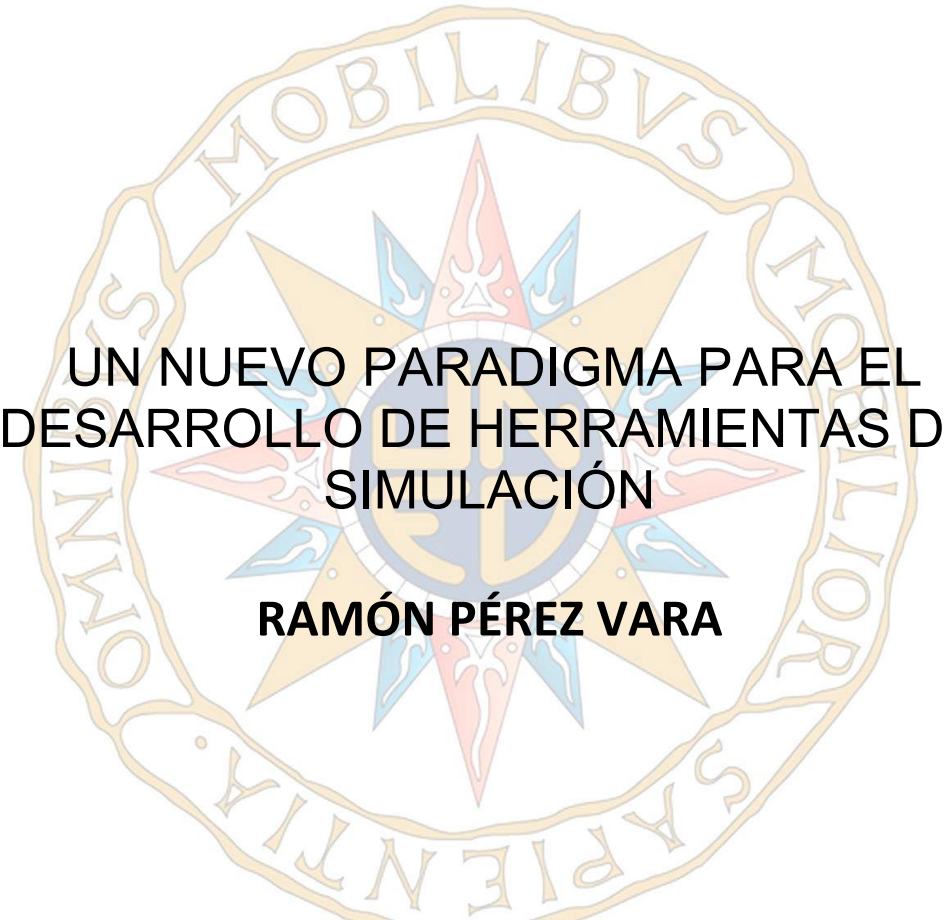
TESIS DOCTORAL

2024



UN NUEVO PARADIGMA PARA EL DESARROLLO DE HERRAMIENTAS DE SIMULACIÓN

RAMÓN PÉREZ VARA

PROGRAMA DE DOCTORADO EN INGENIERÍA DE SISTEMAS Y CONTROL

DIRECTOR: SEBASTIÁN DORMIDO BENCOMO

TESIS DOCTORAL

Un Nuevo Paradigma para el Desarrollo de Herramientas de Simulación

Autor: Ramón Pérez Vara

Ingeniero Superior Aeronáutico por la Universidad Politécnica de Madrid

Director: Sebastián Dormido Bencomo

Catedrático de Universidad

Departamento de Informática y Automática

E.T.S. de Ingeniería Informática, U.N.E.D.

Presentada en la

E.T.S DE INGENIERÍA INFORMÁTICA

de la

UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Madrid, 2024

A Sonia

AGRADECIMIENTOS

Quiero expresar mi más profundo agradecimiento al profesor Sebastián Dormido, director de esta tesis, cuya inquebrantable guía, apoyo y crítica constructiva han sido fundamentales para el desarrollo de este trabajo. Su confianza en mi labor ha sido un impulso indispensable. Asimismo, deseo reconocer al profesor Alfonso Urquía por las enriquecedoras conversaciones sobre simulación que hemos compartido y que han contribuido significativamente a la profundización de mis conocimientos en el tema.

En el ámbito profesional, quiero dar las gracias a todos mis colegas de Empresarios Agrupados, con quienes colaboré en la creación, desarrollo y aplicación posterior de EcosimPro. Su dedicación y trabajo conjunto fueron cruciales para alcanzar los objetivos planteados.

También quiero expresar mi gratitud a mi amigo y colega de trabajo, Raúl Avezuela, cuya amistad inquebrantable y aguda perspicacia en temas de simulación han sido invaluables. Sus contribuciones en las discusiones sobre los temas presentados en esta tesis y sus correcciones a los borradores han sido de gran importancia para la calidad final de este trabajo académico.

Doy las gracias muy especialmente a mis padres, Ramón y Consuelo, por el sacrificio que realizaron para brindarme la oportunidad de estudiar. A mi padre, que me transmitió su curiosidad insaciable y su ética de trabajo, y a mi madre que fue siempre un pilar de apoyo incondicional y cariño. Gracias también a mi hermana Consuelo quien no solo me ha respaldado siempre, sino que también ha aportado su perspectiva científica, heredada también de mi padre, en la revisión de los borradores de esta tesis.

Por último, muchas gracias a mi querida esposa, Sonia, por su amor incondicional y su comprensión del esfuerzo y el tiempo dedicado a esta tesis. Y a nuestro hijo Daniel, le agradezco su cariño y las enriquecedoras discusiones que mantenemos, incluso abordando algunos puntos de esta tesis.

RESUMEN

Este trabajo se enfoca en un tipo de herramientas denominado, herramientas de Modelado y Simulación Orientadas a Objeto (MSOO), concebidas para la simulación de sistemas cuyo comportamiento puede describirse mediante ecuaciones diferenciales, ecuaciones algebraicas y eventos discretos. EcosimPro y los diferentes entornos de simulación basados en el lenguaje Modelica son ejemplos de herramientas de modelado y simulación orientadas a objeto.

La adaptación de las herramientas de MSOO a campos específicos, en relación tanto al tipo de sistema simulado como a los fenómenos representados, se logra mediante el desarrollo de librerías de componentes, sub-modelos reutilizables diseñados que representan partes del sistema. En el contexto de esta tesis, se denomina *"aplicación de MSOO*" al conjunto de una herramienta de MSOO y una librería de simulación destinada a un campo particular.

El MSOO presenta un inconveniente crucial: al enfrentarse a un nuevo problema de simulación, el desarrollador tiende a adaptar el problema a las capacidades intrínsecas de la herramienta de MSOO, descartando posibles técnicas más eficientes presentes en herramientas específicas del campo. Un enfoque ideal para desarrollar una *aplicación de MSOO* para un campo particular implica un estudio exhaustivo del problema y su planteamiento considerando algoritmos de solución específicos, así como una interfaz de usuario optimizada, por ejemplo, en cuanto a mensajes de error. No seguir este enfoque resulta en aplicaciones menos eficientes y de difícil uso para usuarios no expertos en MSOO.

Las herramientas de MSOO ofrecen lenguajes de modelado y simulación con capacidades notoriamente superiores a las de las antiguas versiones del lenguaje Fortran, utilizadas para desarrollar la mayoría de las herramientas clásicas para la simulación de campos específicos. Al contar con un lenguaje más avanzado, parece factible construir *aplicaciones de MSOO* igualmente eficaces y fáciles de usar que las herramientas de simulación específicas. Así que la **pregunta básica** de esta investigación es la siguiente:

¿Sería posible construir *aplicaciones de MSOO* para campos específicos que fuesen igual de eficaces y amigables que los programas de simulación para campos específicos desarrollados utilizando lenguajes de programación genéricos?

Con el fin de responder a esta pregunta, el primer paso ha sido realizar una comparación entre herramientas de simulación específicas y *aplicaciones de MSOO* para identificar en detalle las causas por la cual las *aplicaciones de MSOO* resultan inferiores a las herramientas de simulación específicas. Dada la gran variedad de herramientas de simulación, ha sido preciso limitar dicha comparación a unos pocos campos. Los campos seleccionados en esta Tesis han sido los dos siguientes:

1. Programas de simulación estacionaria de procesos químicos y sistemas de producción de energía.
2. Simulación de transitorios hidráulicos, esto es el golpe de ariete.

A través de esta comparación, se han identificado las causas que contribuyen a que las *aplicaciones de MSOO* sean inferiores a los programas de simulación específicos. Seguidamente, se proponen estrategias y técnicas de programación en EcosimPro, una herramienta de MSOO, para superar estos inconvenientes.

Como ejemplos de esta estrategias y técnicas de programación, se presenta el desarrollo de dos *aplicaciones*: una *aplicación* para el cálculo de estacionarios hidráulicos y otra *aplicación* para el cálculo de transitorios hidráulicos. Estas *aplicaciones* se comparan con programas específicos y se demuestra que ofrecen características análogas a las de los programas de simulación para campos específicos.

ABSTRACT

This work focuses on a type of tools called Object-Oriented Modeling and Simulation (OOMS) tools, designed for simulating systems whose behavior can be described using differential equations, algebraic equations, and discrete events. EcosimPro and various simulation environments based on the Modelica language are examples of object-oriented modeling and simulation tools.

Adapting OOMS tools to specific fields, considering both the simulated system type and the represented phenomena, is achieved through the development of component libraries—reusable sub-models designed to represent specific parts of the system. In the context of this thesis, an "***OOMS application***" refers to the combination of an OOMS tool and a simulation library intended for a particular field.

OOMS presents a crucial drawback: when faced with a new simulation problem, developers tend to tailor the problem to the intrinsic capabilities of the OOMS tool, disregarding potentially more efficient techniques present in field-specific tools. An optimal approach to developing an *OOMS application* for a specific field involves a thorough study of the problem, considering specific solution algorithms and an optimized user interface, including well-crafted. Not following this approach results in less efficient applications that are challenging for non-expert users in OOMS.

OOMS tools offer modeling and simulation languages with significantly superior capabilities to the old versions of the FORTRAN language, which were used to develop most classical tools for simulating specific fields. Having a more advanced language makes it feasible to build *OOMS applications* that are equally effective and user-friendly as specific simulation tools. Therefore, the **fundamental question** of this research is as follows:

> Is it possible to build *OOMS applications* for specific fields that are as effective and user-friendly as simulation programs for specific fields developed using generic programming languages?

To answer this question, the first step has been to compare specific simulation tools with *OOMS applications* to identify in detail the reasons why *OOMS applications* are inferior to specific simulation tools. Given the wide variety of simulation tools, it was necessary to limit this comparison to a few fields. The selected fields in this thesis are as follows:

1. Steady-state simulation programs for chemical processes and energy production systems.
2. Simulation of hydraulic transients, i.e., water hammer.

