativas, como el carbón y la biomasa. El advenimiento de los sistemas de computación de alta velocidad impulsaron el negocio de pequeñas empresas especializadas en el modelado y la simulación de procesos. En líneas generales, la computación científica evolucionó a partir de los programas individuales hacia grandes paquetes diseñados como productos industriales (Luque & Vega, 2005).

Para los años 80 el cálculo científico llegó a su edad de oro, ya que se desarrollaron métodos que forman parte de los algoritmos utilizados en la actualidad. El lenguaje de programación FORTRAN se hizo muy popular entre los científicos y los ingenieros. Las simulaciones se ejecutaban a través de un terminal remoto en sistemas rápidos pero costoso. Más tarde, la entrada de datos se hizo posible mediante la edición de un archivo en una pantalla electrónica, en el cual se especificaban las instrucciones para la ejecución del trabajo denominado "*Keyword File*" (Babu, 2004).

La llegada de las estaciones de trabajo (1985), la adopción de la programación orientada a objetos y la disponibilidad de un nuevo sistema operativo multitarea, *UNIX*, dio origen a la revolución en la computación científica. Luego el dominio de la computadora personal (PC) a principios de 1990 y la relativa estabilización de los sistemas operativos, *LINUX* y *Windows* , permitieron el desarrollo de la nueva generación de *software* de simulación basada en interfaz gráfica de usuario. De esta manera se tendría disponible el poder de los antiguos superordenadores bajo un ambiente más manejable para los usuarios (Dimian, 2003). Los nuevos conceptos de ingeniería informática condujeron a interfaces amigables e incluso a algoritmos más potentes. Finalmente, el desarrollo de los ordenadores personales contribuyó a la rápida extensión de la simulación.

Así, las principales oficinas de ingeniería como algunas empresas y grandes compañías de refinación y petroquímica desarrollaron sus propios simuladores. Adoptando la arquitectura modular secuencial, tales como : *Hysys*, *Chemcad*, *Aspen Plus* y *PRO/II* . Sin embargo, algunos se basaron en el enfoque orientado a las ecuaciones, tal como *SPEEDUP* del Imperial College de Londres (Reino Unido) y *TISFLO DSM* en los Países Bajos.

No obstante, hoy en día la simulación de procesos químicos se concentra en muy pocos sistemas. Dar solución a esto implica una gran cooperación entre las empresas de *software* especializadas y la comunidad de usuarios aprovechando la tecnología de Internet (Dimian, 2003).

1.2.2. Clasificación de los Simuladores

Se han propuesto diferentes enfoques para clasificar los simuladores de proceso como el propuesto por Martínez (2000) y Torres & Castro (2002). Sin embargo, dichos enfoques en la literatura incluyen solamente a los simuladores privativos. A continuación se presenta una clasificación generalizada la cual incluye a los simuladores de procesos disponibles actualmente.

- Según su filosofía de cómputo

Los simuladores de procesos actuales se pueden clasificar según la estructura bajo la cual se plantea el modelo matemático que representa el proceso, dentro de esta clasificación se incluyen los siguientes tipos:

- Simuladores globales u orientados a ecuaciones (OE).

  Bajo este enfoque, se plantea el modelo matemático del proceso construyendo un gran sistema de ecuaciones diferenciales, por lo general altamente no lineales. De esta forma el problema se traduce en resolver este gran sistema de ecuaciones algebraicas, siendo el más adecuado para la simulación dinámica (Luque & Vega, 2005). En la Tabla 1.1 se muestran las ventajas y desventajas de estos simuladores.

Tabla 1.1: Ventajas y desventajas de los simuladores de procesos globales u orientados a ecuaciones

Ventajas	Desventajas
• Entorno flexible para las especificaciones, que pueden ser entradas, salidas o variables internas de la unidad. • Es posible incorporar fácilmente las expresiones de restricción para definir problemas de optimización en forma directa, ya que basta solo con plantear las restricciones y la función de optimización. • Desaparece la distinción entre variables de proceso y parámetros operativos, por lo tanto se simplifican los problemas de diseño.	• Mayor esfuerzo de programación por lo que resulta más difícil de desarrollar. • Dificultades en el manejo de grandes sistemas de ecuaciones, a mayor complejidad, menor confiabilidad en los resultados y dificultad de diagnóstico por parte del usuario en caso de error que produzca la falta de convergencia. • Tanto los modelos matemáticos como las rutinas para cálculo de propiedades termodinámicas deben ser ingresados por el usuario, resultando esto una tarea compleja y que debe ser llevada a cabo con sumo cuidado, de manera tal de ingresar en forma correcta todos los modelos matemáticos.

- Simuladores Secuenciales Modulares (SM).

  Los simuladores secuenciales modulares se basan en unidades de simulación que siguen consecutivamente como su nombre lo indica la misma secuencia que las operaciones unitarias reales, es decir, cada equipo: bomba, válvula, intercambiadores, columna de destilación, reactores entre otros; son representados a través de modelos específicos, coincidiendo además el sentido de la información con el "flujo físico" en la planta o proceso real, llevando a cabo una secuencia de cálculo unidad por unidad. Se puede decir que el elemento básico en un simulador modular es el modelo de la operación unitaria (Luque & Vega, 2005; Martínez, 2000; Scenna et al., 1999).

  Los simuladores SM, involucran el dibujo del diagrama de flujo del proceso, el cual consiste en definir las corrientes de entrada y salida de materia y energía, seleccionar los modelos de operaciones unitarias de la biblioteca de la aplicación, y conectar dichas unidades junto a corrientes (materia, energía o información) y de esta manera producir una representación gráfica del problema de la simulación (Braunschweig & Gani, 2002).

  Asociado con cada modelo de operación unitaria o unidad de cómputo, existe un conjunto de parámetros requeridos, específicos de cada equipo que representa; por ejemplo, en un mezclador, sería la proporción del flujo total en cada corriente de salida en los intercambiadores de calor, serán los coeficientes de transferencia de calor y en un reactor catalítico será la altura del lecho catalítico, entre otros parámetros que se deben colocar en orden correcto.

  En resumen, una unidad de cómputo es en general cualquier conjunto de modelos matemáticos para calcular la información de salida a partir de la información de alimentación o de entrada suministrada (Figura 1.2); es decir, una unidad de cómputo es un modificador de información que recibe información de las corrientes de entrada, las modifica y las transmite a las corrientes de salida (Torres & Castro, 2002). Este enfoque se utiliza en la mayoría de los programas de simulación en estado estacionario. En la Tabla 1.2 se muestran las ventajas y desventajas de estos simuladores.

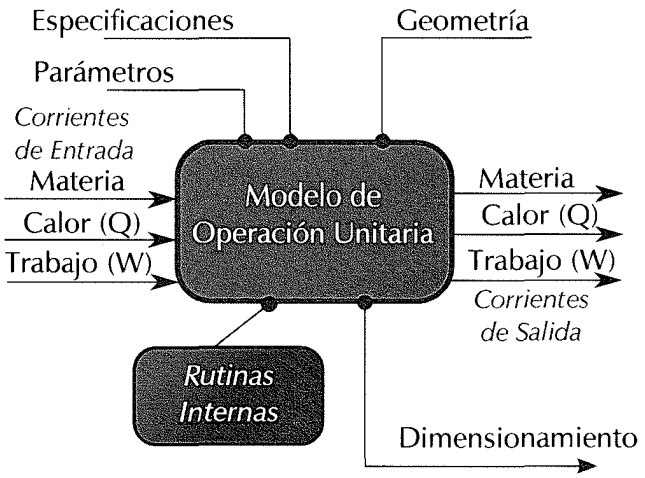


Figura 1.2: Esquema general del modelo de unidad de operación
Fuente: Dimian (2003)

Tabla 1.2: Ventajas y desventajas de los simuladores de procesos secuenciales modulares

Ventajas	Desventajas
■ Los modelos de operaciones unitarias están a disposición del usuario bajo una biblioteca y la información ingresada por el usuario resulta fácil de chequear e interpretar. ■ Posee un esquema simple de cómputo, es decir, se obtienen datos de salidas a partir de datos de entrada.	■ Dificultad para el tratamiento de secuencias de cálculo más complejas, como los bucles anidados o diagramas de flujo simultáneo, en otras palabras, procesos con muchas corrientes de recirculación. ■ Los modelos de operaciones unitarias se deben codificar con sus rutinas de solución. Se requiere de un esfuerzo considerable para añadir o modificar los modelos, es necesario programar los cálculos adicionales según las especificaciones que se requieran.

- Simuladores híbridos o modular secuencial-simultáneo.

  La estrategia de solución es una combinación de los enfoques secuencial-modular y orientado a ecuaciones. Los cálculos se realizan por niveles o capas, es decir, los modelos rigurosos se utilizan en un nivel de unidades, que se resuelven de forma secuencial, mientras que los modelos lineales se utilizan a nivel de diagrama de flujo, que se resuelve de forma global. Los modelos lineales se actualizan sobre la base de los resultados obtenidos con los modelos rigurosos.

El enfoque Secuencial-Modular mantiene una posición dominante en la simulación en estado estacionario y el enfoque orientado a ecuaciones ha demostrado su potencial en la simulación dinámica y optimización en tiempo real. La solución para las futuras generaciones de simuladores parece ser una fusión de estas estrategias.

- Según su licenciamiento

Una licencia de *software* es un contrato entre el licenciante (autor/titular de los derechos de explotación/distribuidor) y el licenciatario del programa informático (usuario consumidor /usuario profesional o empresa), para utilizar el *software* cumpliendo una serie de términos y condiciones establecidas dentro de sus cláusulas (Abella et al., 2003).

Las licencias de *software* pueden establecer entre otras cosas, la cesión de determinados derechos del propietario al usuario final sobre una o varias copias del programa informático, los límites en la responsabilidad por fallos, el plazo de cesión de los derechos, el ámbito geográfico de validez del contrato e incluso pueden establecer determinados compromisos del usuario final hacia el propietario, tales como la no cesión del programa a terceros o la no re-instalación del programa en equipos distintos al que se instaló originalmente.

- Simuladores de Proceso bajo Licencia Privativa

  Dentro de esta categoría se encuentran los denominados simuladores comerciales y/o propietarios. Existe una gran variedad de simuladores de procesos bajo este tipo de licencia, los cuales presentan poderosos motores de cálculo con enormes bases de datos y un fuerte respaldo de bibliotecas para el cálculo riguroso y modelos termodinámicos. Las empresas que desarrollan este tipo de *software* son por lo general grandes y dedican muchos recursos, sobretodo económicos, en su desarrollo e investigación. Debido a este esfuerzo, y bajo la justificación de maximizar ganancias monetarias las empresas toman la decisión de restringir ciertos derechos al licenciatario entre ellos el más resaltante, el acceso al código fuente con la razón de proteger sus intereses (Culebro et al., 2006).

Tabla 1.3: Ventajas y desventajas de los simuladores de procesos químicos bajo licencia privativa

Ventajas	Desventajas
• Las compañías productoras de un simulador privativo por lo general poseen departamentos de control de calidad que llevan a cabo muchas pruebas sobre el software que producen. • Existen simuladores privativos diseñados para aplicaciones de procesos químicos muy específicas. Tales como procesos de destilación reactiva. • Existen compañías que destinan una parte importante de los recursos a la investigación sobre la usabilidad del simulador que producen. Por lo que usualmente los simuladores poseen interfaces gráficas elaboradas que permite al usuario hacer una simulación de procesos de manera muy fácil y rápida.	• Cursos de aprendizaje costosos. Es difícil aprender a utilizar eficientemente el simulador sin haber asistido a costosos cursos de capacitación. • Por lo general, no permiten que sea modificado, desensamblado, copiado o distribuido de formas no especificadas en la propia licencia. En muchos casos resulta arriesgada la utilización de un componente cuyo funcionamiento se desconoce y cuyos resultados son impredecibles, esto motiva a que en algunas circunstancias se desconfíe del *software*. Además de que es imposible encontrar la causa de un resultado erróneo, más allá de los mensajes de advertencia de que pude desplegar el simulador. • El costo de las licencias generalmente es muy elevado y puede oscilar entre los (5000 a 15000 USD/año). El costo usualmente se calcula en función de la cantidad de computadoras en las que el simulador será ejecutado, la envergadura de dichas máquinas (cantidad de procesadores) y la cantidad de usuarios que accederán al mismo. • Es ilegal y/o costosa la adaptación de un módulo nativo del simulador a necesidades particulares. Se requiere permiso expreso del titular del *software*. En caso de que el permiso se conceda, es necesario pagar una elevada suma de dinero. • El soporte de la aplicación depende totalmente de la empresa que distribuye y desarrolla el simulador o de representantes certificados la empresa. • El usuario se vuelve dependiente de una tecnología que no comprende de fondo, ya que está habilitado para ejecutar el simulador, pero no para inspeccionarlo ni modificarlo, y entonces no puede aprender de él.

- Simuladores bajo Licencia Libre

  Denominándose también simuladores libres o de código abierto. Están licenciados de tal manera que los usuarios no tengan ningún tipo de restricciones, respetando la libertad de los usuarios sobre el simulador y, por tanto, una vez obtenido, puede ser usado, copiado, estudiado, modificado, y redistribuido libremente (Culebro et al., 2006).

Tabla 1.4: Ventajas y desventajas de los simuladores de procesos químicos bajo licencia libre

Ventajas	Desventajas
• Garantizan una independencia con respecto al proveedor gracias a la disponibilidad del código fuente. Cualquier empresa o profesional, con los conocimientos adecuados, puede seguir ofreciendo desarrollo o servicios para la aplicación. • El usuario está habilitado para ejecutar el programa, inspeccionarlo y modificarlo, lo que permite aprender de él, a través de su código fuente. • El modelo de desarrollo se basa en compartir su código fuente. Esto permite el progreso del *software* a través de la cooperación comunitaria de usuarios finales, desarrolladores y empresas que pueden trabajar en conjunto para obtener un *simulador* de calidad. Además permite el desarrollo de nuevos productos sin la necesidad de crear todo el proceso desde cero. • El bajo o nulo costo permiten proporcionar a las pequeñas y medianas empresas desarrolladoras de software el fomento de la libre competencia al basar el negocio en servicios y no licencias. • El precio de adquisición de los simuladores suele ser mucho menor que sus contrapartidas privativas. • El hecho de que el código sea público hace que pueda ser observado y estudiado por muchos expertos proporciona mayor seguridad y fiabilidad. Esto se ha revelado como la forma mas rápida y eficaz de encontrar errores que afectan la vulnerabilidad del simulador, pudiendo detectar fácilmente códigos maliciosos o transacciones de información no autorizadas.	• Actualmente no existen compañías únicas que respalden el desarrollo de la simulación de procesos libre, por lo que su desarrollo apenas está en maduración. • En algunos simuladores la interfaz gráfica suele estar menos desarrollada en cuanto a usabilidad y, por tanto, limita la interacción entre el simulador y los usuarios finales debido a la baja estabilidad. • Algunos simuladores de procesos no ofrecen garantía explicitas proveniente de los desarrolladores.

1.2.3. Arquitectura general de un simulador de procesos y CAPE-OPEN

Los Simuladores difieren ampliamente en la arquitectura e implementación, pero todos tienen una funcionalidad común impuesta por las tareas de modelado subyacentes que abordan. Esta funcionalidad se puede resumir en términos de cuatro tipos de componentes clave conceptuales:

- El ejecutivo de simulación: Este componente es el núcleo del simulador, ya que controla la puesta en marcha y la ejecución de una simulación. Es responsable de instalar otros componentes, registrando en un repositorio, la gestión de las interacciones con los usuarios, el acceso y almacenamiento de datos, la presentación de informes y el análisis de los cálculos de simulación.

- Las operaciones unitarias: Estos componentes representan operaciones de la unidad de procesamiento físico (por ejemplo un mezclador o un intercambiador de calor) y, posiblemente puede realizar funciones especializadas tales como la realización de cálculos adicionales para apoyar a la optimización de procesos.

- Los paquetes de propiedades físicas: estos paquetes incluyen tanto propiedades termodinámicas y de transporte de materiales. Estos paquetes brindan al simulador la capacidad para modelar las propiedades y el comportamiento de los materiales que se utilizan o crean por el proceso químico.

- El solucionador numérico: Esto incluye tanto los métodos matemáticos especializados utilizados para evaluar las ecuaciones que describen un funcionamiento de la unidad de proceso y los métodos utilizados para evaluar el diagrama de flujo global del proceso.

A pesar de la existencia de potentes simuladores de proceso, los que se utilizan actualmente son aplicaciones monolíticas cerradas (Barrett et al., 2007). Éstos no brindan flexibilidad cuando se trata integrar nuevos componentes. Otro inconveniente de esta situación es que resulta un gran reto combinar módulos de diferentes proveedores en un solo simulador. Debido a estas contrariedades, existe una necesidad de la integración del proceso de modelado sobre una base más amplia. Así, se introduce un nuevo tipo de arquitectura de simuladores de procesos basada en estándares abiertos definidos en términos del diseño conceptual mencionado anteriormente.

La organización *CAPE-OPEN Laboratories Network* (CO-LaN), industria y asociación académica ha promovido desde el 2001 el uso de estándares en la interfaces de *software* de simulación de procesos denominados CAPE-OPEN, donde se definen un conjunto de reglas e interfaces libres de uso y sin propiedad que permiten crear componentes de *software* interoperables con distintas aplicaciones para simulación de procesos, los cuales permiten una comunicación estandarizada entre componentes de los simuladores (Pons, 2005).

La ventaja de los estándares CAPE-OPEN es que estos no solo son aplicables en más de un simulador de procesos, sino que se dirige a todos los simuladores de proceso que sean compatibles dichos estándares, creando una interoperatividad entre sistemas, evitando incompatibilidades. La definición de estos estándares se lleva a cabo mediante la colaboración de algunas empresas operadoras principales, proveedores de *software*, y universidades a nivel mundial, con el apoyo de la Unión Europea quienes tienen como visión final la de transformar el modelado de procesos en una actividad que consista en compartir un gran número de componentes de simulación. En este sentido los simuladores actualmente,

pueden diferenciarse bajo el criterio del cumplimiento con estos estándares, independientemente de la licencia a la cual estén sometidos (Dimian, 2003; CO-LaN, 1999).

Este tipo de arquitectura abierta puede ser beneficiosa para muchos simuladores. El objetivo específico del proyecto CAPE-OPEN se ha enfocado en herramientas generales para el modelado de procesos y, en particular, su utilización para la simulación en estado estacionario y dinámico (CO-LaN, 1999).

La arquitectura de *software* es un asunto de la informática. Sin embargo, como todo sistema complejo, el usuario debe ser consciente de los elementos principales. La Figura 1.3 presenta de forma gráfica el concepto de CAPE-OPEN. En este esquema se muestran las fronteras entre los diversos componentes que integran un simulador.

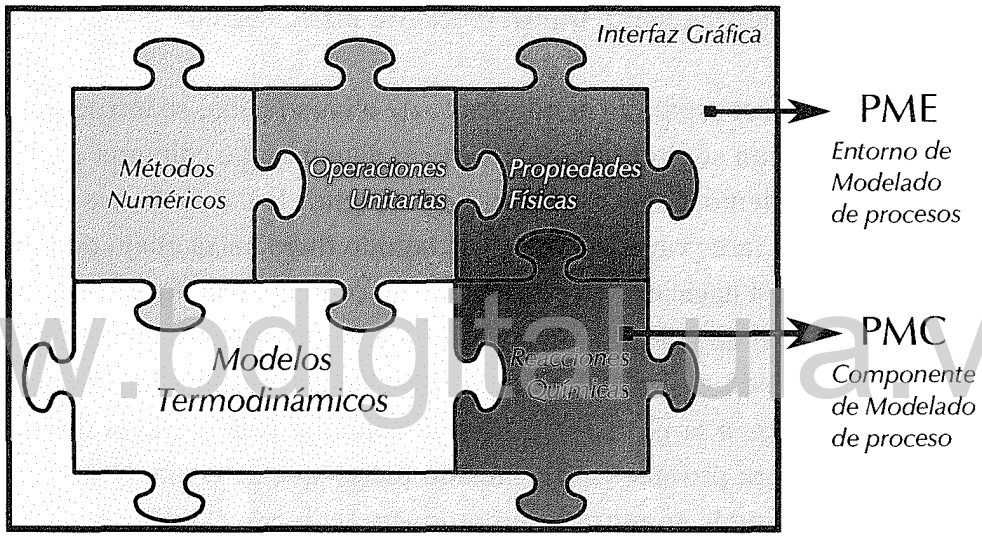


Figura 1.3: Enfoque conceptual CAPE-OPEN
Fuente: (Pernalete et al., 2012)

Resumiendo, los estándares CAPE-OPEN se crearon con la finalidad de que los componentes de modelado de procesos (PMC) sean utilizados en cualquier entorno de modelado de procesos (PME) que sea compatible con dichos estándares. Los PMC son básicamente piezas de software que están definidos para una función específica. La mayoría de sus aplicaciones son para: propiedades físicas, módulos de funcionamiento de una operación unitaria, solucionador numérico y herramientas de análisis del diagramas de flujo de proceso.

Por otro lado el PME consta de la interfaz gráfica y la funcionalidad necesaria para crear la red de modelo de procesos a partir de los PMC, quien se encarga de administrar dicha red basados en los datos de entrada suministrados por el usuario y de orquestar todas las

funciones que desempeñan cada uno de los PMC (Barrett & Yang, 2005; Pernalete et al., 2012).

1.3. Simulación del Proceso de Destilación

La destilación es una operación unitaria cuyo objetivo principal es el de separar mediante vaporización y condensación los diferentes componentes de una mezcla en dos o más fracciones, aprovechando la diferencia de los puntos de ebullición de cada uno de los componentes que la conforman (Kister, 1992; Martínez de la Cuesta & Rus, 2004). La destilación involucra el contacto entre las fases líquido y vapor que fluyen en contracorriente. Cada contacto, denominado etapa, consiste en una mezcla de fases para promover la rápida partición de las especies mediante transferencia de masa, seguida por una separación de fases.

El material que ha de ser separado denominado de alimentación, se introduce en uno o más puntos a lo largo de un equipo cilíndrico vertical denominado columna de destilación o columna de fraccionamiento, como se muestra en la Figura 1.4. Debido a la diferencia en las propiedades entre las fases vapor y líquida, el líquido fluye hacia abajo de la columna, mientras que el vapor fluye hacia arriba, poniéndose en contacto con líquido en cada etapa. Este patrón de flujo global en una columna de destilación proporciona el contacto contracorriente de vapor y líquido en todas las etapas a través de la columna (Perry & Green, 2008). El líquido que llega a la parte inferior de la columna pasa a través de un rehervidor, donde se calienta para proveer vapor llamado *boilup* que se envía de nuevo hasta la columna, y el resto del líquido se retira como producto de fondo (Seader et al., 2011). El vapor que llega a la parte superior de la columna, donde se enfría y se condensa parcial o totalmente en el condensador de tope. Parte de este líquido se devuelve a la columna como reflujo y El resto de este líquido se retira como destilado, o producto de tope.

Los componentes más ligeros (de menor punto de ebullición) tienden a concentrarse en la fase de vapor, mientras que los componentes más pesados (más alto punto de ebullición) tienden a hacer en la fase líquida (King, 1980). El resultado es una fase de vapor que se enriquece en componentes más ligeros y una fase líquida que se concentra más en componentes pesados a medida que viajan por la columna de destilación. La separación general entre el destilado y los fondos depende principalmente de las volatilidades relativas de los componentes, el número de etapas de contacto y la relación de la tasa de flujo de la fase líquida a la velocidad de flujo en fase vapor.

La energía requerida para separar la especie se añade en forma de calor al rehervidor, donde la temperatura es mayor. También, se elimina el calor desde el condensador, donde

la temperatura es más baja. Con frecuencia, esto da lugar a una gran demanda de energía y baja eficiencia global termodinámica (Perry & Green, 2008).

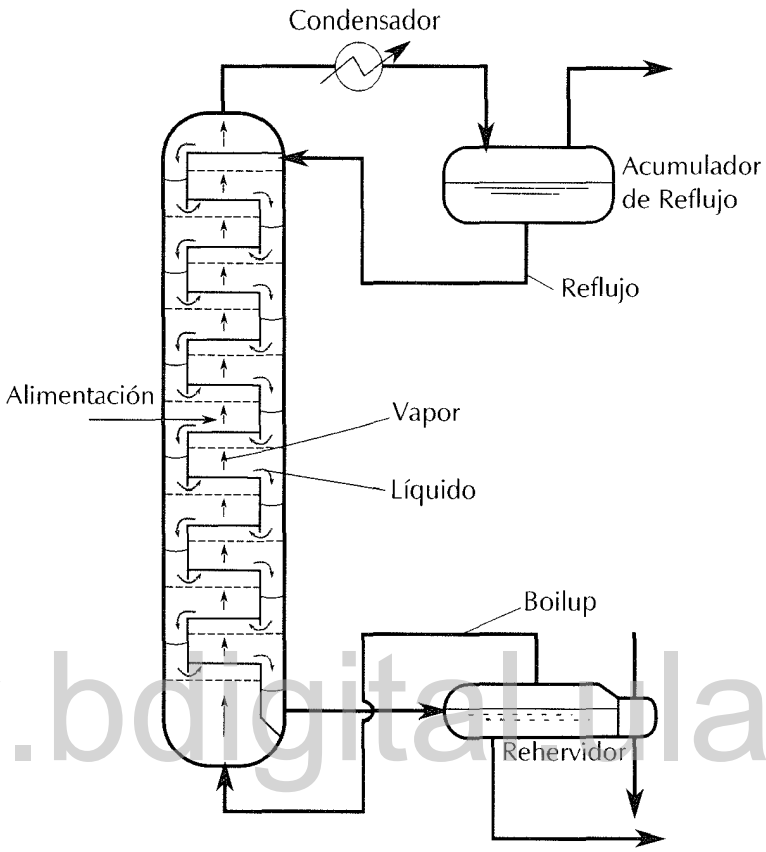


Figura 1.4: Figura esquemática del proceso de destilación

La destilación como operación unitaria es el proceso de separación más importante y común en la industria de procesos químicos, pues por su versatilidad es la tecnología dominante en las industrias químicas. Dicha operación se emplea en las industrias donde se requiere destilar grandes cantidades de fluidos. Entre estas se encuentran: las industrias de procesamiento de petróleo, producción petroquímica, procesamiento de gas natural, procesamiento de alquitrán de hulla, elaboración de cerveza, separación de aire licuado, y la producción de hidrocarburos y solventes industrias similares, pero que encuentra su mayor aplicación en refinerías de petróleo (Kister, 1992). La destilación está implementada en el 95 % de todos los procesos químicos en el mundo (por ejemplo más de 40000 columnas en USA, con una inversión de 8000 MM$ y una energía equivalente a 54 Mton/año de crudo, es decir el 15 % del consumo industrial de energía en USA) (Luque & Vega, 2005). Lo que justifica que la destilación sea una de las operaciones unitarias más estudiadas en toda la historia de la industria química.

Específicamente en la industria del petróleo nacional, la importancia de la destilación se debe en parte al hecho de que afecta directamente la calidad de los productos obtenidos en refinería, las tasas de producción y la utilidad de los procesos. Por esta razón, la evaluación, control y optimización de estas operaciones es fundamental. Sin embargo, presenta inconvenientes para su análisis, requiriendo de rigurosos cálculos, donde los procedimientos de solución resultan difíciles y tediosos sin la ayuda de un computador. No obstante, una vez que éstos cálculos se programan para un ordenador digital de alta velocidad las soluciones son obtenidas rápidamente (Torres & Castro, 2002).

La simulación de unidades de destilación implica la solución del modelo que las representa. En este sentido, involucra la determinación de perfiles de temperatura, presión, flujos y composiciones de las corrientes, y las tasas de transferencia de calor en cada etapa mediante la resolución de las ecuaciones de balance de materia, energía, y las relaciones de equilibrio que modelan el comportamiento de la unidad de destilación. Una característica fundamental del modelo, es que debe ser capaz de representar los diversos tipos de esquemas que se puedan presentar para el proceso de destilación. En este sentido, la disponibilidad de una herramienta es fundamental. Detalles referente al modelado y simulación de estas unidades se darán en el Capítulo 2.

1.4. Objetivos y Alcance de esta tesis

Se propuso el desarrollo de una herramienta libre para la simulación rigurosa de columnas de destilación en estado estacionario, a partir de un simulador de procesos desarrollado bajo licencia libre y estándares abiertos, aprovechando la gran cantidad de funcionalidades que ya se encuentran disponibles en él. En este sentido se seleccionó DWSIM, un simulador de procesos químicos de código abierto compatible con los estándares CAPE-OPEN, y que cuenta con una interfaz gráfica de usuario (*GUI*), con la capacidad de simular los procesos de equilibrio vapor-líquido y vapor-líquido-líquido en estado estacionario, con los modelos termodinámicos y operaciones unitarias más comunes.

Por el hecho de tratarse de un simulador de procesos químico de código abierto, permitió su modificación para que sea adaptado en estos casos particulares de la industria petrolera nacional, gracias al hecho de que se dispone del código fuente y ninguna de las restricciones conocidas del *software* privativo. Este simulador ofrece la posibilidad de su uso extendido en un período de tiempo prolongado, al no requerir el pago de licencias y las consecuencias del bloqueo de su uso tal como sucede con los simuladores privativos. En este sentido, a través de este trabajo se pudo modificar una herramienta de simulación y

obtener un *software* lo suficientemente robusto y adecuado a casos particulares de procesos de destilación de Petróleos de Venezuela S.A (PDVSA).

Las investigaciones para el desarrollo de modelos de columnas de destilación han avanzado con la finalidad de reproducir el comportamiento del proceso real con la mayor exactitud posible (Eckert & Vaněk, 2001; Grossmann et al., 2005; Higler et al., 2004). Este trabajo se plantea desarrollar las rutinas de cálculo para la simulación de unidades de destilación multicomponente en estado estacionario basado en modelo de relaciones de etapas de equilibrio conectados a contracorriente, particularmente adecuado para la mayoría de las columnas de fraccionamiento.

Su codificación se realizó tomando en cuenta la arquitectura bajo la cual está desarrollada DWSIM, para luego verificar y validar la herramienta a través del análisis de dos tipos de casos de estudio :

1. Para el primero se utilizaron ejemplos de literatura y con estos comprobar la verificación del algoritmo implementado. En este caso se configuraron dichos ejemplos además bajo otros dos simuladores y se compararon los resultados obtenidos entre los distintos simuladores, considerando su reproducibilidad.

2. En el segundo se tomaron datos operacionales de unidades destilación con los cuales se realizó su modelado y simulación y de esta manera realizar la validación del modelo implementado.

1.5. Estructura de la Tesis

El capítulo 2, contiene una descripción de los métodos de solución de los modelos matemáticos para columnas de destilación en estado estacionario, considerando en detalle el algoritmo INSIDE-OUT de Russell (1983) como elemento esencial en la simulación de procesos de destilación.

El capítulo 3, trata sobre la arquitectura lógica del simulador de procesos DWSIM, acá se realiza una descripción general de su estructura interna, de los elementos que lo forman y de las interrelaciones entre ellos. Para poder llevar esto a cabo, fue necesario utilizar técnicas de Ingeniería Inversa, lográndose identificar las clases y métodos involucrados en la simulación de columnas de destilación para así incluir las modificaciones necesarias en el código fuente del simulador.

Seguidamente se presenta el capítulo 4 donde se discuten los resultados de los casos de estudio empleados para la verificación y depuración del código implementado, para comprobar que se han programado adecuadamente las estructuras de datos y algoritmos

del modelo que representa al proceso de destilación en estado estacionario, así como también la validación del modelo en base a datos operacionales reales del circuito nacional de refinación.

Continuando con el capítulo 5 se presentan las conclusiones obtenidas de la ejecución de este trabajo tanto de la implementación del algoritmo en DWSIM y también se plantean las recomendaciones respectivas.

Capítulo 2

Modelado y Simulación del Proceso de Destilación

Para calcular o predecir el rendimiento de los sistemas de múltiples etapas, se efectúa el planteamiento de modelos y ecuaciones que relacionan las diversas variables que interactúan dentro de dicho sistema y, debido a la complejidad de las ecuaciones, la única forma práctica para resolver el modelo de forma rigurosa es por medio de algoritmos de solución implementados en programas de computadora (Khoury, 2005).

Este capítulo comienza con el desarrollo del modelo matemático empleado para la simulación de columnas de destilación en estado estacionario, basado en etapas de equilibrio de contacto vapor-líquido donde se incluyen las ecuaciones de balances de masa por componentes, equilibrio de fases, ecuaciones auxiliares y de balances de energía, comúnmente conocidas como las ecuaciones $MESH$ (del inglés *Mass*, *Equilibrium*, *Summation*, *entHalpy*). Posteriormente se describen varias estrategias para resolver estas ecuaciones, que actualmente se encuentran implementadas en el simulador de procesos DWSIM, considerando principalmente el algoritmo *Inside-Out* para su desarrollo e implementación en DWSIM. Este algoritmo se caracteriza por ser el método de resolución más flexible y complejo, utilizado ampliamente en los simuladores de procesos comerciales para los casos de destilación multicomponente.

2.1. Descripción del Modelo de Columna

2.1.1. Modelo de Etapa de Equilibrio

El desarrollo y la aplicación de los modelos basados en el concepto de etapa de equilibrio para la destilación, ha sido objeto de estudio durante muchos años. Hasta hace poco, los procesos de transferencia de masa y energía en una columna de destilación real se con-

sideraban muy complicados para ser modelados de manera fácil y directa. Esta dificultad fue sobrellevada por el modelo de etapa de equilibrio, desarrollado por Sorel (1893). Este modelo conceptual (Figura 2.1) se fundamenta en suponer que los flujos de vapor y líquido que salen de cualquier etapa de la columna, se encuentran en equilibrio termodinámico, es decir que se consideran no tienen tendencia a alejarse de sus actuales condiciones definidas por su presión, temperatura y composiciones.

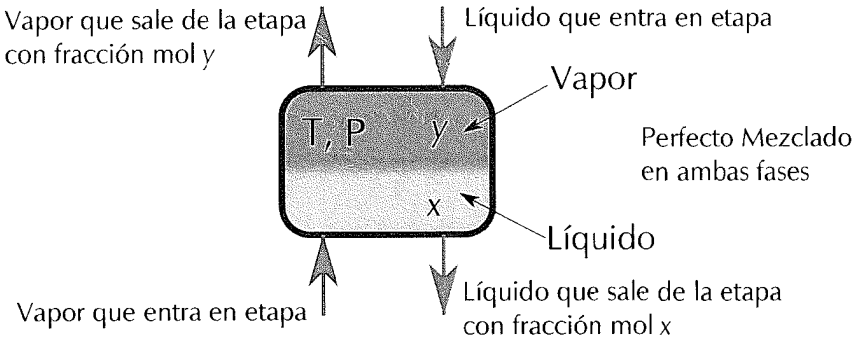


Figura 2.1: Diagrama conceptual de una etapa de equilibrio.(Wilson et al., 2000)

Basado en el concepto anterior, una sección de la columna de destilación se modela como se muestra en la Figura 2.2a, considerando las siguientes suposiciones:

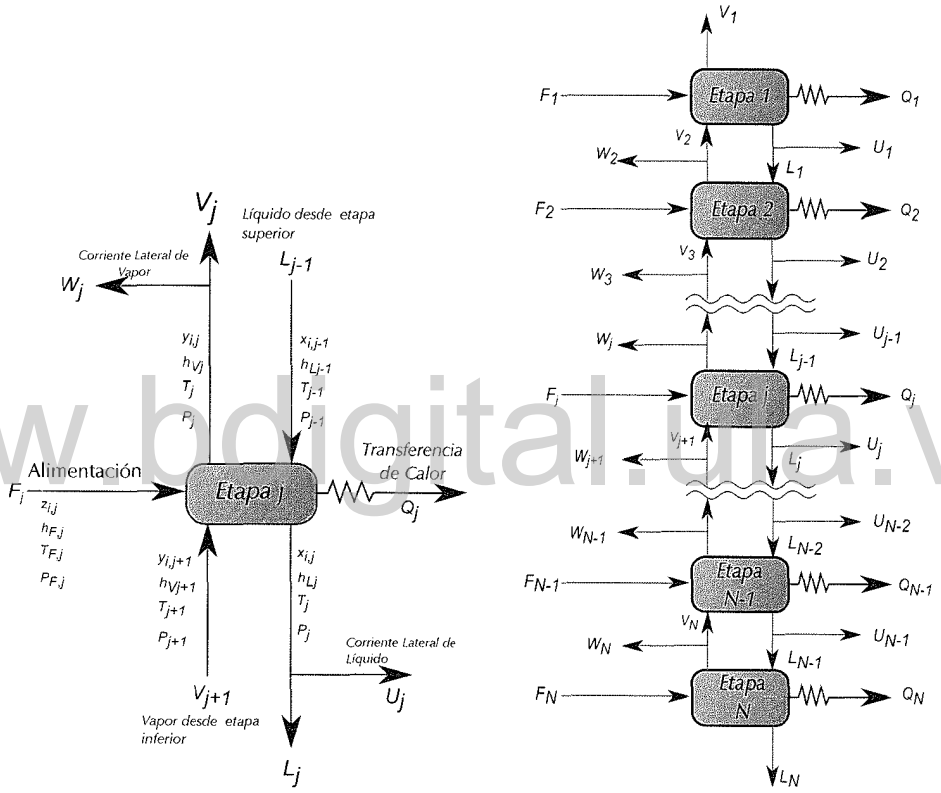
1. Se logra en cada etapa un equilibrio de fases.
2. No se producen reacciones químicas.
3. Se desprecia el arrastre de gotas de líquido en el vapor y la oclusión de burbujas de vapor en el líquido.

Así, una columna hipotética se define a partir del acople de varias de estas etapas de equilibrio como de muestra en la Figura 2.2b. Este sistema en cascada en contracorriente está diseñado para llevar a cabo la separación especificada por la columna real, siendo una descripción razonable de la física real (Wilson et al., 2000). De ser necesario, el número de etapas de equilibrio se puede convertir a su equivalente en etapas reales por medio del concepto eficiencia de etapa, que describe la medida en que el rendimiento de una bandeja de contacto real al rendimiento de una etapa de equilibrio (Seader et al., 2011).

En este esquema, una o más corrientes de materia entran en la etapa, y una o más corrientes salen de ella. Así, también se considera una corriente especial que representa el flujo de energía que se suministra o se retira de la etapa. Tomando como referencia la Figura 2.2a en a la etapa j, existe una corriente de alimentación de flujo molar F_j, con una

composición global en fracciones mol $z_{i,j}$ del componente i, temperatura T_{Fj}, presión P_{Fj} y su correspondiente entalpía molar global h_{Fj}. La presión de la alimentación es igual o mayor que la presión de la etapa P_j.

También en esta etapa se encuentra el líquido ínter-etapa proveniente de la etapa superior $j-1$, de flujo molar L_{j-1}, con una composición en fracción molar $x_{i,j-1}$, entalpía $h_{L_{j-1}}$, temperatura T_{j-1} y presión P_{j-1} . La presión del líquido de la etapa $j-1$ se incrementa de manera adiabática por el cabezal hidrostático de L_{j-1}. De manera similar, desde la etapa inferior $j+1$, entra el vapor ínter-etapa de flujo molar V_{j+1} de igual forma, con una composición en fracción molar $y_{i,j+1}$, entalpía $h_{V_{j+1}}$, temperatura T_{j+1} y presión P_{j+1}.



(a) esquema de una etapa de equilibrio general

(b) cascada a contracorriente General de N etapas

Figura 2.2: Modelo generalizado para columna múltiples etapas (Seader et al., 2011)

Dejando la etapa j, se encuentra también el vapor con propiedades intensivas $y_{i,j}$, h_{V_j}, T_j, y P_j. Esta corriente se puede dividir en una corriente lateral de vapor de flujo molar W_j que sale como producto y una corriente ínter-etapa de flujo molar V_j que bien va hacia la etapa superior $j-1$ o sale como producto al ser $j=1$. También, de manera análoga dejando la etapa j se presenta el líquido con propiedades intensivas $x_{i,j}$, h_{L_J}, T_j, y P_j, el cual se encuentra en equilibrio con el V_j y W_j. Este líquido de manera similar que el vapor, se puede dividir en una corriente lateral de flujo molar U_j y una corriente ínter-etapa y de

flujo molar L_j que se dirige hacia la etapa $j+1$, y sale como producto si $j = N$. Por último se puede transferir energía a una razón Q_j desde $(+)$, o hacia $(-)$ la etapa j para simular intercambios de calor. Las etapas se numeran del tope al fondo de la columna: desde $j = 1$ en el condensador o la parte superior a la etapa $j = N$ en el rehervidor o la etapa inferior.

2.1.2. Ecuaciones MESH

Según Wang & Henke (1966) el modelo de etapa de equilibrio de columnas de destilación, está representado por un conjunto de ecuaciones comúnmente conocidos como las ecuaciones *MESH*, las cuales son las que se utilizan para describir la operación en estado estacionario de una columna de destilación. *MESH* provienen del inglés: ***M**aterial balances* (Balances de materia), ***E**quilibrium equations* (Ecuaciones de Equilibrio), ***S**ummation equations* (Ecuaciones de Sumatorias), *ent**H**alpy balances* (Balances de entalpía) (Treccani, 2008; Seader et al., 2011). A continuación se describe con detalles cada una de las ecuaciones:

1. Las ecuaciones M se emplean para modelar la ley de conservación de materia alrededor de cada etapa.

$$\begin{aligned} M_{i,j} = {} & L_{j-1}x_{i,j-1} + V_{j+1}y_{i,j+1} \\ & + F_j z_{i,j} - (L_j + U_j)\, x_{i,j} - (V_j - W_j)\, y_{i,j} = 0 \end{aligned} \tag{2.1}$$

2. Las ecuaciones E modelan el equilibrio vapor-líquido, donde c es el número de componentes.

$$E_{i,j} = y_{i,j} - K_{i,j}x_{i,j} = 0 \tag{2.2}$$

3. Las ecuaciones S establecen los límites de las sumatorias de fracciones mol de cada componente.

$$(S_y)_j = \sum_{i=1}^{c} y_{i,j} - 1{,}0 = 0 \tag{2.3}$$

$$(S_x)_j = \sum_{i=1}^{c} x_{i,j} - 1{,}0 = 0 \tag{2.4}$$

4. Las ecuaciones H modelan el balance de energía en cada etapa.

$$H_j = L_{j-1}h_{L_{j-1}} + V_{j+1}h_{V_{j+1}} + F_j h_{F_j} \tag{2.5}$$

$$-\left(L_j + U_j\right)h_{L_j} - \left(V_j + W_j\right)h_{V_j} - Q_j = 0 \tag{2.6}$$

Es posible emplear una ecuación balance de materia total en lugar de (2.3) o (2.4). Se deriva mediante la combinación de estas dos ecuaciones y $\sum_j z_{i,j} = 1{,}0$ con (2.1) sumado sobre los componentes C desde la etapa 1 a j para dar:

$$L_j = V_{j+1} + \sum_{m=1}^{j}(F_m - U_m - W_m) - V_1 \tag{2.7}$$

Las ecuaciones anteriores describen totalmente una columna de destilación las cuales deben satisfacerse en cualquier estrategia de solución. Si se modifican las ecuaciones MESH de forma que en las ecuaciones M se sustituyen los $y_{i,j}$ por $K_{i,j}\,x_{i,j}$ y los L_j por su valor en función de W_j, U_j y V_j dado por el balance global de materia, se puede obtener la siguiente ecuación para cada componente y cada etapa a partir de los coeficientesA_j, B_j, C_j y D_j :

$$A_j x_{i,j-1} + B_j x_{i,j} + C_j x_{i,j+1} = D_j \tag{2.8}$$

donde:

$$A_j = V_j + \sum_{m=1}^{j-1}(F_m - U_m - W_m) - V_1,\ 2 \le j \le N \tag{2.9}$$

$$B_j = -\left[V_{j+1} + \sum_{m=1}^{j}(F_m - U_m - W_m) - V_1 + U_j + \left(V_j + W_j\right)K_{i,j}\right]$$
$$1 \le j \le N \tag{2.10}$$

$$C_j = V_{j+1}K_{i,j+1},\ 1 \le j \le N-1 \tag{2.11}$$

$$D_j = -F_j z_{i,j},\ 1 \le j \le N \tag{2.12}$$

Si las ecuaciones M modificadas (2.8) se agrupan por componentes, pueden escribirse como una serie de c sistemas de ecuaciones, uno para cada componente, en donde las matrices de coeficientes son matrices tridiagonales y la variable de salida para cada sistema es la composición x_i para ese componente en toda la cascada en contracorriente de N etapas:

$$\begin{bmatrix} B_1 & C_1 & \cdots & \cdots & \cdots & & \cdots & 0 \\ A_2 & B_2 & C_2 & \cdots & \cdots & & \cdots & 0 \\ \cdots & A_3 & B_3 & C_3 & \cdots & & \cdots & 0 \\ \cdots & \cdots & \ddots & \ddots & \ddots & & & \cdots \\ \cdots & & & \ddots & \ddots & \ddots & & \cdots \\ 0 & \cdots & \cdots & 0 & A_{N-2} & B_{N-2} & C_{N-2} & \cdots \\ 0 & \cdots & \cdots & 0 & 0 & A_{N-1} & B_{N-1} & C_{N-1} \\ 0 & \cdots & \cdots & 0 & 0 & & A_N & B_N \end{bmatrix} \times \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ x_{i,3} \\ \cdots \\ \cdots \\ x_{i,N-2} \\ x_{i,N-1} \\ x_{i,N} \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ \cdots \\ \cdots \\ D_{N-2} \\ D_{N-1} \\ D_N \end{bmatrix} \tag{2.13}$$

La solución eficiente de este conjunto de sistemas de ecuaciones se resuelve una para cada componente, mediante un algoritmo de eliminación Gaussiana modificado por Llewellyn H. Thomas mostrado con detalle en el apéndice A.1. Si se dispone de los valores de los coeficientes A_j, B_j, C_j y D_j en cada etapa, la resolución de los c sistemas de ecuaciones proporcionará el perfil de composición del líquido en la columna.

Por otra parte, las propiedades de los fluidos, son datos suministrados o predichos por algún método termodinámico que correlacionan funciones de la temperatura, la presión, y la composición. Expresado matemáticamente, los datos o correlaciones se pueden escribir de la siguiente manera:

$$K_{i,j} = K_{i,j}(T_j, P_j, \boldsymbol{x}_j, \boldsymbol{y}_j) \tag{2.14}$$

$$hV_j = hV_j(T_j, P_j, \boldsymbol{y}_j) \tag{2.15}$$

$$hL_j = hL_j(T_j, P_j, \boldsymbol{x}_j) \tag{2.16}$$

donde $\boldsymbol{x}_j$ y $\boldsymbol{y}_j$ son vectores de fracciones molares de los componentes en corrientes de la etapa j.

2.1.3. Grados de Libertad

Los grados de libertad de un sistema son las variables independientes que se deben especificar con el fin de definir el sistema por completo. El enfoque matemático para determinar los grados de libertad de cualquier sistema es reunir todas las variables y restar el número de ecuaciones independientes (Zereshki, 2012).

En este sentido, para el modelo de etapa de equilibrio, si las tres propiedades definidas en las ecuaciones 2.14 a 2.16 no se cuentan como ecuaciones ni como variables, cada etapa de equilibrio sólo se define por un total de $2C + 3$ ecuaciones. El sistema en cascada en contracorriente de N etapas, como se muestra en la Figura 2.2b, quedaría representado por $N(2C + 3)$ ecuaciones y $[N(3C + 10) + 1]$ variables.