Through this comparison, the causes contributing to the inferiority of OOMS *applications* to specific simulation programs have been identified. Subsequently, programming strategies and techniques are proposed in EcosimPro, an OOMS tool, to overcome these drawbacks.

As examples of these programming strategies and techniques, the development of two *applications* is presented: one for calculating hydraulic steady-states and another for calculating hydraulic transients. These *applications* are compared with specific programs, demonstrating that they offer features analogous to those of specific simulation programs for specific fields.

ÍNDICE

Índice de Figuras

Índice de Tablas

Índice de Listados de Código

1. INTRODUCCIÓN

1.1. Contexto

La mayoría de los programas de simulación actuales se han desarrollado utilizando lenguajes de programación de propósito general, como Fortran, Java, C y C++. Estos lenguajes proporcionan un control completo sobre el programa desarrollado, permitiendo al programador la libertad de diseñar las estructuras de datos, los mecanismos de intercambio de información entre módulos, los algoritmos de solución y la interfaz con el usuario. Esta flexibilidad brinda la oportunidad de incorporar conocimientos específicos del dominio de simulación tanto en los algoritmos de solución como en la interfaz con el usuario. La integración de este conocimiento del dominio se traduce en algoritmos de solución eficientes, precisos y robustos, y en interfaces de usuario que simplifican en gran medida la introducción de modelos y el análisis de los resultados.

El principal inconveniente para desarrollar programas de simulación utilizando lenguajes de simulación de propósito general es el enorme tiempo requerido para escribir, documentar y depurar el código. Este problema se ha visto exacerbado por la aparición de interfaces gráficas de usuario (IGU; en inglés GUI, *Graphical User Interface*). Las IGU facilitan enormemente el aprendizaje y manejo del software, pero han aumentado significativamente el esfuerzo necesario para desarrollar software de simulación.

Las herramientas de modelado y simulación orientadas a objetos (MSOO), como EcosimPro (EAI 2015) y los diferentes entornos de simulación basados en el lenguaje Modelica (Mattsson y Elmqvist, 1997), se crearon con el objetivo de reducir el esfuerzo necesario para desarrollar *aplicaciones* de simulación. Las herramientas de MSOO están diseñadas en torno a lenguajes concebidos para representar modelos mediante ecuaciones algebraicas diferenciales (DAE, Differential Algebraic Equations), y eventos discretos.

1.2. Evolución de las Herramientas de MSOO

Las herramientas de MSOO se consideran una evolución de las herramientas de simulación continua. A continuación se una visión panorámica que describe la evolución de las herramientas de simulación continua desde los primeros simuladores analógicos hasta las herramientas de MSOO, Astrom et al. (1998).

1.2.1. Simulación Analógica

En los primeros días de la simulación, se empleaban técnicas analógicas para representar y modelar sistemas mediante ecuaciones diferenciales ordinarias. Se construían

dispositivos físicos, generalmente circuitos electrónicos que hacían uso de amplificadores operacionales, para simular el comportamiento de dichas ecuaciones.

1.2.2. Primeros Simuladores Digítales y la Herencia Analógica

Con la aparición de los primeros ordenadores, surgieron numerosos programas de simulación que, si bien continuaban siguiendo el paradigma de la simulación analógica basada en diagramas de bloques, aprovechaban la nueva tecnología para sustituir el dispositivo físico (circuito eléctrico) por un software informático. Muy pronto, surgieron también herramientas basadas en ecuaciones y herramientas mixtas que permitían describir el modelo con diagramas de bloques y ecuaciones.

Tal y como era de esperar, la descripción de los modelos era diferente en cada una de estas herramientas, y esto conllevaba una falta de portabilidad de los modelos. Por este motivo, el Simulation Council Inc. (SCi) nominó un comité para la estandarización de los lenguajes de simulación continua. En 1967, se publicó el resultado de este proyecto de estandarización, la especificación CSSL'67 (Strauss (ed.), 1967).

El estándar CSSL estaba diseñado para permitir una descripción de los modelos bien mediante ecuaciones o bien mediante diagramas de bloques. En ambos casos, un modelo se representaba matemáticamente como un sistema de ecuaciones diferenciales ordinarias. Los objetivos del estándar CSSL'67 fueron tres:

1. Proporcionar herramientas de simulación sencillas, intuitivas y de uso simple para el usuario principiante.
2. Proporcionar flexibilidad y potencia para el modelado de sistemas complejos al usuario avanzado.
3. Prever una expansión flexible del sistema anticipándose a los futuros avances tecnológicos (por ejemplo, pantallas gráficas y ordenadores interactivos).

Entre las herramientas basadas en el especificación CSSL'67 merece mención especial ACSL (Mitchell & Gauthier, 1976), que añadía algunas mejoras y funcionalidades sobre el estándar y que se convirtió en la herramienta estándar de facto para simulación.

1.2.3. Herramientas de Simulación Basadas en Diagramas de Bloques

A pesar del estándar CSSL, las herramientas basadas en diagramas de bloques, procedentes de los antiguos simuladores analógicos, continúan siendo ampliamente utilizadas. Entre estas herramientas, Simulink (Grace, 1991) destaca como la más popular. Los diagramas de bloques tienen un flujo de cálculo unidireccional, desde las entradas hasta las salidas, lo que los hace altamente efectivos en la representación de sistemas de control. Sin embargo, resultan inadecuados para representar sistemas físicos como

circuitos eléctricos, circuitos de fluidos o sistemas mecánicos, ya que en estos sistemas físicos desaparece el concepto de entradas y salidas. Esto implica que el diagrama de bloques del modelo no se asemeja en absoluto al sistema físico que se está modelando (Otter & Cellier, 2019). A modo de ejemplo, la Figura 1-1 muestra un circuito eléctrico junto con su correspondiente modelo construido con diagrama de bloques.

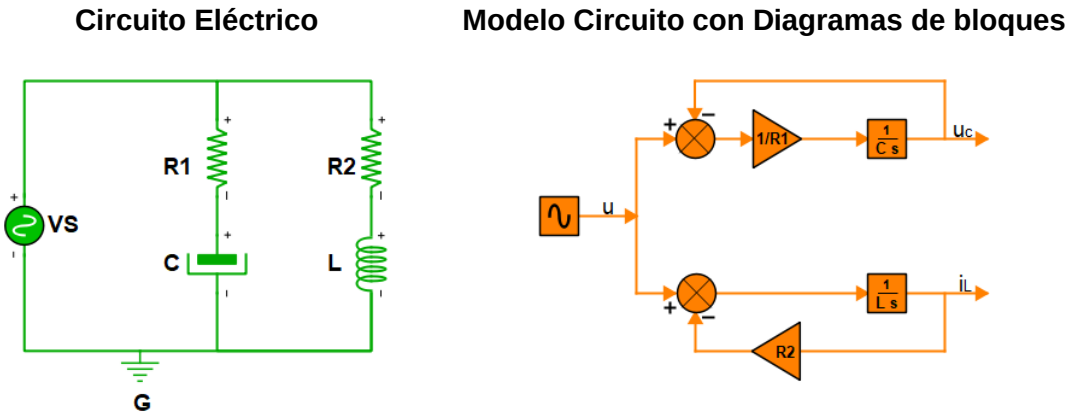


Figura 1-1: Circuito eléctrico (lado izquierdo) y su modelo mediante diagrama de bloques (lado derecho)

1.2.4. Herramientas de Simulación para Campos Específicos

La nueva tecnología de computación digital también permitió el desarrollo de herramientas de simulación adaptadas a campos específicos tales como la simulación de circuitos eléctricos, la transferencia de calor, el flujo de fluidos en redes de tuberías, los procesos químicos, etc. A continuación, se proporcionan algunos ejemplos de campos de simulación y de herramientas de simulación para dichos campos:

- Herramientas para la simulación de flujos en redes de tuberías, como FlowMaster (Mentor Graphics Corporation, 2016), AFT Impulse (Applied Flow Technology, 2017) y Pipenet (Sunrise, 2016).
- Herramientas para la simulación de circuitos eléctricos, como Spice (Nagel & Pederson, 1973).
- Herramientas para la simulación de turbinas de gas, como GasTurb (Kurzke, 1995), GSP (Visser & Broomhead, 2000), NPSS (Jones, 2007).
- Herramientas para la simulación de procesos químicos, como Aspen Plus (Aspen Technology, 2001), Aspen Hysys (HYSYS, n.d.), DYNSIM y COCO.
- Herramientas para la simulación del balance energético de plantas de producción de energía eléctrica, como GateCycle (LLC., n.d.) y Thermoflex (Thermoflow Inc., 2015).

Las herramientas de simulación desarrolladas para campos específicos son eficaces, esto es, consiguen una buena precisión con tiempos de cálculo moderados, y, al mismo tiempo,

son fáciles de usar, ya que el modelo se construye conectando módulos o componentes de tipos predefinidos en la herramienta. Sin embargo, es difícil modificar estas herramientas para incluir nuevos componentes o nuevos fenómenos (Andersson, 1990).

1.2.5. Aparición de las Herramientas de MSOO

Finalmente, emergió una generación de herramientas de simulación basadas en un nuevo paradigma: la modelización física orientada a objeto. Generalmente estas herramientas incluyen un lenguaje de modelización orientado a objeto para facilitar la creación de nuevos módulos o componentes. Dymola (Elmqvist, 1978), JModelica (Åkesson et al., 2009), OpenModelica (Fritzson et al., 2005) y EcosimPro (Empresarios Agrupados International, 2015) son ejemplos de entornos de modelado físico que incluyen lenguajes de modelización orientado a objeto. Dymola, JModelica y OpenModelica utilizan el lenguaje Modelica y EcosimPro utiliza un lenguaje propio denominado EL.

Lenguajes de Modelización Orientados a Objeto

La idea básica de las herramientas de modelado físico fue soportar la modelización modular, esto es la construcción de modelos conectando componentes o módulos, y permitir de manera fácil la creación de nuevos módulos. Este último requerimiento condujo a que las herramientas de modelado físico implementaron lenguajes de modelado orientados a objeto y con carácter no-causal para poder definir nuevos tipos de componentes y nuevos tipos de conexiones de forma sencilla y flexible.

Las herramientas de simulación genéricas anteriores se basaban en el paradigma del diagrama de bloques, en el cual un modelo se divide en bloques con señales de entrada y salida. Se asume que las señales de entrada son calculadas por otros bloques, y cada bloque calcula sus señales de salida. Este paradigma es especialmente adecuado para representar sistemas de control, ya que los componentes de dichos sistemas están diseñados para funcionar de esa manera. Sin embargo, al modelar sistemas físicos como circuitos eléctricos y termo-fluidos, las relaciones entre las entradas y salidas de flujos másicos y energéticos no siguen una causalidad computacional. Por ejemplo, una válvula ubicada en un sistema hidráulico influye en el flujo másico a través de ella y afecta el comportamiento de todos los componentes situados aguas arriba, sin obedecer una relación directa de causalidad computacional.