Ahora, si se establecen N y todos los Fj, z_{ij}, T_{F_j}, P_{F_j}, P_j, U_j, W_j, Q_j correspondientes a $[N(C+7)+1]$ variables de entrada, el modelo generalizado de la columna queda representado por $N(2C+3)$ ecuaciones algebraicas simultáneas no lineales y $N(2C+3)$ variables desconocidas (de salida) que comprenden todas las variables $x_{i,j}$, $y_{i,j}$, L_j, V_j, T_j, considerándose así como las variables independientes. En tal caso que se especifiquen otras variables de entrada, las sustituciones correspondientes se deben de introducir en la lista de variables de salida. En cualquier caso, el resultado es un conjunto que contiene las ecuaciones no lineales que deben ser resueltas mediante una técnica iterativa (Seader et al., 2011; Khoury, 2005).

2.2. Métodos Rigurosos de Columnas de Destilación

Las ecuaciones que conforman el modelo de columna, incluyen muchas relaciones no lineales, y su número se puede encontrar alrededor de las decenas de miles. Por ejemplo, en una columna de 10 componentes, 30 etapas con dos especificaciones, el número de ecuaciones sería $(3 \times 10 + 5) \times 30 + 2 = 1052$).

Es evidente que la solución analítica de estas ecuaciones es imposible. La única manera práctica para resolver estos problemas de manera rigurosa es mediante algoritmos de solución numéricos implementados los simuladores de procesos químicos. (Khoury, 2005).

Históricamente, antes de la década de los años 50, los cálculos de columna se realizaban sin la ayuda de un computador. A pesar de que los procedimientos de cálculo rigurosos estaban disponibles, eran difíciles de aplicar para todas las columnas. Los denominados métodos aproximados o cortos basados el procedimiento clásico de Fenske-Underwood-Gilliland (FUG), eran por lo tanto la herramienta de diseño primario (Kister, 1992).

Los primeros intentos para la resolver el sistema de ecuaciones MESH fueron los métodos clásicos de cálculo etapa a etapa y ecuación a ecuación de Lewis-Matheson (1932) y Thiele-Geddes (1933), aplicables sólo a columnas simples o para chequear diseños finales. No obstante, con la introducción de las computadoras el procedimiento de diseño cambió totalmente. Los cálculos rigurosos, que alguna vez tomaron varios días, a veces semanas, incluso para una columna relativamente sencilla, ahora se realizan de forma rápida y eficaz mediante un ordenador minimizando las imprecisiones e incertidumbres inherentes a los procedimientos aproximados (King, 1980; Seader et al., 2011).

La mayor parte de los métodos modernos son métodos que utilizan procedimientos de separación y desacoplamiento del conjunto de ecuaciones MESH que describe a la columna para calcular las condiciones de su funcionamiento (Seader et al., 2011). En la práctica moderna de destilación, estos programas se han vuelto indispensables para la evaluación

del diseño y el rendimiento de los procesos de separación de etapas múltiples (Kister, 1992; Khoury, 2005).

Los métodos rigurosos para la solución de columnas siguen una estrategia de solución general que se muestra la Figura 2.3.

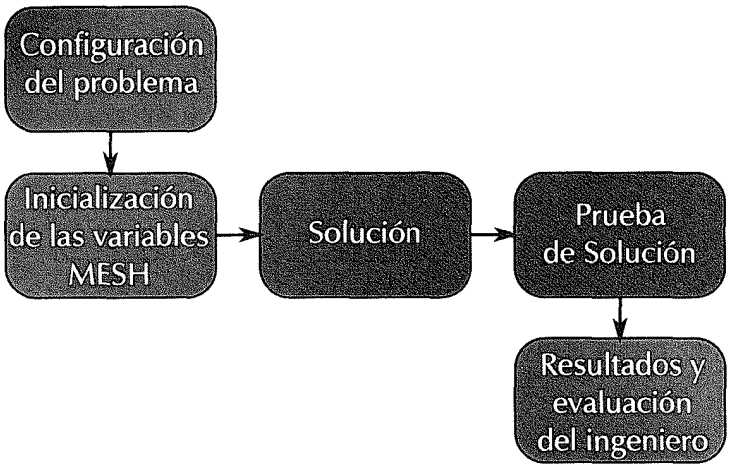


Figura 2.3: Pasos involucrados en un método riguroso (Kister, 1992).

En el primer paso de configuración del problema se especifican las variables de entrada y la especificación de la separación. King (1980); Kister (1992); Seader et al. (2011); McCabe et al. (2005) afirman que en un proceso de diseño, una separación se especifica en términos de grados de pureza y de los flujos de productos. Para una columna sencilla, por ejemplo, se deben establecer dos especificaciones y por lo menos una debe ser de pureza de producto. También afirma que el grado de pureza puede ser sustituido por una propiedad física que es una función de la pureza o composición, mientras que un flujo de producto puede ser sustituido por una especificación de recuperación.

En una simulación, se establece el número de etapas y la ubicación del punto de alimentación para una solución determinada. Una vez que estos se fijan, la composición de destilado (o del fondo) se convierte en una función del reflujo, por lo que el grado de pureza se cumple a través de una especificación de relación de reflujo. Dado que el reflujo es sólo uno de los flujos internos de la columna, un argumento similar se aplica a cualquier otro flujo de fase interna. De ello se desprende que el grado de pureza se puede cumplir mediante la especificación de cualquier flujo de fase interna, ya sea líquida o de vapor. Del mismo modo, otras variables que se pueden especificar incluyen la relación de reflujo a destilado, relación de *boilup* del fondo, el calor del intercambiador, y la temperatura de una etapa especifica (Kister, 1992).

Para una columna con productos laterales, el número de variables especificadas aumenta con cada producto lateral. En la mayoría de los métodos, la tasa de flujo del producto se

especifica para cada producto secundario, pudiéndose especificar la pureza de un producto lateral. Para las columnas con intercambiadores de calor en las etapas intermedias, el número de variables especificadas aumenta por el número de estos intercambiadores. Por lo general, para estos se especifican sus respectivos calores, pero en algunos métodos, estos calores se pueden permitir variar para cumplir una cierta especificación de producto. Para estas columnas complejas, las especificaciones inconsistentes son una de las principales causas de fallas en el cálculo por ello se prefieren especificaciones más simples.

Antes de que los cálculos principales comiencen, se debe dar valores iniciales a las temperaturas de las etapas, T_j y flujos totales, V_j y L_j . Los flujos por componentes de las etapas, v_{ij} y l_{ij} no tienen que ser estimados ya que estos se pueden calcular a partir de los balances de componentes. Pero los balances de componentes dependen de los valores K. En este sentido para los primeros balances de componentes, se deben utilizar valores K independientes de la composición (Wilson et al., 2000).

Con cualquier método riguroso es imprescindible establecer criterios para determinar cuando se alcanza una solución. Cada método riguroso tiene sus propios criterios, pero todos ellos deben cumplir con ciertos criterios físicos. Además de cumplir con los balances de materia y de energía (Kister, 1992; Seader et al., 2011). Existen otros criterios que un método debe cumplir. Si hay especificaciones, tales como velocidades de flujo de productos, flujos de calor, o composiciones de productos, éstos se deben establecer dentro de una cierta tolerancia. En la solución, los perfiles de la velocidad de flujo y temperatura total no deberían cambiar entre iteraciones, es decir, el cambio fraccional acumulado en temperaturas de etapa y en las tasas de vapores de la fase entre iteraciones continuas k y $k+1$ debe ser menor a una tolerancia preestablecida ϵ, según el criterio planteado por Wang & Henke (1966) expresado en la ecuación 2.17.

$$\sum_{j=1}^{N} \frac{|T_{j,k+1} - T_{j,k}|^2}{T_{j,k}} + \sum_{j=1}^{N} \frac{|V_{j,k+1} - V_{j,k}|^2}{V_{j,k}} \leq \epsilon \tag{2.17}$$

Los errores en las ecuaciones MESH deben ser pequeños, como por ejemplo hacer que las funciones tiendan a cero con el método de Newton-Raphson, el cual se describe en el apéndice A.2.

2.2.1. Clasificación de Los Métodos Rigurosos

Se han propuesto muchos algoritmos para la resolución de las ecuaciones, una buena revisión de los cuales se presenta por Wang et al. (1980). Las diferencias entre los métodos derivan de varias consideraciones, incluyendo la elección de las variables independientes,

la agrupación y disposición de las ecuaciones y la trayectoria de convergencia. Los métodos rigurosos se pueden dividir en dos clases como se muestra en la Tabla 2.1.

Tabla 2.1: Clasificación de los Métodos Rigurosos de Resolución (King, 1980).

Métodos Básicos	**Método Extendidos**
Los Métodos de Punto de Burbuja (BP)	Los Métodos de Relajación
Los Métodos de Suma de Caudales (SR)	Los Métodos *Inside-Out*
Los Métodos Newton 2N	Los Métodos Homotópicos o de Continuación
Los Métodos de Correcciones Simultáneas (SC)	Los Modelos Fuera del Equilibrio

- **Los Métodos de Punto de Burbuja (BP)**

  Los métodos BP reciben su nombre debido a que las temperaturas de las etapas se determinan resolviendo directamente la ecuación de punto de burbuja. Los métodos de BP por lo general funcionan mejor para sistemas con componentes de estrecho punto de ebullición, donde la composición tiene un efecto mayor sobre la temperatura que el calor latente de vaporización. Todas las ecuaciones se particionan y se resuelven de manera secuencial a excepción de las ecuaciones M, que se resuelven por separado para cada componente por la técnica de matriz tridiagonal (Seader et al., 2011).

  El primer método de BP propuesto por Wang & Henke (1966) ilustrado en la Figura 2.4, contempló por primera vez el método matriz tridiagonal para calcular las tasas de flujo de componentes o composiciones, calculando además las nuevas las temperaturas de las etapas directamente, sin necesidad de iterar con la ecuación del punto de burbuja. Aquí las temperaturas son aproximadas pero siempre y cuando las composiciones internas se corrijan adecuadamente durante cada iteración, el perfil de temperatura convergerá hacia la solución (Khoury, 2005; Kister, 1992).

  En este método se consideran variables de iteración a $x_{i,j}$, $y_{i,j}$, T_j, L_j, y V_j. Para iniciar los cálculos se asumen valores de las variables de iteración, V_j y T_j. En general, es suficiente establecer un conjunto inicial de valores de V_j el reflujo especificado, destilado, alimentación, y los flujos de corriente secundaria. Los valores T_j iniciales se pueden estimar mediante el cálculo de la temperatura del punto de burbuja de un producto de fondo y la temperatura punto de rocío de un producto destilado asumido, y luego utilizar interpolación lineal para determinar otras temperaturas de las etapas (Seader et al., 2011).

Para obtener x_i a partir de (2.13) por el método de Thomas (A.1) son necesarios los valores de K_{ij}. Por tanto, cuando éstos dependen de la composición, se necesitan también suposiciones iniciales para todos los $x_{i,j}$ y $y_{i,j}$, a no ser que se utilicen en la primera iteración los valores ideales de las $K_{i,j}$. Para cada iteración, el conjunto de valores calculados $x_{i,j}$ de cada etapa, por lo general, no satisfacen la restricción impuesta por las ecuaciones S (2.3 y 2.4) y es aconsejable normalizar el conjunto de valores calculados $x_{i,j}$ mediante:

$$(x_{i,j})_{normalizado} = \frac{x_{i,j}}{\sum_{i=1}^{c} x_{i,j} \neq 1} \tag{2.18}$$

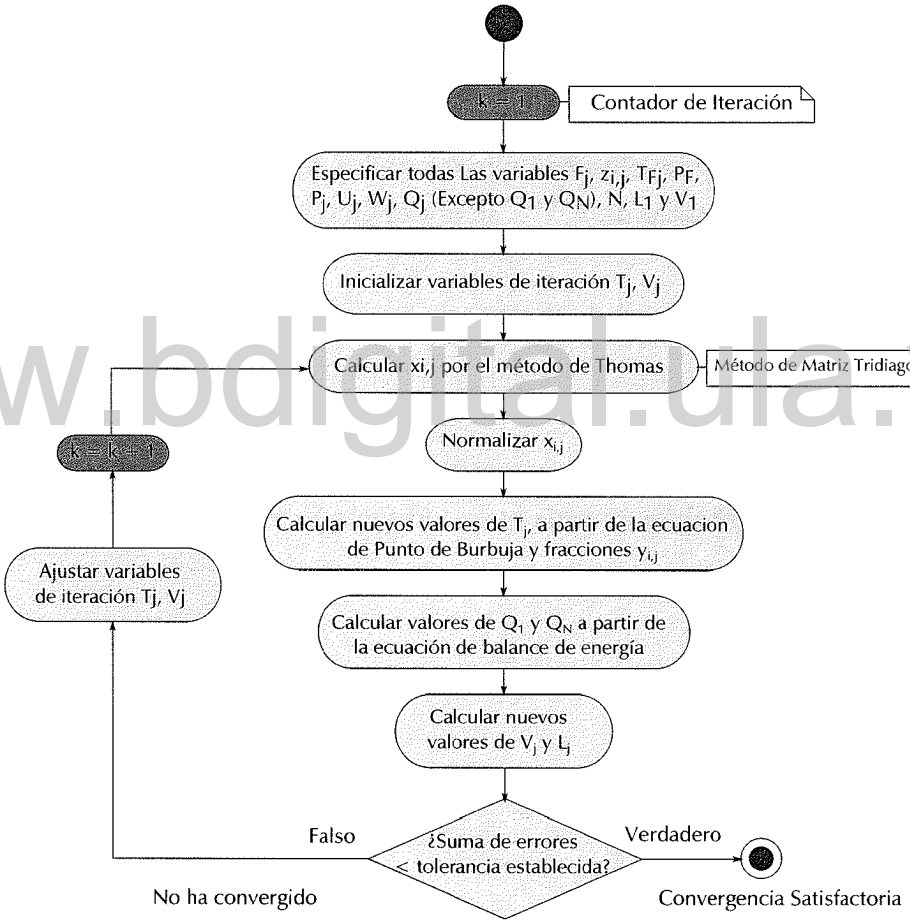


Figura 2.4: Algoritmo de Método BP por Wang–Henke (Seader et al., 2011)

- **Los Métodos de Suma de Caudales (SR)**

  Son métodos adecuados para el modelado de absorbedores y despojadores donde las especies poseen una amplia diferencia de puntos de ebullición, especialmente

aquellos con especies no condensables. Los métodos BP fallan en estos casos porque los cálculos del punto de burbuja de temperatura son demasiado sensibles a la composición en fase líquida, y el balance de energía es mucho más sensible a las temperaturas de las etapas que a las flujos entre etapas. El primer método SR fue desarrollado por Burningham & Otto (1967) en conjunción con la formulación de de matriz-tridiagonal para un conjunto de ecuaciones M modificadas.

Las variables se mantienen iguales al método BP y los valores de $x_{i,j}$ se obtienen resolviendo mediante el algoritmo de Thomas. Solo que en este caso los valores $x_{i,j}$ no se normalizan y se utilizan directamente para calcular nuevos valores de L_j mediante la ecuación de suma de caudales:

$$L_j^{(k+1)} = L_j^{(k)} \sum_{i=1}^{c} x_{i,j} \tag{2.19}$$

donde los valores de $L_j^{(k)}$ se obtienen a partir de los valores $V_j^{(k)}$ de la ecuación (2.7), los valores correspondientes de $V_j^{(k+1)}$ se obtienen a partir de un balance de materia global:

$$V_j = L_{j-1} - L_N + \sum_{m=j}^{N} (F_m - W_m - U_m) \tag{2.20}$$

Finalmente, el perfil de temperatura para la siguiente iteración se obtiene por resolución del balance de entalpía en cada etapa. El método se ilustra en la Figura 2.5.

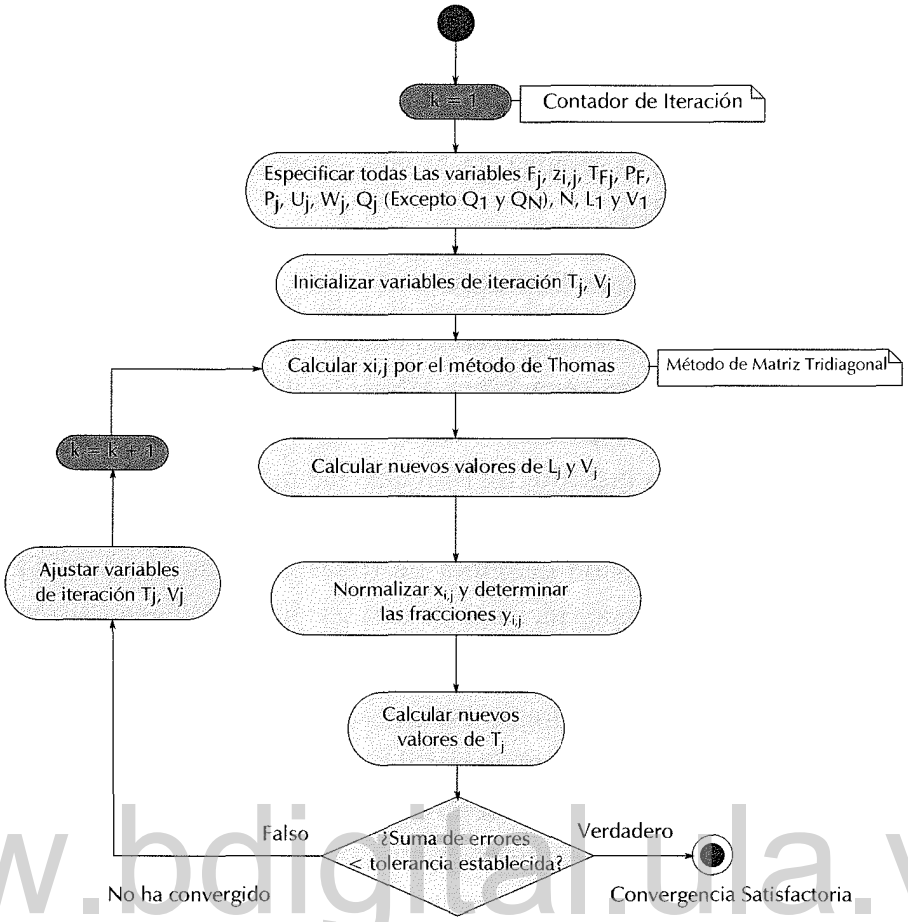


Figura 2.5: Algoritmo del método SR de Burningham & Otto (Seader et al., 2011)

- **Los Métodos de Newton** $2N$

  Tanto en los métodos BP como en los SR, el perfil de temperatura y el perfil de caudales de vapor y de líquido se calculan en pasos separados. En los métodos de Newton 2N el nombre $2N$ significa que hay dos ecuaciones por etapa para un total de $2 \times N$ funciones y variables por columna que se resuelven simultáneamente por un método de Newton Raphson. Los métodos de Newton 2N han demostrado que funcionan bien para las mezclas de amplio punto de ebullición incluyendo fraccionadores de refinería, columnas de absorción-despojamiento y absorbedores con rehervidor. Las variables de iteración son $L_{i,j}$, $y_{i,j}$, y T_j. Entre los métodos más conocidos se encuentran el método desarrollado por Tomich (1970).

  Este Método difiere de los métodos anteriores, principalmente en que las ecuaciones sumatoria y los balances de energía se resuelven simultáneamente. Por otra parte presenta dos importantes beneficios. En primer lugar, es aplicable para resolver columnas de destilación, columnas absorbedoras y columnas híbridas de ambos tipos

de procesos. En segundo lugar, se pueden incorporar en la solución simultánea de las ecuaciones diferentes tipos de especificaciones de rendimiento de la columna. El método también es computacionalmente estable y eficiente, ya que utiliza la modificación de Broyden de la técnica de Newton-Raphson para la resolución de las ecuaciones (Broyden, 1965).

- **Los Métodos de Correcciones Simultáneas (SC)**

  Las tres clases anteriores se catalogan como métodos basados en desacoplamiento de ecuaciones, debido a que las ecuaciones MESH se dividen, se agrupan y se combinan con las variables MESH para ser resueltos en una serie de pasos. Los Métodos SC denominados también métodos de Newton Globales o métodos NR, son capaces de resolver muchos problemas de separaciones multicomponente de etapa múltiple. Están basados en la resolución conjunta de todas las ecuaciones MESH o de una combinación de las mismas por técnicas de corrección simultánea (Kister, 1992).

  En general, los métodos SC resultan más convenientes que los anteriores ya que en sistemas altamente no ideales, donde los valores de las relaciones de equilibrio especialmente los valores de los coeficientes de actividad y de las entalpías dependen fuertemente de la composición, no resulta demasiado adecuado calcular las composiciones a partir de valores de constantes de equilibrio y de entalpías procedentes de la iteración anterior. Estos métodos son los más sensibles a la calidad de las estimaciones iniciales, pero también son más poderosos a la hora de tratar mezclas no ideales, de forma que en ocasiones se requiere aplicar primero otro método riguroso (BP o SR) para encontrar aproximaciones a la solución final que sirvan como estimaciones iniciales para el método SC (Khoury, 2005). Uno de los métodos SC más conocidos es el desarrollado por Naphtali & Sandholm (1971).

  El modelo de etapas de las Figuras 2.2a y 2.2b se emplean nuevamente. Sin embargo, en lugar de resolver las $N\,(2C+3)$ ecuaciones MESH de manera simultánea. Se combinan las ecuaciones (2.3) y (2.4) junto con las otras ecuaciones para eliminar $2N$ variables y de esta manera reducir el problema a una solución simultánea de $N(2C-1)$ ecuaciones. Esto se realiza multiplicando (2.3) y (2.4) porV_j y L_j , respectivamente para dar:

$$V_j = \sum_{i=1}^{c} v_{i,j} \tag{2.21}$$

$$L_j = \sum_{i=1}^{c} l_{i,j} \tag{2.22}$$

donde se usan las siguientes definiciones de fracciones mol:

$$y_{i,j} = \frac{v_{i,j}}{V_j} \tag{2.23}$$

$$x_{i,j} = \frac{l_{i,j}}{L_j} \tag{2.24}$$

Las ecuaciones (2.21) a (2.24) ahora se sustituyen en (2.1), (2.2) y (2.6) para eliminar V_j, L_j, $y_{i,j}$ y $x_{i,j}$ e introducir flujos de componentes $v_{i,j}$ y $l_{i,j}$, obteniéndose las siguientes ecuaciones, donde $s_j = U_j/L_J$ y $S_j = W_j/V_J$ son flujos laterales adimensionales y $f_{i,j} = F_j z_{i,j}$.

$$M_{i,j} = l_{i,j}\,(1 + s_j) + v_{i,j}\,(1 + S_j) - l_{i,j-1} - v_{i,j+1} - f_{i,j} = 0 \tag{2.25}$$

$$E_{i,j} = K_{i,j} l_{i,j} \frac{\sum_{k=1}^{c} v_{k,j}}{\sum_{k=1}^{c} l_{k,j}} - v_{i,j} = 0 \tag{2.26}$$

$$\begin{aligned} H_j = {} & h_{L_j}\,(1 + s_j) \sum_{i=1}^{c} l_{i,j} + h_{V_j}\,(1 + S_j) \sum_{i=1}^{c} v_{i,j} \\ & - h_{L_{j-1}} \sum_{i=1}^{c} l_{i,j-1} - h_{V_{j+1}} \sum_{i=1}^{c} v_{i,j+1} \\ & - h_{F_j} \sum_{i=1}^{c} f_{i,j} - Q_j = 0 \end{aligned} \tag{2.27}$$

Las ecuaciones (2.25), (2.26) y (2.27) se resuelven simultáneamente por el método de Newton-Raphson en el que los conjuntos sucesivos de las variables de salida se calculan de manera iterativa hasta que los valores de las funciones M, E, y H son conducidas hacia los criterios de convergencia o cero. Durante las iteraciones, los valores distintos de cero de las funciones se llaman discrepancias o errores. Si las funciones y variables de salida se agrupan por etapas en orden de arriba a abajo, se forma una estructura de bloque-tridiagonal de la matriz Jacobiana de derivadas

parciales de modo que se puede aplicar una forma de la matriz del algoritmo de Thomas (Seader et al., 2011).

Los métodos extendidos que a continuación se explican, son una ampliación de los primeros cuatro métodos con el fin de aumentar la gama de aplicación de columnas, para resolver sistemas difíciles o columnas más complejas.

- **Los Métodos de Relajación**

  Un método de relajación encuentra una solución de estado estacionario de una columna como si se tratara de una columna que opera en estado dinámico. La columna se inicializa utilizando condiciones reales de operación de puesta en marcha. Se parte de la suposición de valores iniciales para los caudales inter-etapa y para las temperaturas y composiciones en cada etapa. La columna se lleva a las condiciones de estado estacionario por aproximaciones sucesivas de las ecuaciones de estado inestable de destilación. Estas ecuaciones de estado inestable son modificaciones a las ecuaciones MESH para incluir cambios en las variables MESH con respecto al tiempo. Este proceso podría decirse que está imitando la puesta en marcha física de la columna, pero el objetivo no es conseguir la operación dinámica sino buscar la solución en estado estacionario(Khoury, 2005).

- **Los Métodos Homotópicos o de continuación**

  Se basan en un procedimiento de resolución de sistemas de ecuaciones algebraicas no lineales. Generalmente requiere cálculos algo sofisticados, pero tiene la ventaja de que permite encontrar soluciones incluso cuando los valores iniciales son deficientes. Para algunos casos las ecuaciones de las MESH pueden ser difíciles de resolver, ya sea debido a la naturaleza de la columna (muchas alimentaciones o retiros laterales, separadores laterales, operación cerca de reflujo-mínimo, etc.) o debido a las no idealidades de los valores de K o entalpías.

  Los métodos homotópicos comienzan con una solución conocida de la columna y desde allí se sigue un camino a la solución deseada. La solución conocida puede ser en diferentes condiciones o con métodos mucho más simple de valores K y entalpía y los cambios escalonados se realizan a partir de ahí, resolviendo la columna en cada paso, hasta que se alcanza la solución final (Kister, 1992).

  Las métodos homotópicos se pueden dividir en dos clases generales, los basados en homotopías matemáticas y en homotopías físicas o paramétricas. Las homotopías matemáticas no posee relación física con las ecuaciones MESH y esto en ocasiones causa

problemas. Las homotopías físicas tienen una base en las ecuaciones MESH, por lo que son más aplicadas. (Wayburn, 1988) presenta una revisión de los métodos homotópicos así como de sus diferentes aplicaciones.

- **Los Modelos de No-Equilibrio**

  También denominados métodos basados de velocidades de transferencia de masa. Son métodos que eliminan por completo el concepto de etapa de equilibrio y el uso del concepto de eficiencia de etapa. Ellos son los más adecuados para las columnas donde es difícil de definir una etapa teórica y las eficiencias resultan difíciles de predecir o aplicar por cualquier medio. Por ejemplo, en sistemas altamente no ideales y sistemas reactivos. Aplican un enfoque de fenómenos de transporte para predecir las tasas de transferencia de masa. Las tasas de transferencia de masa se calculan continuamente a lo largo de la longitud de la columna y no en etapas de equilibrio discretas (Kooijman & Taylor, 2000; Krishnamurthy & Taylor, 1985).

  Krishnamurthy & Taylor (1985) presentan y colocan a prueba un modelo de no equilibrio que incluye ecuaciones de velocidad de transferencia de masa con la posibilidad de incluir reacciones químicas, entre las ecuaciones MESH tradicionales. Estos incluyen balances de masa y energía en el vapor y el líquido y a través de una interfase. Existe una ecuación de equilibrio para la interfase única. Los métodos de solución para estas ecuaciones es la misma que los métodos globales Newton y el estilo del método es similar al de Naphtali & Sandholm.

2.3. Método INSIDE-OUT de Russell (1983)

En los métodos de BP, SR, y SC, el principal esfuerzo de cálculo se consume en el cálculo de valores K y las entalpías cuando se utilizan modelos termodinámicos rigurosos, ya que los cálculos de propiedades termodinámicas se efectúan en cada iteración. Además, en cada iteración, se necesitan las derivadas de:

1. Todas las propiedades con respecto a la temperatura y las composiciones de ambas fases en el método Newton-Raphson

2. Los valores K con respecto a la temperatura para el método de BP.

3. Vapor y entalpías de líquidos con respecto a la temperatura para el método de SR.

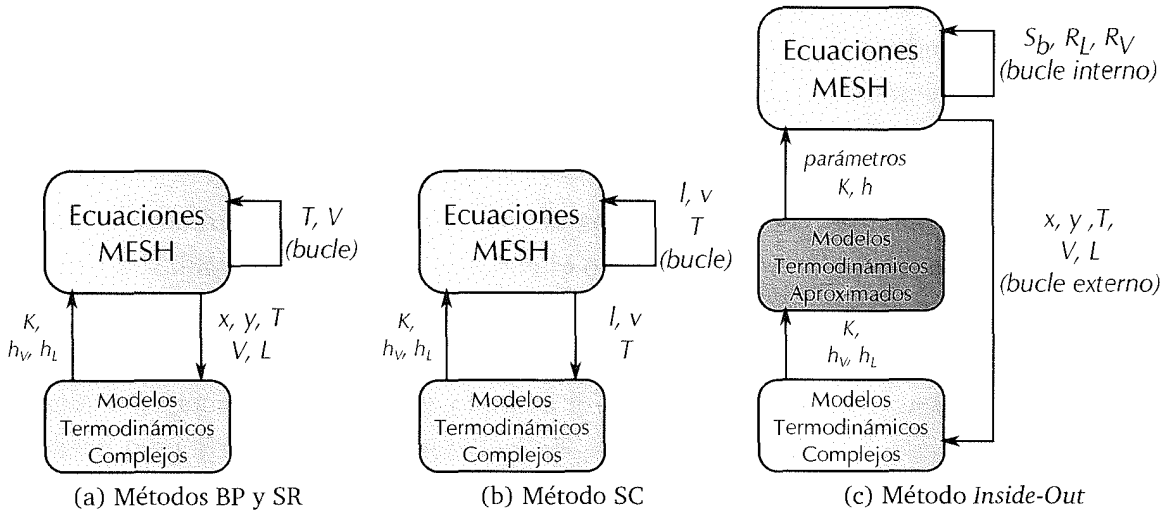


Figura 2.6: Incorporación de correlaciones termodinámicas en bucles interactivos (Seader et al., 2011)

El Método *Inside-Out* se ha desarrollado con tres objetivos primarios: ser flexible en el manejo de las especificaciones de rendimiento, requerir poca información para generar estimaciones iniciales y poseer alta velocidad de cálculo. Las técnicas empleadas para cumplir con estos objetivos están incorporadas en un método original descrito por Russell (1983). Este método, basado en el artículo publicado por Boston & Sullivan (1972), está formado por un algoritmo de dos bucles, un bucle interno donde las composiciones y flujos de columna de platos y perfiles de temperatura se calculan sobre la base de un modelo de valores K y cálculos de entalpía simplificado y un bucle externo, donde estas propiedades se calculan a partir de modelos termodinámicos rigurosos.

El método es rápido debido a que los cálculos termodinámicos rigurosos son relegados al bucle externo, donde no se realizan con frecuencia. El término *INSIDE-OUT* se aplica comúnmente a este tipo de método debido a la estrategia de limitar los cálculos rigurosos que consumen mucho tiempo termodinámicos en el bucle externo. Cabe señalar que la solución general es rigurosa ya que se considera que el sistema ha convergido cuando los valores K y entalpías utilizados en el bucle interno están en concordancia con los calculados rigurosamente en el bucle externo (Kister, 1992; Seader et al., 2011).

En cada paso a través del bucle exterior, los valores K y entalpía de los modelos simples y se actualizan mediante el uso de las variables MESH a partir del bucle interior. Esto establece la siguiente iteración por el bucle interior (Wilson et al., 2000).

Otra de las características del método *Inside-Out* es las variables de iteración que se utilizan. Para este método las variables de iteración para el bucle externo son los parámetros en las ecuaciones aproximadas para las propiedades termodinámicas. Las variables de iteración del bucle interno están relacionadas con los factores de despojamiento $S_{i,j} = {}^{K_{i,j}V_j}/_{L_j}$ y Factores de Retiro Lateral R_{Lj} y R_{Vj}.

El método *Inside-Out* toma ventaja de las siguientes características de los cálculos iterativos:

1. Las volatilidades relativas de componentes varían mucho menos que sus valores K.
2. La entalpía de vaporización varía menos que las entalpías de fase.
3. Los factores de despojamiento de componentes combinan efectos de la temperatura y los flujos de líquido y vapor en cada etapa.
4. El bucle interno del método utiliza volatilidad relativa, la entalpía, y los factores de despojamiento para mejorar la estabilidad y reducir el tiempo de computo.

2.3.1. Ecuaciones MESH para el método *Inside-Out*.

Al igual que con los métodos BP, SR, y SC, se emplea el modelo de equilibrio-etapa de la Figura (2.2). La forma de las ecuaciones es similar al método de SC en el que se utilizan velocidades de flujo de los componentes. Sin embargo, adicionalmente se definen las siguientes variables de iteración para el Bucle Interno:

$$\alpha_{i,j} = {}^{K_{i,j}}/_{K_{b,j}} \tag{2.28}$$

$$S_{b,j} = {}^{K_{b,j}V_j}/_{L_j} \tag{2.29}$$

$$R_{L_j} = 1 + {}^{U_j}/_{L_j} \tag{2.30}$$

$$R_{V_j} = 1 + {}^{W_j}/_{V_j} \tag{2.31}$$

donde K_b es el valor K para un componente hipotético base o de referencia, $S_{b,j}$ es el factor de despojamiento del componente base, R_{L_j} y R_{V_j} son factores de retiro de fase líquida y vapor respectivamente, así para las etapas que no posean corrientes laterales, R_{L_j} y R_{V_j} toman el valor de 1. Las variables previamente definidas (2.21) a (2.24) aún se aplican, convirtiendo las ecuaciones MESH en:

Ecuación M por componente c:

$$l_{i,j-1} - \left(R_{L_j} + \alpha_{i,j} S_{b,j} R_{V_j}\right) l_{i,j} + \left(\alpha_{i,j+1} S_{b,j+1}\right) l_{i,j+1} = -f_{i,j}, i = 1\ a\ c,\ j = 1\ a\ N \tag{2.32}$$

Ecuación E:

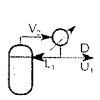
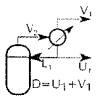
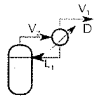
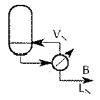
$$v_{i,j} = \alpha_{i,j} S_{b,j} l_{i,j},\ i = 1\ a\ c,\ j = 1\ a\ N \tag{2.33}$$

Ecuación H:

$$H_j = h_{L_j} R_{L_j} L_j + h_{V_j} R_{V_j} V_j - h_{L_{j-1}} L_{j-1} - h_{V_{j+1}} V_{j+1} - h_{F_j} F_j - Q_j = 1\ a\ c,\ j = 1\ a\ N \tag{2.34}$$

Además se pueden añadir a estas ecuaciones las funciones de discrepancia. Es conveniente especificar variables de etapas superiores e inferiores en lugar de los calores de condensador y/o rehervidor, que son tan interdependientes que no se recomienda la especificación de ambos valores. La especificación de otras variables se lleva a cabo mediante la eliminación de la función de balance de energía H_1 y/o H_N y reemplazándolos con funciones discrepancia como se muestran en la Tabla 2.2.

Tabla 2.2: Funciones Alternas para H_1 y H_N

Especificación		Reemplazo por H_1			Reemplazo por H_N
		Condensador Total	Condensador Parcial	Condensador Reflujo Total	Rehervidor Parcial
					
Flujo de producto	Másica	$\dot{U}_1 - \dot{D} = 0$		$\dot{V}_1 - \dot{D} = 0$	$\dot{L}_N - \dot{B} = 0$
	Molar	$U_1 - D = 0$		$V_1 - D = 0$	$L_N - B = 0$
Fracción de Componente	Másica	$\dot{u}_{i,1} - (U_1)\dot{x}_{i,U_1} = 0$		$\dot{v}_{i,1} - (V_1)\,\dot{y}_{i,D} = 0$	$\dot{l}_{i,N} - \left(\dot{L}_N\right)\dot{x}_{i,B} = 0$
	Molar	$u_{i,1} - (U_1)\,x_{i,U_1} = 0$		$v_{i,1} - (V_1)\,y_{i,D} = 0$	$l_{i,N} - (L_N)\,x_{i,B} = 0$
Flujo de componente	Másica	$\dot{u}_{i,1} - \dot{d}_{i,U_1} = 0$		$\dot{v}_{i,1} - \dot{d}_{i,D} = 0$	$\dot{l}_{i,N} - \dot{b}_{i,B} = 0$
	Molar	$u_{i,1} - d_{i,U_1} = 0$		$v_{i,1} - d_{i,D} = 0$	$l_{i,N} - b_{i,B} = 0$
Fracción de Recuperación	Másico	$\sum_{j=1}^{N} \dot{f}_{i,j} - \left(\dot{U}_1\right)\dot{x}_{i,U_1} = 0$		$\sum_{j=1}^{N} \dot{f}_{i,j} - \left(\dot{V}_1\right)\dot{y}_{i,D} = 0$	$\sum_{j=1}^{N} \dot{f}_{i,j} - \left(\dot{L}_N\right)\dot{x}_{i,B} = 0$
	Molar	$\sum_{j=1}^{N} f_{i,j} - (U_1)x_{i,U_1} = 0$		$\sum_{j=1}^{N} f_{i,j} - (V_1)y_{i,D} = 0$	$\sum_{j=1}^{N} f_{i,j} - (L_N)x_{i,B} = 0$
Velocidad de Reflujo/*Boilup*	Másica	$\dot{L}_1 - \dot{D} = 0$			$\dot{V}_N - \dot{B} = 0$
	Molar	$L_1 - D = 0$			$V_N - B = 0$
Relación de Reflujo/*Boilup*	Másica	$\dot{L}_1 - \left(\dot{L}/\dot{D}\right)\dot{U}_1 = 0$		$\dot{L}_1 - \left(\dot{L}/\dot{D}\right)\dot{V}_1 = 0$	$\dot{V}_N - \left(\dot{V}/\dot{B}\right)\dot{B} = 0$
	Molar	$L_1 - (L/D)U_1 = 0$		$L_1 - (L/D)V_1 = 0$	$V_N - (V/B)B = 0$
Temperatura de Etapa		$T_1 - T_D = 0$			$T_N - T_B = 0$

2.3.2. Modelos aproximados de propiedades termodinámicas

Son los que se utilizan para calcular las constantes de equilibrio y las entalpías en el bucle interno. Los parámetros de estos modelos se determinan en el bucle externo, a partir de los valores proporcionados por modelos rigurosos. Los modelos aproximados en el método están diseñados para facilitar el cálculo de las temperaturas de la etapa y los factores de despojamiento.

- **Cálculo de los valores K**

  El modelo de valores K aproximado utilizado en el método de Russell (1983), difiere ligeramente del método original de Boston & Sullivan (1974). Se basa en seleccionar un componente de referencia b, que se va a considerar para el cálculo de las constantes de equilibrio a partir de la expresión:

$$K_{b,j} = \exp\left(A_j - \frac{B_j}{T_j}\right) \tag{2.35}$$

  Se puede seleccionar como referencia b cualquier componente de la alimentación u otro componente hipotético, a partir de una ponderación de vapor-composición usando las siguientes relaciones:

$$K_{b,j} = \exp\left(\sum_i w_{i,j} \ln K_{i,j}\right) \tag{2.36}$$

$$w_{i,j} = \frac{y_{i,j}\left(\partial \ln K_{i,j}/\partial\left(1/T\right)\right)}{\sum_i y_{i,j}\left(\partial \ln K_{i,j}/\partial\left(1/T\right)\right)} \tag{2.37}$$

Para cada etapa j se determina un modelo K_b y valores de $\alpha_{i,j}$ a partir de los valores $K_{i,j}$ arrojados por los modelos rigurosos. En la etapa de superior, el componente base representaría un componente cercano a un componente ligero, mientras que en la etapa de fondo, el componente base representaría a un compuesto cercano a uno pesado.

Para obtener los valores de Aj y Bj en (2.35), dos temperaturas deben ser seleccionadas para cada etapa. Por ejemplo, se podrían seleccionar las temperaturas estimadas o reales, de las dos etapas adyacentes, $j-1$ y $j+1$, . Llamando a estas T_1 y T_2 y empleando (2.35) en cada etapa j:

$$B_j = \frac{\ln\left(K_{b_{T1}}/K_{b_{T2}}\right)}{\left(\frac{1}{T_2} - \frac{1}{T_1}\right)} \tag{2.38}$$

$$A_j = \ln K_{b_{T1}} + B/T_1 \tag{2.39}$$

- **Cálculo de entalpías**

  Russell (1983) emplea el mismo modelo de entalpía de Boston & Sullivan (1972). Así, tanto para la fase vapor como para la fase líquida se tiene la expresión:

$$h = h_V^\circ + \left(h - h_V^\circ\right) = h_V^\circ - \Delta H \tag{2.40}$$

  donde h_V° es la entalpía de mezcla de gases ideales.

  Las partes que consumen mucho tiempo de los cálculos de entalpía son los dos términos de entalpía de partición ΔH, tornándose complejas cuando se emplea una ecuación de estado. Por lo tanto, en las ecuaciones aproximadas de entalpía, la entalpía de partición se sustituyen por las funciones lineales simples:

$$\Delta H_{V_j} = c_j - d_j\left(T_j - T^*\right) \tag{2.41}$$

$$\Delta H_{L_j} = e_j - f_j\,(T_j - T^*) \tag{2.42}$$

donde T^* es una temperatura de referencia y los parámetros c, d, e, y f se evalúan a partir de los modelos rigurosos en cada iteración de bucle exterior.

$$d_j = (H_{V_{j1}} - H_{V_{j2}})/(T_{j1} - T_{j2}) \tag{2.43}$$

$$c_j = H_{V_{j1}} - d_j\,(T_{j1} - T_j) \tag{2.44}$$

$$f_j = (H_{L_{j1}} - H_{L_{j2}})/(T_{j1} - T_{j2}) \tag{2.45}$$

$$e_j = H_{L_{j1}} - f_j\,(T_{j1} - T_j) \tag{2.46}$$

2.3.3. Algoritmo *Inside-Out*

El algoritmo consiste en un procedimiento de inicialización, las iteraciones de bucle interno y las iteraciones del bucle externo. Se muestra en la Figura 2.8 su diagrama de flujo general.

- **Procedimiento de Inicialización**

  En primer lugar, es necesario proporcionar estimaciones razonables de las variables P_j, T_j,V_j,L_j, $x_{i,j}$ y $y_{i,j}$,para cada etapa

  1. Definir el número de etapas teóricas N, las condiciones de todas las alimentaciones (F_j, $z_{i,j}$, P_{F_j}, T_{F_j}), ubicación de alimentaciones en la columna, y el perfil de presión.
  2. Establecer la ubicación para cada corriente lateral en cada etapa y para cada intercambiador de calor.
  3. Proporcionar una especificación adicional para el producto de tope y fondo y cada intercambiador de calor intermedio.
  4. Si no se especifica, estimar el flujo de cada corriente producto y/o lateral establecidos, y estimar cada valor de V_j. Obtener valores de L_j de la ecuación total de balance de materiales 2.20.

5. Estimar un perfil de temperatura inicial, T_j, esto se realizando partiendo de un cálculo de punto de burbuja y rocío a la presión promedio de la columna de una *alimentación compuesta* que se obtiene combinando todas las corrientes de alimentación. La temperatura de burbuja se establece como la temperatura de tope T_1, mientras que la temperatura de rocío como la temperatura de fondo T_N. Luego por interpolación obtener las temperaturas de las etapas intermedias. Las temperaturas de referencia T^* adoptan los mismos valores de T_j para ser empleadas en (2.41) y (2.42).
6. Realizar un cálculo *Flash* isotérmico a la alimentación compuesta a la temperatura y presión promedio de la columna. Las composiciones del vapor y líquido resultantes x_i y y_i se establecen como las composiciones iniciales de las etapas.
7. Con los estimados iniciales de los primeros 6 pasos, emplear un modelo termodinámico formal para determinar los valores de los parámetros K y h que se emplean para la estimación de los parámetros del modelo termodinámico aproximado Aj,Bj, c_j, d_j, e_j, f_j, $K_{b,j}$ y $\alpha_{i,j}$ del bucle interno.
8. Determinar los valores iniciales de $S_{b,j}$, R_{Lj}, y R_{Vj} a partir de (2.29), (2.30) y (2.31).

- **Secuencia de cálculo del Bucle Interno**

A partir de este paso se da comienzo a la secuencia iterativa de los cálculos de bucle interno, con base a los valores de los parámetros enumerados en el paso 7, obtenidos inicialmente a partir del procedimiento de inicialización y después de los cálculos de los parámetros K y h a partir del modelo termodinámico riguroso.

9. Calcular los flujos de líquido por componentes $l_{i,j}$, a partir de las N ecuaciones (2.32) para cada uno de los componentes mediante el algoritmo de matriz tridiagonal (A.1).
10. Determinar $v_{i,j}$ a partir de la ecuación (2.33).
11. Evaluar nuevos valores de flujo para V_j y L_j, a partir de los flujos de componentes ya determinados en 9 y 10, utilizando las ecuaciones (2.21) y (2.22).
12. Para calcular un conjunto revisado de temperaturas etapa, T_j, calcular un conjunto de valores x_i para cada etapa a partir de 2.24. Luego un conjunto revisado de valores $K_{b,j}$ de una combinación de la ecuación de punto de burbuja ($\sum_i z_i K_i = 1$) con (2.28), lo que resulta:

$$K_{b,j} = {}^{1}/\sum_{i=1}^{c}(\alpha_{i,j}x_{i,j}) \tag{2.47}$$

A partir de estos nuevos valores de $K_{b,j}$ se calcula un grupo de temperatura de etapas a partir de un rearreglo de (2.35):

$$T_j = \frac{B_j}{A_j - \ln K_{b,j}} \tag{2.48}$$

En este punto del bucle interno, existe un conjunto revisado de $v_{i,j}$, $l_{i,j}$, y T_j, que satisfacen las ecuaciones de balance de materia por componente y balance de energía para las propiedades estimadas. No obstante, estos valores no satisfacen las ecuaciones de balance de energía y especificaciones al menos que los factores de despojamiento y las velocidades de retiros laterales estén correctas.

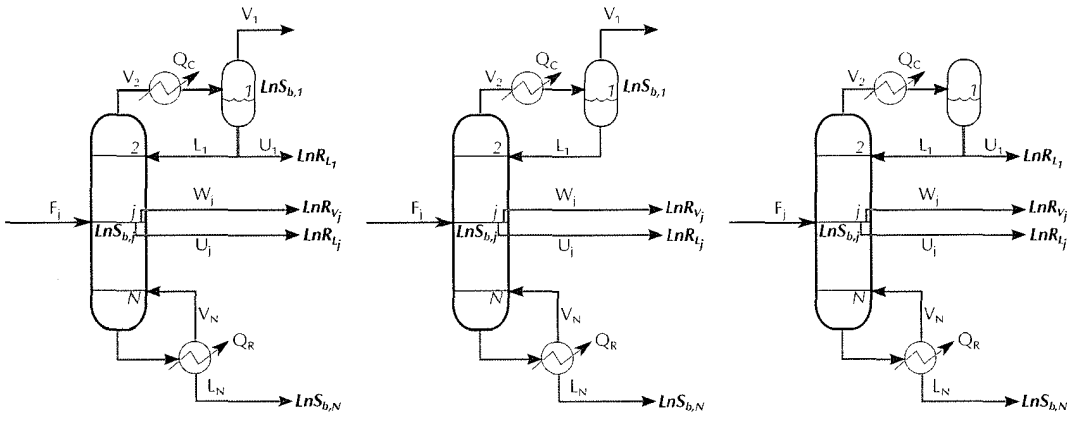
13. Seleccionar las variables de iteración como:

$$\ln S_{b,j} = \ln\left(K_{b,j}V_j/L_j\right) \tag{2.49}$$

$$\ln R_{L,j} = \ln\left(U_j/L_j\right) \tag{2.50}$$

$$\ln R_{V,j} = \ln\left(W_j/L_j\right) \tag{2.51}$$

junto con otras variables de iteración de ser necesario, como se muestra en la Figura 2.7. La transformación logarítmica se utiliza para mantener positivos los factores despojamiento y los factores de retiros laterales.



(a) Condensador parcial (b) Condensador reflujo total (c) Condensador total

Figura 2.7: Selección de variables de iteración para distintas configuraciones

De la Figura 2.7 se puede observar que para una columna de destilación simple solo con productos de tope y fondo, no se necesitarían otras variables de iteración de bucle interno adicionalmente a los factores de despojamiento $\ln Sb_j$. Además se deben especificar los flujos de calor del condensador y rehervidor respectivamente. Sin embargo, en tal caso se establezcan otras especificaciones, por ejemplo, si se especifican la relación de reflujo (L/D) y velocidad de flujo de producto (B) en lugar de dos flujos de calor del condensador y el rehervidor (que es la situación más común), se debe sustituir el Balance de Energía respectivo de la etapa de tope (H_1) y fondo (H_N), determinados a partir de la ecuación (2.34) por las funciones de discrepancia como se estableció en (2.3.1) según sea el caso. Además para cada corriente lateral, se seleccionará su factor de retiro correspondiente (2.50 y/o 2.51) como una variable de iteración adicional en el bucle interno junto con una especificación de pureza u alguna otra variable asociada con dicha corriente.

14. Calcular entalpías a partir de las ecuaciones (2.40) a (2.42).

15. Calcular las funciones de energía H_j normalizadas de las ecuaciones de balance de energía (2.34), recordando cuando serán sustituidas con sus respectivas funciones según la Tabla (2.2). Para efectuar la normalización es necesario dividir las funciones por un factor de escalamiento (2.52), el cual es aproximadamente igual al calor latente de vaporización, esto se realiza con la finalidad de mantener los valores de las funciones dentro de un mismo orden de magnitud.

$$FH_j = (\Delta H_{V_J} - \Delta H_{L_J}) \tag{2.52}$$

16. Calcular el Jacobiano de las funciones de energía y/o discrepancias H_J, con respecto a las variables de iteración (2.49, 2.50, 2.51), perturbando cada variable de iteración y recalculando los valores de las funciones de energía y/o discrepancias desde el paso 9 al 15, mediante diferenciación numérica.

$$J_H = \begin{bmatrix} \frac{\partial H_1}{\partial \ln S_{b,1}} & \cdots & \frac{\partial H_1}{\partial \ln S_{b,N}} & \frac{\partial H_1}{\partial \ln R_{L_1}} & \cdots & \frac{\partial H_1}{\partial \ln R_{L_N}} & \frac{\partial H_1}{\partial \ln R_{V_1}} & \cdots & \frac{\partial H_1}{\partial \ln R_{V_N}} \\ \frac{\partial H_2}{\partial \ln S_{b,1}} & \cdots & \frac{\partial H_2}{\partial \ln S_{b,N}} & \frac{\partial H_2}{\partial \ln R_{L_1}} & \cdots & \frac{\partial H_2}{\partial \ln R_{L_N}} & \frac{\partial H_2}{\partial \ln R_{V_1}} & \cdots & \frac{\partial H_2}{\partial \ln R_{V_N}} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial H_N}{\partial \ln S_{b,1}} & \cdots & \frac{\partial H_N}{\partial \ln S_{b,N}} & \frac{\partial H_N}{\partial \ln R_{L_1}} & \cdots & \frac{\partial H_N}{\partial \ln R_{L_N}} & \frac{\partial H_N}{\partial \ln R_{V_1}} & \cdots & \frac{\partial H_N}{\partial \ln R_{V_N}} \end{bmatrix} \tag{2.53}$$

17. Calcular las correcciones a las variables de iteración del bucle interno mediante iteración Newton-Raphson (A.2).

18. Determinar los nuevos valores de las variables de iteración a partir de la suma de los valores anteriores a partir de (A.18), utilizando factor de amortiguamiento (A.21) de ser necesario necesario para reducir la suma de cuadrados de las discrepancias normalizadas (A.19). Para la búsqueda del factor de amortiguamiento óptimo es posible emplear el método de minimización de Brent.

19. Verificar si la suma de los cuadrados de las funciones de discrepancia es lo suficientemente pequeña, es decir, menor que una tolerancia ϵ establecida para el bucle interno. De ser así, se procede al procedimiento de cálculo del bucle externo. Si no, repetir los pasos 15 a 18 utilizando el último valor de la variable de iteración.

20. Después de la convergencia de los pasos 15 a 19, los pasos 8 hasta 12 habrán producido un conjunto mejorado de variables primitivas $x_{i,j}$, $y_{i,j}$, $v_{i,j}$, $l_{i,j}$, T_j , V_j y L_j. Estas variables primitivas serán los datos de entrada a los cálculos de bucle externo. Los valores de estas variables no estarán correctos hasta que las propiedades termodinámicas aproximados concuerdan con las propiedades de los modelos rigurosos.

- **Secuencia de Cálculo del Bucle Externo**

Cada bucle externo procede de la siguiente manera:

21. Utilizando los valores de las variables primitivas del paso 20, calcular las volatilidades relativas y entalpías de las corrientes mediante los métodos termodinámicos rigurosos (MTR). Si la diferencia relativa de $\alpha_{i,j}$ determinadas por el método termodinámico aproximado (MTA) respecto a las calculadas con los MTR es menor que la tolerancia establecida para el bucle externo ϵ_2 como lo expresa la ecuación 2.54, tanto el bucle interno como el externo se consideran que han convergido, y el problema esta resuelto. Si no, ir al paso 22.

$$\tau = \sum_{i=1}^{n} \left(\alpha_{i,jMTA} - \alpha_{i,jMTR}/\alpha_{i,jMTA}\right)^2 \leq \epsilon_2 \tag{2.54}$$

22. Determinar los valores de los parámetros K y h del bucle iterativo externo a partir los MTR, tal como como en la etapa de inicialización 7.

23. Calcular los valores de $S_{b,j}$, R_{Lj} y R_{Vj} , como en el paso de inicialización 8 e iniciar una nueva secuencia de cálculo de bucle interno en el paso 9.

Algunas dificultades de convergencia surgen principalmente debido a pobres estimaciones iniciales y/o especificaciones de la columna que no se logren alcanzar. Aunado a estas dificultades se encuentran la gran variación en la magnitud relativa de las variables, errores de redondeo, y las matrices dispersas que resultan de las ecuaciones.

Las estimaciones iniciales pobres se traducen en flujos negativos o cero en ciertos lugares de la columna. Para contrarrestar esta tendencia, todos los factores de despojamiento por componentes utilizan un multiplicador escalar, S_b, llamado factor de despojamiento base, lo que resulta en:

$$S_{i,j} = S_b \alpha_{i,j} S_{b,j} \tag{2.55}$$

El valor de S_b se elige inicialmente para forzar a los resultados del procedimiento de inicialización para establecer una distribución razonable de los flujos de los componentes a lo largo de la columna. Boston & Sullivan (1974) recomiendan calcular un nuevo S_b para cada nuevo conjunto de valores de $S_{b,j}$ mientras que Russell (1983) recomienda que S_b se calcule sólo una vez.

Con la precisión superior y las capacidades de los métodos modernos, una columna de destilación no debe ser diseñada sin ellos. Un cálculo realizado por un método aproximado es inferior en cuanto a precisión, y en algunos casos puede dar resultados falsos. En trabajos de diseño de columnas el papel de los cálculos cortos se limitan a la eliminación de las opciones menos deseables de diseño, proporcionando al diseñador una estimación inicial para utilizar seguidamente los métodos rigurosos y dar solución de problemas de diseño final. Así, los métodos rigurosos se utilizan como técnica principal en el diseño primario y optimización (Kister, 1992; Seader et al., 2011).

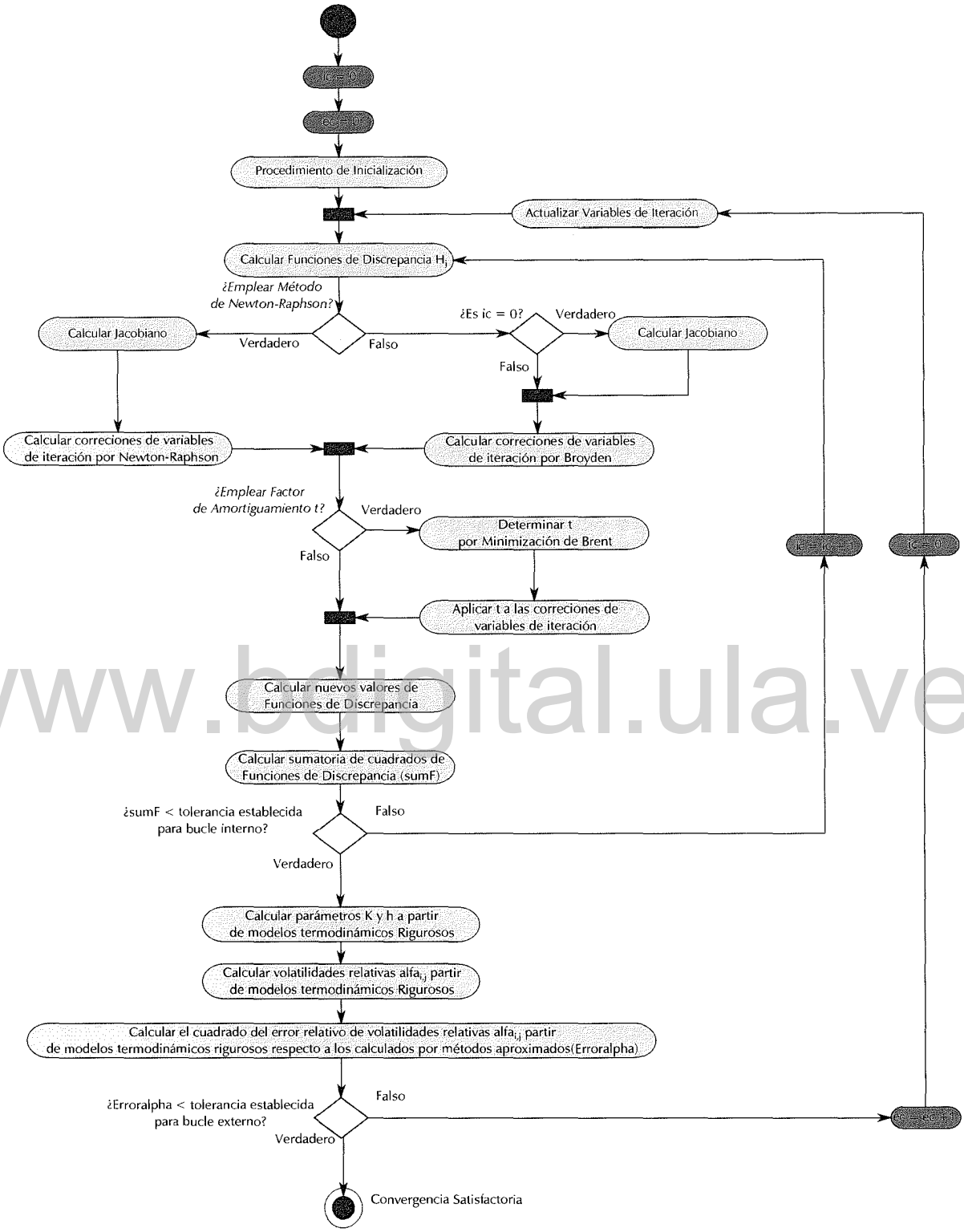


Figura 2.8: Diagrama de flujo general del algoritmo INSIDE-OUT de RUSSELL (1983)

La columna de destilación es probablemente la unidad más sofisticada en la simulación de procesos químicos. La disponibilidad de las computadoras ha hecho posible el uso de los métodos de solución riguroso para columnas de destilación basados en el modelo de etapa de equilibrio para sistemas multicomponentes.

Dentro de la clasificación de los distintos métodos rigurosos aplicados para la simulación de columnas de destilación, se mencionó un conjunto de métodos en particular, los métodos homotópicos y los métodos basados en velocidades de transferencia de masa o de No-equilibrio. El modelado de columnas de destilación basados en estos últimos pueden superar en cierta medida las deficiencias de los otros métodos. Por ejemplo, los métodos homotópicos resultan particularmente eficientes para soluciones líquidas altamente no ideales y cuando se tienen pobres estimados iniciales, mientras que los métodos de No-equilibrio, superan las deficiencias del modelo de etapa de equilibrio al incluir parámetros de transferencia de masa dentro del conjunto de ecuaciones MESH, con la finalidad de desarrollar un modelo más adaptado a la realidad (Perry & Green, 2008; Kister, 1992).

A pesar de las grandes ventajas que estos métodos pueden brindar. Una gran limitación de los métodos homotópicos por ejemplo, es su consumo de cómputo y de tiempo para presentar una solución. Los métodos basados en el modelo de No-equilibrio requieren el conocimiento de parámetros críticos, tales como los datos de transferencia de masa, o la geometría del dispositivo de contacto y que comparado con los modelos de etapa de equilibrio estos últimos pueden ser calculados independientemente de datos de la geometría de la columna o de los parámetros de transferencia de masa (Kooijman & Taylor, 2000; Kister, 1992).

Esto conduce a que los modelos clásicos de la etapa de equilibrio sean los más prácticos y los más utilizados. En particular el método *Inside-Out* Russell (1983) resulta más simple de implementar en comparación con los modelos homotópicos y de No-equilibrio. Aunque la convergencia del método *Inside-Out* no está garantizada, el método posee una robustez razonable y funciona adecuadamente con la mayoría columnas fraccionadoras empleadas en la industria petrolera nacional, además de que es computacionalmente eficiente (Seader et al., 2011; Kister, 1992).

Por estas razones se elige al método *Inside-Out* de Russell (1983) para su codificación e implementación dentro de DWSIM con base en su arquitectura de software.

Capítulo 3

Arquitectura del Simulador de Procesos DWSIM

En el capítulo 1 se resaltó que la simulación de procesos químicos está naturalmente vinculada al uso de programas de computador denominados simuladores y que éstos difieren ampliamente en su arquitectura e implementación. Pero a pesar de estas diferencias, la mayoría de los simuladores que se utilizan actualmente son aplicaciones monolíticas cerradas que se basan en arquitecturas de *software* tradicionales (Barrett et al., 2007; Pernalete et al., 2012).

A lo largo de los últimos años, se han propuesto nuevas arquitecturas para el desarrollo y la implantación de *software* complejo así como el desarrollo de nuevos lenguajes de programación, algunos de los cuales han ganado un enorme éxito en los últimos años (Braunschweig & Gani, 2002). De esta manera por un lado se introduce un nuevo tipo de paradigma de programación adoptado por la comunidad de desarrolladores denominado programación orientada a objetos (POO), el cual implica la creación de modelos del mundo real y la construcción de programas informáticos basados en esos modelos. Por otro lado la introducción de un nuevo tipo de arquitectura para simuladores de procesos basada en estándares abiertos definidos en términos del diseño conceptual mencionado en la sección 1.2.3.

Tal es el caso de DWSIM, un simulador de procesos químicos en estado estacionario que está desarrollado bajo este nuevo paradigma de programación inspirado además en un PME basado en arquitectura CAPE-OPEN planteada por Barrett & Yang (2005) centrada en ofrecer interoperabilidad entre componentes de software de un simulador de procesos químicos. Además DWSIM es una aplicación que se destaca por ser el único simulador de procesos químico a nivel mundial disponible bajo licencia libre y compatible con los estándares CAPE-OPEN, brindando las ventajas discutidas anteriormente en la sección 1.2.2. DWSIM fue creado en 2006 por el Ingeniero Daniel Wagner Oliveira de

Medeiros y actualmente siendo desarrollado junto con Gregor Reichert y Gustavo León. Las actividades relacionadas con el desarrollo de DWSIM se actualizan regularmente en `http://dwsim.inforside.com.br/blog/index.php` (Wagner, 2006).

El siguiente capítulo tiene como finalidad la descripción de la arquitectura del simulador de procesos DWSIM junto con una breve descripción del código fuente, para la cual fue necesario apoyarse en la ingeniería inversa, mediante el análisis sintáctico de su código fuente, extrayendo los aspectos generales importantes del simulador. De esta manera se logró comprender la organización general del simulador e identificar los componentes específicos involucrados para la simulación de columnas de destilación.

De acuerdo al nivel de responsabilidad dentro del desarrollo de *software*, son muchas las partes involucradas y de interés en su arquitectura (Kroll & Kruchten, 2003). Una única representación de la arquitectura del sistema resultaría demasiado compleja y poco útil, pues contendría mucha información irrelevante. Es por ello que se plantea la necesidad de representaciones que contengan únicamente elementos que resultan de importancia mediante el uso de vistas arquitectónicas.

Buschmann et al. (2007) establece que una vista arquitectónica representa un aspecto parcial de una arquitectura de software, que muestra propiedades específicas del sistema. Así, se puede definir a una vista arquitectónica como una descripción simplificada o abstracción de un sistema desde una perspectiva específica, que cubre intereses particulares y omite entidades no relevantes a esta perspectiva (Kroll & Kruchten, 2003).

En este sentido, se puede representar DWSIM en una vista arquitectónica general, sobre la cual es posible identificar los elementos principales del simulador de procesos y una vista arquitectónica más específica donde se identifican las clases y objetos relevante involucrados en la simulación del proceso de destilación.

3.1. Vista Arquitectónica General

Una vista general de la arquitectura del simulador se presenta en la Figura 3.1. La aplicación consta de 3 elementos principales, Ejecutivo del Simulador, Módulo de Serialización y La Interfaz Gráfica de Usuario (GUI).

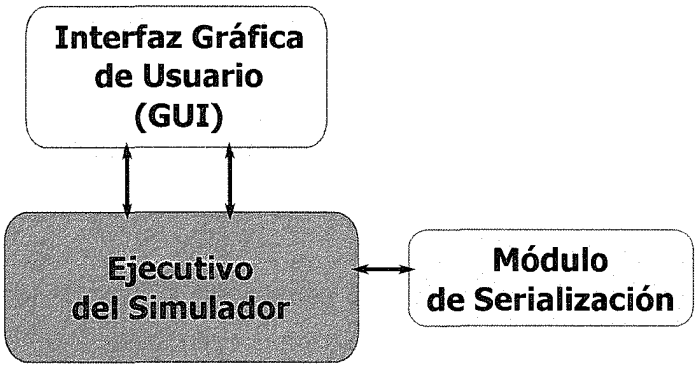


Figura 3.1: Arquitectura General de DWSIM

- Ejecutivo del Simulador:

  Considerado el corazón del sistema. Su función es gestionar tanto las tareas de cálculo y de intercambio de datos, como por ejemplo las secuencia de cálculo, la recuperación de los parámetros de propiedades físicas y termodinámicas, rutinas para las operaciones unitarias, seguimiento para la convergencia y gestión del sistema de archivos de datos, así como las bibliotecas de clases de componentes que implementan la funcionalidad especificada para las interfaces CAPE-OPEN.

- Módulo de Serialización

  Forma una unidad lógica o biblioteca que permite controlar cómo los objetos y estructuras de datos se traducen en un formato específico legible para el usuario que puede ser almacenado en memoria o transmitido por una conexión, proceso que se conoce como serialización (*marshalling* en inglés). La serialización es un mecanismo usado en gran medida para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos a varias aplicaciones o localizaciones (McMonnies, 2004).

- La Interfaz Gráfica de Usuario (GUI)

  Su función principal consiste en proporcionar el entorno visual para permitir la comunicación del usuario con el simulador. DWSIM por seguir una estrategia de solución modular secuencial, su GUI, se encarga de ayudar a dibujar el diagrama de flujo de proceso, proveer al usuario mediante un conjunto de imágenes y objetos gráfico, las distintas ventanas para introducir las especificaciones para las operaciones unitarias, así como la configuración de paquetes de cálculos termodinámicos y rutinas de cálculo para la simulación.

3.2. Arquitectura del Ejecutivo del Simulador

Dentro del ejecutivo del simulador de DWSIM se encuentran los bloques de código que conforman las partes principales de la aplicación. A nivel de POO, estos bloques pueden ser representados como módulos o un conjunto de funciones relacionadas denominados objetos.

Las tecnologías orientadas a objetos se han convertido en uno de los motores clave de la industria del software. El desarrollo de programas orientados a objetos es un enfoque diferente del mundo informático que supera y amplían conceptos antiguos de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación. Implica la creación de modelos del mundo real y la construcción de programas informáticos basados en esos modelos (Aguilar, 2003).

En este sentido, el ejecutivo de simulación de DWSIM se divide en objetos auto contenidos que representa una parte diferente de la aplicación. Sin embargo en la POO, desde el punto de vista técnico se habla de *clases* y no de *objetos*, dado que las clases equivalen a modelos o plantillas que describen cómo se construyen dichos objetos. Cada vez que se construye un objeto a partir de una clase se está creando lo que se denomina una instancia de esa clase (McMillan, 2004). Por consiguiente, los objetos no son más que instancias de una clase. En un contexto del mundo real, se puede pensar en "Columna" como una clase y una columna concreta con determinadas dimensiones, numero de etapas y determinadas especificaciones como una instancia de esta clase "Columna".

3.2.1. Diagrama de Clases del Ejecutivo del Simulador

Una clase por lo general representa un sustantivo, lugar o cosa. Describe lo que será un objeto, pero no es el objeto en sí. Es el modelo de un concepto dentro de un programa de computadora. Fundamentalmente, delimita los posibles estados y define el comportamiento del concepto que representa (Aguilar, 2003; McMillan, 2004). Encapsula el estado a través de espacios de almacenaje de datos llamados atributos o, lo que es lo mismo, sus propiedades o características, y encapsula el comportamiento a través de secciones de código llamadas métodos que acceden a los atributos de una manera predefinida (McMillan, 2004).

En la Figura 3.2 se muestra el diagrama de clases general de DWSIM donde describe la estructura del ejecutivo del simulador mostrando algunas de sus clases principales. El diagrama de clases es la descripción más importante y más utilizada de un sistema orientado a objetos. La recuperación del diagrama de clases a partir del código fuente es una

tarea difícil. La decisión acerca de cuáles son los elementos para mostrar u ocultar afecta profundamente a la capacidad de uso del diagrama (Tonella & Potrich, 2005).

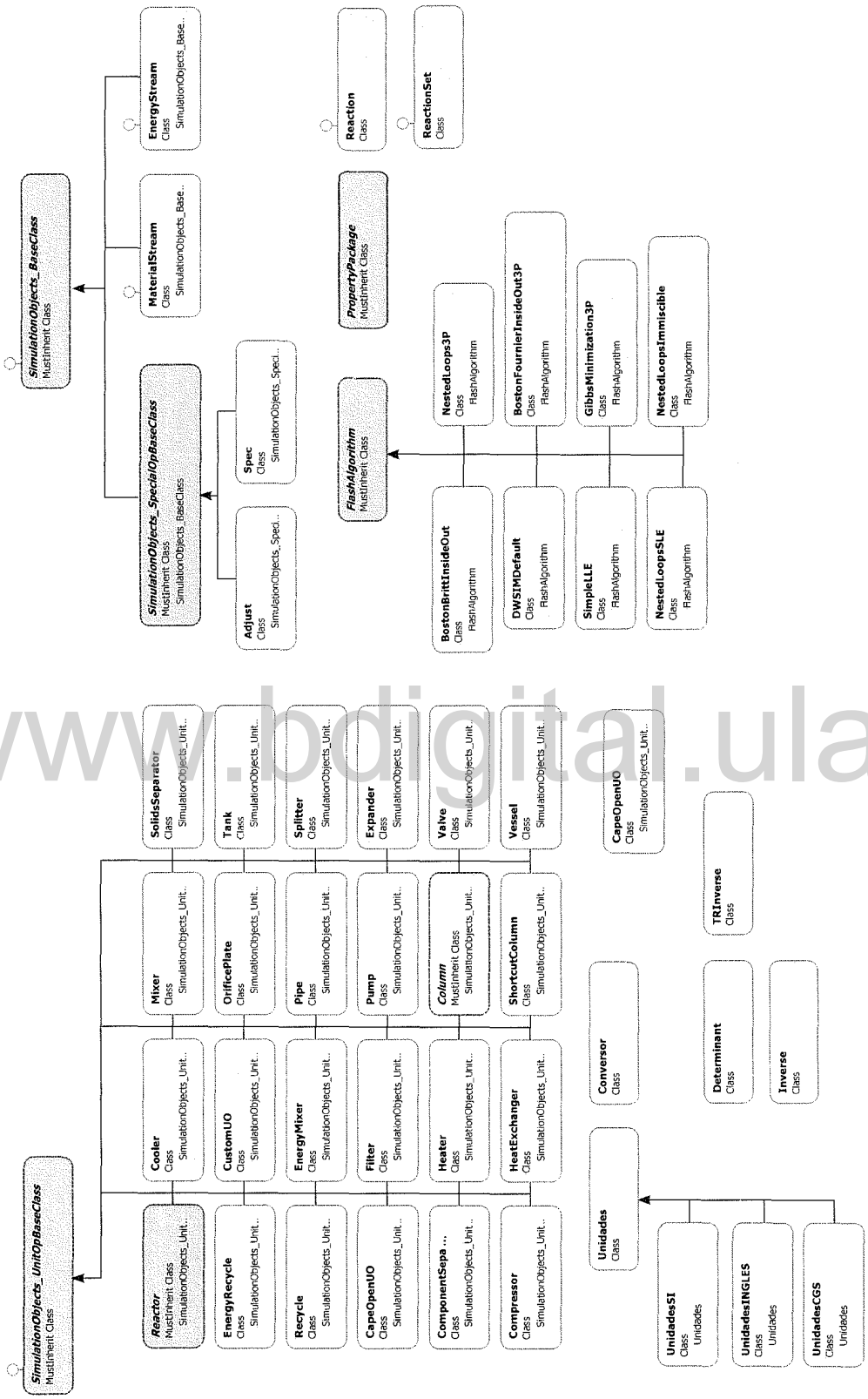


Figura 3.2: Diagrama de Clases Simplificado de DWSIM

En este diagrama se observa las clases de los componentes de modelado de proceso (PMC) para su uso dentro de la interfaz gráfica de usuario, es decir, sobre el diagrama de flujo dentro de la simulación. Se muestra las clases específicas tales como las correspondientes a los modelos de propiedades termodinámicas (`PropertyPackage`, `FlashAlgorithm`), las clases de reacciones químicas (`Reaction`, `ReactionSet`), algunas rutinas de cálculo matricial (`TRInverse`, `Determinant`, `Inverse`). También la clase de operación unitaria que implementa la funcionalidad especificada para las interfaces CAPE-OPEN (`CapeOpenUO`). Existen otras clases que actúan como clases auxiliares de los PMC, que por fines prácticos no se muestran en el diagrama. No obstante, se muestra un completo diagrama de clases de DWSIM en el Apéndice C.

Aunado a esto se observa que las clases se organizan en una estructura arbórea siguiendo una jerarquía. En este sentido, las clases base se dividen en subclases empleando el concepto de herencia, el cual es una relación que se da entre una clase general y otra clase más específica (McMillan, 2004). Así, las clases derivadas o de menor jerarquía heredan las características (atributos y métodos) de su clase base o clases superiores.

Cada clase contiene cierto número de atributos, cada uno de los cuales tendrá, a su vez, uno o varios valores. En la POO, los atributos corresponden a las clásicas "variables" de la programación estructurada. Son, por lo tanto, datos encapsulados dentro del objeto. Los atributos de un objeto pueden tener un valor único o pueden contener un conjunto de valores mas o menos estructurados (matrices, vectores, listas, etc.), que además pueden ser de un determinado tipo (*integer*, *string*, *float*, *char* .etc) (Aguilar, 2003; McMonnies, 2004). Junto con los atributos de las clases se encuentran los métodos, que son funciones que operan sobre los atributos de los objetos, y proporcionan el comportamiento del objeto (Tonella & Potrich, 2005).

3.2.2. Diagrama de Objetos de Columna de Destilación

La clase `SimulationObjects_UnitOpBaseClass` es la que contiene las características comunes a los PMC de operaciones unitarias concretas utilizadas, es una clase de la que no se puede crear objetos, la utilidad de estas clases estriba en que otras clases hereden de ésta. Dentro de la categoría de PMC destinados para la simulación de operaciones unitarias, se encuentra la clase `Column`, ésta hereda los atributos y métodos de la clase madre `SimulationObjects_UnitOpBaseClass` y a su vez se deriva la subclase `DistillationColumn` la cual contienen atributos y métodos adicionales para la instanciación e inicialización de una la columna de destilación.

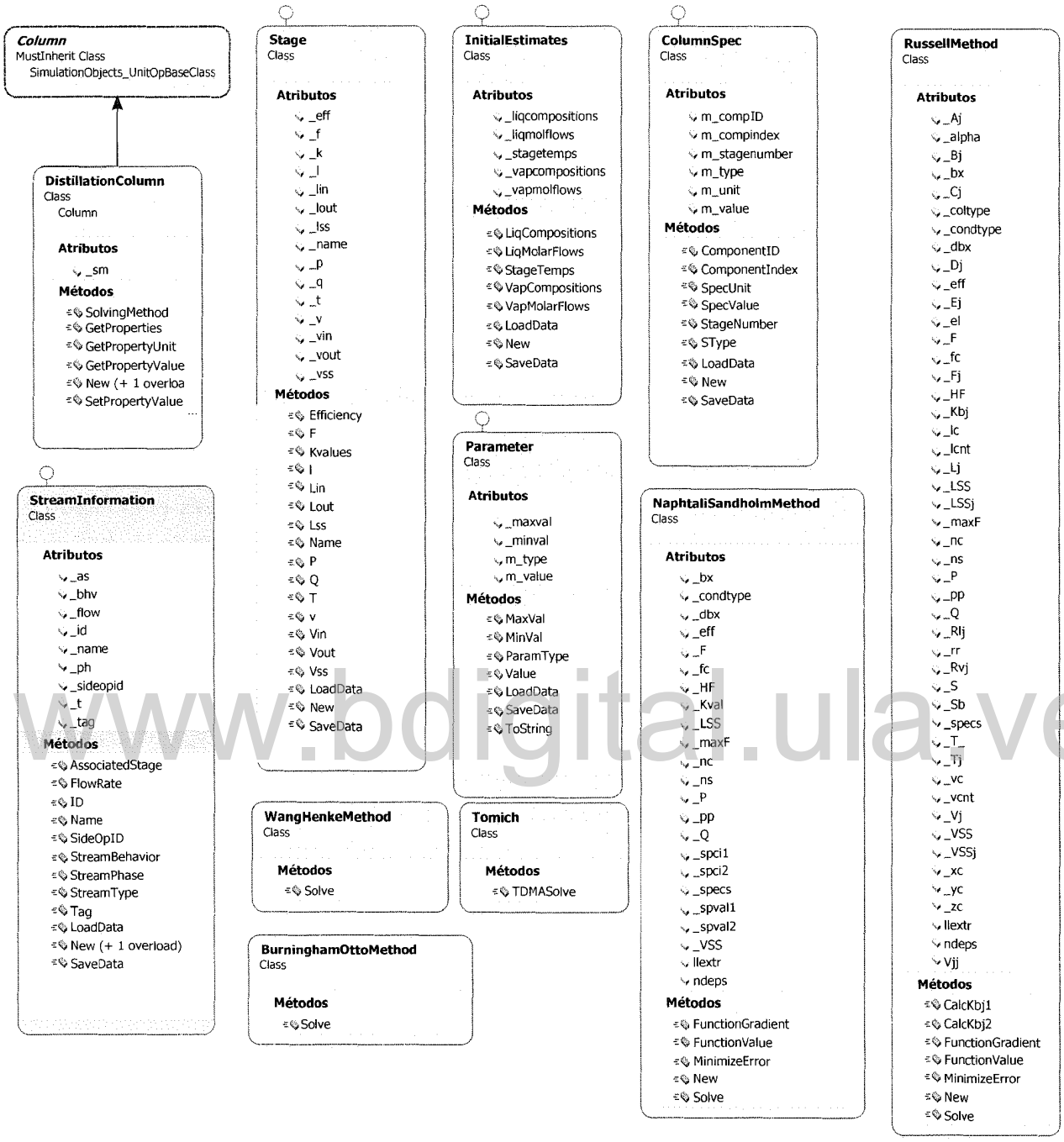


Figura 3.3: Clase `DistillationColumn`

La clase `DistillationColumn` mediante sus métodos es capaz de interactuar con otros clases que se muestran en la Figura 3.3, descritas brevemente en la Tabla 3.1, los cuales hacen posible el modelaje y simulación del proceso de destilación multicomponente en

estado estacionario. Por otra parte, se muestra una descripción de los métodos miembro de la clase `RussellMethod` en la Tabla 3.2.

Tabla 3.1: Funciones de las clases que interactúan con *DistillationColumn*

Nombre de la clase	Función
`Stage`	Modelar la etapa de equilibrio que compone la columna de destilación. Gestiona la información referente a los distintos parámetros involucrados en las ecuaciones de Balance de materia y Energía.
`InitialEstimates`	Gestiona los valores iniciales para flujos de vapor, flujos de líquido, perfil de temperatura y perfil de presión de la columna de destilación.
`ColumnSpec`	Maneja la información relacionada con los valores establecidos por el usuario para las especificaciones de la columna de destilación así como el conjunto de ecuaciones y funciones de discrepancias.
`Parameter`	Clase auxiliar que emplea métodos para devolver valores máximos y mínimos de un conjunto de datos.
`Tomich`	Clase auxiliar donde se implementa el método de resolución matriz tridiagonal detallado en A.1.
`StreamInformation`	Gestiona la información de las distintas corrientes de materia y energía que se encuentren acopladas con la columna de destilación.
`WankHenkeMethod`	Clase en donde se implementa el método riguroso de punto de burbuja BP desarrollado por Wang & Henke (1966) descrito en la sección 2.2.1.
`BurninghamOttoMethod`	Clase en donde se implementa el método riguroso de suma de caudales SR desarrollado por Burningham & Otto (1967) descrito en la sección 2.2.1.
`NaphtaliSandholmMethod`	Clase en donde se implementa el método riguroso de corrección simultánea desarrollado por Naphtali & Sandholm (1971) descrito en la sección 2.2.1.
`RussellMethod`	Clase en donde se implementa el método riguroso Inside-Out desarrollado por Russell (1983) descrito en la sección 2.3.

Tabla 3.2: Métodos miembro de la clase RUSSELLMETHOD

Nombre del método	Función
`CalcKbj1`	Método que se encarga de determinar y devolver los valores de $K_{b,j}$ discutido en 2.3.2, a partir de un método simplificado.
`CalcKbj2`	Método que se encarga de determinar y devolver los valores de $K_{b,j}$ discutido en 2.3.2, a partir del modelo de la ecuaciones 2.36 y 2.37.
`FunctionValue`	Método que se encarga de determinar y devolver los valores de las funciones de discrepancia H_j discutido en la sección 2.3.3. abarca el paso 7 hasta 15.
`FunctionGradient`	Se encarga de determinar y devolver los valores de los elementos diferenciales del Jacobiano invocando al método `FunctionValue` empleando diferencias finitas descrito en el paso 16.
`MinimizeError`	Se encarga de determinar y devolver el valor del factor de amortiguamiento t de la ecuación A.21 empleando el algoritmo de minimización de Brent.
`Solve`	Método principal que implementa el completo Algoritmo Inside-Out de Russell (1983) detallado en la sección 2.3.3. Este método encapsula los métodos anteriores, invocándolos según cuando sea necesario, excluyendo el procedimiento de inicialización.
`New`	Un método especial reservado del lenguaje de programación es una subrutina por defecto cuya función es construir el objeto de la clase `RussellMethod`. A pesar de que se trata de un método sin parámetros, este será invocado cada vez que se construya un objeto sin especificar ningún argumento, en cuyo caso el objeto será iniciado con los valores predeterminados por el sistema.

3.3. Breve Descripción del Código Fuente de DWSIM

El código fuente completo y actualizado de ***DWSIM*** se almacena de forma pública en la plataforma de desarrollo colaborativo de software *GitHub* para alojar proyectos utilizando el sistema de control de versiones Git `https://github.com/DanWBR/dwsim3`. DWSIM está desarrollado bajo el lenguaje de *Visual Basic .NET* orientado a objetos (*VB.NET*), y que puede ser compilado con el entorno de desarrollo integrado (IDE) Visual Basic 2010 en su edición *express* ó el IDE bajo licencia libre *SharpDevelop* 4. El código fuente de DWSIM dispone de contenedores que administran los elementos necesarios para el desarrollo, como referencias, conexiones de datos, carpetas y archivos. Estos contenedores se denominan

soluciones y proyectos. Así, el archivo de solución principal de DWSIM contiene propiedades que se pueden acceder mediante el explorador de soluciones del IDE que permite ver y administrar sus proyectos asociados.

En la Figura 3.4 se muestra una instantánea fotográfica del archivo solución del código fuente de DWSIM donde se pueden identificar 6 proyectos, asociados a cada componente de la arquitectura general descrita en la Figura 1.2.3. `DWSIM` y `NaturalGasPlugin`, son proyectos asociados al Ejecutivo del Simulador, el proyecto `CustomXMLSerializer` con el Módulo de Serialización de DWSIM y los proyectos restantes `DesignSurface`, `GraphicObjects`, `PropertyGridEx` están asociados al la Interfaz Gráfica de Usuario (GUI).

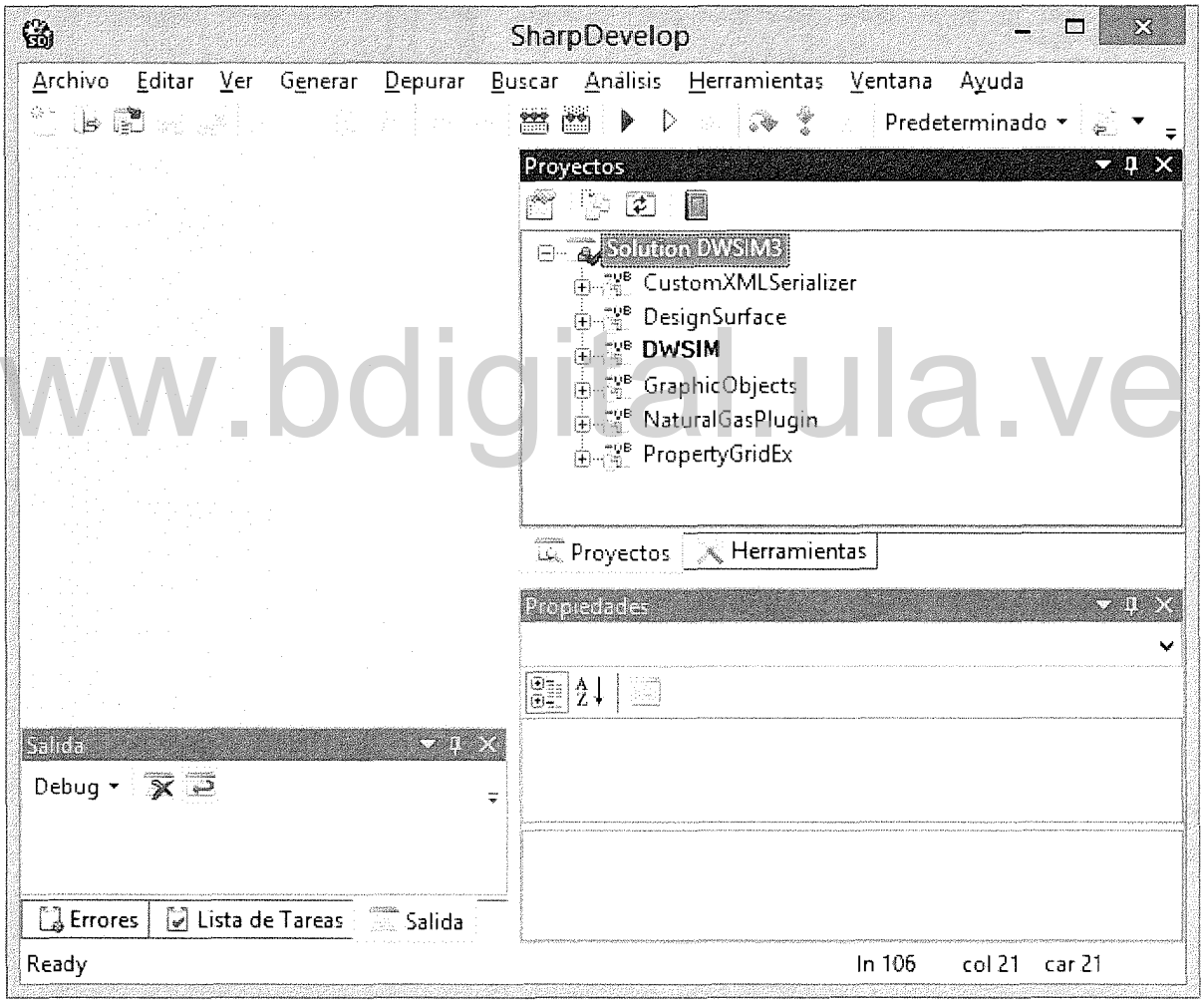


Figura 3.4: Instantánea fotográfica del explorador de solución para DWSIM 3 en *SharpDevelop 4*

Dentro del proyecto `DWSIM` se encuentran contenidos los archivos *Scripts* del Ejecutivo del Simulador, específicamente los relacionados con las clases descritas en la sección 3.2.1. Los archivos Scripts `RigorousColumn.vb` y `RigorousColumnSolver.vb`. Estos archivos contienen los códigos de las clases descritas en la sección 3.2.2 relacionadas con los cálculos rigurosos de la columna de destilación, los cuales fueron modificados para efectos de este trabajo (Apéndice E) tomando como referencia los diagramas de actividades desarrollados en el Apéndice B. En la Figura 3.5 se muestra la localización de ambos archivos dentro del proyecto DWSIM.

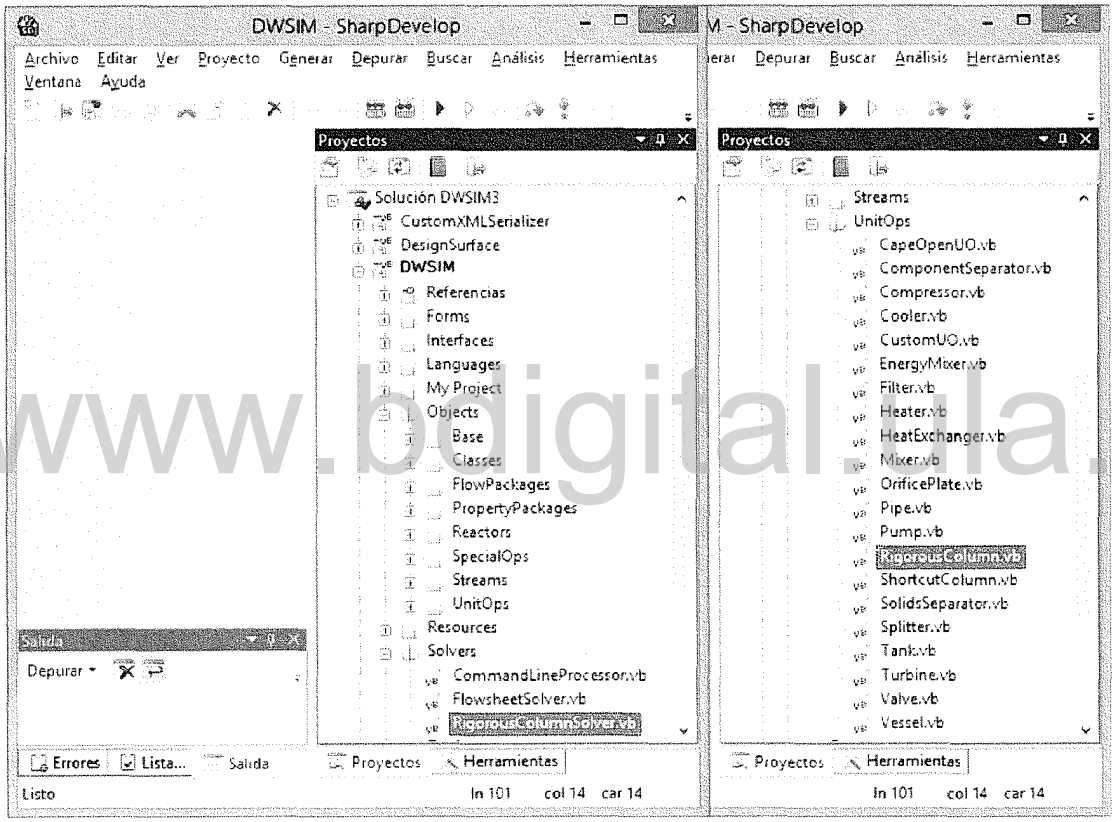


Figura 3.5: Localización de archivos *Scripts* `RigorousColumn.vb` y `RigorousColumnSolver.vb` dentro del proyecto DWSIM del código fuente

De la descripción de las clases, objetos y código fuente, se puede concluir que DWSIM es un software de simulación de procesos sofisticado, y no una colección de algoritmos desorganizados para la resolución de los balances de materia y energía de las diferentes operaciones unitarias. El simulador de procesos DWSIM al estar desarrollado bajo una

tecnología basada en objetos, permite dar un tratamiento sobre las clases de interés para el programador.

Por otra parte DWSIM cuenta con los elementos necesarios para simular columnas de destilación rigurosa exhibiendo las ventajas reconocidas del software libre. Esto permite a los usuarios ver cómo se realizan realmente los cálculos durante una simulación del proceso de destilación mediante el estudio de su código fuente, lo que brinda una mejor comprensión y llevar a cabo adaptaciones con base a requerimientos propios o mejoras mediante su modificación. DWSIM por otra parte es compatible con los estándares CAPE-OPEN lo que habilita la posibilidad de desarrollar, incorporar y utilizar componentes de simulación externos compatibles con el estándar.

Capítulo 4

Casos de Estudio

Una vez realizado el estudio de la arquitectura general del simulador de procesos DWSIM, se logró identificar e inspeccionar las distintas clases y métodos involucrados en el modelado y simulación de columnas de destilación. Y de esta manera realizar la implementación del algoritmo Inside-Out de Russell (1983) a través de su codificación y posterior mejora. En esta sección de discuten los casos de estudio empleados para la verificación y validación del algoritmo de resolución de columna de destilación implementado en DWSIM.

4.1. Casos de Estudio de Literatura

Los casos descritos en la Tabla 4.1, donde se muestras las especificaciones y configuración de las columnas de destilación, se efectuaron con la finalidad de demostrar la consistencia interna del código implementado en el simulador de procesos DWSIM, es decir, comprobar que la implementación del código corresponde al algoritmo Inside-Out basado en el modelo de etapas de equilibrio. Para ello, se realizó la lectura del código junto con un proceso de inspección en tiempo de ejecución para evaluar variables y expresiones, proceso denominado depuración de código (*debugging*)(Aguilar, 2003). De esta manera verificar que se han introducido adecuadamente los datos y la lógica del método Inside-Out de Russell (1983). Los resultados arrojados por la implementación realizada en DWSIM se compararon con los resultados arrojados por los modelos Inside-Out de otros dos simuladores privativos, en este caso, el simulador de procesos químicos PRO/II® y el simulador ChemSep™ en su versión gratuita. Los archivos de simulación están disponibles para descargar por medio del siguiente enlace `http://bit.ly/1iBoSnG`. (*Password*: DWSIM).