Cuando aparecen las herramientas de modelado físico, la programación orientada a objeto había avanzado de forma muy notable y se aplicaron conceptos similares a los de la programación orientada a objeto a los lenguajes de modelización. En la programación orientada a objetos, los programas se construyen con *objetos* que pertenecen a diferentes *clases*. Los principios de la programación orientada a objeto son:

- **Abstracción**: el principio de abstracción lo que implica es que la clase debe representar las características de la entidad hacia el mundo exterior, pero ocultando la complejidad que llevan aparejada.
- **Encapsulamiento**: la encapsulación es la característica de la programación orientada a objeto que permite que todo lo referente a un objeto quede aislado dentro de éste.
- **Herencia**: una clase hereda las características de una clase base o padre. Es un mecanismo que permite crear nuevas clases basadas en clases existentes.
- **Polimorfismo**: varios objetos de diferentes clases, pero con una base común, se pueden usar de manera indistinta,

Los lenguajes de modelado orientado a objeto han trasladado estos principios a la creación de modelos. Permiten definir clases de sub-modelos reusables (componentes) y construir modelos invocando objetos de estas clases.

AMESim, una notable herramienta de simulación sin lenguaje

Ahora bien, existen herramientas de simulación física que permiten desarrollar nuevos módulos pero que no cuentan con un lenguaje de modelización orientado a objeto, AMESim es una herramienta de este tipo, en la que los módulos nativos de la herramienta se desarrollan utilizando el lenguaje C. Cada componente en AMESim es un bloque de código en C que implementa las ecuaciones y algoritmos necesarios para representar el comportamiento de este. AMESim ha tenido un notable éxito comercial en el campo de la simulación física y el modelado de sistemas dinámicos. La herramienta ha sido ampliamente adoptada en diversas industrias, incluyendo la de automoción y la aeroespacial.

El éxito de AMESim se ha visto impulsado por su enfoque centrado en los usuarios. La herramienta se distingue por ofrecer una interfaz de usuario intuitiva y fácil de usar. Esta característica, combinada con la disponibilidad de librerías de componentes especializados para diversos campos de simulación, ha mejorado significativamente la productividad y eficiencia de los usuarios.

1.3. Motivación

En los últimos años, han surgido herramientas modernas de Modelado y Simulación Orientadas a Objetos, como EcosimPro (Empresarios Agrupados International, 2015) y varias herramientas basadas en el lenguaje Modelica (Fritzson & Engelson, 1998; Mattsson & Elmqvist, 1997). Estas herramientas han despertado un creciente interés en la comunidad de simulación debido a su enfoque genérico y su capacidad para adaptarse a diferentes campos.

Desde un punto de vista conceptual o teórico, las herramientas de MSOO podrían haber reemplazado a muchas herramientas de simulación específicas. Las herramientas de

MSOO están diseñadas para simular modelos representados mediante DAE's híbridos: ecuaciones algebraico diferenciales y eventos discretos. Dado que existen numerosas herramientas específicas de simulación cuya formulación cae dentro del ámbito de los DAE's híbridos, como las herramientas de cálculo de estacionarios que representan el modelo utilizando solo ecuaciones algebraicas, las herramientas de MSOO podrían haber proporcionado una solución más general y adaptable.

Sin embargo, a pesar de estas expectativas, las herramientas de MSOO todavía ocupan un nicho relativamente pequeño en el panorama general de las herramientas de simulación de sistemas. Por lo general, se recurre a ellas solamente cuando no hay disponibles herramientas de simulación específicas adecuadas o cuando se requiere llevar a cabo simulaciones multidisciplinarias que abarcan múltiples dominios. Esto se debe a que se considera que son más difíciles de usar, requieren un conocimiento experto, y se perciben como menos eficaces en términos de precisión y tiempo de cálculo en comparación con las herramientas específicas.

El propósito de esta Tesis es primeramente identificar por qué las herramientas modernas de modelado y simulación orientadas a objeto resultan inferiores a las herramientas de simulación específicas y no han conseguido desplazarlas, y una vez identificadas dichas causas contestar a la siguiente pregunta:

¿Sería posible construir *aplicaciones de MSOO* para campos específicos que fuesen igual de eficaces y amigables que los programas de simulación para campos específicos desarrollados utilizando lenguajes de programación genéricos?

1.4. Estructura de la Tesis

La presente tesis está estructurada en capítulos, cuyos contenidos se detallan a continuación:

Capítulo 1: Introducción

Capítulo 2: Descripción general de las herramientas de modelado y simulación orientadas a objetos.

Capítulo 3: Visión general de las características específicas de las herramientas de simulación estacionaria, con un enfoque especial en los métodos de solución utilizados por las herramientas de dominio específico.

Capítulo 4: Comparación entre herramientas de simulación específicas y aplicaciones equivalentes de MSOO, con el objetivo de identificar diferencias y puntos de mejora.

Capítulo 5: Propuesta de una serie de técnicas para el desarrollo de aplicaciones de MSOO que sean comparables en términos de facilidad de uso y eficacia a las herramientas de simulación específicas.

Capítulo 6: Aplicación de las técnicas propuestas en el capítulo 5 para el desarrollo de una librería destinada al cálculo de estados estacionarios en redes de tuberías con líquidos.

Capítulo 7: Aplicación de las técnicas propuestas en el capítulo anterior para el desarrollo de una librería destinada a la simulación de transitorios en redes de tuberías con líquidos.

Capítulo 8: Conclusiones y futuras líneas de investigación.

2. MODELADO Y SIMULACIÓN ORIENTADO A OBJETO

Las herramientas MSOO suelen ofrecer un lenguaje de modelado, orientado a objetos, y un entorno de simulación con dos partes claramente diferenciadas: resolvedores y algoritmos para encontrar buenas soluciones al problema de simulación matemática, y una interfaz gráfica de usuario con herramientas interactivas para construir modelos, ejecutar simulaciones y analizar los resultados.

2.2. Lenguajes de Modelado y Simulación Orientados a Objeto

La idea principal detrás de las herramientas MSOO es facilitar la creación de bibliotecas de modelos de unidades reutilizables, generalmente denominados tipos o clases de componentes, por medio de lenguajes de modelado declarativo. Luego se pueden construir modelos complejos instanciando y conectando componentes en un proceso conocido como agregación.

Las herramientas MSOO se pueden adaptar para simular dominios específicos. Los componentes generalmente representan partes físicas del sistema modelado. Por ejemplo, una biblioteca para simulación hidráulica incluye componentes como tuberías, válvulas, uniones, tanques y bombas.

Los componentes tienen "puertos de conexión" que son puntos terminales para diferentes tipos de conexiones, como conexiones para el flujo de fluidos, conexiones de transferencia de calor, conexiones de corriente eléctrica (cables), enlaces mecánicos y señales de control. Un modelo de un sistema se construye a nivel topológico (o nivel de diagrama de flujo) insertando símbolos que representan componentes y conectando dichos símbolos de la misma manera que los componentes están conectados en el sistema real.

Los lenguajes MSOO son declarativos, y permiten definir nuevos tipos de puertos y nuevos tipos de componentes.

Un tipo de puerto se define mediante un conjunto de variables y la declaración de las ecuaciones que ligan las variables de puertos conectados. Las ecuaciones de conexionado más comunes son relaciones de igualdad para el caso de variables que representan un esfuerzo, como el voltaje en circuitos eléctricos y la presión en circuitos fluidos, y ecuaciones de suma nula para el caso de variables que representan un flujo de masa o energía. La declaración de las variables del puerto incluye prefijos para indicar las variables de tipo “esfuerzo” y de tipo “flujo”. La Figura 2-1 muestra un ejemplo de declaración de un puerto hidráulico en el lenguaje de EcosimPro, donde la variable `H`, que representa la altura piezométrica, tiene el prefijo **`EQUAL`** para indicar que todas las alturas piezométricas en puertos conectados son iguales y la variable `Q`, que representa el flujo volumétrico tiene el

prefijo **SUM** para indicar que la suma de flujos desde puertos conectados debe ser nula. Aparte de las ecuaciones de conexión, que ligan las variables de puertos conectados, también es posible declarar ecuaciones que ligan las ecuaciones de un puerto individual, como la última ecuación que liga la presión `P` con la altura piezométrica `H`, la altura geométrica `z` y dos constantes, la densidad `rho` y la gravedad `GRAV`.

```
PORT hydraulic
      SUM   REAL Q   UNITS  "m3/s"    "Volume flow"
      EQUAL REAL H   UNITS  "m"       "Piezometric head"
      EQUAL REAL z   UNITS  "m"       "Elevation"
            REAL P   UNITS  "MPa(g)"  "Gauge Pressure"
   CONTINUOUS
      P = rho * GRAV * (H-z) /1.e6
END PORT
```

Figura 2-1: Ejemplo de Declaración en EcosimPro de un tipo de Conexión Hidráulica

El otro ejemplo es la declaración en lenguaje EcosimPro de un componente hidráulico que representa una válvula de salida hidráulica, cuyo código se muestra en la Figura 2-2. La declaración de un tipo de componente incluye la declaración de los puertos, datos y variables internas del componente. seguido de las ecuaciones continuas, que pueden ser algebraicas o diferenciales. Opcionalmente, los componentes de EcosimPro también pueden incluir un bloque **INIT** con sentencias que se ejecutan al inicio de la simulación, un bloque **DISCRETE** en el que se definen eventos discretos y sus acciones asociadas, y un bloque **TOPOLOGY**, donde es posible definir una agregación de componentes (esto es, un conjunto de componentes conectados entre sí, aunque el ejemplo presentado no los incluye).

```
COMPONENT ExitValve
   PORTS
      IN  hydraulic h_in
   DATA
      REAL Cd_A = 0.1   UNITS "m2"  "Effective exit area"
      REAL z = 0.       UNITS "m"   "Valve elevation"
   DECLS
      BOUND REAL pos    UNITS "p.u."   "Valve position"
      REAL Q            UNITS "m3/s"   "Outlet flow"
   CONTINUOUS
      f_in.Q = Cd_A * pos * sqrt(2.*G) *ssqrt(f_in.H- f_out.H)
      f_out.Q = Q
END COMPONENT
```

*Figura 2-2: Ejemplo en EcosimPro de la Declaración de un componente "**ExitValve**" (Válvula de Salida) de una Librería Hidráulica*

2.3. Entorno de Simulación de las Herramientas de MSOO

Además de proporcionar un lenguaje de modelado, una herramienta MSOO ofrece un entorno de simulación que incluye un editor de modelos gráficos, un lenguaje de

secuencias de comandos de experimentos para definir las simulaciones que se realizarán y una herramienta gráfica para trazar y mostrar los resultados de la simulación.

Las herramientas basadas en Modelica y EcosimPro son ejemplos de herramientas de modelado orientadas a objetos. Modelica es un lenguaje de modelado estándar y no patentado, orientado a objetos, para el cual existen diversos entornos de simulación disponibles.

EcosimPro es una herramienta MSOO con su propio lenguaje de modelado. También proporciona un lenguaje de descripción de experimentos dedicado que puede configurar diferentes escenarios de simulación. Los escenarios pueden ser simples, como el cálculo de estados estacionarios o soluciones transitorias, o complejos, como la solución de problemas de optimización. El entorno de simulación de EcosimPro ofrece un editor gráfico de modelos (ver Figura 2-3) que se puede utilizar para crear modelos arrastrando y soltando componentes de una paleta en un lienzo, conectando los puertos y rellenando datos en los componentes; y también ofrece una herramienta de monitorización para analizar los resultados, que permite la generación de gráficas e informes (ver Figura 2-4).

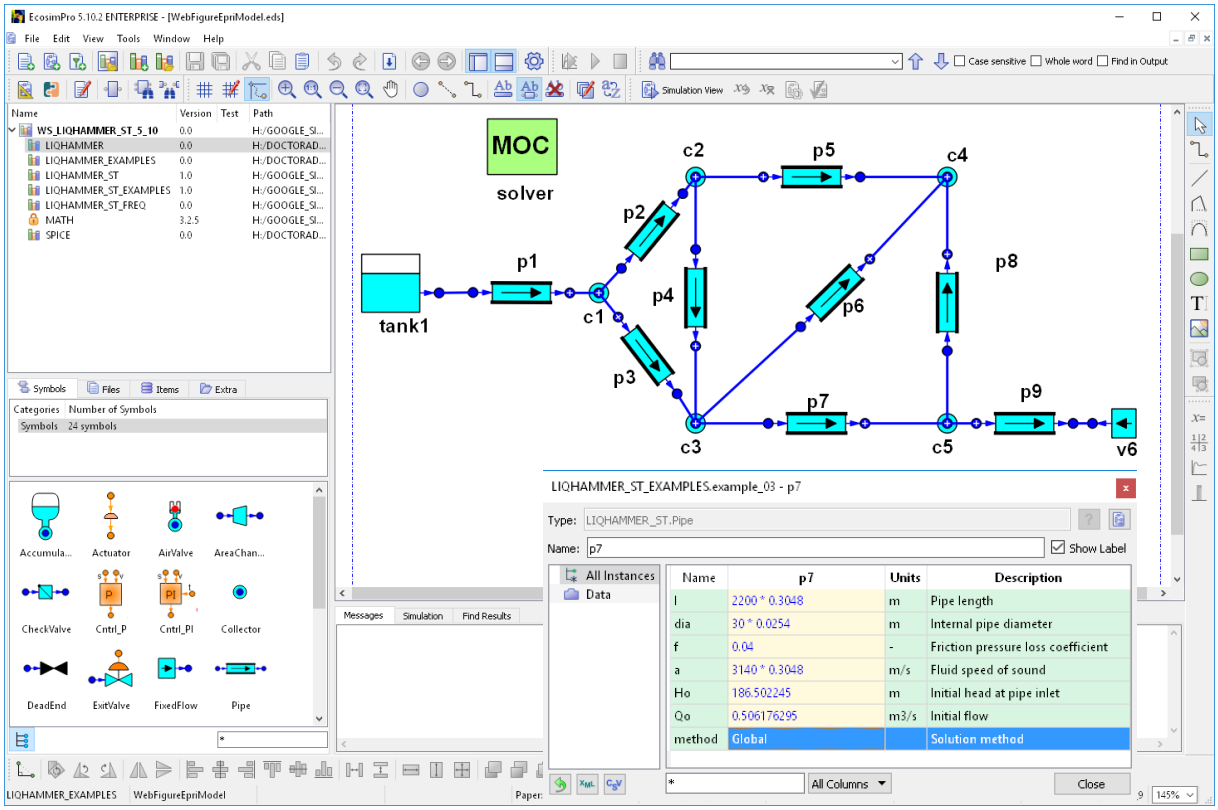


Figura 2-3: Ventana para la Creación de Modelos del Entorno de Simulación de EcosimPro

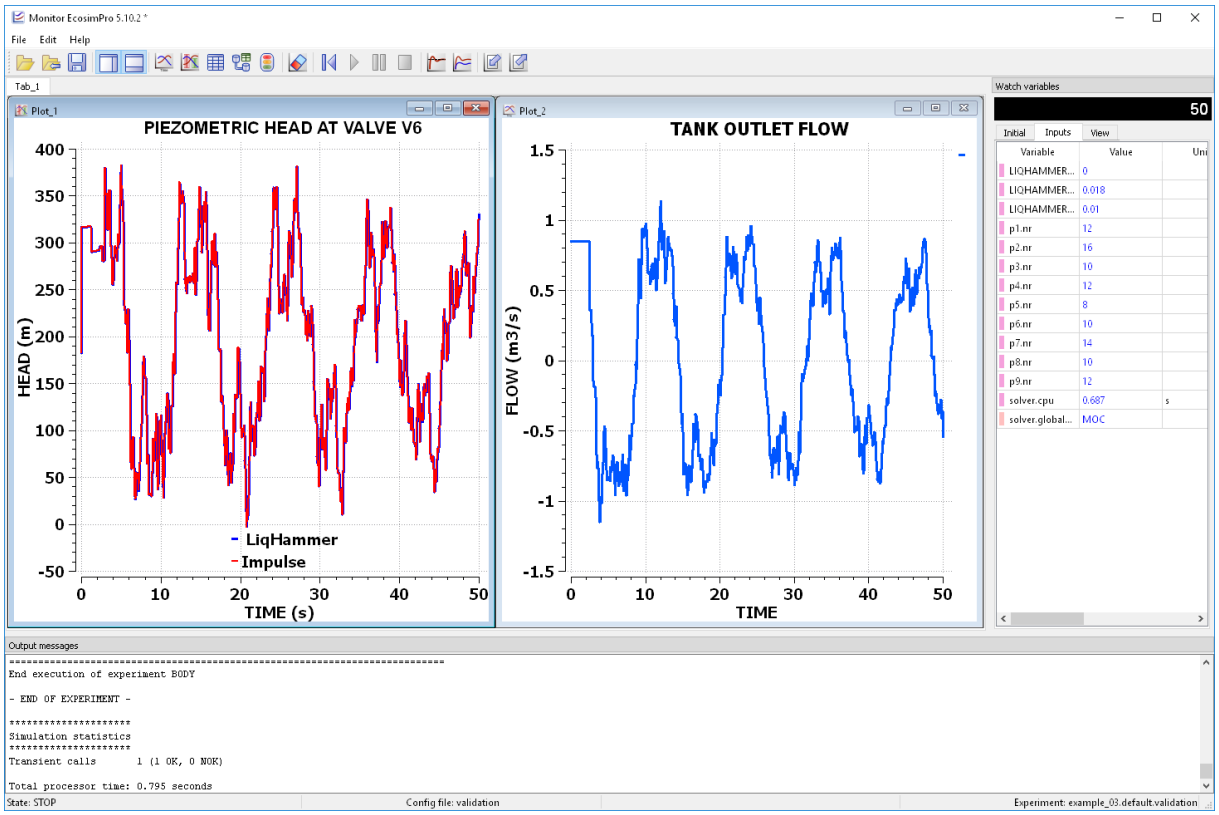


Figura 2-4: Herramienta de Monitorización de Resultados del Entorno de Simulación de EcosimPro

2.4. Formulación Matemática de las Herramientas de MSOO

2.4.1. Ecuaciones algebraicas diferenciales

El formalismo matemático que ha posibilitado la aparición de las herramientas de simulación orientadas a objeto ha sido los sistemas de ecuaciones diferenciales y algebraicas o DAE´s (Differential and Algebraic Equations). En las herramientas de modelado y simulación orientadas a objeto, una conexión entre componentes se considera como unas ecuaciones de ligadura entre variables de los componentes conectados; esto conduce de forma natural a formulaciones matemáticas del tipo DAE. Este enfoque es muy conveniente para construir bibliotecas o librerías de modelos reutilizables (Astrom et al., 1998).

Desde un punto de vista matemático, las herramientas MSOO representan el modelo como un sistema de ecuaciones algebraicas diferenciales (DAE's) con discontinuidades. La forma general de un sistema de ecuaciones algebraico diferenciales es:

$$\boldsymbol{F}(\boldsymbol{x}', \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p}, t) = \boldsymbol{0} \tag{2-1}$$

$$\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p}, t) = \boldsymbol{0} \tag{2-2}$$

donde $\boldsymbol{x'}$ son las derivadas de las variables de estado $\boldsymbol{x, z}$ son las variables algebraicas, $\boldsymbol{p}$ son las variables discretas y t es el tiempo. Los DAE's consisten en ecuaciones diferenciales (el vector $\boldsymbol{F}$) y restricciones algebraicas (el vector $\boldsymbol{G}$), siendo una generalización de las ecuaciones diferenciales ordinarias (ODE's). Se utilizan para representar el comportamiento dinámico continuo del sistema. Los primeros métodos numéricos para la integración de DAE's aparecieron en la década de 1970. Merece mención especial DASSL, que fue el algoritmo por defecto en muchas de las herramientas de simulación orientadas a objeto como Dymola, EcosimPro y Amesim.

2.4.2. Eventos y Comportamiento Discontinuo

Los eventos y sus acciones asociadas definen el comportamiento discreto del sistema. Los eventos tienen una duración cero y consisten en una condición de activación y la acción que se realizará cuando se active la acción. La condición de disparo es una expresión booleana y el evento ocurre cuando la condición cambia de falsa a verdadera. Según la condición de activación, los eventos se pueden clasificar en eventos de tiempo y eventos de estado.

Un evento de tiempo es aquel cuya condición de disparo depende únicamente de la variable de tiempo; por lo tanto, su momento de ocurrencia se conoce de antemano, y el tiempo final del paso de integración se puede establecer igual al tiempo del evento. Por el contrario, la condición de activación de un evento de estado depende de cualquier variable continua en el tiempo $(\boldsymbol{x', x, z})$, por lo que su momento de ocurrencia no se puede predecir de antemano. Las condiciones de activación de los eventos de estado se monitorizan al final de los pasos de integración del resolvedor DAE. Si alguna condición cambia a verdadera, la herramienta encuentra el tiempo preciso de ocurrencia del evento y ajusta el final del último paso de integración al tiempo de ocurrencia del evento.

La acción de evento describe cómo se relacionan entre sí el estado nuevo y el antiguo. Los eventos en las herramientas MSOO se especifican mediante cláusulas condicionales, ecuaciones condicionales y asignaciones retrasadas.