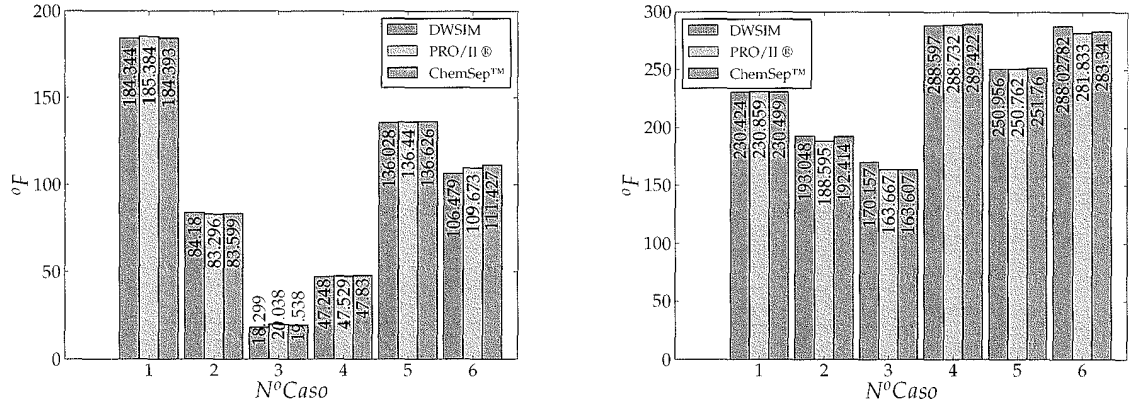
Tabla 4.1: Características de Casos de Estudio de Literatura

	Caso 1	Caso 2	Caso 3	Caso 4	Caso 5	Caso 6	
Presión de Columna (psi)	16,17	100,00	400	20	290,08	239,31	
Número de Etapas de Equilibrio	10	5	13	30	31	16	
ΔP *Caída de Presión (psi)*	0,00	0,00	0,00	5	0,00	0,00	
Tipo de Condensador	Total 	Total 	Full-Reflux 	Total 	Total 	Parcial 	
Vapor Producto de Tope (lbmol/h)	-	-	530,000	-	-	33,071	
Líquido Producto de Tope (lbmol/h)	46,54	50,000	-	14,08	110,7	11,023	
Producto de Fondo (lbmol/h)	46,69	50,000	270,000	39,94	330.30	88,184	
Retiro Lateral Líquido (lbmol/h)	-	-	-	19,53	-	6,614	
Etapa del Retiro de Líquido	-	-	-	10	-	3	
Retiro Lateral Vapor (lbmol/h)	-	-	-	24,78	-	81,571	
Etapa del Retiro de Vapor	-	-	-	24	-	13	
Relación de Reflujo	2,00	2,00	1,89	20,00	6,00	4,24	
Alimentaciones	1	1	1	1	1	1	2
Metano (lbmol/h)	-	-	160,00	-	-	-	-
Etano (lbmol/h)	-	-	370,00	-	-	5,51	1,10
Propano (lbmol/h)	-	30,000	240,00	-	110,23113	30,86	13,23
N-butano (lbmol/h)	-	30,000	25,00	14,08	110,23113	41,89	39,68
Isobutano (lbmol/h)	-	-	-	-	110,23113	-	-
N-pentano (lbmol/h)	-	40,000	5,00	19,53	110,23113	11,02	66,14
N-hexano (lbmol/h)	-	-	-	24,78	-	1,102	9,92
N-octano (lbmol/h)	-	-	-	39,94	-	-	-
Benceno (lbmol/h)	46,62	-	-	-	-	-	-
Tolueno (lbmol/h)	46,62	-	-	-	-	-	-
Presión de Alimentación (psi)	16,17	100,00	400,00	25,00	290,08	239,31	
Temperatura de Alimentación (°F)	Temp. de Burbuja	Temp. de Burbuja	105,00	150,00	207,77	151,79	235,67
Etapa de Alimentación	5	3	7	14	13	6	9

Primero se realizó una comparación global de los distintos casos tomando como referencia los valores obtenidos por el algoritmo. Se establecieron como variables de comparación las temperaturas, los flujos de calor tanto del condensador como del rehervidor así como el número de iteraciones obtenidos por cada uno de los simuladores, los que se muestran en las Figuras 4.1-4.3.

Se confirma claramente la reproducibilidad de los datos arrojados simuladores de procesos. Se muestra concordancia entre los valores de temperaturas y flujos de calor predichos por cada simulador en las Figuras 4.1 y 4.2. Sin embargo para el caso número 5 se observan

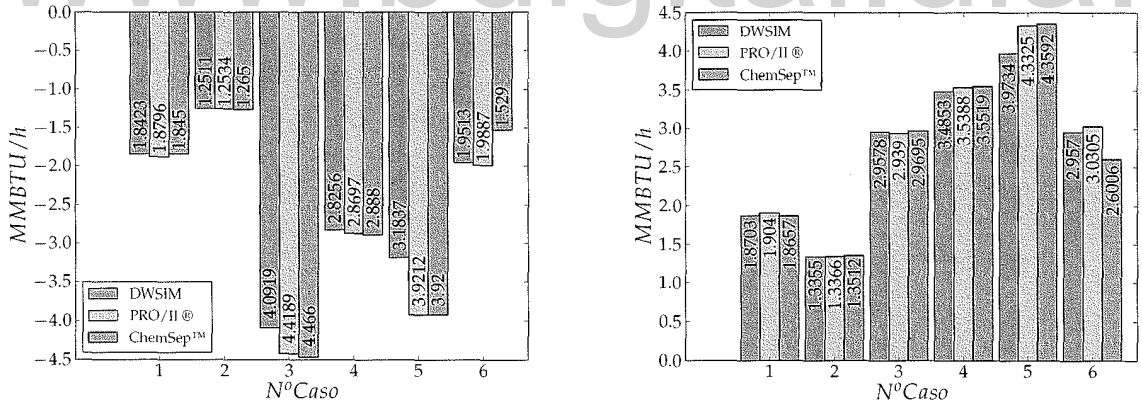
diferencias significativas en los valores de flujos de calor arrojados por DWSIM en comparación con los reportados por PRO/II® y ChemSep™, específicamente una diferencia de aproximadamente -0,737 $^{MMBTU}/_{h}$ respecto al valor promedio -3,92 $^{MMBTU}/_{h}$ de flujo de calor en el condensador y una diferencia de 0,3724 $^{MMBTU}/_{h}$ respecto al valor promedio de 4,35 $^{MMBTU}/_{h}$ de flujo de calor para el rehervidor. Seguidamente del caso número 6 donde existe una leve diferencia significativa con respecto a los valores arrojados por ChemSep™.



(a) Temperaturas de Condensador

(b) Temperaturas de Rehervidor

Figura 4.1: Comparación Temperaturas para Casos de Literatura



(a) Flujos de Calor de condensador

(b) Flujos de Calor de Rehervidor

Figura 4.2: Comparación Flujos de Calor para Casos de Literatura

Estas desviaciones en la práctica, por ejemplo en el área de diseño, pueden llevar a diseño inadecuado de equipos, por lo que es responsabilidad del diseñador verificar rigurosamente los resultados antes de tomar decisiones finales de diseño (Taylor, 2007). Esta

desviación de los valores de flujos de calor es consecuencia de los perfiles de composición del producto de tope y fondo, los que se muestran en la Tabla 4.2, quienes repercuten directamente en los cálculos de balance de energía.. Se evidencia que los resultados arrojados por DWSIM para el caso número 5 y 6 reflejan una mejor distribución y separación de los componentes en términos de sus volatilidades en la columna de destilación respecto a PRO/II®.

		DWSIM	PRO/II®	ChemSep™
Producto de Tope de Columna	Propano ($^{lbmol}/_{h}$)	107,985	107,2948	107,316
	Isobutano($^{lbmol}/_{h}$)	1,95351	2,5291	2,51727
	N-butano($^{lbmol}/_{h}$)	0,29272	0,3915	0,39724
	N-pentano($^{lbmol}/_{h}$)	0,00019	0,0003	0,000269
Producto de Fondo de Columna	Propano ($^{lbmol}/_{h}$)	2,24635	2,9364	2,9150421
	Isobutano($^{lbmol}/_{h}$)	108,278	107,7021	107,714
	N-butano($^{lbmol}/_{h}$)	109,938	109,8397	109,834
	N-pentano($^{lbmol}/_{h}$)	110,231	110,231	110,231

(a) Caso 5

		DWSIM	PRO/II®	ChemSep™
Producto de Tope de Columna (Vapor)	Etano ($^{lbmol}/_{h}$)	5,992	5,678	5,672
	Propano($^{lbmol}/_{h}$)	27,034	26,716	26,434
	N-butano($^{lbmol}/_{h}$)	0,0438	0,674	0,961923
	N-pentano($^{lbmol}/_{h}$)	0,00	0,0019	0,0012
	N-hexano($^{lbmol}/_{h}$)	0,00	0,00	0,00
Producto de Tope de Columna (Líquido)	Etano ($^{lbmol}/_{h}$)	0,56902	0,8129	0,8069
	Propano($^{lbmol}/_{h}$)	10,3927	9,6171	9,3701
	N-butano($^{lbmol}/_{h}$)	0,06111	0,59	0,8437
	N-pentano($^{lbmol}/_{h}$)	0,00002	0,0033	0,0025
	N-hexano($^{lbmol}/_{h}$)	0,00	0,00	0,00
Producto de Fondo de Columna	Etano ($^{lbmol}/_{h}$)	0,00	0,00	0,00
	Propano($^{lbmol}/_{h}$)	0,00291	0,1214	0,1936
	N-butano($^{lbmol}/_{h}$)	16,0804	23,2045	23.8928
	N-pentano($^{lbmol}/_{h}$)	61,63087	54,8643	54.3196
	N-hexano($^{lbmol}/_{h}$)	10,47066	9,9946	9.77881

(b) Caso 6

Tabla 4.2: Perfiles de flujos de componentes para productos de tope y fondo

Seguidamente dentro de la comparación global se muestra en la Figura 4.3 el número de iteraciones que les tomó a cada simulador para converger a una solución satisfactoria.

Se observa claramente que PRO/II® resultó ser el simulador con menor número de iteraciones, seguido de DWSIM con 3 iteraciones y 4 para los casos 1 y 2 respectivamente. Para el caso 3 se observa un resultado igualado para PRO/II® y ChemSep™ con 4 iteraciones y DWSIM con 6 iteraciones, mientras que para los casos restantes se observa una igualdad entre los simuladores exceptuando a ChemSep™ con 7 iteraciones en el caso 4.

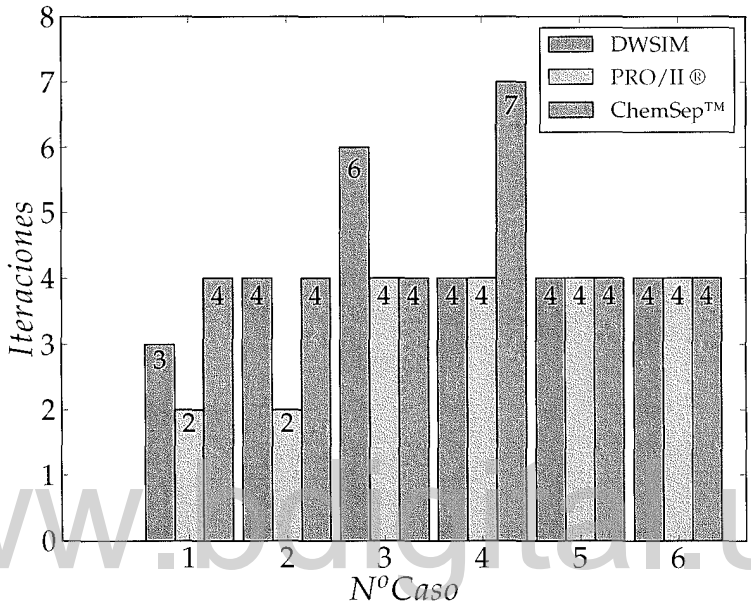
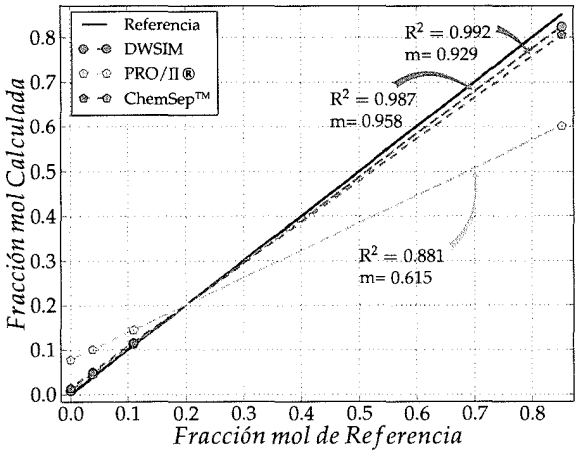


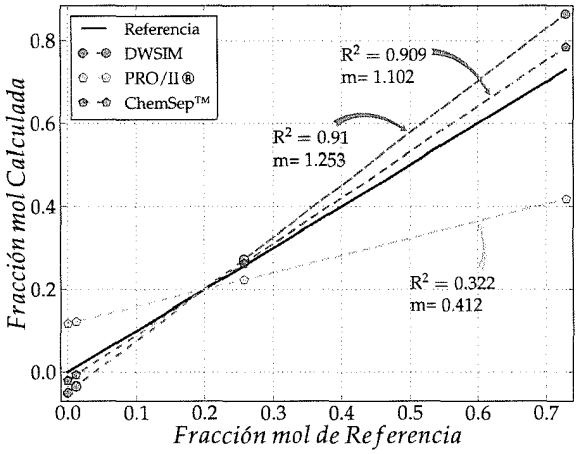
Figura 4.3: Número de Iteraciones de Casos de Literatura

Ahora realizando una comparación más especifica, se generaron gráficas de paridad de los valores de las fracciones molares de cada componente mostrados en el Apéndice D.1 para el caso número 6 con finalidad de comparar las correlaciones entre los datos de referencia reportados por Monroy-Loperena & Vacahern (2013) y los datos predichos por DWSIM, PRO/II® y ChemSep™. Estas gráficas se muestran en la Figura 4.4 donde además se indica sus coeficientes de correlación (R^2) y sus pendiente (m) del ajuste lineal. Considerando que los resultados del simulador que mejor corresponda a los datos de referencia serán aquellos cuyos valores R^2, m sean lo más cercano posible a 1. Al estudiar la gráfica de paridad y comparando en primer lugar a DWSIM con PRO/II® se evidencia que DWSIM se ajusta mejor a los datos de referencia que el modelo resuelto por el algoritmo Inside-Out del simulador PRO/II® en las Figuras 4.4a, 4.4b, 4.4c y 4.4d, en el cual se observa para caso de DWSIM los coeficientes de correlación R^2 son mayores y la pendiente m son más

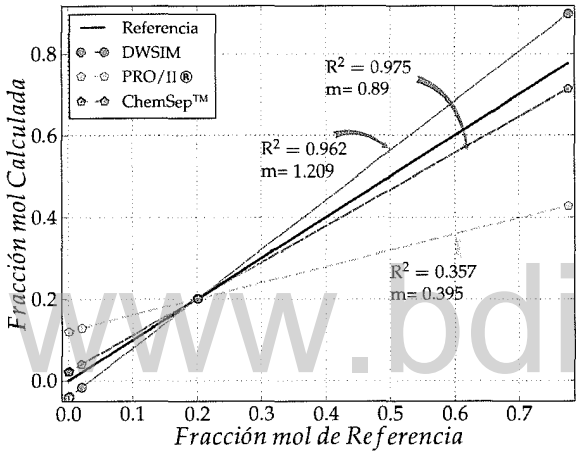
cercanas a 1. Caso contrario en el caso de las fracciones molares del producto de fondo en la Figura 4.4e. En segundo lugar realizando la comparación entre DWSIM y ChemSep™ se evidencia que ChemSep™ se ajusta mejor a los datos de referencia que el modelo Inside-Out implementado en DWSIM, ya que que en el caso de ChemSep™ los coeficientes de correlación R^2 son mayores y la pendiente m son más cercanas a 1.



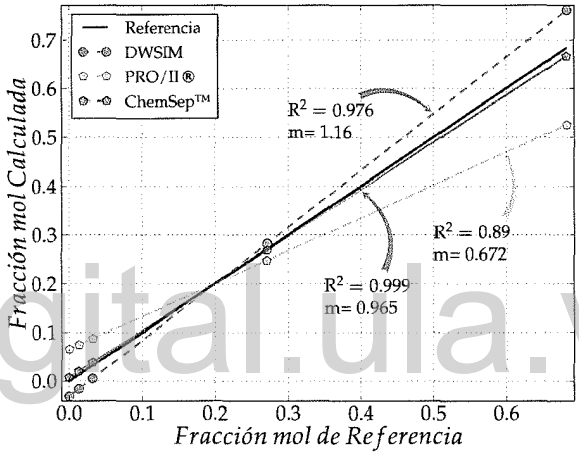
(a) Producto vapor de tope



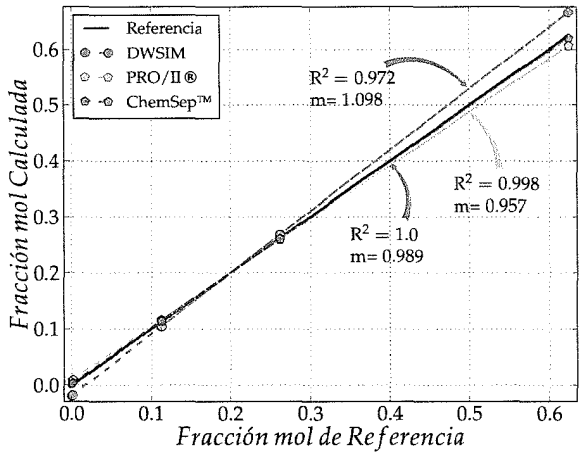
(b) Producto líquido de tope



(c) Retiro líquido lateral



(d) Retiro vapor lateral



(e) Producto líquido de fondo

Figura 4.4: Diagramas de paridad de las fracciones molares de productos del caso literatura 6

Estos resultados no solo demuestran claramente la consistencia interna del código implementado en DWSIM, quedando verificado el modelo Inside-Out de Russell (1983) basado en el modelo de etapas de equilibrio descrito en la sección 2.3.3. Si no que además reflejan que el desempeño del modelo DWSIM es mejor comparado con el modelo implementado en el simulador de proceso comercial PRO/II®.

4.2. Casos de Estudio Operacionales

Junto con la verificación del modelo realizada en la sección 4.1, se encuentra la validación del mismo. Es así como en esta sección se comprueba la exactitud del modelo desarrollado. Esto se lleva a cabo comparando las predicciones del modelo con mediciones realizadas en sistemas reales.

Fueron seleccionados dos casos de estudio operacionales pertenecientes al circuito nacional de refinación de PDVSA. El primero corresponde a una columna separadora de compuestos aromáticos del Complejo Refinador El Palito. Las especificaciones y configuración de la simulación se detallan en la Tabla 4.3, donde se muestran dos escenarios de operación. El segundo caso a una depropanizadora del complejo FCC Cardón perteneciente al Centro de Refinación Paraguaná (CRP), los cuales se detallan en la Tabla 4.5, resaltando que para este último caso se empleó como un paquete termodinámico externo que lleva por nombre *Thermodynamics for Engineering Applications* (TEA) aprovechando la compatibilidad de CAPE-OPEN existente en DWSIM. TEA está disponible a través de la instalación del simulador de procesos privativo de uso gratuito COCO(`http://www.cocosimulator.org/index_download.html`). La configuración de DWSIM con TEA se lleva a cabo durante la selección del paquete termodinámico al momento de crear la simulación del proceso en DWSIM. Seleccionando CAPE-OPEN como paquete termodinámico en lugar de utilizar los paquetes nativos de DWSIM y posteriormente seleccionando TEA en las opciones del *Property Package Manager*. En la Figura 4.5 se muestra una instantánea fotográfica de la ventana de configuración del paquete TEA en DWSIM.

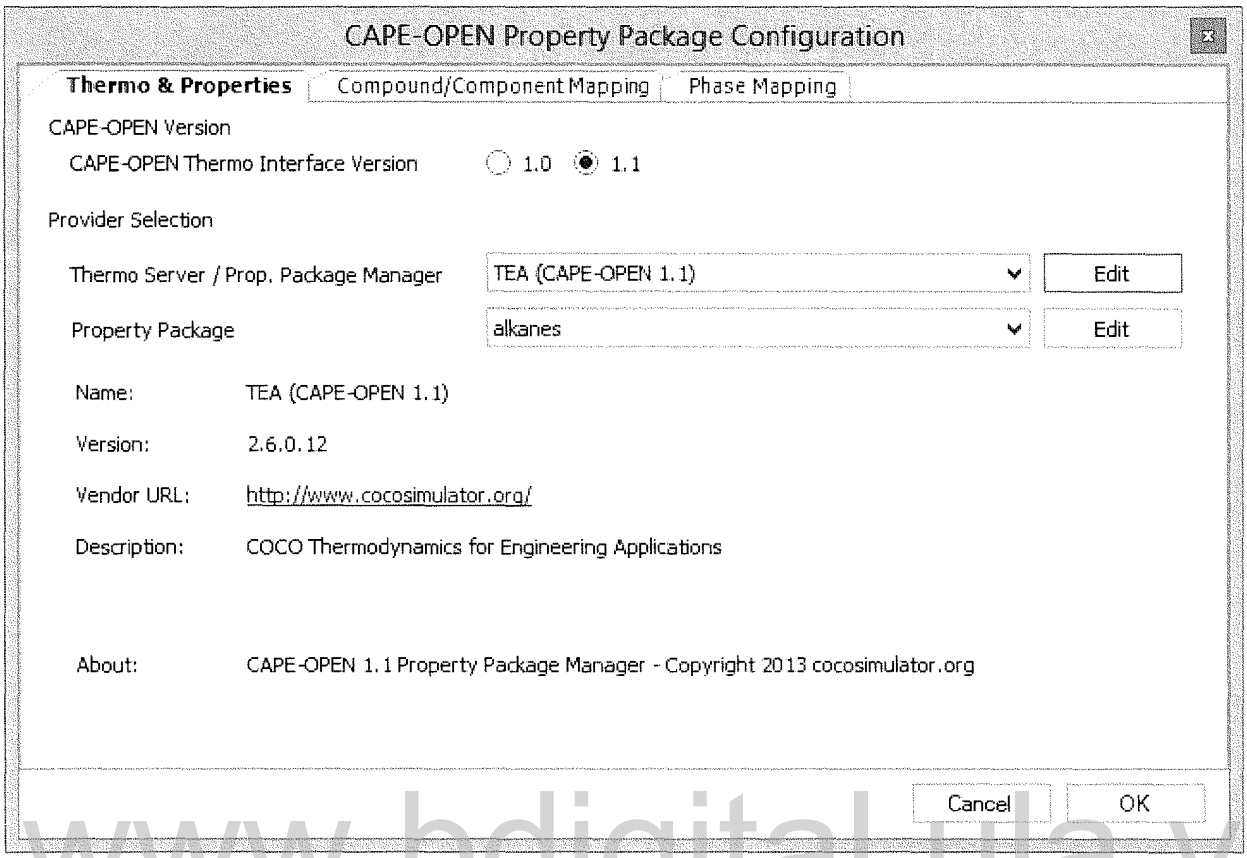
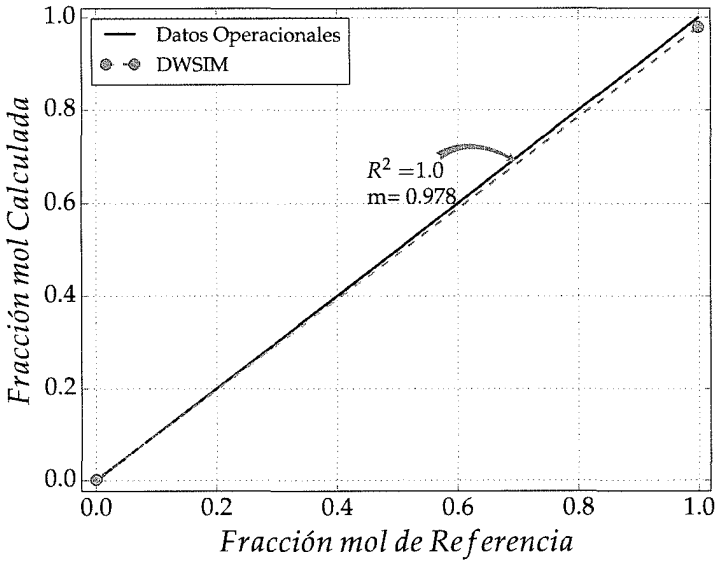


Figura 4.5: Instantánea fotográfica de ventana de configuración del paquete termodinámico CAPE-OPEN TEA en DWSIM

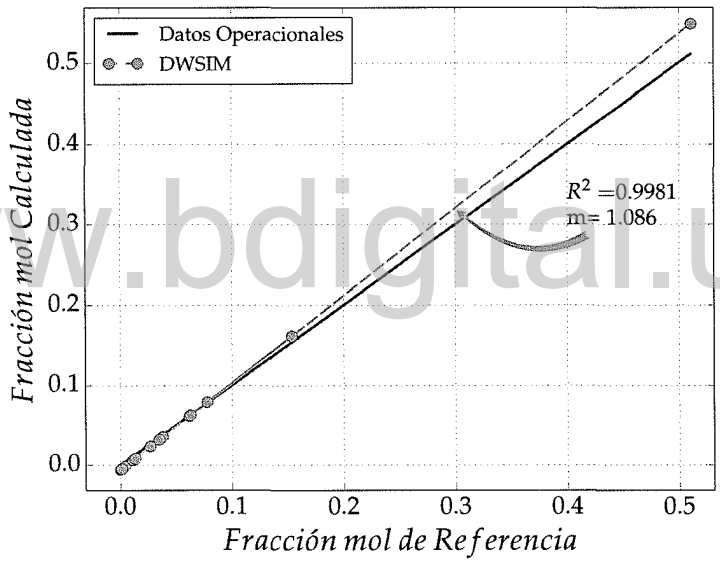
De manera análoga al caso de literatura número 6, se construyen diagramas de paridad para los distintos casos operacionales a partir de los valores tabulados en el apéndice D.2, con la finalidad de realizar una comparación y observar la concordancia de los datos arrojados por el simulador de procesos y los datos operacionales reales. En las Figuras 4.6 y 4.7 se muestran los diagramas respectivos del caso operacional número 1. Al estudiar los diagramas de paridad en ambos escenarios operacionales, se puede comprobar las fracciones molares de los componentes arrojados por DWSIM tanto del producto de tope como en el producto de fondo concuerdan muy bien con los datos medidos en operación, ya que los coeficientes de correlación R^2 y la pendiente m obtenidos de la regresión lineal son muy cercanos a 1.

Tabla 4.3: Características de Caso de Estudio Operacional 1

	Escenario 1	**Escenario 2**
Δ PColumna (psi)	7	6
Número de Etapas de Equilibrio	40	40
Tipo de Condensador	Total 	Total 
Presión de Condensador (psi)	16,7	19,7
Producto de Fondo ($^{lb}/_{h}$)	36725,00	31970,00
Relación de Reflujo	4,4	4,55
Alimentación	1	1
Benceno ($^{lb}/_{h}$)	3343,325	12857,2452
Tolueno ($^{lb}/_{h}$)	18766,662	13131,5033
Etilbenceno($^{lb}/_{h}$)	2266,253	2219,3958
P-xileno ($^{lb}/_{h}$)	2314,302	2266,4504
M-xileno ($^{lb}/_{h}$)	5665,634	5548,4901
O-xileno ($^{lb}/_{h}$)	2854,839	2795,8115
Cumeno ($^{lb}/_{h}$)	104,104	129,7262
1,2,3-Trimetilbenceno ($^{lb}/_{h}$)	1393,386	1736,3359
Nonano ($^{lb}/_{h}$)	4,004	0,0
N-propilbenceno ($^{lb}/_{h}$)	152,151	189,5999
O-Etiltolueno ($^{lb}/_{h}$)	1277,27	1591,6414
Indano ($^{lb}/_{h}$)	28,028	34,9263
N-butilbenceno ($^{lb}/_{h}$)	1009,003	1257,3466
N-pentilbenceno ($^{lb}/_{h}$)	396,394	493,9576
Bifenil ($^{lb}/_{h}$)	460,458	573,7892
Flujo Total ($^{lb}/_{h}$)	40035,81	44826,22
Temperatura (°F)	179,3	243,0
Etapa	17	17
Paquete Termodinámico	PR	PR

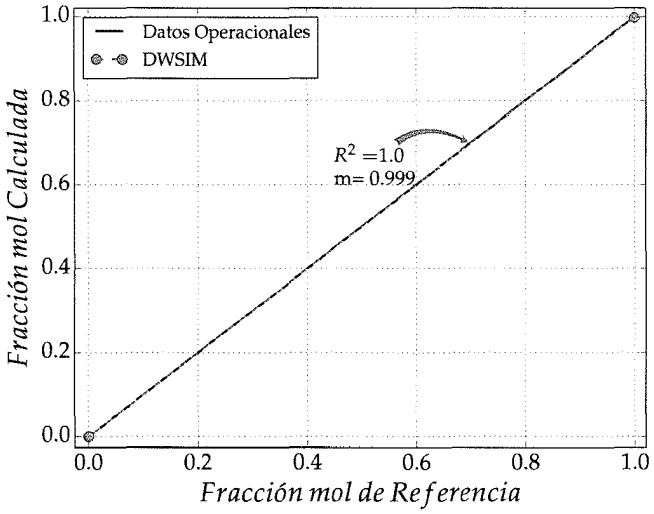


(a) Producto de Tope

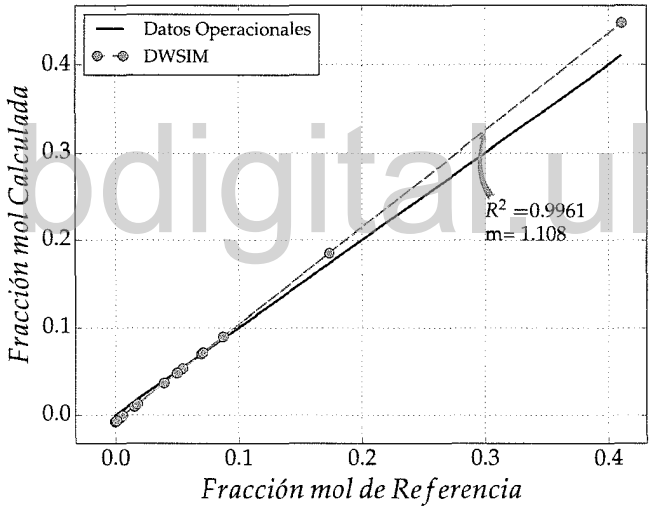


(b) Producto de Fondo

Figura 4.6: Diagramas de Paridad para Caso Operacional 1 Escenario 1



(a) Producto de Tope



(b) Producto de Fondo

Figura 4.7: Diagramas de Paridad para Caso Operacional 1 Escenario 2

Además, para el caso operacional número 1 se muestra en la Tabla 4.4 una comparación entre las temperaturas de tope y fondo predichas por DWSIM y las medidas en operación, así como también los flujos másicos de producto de tope y fondo de la columna de destilación. Para estas variables se observan porcentajes de desviaciones menores a 2 % en ambos escenarios de operación.

Tabla 4.4: Porcentajes de desviación de variables para Caso Operacional 1

(a) Escenario 1

	DWSIM	Operacional	% Desviación
Temp. Producto de Tope (^oF)	182,58	183,210	0,344
Temp. Producto de Fondo (^oF)	281,596	283.540	0,686
Flujo de Producto de Tope ($^{lb}/_h$)	3352,233	3308,983	1,336
Flujo de Producto de Tope ($^{lb}/_h$)	36683,580	36726,859	0,120

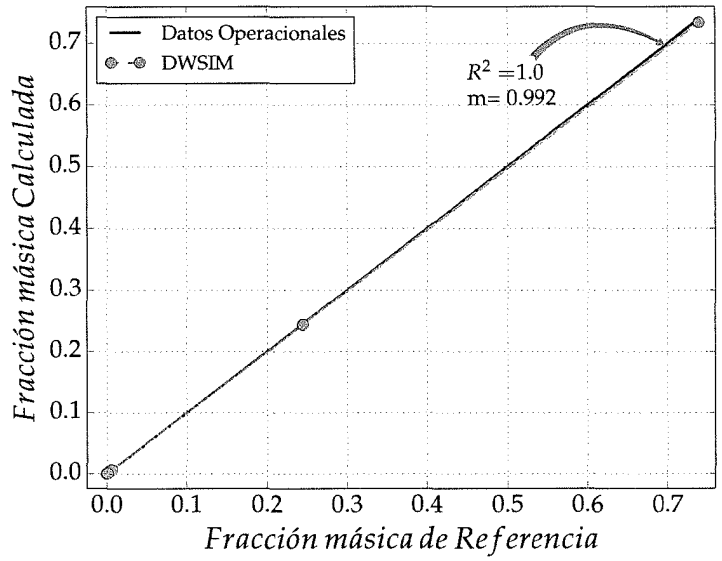
(b) Escenario 2

	DWSIM	Operacional	% Desviación
Temp. Producto de Tope (^oF)	192,315	194,740	1,245
Temp. Producto de Fondo (^oF)	300,234	301,284	0,013
Flujo de Producto de Tope ($^{lb}/_h$)	12877,811	12856,000	0,153
Flujo de Producto de Tope ($^{lb}/_h$)	31950,595	31970,000	0,061

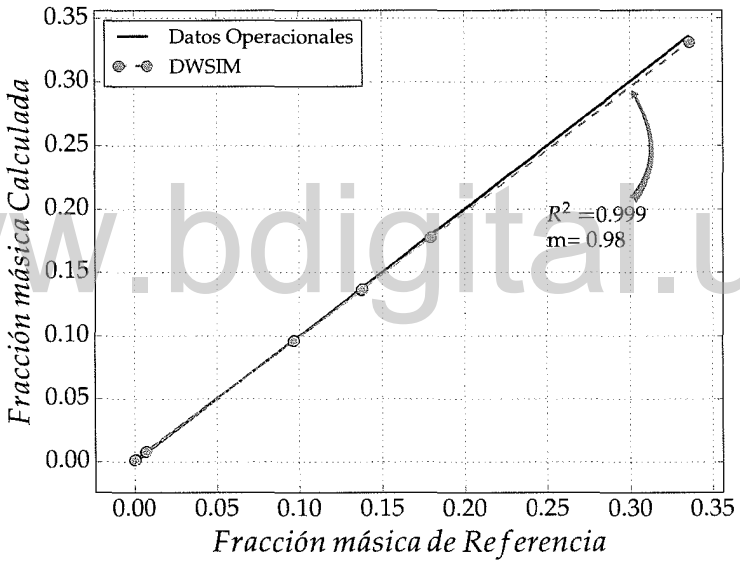
En cuanto al caso operacional número 2, se muestran en la Figura 4.8 los diagramas de paridad donde se comparan las fracciones másicas de los componentes para los productos de tope (4.8a) y fondo (4.8b). De manera similar al caso operacional número 1, se evidencia una buena concordancia de las fracciones másicas predichas por DWSIM respecto a los datos reales, al presentar valores muy cercanos el coeficiente de correlación R^2 y la pendiente m obtenidas de la regresión lineal.

Tabla 4.5: Características de Caso de Estudio Operacional 2

Δ PColumna (kPa)	34,47
Número de Etapas de Equilibrio	30
Tipo de Condensador	Total 
Presión de Condensador (kPa)	1999,48
Producto de Fondo (kg/s)	10,576
% Recuperación Propano	96,0
Alimentación	1
H_2S (kg/s)	0,01946
Etano (kg/s)	0,02199
Etileno (kg/s)	0,00194
Propileno (kg/s)	1,60316
Propano (kg/s)	4,81219
Isobutano (kg/s)	3,553
N-butano (kg/s)	1,01403
1-buteno (kg/s)	1,4392
Isobuteno (kg/s)	1,88237
Trans-2-Buteno (kg/s)	1,44066
Cis-2-Buteno (kg/s)	1,01054
1,3-butadieno (kg/s)	0,07509
3-metilbuteno (kg/s)	0,00576
IsoPentano (kg/s)	0,00597
N-hexano (kg/s)	0,00064
Temperatura (°C)	83
Presión (kPa)	2068
Etapa	14
Paquete Termodinámico	CAPE-OPEN Peng-Robinson



(a) Producto de Tope



(b) Producto de Fondo

Figura 4.8: Diagramas de Paridad para Caso Operacional 2

Por último se realiza la comparación de las temperaturas de tope y fondo junto con los flujos másicos de los productos de la columna del caso operacional número 2. Esta se puede observar en la Tabla (4.6). Se muestra un porcentaje de desviación de 0.45 % para la temperatura del fondo de la columna seguido de 2,948 % y 2,395 % para los flujos de producto de tope y fondo respectivamente y finalmente para la temperatura de tope de la columna con una desviación significativa del 15,25 %. A pesar de presentar una desviación

con una magnitud mayor al 10 % para la predicción de la temperatura del tope de la columna, el modelo representa una descripción razonable de la columna depropanizadora al predecir con bastante exactitud las demás variables de operación y las fracciones másicas correspondiente a los productos.

Tabla 4.6: Porcentajes de desviación de variables para Caso Operacional 2

	DWSIM	Operacional	% Desviación
Temp. de Tope (^oC)	53,750	46,636	15,254
Temp. de Fondo (^oC)	104,475	104,950	0,453
Flujo de Producto de Tope ($^{kg}/_{s}$)	6,310	6,129	2,948
Flujo de Producto de Tope ($^{kg}/_{s}$)	10,576	10,329	2,395

En definitiva, estos resultados muestran que el modelo Inside-Out de Russell (1983) basado en el modelo de etapa de equilibrio implementado en DWSIM es capaz de reproducir razonablemente los datos de las columnas de fraccionamiento descritas en los casos operacionales. Ademas, se confirma de manera más contundente la correcta implementación del mismo dentro de DWSIM.

Capítulo 5

Conclusiones y Recomendaciones

Conclusiones

1. Se logró identificar los distintos elementos que conforman el código fuente de DWSIM a través de una metodología de ingeniería inversa. Esto permitió identificar tanto la arquitectura general del simulador y las clases específicas involucradas con la simulación de columnas de destilación en estado estacionario para ser modificadas.

2. Se construyeron los diagramas de actividades detallados que describen el algoritmo del método Inside-Out de Russell (1983) para destilación multicomponente. Estos favorecieron la comprensión del método y auditoria del código fuente al permitir identificar los problemas y las oportunidades de mejora.

3. Se realizó con éxito la adaptación de DWSIM para simular columnas de destilación reales pertenecientes al circuito nacional de refinación utilizando Inside-Out. Esto se comprobó al evidenciar su reproducibilidad junto con otros simuladores de procesos con casos de estudio de referencia y su eficacia en las predicciones obtenidas con las simulaciones estuvieron concordancia con los datos operacionales.

4. La compatibilidad de DWSIM con los estándares CAPE-OPEN brindó la posibilidad de incorporar paquetes termodinámico externo en escenarios donde existía baja calidad de los cálculos de EVL.

5. La adaptación de DWSIM se llevó a cabo a través de la participación de la academia, la industria y la comunidad de software libre, empleando una metodología de desarrollo de software colaborativo. A través del uso de la plataforma en linea GitHub empleando el sistema de control de versiones distribuido GIT, lográndose gestionar los diversos cambios que se realizaron sobre el código fuente de DWSIM.

6. El simulador de procesos DWSIM posibilita su uso como herramienta de aprendizaje para los estudiantes de ingeniería química e ingenieros químicos, de manera que permite comprender mejor la metodología de cálculos riguroso no solo del proceso de destilación, sino de otra operación unitaria disponible sin costo alguno fomentando el desarrollo de tecnologías propias.

Recomendaciones

1. Estudiar los métodos termodinámicos nativos de DWSIM, ya que presentan un desempeño no satisfactorio en algunos cálculos de EVL.

2. Adaptar el método Inside-Out para el soporte de reflujos circulantes (pumparounds), ya que son de suma importancia en el modelaje y simulación para muchos tipos de columnas de destilación en el circuito nacional de refinación de PDVSA.

3. Mejorar la herramienta para la creación de pseudocomponentes de DWSIM, ya que se considera herramienta útil para la simulación de procesos de refinación para modelar las corrientes de procesos.

4. En la implementación realizada se pueden presentar límites erróneos en las variables de iteración, que a su vez puede dar lugar a divergencias, especialmente para problemas de columnas de destilación complejas o problemas con estimaciones iniciales deficientes. Por esta razón se recomienda realizar un estudio de optimización para robustecer el algoritmo con técnicas que proporcionen garantías tanto matemáticas y computacionales más rigurosas.

5. Es importante resaltar que nuevas investigaciones deben ser orientadas hacia el mejoramiento de DWSIM, por lo que se recomienda incentivar este tipo de proyecto de investigación a través de la colaboración entre la academia, la industria y la comunidad del software libre. En este sentido, las universidades y cualquier otra entidad con capacidad de investigación y desarrollo pueden realizar grandes aportes enfocados en la mejora del simulador, fomentando la cooperación entre los sectores productivos y educación a nivel nacional e internacional.

Referencias Bibliográficas

Abella, A., Sanchez, J., Santos, R., & Segovia, M. (2003). *Libro Blanco del Software Libre en España*. España: Documentación Libre GNU.

Aguilar, L. J. (2003). *Fundamentos de Programación: Algoritmos, Estructuras de Datos y Objetos*. McGraw-Hill.

Babu, B. V. (2004). *Process Plant Simulation*. New Delhi, India; New York: Oxford University Press.

Barrett, J. & Yang, J. (2005). Development of a chemical process modeling environment based on cape-open interface standards and the microsoft .net framework. *Computers & Chemical Engineering*, 30(2), 191–201.

Barrett, W. M., Pons, M., von Wedel, L., & Braunschweig, B. (2007). An overview of the interoperability roadmap for COM/.NET-based CAPE-OPEN.

Boston, J. F. & Sullivan, S. L. (1972). An improved algorithm for solving the mass balance equations in multistage separation processes. *The Canadian Journal of Chemical Engineering*, 50, 663–669.

Boston, J. F. & Sullivan, S. L. (1974). A new class of solution methods for multicomponent, multistage separation processes. *The Canadian Journal of Chemical Engineering*, 52(1), 52–63.

Braunschweig, B. & Gani, R. (2002). *Computer-Aided Chemical Engineering: Software Architectures and Tools for Computer Aided Process Engineering*, volume 11. Amsterdam, The Netherlands: ELSEVIER SCIENCE B.V.

Broyden, C. G. (1965). A class of methods for solving nonlinear simultaneous equations. *Mathematics of Computation*, 19(92), 577–593.

Burningham, D. & Otto, F. (1967). Which computer design for absorbers? *Hydrocarbon Process*, 46(10), 163–170.

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *A Pattern language for Distributed Computing*. Pattern-oriented software architecture / Frank Buschmann ..., Vol. 4.; Wiley series in software design patterns.; Wiley software patterns series. Wiley.

CO-LaN (1999). *CAPE-OPEN Next Generation Computer Aided Process Engineering Open Simulation Environment*. Technical report, CAPE-OPEN Consortium.

Culebro, M., Gómez, W., & Torres, S. (2006). *Software Libre vs Software Propietario Ventajas y Desventajas*. México: Creative Commons.

Dimian, A. C. (2003). *Integrated Design and Simulation of Chemical Processes*. Amsterdam; Boston: Elsevier.

Eckert, E. & Vaněk, T. (2001). Some aspects of rate-based modelling and simulation of three-phase distillation columns. *Computers & Chemical Engineering*, 25(4–6), 603–612.

Finlayson, B. A. (2006). *Introduction to Chemical Engineering Computing*. Hoboken, N.J.: Wiley-Interscience.

Grossmann, I. E., Aguirre, P. A., & Barttfeld, M. (2005). Optimal synthesis of complex distillation columns using rigorous models. *Computers & Chemical Engineering*, 29(6), 1203–1215.

Higler, A., Chande, R., Taylor, R., Baur, R., & Krishna, R. (2004). Nonequilibrium modeling of three-phase distillation. *Computers & Chemical Engineering*, 28(10), 2021–2036.

Khoury, F. M. (2005). *Multistage Separation Processes*. Boca Raton: CRC Press.

King, C. J. (1980). *Separation processes*. New York: McGraw-Hill.

Kister, H. Z. (1992). *Distillation Design*. New York: McGraw-Hill.

Kooijman, H. A. & Taylor, R. (2000). *The ChemSep Book*. Germany: Printed by Books on Demand, 2 edition.

Krishnamurthy, R. & Taylor, R. (1985). A nonequilibrium stage model of multicomponent separation processes. part i: Model description and method of solution. *AIChE Journal*, 31(3), 449–456.

Kroll, P. & Kruchten, P. (2003). *The Rational Unified Process Made Easy : A Practitioner's Guide to the Rup*. Addison-Wesley object technology series. Addison-Wesley.

Luque, S. & Vega, A. B. (2005). *Simulación y optimización avanzadas en la industria química y de procesos*. Oviedo: Departamento de Ingeniería Química y Tecnología del Medio Ambiente, Universidad de Oviedo.

Martínez, V. H. (2000). *Simulación de Procesos en Ingeniería Química*. Plaza y Valdes.

Martínez de la Cuesta, P. J. & Rus, E. (2004). *Operaciones de separación en ingeniería química*. Madrid: Prentice Hall.

MathPro (2011). Introducción a la refinación del petróleo y producción de gasolina y diesel con contenido ultra bajo de azufre. http://goo.gl/VoA8qt.

McCabe, W. L., Smith, J. C., & Harriott, P. (2005). *Unit Operations of Chemical Engineering*. Boston: McGraw-Hill.

McMillan, M. (2004). *Object-oriented Programming with Visual Basic.NET*.

McMonnies, A. (2004). *Object-oriented Programming in Visual Basic .net*. Harlow, England; New York: Pearson Addison-Wesley.

Monroy-Loperena, R. & Vacahern, M. (2013). A simple, reliable and fast algorithm for the simulation of multicomponent distillation columns. *Chemical Engineering Research and Design*, 91(3), 389 – 395.

Naphtali, L. M. & Sandholm, D. P. (1971). Multicomponent separation calculations by linearization. *AIChE Journal*, 17, 148–153.

O'Connell, J. & Haile, J. (2005). *Thermodynamics: Fundamentals for Applications*. Cambridge Series in Chemical Engineering. Cambridge University Press.

PDVSA (2005). Sitio web pdvsa. http://goo.gl/j09zD.

Pernalete, C. (2013). Un agente inteligente para la optimización en lÃnea del proceso de mezclas de gasolina. Master's thesis, Universidad de los Andes.

Pernalete, C., Torres, B., & Moreno, A. (2012). Reactores de la tecnología HDHPLUS® como componentes de simulación interoperables. *Ciencia e Ingeniería*.

Perry, R. H. & Green, D. W. (2008). *Perry's chemical engineers' handbook*. New York: McGraw-Hill, 7 edition.

Pons, M. (2005). What is the cape-open laboratories network (CO-LaN).

Russell, R. A. (1983). A flexible and reliable method solves single-tower and crude distillation-column problems. *Chemical Engineering*, 90(1), 53–59.

Scenna, N. J., Aguirre, P. A., Benz, S. J., & Chiotti, O. J. (1999). *Modelado, Simulación y Optimización de Procesos Químicos*.

Seader, J. D., Henley, E. J., & Roper, D. K. (2011). *Separation Process Principles : Chemical and Biochemical Operations*. Hoboken, NJ: Wiley.

Sorel, E. (1893). La rectification de l'alcool. In *Encyclopedie Scientifique Des Aide-memoire*. Gauthier-Villars et fils.

Speight, J. G. (2008). *Synthetic fuels handbook properties, process, and performance*. New York: McGraw-Hill.

Taylor, J. R. (2007). Understanding and combating design error in process plant design. *Safety Science*, 45(1-2), 75 – 105. Safety by Design Based on a workshop of the New Technology and Work Network.

Tomich, J. F. (1970). A new simulation method for equilibrium stage processes. *AIChE Journal*, 16(2), 229–232.

Tonella, P. & Potrich, A. (2005). *Reverse Engineering of Object Oriented Code*. Monographs in computer science. Springer Science+Business Media.

Torres, R. & Castro, J. (2002). *Análisis y Simulación de Procesos de Refinación del Petróleo*. México: Alfaomega.

Treccani, P. (2008). *Encyclopaedia of Hydrocarbons: VOLUME II - REFINING AND PETROCHEMICALS*, volume 2. Istituto della Enciclopedia Italiana Iondata da Giovanni Treccani S.p.A.