Con las cláusulas **`WHEN`**, el valor de las variables discretas se puede cambiar y el valor de las variables de estado se puede reinicializar en el momento en que se produce la condición. El pseudocódigo de una cláusula **`WHEN`** es el siguiente:

$$\begin{aligned} &\boldsymbol{WHEN}\ condition\ \boldsymbol{THEN} \\ &\quad \boldsymbol{x_a} = \boldsymbol{\delta_1}\left(\boldsymbol{x_b'}, \boldsymbol{x_b}, \boldsymbol{z_b}, \boldsymbol{p_b}, t_e\right) \\ &\quad \boldsymbol{p_a} = \boldsymbol{\delta_2}\left(\boldsymbol{x_b'}, \boldsymbol{x_b}, \boldsymbol{z_b}, \boldsymbol{p_b}, t_e\right) \\ &\boldsymbol{END\ WHEN} \end{aligned} \tag{2-3}$$

donde el subíndice a representa el valor de las variables después de la ejecución del evento, el subíndice b representa el valor de las variables antes de la ejecución del evento

y t_e es el valor del tiempo de activación del evento. Las variables de tiempo discreto p solo cambian sus valores en los tiempos de los eventos, por lo que son constantes entre eventos.

Las cláusulas de ecuación condicional permiten cambiar las ecuaciones activas en el modelo. El pseudocódigo de una ecuación condicional es el siguiente:

$$f1 = \boldsymbol{ZONE}\ (condition)\ f2 \quad \boldsymbol{OTHER}\ f3 \tag{2-4}$$

donde $f1$, $f2$ y $f3$ son expresiones continuas de las variables del modelo. Si la condición es verdadera, entonces $f1 = f2$, y si la condición es falsa, $f1 = f3$. La herramienta genera automáticamente una variable discreta interna α, cuyo valor es uno siempre que la condición sea verdadera y cero mientras la condición sea falsa, y traduce la cláusula de la ecuación condicional en la siguiente ecuación:

$$f1 = \alpha \cdot f2 + (1 - \alpha) \cdot f3 \tag{2-5}$$

Las asignaciones retrasadas se pueden usar como una de las acciones dentro de una cláusula **WHEN**. La sintaxis de una asignación retrasada es

$$variable = expression\ \boldsymbol{AFTER}\ delay \tag{2-6}$$

Si se activa una asignación retrasada, los valores de la expresión y el retraso se evalúan en el tiempo actual y se programa un nuevo evento de tiempo después del retraso. Posteriormente, la variable se establece en el valor evaluado de la expresión cuando se activa el nuevo evento de tiempo.

Es relevante destacar que tras la ejecución de un evento, la herramienta modifica los valores de las derivadas x' y las incógnitas algebraicas z para satisfacer las ecuaciones de restricción algebraica. Cada vez que se activa un evento, significa detener la integración para ejecutar las acciones del evento y luego reiniciarla. Este proceso puede consumir una gran cantidad de tiempo de CPU en los casos en que ocurren muchos eventos.

2.5. Proceso de Simulación en Herramientas de MSOO

La forma en que funciona una herramienta de MSOO es muy diferente de la forma en que funciona una herramienta de simulación tradicional. Los programas de simulación tradicionales funcionan de forma similar a la de un intérprete de un lenguaje de programación. Primero, el programa lee una descripción del modelo en un formato determinado y rellena las estructuras de datos que representan el modelo. Seguidamente, se llama al algoritmo de solución, que procesa las estructuras de datos del modelo y genera los resultados.

Las herramientas de MSOO funcionan de forma más similar a compiladores; compilan el modelo y lo traducen en un programa específico para el modelo en un lenguaje de programación general. La Figura 2-5 muestra el flujo de trabajo típico de una herramienta de MSOO.

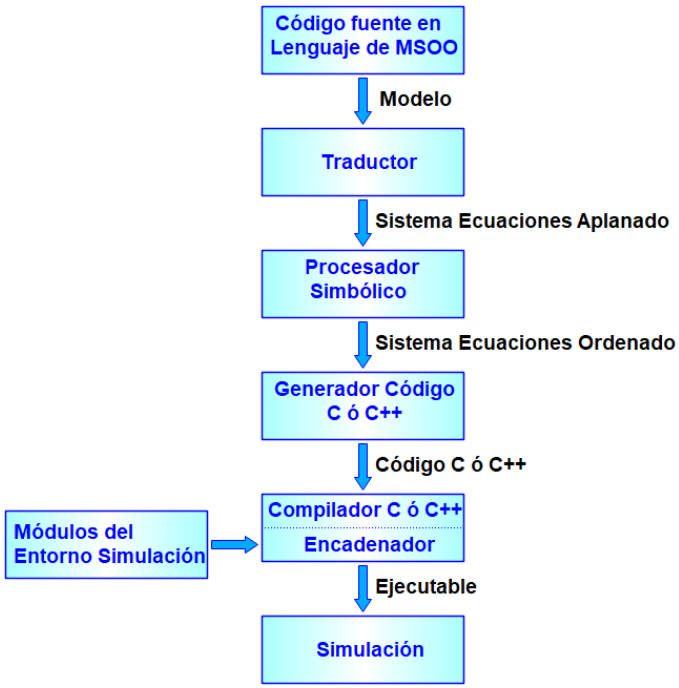


Figura 2-5: Flujo de trabajo típico de una herramienta de MSOO

Primeramente, el procesador del lenguaje de simulación lee el modelo, genera un sistema de ecuaciones aplanado, que representa el modelo juntando las ecuaciones de todos los componentes y conexiones. El calificativo aplanado indica que la organización del modelo desaparece completamente en el sistema de ecuaciones aplanado. A continuación, procesan simbólicamente el sistema de ecuaciones aplanado y lo organizan en forma adecuada a los algoritmos de solución, el resultado de este paso es lo que se denomina el sistema de ecuaciones ordenado. Seguidamente, se genera un código específico para el sistema de ecuaciones ordenado del modelo en un lenguaje de propósito general como Fortran, C y C++; por ejemplo, Dymola, genera código C y EcosimPro genera código C++. Por último, el código generado se compila y se encadena con algoritmos de solución pertenecientes al entorno de la herramienta de simulación y el modelo queda listo para ser simulado. Por defecto, las simulaciones son transitorias o dinámicas, aunque también es posible realizar simulaciones estacionarias.

Es importante indicar que cuando se declaran las ecuaciones en los componentes y en las conexiones, no se considera ninguna causalidad para las mismas. Las ecuaciones son meras relaciones entre los valores que pueden tomar las variables. Esta falta de causalidad se transmite al sistema de ecuaciones aplanado, el cual tiene unas características que dificultan su solución directa (Fritzson, 2014):

- El número de ecuaciones puede ser muy elevado, en algunos casos varios cientos de miles.
- Tiene una estructura muy dispersa. Las ecuaciones incluyen solo variables de un único componente o de componentes directamente conectados.
- En algunos casos, existen ligaduras entre las variables dinámicas, que generan dificultades numéricas a los algoritmos de solución de DAE's. Esto ocurre principalmente en el caso de sistemas mecánicos y eléctricos.
- En las ecuaciones pueden aparecer, aparte de números reales, variables discretas de tipo entero, booleano y enumerado.

2.6. Manipulación Simbólica

Desde un punto de vista puramente teórico, se podría delegar la solución de todas las ecuaciones del modelo generado con una herramienta de MSOO a un algoritmo numérico de solución de ecuaciones algebraico diferenciales (DAE´s), pero esto no es eficiente, excepto en el caso de modelos de tamaño muy reducido, ya que requeriría recursos de computación significativos, tanto en términos de memoria como de tiempo de cálculo. Por lo tanto, todas las herramientas de MSOO efectúan una manipulación simbólica del sistema con los siguientes propósitos:

- Comprobación del sistema de ecuaciones: Se verifica que el número de ecuaciones y variables sea consistente, es decir, que sean iguales. Además, se verifica que el sistema de ecuaciones no presente singularidades estructurales que dificulten su resolución adecuada
- Reducción del tamaño del sistema de ecuaciones: Se intenta calcular de manera explícita tantas variables como sea posible, y en el caso de la existencia de subconjuntos de ecuaciones algebraicas acopladas, se intenta resolverlos de forma eficiente.
- Eliminación de ligaduras entre variables de estado: Se modifica el sistema de ecuaciones derivando alguna de ellas. Esto se realiza porque la mayoria de los algoritmos de integración de DAE's no funcionan correctamente cuando hay ligaduras entre las variables de estado.

Todas las herramientas de MSOO manipulan simbólicamente el sistema de ecuaciones aplanado. Sin embargo, dicha manipulación simbólica es específica de cada herramienta y los detalles de los algoritmos o de cierta parte de los algoritmos utilizados no son públicos. Incluso las herramientas diseñadas alrededor del lenguaje Modelica no realizan un procesamiento simbólico común, pues el estándar de Modelica se limita a definir el lenguaje sin entrar en la manipulación simbólica a efectuar. Así que, aunque la idea de un lenguaje estándar es atractiva, debe tenerse en cuenta que la portabilidad de modelos entre los

diferentes entornos de simulación que utilizan Modelica no está garantizada; se suele dar la situación de que un modelo de Modelica funcione solo en la herramienta en que se ha desarrollado.

La Figura 2-6 muestra las fases típicas de la manipulación simbólica en una herramienta de MSOO. Dicha figura no es una descripción detallada para ninguna herramienta en concreto, solamente pretende ilustrar las fases típicas del análisis simbólico en este tipo de herramientas.