Wagner, D. (2006). Dwsim development blog: Notes and discussions about dwsim development. http://goo.gl/8sN4A4.

Wang, J. C. & Henke, G. E. (1966). Tridiagonal matrix for distillation. *Hydrocarbon Process*, 45, 155–163.

Wayburn, T. L. (1988). A review of continuation methods and their application to separation problems. *Comp. and Sys. Tech. Div. Commun.*, 11(1), –.

Wilson, I. D., Adlard, E. R., Cooke, M., & Poole, C. F. (2000). *Encyclopedia of Separation Science*. San Diego: Academic Press.

Zeigler, B. P., Kim, T. G., & Praehofer, H. (2000). *Theory of Modeling and Simulation*. Orlando, FL, USA: Academic Press, Inc., 2nd edition.

Zereshki, S. (2012). *Distillation - Advances from Modeling to Applications*. InTech.

Apéndice A

Métodos Numéricos

A.1. Método de Matriz Tridiagonal

El algoritmo de la matriz Tridiagonal (TDMA), también conocido como el algoritmo de Thomas, es una forma simplificada de la eliminación gaussiana que puede utilizarse para resolver un sistema tridiagonal de ecuaciones.

$$A_k x_{k-1} + B_k x_k + C_k x_{k+1} = D_k, \quad k = 1, ...n, \tag{A.1}$$

o, en forma de ecuación matricial $Mx = D$ donde $A_1 = 0$ y $C_n = 0$ (A.2).

$$\begin{pmatrix} B_1 & C_1 & \cdots & \cdots & \cdots & & \cdots & 0 \\ A_2 & B_2 & C_2 & \cdots & \cdots & & \cdots & 0 \\ \cdots & A_3 & B_3 & C_3 & \cdots & & \cdots & 0 \\ \cdots & \cdots & \ddots & \ddots & \ddots & & & \cdots \\ \cdots & & & \ddots & \ddots & \ddots & & \cdots \\ 0 & \cdots & \cdots & 0 & A_{n-2} & B_{n-2} & C_{n-2} & \cdots \\ 0 & \cdots & \cdots & 0 & 0 & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & & A_n & B_n \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ \cdots \\ \cdots \\ D_{n-2} \\ D_{n-1} \\ D_n \end{pmatrix} \tag{A.2}$$

La Matriz principal es una matriz cuyos únicos elementos distintos de cero se encuentran en la diagonal principal y en las diagonales adyacentes por arriba y por debajo de esta.

El TDMA constan de dos partes: una fase de eliminación hacia delante y una fase de sustitución hacia atrás, con la finalidad de resolver o determinar los valores de x_k.

1. Fase de Eliminación

a) Es esta primera fase la ecuación A.2 se convierte a la forma $Ux = \rho$. Así el primer paso es dividir la fila 1 por B_1 para obtener x_1 en función de x_2:

$$B_1 x_1 + C_1 x_2 = D_1$$

$$x_1 + \frac{C_1}{B_1} x_2 = \frac{D_1}{B_1}$$

Reescribiendo

$$x_1 + \gamma_1 x_2 = \rho_1, \qquad , \gamma_1 = \frac{C_1}{B_1}, \qquad , \rho_1 = \frac{D_1}{B_1}$$

$$x_1 = \rho_1 - \gamma_1 x_2 \tag{A.3}$$

Reemplazando así los coeficientes B_1, C_1 y D_1

$$\begin{pmatrix} 1 & \gamma_1 & \cdots & \cdots & \cdots & & \cdots & 0 \\ A_2 & B_2 & C_2 & \cdots & \cdots & & \cdots & 0 \\ \cdots & A_3 & B_3 & C_3 & \cdots & & \cdots & 0 \\ \cdots & \cdots & \ddots & \ddots & \ddots & & & \cdots \\ \cdots & & & \ddots & \ddots & \ddots & & \cdots \\ 0 & \cdots & \cdots & 0 & A_{n-2} & B_{n-2} & C_{n-2} & \cdots \\ 0 & \cdots & \cdots & 0 & 0 & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & & A_n & B_n \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} \rho_1 \\ D_2 \\ D_3 \\ \cdots \\ \cdots \\ D_{n-2} \\ D_{n-1} \\ D_n \end{pmatrix} \tag{A.4}$$

b) Luego la ecuación A.1 se combina con A.3 para despejar x_2 resultando en:

$$x_2 = \frac{D_2 - A_2 \rho_1}{B_2 - A_2 \gamma_1} - \left(\frac{C_2}{B_2 - A_2 \gamma_1} \right) x_3$$

reescribiendo

$$\rho_2 = \frac{D_2 - A_2 \rho_1}{B_2 - A_2 \gamma_1} \qquad \gamma_2 = \frac{C_2}{B_2 - A_2 \gamma_1},$$

quedando:

$$x_2 = \rho_2 - \gamma_2 x_3 \tag{A.5}$$

Así, se reemplazan los valores de los coeficientes A_2, B_2,C_2 y D_2

$$\begin{pmatrix} 1| & \gamma_1 & \cdots & \cdots & \cdots & & \cdots & 0 \\ 0 & 1 & \gamma_2 & \cdots & \cdots & & \cdots & 0 \\ \cdots & A_3 & B_3 & C_3 & \cdots & & \cdots & 0 \\ \cdots & \cdots & \ddots & \ddots & \ddots & & & \cdots \\ \cdots & & & \ddots & \ddots & \ddots & & \cdots \\ 0 & \cdots & \cdots & 0 & A_{n-2} & B_{n-2} & C_{n-2} & \cdots \\ 0 & \cdots & \cdots & 0 & 0 & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & & A_n & B_n \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ D_3 \\ \cdots \\ \cdots \\ D_{n-2} \\ D_{n-1} \\ D_n \end{pmatrix} \tag{A.6}$$

c) Siguiendo esta secuencia, en general se puede decir:

$$\gamma_k = \frac{C_k}{B_k - A_k\gamma_{k-1}} \tag{A.7}$$

$$\rho_k = \frac{D_k - A_k\rho_{k-1}}{B_k - A_k\gamma_{k-1}} \tag{A.8}$$

$$x_k = \gamma_k - \rho_k x_{k+1} \tag{A.9}$$

En este punto, con los valores sustituidos $A_k \leftarrow 0$, $B_k \leftarrow 1$,$C_k \leftarrow \gamma_k$ y $D_k \leftarrow \rho_k$ la matriz se ha reducido una forma diagonal superior, por lo que las ecuaciones quedan en la forma $Ux = \rho$ (A.10)

$$\begin{pmatrix} 1| & \gamma_1 & \cdots & \cdots & \cdots & & \cdots & 0 \\ 0 & 1 & \gamma_2 & \cdots & \cdots & & \cdots & 0 \\ \cdots & 0 & 1 & \gamma_3 & \cdots & & \cdots & 0 \\ \cdots & \cdots & \ddots & \ddots & \ddots & & & \cdots \\ \cdots & & & \ddots & \ddots & \ddots & & \cdots \\ 0 & \cdots & \cdots & 0 & 0 & 1 & \gamma_{n-2} & \cdots \\ 0 & \cdots & \cdots & 0 & 0 & 0 & 1 & \gamma_{n-1} \\ 0 & \cdots & \cdots & 0 & 0 & & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \cdots \\ \cdots \\ x_{n-2} \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \\ \cdots \\ \cdots \\ \rho_{n-2} \\ \rho_{n-1} \\ \rho_n \end{pmatrix} \tag{A.10}$$

2. Fase de Sustitución Hacia atrás

a) A partir de A.10 para la fila n , (A.9) queda:

$$x_n = \rho_n \tag{A.11}$$

b) Los valores sucesivos de x_k se calculan recursivamente mediante sustitución haca atrás a partir de (A.9), desde la fila $n-1$ hasta 1, así para la fila $n-1$:

$$x_{n-1} = \gamma_{n-1} - \rho_{n-1} x_n \tag{A.12}$$

El algoritmo de Thomas evita la acumulación de errores de truncamiento porque ninguno de los pasos que implica la sustracción de cantidades casi iguales. Además, los valores calculados de x_k son casi siempre positivos. El algoritmo es superior a las rutinas alternativas de inversión de matrices. Aunque es raro, el algoritmo puede ser inestable si el término $B_k - A_k \gamma_{k-1}$ es cero o numéricamente cero para cualquier k. Esto ocurrirá si la matriz tridiagonal es singular, pero en casos muy específicos puede ocurrir si es no singular. La condición para el algoritmo sea estable es que la matriz sea diagonalmente dominante, esto es:

$$\|B_k\| = \|A_k\| + \|C_k\| \tag{A.13}$$

A.2. Método de Newton-Raphson

En análisis numérico, el método de Newton-Raphson es un algoritmo eficiente para encontrar aproximaciones de las raíces de una función real. Se puede utilizar el método de Newton para resolver sistemas de n ecuaciones no lineales, lo que equivale a la búsqueda de los ceros de funciones continuamente diferenciables.

Este tipo de sistemas se pueden representar como $F(X) = 0$ donde 0 es el vector nulo de n componentes, X es un vector de $\mathbb{R}$ y F es la función vectorial dependiente de n variables reales dada por:

$$F \ : \ \mathbb{R}^n \rightarrow \mathbb{R}^n$$

$$X = (x_1, x_2, \ldots, x_n)^T \ \rightarrow \ F(X) = \left\{ \begin{array}{c} f_1(x_1, x_2, ..., x_n) = 0 \\ f_2(x_1, x_2, ..., x_n) = 0 \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ f_n(x_1, x_2, ..., x_n) = 0 \end{array} \right\} \tag{A.14}$$

La iteración de Newton-Raphson se lleva a cabo mediante la resolución de correcciones de la ecuación matricial para el valor ΔX de las variables de salida de la forma :

$$\left(\frac{\partial F}{\partial X}\right) \cdot \Delta X^{(k)} = -F^{(k)} \tag{A.15}$$

donde $\dfrac{\partial F}{\partial X}$ es el siguiente Jacobiano $(n \times n)$:

$$\frac{\partial F}{\partial X} = J_F = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{A.16}$$

$$\Delta X^{(k)} = -\left[\left(\frac{\partial F}{\partial X}\right)^{-1}\right]^{(k)} F^{(k)} \tag{A.17}$$

Las correcciones ΔX se utilizan para calcular la siguiente aproximación para el conjunto de variables de salida a partir de:

$$X^{(k+1)} = X^{(k)} + \Delta X^{(k)} \tag{A.18}$$

En este sentido el conjunto de funciones $F(X)$ se resuelven simultáneamente de forma iterativa mediante el cálculo de un conjunto de valores X sucesivos hasta que los valores de $F(X)$ se dirigan hacia un criterio de convergencia o valores cero. Durante las iteraciones, los valores distintos de cero de las funciones se denominan discrepancias o errores.

El procedimiento de Newton–Raphson requiere estimaciones iniciales para los valores de las variables de iteración. Basados en estos valores iniciales, se compara la suma de los cuadrados de las discrepancias o errores de las funciones con el criterio de convergencia τ para una tolerancia ϵ:

$$\tau = \sum_{i=1}^{n} (F_i)^2 \leq \epsilon \tag{A.19}$$

$$\epsilon = n \left(\sum_{i=1}^{n} (F_i)^2\right) 10^{-10} \tag{A.20}$$

Alternativamente, es posible emplear un factor escalar no negativo t denominado factor de amortiguamiento, que se puede aplicar a los valores de las variables en cada iteración.

$$X^{(k+1)} = X^{(k)} + t\Delta X^{(k)} \tag{A.21}$$

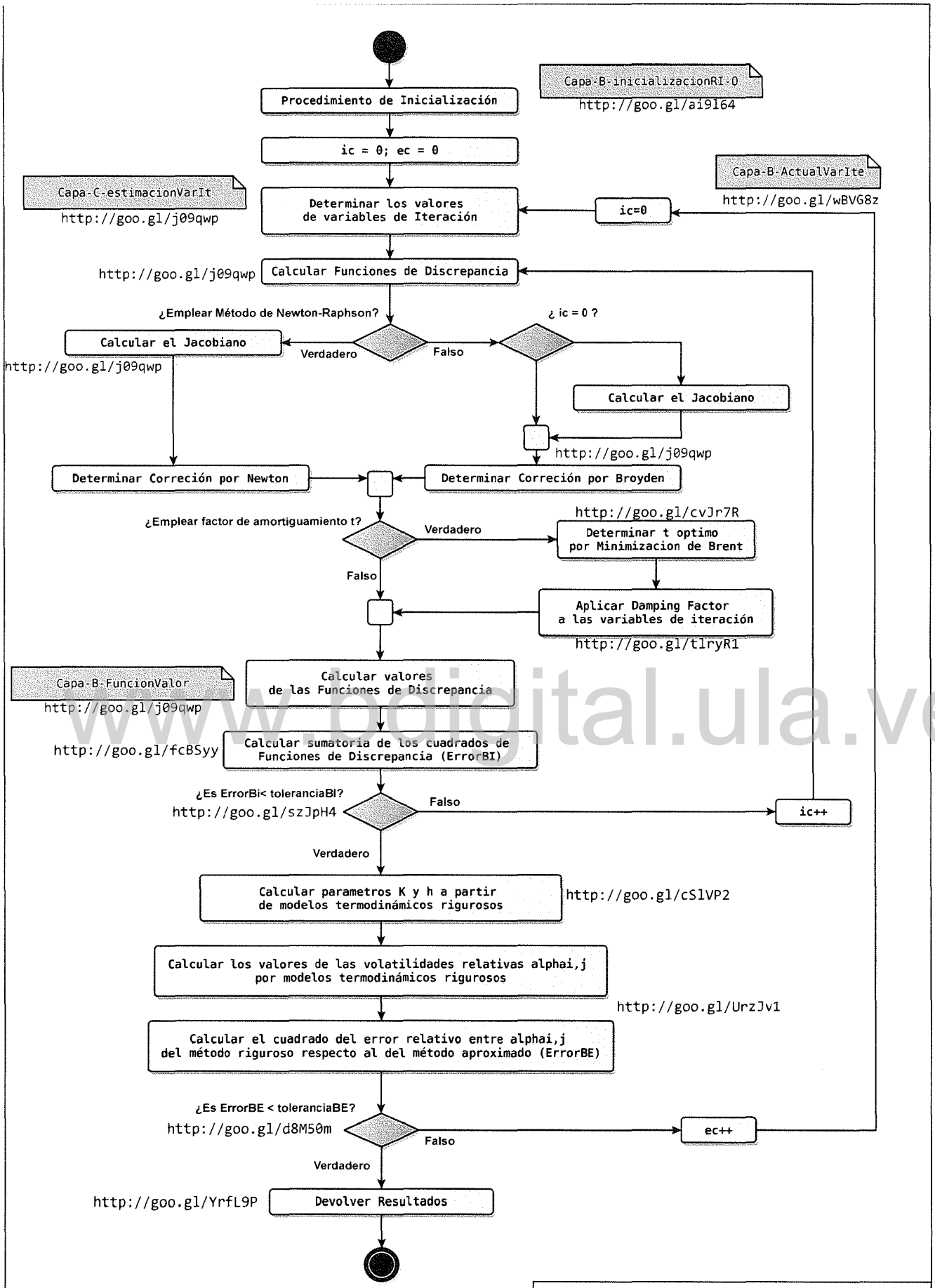
Haciendo que t varíe entre valores ligeramente mayores a 0 hasta 2, se puede amortiguar o acelerar la convergencia, según se requiera para cada iteración. Lo que se requiere es un valor de t que minimice la suma de los cuadrados dado por (A.19). Para encontrar t

en cada iteración se emplea un procedimiento de optimización como búsqueda de Fibonnacci o minimización de Brent. De no existir un valor optimo de t, este puede tomar un valor de 1

Apéndice B

Diagramas de Actividades del Método Inside-Out Russell (1983)