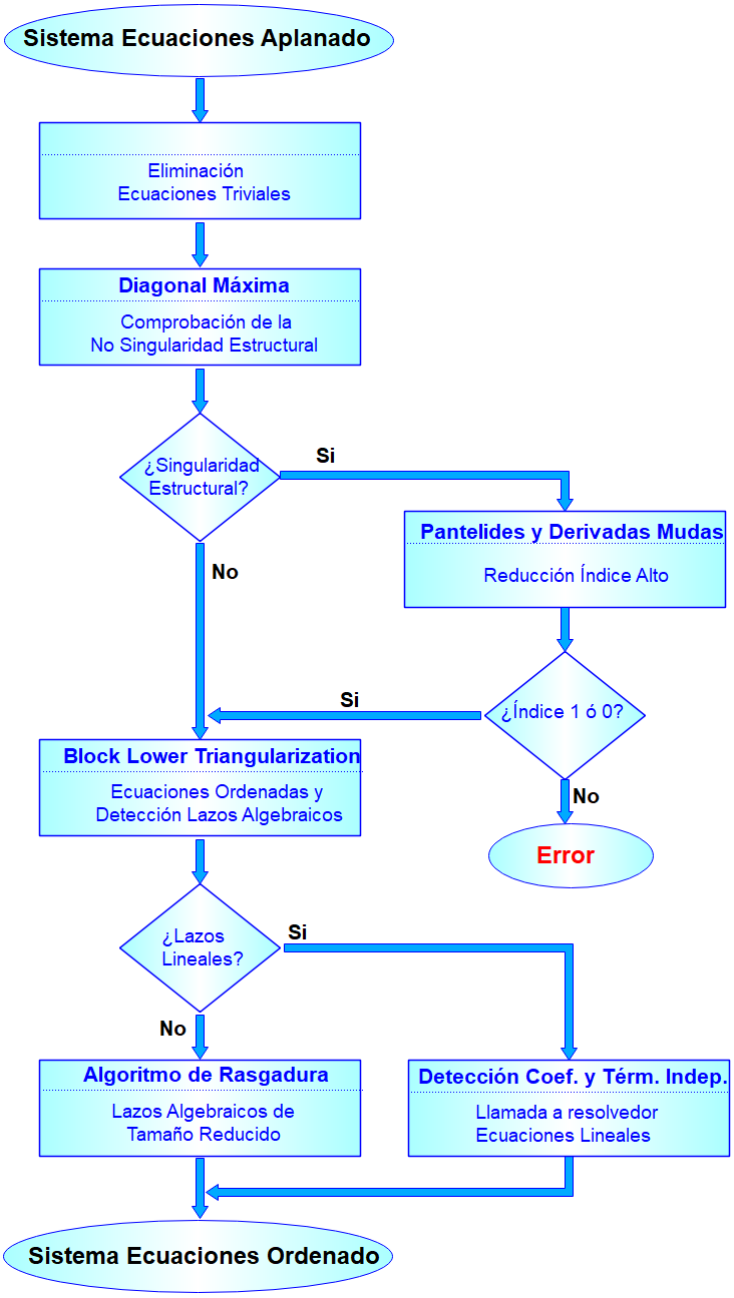


Figura 2-6: Fases de la Manipulación Simbólica de una Herramienta de MSOO

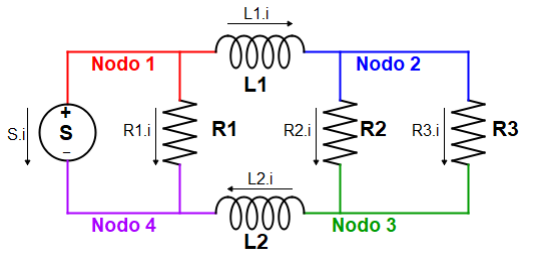
2.6.1. Eliminación de ecuaciones triviales

Primeramente, se eliminan todas las ecuaciones triviales generadas principalmente por las conexiones. En las herramientas basadas en Modelica, se detectan ecuaciones triviales de la forma:

$$y_i = \pm y_j \tag{2-7}$$

y su eliminación se efectúa seleccionando una de las variables y reemplazando todas las restantes por la variable seleccionada.

EcosimPro realiza una detección de ecuaciones triviales algo más sofisticada, pues procesa las ecuaciones lineales de coeficientes constantes. La Figura 2-7 muestra un ejemplo de circuito eléctrico y las ecuaciones de suma de corrientes igual a cero que se generan de forma automática para los cuatro nodos. Dichas ecuaciones son lineales y con coeficientes constantes (+1, 0, -1).



$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 \\ -1 & -1 & 0 & 0 & 0 & -1 \end{bmatrix} \cdot \begin{bmatrix} S.i \\ R1.i \\ L1.i \\ R2.i \\ R3.i \\ L2.i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figura 2-7: Ejemplo de Circuito Eléctrico y sus Ecuaciones de Coeficientes Constantes.

Se tienen 4 ecuaciones (una ecuación por nodo) y 6 incógnitas (las corrientes por los 6 componentes). Ahora bien, hay ecuaciones redundantes pues la última ecuación es combinación lineal de las tres primeras y no añade ninguna información nueva. EcosimPro procesa estas ecuaciones realizando una eliminación gaussiana; durante dicha eliminación se detectan las ecuaciones redundantes (filas nulas) y se descartan. Al final de la eliminación Gaussiana, se llega a la siguiente matriz:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} S.i \\ L1.i \\ R2.i \\ R3.i \\ L2.i \\ R1.i \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \Rightarrow \quad \begin{cases} S.i = -R1.i - L1.i \\ L1.i = L2.i \\ R2.i = -R3.i + L2.i \end{cases} \tag{2-8}$$

La ventaja de aplicar eliminación Gaussiana a las ecuaciones del modelo con coeficientes constantes es que se consigue detectar igualdades de variables, que de otra forma pasarían inadvertidas. En el ejemplo considerado se encuentra que las corrientes por las inductancias **`L1`** y **`L2`** son idénticas. En caso de no haberse identificado dicha igualdad, como las corrientes por **`L1`** y **`L2`** son variables de estado, existiría un problema de alto índice que el algoritmo de Pantelides sería incapaz de detectar, pues dicho algoritmo solo mira la estructura de las ecuaciones. Esto podría producir el fallo del algoritmo de integración. Este tratamiento de las ecuaciones lineales de coeficientes constantes permite una mayor simplificación del sistema y es capaz de evitar algunos problemas de alto índice. En el ejemplo, considerado una vez que se detecta que las corrientes por L1 y L2 son idénticas, y se sustituyen por una variable común, desaparece el problema de alto índice.

2.6.2. Comprobación de que el sistema ecuaciones no es estructuralmente singular

Seguidamente, se comprueba que el sistema de ecuaciones no es estructuralmente singular aplicando el algoritmo de la transversal máxima (Duff, 1981). Un sistema de ecuaciones no es estructuralmente singular, si el número de variables y el número de ecuaciones es idéntico y es posible encontrar una relación biunívoca entre variables y ecuaciones, que asocie a cada ecuación una de las variables que interviene en dicha ecuación. Cuando un sistema de ecuaciones resulta ser estructuralmente singular, significa que contiene un subconjunto de k ecuaciones que tienen menos incógnitas que k (donde k es un número entero). A continuación, se proporcionan dos ejemplos, uno de un sistema no estructuralmente singular y el otro de un sistema estructuralmente singular.

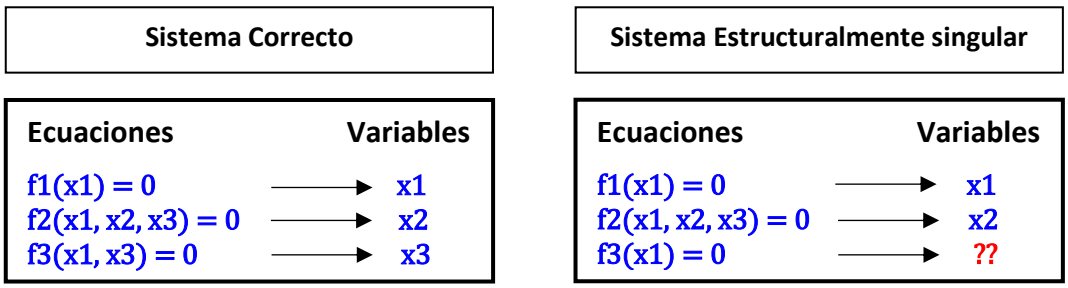


Figura 2-8: Ejemplos Simples de Aplicación del Algoritmo de la Transversal Máxima

La matriz de incidencia de un sistema de ecuaciones es una matriz booleana donde las filas representan ecuaciones y las columnas representan variables, y en la que los elementos no nulos indican que variables aparecen en cada ecuación. Por ejemplo, las matrices de incidencia de los dos sistemas anteriores son:

$\begin{matrix} & x1 & x2 & x3 \\ f1 & 1 & 0 & 0 \\ f2 & 1 & 1 & 1 \\ f3 & 1 & 0 & 1 \end{matrix}$	$\begin{matrix} & x1 & x2 & x3 \\ f1 & 1 & 0 & 0 \\ f2 & 1 & 1 & 1 \\ f3 & 1 & 0 & 0 \end{matrix}$

Desde un punto de vista matricial, el algoritmo de la transversal máxima cambia el orden de las columnas de la matriz de incidencia buscando que todos los términos de la diagonal sean no nulos. En caso de tener éxito, la diagonal define la correspondencia biunívoca entre ecuaciones y variables.

Al aplicar el algoritmo de la transversal máxima, se consideran que son conocidos el tiempo y las siguientes variables:

- Datos y constantes en el modelo, ya que mantienen su valor durante el curso de la simulación.
- Condiciones de contorno del modelo, que son todas las variables que se suponen conocidas en función del tiempo y que sirven como entradas al modelo.
- Todas las presuntas variables de estado, es decir, todas aquellas variables cuya derivada aparece en la formulación. Se asume que son inicialmente conocidas y que el algoritmo de integración calcula y actualiza dichas variables durante el transcurso de la simulación.
- En cambio, las derivadas se suponen variables desconocidas.

En caso de que el algoritmo tenga éxito se puede afirmar que el sistema de ecuaciones es o un sistema de ecuaciones algebraicas o un sistema de ecuaciones diferenciales ordinarias o un sistema de ecuaciones algebraico diferenciales de índice uno.

Es importante observar que, aunque un sistema de ecuaciones no resulte estructuralmente singular, puede ser numéricamente singular y ser imposible su simulación. La aplicación de este algoritmo sirve para detectar sistemas de ecuaciones algebraico diferenciales de índice alto o errores burdos en la formulación del modelo.

2.6.3. Reducción de problemas de índice superior

En el caso de que el sistema de ecuaciones haya resultado estructuralmente singular, ello puede deberse a que existen ligaduras entre las variables de estado, esto es lo que se conoce como un problema de índice superior.

El índice de un sistema de ecuaciones algebraico diferenciales es un número entero (mayor o igual que cero) que mide la dificultad de integrar numéricamente el sistema de ecuaciones. Se define como el número de veces que hay que diferenciar el sistema de

ecuaciones para obtener un sistema de ecuaciones ordinarias. Por tanto, un sistema de ecuaciones diferenciales ordinarias es un sistema de índice 0.

Por ejemplo, la ecuación algebraica $x(t)^2 + x(t) = u(t)$, en la que $x(t)$ es la incógnita y $u(t)$ es una función conocida del tiempo, es un DAE de índice 1, pues se transforma en un ODE con una única diferenciación.

$$x(t)^2 + x(t) = u(t) \xrightarrow{Diferenciación} (2x(t)+1)x'(t) \tag{2-9}$$

En cambio, el siguiente sistema de ecuaciones con incógnitas x_1, x_2 tiene índice 2, ya que requiere 2 diferenciaciones para transformarlo en un ODE

$$\left.\begin{aligned} x_1(t) &= u(t) \\ x_2(t) &= x_1'(t) \end{aligned}\right\} \xrightarrow{Diferenc.1ª} \left.\begin{aligned} x_1'(t) &= u'(t) \\ x_2(t) &= x_1'(t) \end{aligned}\right\} \xrightarrow{Diferenc.2º} \begin{aligned} x_1''(t) &= u''(t) \\ x_2'(t) &= x_1''(t) \end{aligned} \tag{2-10}$$

Normalmente, los métodos de solución de DAE's experimentan problemas numéricos con los sistemas de índice superior, de manera que es necesario reducir el índice del sistema antes de proceder a su solución. Con el fin de reducir el índice, las herramientas de MSOO aplican el algoritmo de Pantelides (Pantelides, 1988), que determina qué ecuaciones deben derivarse y agregarse al modelo para reducir el índice a uno. No obstante, la inclusión de ecuaciones adicionales conduce a un mayor número de ecuaciones que de incógnitas. Para abordar esta situación, se recurre al método de derivadas mudas (Mattsson & Söderlind, 1993), que permite equilibrar el sistema de índice reducido.