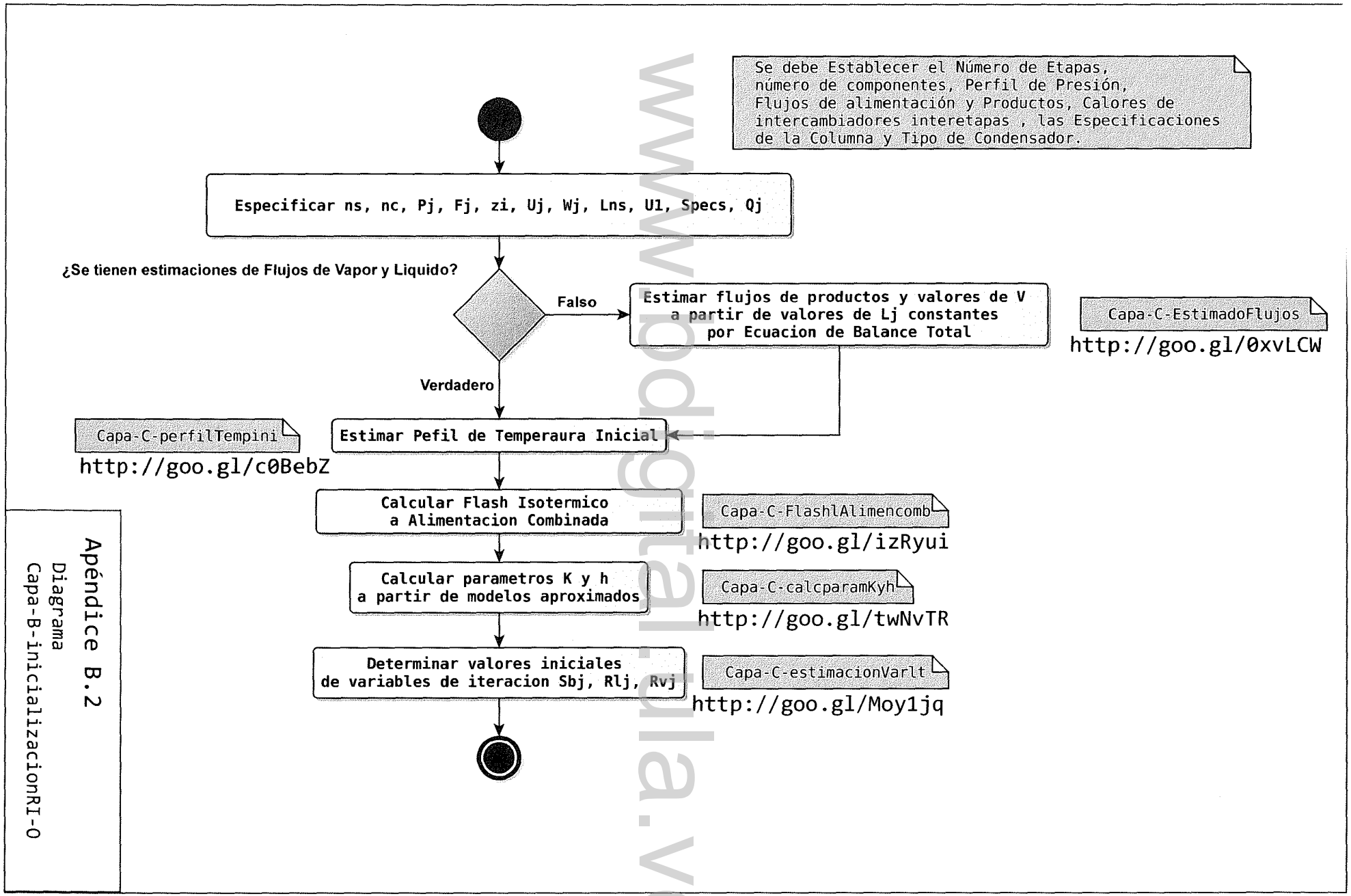
B.1. Diagramas de Actividades Capa-A



Apéndice B.1

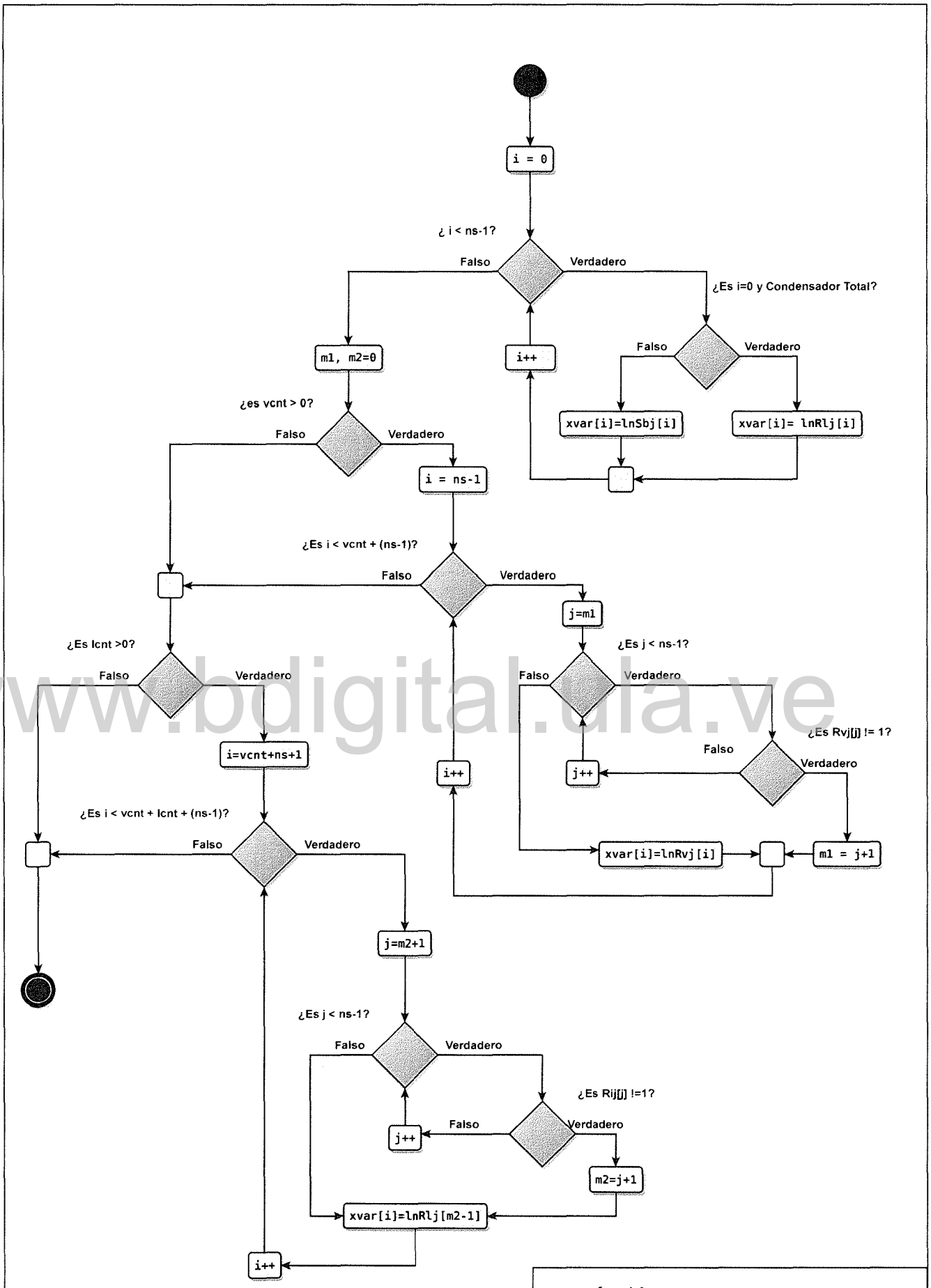
Diagrama Capa-A-Russell

B.2. Diagramas de Actividades Capa-B



Apéndice B.2

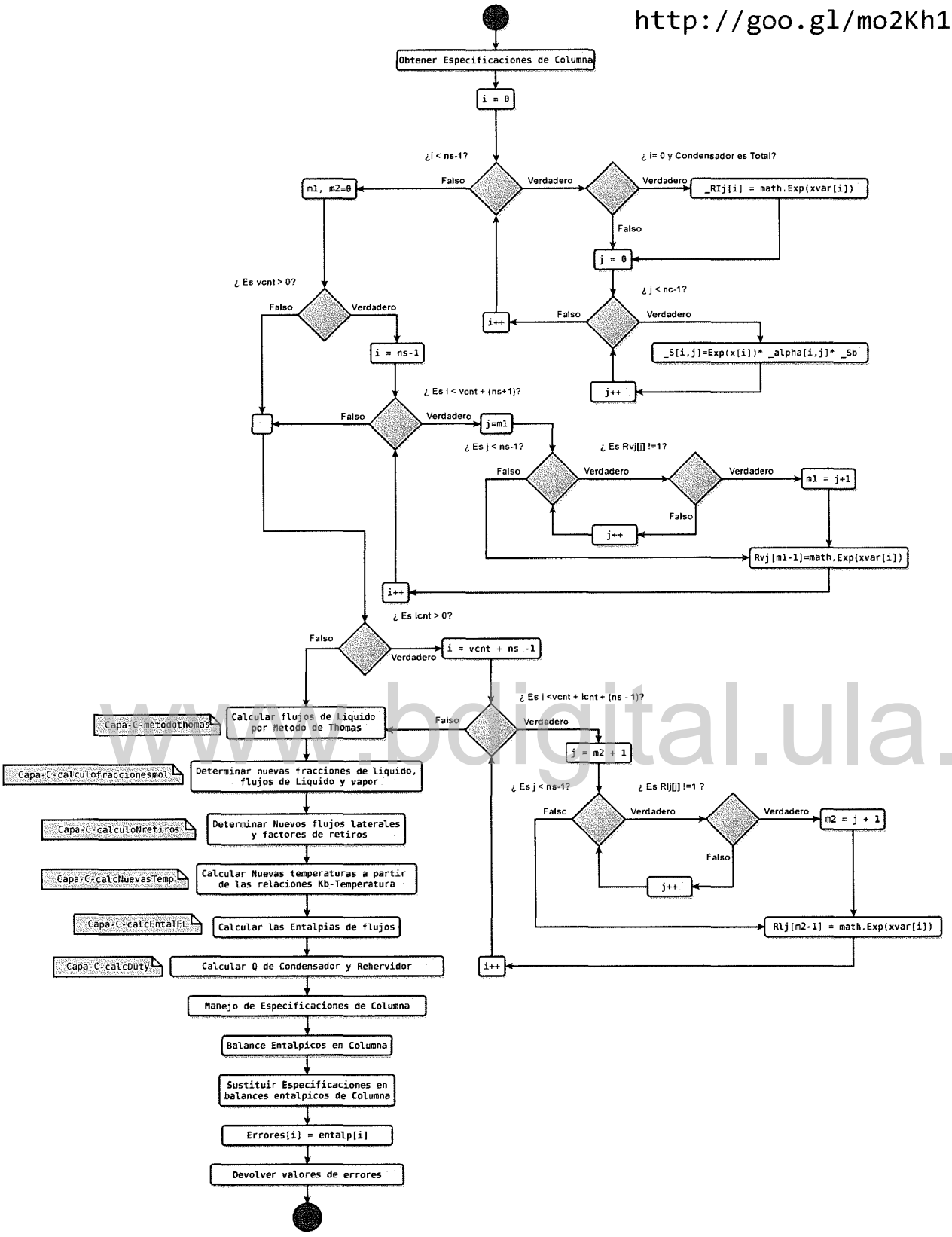
Diagrama Capa-B-inicializacionRI-O



Apéndice B.2

Diagrama
Capa-B-ActualVarIte

http://goo.gl/mo2Kh1

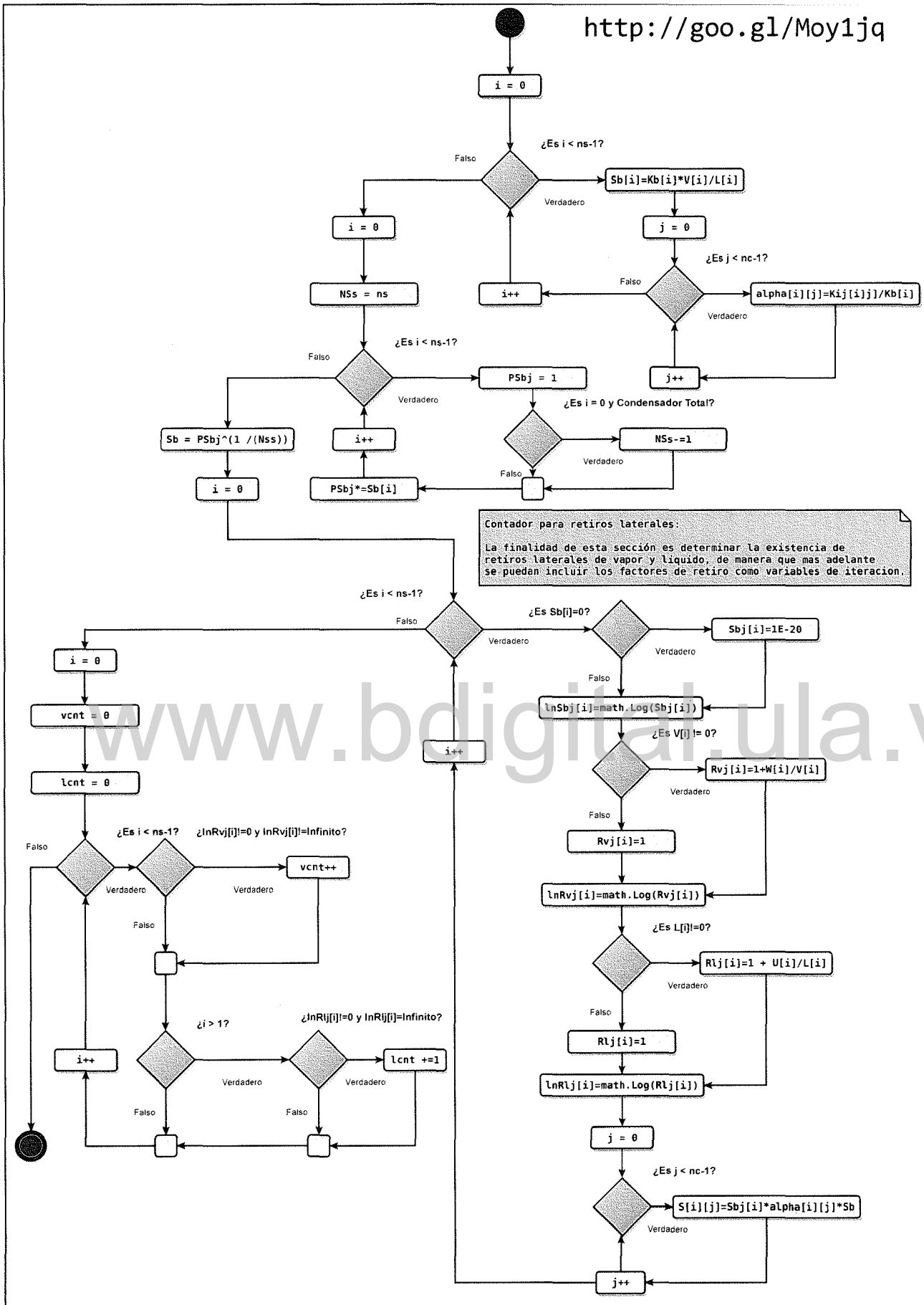


Apéndice B.2

Diagrama
Capa-B-FuncionValor

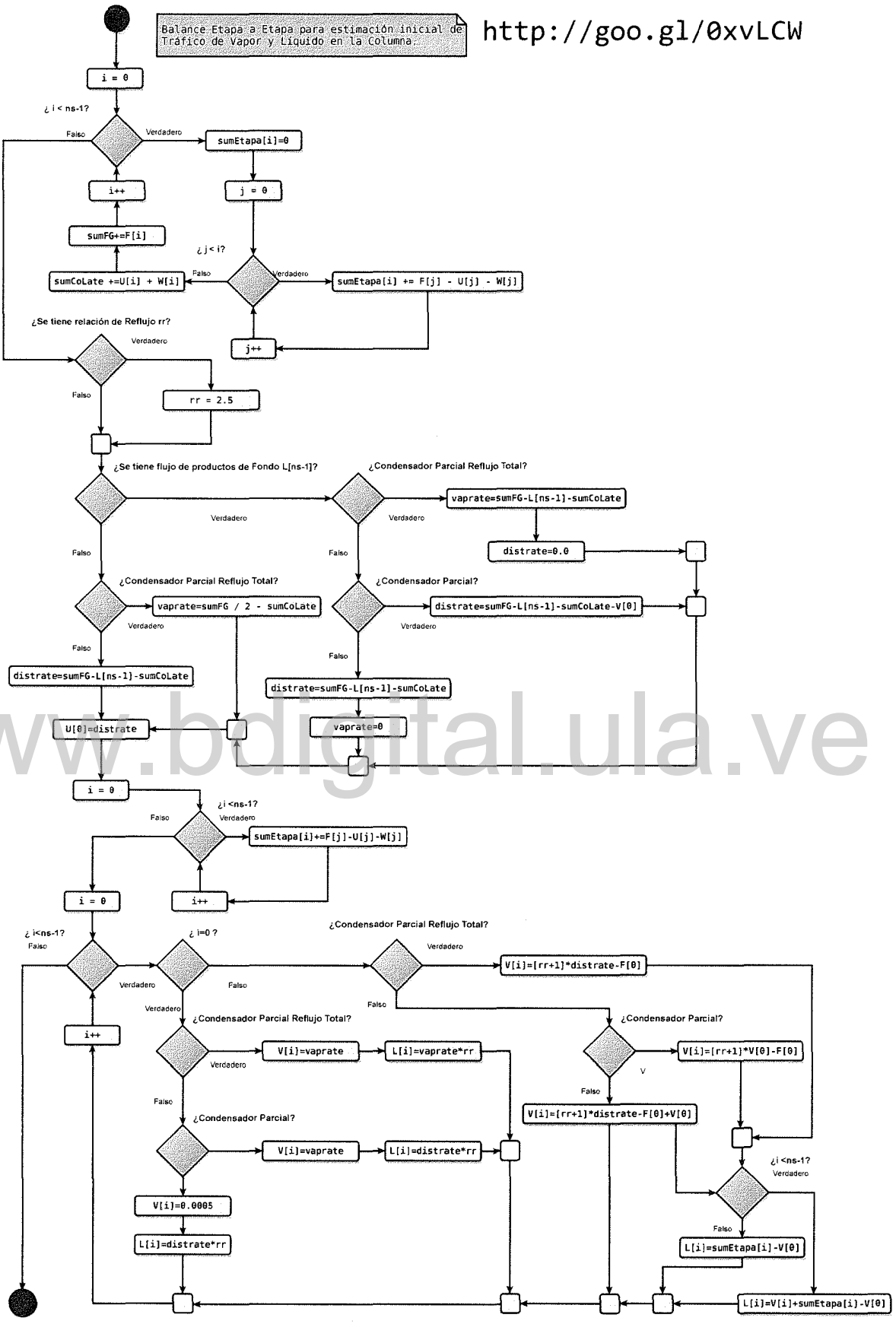
B.3. Diagramas de Actividades Capa-C

http://goo.gl/Moy1jq



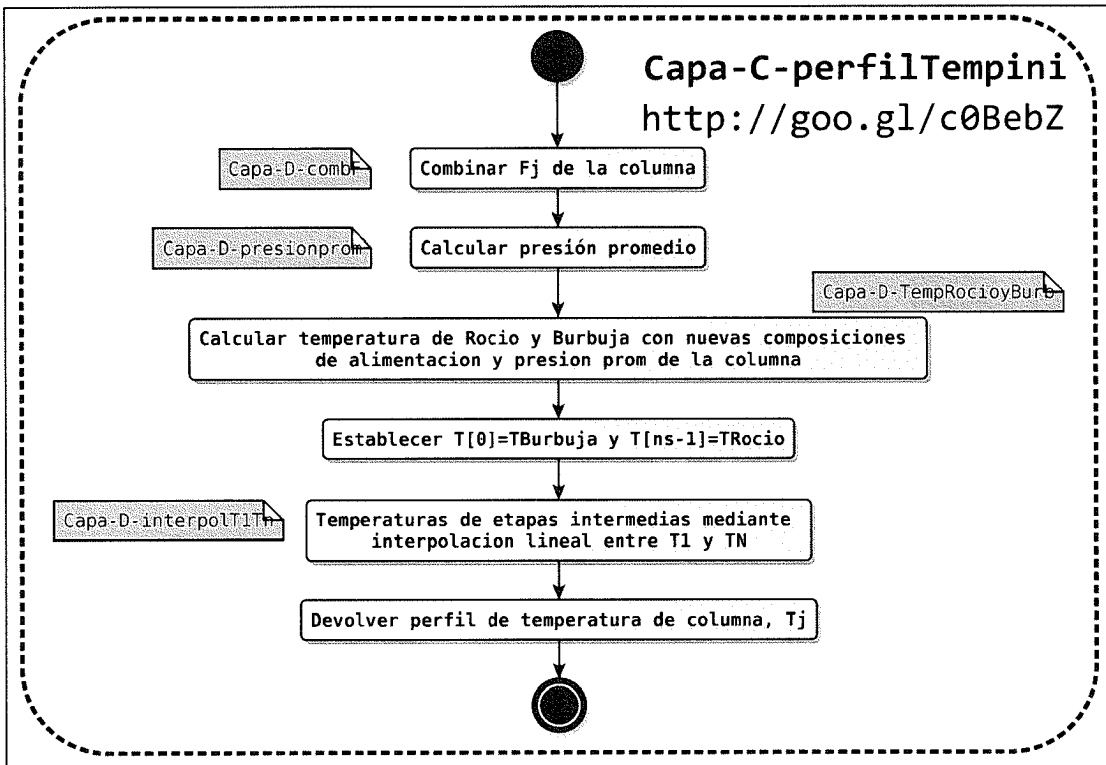
Apéndice B.3

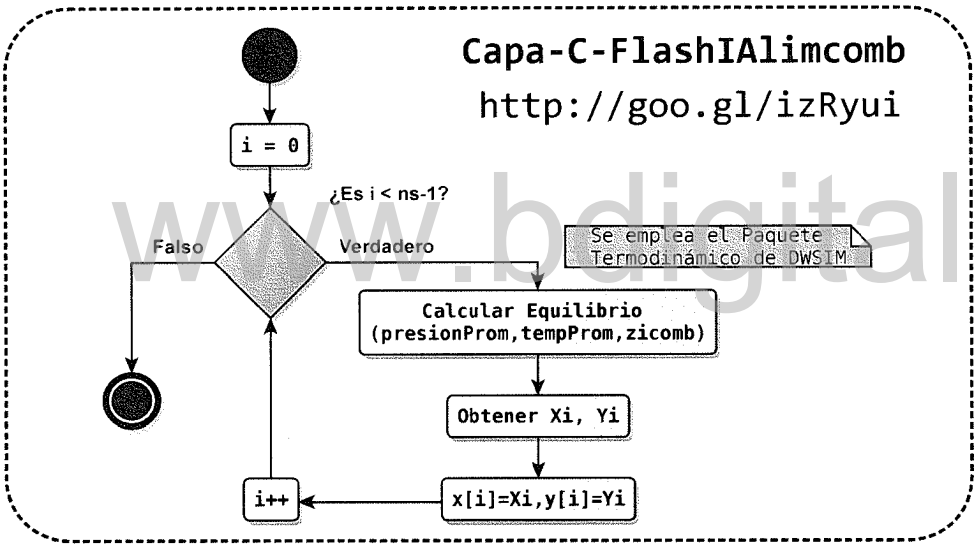
Diagrama
Capa-C-estimacionVarIt

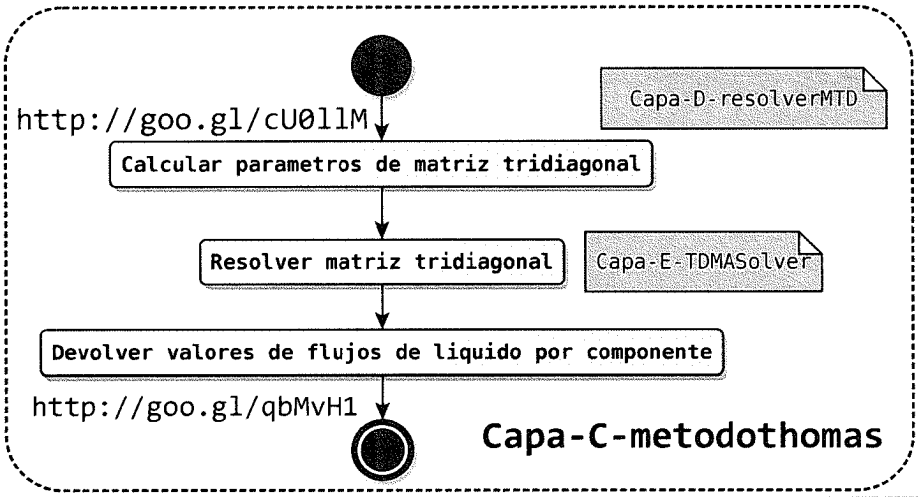


Apéndice B.3

Diagrama
Capa-C-EstimadoFlujos

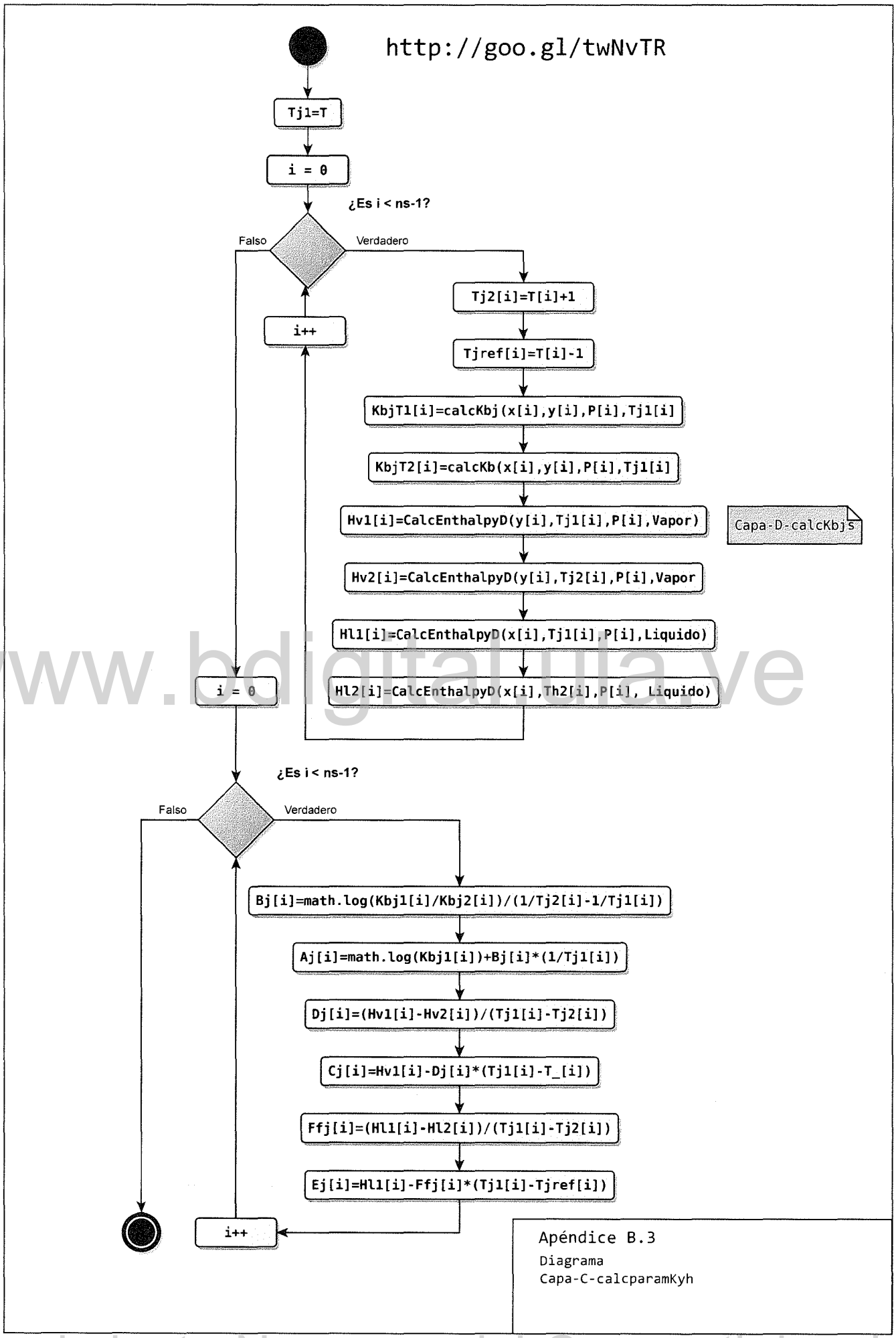


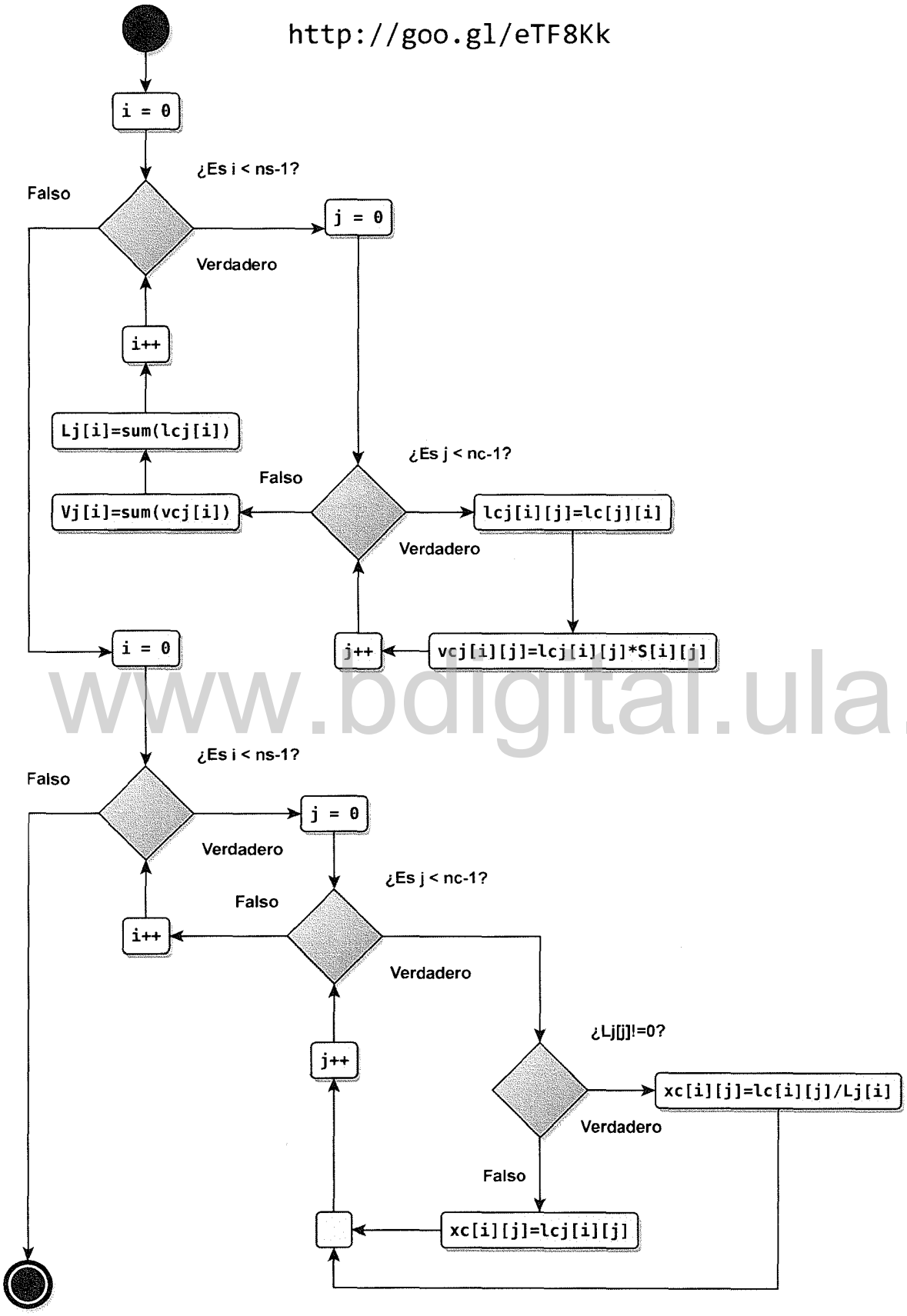




Apéndice B.3

Diagramas:
Capa-C-perfilTempini
Capa-C-FlashIAlimcomb
Capa-C-metodothomas

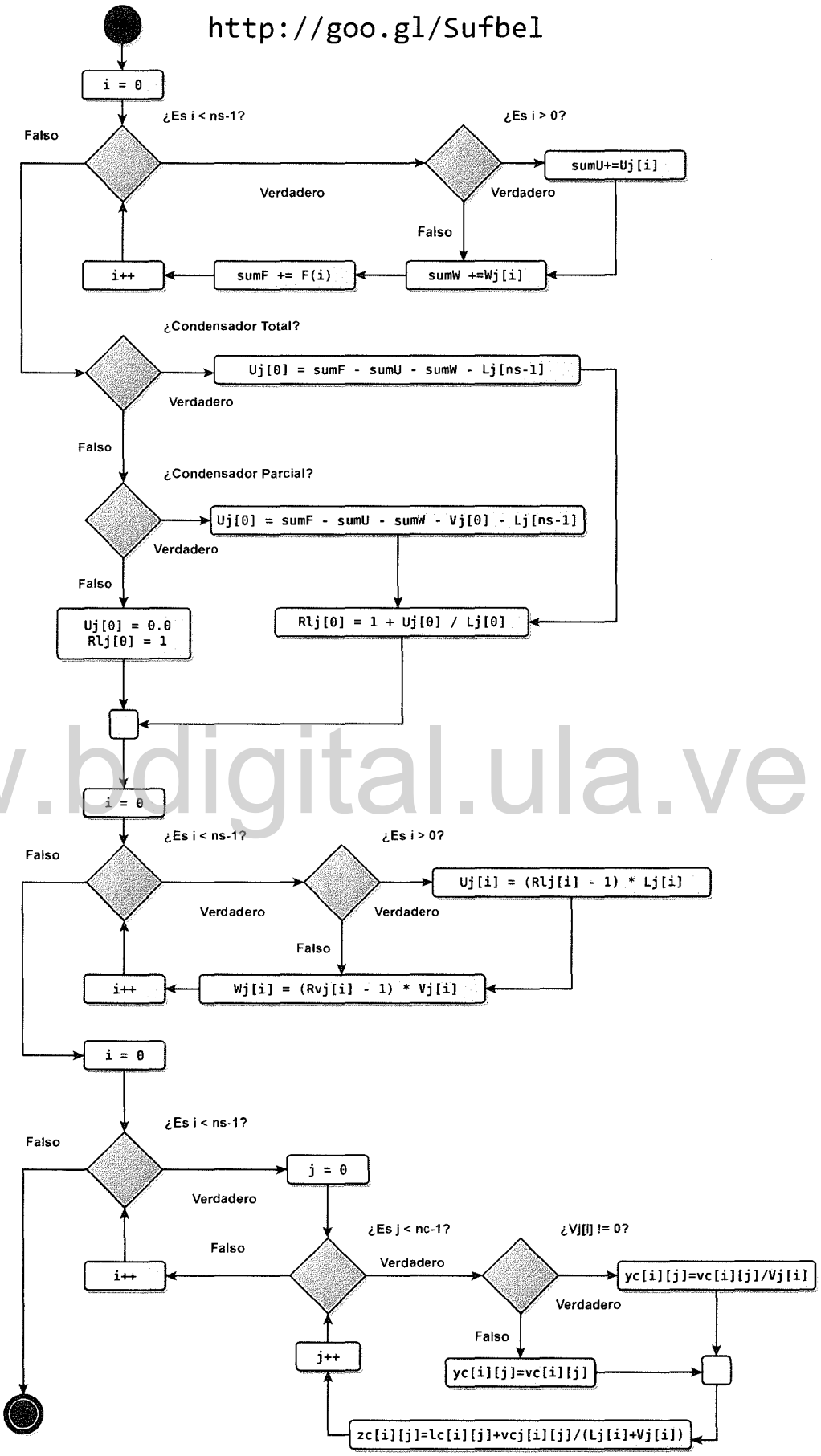




Apéndice B.3

Diagrama
Capa-C-calculofraccionesmol

http://goo.gl/Sufbel



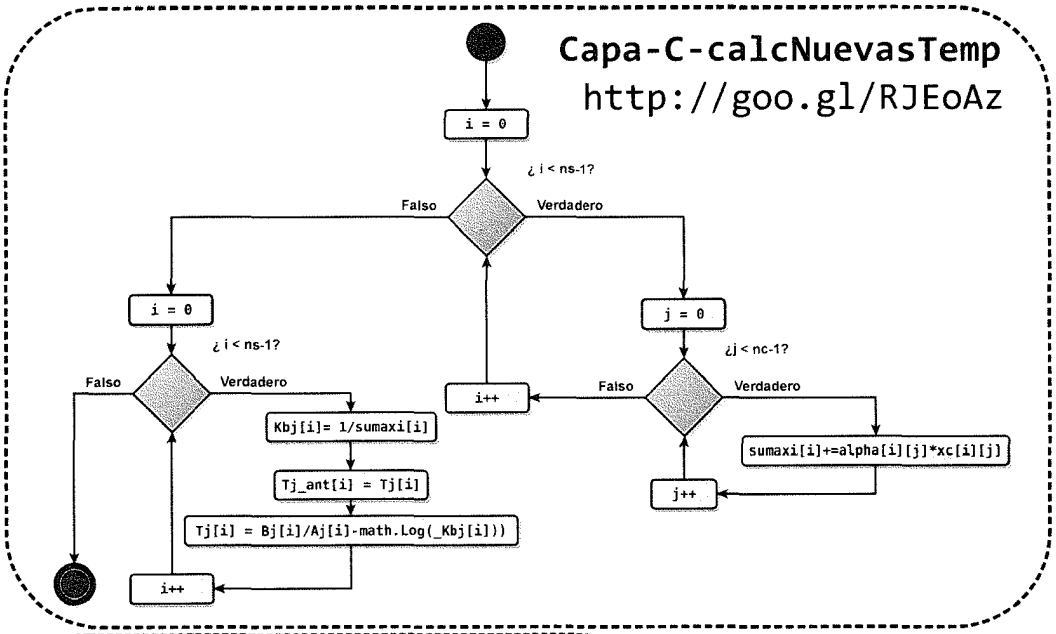
Apéndice B.3

Diagrama

Capa-C-calculoNretiros

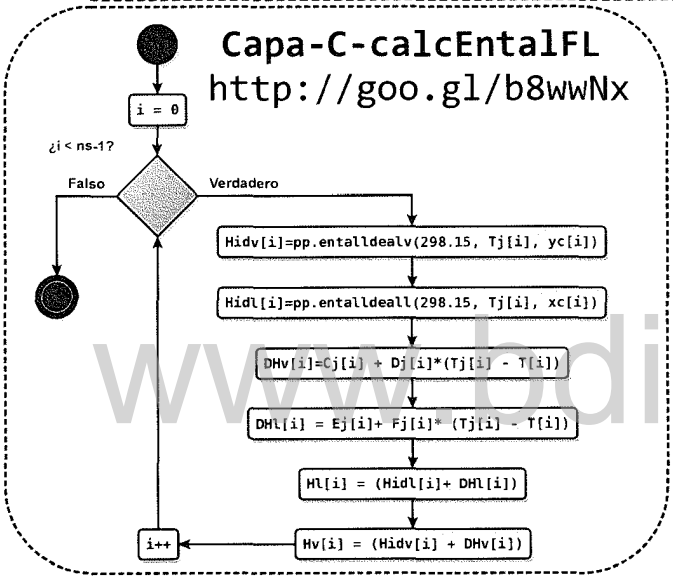
Capa-C-calcNuevasTemp

http://goo.gl/RJEoAz



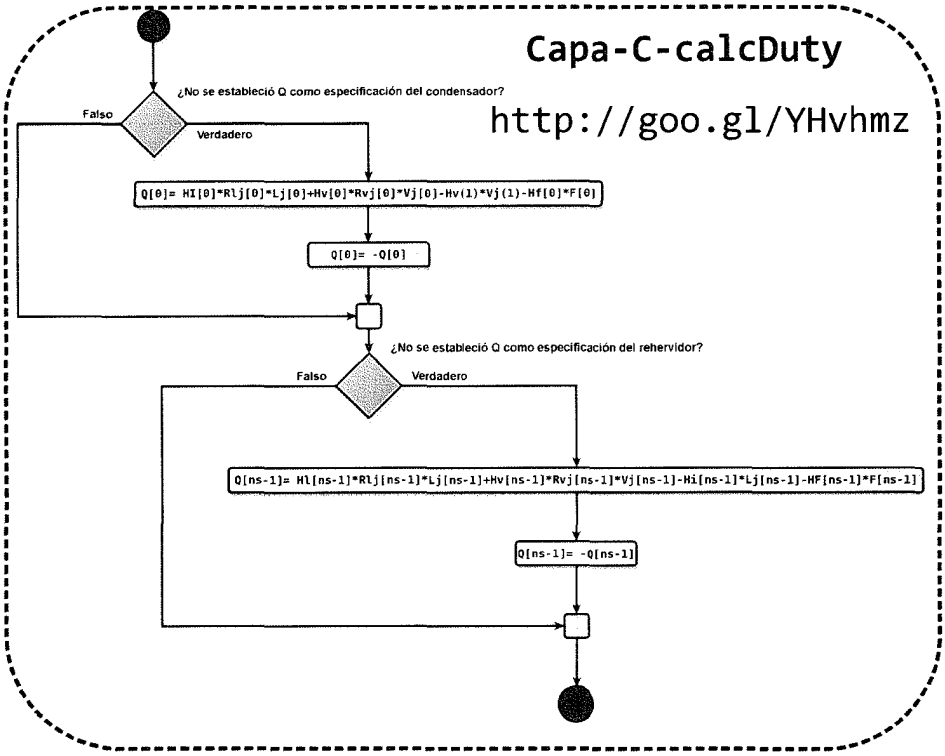
Capa-C-calcEntalFL

http://goo.gl/b8wwNx



Capa-C-calcDuty

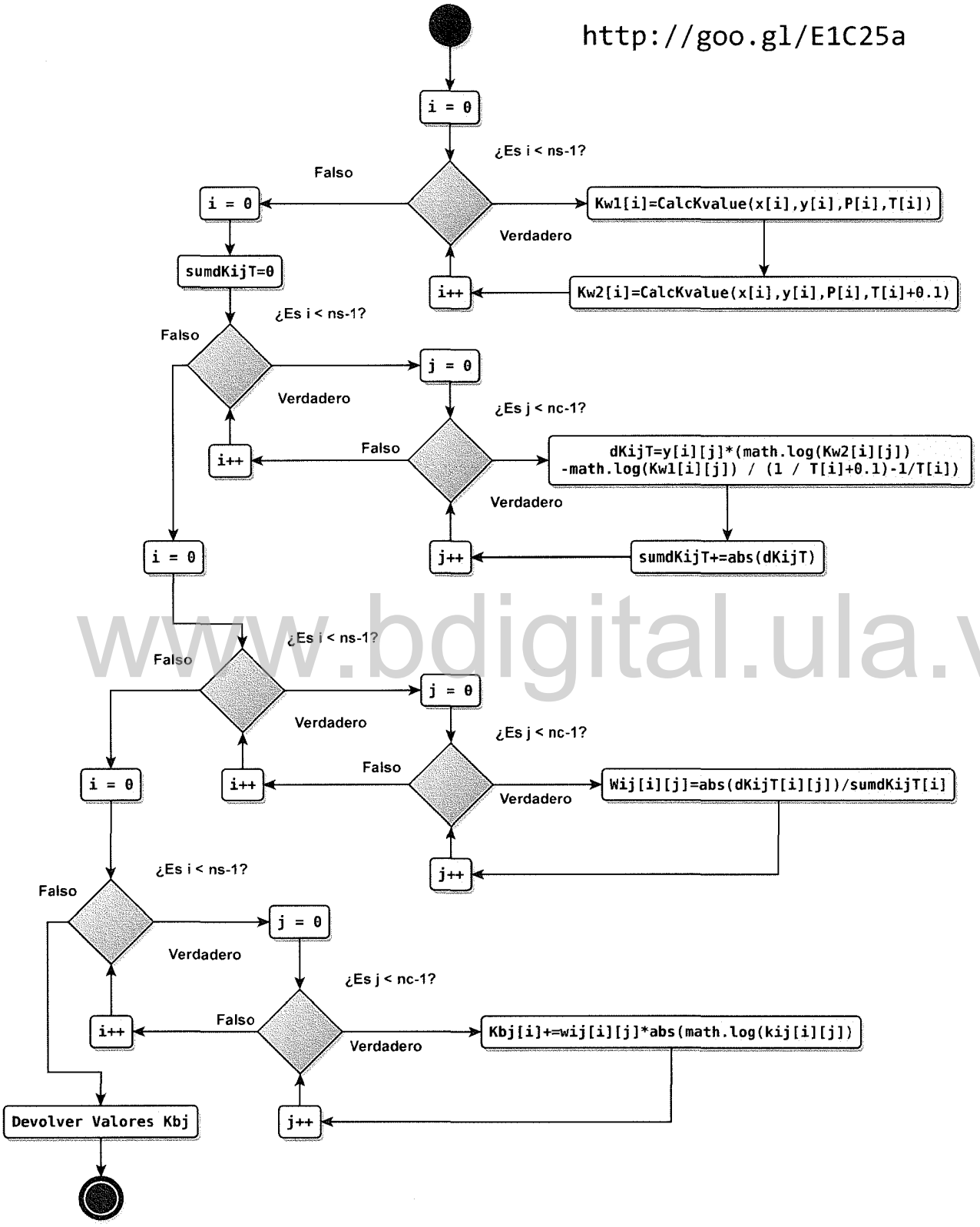
http://goo.gl/YHvhmz



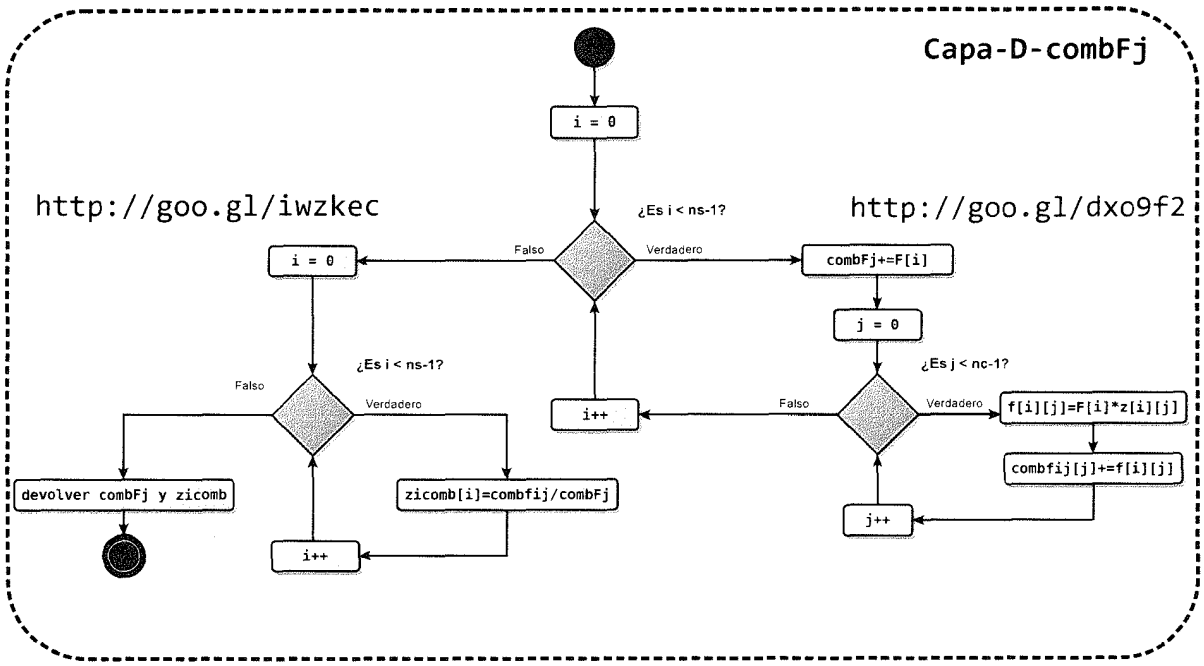
Apéndice B.3
Diagramas:
Capa-C-calcNuevasTemp
Capa-C-calcEntalFL
Capa-C-calcDuty

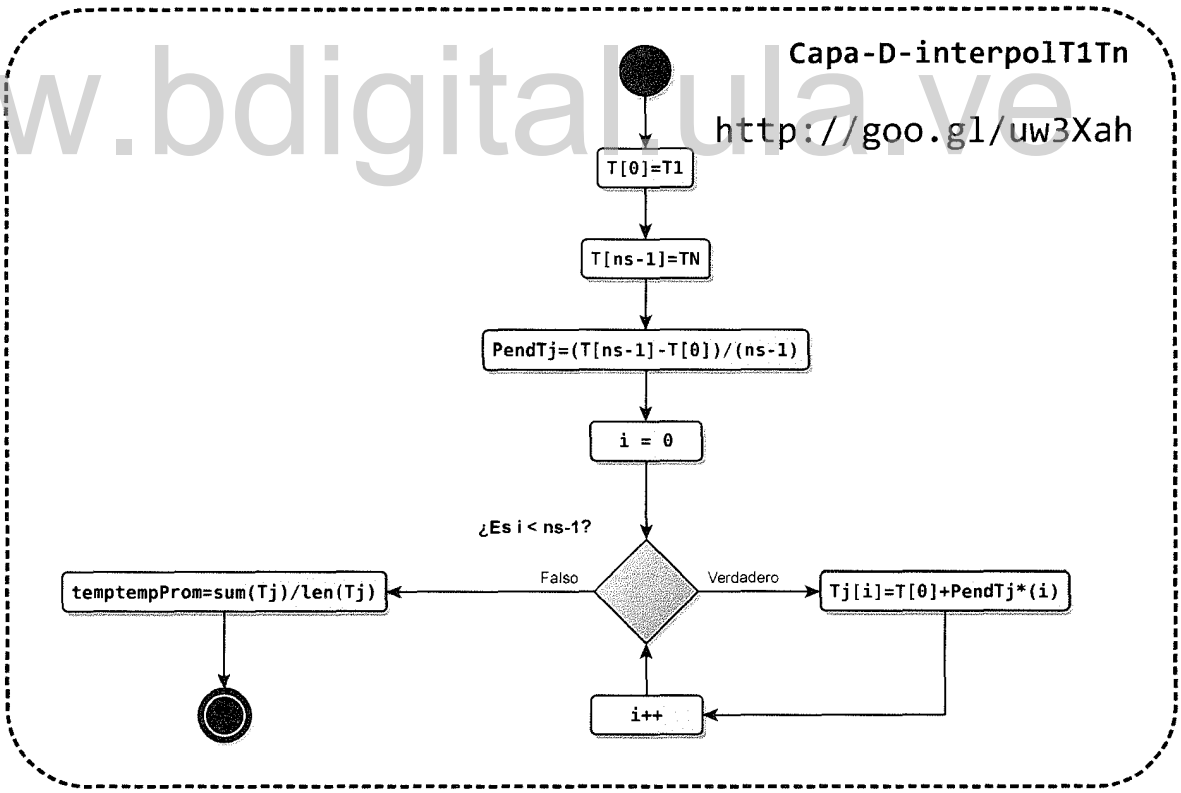
B.4. Diagramas de Actividades Capa-D

http://goo.gl/E1C25a

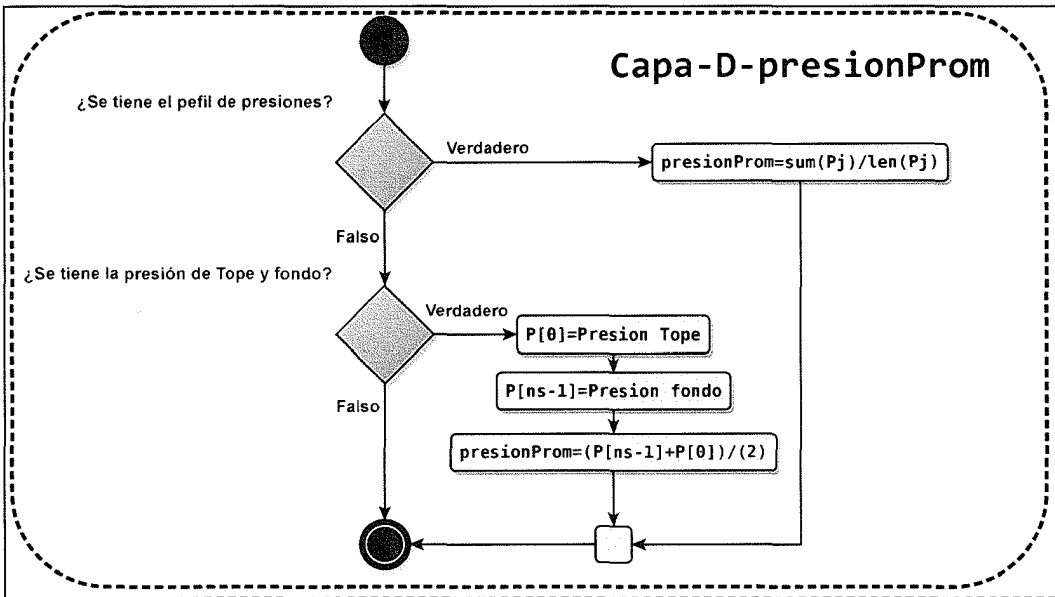


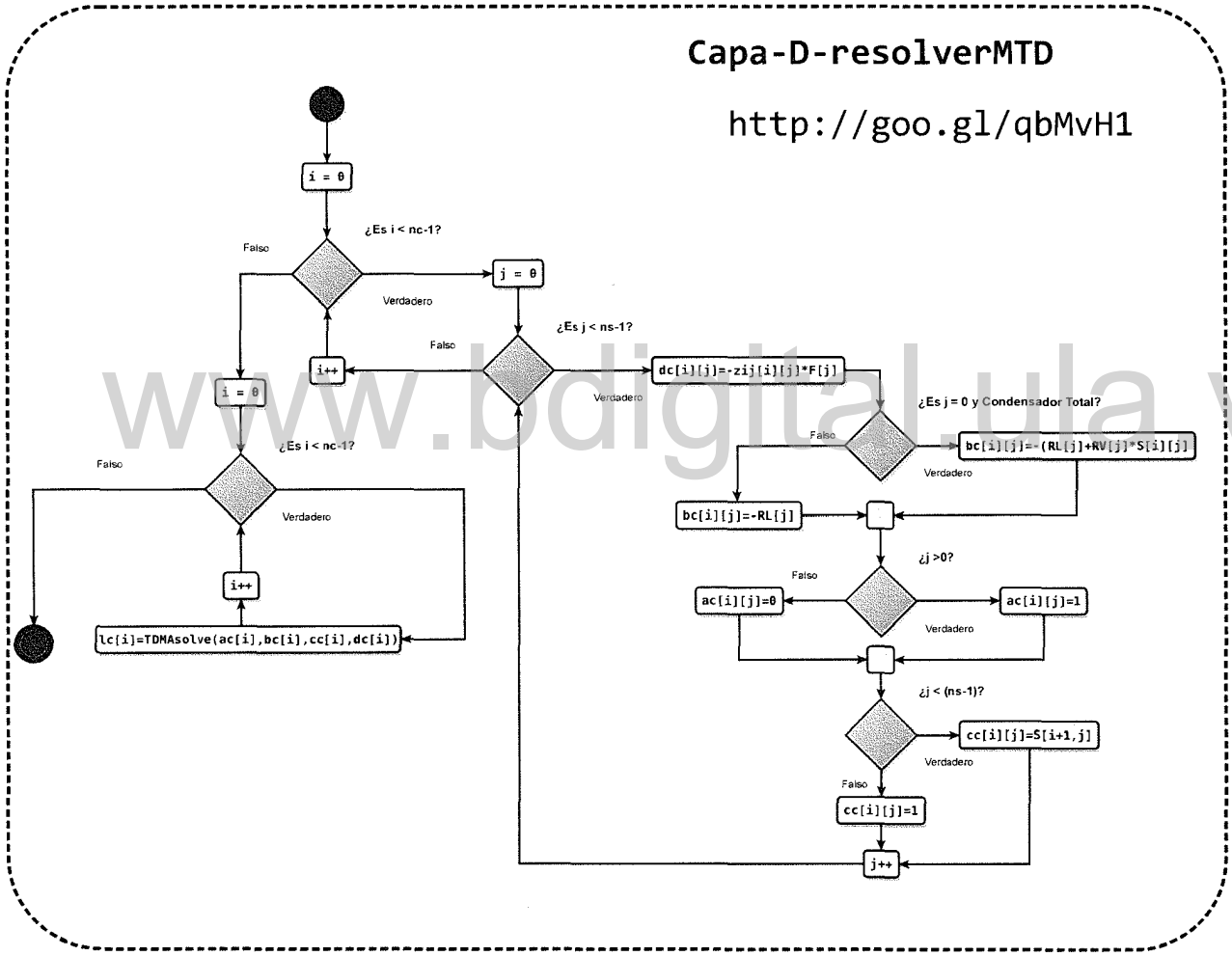
Apéndice B.4
Diagrama
Capa-D-calcKbjs

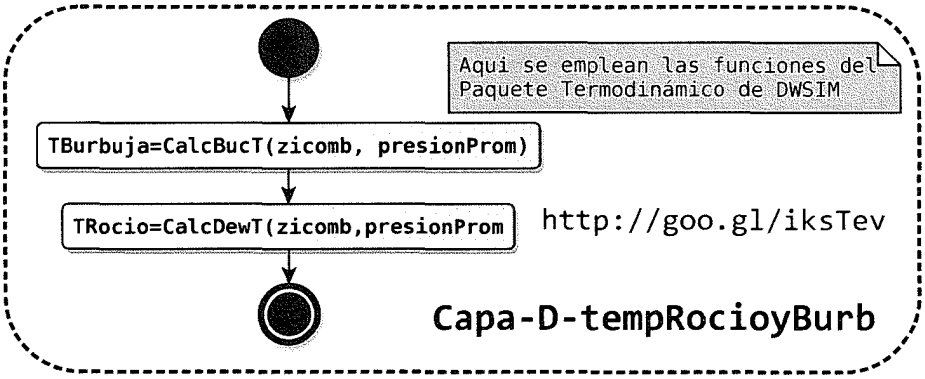




Apéndice B.4
Diagramas:
Capa-D-combFj
Capa-D-interpolT1Tn

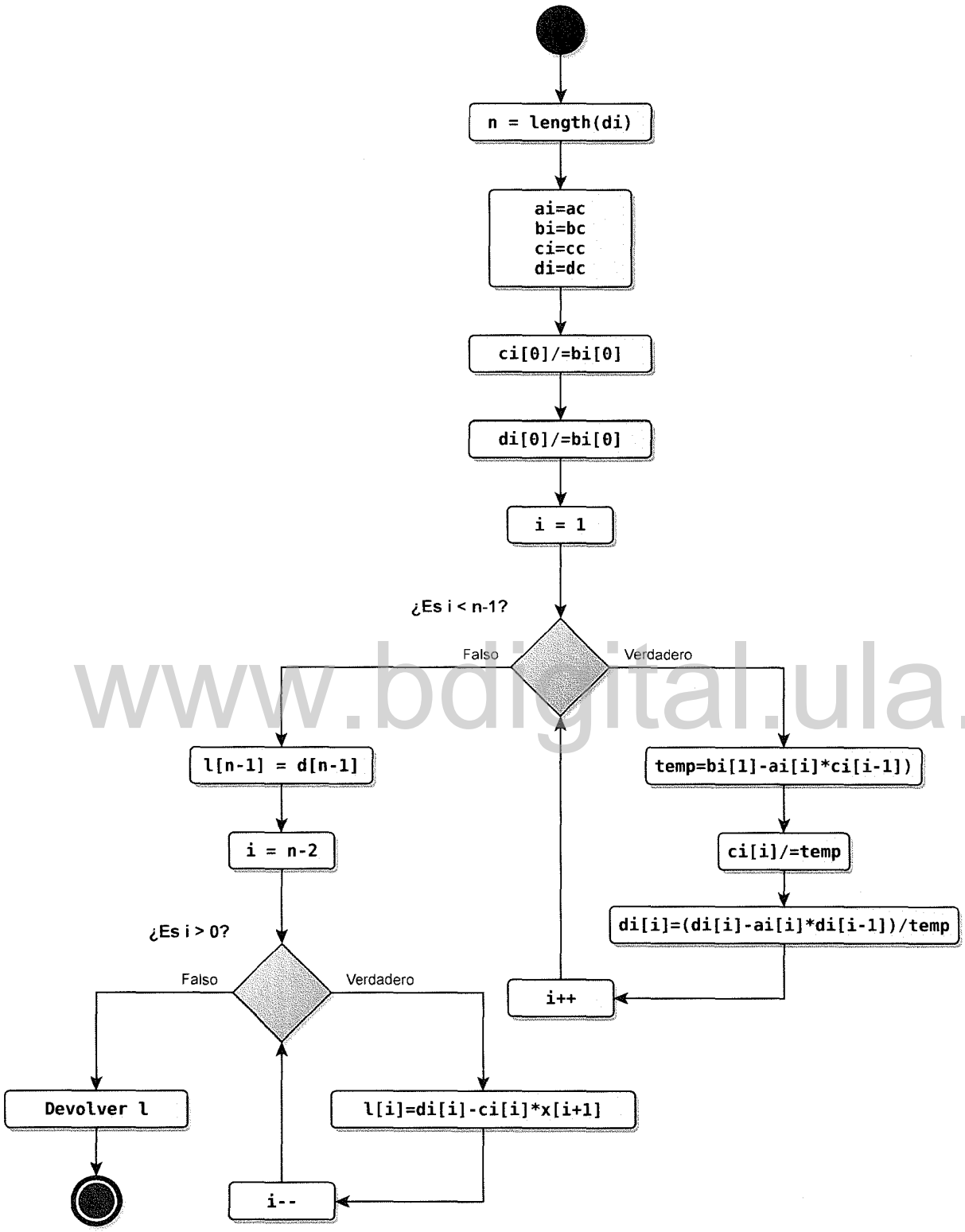






Apéndice B.4
Diagramas:
Capa-D-presionProm
Capa-D-resolverMTD
Capa-D-tempRocioyBurb

B.5. Diagramas de Actividades Capa-E



Apéndice B.5
Diagrama
Capa-E-TDMASolver

Apéndice C

Diagrama de Clases de DWSIM 3

Apéndice D

Tablas de Valores para Gráficas de Paridad de Simulaciones

D.1. Caso de Literatura 6

		DWSIM	PRO/II®	ChemSep™	Referencia
Producto vapor de tope	Etano	0,1812	0,1644	0,1715	0,1098
	Propano	0,8175	0,5923	0,7994	0,8518
	N-Butano	0,001324	0,2310	0,02909	0,0383
	N-Pentano	1E-07	0,0123	3,604E-05	0,0001
	N-Hexane	0,00000	2,667E-05	2,377E-08	0,0000
Producto líquido de tope	Etano	0,05162	0,05628	0,07320	0,25760
	Propano	0,9428	0,4725	0,85004	0,73020
	N-Butano	0,005542	0,4270	0,07653	0,01220
	N-Pentano	0,005542	0,04390	2,276E-04	0,00000
	N-Hexane	0,000	0,00028	3,492E-07	0,00000
Retiro Lateral Líquido	Etano	0,007996	0,03425	0,01879	0,020667
	Propano	0,9328728	0,3502	0,693	0,7763
	N-Butano	0,0589407	0,5088	0,285	0,2010
	N-Pentano	0,0001898	0,1040	0,00416	0,0020
	N-Hexane	0,0000	0,0027	3,0754E-05	0,0000
Retiro Lateral Vapor	Etano	1,9E-06	0,003992	1,3334E-04	8,1081E-05
	Propano	0,0060455	0,19153	0,04312	0,0321
	N-Butano	0,7967989	0,50933	0,6618	0,6836
	N-Pentano	0,190381	0,2777	0,2796	0,2709
	N-Hexane	0,0067727	0,01742	0,01525	0,0133
Producto líquido de fondo	Etano	0,00	3,99998E-05	7,2438E-07	0,00
	Propano	3,3E-05	0,01540	0,002195	0,001375
	N-Butano	0,1824	0,2757	0,2709	0,261700
	N-Pentano	0,6989	0,6002	0,61598	0,624250
	N-Hexane	0,1187	0,1086	0,1109	0,112675

Tabla D.1: Valores de fracciones molares para caso de literatura 6

D.2. Casos Operacionales

	DWSIM	Referencia
Benceno	0,9791	0,9994
Tolueno	0,021	0,00060
Etilbenceno	0,000	0,000
P-xileno	0,000	0,000
M-xileno	0,000	0,000
O-xileno	0,000	0,000
Cumeno	0,000	0,000
1,2,3-Trimetilbenceno	0,000	0,000
Nonano	0,000	0,000
N-propilbenceno	0,000	0,000
O-Etiltolueno	0,000	0,000
Indano	0,000	0,000
N-butilbenceno	0,000	0,000
N-pentilbenceno	0,000	0,000
Bifenil	0,000	0,000

(a) Producto de Tope

	DWSIM	Referencia
Benceno	0,00254	0,001
Tolueno	0,5557	0,511
Etilbenceno	0,05848	0,062
P-xileno	0,05974	0,063
M-xileno	0,1463	0,154
O-xileno	0,07369	0,078
Cumeno	0,0023736	0,003
1,2,3-Trimetilbenceno	0,0317694	0,038
Nonano	8,56E-05	0,000
N-propilbenceno	0,0034691	0,004
O-Etiltolueno	0,0291224	0,035
Indano	0,0006499	0,001
N-butilbenceno	0,0206012	0,027
N-pentilbenceno	0,0073275	0,011
Bifenil	0,0081827	0,013

(b) Producto de Fondo

Tabla D.2: Valores de fracciones molares para caso operacional 1 escenario 1

	DWSIM	Referencia
Benceno	0,99840	1,000
Tolueno	0,001210	0,000
Etilbenceno	0,000	0,000
P-xileno	0,000	0,000
M-xileno	0,000	0,000
O-xileno	0,000	0,000
Cumeno	0,000	0,000
1,2,3-Trimetilbenceno	0,000	0,000
Nonano	0,000	0,000
N-propilbenceno	0,000	0,000
O-Etiltolueno	0,000	0,000
Indano	0,000	0,000
N-butilbenceno	0,000	0,000
N-pentilbenceno	0,000	0,000
Bifenil	0,000	0,000

(a) Producto de Tope

	DWSIM	Referencia
Benceno	0,000	0,03128
Tolueno	0,458749	0,4108
Etilbenceno	0,0673856	0,06941
P-xileno	0,0688143	0,07088
M-xileno	0,168464	0,1735
O-xileno	0,0848868	0,08746
Cumeno	0,0034791	0,004066
1,2,3-Trimetilbenceno	0,0465664	0,0543
Nonano	0,000	0,000
N-propilbenceno	0,0050848	0,005943
O-Etiltolueno	0,0426866	0,0498
Indano	0,0009527	0,00109,5
N-butilbenceno	0,0301965	0,0393,2
N-pentilbenceno	0,0107405	0,0154,5
Bifenil	0,0119939	0,01795

(b) Producto de Fondo

Tabla D.3: Valores de fracciones molares para caso operacional 1 escenario 1

	DWSIM	Referencia
Benceno	0,00308	0,00262
Tolueno	0,00349	0,00352
Etilbenceno	0,00031	0,00031
P-xileno	0,25169	0,24520
M-xileno	0,73135	0,73942
O-xileno	0,00714	0,00637
Cumeno	0,00014	0,00005
1,2,3-Trimetilbenceno	0,00098	0,00055
Nonano	0,00150	0,00096
N-propilbenceno	0,00020	0,00005
O-Etiltolueno	0,00007	0,00002
Indano	0,00004	0,00004
N-butilbenceno	0,00000	0,00000
N-pentilbenceno	0,00000	0,00000
Bifenil	0,00000	0,00090

(a) Producto de Tope

	DWSIM	Referencia
Benceno	0,000	0,000
Tolueno	0,000	0,000
Etilbenceno	0,000	0,000
P-xileno	0,00142	0,000
M-xileno	0,01868	0,00668
O-xileno	0,33168	0,33630
Cumeno	0,09579	0,09704
1,2,3-Trimetilbenceno	0,13549	0,13744
Nonano	0,17709	0,17961
N-propilbenceno	0,13610	0,13788
O-Etiltolueno	0,09550	0,09672
Indano	0,00707	0,00716
N-butilbenceno	0,00054	0,00055
N-pentilbenceno	0,00056	0,00057
Bifenil	0,00006	0,00004

(b) Producto de Fondo

Tabla D.4: Valores de fracciones másicas para caso operacional 2

Apéndice E

Archivos de Código Fuente Modificados de DWSIM

E.1. Archivo *Script parcial* RigorousColumn.vb

```
3126 Public Overrides Function Calculate(Optional ByVal args As Object = Nothing) As Integer
3127
3128     Dim objargs As New DWSIM.Outros.StatusChangeEventArgs
3129
3130     'Validate unitop status.
3131     Me.Validate()
3132
3133     'Check connectors' positions
3134     Me.CheckConnPos()
3135
3136     'prepare variables
3137
3138     Dim llextractor As Boolean = False
3139     Dim myabs As AbsorptionColumn = TryCast(Me, AbsorptionColumn)
3140     If myabs IsNot Nothing Then
3141         If CType(Me, AbsorptionColumn).OperationMode = AbsorptionColumn.OpMode.Absorber
                Then
3142             llextractor = False
3143         Else
3144             llextractor = True
3145         End If
3146     End If
3147
3148     Dim pp As PropertyPackages.PropertyPackage = Me.PropertyPackage
3149
3150     Dim nc, ns, maxits, i, j As Integer
3151     Dim firstF As Integer = -1
3152     Dim lastF As Integer = -1
3153     nc = Me.FlowSheet.Options.SelectedComponents.Count
3154     ns = Me.Stages.Count - 1
3155     maxits = Me.MaxIterations
3156
3157     Dim tol(4) As Double
```

```
3158      tol(0) = Me.InternalLoopTolerance
3159      tol(1) = Me.ExternalLoopTolerance
3160
3161      Dim F(ns), Q(ns), V(ns), L(ns), VSS(ns), LSS(ns), HF(ns), T(ns), FT(ns), P(ns), fracv(
              ns), eff(ns), _
3162          distrate, rr, vaprate As Double
3163
3164      Dim x(ns)() As Double, y(ns)() As Double, z(ns)() As Double, fc(ns)() As Double
3165      Dim idealK(ns)(), Kval(ns)(), Pvap(ns)() As Double
3166
3167      For i = 0 To ns
3168          Array.Resize(x(i), nc)
3169          Array.Resize(y(i), nc)
3170          Array.Resize(fc(i), nc)
3171          Array.Resize(z(i), nc)
3172          Array.Resize(idealK(i), nc)
3173          Array.Resize(Kval(i), nc)
3174          Array.Resize(Pvap(i), nc)
3175      Next
3176
3177      Dim sumcf(nc - 1), sumF, zm(nc - 1) As Double
3178
3179      i = 0
3180
3181      Dim stream As New SimulationObjects.Streams.MaterialStream("", "")
3182
3183      For Each ms As StreamInformation In Me.MaterialStreams.Values
3184          Select Case ms.StreamBehavior
3185              Case StreamInformation.Behavior.Feed
3186                  stream = FlowSheet.Collections.CLCS_MaterialStreamCollection(ms.Name)
3187                  pp.CurrentMaterialStream = stream
3188                  F(StageIndex(ms.AssociatedStage)) = stream.Fases(0).SPMProperties.molarflow
                          .GetValueOrDefault
3189                  HF(StageIndex(ms.AssociatedStage)) = stream.Fases(0).SPMProperties.enthalpy
                          .GetValueOrDefault * _
3190                                                      stream.Fases(0).SPMProperties.
                                                              molecularWeight.GetValueOrDefault
3191                  FT(StageIndex(ms.AssociatedStage)) = stream.Fases(0).SPMProperties.
                          temperature.GetValueOrDefault
3192                  sumF += F(StageIndex(ms.AssociatedStage))
3193                  j = 0
3194                  For Each comp As ClassesBasicasTermodinamica.Substancia In stream.Fases(0).
                          Componentes.Values
3195                      fc(StageIndex(ms.AssociatedStage))(j) = comp.FracaoMolar.
                              GetValueOrDefault
3196                      z(StageIndex(ms.AssociatedStage))(j) = comp.FracaoMolar.
                              GetValueOrDefault
3197                      sumcf(j) += comp.FracaoMolar.GetValueOrDefault * F(StageIndex(ms.
                              AssociatedStage))
3198                      j = j + 1
3199                  Next
3200                  If firstF = -1 Then firstF = StageIndex(ms.AssociatedStage)
3201              Case StreamInformation.Behavior.Sidedraw
3202                  If ms.StreamPhase = StreamInformation.Phase.L Then
```

```
3203                    LSS(StageIndex(ms.AssociatedStage)) = ms.FlowRate.Value
3204                Else
3205                    VSS(StageIndex(ms.AssociatedStage)) = ms.FlowRate.Value
3206                End If
3207            Case StreamInformation.Behavior.InterExchanger
3208                Q(StageIndex(ms.AssociatedStage)) = -FlowSheet.Collections.
                        CLCS_EnergyStreamCollection(ms.Name).Energia.GetValueOrDefault
3209        End Select
3210        i += 1
3211    Next
3212
3213    Dim cv As New SistemasDeUnidades.Conversor
3214
3215    vaprate = cv.ConverterParaSI(Me.VaporFlowRateUnit, Me.VaporFlowRate)
3216
3217    Dim sum1(ns), sum0_ As Double
3218    sum0_ = 0
3219    For i = 0 To ns
3220        sum1(i) = 0
3221        For j = 0 To i
3222            sum1(i) += F(j) - LSS(j) - VSS(j)
3223        Next
3224        sum0_ += LSS(i) + VSS(i)
3225    Next
3226
3227    If Me.Specs("C").SType = ColumnSpec.SpecType.Stream_Ratio Then
3228        rr = Me.Specs("C").SpecValue
3229    Else
3230        rr = 2.5
3231    End If
3232    If Me.Specs("R").SType = ColumnSpec.SpecType.Product_Molar_Flow_Rate Then
3233        If Me.CondenserType = condtype.Full_Reflux Then
3234            vaprate = sumF - cv.ConverterParaSI(Me.Specs("R").SpecUnit, Me.Specs("R").
                        SpecValue) - sum0_
3235            distrate = 0.0
3236        ElseIf Me.CondenserType = condtype.Partial_Condenser Then
3237            If Me.Specs("C").SType = ColumnSpec.SpecType.Product_Molar_Flow_Rate Then
3238                distrate = cv.ConverterParaSI(Me.Specs("C").SpecUnit, Me.Specs("C").
                            SpecValue)
3239            Else
3240                distrate = sumF - cv.ConverterParaSI(Me.Specs("R").SpecUnit, Me.Specs("R").
                            SpecValue) - sum0_ - vaprate
3241            End If
3242
3243        Else
3244            distrate = sumF - cv.ConverterParaSI(Me.Specs("R").SpecUnit, Me.Specs("R").
                        SpecValue) - sum0_
3245            vaprate = 0.0
3246        End If
3247    Else
3248        If Me.CondenserType = condtype.Full_Reflux Then
3249            vaprate = sumF / 2 - sum0_
3250        Else
3251            distrate = sumF / 2 - sum0_ - vaprate
```

```
3252            End If
3253        End If
3254
3255        compids = New ArrayList
3256        compids.Clear()
3257        For Each comp As ClassesBasicasTermodinamica.Substancia In stream.Fases(0).Componentes
                .Values
3258            compids.Add(comp.Nome)
3259        Next
3260
3261        For i = ns To 0 Step -1
3262            If F(i) <> 0 Then
3263                lastF = i
3264                Exit For
3265            End If
3266        Next
3267
3268        For i = 0 To nc - 1
3269            zm(i) = sumcf(i) / sumF
3270        Next
3271
3272        Dim T1, T2 As Double
3273
3274        Select Case Me.ColumnType
3275            Case ColType.DistillationColumn
3276                LSS(0) = distrate
3277            Case ColType.RefluxedAbsorber
3278                LSS(0) = distrate
3279        End Select
3280
3281        P(ns) = Me.ReboilerPressure
3282        P(0) = Me.CondenserPressure
3283
3284        Select Case Me.ColumnType
3285            Case ColType.AbsorptionColumn
3286                T1 = MathEx.Common.Max(FT)
3287                T2 = T1
3288            Case ColType.ReboiledAbsorber
3289                T1 = MathEx.Common.Max(FT)
3290                T2 = T1
3291            Case ColType.RefluxedAbsorber
3292                T1 = MathEx.Common.Max(FT)
3293                T2 = T1
3294            Case ColType.DistillationColumn
3295                Try
3296                    T1 = pp.DW_CalcBubT(zm, P(0))(4) '* 1.01
3297                Catch ex As Exception
3298                    T1 = MathEx.Common.Min(FT)
3299                End Try
3300                Try
3301                    T2 = pp.DW_CalcDewT(zm, P(ns))(4) '* 0.99
3302                Catch ex As Exception
3303                    T2 = MathEx.Common.Max(FT)
3304                End Try
```

```
3305        End Select
3306
3307        For i = 0 To ns
3308            sum1(i) = 0
3309            For j = 0 To i
3310                sum1(i) += F(j) - LSS(j) - VSS(j)
3311            Next
3312        Next
3313
3314        T(0) = T1
3315        T(ns) = T2
3316
3317        'Dim pv(nc - 1) As Double
3318        'For i = 0 To nc - 1
3319        ' pv(i) = pp.AUX_PVAPi(i, (T1 + T2) / 2)
3320        'Next
3321
3322        'Dim vapfrac As Double = Me.CalcIdealVapFrac(zm, pv, (P(0) + P(ns)) / 2)
3323
3324        i = 0
3325        For Each st As Stage In Me.Stages
3326            P(i) = st.P
3327            eff(i) = st.Efficiency
3328            If Me.UseTemperatureEstimates Then
3329                T(i) = Me.InitialEstimates.StageTemps(i).Value
3330            Else
3331                T(i) = (T2 - T1) * (i) / ns + T1
3332            End If
3333            If Me.UseVaporFlowEstimates Then
3334                V(i) = Me.InitialEstimates.VapMolarFlows(i).Value
3335            Else
3336                If i = 0 Then
3337                    Select Case Me.ColumnType
3338                        Case ColType.DistillationColumn
3339                            If Me.CondenserType = condtype.Total_Condenser Then
3340                                V(0) = 0.0005
3341                            Else
3342                                V(0) = vaprate
3343                            End If
3344                        Case ColType.RefluxedAbsorber
3345                            If Me.CondenserType = condtype.Total_Condenser Then
3346                                V(0) = 0.0005
3347                            Else
3348                                V(0) = vaprate
3349                            End If
3350                        Case Else
3351                            V(0) = F(lastF)
3352                    End Select
3353                Else
3354                    Select Case Me.ColumnType
3355                        Case ColType.DistillationColumn
3356                            If Me.CondenserType = condtype.Partial_Condenser Then
3357                                V(i) = (rr + 1) * (distrate + vaprate) - F(0)
3358                            ElseIf Me.CondenserType = condtype.Full_Reflux Then
```

```
3359                        V(i) = (rr + 1) * V(0) - F(0)
3360                    Else
3361                        V(i) = (rr + 1) * distrate - F(0)
3362                    End If
3363                Case ColType.RefluxedAbsorber
3364                    V(i) = (rr + 1) * distrate - F(0) + V(0)
3365                Case ColType.AbsorptionColumn
3366                    V(i) = F(lastF)
3367                Case ColType.ReboiledAbsorber
3368                    V(i) = F(lastF)
3369            End Select
3370        End If
3371    End If
3372    If Me.UseLiquidFlowEstimates Then
3373        L(i) = Me.InitialEstimates.LiqMolarFlows(i).Value
3374    Else
3375        If i = 0 Then
3376            Select Case Me.ColumnType
3377                Case ColType.DistillationColumn
3378                    If Me.CondenserType = condtype.Partial_Condenser Then
3379                        L(0) = (distrate + vaprate) * rr
3380                    ElseIf Me.CondenserType = condtype.Full_Reflux Then
3381                        L(0) = vaprate * rr
3382                    Else
3383                        L(0) = distrate * rr
3384                    End If
3385                Case ColType.RefluxedAbsorber
3386                    If Me.CondenserType = condtype.Partial_Condenser Then
3387                        L(0) = distrate * rr
3388                    ElseIf Me.CondenserType = condtype.Full_Reflux Then
3389                    Else
3390                        L(0) = distrate * rr
3391                    End If
3392                Case Else
3393                    L(0) = F(firstF)
3394                    If L(0) = 0 Then L(i) = 0.00001
3395            End Select
3396        Else
3397            Select Case Me.ColumnType
3398                Case ColType.DistillationColumn
3399                    If i < ns Then L(i) = V(i) + sum1(i) - V(0) Else L(i) = sum1(i) - V
                            (0)
3400                Case ColType.RefluxedAbsorber
3401                    If i < ns Then L(i) = V(i) + sum1(i) - V(0) Else L(i) = sum1(i) - V
                            (0)
3402                Case ColType.AbsorptionColumn
3403                    L(i) = F(firstF)
3404                Case ColType.ReboiledAbsorber
3405                    L(i) = F(firstF)
3406            End Select
3407            If L(i) = 0 Then L(i) = 0.00001
3408        End If
3409    End If
3410    If Me.UseCompositionEstimates Then
```

```
3411            j = 0
3412            For Each par As Parameter In Me.InitialEstimates.LiqCompositions(i).Values
3413                x(i)(j) = par.Value
3414                j = j + 1
3415            Next
3416            j = 0
3417            For Each par As Parameter In Me.InitialEstimates.VapCompositions(i).Values
3418                y(i)(j) = par.Value
3419                j = j + 1
3420            Next
3421            For j = 0 To nc - 1
3422                Kval(i)(j) = y(i)(j) / x(i)(j)
3423                z(i)(j) = zm(j)
3424            Next
3425        Else
3426            x(i) = pp.FlashBase.Flash_PT(zm, P(i), T(i), PropertyPackage)(2)
3427            y(i) = pp.FlashBase.Flash_PT(zm, P(i), T(i), PropertyPackage)(3)
3428            z(i) = zm
3429            For j = 0 To nc - 1
3430                Kval(i)(j) = y(i)(j) / x(i)(j)
3431            Next
3432            'Dim tmp As Object = pp.DW_CalcKvalue(zm, (T1 + T2) / 2, (P(0) + P(ns)) / 2)
3433            'For j = 0 To nc - 1
3434            ' fracv(i) = Me.CalcIdealVapFrac(zm, Pvap(i), P(i))
3435            ' If fracv(i) < 0 Then fracv(i) = 0.000001
3436            ' If fracv(i) > 1 Then fracv(i) = 0.999999
3437            ' y(i)(j) = zm(j) * idealK(i)(j) / ((idealK(i)(j) - 1) * fracv(i) + 1)
3438            ' x(i)(j) = y(i)(j) / idealK(i)(j)
3439            ' z(i)(j) = zm(j)
3440            'Next
3441            'Dim sumx, sumy As Double
3442            'sumx = 0
3443            'sumy = 0
3444            'For j = 0 To nc - 1
3445            ' sumx += x(i)(j)
3446            ' sumy += y(i)(j)
3447            'Next
3448            'For j = 0 To nc - 1
3449            ' x(i)(j) = x(i)(j) / sumx
3450            ' y(i)(j) = y(i)(j) / sumy
3451            'Next
3452            'For j = 0 To nc - 1
3453            ' Kval(i)(j) = idealK(i)(j)
3454            'Next
3455        End If
3456        i = i + 1
3457    Next
3458    Select Case Me.ColumnType
3459        Case ColType.DistillationColumn
3460            Q(0) = 0
3461            Q(ns) = 0
3462        Case ColType.ReboiledAbsorber
3463            Q(ns) = 0
3464        Case ColType.RefluxedAbsorber
```

```
3465            Q(0) = 0
3466        End Select
3467
3468        'store initial values
3469
3470        x0.Clear()
3471        y0.Clear()
3472        K0.Clear()
3473        For i = 0 To ns
3474            x0.Add(x(i))
3475            y0.Add(y(i))
3476            K0.Add(Kval(i))
3477        Next
3478        T0 = T
3479        P0 = P
3480        V0 = V
3481        L0 = L
3482        VSS0 = VSS
3483        LSS0 = LSS
3484
3485        'process specifications
3486        For Each sp As Auxiliary.SepOps.ColumnSpec In Me.Specs.Values
3487            If sp.SType = ColumnSpec.SpecType.Component_Fraction Or _
3488            sp.SType = ColumnSpec.SpecType.Component_Mass_Flow_Rate Or _
3489            sp.SType = ColumnSpec.SpecType.Component_Molar_Flow_Rate Or _
3490            sp.SType = ColumnSpec.SpecType.Component_Recovery Then
3491                i = 0
3492                For Each comp As DWSIM.ClassesBasicasTermodinamica.Substancia In stream.Fases
                        (0).Componentes.Values
3493                    If sp.ComponentID = comp.Nome Then sp.ComponentIndex = i
3494                    i = i + 1
3495                Next
3496            End If
3497            If sp.StageNumber = -1 And sp.SpecValue = Me.DistillateFlowRate Then
3498                sumF = 0
3499                Dim sumLSS As Double = 0
3500                Dim sumVSS As Double = 0
3501                For i = 0 To ns
3502                    sumF += F(i)
3503                    sumLSS += LSS(i)
3504                    sumVSS += VSS(i)
3505                Next
3506                sp.SpecValue = sumF - sumLSS - sumVSS - V(0)
3507                sp.StageNumber = 0
3508            End If
3509        Next
3510
3511        Dim result As Object
3512
3513        Select Case Me.SolvingMethod
3514            Case 0 'IO
3515                Dim rm As New SolvingMethods.RussellMethod
3516                result = rm.Solve(nc, ns, maxits, tol, F, V, Q, L, VSS, LSS, Kval, x, y, z, fc,
                        HF, T, P, Me.CondenserType, eff, Me.UseDampingFactor, Me.UseNewtonUpdate,
```

```
                Me.AdjustSb, Me.UseIdentityAsJacobianInverse, Me.ColumnType, Me.
                KbjWeightedAverage, pp, Me.Specs, Me.StoreAndReuseJacobian, Me.
                JacobianMatrix, Me.IO_NumericalDerivativeStep, Me.IO_MaxVarChgFac, Me.
                IO_DampingFactorMin, Me.IO_DampingFactorMax, Me.IO_ExtLoop_DeltaT,
                llextractor)
3517            ic = result(9)
3518            ec = result(11)
3519        Case 1 'BP
3520            result = SolvingMethods.WangHenkeMethod.Solve(nc, ns, maxits, tol, F, V, Q, L,
                    VSS, LSS, Kval, x, y, z, fc, HF, T, P, Me.CondenserType, Me.
                    StopAtIterationNumber, eff, Me.ColumnType, pp, Me.Specs)
3521            ic = result(9)
3522        Case 2 'SR
3523            result = SolvingMethods.BurninghamOttoMethod.Solve(nc, ns, maxits, tol, F, V, Q
                    , L, VSS, LSS, Kval, x, y, z, fc, HF, T, P, Me.StopAtIterationNumber, eff,
                    pp, Me.Specs, llextractor)
3524            ic = result(9)
3525        Case 3 'SC
3526            Dim scm As New SolvingMethods.NaphtaliSandholmMethod
3527            result = scm.Solve(nc, ns, maxits, tol, F, V, Q, L, VSS, LSS, Kval, x, y, z, fc
                    , HF, T, P, Me.CondenserType, eff, Me.UseDampingFactor, Me.UseNewtonUpdate,
                     Me.UseIdentityAsJacobianInverse, Me.ColumnType, pp, Me.Specs, Me.
                    StoreAndReuseJacobian, Me.JacobianMatrix, Me.SC_DampingFactor, Me.
                    SC_MaximumTemperatureChange, Me.SC_NumericalDerivativeStep, Me.
                    SC_MaxVarChgFac, llextractor)
3528            ec = result(11)
3529        Case Else
3530            result = Nothing
3531    End Select
3532
3533    '{Tj, Vj, Lj, VSSj, LSSj, yc, xc, K, Q, ic, t_error}
3534
3535    Me.CondenserDuty = result(8)(0)
3536    Me.ReboilerDuty = result(8)(ns)
3537
3538    'store final values
3539    xf.Clear()
3540    yf.Clear()
3541    Kf.Clear()
3542    For i = 0 To ns
3543        yf.Add(result(5)(i))
3544        xf.Add(result(6)(i))
3545        If Me.SolvingMethod = SolvingMethods.DistColSolvingMethod.Russell_InsideOut Or _
3546           Me.SolvingMethod = SolvingMethods.DistColSolvingMethod.
                   NaphtaliSandholm_SimultaneousCorrection Then
3547            Dim obj(nc - 1) As Double
3548            For j = 0 To nc - 1
3549                obj(j) = result(7)(i, j)
3550            Next
3551            Kf.Add(obj)
3552            Me.JacobianMatrix = result(13)
3553        Else
3554            Kf.Add(result(7)(i))
3555        End If
```

```
3556        Next
3557        Tf = result(0)
3558        Vf = result(1)
3559        Lf = result(2)
3560        VSSf = result(3)
3561        LSSf = result(4)
3562        Q = result(8)
```