Es necesario notar que el algoritmo de Pantelides no siempre funciona. Se producen situaciones en las que falla. Existen ejemplos de sistemas de índice 1 para los que la aplicación del algoritmo de Pantelides puede conducir a un número grande de iteraciones y diferenciaciones (Reissig et al., 2000).

2.6.4. Ordenación de ecuaciones y solución de lazos algebraicos

El siguiente paso del procesamiento simbólico es ordenar las ecuaciones e identificar los lazos de ecuaciones algebraicos. Se aplica para ello el algoritmo de Tarjan (Tarjan, 1972), el cual permuta filas y columnas de la matriz de incidencia hasta conseguir una matriz triangular inferior por bloques (BLT, block lower triangular) de la forma:

$$\mathbf{A} = \begin{bmatrix} A_{11} & & & & & & \\ A_{21} & A_{22} & & & & & \\ A_{31} & A_{32} & A_{33} & & & & \\ \cdot & & & \cdot & & & \\ \cdot & & & & \cdot & & \\ \cdot & & & & & \cdot & \\ A_{N1} & \cdot & \cdot & \cdot & \cdot & \cdot & A_{NN} \end{bmatrix} \tag{2-11}$$

donde los bloques A_{ii} de la diagonal son cuadrados y no pueden permutarse para formar bloques de forma triangular.

A continuación, se presenta un ejemplo de aplicación del algoritmo de Tarjan al siguiente sistema de ecuaciones (2-12), que consta de 5 ecuaciones con 5 incógnitas:

$$\begin{aligned} f_1(x_2,x_3) &= 0 \\ f_2(x_1,x_5) &= 0 \\ f_3(x_2,x_4,x_5) &= 0 \\ f_4(x_3) &= 0 \\ f_5(x_3,x_4,x_5) &= 0 \end{aligned} \qquad \begin{array}{c} \\ f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \end{array} \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & x_4 & x_5 \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{array}\right] \end{array} \tag{2-12}$$

El algoritmo de Tarjan ordena las ecuaciones y las variables, y encuentra 4 bloques de tamaños 1,1,2,1. Un bloque de tamaño 1 indica que la variable asociada al bloque tiene que ser calculada justamente a partir de la ecuación correspondiente, y el cálculo es explícito en el caso de que sea posible despejar la variable de la ecuación. Bloques de tamaño mayor que 1 señalan la existencia de lazos algebraicos, demandando la resolución conjunta de las ecuaciones y variables del bloque.

$$\xrightarrow{Tarjan} \begin{array}{c} \\ f_4 \\ f_1 \\ f_3 \\ f_5 \\ f_2 \end{array} \begin{array}{c} \begin{array}{ccccc} x_3 & x_2 & x_4 & x_5 & x_1 \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{array}\right] \end{array} \longrightarrow \begin{aligned} f_4(x_3) &= 0 \\ f_1(x_2,x_3) &= 0 \\ f_3(x_2,x_4,x_5) &= 0 \\ f_5(x_3,x_4,x_5) &= 0 \\ f_2(x_1,x_5) &= 0 \end{aligned} \tag{2-13}$$

En modelos dinámicos, se suelen emplear formulaciones que conducen a un número reducido de lazos, por lo que el algoritmo ordena la mayor parte de las ecuaciones en una secuencia explícita. En el caso de modelos estacionarios de sistemas de proceso químico o de sistemas termo-hidráulicas de producción de energía, se tiene una gran sub-sistema

de ecuaciones acopladas que incluye la práctica totalidad de las ecuaciones del sistema, y la aplicación del algoritmo de Tarjan no ayuda a reducir el tamaño del problema de forma significativa.

En ocasiones, todas las ecuaciones de un bucle algebraico son lineales con respecto a las variables del bucle. En estos casos, los bucles algebraicos lineales se pueden resolver de manera eficiente durante la simulación empleando algoritmos para la solución de sistemas de ecuaciones lineales. Estos algoritmos proporcionan una solución directa del sistema de ecuaciones, lo cual agiliza el proceso de cálculo y permite obtener resultados de manera rápida y precisa. En algunos campos de simulación, el sistema de ecuaciones lineales puede ser de gran tamaño y bastante disperso, por lo que resulta beneficioso contar con herramientas de MSOO que implementen algoritmos para la resolución de sistemas de ecuaciones lineales dispersos. Recientemente, EcosimPro ha incorporado un algoritmo de este tipo.

Sin embargo, los bucles algebraicos no lineales requieren un enfoque distinto. En estos casos, las herramientas de MSOO emplean una técnica de manipulación simbólica, el rasgado del sistema de ecuaciones, para intentar reducir la complejidad del sistema de ecuaciones involucrado. El rasgado permite simplificar el sistema y obtener un sistema reducido que es más manejable. Una vez obtenido el sistema reducido, se utiliza un algoritmo especializado para la resolución de ecuaciones no lineales. Una biblioteca de software ampliamente utilizada en herramientas de MSOO para este propósito es Minpack-1 (Moré et al., 1980). Minpack-1 proporciona una colección de algoritmos numéricos diseñados específicamente para resolver sistemas de ecuaciones no lineales y realizar optimización.

2.6.5. Rasgadura (Tearing) de Lazos Algebraicos

En la simulación orientada a objeto, particularmente en las herramientas basadas en el lenguaje Modelica y en EcosimPro, se utiliza comúnmente una manipulación simbólica denominada rasgadura (“tearing”) (Elmqvist & Otter, 1994) para la solución de los sistemas de ecuaciones algebraicos no lineales. El objetivo de esta técnica es reducir el tamaño de los sistemas de ecuaciones

En un sistema de ecuaciones algebraicas disperso dado por:

$$\boldsymbol{f}(\boldsymbol{x}) = \mathbf{0} \qquad \text{donde } \boldsymbol{f}: \mathbb{R}^{\boldsymbol{n}} \rightarrow \mathbb{R}^{\boldsymbol{n}} \ . \tag{2-14}$$

La técnica de rasgadura implica permutar ecuaciones y variables con el fin de calcular explícitamente la mayoría de las variables del sistema a partir de un conjunto reducido de variables de rasgadura ("tearing"). Gracias a este proceso, las únicas variables desconocidas en el sistema son las variables de rasgadura,

En términos más específicos, la rasgadura implica encontrar matrices de permutación $\boldsymbol{P}$ y $\boldsymbol{Q}$, tales que después de la transformación:

$$\begin{bmatrix} \boldsymbol{g} \\ \boldsymbol{h} \end{bmatrix} = \boldsymbol{P}\, \boldsymbol{f} \qquad\qquad \begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} = \boldsymbol{Q}\, \boldsymbol{x}, \tag{2-15}$$

$\boldsymbol{g}(\boldsymbol{y}, \boldsymbol{z}) = \boldsymbol{0}$, puede reescribirse en la siguiente forma explícita equivalente:

$$\boldsymbol{y}_i = \tilde{g}_i\,(\boldsymbol{y}_{1:i-1}, \boldsymbol{z}) \tag{2-16}$$

utilizando para ello las transformaciones simbólicas pertinentes. En la ecuación previa, los subíndices $1{:}\,i-1$ se utilizan para indicar el conjunto $1, 2, \ldots, i-1$. De la ecuación (2-16) se infiere que la matriz Jacobiana de $\boldsymbol{P}\, \boldsymbol{f}$ tiene la forma:

$$\mathrm{J} = \begin{array}{c} \begin{matrix} y & z \end{matrix} \\ \begin{bmatrix} \mathrm{A} & \mathrm{B} \\ \mathrm{C} & \mathrm{D} \end{bmatrix} \end{array} \begin{matrix} \\ g \\ h \end{matrix}$$

donde $\boldsymbol{A}$ es triangular inferior (2-17)

Una selección particular de $\boldsymbol{P}$, $\boldsymbol{Q}$, g, h, y, z que satisfaga (2-15)y (2-16) se designa como una clasificación. Dada una clasificación, el sistema de ecuaciones puede reescribirse como:

$$\begin{aligned} \boldsymbol{g}(\boldsymbol{y}, \boldsymbol{z}) &= \boldsymbol{0} \\ \boldsymbol{h}(\boldsymbol{y}, \boldsymbol{z}) &= \boldsymbol{0}. \end{aligned} \tag{2-18}$$

El requerimiento expresado por (2-16) significa que en $\boldsymbol{g}(\boldsymbol{y}, \boldsymbol{z}) = \boldsymbol{0}$ tiene que ser posible despejar las $\boldsymbol{y}$, obteniendo $\boldsymbol{y} = \bar{\boldsymbol{g}}(\boldsymbol{z})$. Sustituyendo este resultado en $\boldsymbol{h}$, se obtiene $\boldsymbol{h}(\bar{\boldsymbol{g}}(\boldsymbol{z}), \boldsymbol{z}) = \boldsymbol{0}$, o bien:

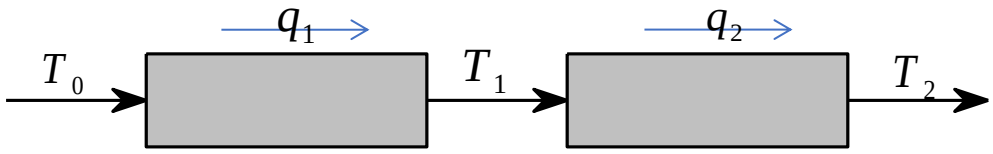
$$\boldsymbol{r}(\boldsymbol{z}) = \boldsymbol{0} \tag{2-19}$$

De esta manera, el sistema de ecuaciones original (2-14) se reduce al sistema (2-19), que suele ser de tamaño considerablemente menor que el original.

Es interesante señalar que un método prácticamente idéntico al de la rasgadura aparece a veces con el nombre de método de substitución y Newton-Raphson (Figueiredo et al., 2002; Maciel et al., 2006).