E.2. Archivo *Script parcial* RigorousColumnSolver.vb

```
52  <System.Serializable()> Public Class Tomich
53
54      Public Shared Function TDMASolve(ByVal a As Double(), ByVal b As Double(), ByVal c As
            Double(), ByVal d As Double()) As Double()
55
56          ' Warning: will modify c and d!
57
58          Dim n As Integer = d.Length
59          ' All arrays should be the same length
60          Dim x As Double() = New Double(n - 1) {}
61          Dim id As Double
62
63          ' Modify the coefficients.
64
65          c(0) /= b(0)
66          ' Division by zero risk.
67          d(0) /= b(0)
68          ' Division by zero would imply a singular matrix.
69          For i As Integer = 1 To n - 1
70              id = b(i) - c(i - 1) * a(i)
71              c(i) /= id
72              ' This calculation during the last iteration is redundant.
73              d(i) = (d(i) - d(i - 1) * a(i)) / id
74          Next
75
76          ' Now back substitute.
77
78          x(n - 1) = d(n - 1)
79          For i As Integer = n - 2 To 0 Step -1
80              x(i) = d(i) - c(i) * x(i + 1)
81          Next
82
83          Return x
84      End Function
85
86  End Class
87
88  <System.Serializable()> Public Class RussellMethod
89
90      Sub New()
91
92      End Sub
93
```

```
94
95      Private Function CalcKbj1(ByVal ns As Integer, ByVal nc As Integer, ByVal K(,) As
            Object, _
96                                ByVal z()() As Double, ByVal y()() As Double, ByVal T()
                                      As Double, _
97                                ByVal P() As Double, ByRef pp As PropertyPackages.
                                      PropertyPackage) As Object
98
99          Dim i, j As Integer
100
101         Dim Kbj1(ns) As Object
102
103         For i = 0 To ns
104             Kbj1(i) = K(i, 0)
105             For j = 1 To nc - 1
106                 If Abs(K(i, j) - 1) < Abs(Kbj1(i) - 1) And z(i)(j) <> 0 Then Kbj1(i) = K(i,
                        j)
107             Next
108         Next
109
110         Return Kbj1
111
112     End Function
113
114     Private Function CalcKbj2(ByVal ns As Integer, ByVal nc As Integer, ByVal K(,) As
            Object, _
115                               ByVal z()() As Double, ByVal y()() As Double, ByVal T() As
                                      Double, _
116                               ByVal P() As Double, ByRef pp As PropertyPackages.
                                      PropertyPackage) As Object
117
118         Dim i, j As Integer
119
120         Dim Kbj1(ns) As Object
121         Dim Kw11(ns)(), Kw21(ns)() As Double
122         Dim wi(ns, nc - 1), ti(ns, nc - 1), sumwi(ns), sumti(ns) As Double
123         For i = 0 To ns
124             Array.Resize(Kw11(i), nc)
125             Array.Resize(Kw21(i), nc)
126         Next
127
128         For i = 0 To ns
129             Kw11(i) = pp.DW_CalcKvalue(z(i), T(i), P(i))
130             Kw21(i) = pp.DW_CalcKvalue(z(i), T(i) + 0.1, P(i))
131             CheckCalculatorStatus()
132         Next
133
134         For i = 0 To ns
135             sumti(i) = 0
136             For j = 0 To nc - 1
137                 ti(i, j) = y(i)(j) * (Log(Kw21(i)(j)) - Log(Kw11(i)(j))) / (1 / (T(i) +
                        0.1) - 1 / T(i))
138                 sumti(i) += Abs(ti(i, j))
139             Next
```

```
140         Next
141
142         For i = 0 To ns
143             If sumti(i) <> 0 Then
144                 For j = 0 To nc - 1
145                     wi(i, j) = Abs(ti(i, j)) / sumti(i)
146                 Next
147             Else
148                 For j = 0 To nc - 1
149                     wi(i, j) = z(i)(j)
150                 Next
151             End If
152         Next
153
154         For i = 0 To ns
155             Kbj1(i) = 0
156             For j = 0 To nc - 1
157                 Kbj1(i) += wi(i, j) * Log(K(i, j))
158             Next
159             Kbj1(i) = Exp(Kbj1(i))
160             If Kbj1(i) < 0 Then
161                 Kbj1(i) = K(i, 0)
162                 For j = 1 To nc - 1
163                     If Abs(K(i, j) - 1) < Abs(Kbj1(i) - 1) Then Kbj1(i) = K(i, j)
164                 Next
165             End If
166         Next
167
168         Return Kbj1
169
170     End Function
171
172     Dim ndeps As Double = 0.01
173
174     Dim _ns, _nc, _el As Integer
175     Dim _Bj, _Aj, _Cj, _Dj, _Ej, _Fj, _eff, _T_, _Tj, _Lj, _Vj, _LSSj, _VSSj, _LSS, _VSS,
            _Rlj, _Rvj, _F, _P, _HF, _Q, Vjj As Double()
176     Dim _S, _alpha As Double(,)
177     Dim _fc, _xc, _yc, _lc, _vc, _zc As Double()()
178     Dim _Kbj As Object()
179     Dim _rr, _Sb, _maxF As Double
180     Dim _pp As PropertyPackages.PropertyPackage
181     Dim _coltype As Column.ColType
182     Dim _condtype As Column.condtype
183     Dim _bx, _dbx As Double()
184     Dim _vcnt, _lcnt As Integer
185     Dim _specs As Dictionary(Of String, SepOps.ColumnSpec)
186     Dim llextr As Boolean = False
187
188     Public Function FunctionValue(ByVal x() As Double) As Double()
189
190
191         Dim errors(x.Length - 1) As Double
192
```

```
193         Dim cv As New SistemasDeUnidades.Conversor
194         Dim spval1, spval2, spfval1, spfval2 As Double
195         Dim spci1, spci2 As Integer
196
197         spval1 = cv.ConverterParaSI(_specs("C").SpecUnit, _specs("C").SpecValue)
198         spci1 = _specs("C").ComponentIndex
199         spval2 = cv.ConverterParaSI(_specs("R").SpecUnit, _specs("R").SpecValue)
200         spci2 = _specs("R").ComponentIndex
201
202         Dim sum1(_ns) As Double
203         Dim i, j As Integer
204
205         For i = 0 To _ns
206             If i = 0 And _condtype <> Column.condtype.Full_Reflux Then
207                 _Rlj(i) = Exp(x(i))
208             Else
209                 For j = 0 To _nc - 1
210                     _S(i, j) = Exp(x(i)) * _alpha(i, j) * _Sb
211                 Next
212             End If
213         Next
214
215         Dim m1, m2 As Integer
216         m1 = 0
217         m2 = 0
218
219         If _vcnt > 0 Then
220             For i = _ns + 1 To _vcnt + _ns
221                 For j = m1 To _ns
222                     If _Rvj(j) <> 1 Then
223                         m1 = j + 1
224                         Exit For
225                     End If
226                 Next
227                 _Rvj(m1 - 1) = Exp(x(i))
228             Next
229         End If
230
231         If _lcnt > 0 Then
232             For i = _vcnt + _ns + 1 To _vcnt + _lcnt + _ns
233                 For j = m2 + 1 To _ns
234                     If _Rlj(j) <> 1 Then
235                         m2 = j + 1
236                         Exit For
237                     End If
238                 Next
239                 _Rlj(m2 - 1) = Exp(x(i))
240             Next
241         End If
242         If _condtype = Column.condtype.Partial_Condenser Then
243             For j = 0 To _nc - 1
244                 _S(0, j) = Exp(x(_el)) * _alpha(0, j) * _Sb
245             Next
246         End If
```

```
247         'step4
248
249         'find component liquid flows by the tridiagonal matrix method
250
251         Dim Bs(_ns, _nc - 1), Cs(_ns, _nc - 1) As Double
252         Dim at(_nc - 1)(), bt(_nc - 1)(), ct(_nc - 1)(), dt(_nc - 1)(), xt(_nc - 1)() As
                Double
253
254         For i = 0 To _nc - 1
255             Array.Resize(at(i), _ns + 1)
256             Array.Resize(bt(i), _ns + 1)
257             Array.Resize(ct(i), _ns + 1)
258             Array.Resize(dt(i), _ns + 1)
259             Array.Resize(xt(i), _ns + 1)
260         Next
261
262         Dim ic0 As Integer = 0
263
264         For i = 0 To _ns
265             For j = 0 To _nc - 1
266                 If i = 0 And _condtype = Column.condtype.Total_Condenser Then
267                     Bs(i, j) = -(_Rlj(i))
268                 Else
269                     Bs(i, j) = -(_Rlj(i) + _S(i, j) * _Rvj(i))
270                 End If
271                 If i < _ns Then Cs(i, j) = _S(i + 1, j)
272             Next
273         Next
274
275         For i = 0 To _nc - 1
276             For j = 0 To _ns
277                 dt(i)(j) = -_fc(j)(i) * _F(j)
278                 bt(i)(j) = Bs(j, i)
279                 If j < _ns Then ct(i)(j) = Cs(j, i)
280                 If j > 0 Then at(i)(j) = 1
281             Next
282         Next
283
284         'solve matrices
285
286         'tomich
287         For i = 0 To _nc - 1
288             xt(i) = Tomich.TDMASolve(at(i), bt(i), ct(i), dt(i))
289         Next
290
291         For i = 0 To _ns
292             Array.Resize(_xc(i), _nc)
293             Array.Resize(_yc(i), _nc)
294             Array.Resize(_lc(i), _nc)
295             Array.Resize(_vc(i), _nc)
296             Array.Resize(_zc(i), _nc)
297         Next
298
299         'step5
```

```
300
301         For i = 0 To _ns
302             _Lj(i) = 0
303             For j = 0 To _nc - 1
304                 _lc(i)(j) = xt(j)(i)
305                 _Lj(i) += _lc(i)(j)
306             Next
307             'If _Lj(i) < 0 Then
308             ' _Lj(i) = 1.0E-20
309             'End If
310         Next
311
312         For i = 0 To _ns
313             For j = 0 To _nc - 1
314                 _xc(i)(j) = _lc(i)(j) / _Lj(i)
315             Next
316         Next
317
318         For i = _ns To 0 Step -1
319             _Vj(i) = 0
320             For j = 0 To _nc - 1
321                 If i < _ns Then
322                     If _eff(i) <> 1.0 Then
323                         _vc(i)(j) = _eff(i) * (_S(i, j) * _lc(i)(j) - _vc(i + 1)(j) * _Vj(i)
                                / _Vj(i + 1)) + _vc(i + 1)(j) * _Vj(i) / _Vj(i + 1)
324                     Else
325                         _vc(i)(j) = _S(i, j) * _lc(i)(j)
326                     End If
327                 Else
328                     _vc(i)(j) = _S(i, j) * _lc(i)(j)
329                 End If
330                 _Vj(i) += _vc(i)(j)
331             Next
332             'If _Vj(i) < 0 Then
333             ' _Vj(i) = 1.0E-20
334             'End If
335         Next
336
337         'departures from product flows
338
339         Dim sumLSS As Double = 0
340         Dim sumVSS As Double = 0
341         Dim sumF As Double = 0
342         For i = 0 To _ns
343             If i > 0 Then sumLSS += _LSSj(i)
344             sumVSS += _VSSj(i)
345             sumF += _F(i)
346         Next
347         If _condtype = Column.condtype.Total_Condenser Then
348             _LSSj(0) = sumF - sumLSS - sumVSS - _Lj(_ns)
349             _Rlj(0) = 1 + _LSSj(0) / _Lj(0)
350         ElseIf _condtype = Column.condtype.Partial_Condenser Then
351             _LSSj(0) = sumF - sumLSS - sumVSS - _Vj(0) - _Lj(_ns)
352             _Rlj(0) = 1 + _LSSj(0) / _Lj(0)
```

```
353         Else
354             _LSSj(0) = 0.0
355             _Rlj(0) = 1
356         End If
357         'If _Lj(0) <> 0 Or Not Double.IsNaN(_Lj(0)) Or Not Double.IsInfinity(_Lj(0)) Then
358         ' _Rlj(0) = 1 + _LSSj(0) / _Lj(0)
359         'Else
360         ' _Rlj(0) = 1
361         'End If
362
363         For i = 0 To _ns
364             _VSSj(i) = (_Rvj(i) - 1) * _Vj(i)
365             If i > 0 Then _LSSj(i) = (_Rlj(i) - 1) * _Lj(i)
366         Next
367
368         'For i = 0 To _ns
369         ' sum1(i) = 0
370         ' For j = 0 To i
371         ' sum1(i) += _F(j) - _LSSj(j) - _VSSj(j)
372         ' Next
373         'Next
374
375         ''Ljs
376         'For i = 0 To _ns
377         ' If i < _ns Then _Lj(i) = _Vj(i + 1) + sum1(i) - _Vj(0) Else _Lj(i) = sum1(i) -
                _Vj(0)
378         ' If _Lj(i) < 0 Then
379         ' _Lj(i) = 0.0000000001
380         ' End If
381         'Next
382
383         For i = 0 To _ns
384             For j = 0 To _nc - 1
385                 If _Vj(i) <> 0 Then
386                     _yc(i)(j) = _vc(i)(j) / _Vj(i)
387                 Else
388                     _yc(i)(j) = _vc(i)(j)
389                 End If
390                 _zc(i)(j) = (_lc(i)(j) + _vc(i)(j)) / (_Lj(i) + _Vj(i))
391             Next
392         Next
393
394         Dim sum_axi(_ns) As Double
395
396         For i = 0 To _ns
397             sum_axi(i) = 0
398             For j = 0 To _nc - 1
399                 sum_axi(i) += _alpha(i, j) * _xc(i)(j)
400             Next
401         Next
402
403         'step6
404
405         'calculate new temperatures
```

```
406
407         Dim _Tj_ant(_ns) As Double
408
409         For i = 0 To _ns
410             _Kbj(i) = 1 / sum_axi(i)
411             _Tj_ant(i) = _Tj(i)
412             _Tj(i) = _Bj(i) / (_Aj(i) - Log(_Kbj(i)))
413             If Abs(_Tj(i) - _Tj_ant(i)) > 100 Or Double.IsNaN(_Tj(i)) Or Double.IsInfinity(
                    _Tj(i)) Then
414                 'switch to a bubble point temperature calculation...
415                 Dim tmp = _pp.DW_CalcBubT(_xc(i), _P(i), _Tj(i), Nothing, False)
416                 _Tj(i) = tmp(4)
417                 CheckCalculatorStatus()
418             End If
419         Next
420
421         'step7
422
423         'calculate enthalpies
424
425         Dim Hv(_ns), Hl(_ns), Hidv(_ns), Hidl(_ns), DHv(_ns), DHl(_ns) As Double
426
427         For i = 0 To _ns
428             Hidv(i) = _pp.RET_Hid(298.15, _Tj(i), _yc(i))
429             Hidl(i) = _pp.RET_Hid(298.15, _Tj(i), _xc(i))
430             DHv(i) = _Cj(i) + _Dj(i) * (_Tj(i) - _T_(i))
431             DHl(i) = _Ej(i) + _Fj(i) * (_Tj(i) - _T_(i))
432             Hv(i) = (Hidv(i) + DHv(i)) * _pp.AUX_MMM(_yc(i)) / 1000
433             Hl(i) = (Hidl(i) + DHl(i)) * _pp.AUX_MMM(_xc(i)) / 1000
434             'If llextr Then
435             ' Hv(i) = _pp.DW_CalcEnthalpy(_yc(i), _Tj(i), _P(i), PropertyPackages.State.
                    Liquid) * _pp.AUX_MMM(_yc(i)) / 1000
436             'Else
437             ' Hv(i) = _pp.DW_CalcEnthalpy(_yc(i), _Tj(i), _P(i), PropertyPackages.State.
                    Vapor) * _pp.AUX_MMM(_yc(i)) / 1000
438             'End If
439             'Hl(i) = _pp.DW_CalcEnthalpy(_xc(i), _Tj(i), _P(i), PropertyPackages.State.
                    Liquid) * _pp.AUX_MMM(_xc(i)) / 1000
440             CheckCalculatorStatus()
441         Next
442
443         'reboiler and condenser heat duties
444         Select Case _coltype
445             Case Column.ColType.DistillationColumn
446                 If Not _specs("C").SType = ColumnSpec.SpecType.Heat_Duty Then
447                     _Q(0) = Hl(0) * _Rlj(0) * _Lj(0) + Hv(0) * _Rvj(0) * _Vj(0) - Hv(1) *
                            _Vj(1) - _HF(0) * _F(0)
448                     _Q(0) = -_Q(0)
449                 End If
450                 If Not _specs("R").SType = ColumnSpec.SpecType.Heat_Duty Then
451                     _Q(_ns) = Hl(_ns) * _Rlj(_ns) * _Lj(_ns) + Hv(_ns) * _Rvj(_ns) * _Vj(
                            _ns) - Hl(_ns - 1) * _Lj(_ns - 1) - _HF(_ns) * _F(_ns)
452                     _Q(_ns) = -_Q(_ns)
453                 End If
```

```
454             Case Column.ColType.AbsorptionColumn
455                 'use provided values
456             Case Column.ColType.RefluxedAbsorber
457                 If Not _specs("C").SType = ColumnSpec.SpecType.Heat_Duty Then
458                     _Q(0) = Hl(0) * _Rlj(0) * _Lj(0) + Hv(0) * _Rvj(0) * _Vj(0) - Hv(1) *
                            _Vj(1) - _HF(0) * _F(0)
459                     _Q(0) = -_Q(0)
460                 End If
461             Case Column.ColType.ReboiledAbsorber
462                 If Not _specs("R").SType = ColumnSpec.SpecType.Heat_Duty Then
463                     _Q(_ns) = Hl(_ns) * _Rlj(_ns) * _Lj(_ns) + Hv(_ns) * _Rvj(_ns) * _Vj(
                            _ns) - Hl(_ns - 1) * _Lj(_ns - 1) - _HF(_ns) * _F(_ns)
464                     _Q(_ns) = -_Q(_ns)
465                 End If
466         End Select
467
468         'handle user specs
469
470         'Condenser Specs
471         Select Case _specs("C").SType
472             Case ColumnSpec.SpecType.Component_Fraction
473                 If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                        condtype.Partial_Condenser Then
474                     If _specs("C").SpecUnit = "M" Then
475                         spfval1 = _xc(0)(spci1) - spval1
476                     Else 'W
477                         spfval1 = _pp.AUX_CONVERT_MOL_TO_MASS(_xc(0))(spci1) - spval1
478                     End If
479                 Else
480                     If _specs("C").SpecUnit = "M" Then
481                         spfval1 = _yc(0)(spci1) - spval1
482                     Else 'W
483                         spfval1 = _pp.AUX_CONVERT_MOL_TO_MASS(_yc(0))(spci1) - spval1
484                     End If
485                 End If
486             Case ColumnSpec.SpecType.Component_Mass_Flow_Rate
487                 If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                        condtype.Partial_Condenser Then
488                     spfval1 = _LSSj(0) * _xc(0)(spci1) - spval1 / _pp.RET_VMM()(spci1) *
                            1000 / _maxF
489                 Else
490                     spfval1 = _Vj(0) * _yc(0)(spci1) - spval1 / _pp.RET_VMM()(spci1) * 1000
                            / _maxF
491                 End If
492             Case ColumnSpec.SpecType.Component_Molar_Flow_Rate
493                 If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                        condtype.Partial_Condenser Then
494                     spfval1 = _LSSj(0) * _xc(0)(spci1) - spval1 / _maxF
495                 Else
496                     spfval1 = _Vj(0) * _yc(0)(spci1) - spval1 / _maxF
497                 End If
498             Case ColumnSpec.SpecType.Component_Recovery
499                 Dim rec As Double = spval1 / 100
500                 Dim sumc As Double = 0
```

```
501             For j = 0 To _ns
502                 sumc += _fc(j)(spci1)
503             Next
504             sumc *= rec
505             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
506                 If _specs("C").SpecUnit = "%␣M/M" Then
507                     spfval1 = _xc(0)(spci1) * _LSSj(0) - sumc
508                 Else '% W/W
509                     spfval1 = _pp.RET_VMM()(spci1) * 1000 * (_xc(0)(spci1) * _LSSj(0) -
                            sumc)
510                 End If
511             Else
512                 If _specs("C").SpecUnit = "%␣M/M" Then
513                     spfval1 = _yc(0)(spci1) * _Vj(0) - sumc
514                 Else '% W/W
515                     spfval1 = _pp.RET_VMM()(spci1) * 1000 * (_yc(0)(spci1) * _Vj(0) -
                            sumc)
516                 End If
517             End If
518         Case ColumnSpec.SpecType.Heat_Duty
519             _Q(0) = spval1 / _maxF
520         Case ColumnSpec.SpecType.Product_Mass_Flow_Rate
521             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
522                 spfval1 = _LSSj(0) - spval1 / _pp.AUX_MMM(_xc(0)) * 1000 / _maxF
523             Else
524                 spfval1 = _Vj(0) - spval1 / _pp.AUX_MMM(_yc(0)) * 1000 / _maxF
525             End If
526         Case ColumnSpec.SpecType.Product_Molar_Flow_Rate
527             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
528                 spfval1 = _LSSj(0) - spval1 / _maxF
529             Else
530                 spfval1 = _Vj(0) - spval1 / _maxF
531             End If
532         Case ColumnSpec.SpecType.Stream_Ratio
533             If _condtype = Column.condtype.Total_Condenser Then
534                 spfval1 = _Lj(0) - spval1 * _LSSj(0)
535             ElseIf _condtype = Column.condtype.Partial_Condenser Then
536                 spfval1 = _Lj(0) - spval1 * (_LSSj(0) + _Vj(0))
537             Else
538                 spfval1 = _Lj(0) - spval1 * _Vj(0)
539             End If
540         Case ColumnSpec.SpecType.Temperature
541             spfval1 = _Tj(0) - spval1
542     End Select
543
544     'Reboiler Specs
545     Select Case _specs("R").SType
546         Case ColumnSpec.SpecType.Component_Fraction
547             If _specs("R").SpecUnit = "M" Then
548                 spfval2 = _xc(_ns)(spci2) - spval2
549             Else 'W
```

```
550                 spfval2 = _pp.AUX_CONVERT_MOL_TO_MASS(_xc(_ns))(spci2) - spval2
551             End If
552         Case ColumnSpec.SpecType.Component_Mass_Flow_Rate
553             spfval2 = _Lj(_ns) * _xc(_ns)(spci2) - spval2 / _pp.RET_VMM()(spci2) * 1000
                    / _maxF
554         Case ColumnSpec.SpecType.Component_Molar_Flow_Rate
555             spfval2 = _Lj(_ns) * _xc(_ns)(spci2) - spval2 / _maxF
556         Case ColumnSpec.SpecType.Component_Recovery
557             Dim rec As Double = spval2 / 100
558             Dim sumc As Double = 0
559             For j = 0 To _ns
560                 sumc += _fc(j)(spci2)
561             Next
562             sumc *= rec
563             If _specs("R").SpecUnit = "% M/M" Then
564                 spfval2 = _lc(_ns)(spci2) - sumc
565             Else '% W/W
566                 spfval2 = _pp.RET_VMM()(spci2) * 1000 * (_lc(_ns)(spci2) - sumc)
567             End If
568         Case ColumnSpec.SpecType.Heat_Duty
569             _Q(_ns) = spval2 / _maxF
570         Case ColumnSpec.SpecType.Product_Mass_Flow_Rate
571             spfval2 = _Lj(_ns) - spval2 / _pp.AUX_MMM(_xc(_ns)) * 1000 / _maxF
572         Case ColumnSpec.SpecType.Product_Molar_Flow_Rate
573             spfval2 = _Lj(_ns) - spval2 / _maxF
574         Case ColumnSpec.SpecType.Stream_Ratio
575             spfval2 = _Vj(_ns) - spval2 * _Lj(_ns)
576         Case ColumnSpec.SpecType.Temperature
577             spfval2 = _Tj(_ns) - spval2
578     End Select
579
580     'enthalpy balances
581
582     Dim entbal(_ns) As Double
583
584     For i = 0 To _ns
585         If i = 0 Then
586             entbal(i) = (Hl(i) * _Rlj(i) * _Lj(i) + Hv(i) * _Rvj(i) * _Vj(i) - Hv(i +
                    1) * _Vj(i + 1) - _HF(i) * _F(i) - _Q(i))
587         ElseIf i = _ns Then
588             entbal(i) = (Hl(i) * _Rlj(i) * _Lj(i) + Hv(i) * _Rvj(i) * _Vj(i) - Hl(i -
                    1) * _Lj(i - 1) - _HF(i) * _F(i) - _Q(i))
589         Else
590             entbal(i) = (Hl(i) * _Rlj(i) * _Lj(i) + Hv(i) * _Rvj(i) * _Vj(i) - Hl(i -
                    1) * _Lj(i - 1) - Hv(i + 1) * _Vj(i + 1) - _HF(i) * _F(i) - _Q(i))
591         End If
592         entbal(i) /= (Hv(i) - Hl(i))
593     Next
594
595     Select Case _coltype
596         Case Column.ColType.DistillationColumn
597             entbal(0) = spfval1
598             entbal(_ns) = spfval2
599         Case Column.ColType.AbsorptionColumn
```

```
600             'do nothing
601         Case Column.ColType.ReboiledAbsorber
602             entbal(_ns) = spfval2
603         Case Column.ColType.RefluxedAbsorber
604             entbal(0) = spfval1
605     End Select
606
607     For i = 0 To x.Length - 1
608         If i <= _ns Then
609             errors(i) = entbal(i)
610         ElseIf i > _ns And i <= _vcnt + _ns Then
611             For j = 0 To _ns
612                 If _Rvj(j) <> 1 Then
613                     errors(i) = (_VSS(j) - _VSSj(j)) '/ _VSS(j)
614                     i += 1
615                 End If
616             Next
617         End If
618         If i > _vcnt + _ns And i <= _vcnt + _lcnt + _ns Then
619             For j = 1 To _ns
620                 If _Rlj(j) <> 1 Then
621                     errors(i) = (_LSS(j) - _LSSj(j)) '/ _LSS(j)
622                     i += 1
623                 End If
624             Next
625         End If
626     Next
627     If _condtype = Column.condtype.Partial_Condenser Then errors(_el) = (_Vj(0) - Vjj
        (0))
628     Return errors
629
630 End Function
631
632 Private Function FunctionGradient(ByVal x() As Double) As Double(,)
633
634     Dim epsilon As Double = ndeps
635     Dim hs As Double
636     Dim f1(), f2() As Double
637     Dim g(x.Length - 1, x.Length - 1), x1(x.Length - 1), x2(x.Length - 1) As Double
638     Dim i, j, k As Integer
639
640     f1 = FunctionValue(x)
641     For i = 0 To x.Length - 1
642         For j = 0 To x.Length - 1
643             If i <> j Then
644                 'x1(j) = x(j)
645                 x2(j) = x(j)
646             Else
647                 'x1(j) = x(j)
648                 'x2(j) = x(j) * (1 + epsilon) + (epsilon / 2) ^ 2
649                 x2(j) = x(j) + epsilon
650             End If
651         Next
652         f2 = FunctionValue(x2)
```

```
653             For k = 0 To x.Length - 1
654                 hs = epsilon
655                 g(k, i) = (f2(k) - f1(k)) / hs
656             Next
657         Next
658
659         Return g
660
661     End Function
662
663     Public Function MinimizeError(ByVal t As Double) As Double
664
665         Dim cv As New SistemasDeUnidades.Conversor
666         Dim spval1, spval2, spfval1, spfval2 As Double
667         Dim spci1, spci2 As Integer
668
669         spval1 = cv.ConverterParaSI(_specs("C").SpecUnit, _specs("C").SpecValue)
670         spci1 = _specs("C").ComponentIndex
671         spval2 = cv.ConverterParaSI(_specs("R").SpecUnit, _specs("R").SpecValue)
672         spci2 = _specs("R").ComponentIndex
673
674         Dim sum1(_ns) As Double
675         Dim i, j As Integer
676
677         For i = 0 To _ns
678             If i = 0 And _condtype <> Column.condtype.Full_Reflux Then
679                 _Rlj(i) = Exp(_bx(i) + _dbx(i) * t)
680             Else
681                 For j = 0 To _nc - 1
682                     _S(i, j) = Exp(_bx(i) + _dbx(i) * t) * _alpha(i, j) * _Sb
683                 Next
684             End If
685
686         Next
687
688         Dim m1, m2 As Integer
689
690         m1 = 0
691         m2 = 0
692
693         If _vcnt > 0 Then
694             For i = _ns + 1 To _vcnt + _ns
695                 For j = m1 To _ns
696                     If _Rvj(j) <> 1 Then
697                         m1 = j + 1
698                         Exit For
699                     End If
700                 Next
701                 _Rvj(m1 - 1) = Exp(_bx(i) + _dbx(i) * t)
702             Next
703         End If
704
705         If _lcnt > 0 Then
706             For i = _vcnt + _ns + 1 To _vcnt + _lcnt + _ns
```

```
            For j = m2 + 1 To _ns
                If _Rlj(j) <> 1 Then
                    m2 = j + 1
                    Exit For
                End If
            Next
            _Rlj(m2 - 1) = Exp(_bx(i) + _dbx(i) * t)
        Next
    End If
    If _condtype = Column.condtype.Partial_Condenser Then
        For j = 0 To _nc - 1
            _S(0, j) = Exp(_bx(_el) + _dbx(_el) * t) * _alpha(0, j) * _Sb
        Next
    End If

    'step4

    'find component liquid flows by the tridiagonal matrix method

    Dim Bs(_ns, _nc - 1), Cs(_ns, _nc - 1) As Double
    Dim at(_nc - 1)(), bt(_nc - 1)(), ct(_nc - 1)(), dt(_nc - 1)(), xt(_nc - 1)() As
        Double

    For i = 0 To _nc - 1
        Array.Resize(at(i), _ns + 1)
        Array.Resize(bt(i), _ns + 1)
        Array.Resize(ct(i), _ns + 1)
        Array.Resize(dt(i), _ns + 1)
        Array.Resize(xt(i), _ns + 1)
    Next

    Dim ic0 As Integer = 0

    For i = 0 To _ns
        For j = 0 To _nc - 1
            If i = 0 And _condtype = Column.condtype.Total_Condenser Then
                Bs(i, j) = -(_Rlj(i))
            Else
                Bs(i, j) = -(_Rlj(i) + _S(i, j) * _Rvj(i))
            End If
            If i < _ns Then Cs(i, j) = _S(i + 1, j)
        Next
    Next

    For i = 0 To _nc - 1
        For j = 0 To _ns
            dt(i)(j) = -_fc(j)(i) * _F(j)
            bt(i)(j) = Bs(j, i)
            If j < _ns Then ct(i)(j) = Cs(j, i)
            If j > 0 Then at(i)(j) = 1
        Next
    Next

    'solve matrices
```

```
760
761         'tomich
762         For i = 0 To _nc - 1
763             xt(i) = Tomich.TDMASolve(at(i), bt(i), ct(i), dt(i))
764             CheckCalculatorStatus()
765         Next
766
767         For i = 0 To _ns
768             Array.Resize(_xc(i), _nc)
769             Array.Resize(_yc(i), _nc)
770             Array.Resize(_lc(i), _nc)
771             Array.Resize(_vc(i), _nc)
772             Array.Resize(_zc(i), _nc)
773         Next
774
775         'step5
776
777         For i = 0 To _ns
778             _Lj(i) = 0
779             For j = 0 To _nc - 1
780                 'lc(i)(j) = lm(j)(i, 0)
781                 _lc(i)(j) = xt(j)(i)
782                 'If _lc(i)(j) < 0 Then _lc(i)(j) = 0
783                 _Lj(i) += _lc(i)(j)
784             Next
785             'If _Lj(i) < 0 Then _Lj(i) = 0.0000000001
786         Next
787
788         For i = 0 To _ns
789             _Vj(i) = 0
790             For j = 0 To _nc - 1
791                 _xc(i)(j) = _lc(i)(j) / _Lj(i)
792                 'If Double.IsNaN(_xc(i)(j)) Then _xc(i)(j) = 0
793             Next
794         Next
795
796         For i = _ns To 0 Step -1
797             _Vj(i) = 0
798             For j = 0 To _nc - 1
799                 If i < _ns Then
800                     If _eff(i) <> 1.0 Then
801                         _vc(i)(j) = _eff(i) * (_S(i, j) * _lc(i)(j) - _vc(i + 1)(j) * _Vj(i)
                                / _Vj(i + 1)) + _vc(i + 1)(j) * _Vj(i) / _Vj(i + 1)
802                     Else
803                         _vc(i)(j) = _S(i, j) * _lc(i)(j)
804                     End If
805                 Else
806                     _vc(i)(j) = _S(i, j) * _lc(i)(j)
807                 End If
808                 'If _vc(i)(j) < 0 Or Double.IsNaN(_vc(i)(j)) Then _vc(i)(j) = 0
809                 _Vj(i) += _vc(i)(j)
810             Next
811             'If _Vj(i) < 0 Then _Vj(i) = 0.0000000001
812         Next
```

```
813
814         'departures from product flows
815
816         Dim sumLSS As Double = 0
817         Dim sumVSS As Double = 0
818         Dim sumF As Double = 0
819         For i = 0 To _ns
820             If i > 0 Then sumLSS += _LSSj(i)
821             sumVSS += _VSSj(i)
822             sumF += _F(i)
823         Next
824         If _condtype = Column.condtype.Total_Condenser Then
825             _LSSj(0) = sumF - sumLSS - sumVSS - _Lj(_ns)
826             _Rlj(0) = 1 + _LSSj(0) / _Lj(0)
827         ElseIf _condtype = Column.condtype.Partial_Condenser Then
828             _LSSj(0) = sumF - sumLSS - sumVSS - _Vj(0) - _Lj(_ns)
829             _Rlj(0) = 1 + _LSSj(0) / _Lj(0)
830         Else
831             _LSSj(0) = 0.0
832             _Rlj(0) = 1
833         End If
834         'If _Lj(0) <> 0 Or Not Double.IsNaN(_Lj(0)) Or Not Double.IsInfinity(_Lj(0)) Then
835         ' _Rlj(0) = 1 + _LSSj(0) / _Lj(0)
836         'Else
837         ' _Rlj(0) = 1
838         'End If
839
840         For i = 0 To _ns
841             _VSSj(i) = (_Rvj(i) - 1) * _Vj(i)
842             If i > 0 Then _LSSj(i) = (_Rlj(i) - 1) * _Lj(i)
843         Next
844
845         'For i = 0 To _ns
846         ' sum1(i) = 0
847         ' For j = 0 To i
848         ' sum1(i) += _F(j) - _LSSj(j) - _VSSj(j)
849         ' Next
850         'Next
851
852         ''Ljs
853         'For i = 0 To _ns
854         ' If i < _ns Then _Lj(i) = _Vj(i + 1) + sum1(i) - _Vj(0) Else _Lj(i) = sum1(i) -
                _Vj(0)
855         'Next
856
857         For i = 0 To _ns
858             For j = 0 To _nc - 1
859                 If _Vj(i) <> 0 Then
860                     _yc(i)(j) = _vc(i)(j) / _Vj(i)
861                 Else
862                     _yc(i)(j) = _vc(i)(j)
863                 End If
864                 _zc(i)(j) = (_lc(i)(j) + _vc(i)(j)) / (_Lj(i) + _Vj(i))
865             Next
```

```
866         Next
867
868         Dim sum_axi(_ns) As Double
869
870         For i = 0 To _ns
871             sum_axi(i) = 0
872             For j = 0 To _nc - 1
873                 sum_axi(i) += _alpha(i, j) * _xc(i)(j)
874             Next
875         Next
876
877         'step6
878
879         'calculate new temperatures
880
881         Dim _Tj0(_ns) As Double
882
883         For i = 0 To _ns
884             _Kbj(i) = 1 / sum_axi(i)
885             _Tj0(i) = _Tj(i)
886             _Tj(i) = _Bj(i) / (_Aj(i) - Log(_Kbj(i)))
887             If Double.IsNaN(_Tj(i)) Or _Tj(i) = 0 Then _Tj(i) = _T_(i)
888         Next
889
890         'step7
891
892         'calculate enthalpies
893
894         Dim Hv(_ns), Hl(_ns), Hidv(_ns), Hidl(_ns), DHv(_ns), DHl(_ns) As Double
895
896         For i = 0 To _ns
897             Hidv(i) = _pp.RET_Hid(298.15, _Tj(i), _yc(i))
898             Hidl(i) = _pp.RET_Hid(298.15, _Tj(i), _xc(i))
899             DHv(i) = _Cj(i) + _Dj(i) * (_Tj(i) - _T_(i))
900             DHl(i) = _Ej(i) + _Fj(i) * (_Tj(i) - _T_(i))
901             Hv(i) = (Hidv(i) + DHv(i)) * _pp.AUX_MMM(_yc(i)) / 1000
902             Hl(i) = (Hidl(i) + DHl(i)) * _pp.AUX_MMM(_xc(i)) / 1000
903             'If llextr Then
904             ' Hv(i) = _pp.DW_CalcEnthalpy(_yc(i), _Tj(i), _P(i), PropertyPackages.State.
                    Liquid) * _pp.AUX_MMM(_yc(i)) / 1000
905             'Else
906             ' Hv(i) = _pp.DW_CalcEnthalpy(_yc(i), _Tj(i), _P(i), PropertyPackages.State.
                    Vapor) * _pp.AUX_MMM(_yc(i)) / 1000
907             'End If
908             'Hl(i) = _pp.DW_CalcEnthalpy(_xc(i), _Tj(i), _P(i), PropertyPackages.State.
                    Liquid) * _pp.AUX_MMM(_xc(i)) / 1000
909             CheckCalculatorStatus()
910         Next
911
912         'reboiler and condenser heat duties
913         Select Case _coltype
914             Case Column.ColType.DistillationColumn
915                 If Not _specs("C").SType = ColumnSpec.SpecType.Heat_Duty Then
916                     _Q(0) = Hl(0) * _Rlj(0) * _Lj(0) + Hv(0) * _Rvj(0) * _Vj(0) - Hv(1) *
```

```
                    _Vj(1) - _HF(0) * _F(0)
917                 _Q(0) = -_Q(0)
918             End If
919             If Not _specs("R").SType = ColumnSpec.SpecType.Heat_Duty Then
920                 _Q(_ns) = Hl(_ns) * _Rlj(_ns) * _Lj(_ns) + Hv(_ns) * _Rvj(_ns) * _Vj(
                        _ns) - Hl(_ns - 1) * _Lj(_ns - 1) - _HF(_ns) * _F(_ns)
921                 _Q(_ns) = -_Q(_ns)
922             End If
923         Case Column.ColType.AbsorptionColumn
924             'use provided values
925         Case Column.ColType.RefluxedAbsorber
926             If Not _specs("C").SType = ColumnSpec.SpecType.Heat_Duty Then
927                 _Q(0) = Hl(0) * _Rlj(0) * _Lj(0) + Hv(0) * _Rvj(0) * _Vj(0) - Hv(1) *
                        _Vj(1) - _HF(0) * _F(0)
928                 _Q(0) = -_Q(0)
929             End If
930         Case Column.ColType.ReboiledAbsorber
931             If Not _specs("R").SType = ColumnSpec.SpecType.Heat_Duty Then
932                 _Q(_ns) = Hl(_ns) * _Rlj(_ns) * _Lj(_ns) + Hv(_ns) * _Rvj(_ns) * _Vj(
                        _ns) - Hl(_ns - 1) * _Lj(_ns - 1) - _HF(_ns) * _F(_ns)
933                 _Q(_ns) = -_Q(_ns)
934             End If
935     End Select
936
937     'enthalpy balances
938
939     Dim entbal(_ns) As Double
940
941     'handle user specs
942
943     'Condenser Specs
944     Select Case _specs("C").SType
945         Case ColumnSpec.SpecType.Component_Fraction
946             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
947                 If _specs("C").SpecUnit = "M" Then
948                     spfval1 = _xc(0)(spci1) - spval1
949                 Else 'W
950                     spfval1 = _pp.AUX_CONVERT_MOL_TO_MASS(_xc(0))(spci1) - spval1
951                 End If
952             Else
953                 If _specs("C").SpecUnit = "M" Then
954                     spfval1 = _yc(0)(spci1) - spval1
955                 Else 'W
956                     spfval1 = _pp.AUX_CONVERT_MOL_TO_MASS(_yc(0))(spci1) - spval1
957                 End If
958             End If
959         Case ColumnSpec.SpecType.Component_Mass_Flow_Rate
960             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
961                 spfval1 = _LSSj(0) * _xc(0)(spci1) - spval1 / _pp.RET_VMM()(spci1) *
                        1000 / _maxF
962             Else
963                 spfval1 = _Vj(0) * _yc(0)(spci1) - spval1 / _pp.RET_VMM()(spci1) * 1000
```

```
                    / _maxF
964             End If
965         Case ColumnSpec.SpecType.Component_Molar_Flow_Rate
966             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
967                 spfval1 = _LSSj(0) * _xc(0)(spci1) - spval1 / _maxF
968             Else
969                 spfval1 = _Vj(0) * _yc(0)(spci1) - spval1 / _maxF
970             End If
971         Case ColumnSpec.SpecType.Component_Recovery
972             Dim rec As Double = spval1 / 100
973             Dim sumc As Double = 0
974             For j = 0 To _ns
975                 sumc += _fc(j)(spci1)
976             Next
977             sumc *= rec
978             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
979                 If _specs("C").SpecUnit = "%␣M/M" Then
980                     spfval1 = _xc(0)(spci1) * _LSSj(0) - sumc
981                 Else '% W/W
982                     spfval1 = _pp.RET_VMM()(spci1) * 1000 * (_xc(0)(spci1) * _LSSj(0) -
                            sumc)
983                 End If
984             Else
985                 If _specs("C").SpecUnit = "%␣M/M" Then
986                     spfval1 = _yc(0)(spci1) * _Vj(0) - sumc
987                 Else '% W/W
988                     spfval1 = _pp.RET_VMM()(spci1) * 1000 * (_yc(0)(spci1) * _Vj(0) -
                            sumc)
989                 End If
990             End If
991         Case ColumnSpec.SpecType.Heat_Duty
992             _Q(0) = spval1 / _maxF
993         Case ColumnSpec.SpecType.Product_Mass_Flow_Rate
994             If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
995                 spfval1 = _LSSj(0) - spval1 / _pp.AUX_MMM(_xc(0)) * 1000 / _maxF
996             Else
997                 spfval1 = _Vj(0) - spval1 / _pp.AUX_MMM(_yc(0)) * 1000 / _maxF
998             End If
999         Case ColumnSpec.SpecType.Product_Molar_Flow_Rate
1000            If _condtype = Column.condtype.Total_Condenser Or _condtype = Column.
                    condtype.Partial_Condenser Then
1001                spfval1 = _LSSj(0) - spval1 / _maxF
1002            Else
1003                spfval1 = _Vj(0) - spval1 / _maxF
1004            End If
1005        Case ColumnSpec.SpecType.Stream_Ratio
1006            If _condtype = Column.condtype.Total_Condenser Then
1007                spfval1 = _Lj(0) - spval1 * _LSSj(0)
1008            ElseIf _condtype = Column.condtype.Partial_Condenser Then
1009                spfval1 = _Lj(0) - spval1 * (_LSSj(0) + _Vj(0))
1010            Else
```

```
                spfval1 = _Lj(0) - spval1 * _Vj(0)
            End If
        Case ColumnSpec.SpecType.Temperature
            spfval1 = _Tj(0) - spval1
    End Select

    'Reboiler Specs
    Select Case _specs("R").SType
        Case ColumnSpec.SpecType.Component_Fraction
            If _specs("R").SpecUnit = "M" Then
                spfval2 = _xc(_ns)(spci2) - spval2
            Else 'W
                spfval2 = _pp.AUX_CONVERT_MOL_TO_MASS(_xc(_ns))(spci2) - spval2
            End If
        Case ColumnSpec.SpecType.Component_Mass_Flow_Rate
            spfval2 = _Lj(_ns) * _xc(_ns)(spci2) - spval2 / _pp.RET_VMM()(spci2) * 1000
                / _maxF
        Case ColumnSpec.SpecType.Component_Molar_Flow_Rate
            spfval2 = _Lj(_ns) * _xc(_ns)(spci2) - spval2 / _maxF
        Case ColumnSpec.SpecType.Component_Recovery
            Dim rec As Double = spval2 / 100
            Dim sumc As Double = 0
            For j = 0 To _ns
                sumc += _fc(j)(spci2)
            Next
            sumc *= rec
            If _specs("R").SpecUnit = "%␣M/M" Then
                spfval2 = _lc(_ns)(spci2) - sumc
            Else '% W/W
                spfval2 = _pp.RET_VMM()(spci2) * 1000 * (_lc(_ns)(spci2) - sumc)
            End If
        Case ColumnSpec.SpecType.Heat_Duty
            _Q(_ns) = spval2 / _maxF
        Case ColumnSpec.SpecType.Product_Mass_Flow_Rate
            spfval2 = _Lj(_ns) - spval2 / _pp.AUX_MMM(_xc(_ns)) * 1000 / _maxF
        Case ColumnSpec.SpecType.Product_Molar_Flow_Rate
            spfval2 = _Lj(_ns) - spval2 / _maxF
        Case ColumnSpec.SpecType.Stream_Ratio
            spfval2 = _Vj(_ns) - spval2 * _Lj(_ns)
        Case ColumnSpec.SpecType.Temperature
            spfval2 = _Tj(_ns) - spval2
    End Select

    For i = 0 To _ns
        If i = 0 Then
            entbal(i) = (Hl(i) * _Rlj(i) * _Lj(i) + Hv(i) * _Rvj(i) * _Vj(i) - Hv(i +
                1) * _Vj(i + 1) - _HF(i) * _F(i) - _Q(i))
        ElseIf i = _ns Then
            entbal(i) = (Hl(i) * _Rlj(i) * _Lj(i) + Hv(i) * _Rvj(i) * _Vj(i) - Hl(i -
                1) * _Lj(i - 1) - _HF(i) * _F(i) - _Q(i))
        Else
            entbal(i) = (Hl(i) * _Rlj(i) * _Lj(i) + Hv(i) * _Rvj(i) * _Vj(i) - Hl(i -
                1) * _Lj(i - 1) - Hv(i + 1) * _Vj(i + 1) - _HF(i) * _F(i) - _Q(i))
        End If
```

```
1061            entbal(i) = entbal(i) / (Hv(i) - Hl(i))
1062            Select Case _coltype
1063                Case Column.ColType.DistillationColumn
1064                    entbal(0) = spfval1
1065                    entbal(_ns) = spfval2
1066                Case Column.ColType.AbsorptionColumn
1067                    'do nothing
1068                Case Column.ColType.ReboiledAbsorber
1069                    entbal(_ns) = spfval2
1070                Case Column.ColType.RefluxedAbsorber
1071                    entbal(0) = spfval1
1072            End Select
1073        Next
1074
1075        For i = 0 To _ns
1076            _Tj(i) = _Tj0(i)
1077        Next
1078        Dim errors(_bx.Length - 1) As Double
1079
1080        For i = 0 To _bx.Length - 1
1081            If i <= _ns Then
1082                errors(i) = entbal(i)
1083            ElseIf i > _ns And i <= _vcnt + _ns Then
1084                For j = 0 To _ns
1085                    If _Rvj(j) <> 1 Then
1086                        errors(i) = (_VSS(j) - _VSSj(j)) '/ _VSS(j)
1087                        i += 1
1088                    End If
1089                Next
1090            End If
1091            If i > _vcnt + _ns And i <= _vcnt + _lcnt + _ns Then
1092                For j = 1 To _ns
1093                    If _Rlj(j) <> 1 Then
1094                        errors(i) = (_LSS(j) - _LSSj(j)) '/ _LSS(j)
1095                        i += 1
1096                    End If
1097                Next
1098            End If
1099        Next
1100        If _condtype = Column.condtype.Partial_Condenser Then errors(_el) = (_Vj(0) - Vjj
                (0))
1101
1102        Dim il_err As Double = 0
1103        For i = 0 To _el
1104            il_err += errors(i) ^ 2
1105        Next
1106
1107        Return il_err
1108
1109    End Function
1110
1111    Public Function Solve(ByVal nc As Integer, ByVal ns As Integer, ByVal maxits As
            Integer, _
1112                          ByVal tol As Array, ByVal F As Array, ByVal V As Array, _
```

```
                        ByVal Q As Array, ByVal L As Array, _
                        ByVal VSS As Array, ByVal LSS As Array, ByVal Kval()() As Double
                            , _
                        ByVal x()() As Double, ByVal y()() As Double, ByVal z()() As
                            Double, _
                        ByVal fc()() As Double, _
                        ByVal HF As Array, ByVal T As Array, ByVal P As Array, _
                        ByVal condt As DistillationColumn.condtype, _
                        ByVal eff() As Double, _
                        ByVal UseDampingFactor As Boolean, _
                        ByVal UseNewtonUpdate As Boolean, _
                        ByVal AdjustSb As Boolean, ByVal UseIJ As Boolean, _
                        ByVal coltype As Column.ColType, ByVal KbjWA As Boolean, _
                        ByVal pp As PropertyPackages.PropertyPackage, _
                        ByVal specs As Dictionary(Of String, SepOps.ColumnSpec), _
                        ByVal reuseJ As Boolean, ByVal jac0 As Object, _
                        ByVal epsilon As Double, _
                        ByVal maxvarchgfac As Integer, _
                        ByVal dfmin As Double, ByVal dfmax As Double, _
                        ByVal deltat_el As Double, _
                        Optional ByVal llex As Boolean = False) As Object

        Dim doparallel As Boolean = My.Settings.EnableParallelProcessing
        Dim poptions As New ParallelOptions() With {.MaxDegreeOfParallelism = My.Settings.
            MaxDegreeOfParallelism}

        llextr = llex 'liq-liq extractor

        ndeps = epsilon

        Dim brentsolver As New BrentOpt.BrentMinimize
        brentsolver.DefineFuncDelegate(AddressOf MinimizeError)

        Dim cv As New SistemasDeUnidades.Conversor
        Dim spval1, spval2 As Double
        Dim spci1, spci2 As Integer

        spval1 = cv.ConverterParaSI(specs("C").SpecUnit, specs("C").SpecValue)
        spci1 = specs("C").ComponentIndex
        spval2 = cv.ConverterParaSI(specs("R").SpecUnit, specs("R").SpecValue)
        spci2 = specs("R").ComponentIndex

        _ns = ns
        _nc = nc


        Dim ic, ec, iic As Integer
        Dim Tj(ns), Tj_ant(ns), T_(ns) As Double
        Dim Lj(ns), Vj(ns), xc(ns)(), yc(ns)(), lc(ns)(), vc(ns)(), zc(ns)() As Double
        Dim sum1(ns) As Double
        Dim i, j, w, m1, m2 As Integer

        'step0
        'normalize initial estimates
```

```
1164
1165        Dim maxF As Double = Common.Max(F)
1166
1167        For i = 0 To ns
1168            F(i) = F(i) / maxF
1169            HF(i) = HF(i) / 1000
1170            L(i) = L(i) / maxF
1171            V(i) = V(i) / maxF
1172            LSS(i) = LSS(i) / maxF
1173            VSS(i) = VSS(i) / maxF
1174            Q(i) = Q(i) / maxF
1175        Next
1176
1177        'step1
1178
1179        Dim sumF As Double = 0
1180        Dim sumLSS As Double = 0
1181        Dim sumVSS As Double = 0
1182        For i = 0 To ns
1183            sumF += F(i)
1184            If i > 0 Then sumLSS += LSS(i)
1185            sumVSS += VSS(i)
1186        Next
1187
1188        Dim B As Double
1189        If condt = Column.condtype.Total_Condenser Then
1190            B = sumF - sumLSS - sumVSS - LSS(0)
1191        ElseIf condt = Column.condtype.Partial_Condenser Then
1192            B = sumF - sumLSS - sumVSS - V(0) - LSS(0)
1193        Else
1194            B = sumF - sumLSS - sumVSS - V(0)
1195        End If
1196
1197        'step2
1198
1199        Dim lsi, vsi As New ArrayList
1200        Dim el As Integer
1201
1202        'size jacobian
1203
1204        el = ns
1205        For i = 0 To ns
1206            If VSS(i) <> 0 Then
1207                el += 1
1208                vsi.Add(i)
1209            End If
1210            If LSS(i) <> 0 And i > 0 Then
1211                el += 1
1212                lsi.Add(i)
1213            End If
1214        Next
1215        If condt = Column.condtype.Partial_Condenser Then el += 1
1216
1217        Dim hes(el, el) As Double
```

```
1218        Dim bx(el), bxb(el), bf(el), bfb(el), bp(el), bp_ant(el) As Double
1219
1220        Dim f1(el), f2(el), f3(el), f4(el) As Double
1221        Dim u As Integer = 0
1222
1223        'find Kbref
1224
1225        Dim Kbj(ns), Kbj_ant(ns) As Object
1226        Dim K(ns, nc - 1), K_ant(ns, nc - 1), K2j(ns, nc - 1) As Object
1227
1228        Dim Kw1(ns)(), Kw2(ns)() As Object
1229        Dim wi(ns, nc - 1), ti(ns, nc - 1), sumwi(ns), sumti(ns) As Double
1230        For i = 0 To ns
1231            Array.Resize(Kw1(i), nc)
1232            Array.Resize(Kw2(i), nc)
1233        Next
1234
1235        Dim tmp0 As Object = Nothing
1236
1237        For i = 0 To ns
1238            If Not llextr Then tmp0 = pp.DW_CalcKvalue(z(i), T(i), P(i))
1239            For j = 0 To nc - 1
1240                If Not llextr Then K(i, j) = tmp0(j) Else K(i, j) = Kval(i)(j)
1241                If Double.IsNaN(K(i, j)) Or Double.IsInfinity(K(i, j)) Or K(i, j) = 0 Then
                        K(i, j) = pp.AUX_PVAPi(j, T(i)) / P(i)
1242            Next
1243            CheckCalculatorStatus()
1244        Next
1245
1246        If KbjWA = False Then
1247            Kbj = CalcKbj1(ns, nc, K, z, y, T, P, pp)
1248        Else
1249            Kbj = CalcKbj2(ns, nc, K, z, y, T, P, pp)
1250        End If
1251
1252        'relative volatilities
1253
1254        Dim alpha(ns, nc - 1), alpha_ant(ns, nc - 1) As Double
1255
1256        For i = 0 To ns
1257            For j = 0 To nc - 1
1258                alpha(i, j) = K(i, j) / Kbj(i)
1259            Next
1260        Next
1261
1262        'initialize A/B/C/D/E/F
1263
1264        Dim Kbj1(ns), Kbj2(ns) As Object
1265        Dim Tj1(ns), Tj2(ns), Aj(ns), Bj(ns), Cj(ns), Dj(ns), Ej(ns), Fj(ns) As Double
1266        Dim Aj_ant(ns), Bj_ant(ns), Cj_ant(ns), Dj_ant(ns), Ej_ant(ns), Fj_ant(ns) As
            Double
1267        Dim Hl1(ns), Hl2(ns), Hv1(ns), Hv2(ns) As Double
1268        Dim K2(ns)() As Double
1269        Dim Hv(ns), Hl(ns), DHv(ns), DHl(ns), Hidv(ns), Hidl(ns) As Double
```

```
1270
1271        For i = 0 To ns
1272            T_(i) = T(i) - 1
1273            Tj(i) = T(i)
1274            'Kbjs, Ts
1275            Tj1(i) = T(i)
1276            Tj2(i) = T(i) + 1
1277            Kbj1(i) = Kbj(i)
1278            'new Ks
1279            If llextr Then
1280                K2(i) = pp.DW_CalcKvalue(x(i), y(i), Tj2(i), P(i), "LL")
1281                Hv1(i) = pp.DW_CalcEnthalpyDeparture(y(i), Tj1(i), P(i), PropertyPackages.
                        State.Liquid)
1282                Hv2(i) = pp.DW_CalcEnthalpyDeparture(y(i), Tj2(i), P(i), PropertyPackages.
                        State.Liquid)
1283            Else
1284                K2(i) = pp.DW_CalcKvalue(x(i), y(i), Tj2(i), P(i))
1285                Hv1(i) = pp.DW_CalcEnthalpyDeparture(y(i), Tj1(i), P(i), PropertyPackages.
                        State.Vapor)
1286                Hv2(i) = pp.DW_CalcEnthalpyDeparture(y(i), Tj2(i), P(i), PropertyPackages.
                        State.Vapor)
1287            End If
1288            Hl1(i) = pp.DW_CalcEnthalpyDeparture(x(i), Tj1(i), P(i), PropertyPackages.State
                    .Liquid)
1289            Hl2(i) = pp.DW_CalcEnthalpyDeparture(x(i), Tj2(i), P(i), PropertyPackages.State
                    .Liquid)
1290            For j = 0 To nc - 1
1291                K2j(i, j) = K2(i)(j)
1292                If Double.IsNaN(K2(i)(j)) Or Double.IsInfinity(K2(i)(j)) Then K2(i)(j) = pp
                        .AUX_PVAPi(j, T(i)) / P(i)
1293            Next
1294        Next
1295
1296        If KbjWA = False Then
1297            Kbj2 = CalcKbj1(ns, nc, K2j, z, y, Tj2, P, pp)
1298        Else
1299            Kbj2 = CalcKbj2(ns, nc, K2j, z, y, Tj2, P, pp)
1300        End If
1301
1302        For i = 0 To ns
1303            Bj(i) = Log(Kbj1(i) / Kbj2(i)) / (1 / Tj2(i) - 1 / Tj1(i))
1304            Aj(i) = Log(Kbj1(i)) + Bj(i) * (1 / Tj1(i))
1305            Dj(i) = (Hv1(i) - Hv2(i)) / (Tj1(i) - Tj2(i))
1306            Cj(i) = Hv1(i) - Dj(i) * (Tj1(i) - T_(i))
1307            Fj(i) = (Hl1(i) - Hl2(i)) / (Tj1(i) - Tj2(i))
1308            Ej(i) = Hl1(i) - Fj(i) * (Tj1(i) - T_(i))
1309        Next
1310
1311        'external loop
1312
1313        Dim Sb, sbf, sbf_ant, sbf_ant2, sbx, sbx_ant, sbx_ant2, fval As Double
1314        Dim SbOK As Boolean = True
1315        Dim BuildingJacobian As Boolean = False
1316
```

```
1317        If AdjustSb Then SbOK = False
1318
1319        Sb = 1
1320
1321        Dim el_err As Double = 0.0#
1322        Dim el_err_ant As Double = 0.0#
1323        Dim il_err As Double = 0.0#
1324        Dim il_err_ant As Double = 0.0#
1325
1326        'independent variables -> stripping and withdrawal factors
1327
1328        Dim Sbj(ns), lnSbj0(ns), lnSbj(ns), S(ns, nc - 1) As Double
1329        Dim Rvj(ns), Rlj(ns), lnRvj(ns), lnRlj(ns), lnRvj0(ns), lnRlj0(ns) As Double
1330        Dim VSSj(ns), LSSj(ns), PSbj As Double
1331        Dim Nss As Integer = ns + 1
1332        'Calculo de Sbj, Rlj y Rvj del Lazo Externo
1333
1334        For i = 0 To ns
1335            Sbj(i) = Kbj(i) * V(i) / L(i)
1336        Next
1337        If AdjustSb Then
1338            SbOK = False
1339            PSbj = 1
1340            For i = 0 To ns
1341                If i = 0 And condt = Column.condtype.Total_Condenser Then
1342                    Nss -= 1
1343                Else
1344                    PSbj *= Sbj(i)
1345                End If
1346            Next
1347            Sb = PSbj ^ (1 / (Nss))
1348        Else
1349            Sb = 1
1350        End If
1351
1352        For i = 0 To ns
1353            If Sbj(i) = 0 Then Sbj(i) = 1.0E-20
1354            lnSbj(i) = Log(Sbj(i))
1355            If V(i) <> 0 Then Rvj(i) = 1 + VSS(i) / V(i) Else Rvj(i) = 1
1356            lnRvj(i) = Log(Rvj(i))
1357            If L(i) <> 0 Then Rlj(i) = 1 + LSS(i) / L(i) Else Rlj(i) = 1
1358            lnRlj(i) = Log(Rlj(i))
1359            For j = 0 To nc - 1
1360                S(i, j) = Sbj(i) * alpha(i, j) * Sb
1361            Next
1362        Next
1363
1364        Dim vcnt, lcnt As Integer
1365
1366        vcnt = 0
1367        For i = 0 To ns
1368            If lnRvj(i) <> 0 And Not Double.IsInfinity(lnRvj(i)) Then
1369                vcnt += 1
1370            End If
```

```
1371        Next
1372
1373        lcnt = 0
1374        For i = 1 To ns
1375            If lnRlj(i) <> 0 And Not Double.IsInfinity(lnRlj(i)) Then
1376                lcnt += 1
1377            End If
1378        Next
1379
1380        'internal loop
1381
1382        Dim check1 As Boolean = False
1383        Dim num, denom, x0, fx0 As New ArrayList
1384
1385        w = 0
1386
1387 1: Dim ic0 As Integer = 0
1388
1389        Do
1390
1391            If Not SbOK Then
1392
1393                sbf_ant2 = sbf_ant
1394                sbf_ant = sbf
1395
1396                sumF = 0
1397                sumLSS = 0
1398                sumVSS = 0
1399                For j = 0 To _ns
1400                    sumF += F(j)
1401                    If j > 0 Then sumLSS += LSS(j)
1402                    sumVSS += VSS(j)
1403                Next
1404
1405                sbf = sumF - sumLSS - sumVSS - V(0) - L(ns) - LSS(0)
1406
1407                sbx_ant2 = sbx_ant
1408                sbx_ant = sbx
1409
1410                If ic0 > 1 Then
1411                    If Abs((-sbf * (sbx - sbx_ant2) / (sbf - sbf_ant2)) / sbx) > 1 Then
1412                        sbx = sbx_ant2 * 1.01
1413                    Else
1414                        sbx = sbx - sbf * (sbx - sbx_ant2) / (sbf - sbf_ant2)
1415                    End If
1416                Else
1417                    sbx = Sb * 1.01
1418                End If
1419
1420                If sbx < 0 Then sbx = Abs(sbx)
1421
1422                Sb = sbx
1423
1424                For i = 0 To ns
```

```
1425                Sbj(i) = Kbj(i) * V(i) / L(i)
1426                If Sbj(i) = 0 Then Sbj(i) = 1.0E-20
1427                lnSbj(i) = Log(Sbj(i))
1428                If V(i) <> 0 Then Rvj(i) = 1 + VSS(i) / V(i) Else Rvj(i) = 1
1429                lnRvj(i) = Log(Rvj(i))
1430                If L(i) <> 0 Then Rlj(i) = 1 + LSS(i) / L(i) Else Rlj(i) = 1
1431                lnRlj(i) = Log(Rlj(i))
1432                For j = 0 To nc - 1
1433                    S(i, j) = Sbj(i) * alpha(i, j) * Sb
1434                Next
1435            Next
1436        End If
1437
1438        If SbOK Then Exit Do
1439
1440        If sbx > 10 Then Sb = sbx_ant2
1441
1442        ic0 += 1
1443
1444        CheckCalculatorStatus()
1445
1446        If Double.IsNaN(sbf) Then Throw New Exception(DWSIM.App.GetLocalString("
               DCSbError"))
1447
1448    Loop Until Abs(sbf) < 0.001
1449
1450    SbOK = True
1451
1452    Dim fx(el), dfdx(el, el), dfdx_ant(el, el), dx(el), xvar(el), xvar_ant(el), itol
           As Double
1453    Dim jac As New Mapack.Matrix(el + 1, el + 1), hesm As New Mapack.Matrix(el + 1, el
           + 1)
1454    Dim perturb As Boolean = False, bypass As Boolean = False
1455
1456    ec = 0
1457    ic = 0
1458    Do
1459
1460        iic = 0
1461
1462        'step3
1463
1464
1465        For i = 0 To ns
1466            'store initial values
1467            lnSbj0(i) = lnSbj(i)
1468            lnRvj0(i) = lnRvj(i)
1469            lnRlj0(i) = lnRlj(i)
1470        Next
1471
1472        'update inner loop parameters
1473
1474        Dim lnSbj_ant(ns), lnRvj_ant(ns), lnRlj_ant(ns), df, df_ant, xlowbound As
               Double
```

```
1475
1476            df_ant = df
1477
1478            _Bj = Bj.Clone
1479            _Aj = Aj.Clone
1480            _Cj = Cj.Clone
1481            _Dj = Dj.Clone
1482            _Ej = Ej.Clone
1483            _Fj = Fj.Clone
1484            _eff = eff.Clone
1485            _Tj = Tj.Clone
1486            _T_ = T_.Clone
1487            _Lj = Lj.Clone
1488            _Vj = Vj.Clone
1489            Vjj = V.Clone
1490            _LSSj = LSS.Clone
1491            _VSSj = VSS.Clone
1492            _LSS = LSS.Clone
1493            _VSS = VSS.Clone
1494            _Rlj = Rlj.Clone
1495            _Rvj = Rvj.Clone
1496            _F = F.Clone
1497            _P = P.Clone
1498            _HF = HF.Clone
1499            _Q = Q.Clone
1500            _S = S.Clone
1501            _condtype = condt
1502            _alpha = alpha.Clone
1503            _fc = fc.Clone
1504            _xc = xc.Clone
1505            _yc = yc.Clone
1506            _lc = lc.Clone
1507            _vc = vc.Clone
1508            _zc = zc.Clone
1509            _Kbj = Kbj.Clone
1510            _pp = pp
1511            _coltype = coltype
1512            _bx = bx.Clone
1513            _dbx = bp.Clone
1514            _Sb = Sb
1515            _vcnt = vcnt
1516            _lcnt = lcnt
1517            _specs = specs
1518            _maxF = maxF
1519            _el = el
1520
1521            'solve using newton's method
1522
1523            For i = 0 To ns
1524                If i = 0 And condt <> Column.condtype.Full_Reflux Then
1525                    xvar(i) = lnRlj(i)
1526                Else
1527                    xvar(i) = lnSbj(i)
1528                End If
```

```
            Next

            m1 = 0

            If vcnt > 0 Then
                For i = ns + 1 To vcnt + ns
                    For j = m1 To ns
                        If Rvj(j) <> 1 Then
                            m1 = j + 1
                            Exit For
                        End If
                    Next
                    xvar(i) = lnRvj(m1 - 1)
                Next
            End If

            m2 = 0

            If lcnt > 0 Then
                For i = vcnt + ns + 1 To vcnt + lcnt + ns
                    For j = m2 + 1 To ns
                        If Rlj(j) <> 1 Then
                            m2 = j + 1
                            Exit For
                        End If
                    Next
                    xvar(i) = lnRlj(m2 - 1)
                Next
            End If
            If condt = Column.condtype.Partial_Condenser Then xvar(el) = lnSbj(0)

            ic0 = 0

            'first run (to initialize variables)
            'fx = Me.FunctionValue(xvar)

            Do

restart: fx = Me.FunctionValue(xvar)
                If UseNewtonUpdate Then
                    dfdx_ant = dfdx
                    dfdx = Me.FunctionGradient(xvar)
                    Dim success As Boolean
                    success = MathEx.SysLin.rsolve.rmatrixsolve(dfdx, fx, el + 1, dx)
                    If Not success Then
                        dfdx = dfdx_ant
                        success = MathEx.SysLin.rsolve.rmatrixsolve(dfdx, fx, el + 1, dx)
                    End If
                    For i = 0 To el
                        dx(i) = -dx(i)
                    Next
                    _bx = xvar.Clone
                    _dbx = dx.Clone
                Else
```

```
1583                If ic = 0 Then
1584                    If UseIJ Then
1585                        For i = 0 To el
1586                            For j = 0 To el
1587                                If i = j Then hes(i, j) = 1 Else hes(i, j) = 0
1588                            Next
1589                        Next
1590                    Else
1591                        dfdx = Me.FunctionGradient(xvar)
1592                        For i = 0 To el
1593                            For j = 0 To el
1594                                jac(i, j) = dfdx(i, j)
1595                            Next
1596                        Next
1597                        hesm = jac.Inverse
1598                        For i = 0 To el
1599                            For j = 0 To el
1600                                hes(i, j) = hesm(i, j)
1601                            Next
1602                        Next
1603                    End If
1604                    bx = xvar
1605                    bf = fx
1606                    Broyden.broydn(el, bx, bf, bp, bxb, bfb, hes, 0)
1607                    dx = bp
1608                Else
1609                    bx = xvar
1610                    bf = fx
1611                    Broyden.broydn(el, bx, bf, bp, bxb, bfb, hes, 1)
1612                    dx = bp
1613                End If
1614                _bx = xvar.Clone
1615                _dbx = bp.Clone
1616            End If
1617
1618            'this call to the brent solver calculates the damping factor which
                    minimizes the error (fval).
1619            itol = tol(0) * ns
1620            df = 1
1621            If UseDampingFactor Then fval = brentsolver.brentoptimize(dfmin, dfmax, tol
                    (0), df)
1622
1623            perturb = False
1624            bypass = False
1625            xlowbound = 0.1
1626            For i = 0 To el
1627                xvar_ant(i) = xvar(i)
1628                xvar(i) += dx(i) * df
1629
1630                'If Abs((dx(i) * df) / xvar_ant(i)) > 10 Then
1631                ' 'perturb = True
1632                ' xvar(i) = xvar_ant(i) - df * (xvar_ant(i) - xlowbound) * 0.5
1633                'End If
1634                'If Double.IsNaN(dx(i)) Or Double.IsInfinity(dx(i)) Then
```

```
1635                ' bypass = True
1636                'End If
1637            Next
1638
1639            'If perturb Then
1640            ' For i = 0 To el
1641            ' xvar(i) = xvar_ant(i) * (1 + 0.3 * Math.Sign(dx(i)))
1642            ' Next
1643            'End If
1644
1645            'If bypass Then
1646            ' For i = 0 To el
1647            ' xvar(i) = xvar_ant(i) * 0.95
1648            ' Next
1649            'End If
1650
1651
1652            il_err_ant = il_err
1653
1654            il_err = 0
1655            For i = 0 To el
1656                il_err += fx(i) ^ 2
1657            Next
1658
1659            ic += 1
1660            ic0 += 1
1661
1662            'If bypass Then GoTo restart
1663
1664            If ic0 >= maxits Then Throw New Exception(DWSIM.App.GetLocalString("
                    DCMaxIterationsReached"))
1665            If Double.IsNaN(il_err) Then Throw New Exception(DWSIM.App.GetLocalString("
                    DCGeneralError"))
1666            If MathEx.Common.AbsSum(dx) = 0.0# Or Abs((il_err - il_err_ant) / il_err) <
                     itol Then Exit Do
1667
1668
1669            CheckCalculatorStatus()
1670
1671        Loop Until il_err < itol
1672
1673        For i = 0 To ns
1674            If i = 0 And _condtype = Column.condtype.Total_Condenser Then
1675                lnRlj_ant(i) = lnRlj(i)
1676                lnRlj(i) = xvar(i)
1677                Rlj(i) = Exp(lnRlj(i))
1678            Else
1679                lnSbj_ant(i) = lnSbj(i)
1680                lnSbj(i) = xvar(i)
1681                Sbj(i) = Exp(lnSbj(i))
1682                For j = 0 To nc - 1
1683                    S(i, j) = Sbj(i) * alpha(i, j) * Sb
1684                Next
1685            End If
```

```
1686            Next
1687
1688            m1 = 0
1689
1690            If vcnt > 0 Then
1691                For i = ns To vcnt + ns
1692                    For j = m1 To ns
1693                        If Rvj(j) <> 1 Then
1694                            m1 = j + 1
1695                            Exit For
1696                        End If
1697                    Next
1698                    lnRvj_ant(m1 - 1) = lnRvj(m1 - 1)
1699                    lnRvj(m1 - 1) = xvar(i)
1700                    Rvj(m1 - 1) = Exp(lnRvj(m1 - 1))
1701                Next
1702            End If
1703
1704            m2 = 0
1705
1706            If lcnt > 0 Then
1707                For i = vcnt + ns + 1 To vcnt + lcnt + ns
1708                    For j = m2 + 1 To ns
1709                        If Rlj(j) <> 1 Then
1710                            m2 = j + 1
1711                            Exit For
1712                        End If
1713                    Next
1714                    lnRlj_ant(m2 - 1) = lnRlj(m2 - 1)
1715                    lnRlj(m2 - 1) = xvar(i)
1716                    Rlj(m2 - 1) = Exp(lnRlj(m2 - 1))
1717                Next
1718            End If
1719            If condt = Column.condtype.Partial_Condenser Then
1720                lnSbj_ant(0) = lnSbj(0)
1721                lnSbj(0) = xvar(el)
1722                Sbj(0) = Exp(lnSbj(0))
1723                For j = 0 To nc - 1
1724                    S(0, j) = Sbj(0) * alpha(0, j) * Sb
1725                Next
1726            End If
1727            ic += 1
1728            iic += 1
1729
1730            'step9 (external loop)
1731
1732            Tj_ant = Tj.Clone
1733            Tj = _Tj.Clone
1734            T_ = _T_.Clone
1735            Lj = _Lj.Clone
1736            Vj = _Vj.Clone
1737            Q = _Q.Clone
1738            LSSj = _LSSj.Clone
1739            VSSj = _VSSj.Clone
```

```
1740            xc = _xc.Clone
1741            yc = _yc.Clone
1742            lc = _lc.Clone
1743            vc = _vc.Clone
1744            zc = _zc.Clone
1745            Kbj = _Kbj.Clone
1746
1747            For i = 0 To ns
1748
1749                T_(i) = Tj(i)
1750                Tj1(i) = Tj(i)
1751                Tj2(i) = Tj(i) + deltat_el * (i + 1)
1752
1753            Next
1754
1755            'update external loop variables using rigorous models
1756
1757            el_err_ant = el_err
1758            el_err = 0
1759
1760            Dim tmp(ns) As Object
1761
1762            If doparallel Then
1763                If My.Settings.EnableGPUProcessing Then
1764                    My.MyApplication.gpu.EnableMultithreading()
1765                End If
1766                My.MyApplication.IsRunningParallelTasks = True
1767                Dim task1 As Task = Task.Factory.StartNew(Sub() Parallel.For(0, ns + 1,
                        poptions,
1768                                                                  Sub(ipar)
1769                                                                      If llextr Then
1770                                                                          tmp(ipar) = pp.DW_CalcKvalue(xc
                                                                                (ipar), yc(ipar), Tj(ipar),
                                                                                 P(ipar), "LL")
1771                                                                      Else
1772                                                                          tmp(ipar) = pp.DW_CalcKvalue(xc
                                                                                (ipar), yc(ipar), Tj(ipar),
                                                                                 P(ipar))
1773                                                                      End If
1774                                                                  End Sub))
1775                While Not task1.IsCompleted
1776                    Application.DoEvents()
1777                End While
1778                For i = 0 To ns
1779                    For j = 0 To nc - 1
1780                        K_ant(i, j) = K(i, j)
1781                        K(i, j) = tmp(i)(j)
1782                    Next
1783                    Kbj_ant(i) = Kbj(i)
1784                Next
1785                My.MyApplication.IsRunningParallelTasks = False
1786                If My.Settings.EnableGPUProcessing Then
1787                    My.MyApplication.gpu.DisableMultithreading()
1788                    My.MyApplication.gpu.FreeAll()
```

```
            End If
        Else
            For i = 0 To ns
                If llextr Then
                    tmp(i) = pp.DW_CalcKvalue(xc(i), yc(i), Tj(i), P(i), "LL")
                Else
                    tmp(i) = pp.DW_CalcKvalue(xc(i), yc(i), Tj(i), P(i))
                End If
                For j = 0 To nc - 1
                    K_ant(i, j) = K(i, j)
                    K(i, j) = tmp(i)(j)
                Next
                Kbj_ant(i) = Kbj(i)
            Next
        End If

        If KbjWA = False Then
            Kbj1 = CalcKbj1(ns, nc, K, zc, yc, Tj1, P, pp)
        Else
            Kbj1 = CalcKbj2(ns, nc, K, zc, yc, Tj1, P, pp)
        End If
        'Kbj1 = Kbj
        Kbj = Kbj1

        'update relative volatilities

        For i = 0 To ns
            For j = 0 To nc - 1
                alpha_ant(i, j) = alpha(i, j)
                alpha(i, j) = K(i, j) / Kbj(i)
                el_err += Abs((alpha(i, j) - alpha_ant(i, j)) / alpha_ant(i, j)) ^ 2
            Next
        Next

        For i = 0 To ns
            Sbj(i) = Kbj(i) * Vj(i) / Lj(i)
            If Sbj(i) = 0.0# Then Sbj(i) = 1.0E-20
            lnSbj(i) = Log(Sbj(i))
            If Vj(i) <> 0 Then Rvj(i) = 1 + VSSj(i) / Vj(i) Else Rvj(i) = 1
            lnRvj(i) = Log(Rvj(i))
            If Lj(i) <> 0 Then Rlj(i) = 1 + LSSj(i) / Lj(i) Else Rlj(i) = 1
            lnRlj(i) = Log(Rlj(i))
            For j = 0 To nc - 1
                S(i, j) = Sbj(i) * alpha(i, j) * Sb
            Next
        Next

        'update A/B/C/D/E/F

        If doparallel Then
            My.MyApplication.IsRunningParallelTasks = True
            If My.Settings.EnableGPUProcessing Then
                My.MyApplication.gpu.EnableMultithreading()
            End If
```

```
1843            Dim task1 As Task = Task.Factory.StartNew(Sub() Parallel.For(0, ns + 1,
                    poptions,
1844                                                  Sub(ipar)
1845                                                      'new Ks
1846                                                      K2(ipar) = pp.DW_CalcKvalue(xc(ipar
                                                              ), yc(ipar), Tj2(ipar), P(ipar)
                                                              )
1847                                                  End Sub))
1848            While Not task1.IsCompleted
1849                Application.DoEvents()
1850            End While
1851            For i = 0 To ns
1852                For j = 0 To nc - 1
1853                    K2j(i, j) = K2(i)(j)
1854                    If Double.IsNaN(K2(i)(j)) Or Double.IsInfinity(K2(i)(j)) Then K2(i)(
                            j) = pp.AUX_PVAPi(j, Tj(i)) / P(i)
1855                Next
1856            Next
1857            My.MyApplication.IsRunningParallelTasks = False
1858            If My.Settings.EnableGPUProcessing Then
1859                My.MyApplication.gpu.DisableMultithreading()
1860                My.MyApplication.gpu.FreeAll()
1861            End If
1862        Else
1863            For i = 0 To ns
1864
1865                'new Ks
1866                K2(i) = pp.DW_CalcKvalue(xc(i), yc(i), Tj2(i), P(i))
1867
1868                For j = 0 To nc - 1
1869                    K2j(i, j) = K2(i)(j)
1870                    If Double.IsNaN(K2(i)(j)) Or Double.IsInfinity(K2(i)(j)) Then K2(i)(
                            j) = pp.AUX_PVAPi(j, Tj(i)) / P(i)
1871                Next
1872
1873            Next
1874        End If
1875
1876        If doparallel Then
1877            My.MyApplication.IsRunningParallelTasks = True
1878            If My.Settings.EnableGPUProcessing Then
1879                My.MyApplication.gpu.EnableMultithreading()
1880            End If
1881            Dim task1 As Task = Task.Factory.StartNew(Sub() Parallel.For(0, ns + 1,
                    poptions,
1882                                                  Sub(ipar)
1883                                                      'enthalpies
1884                                                      If llextr Then
1885                                                          Hv1(ipar) = pp.
                                                                  DW_CalcEnthalpyDeparture(
                                                                  yc(ipar), Tj1(ipar), P(
                                                                  ipar), PropertyPackages.
                                                                  State.Liquid)
1886                                                          Hv2(ipar) = pp.
```

```
                                                    DW_CalcEnthalpyDeparture(
                                                    yc(ipar), Tj2(ipar), P(
                                                    ipar), PropertyPackages.
                                                    State.Liquid)
1887                                                Else
1888                                                    Hv1(ipar) = pp.
                                                        DW_CalcEnthalpyDeparture(
                                                        yc(ipar), Tj1(ipar), P(
                                                        ipar), PropertyPackages.
                                                        State.Vapor)
1889                                                    Hv2(ipar) = pp.
                                                        DW_CalcEnthalpyDeparture(
                                                        yc(ipar), Tj2(ipar), P(
                                                        ipar), PropertyPackages.
                                                        State.Vapor)
1890                                                End If
1891                                                Hl1(ipar) = pp.
                                                    DW_CalcEnthalpyDeparture(xc(
                                                    ipar), Tj1(ipar), P(ipar),
                                                    PropertyPackages.State.Liquid)
1892                                                Hl2(ipar) = pp.
                                                    DW_CalcEnthalpyDeparture(xc(
                                                    ipar), Tj2(ipar), P(ipar),
                                                    PropertyPackages.State.Liquid)
1893                                            End Sub))
1894            While Not task1.IsCompleted
1895                Application.DoEvents()
1896            End While
1897            My.MyApplication.IsRunningParallelTasks = False
1898            If My.Settings.EnableGPUProcessing Then
1899                My.MyApplication.gpu.DisableMultithreading()
1900                My.MyApplication.gpu.FreeAll()
1901            End If
1902        Else
1903            For i = 0 To ns
1904
1905                'enthalpies
1906                If llextr Then
1907                    Hv1(i) = pp.DW_CalcEnthalpyDeparture(yc(i), Tj1(i), P(i),
                           PropertyPackages.State.Liquid)
1908                    Hv2(i) = pp.DW_CalcEnthalpyDeparture(yc(i), Tj2(i), P(i),
                           PropertyPackages.State.Liquid)
1909                Else
1910                    Hv1(i) = pp.DW_CalcEnthalpyDeparture(yc(i), Tj1(i), P(i),
                           PropertyPackages.State.Vapor)
1911                    Hv2(i) = pp.DW_CalcEnthalpyDeparture(yc(i), Tj2(i), P(i),
                           PropertyPackages.State.Vapor)
1912                End If
1913                Hl1(i) = pp.DW_CalcEnthalpyDeparture(xc(i), Tj1(i), P(i),
                        PropertyPackages.State.Liquid)
1914                Hl2(i) = pp.DW_CalcEnthalpyDeparture(xc(i), Tj2(i), P(i),
                        PropertyPackages.State.Liquid)
1915
1916            Next
```

```
1917            End If
1918
1919            If KbjWA = False Then
1920                Kbj2 = CalcKbj1(ns, nc, K2j, zc, yc, Tj2, P, pp)
1921            Else
1922                Kbj2 = CalcKbj2(ns, nc, K2j, zc, yc, Tj2, P, pp)
1923            End If
1924
1925            Dim Aerr(ns), Berr(ns) As Double
1926
1927            For i = 0 To ns
1928                Bj_ant(i) = Bj(i)
1929                Bj(i) = Log(Kbj1(i) / Kbj2(i)) / (1 / Tj2(i) - 1 / Tj1(i))
1930                Berr(i) = Bj(i) - Bj_ant(i)
1931                Aj_ant(i) = Aj(i)
1932                Aj(i) = Log(Kbj1(i)) + Bj(i) * (1 / Tj1(i))
1933                Aerr(i) = Aj(i) - Aj_ant(i)
1934                Dj_ant(i) = Dj(i)
1935                Dj(i) = (Hv1(i) - Hv2(i)) / (Tj1(i) - Tj2(i))
1936                Cj_ant(i) = Cj(i)
1937                Cj(i) = Hv1(i) - Dj(i) * (Tj1(i) - T_(i))
1938                Fj_ant(i) = Fj(i)
1939                Fj(i) = (Hl1(i) - Hl2(i)) / (Tj1(i) - Tj2(i))
1940                Ej_ant(i) = Ej(i)
1941                Ej(i) = Hl1(i) - Fj(i) * (Tj1(i) - T_(i))
1942            Next
1943
1944            ec += 1
1945
1946            If ec >= maxits Then Throw New Exception(DWSIM.App.GetLocalString("
                    DCMaxIterationsReached"))
1947            If Double.IsNaN(el_err) Then Throw New Exception(DWSIM.App.GetLocalString("
                    DCGeneralError"))
1948
1949            If AdjustSb Then SbOK = False
1950            Sb = 1
1951
1952            CheckCalculatorStatus()
1953
1954        Loop Until Abs(el_err) < tol(1) * el
1955
1956        If Abs(il_err) > tol(0) * ns Then
1957            My.Application.ActiveSimulation.WriteToLog("The␣sum␣of␣squared␣absolute␣errors␣
                    (internal␣loop)␣isn't␣changing␣anymore.␣Final␣value␣is␣" & il_err & ".",
                    Color.Green, FormClasses.TipoAviso.Aviso)
1958        End If
1959
1960        ' finished, de-normalize and return arrays
1961
1962        For i = 0 To ns
1963            Lj(i) = Lj(i) * maxF
1964            Vj(i) = Vj(i) * maxF
1965            LSSj(i) = LSSj(i) * maxF
1966            VSSj(i) = VSSj(i) * maxF
```

```
1967            F(i) = F(i) * maxF
1968            L(i) = L(i) * maxF
1969            V(i) = V(i) * maxF
1970            LSS(i) = LSS(i) * maxF
1971            VSS(i) = VSS(i) * maxF
1972            Q(i) = Q(i) * maxF
1973        Next
1974
1975        If Not UseNewtonUpdate Then
1976            For i = 0 To el
1977                For j = 0 To el
1978                    hesm(i, j) = hes(i, j)
1979                Next
1980            Next
1981            jac = hesm.Inverse
1982            For i = 0 To el
1983                For j = 0 To el
1984                    dfdx(i, j) = jac(i, j)
1985                Next
1986            Next
1987        End If
1988
1989        Return New Object() {Tj, Vj, Lj, VSSj, LSSj, yc, xc, K, Q, ic, il_err, ec, el_err,
                dfdx}
1990
1991    End Function
1992
1993 End Class
```