Ejemplo de Rasgadura de Modelo con Lazo Algebraico

La técnica de rasgadura se ilustra con el ejemplo 1, presentado en la Figura 2-9, que muestra un modelo con un lazo algebraico. El modelo consiste en dos conductores térmicos conectados en serie, siendo la conductividad térmica de los conductores función de su temperatura media. La serie de los dos conductores está sometida a dos fuentes de calor con temperaturas conocidas (T_0 y T_2) en los extremos.



$$k_1 = k\left(\frac{T_0 + T_1}{2}\right) \qquad q_1 = q_2 \qquad k_2 = k\left(\frac{T_1 + T_2}{2}\right)$$

$$q_1 = k_1\left(T_0 - T_1\right) \qquad\qquad q_2 = k_2\left(T_1 - T_2\right)$$

donde $k(T_{ave})$ es una función que se supone de la siguiente forma:

$$k(T_{ave}) = k_0 + k_1 \cdot T_{ave} + k_2 \cdot T_{ave}^2$$

Figura 2-9: Ejemplo 1 de Modelo con Lazo Algebraico para Ilustrar la Rasgadura

En el sistema de ecuaciones del ejemplo, hay 5 variables desconocidas (T_1, q_1, q_2, k_1, k_2). La solución por fuerza bruta implicaría plantear el siguiente sistema de 5 ecuaciones implícitas con 5 variables desconocidas:

$$F_1(T_1, q_1, q_2, k_1, k_2) = k_1 - k((T_0 + T_1)/2) = 0 \quad (2\text{-}20)$$

$$F_2(T_1, q_1, q_2, k_1, k_2) = q_1 - k_1 \cdot (T_0 - T_1) = 0 \quad (2\text{-}21)$$

$$F_3(T_1, q_1, q_2, k_1, k_2) = k_2 - k((T_1 + T_2)/2) = 0 \quad (2\text{-}22)$$

$$F_4(T_1, q_1, q_2, k_1, k_2) = q_2 - k_2 \cdot (T_1 - T_2) = 0 \quad (2\text{-}23)$$

$$F_5(T_1, q_1, q_2, k_1, k_2) = q_1 - q_2 = 0 \quad (2\text{-}24)$$

Ahora bien, al utilizar la técnica de rasgadura y seleccionar T_1 como variable de rasgadura, es posible despejar las 4 variables restantes de forma explícita utilizando para ello las 4 primeras ecuaciones y considerar como implícita solo la última ecuación, tal y como se muestra en la Figura 2-10. Además, dicha ecuación implícita es muy fácil de resolver, ya que la variación del valor del residuo con respecto a la variable de rasgadura es monótona

y presenta una linealidad considerable, tal y como se muestra en el lado derecho de la Figura 2-10.

$$k_1 = k((T_0 + T_1)/2)$$
$$q_1 = k_1 \cdot (T_0 - T_1)$$
$$k_2 = k((T_1 + T_2)/2)$$
$$q_2 = k_2 \cdot (T_1 - T_2)$$
$$H_1(T_1) \ = \ q_1 - q_2 = 0$$

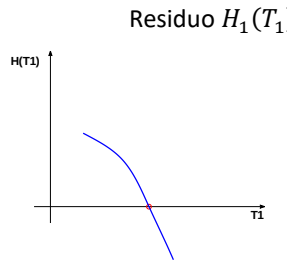


Figura 2-10: Rasgadura I del Sistema de Ecuaciones Algebraicas del Modelo Ejemplo 1

Así, el uso de la técnica de rasgadura en el modelo con lazo algebraico permite reducir considerablemente la complejidad del sistema de ecuaciones original, pasando de un sistema de dimensiones 5x5 a un sistema simplificado de dimensiones 1x1.

La rasgadura de lazos algebraicos puede considerarse como una técnica simbólica de tipo disperso (Cellier & Kofman, 2006). Esta técnica se aprovecha de la estructura dispersa de la matriz de incidencia en modelos que representan redes o circuitos, lo que permite simplificar el sistema de ecuaciones de manera eficiente.

En los modelos de simulación que representan mallas o circuitos, la matriz de incidencia presenta una dispersión significativa debido a los patrones de conectividad específicos. Por ejemplo, en los circuitos eléctricos, componentes como resistores y capacitores están conectados a través de nodos y ramas, pero cada componente solo está vinculado a unos pocos.

La búsqueda de la rasgadura óptima, que logra la máxima reducción en el tamaño del sistema de ecuaciones, es un problema NP completo. Esto implica que el tiempo de cálculo necesario para encontrar la rasgadura óptima crece de manera exponencial a medida que aumenta el tamaño del sistema de ecuaciones. Debido a esta complejidad, se utilizan algoritmos heurísticos, como el algoritmo de Cellier (Cellier & Kofman, 2006; Täuber et al., 2014) y el algoritmo de Carpanzano (Carpanzano, 2000), para encontrar rasgaduras que, aunque no sean óptimas, permitan reducir de forma significativa el tamaño del sistema de ecuaciones. Estos algoritmos se basan en técnicas aproximadas que buscan una solución aceptable en un tiempo razonable, evitando el coste computacional exponencial de encontrar la rasgadura óptima.

En general, las herramientas de modelado y simulación orientadas a objetos (MSOO), a excepción de las de código abierto, implementan algoritmos de selección automática de rasgadura que no son de acceso público o se mantienen en secreto. Adicionalmente, algunas herramientas de MSOO, como EcosimPro y J-Modelica, han implementado directivas específicas para orientar el proceso de rasgado de los lazos algebraicos. En

EcosimPro, estas directivas forman parte de su lenguaje de simulación, mientras que J-Modelica utiliza las anotaciones de Modelica.

Las anotaciones en Modelica son construcciones que permiten agregar información específica de la herramienta, fundamentalmente de tipo gráfico o documental. En principio, las anotaciones no deberían afectar el comportamiento del modelo, pero no es el caso de las anotaciones de guiado de la rasgadura. Estas directivas y anotaciones se utilizan para influir en el algoritmo de rasgadura y mejorar los resultados, o incluso pueden llegar a proporcionar una definición completa de la rasgadura que evite el uso de algoritmos heurísticos.

2.6.6. Problemas de la Rasgadura de Ecuaciones (Tearing)

La aplicación automática de la técnica de rasgadura de ecuaciones en las herramientas de MSOO presenta varios problemas, que incluyen:

- **Falta de robustez numérica**: la rasgadura, al ser una técnica simbólica que aprovecha la estructura dispersa del sistema de ecuaciones, presenta el inconveniente típico de las técnicas de tipo simbólico, que es la falta de robustez numérica. Esto significa que la técnica puede ser sensible a pequeñas variaciones en los valores numéricos de las variables y puede conducir a dificultades de convergencia en el sistema de ecuaciones.
- **No es adecuada para la solución de sistemas grandes de ecuaciones:** el procesamiento simbólico se vuelve muy lento cuando aumenta el tamaño del sistema. Por ejemplo, Baharev et al., 2017 han reportado tiempos de procesamiento simbólico del orden de minutos para sistemas de solo 300 ecuaciones. Esto limita su aplicabilidad en simulaciones a gran escala que involucran miles de ecuaciones.
- **Variabilidad de los conjuntos de variables y ecuaciones de rasgadura seleccionadas:** pequeños cambios en un modelo, como agregar o eliminar componentes, pueden ocasionar cambios significativos en las variables y ecuaciones elegidas para la rasgadura. Esto dificulta la inicialización adecuada de las variables, ya que los valores iniciales ajustados previamente se pierden con los cambios en las variables seleccionadas. Además, el algoritmo de rasgadura puede elegir variables de iteración para las cuales el usuario tiene menos intuición sobre los valores iniciales adecuados o límites.
- **Dificulta el uso de métodos numéricos dispersos**: múltiples herramientas de simulación comerciales utilizan algoritmos numéricos para la solución de ecuaciones dispersas. Estos algoritmos se aplican después de llevar a cabo una rasgadura no óptima, pero siguiendo reglas preestablecidas. Por ejemplo, el método de los nodos y el método de las mallas para el análisis de circuitos eléctricos pueden considerarse como rasgaduras no óptimas que siguen reglas específicas. Sin embargo, las herramientas de MSOO intentan realizar una rasgadura óptima, lo que, si tiene éxito,

resulta en un sistema de ecuaciones denso. Este enfoque de las herramientas de MSOO implica la pérdida de información sobre la dispersión en las ecuaciones algebraicas y limita el uso eficiente de algoritmos diseñados para ecuaciones dispersas.

- **Limitación en la portabilidad de modelos**: incluso entre herramientas de MSOO basadas en el lenguaje estándar Modelica, la portabilidad de modelos con lazos algebraicos suele ser imposible debido a que cada herramienta tiene sus propios algoritmos de rasgadura, así como mecanismos diferentes para guiar el algoritmo de rasgadura y controlar la inicialización de las variables desconocidas.

A continuación, se presenta una discusión sobre el problema de la falta de robustez numérica de la rasgadura.

Falta de Robustez Numérica de la Rasgadura

La rasgadura no es una técnica numéricamente robusta, ya que se aplica antes de conocer el valor numérico de las variables y de los residuos de las ecuaciones. Durante el proceso de selección de variables y ecuaciones para la rasgadura, existe la posibilidad de elegir combinaciones que conduzcan a un sistema de ecuaciones con dificultades de convergencia.

El sistema de ecuaciones del ejemplo anterior sirve también para ilustrar este tipo de problemas. Una segunda opción de rasgadura, que se muestra en la Figura 2-11 y que también utiliza T_1 como variable de rasgadura, es igual de válida simbólicamente que la primera, debido a que también reduce el tamaño del sistema de 5x5 a 1x1.

$$k_1 = k((T_0 + T_1)/2)$$
$$q_1 = k_1 \cdot (T_0 - T_1)$$
$$q_2 = q_1$$
$$k_2 = k((T_1 + T_2)/2)$$
$$H_1(T_1) = k_2 - \frac{q_2}{(T_1 - T_2)} = 0$$

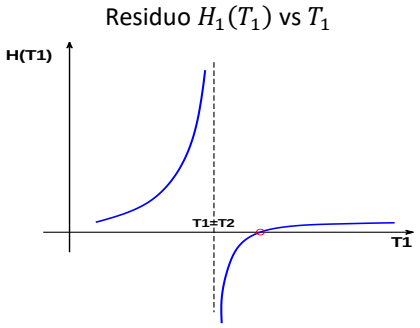


Figura 2-11: Rasgadura II del Sistema de Ecuaciones Algebraicas del Modelo Ejemplo 1

Sin embargo, esta segunda opción de rasgadura resulta significativamente inferior a la primera desde un punto de vista numérico. En el lado derecho de la Figura 2-11, se muestra el valor del residuo en función de la variable rasgadura para esta segunda rasgadura. Es evidente que el residuo no varía de forma monótona con respecto a la variable de rasgadura, excepto para valores de $T_1 > T_2$. La consecuencia es que el sistema reducido solo converge, al menos en el caso de EcosimPro, cuando el valor inicial de T_1 es mayor

que T_2. Esta falta de monotonía en el comportamiento del residuo dificulta la convergencia y afecta negativamente a la solución del sistema de ecuaciones.

En su propuesta de un algoritmo de rasgadura, Cellier & Kofman (2006) resaltan la importancia de evitar divisiones al despejar variables para prevenir el riesgo de divisiones por cero durante el cálculo de la solución. Si se hubiera seguido esta heurística, la segunda opción de rasgadura habría sido descartada. No obstante, es relevante destacar que el problema no se origina debido a divisiones por cero durante el cálculo de la solución, sino que es debido al comportamiento fuertemente no monótono de la función de residuos.

Baharev et al. (2017) analizaron diversos modos de fallo de los algoritmos de rasgadura, identificando los cuatro mecanismos de fallo siguientes:

- Modo de Fallo No 1: el residuo es extremadamente sensible a la variable de rasgadura
- Modo de Fallo No. 2: el residuo es muy poco sensible a los cambios en la variable de rasgadura
- Modo de Fallo No. 3: la técnica de rasgadura es incapaz de reducir el tamaño del sistema debido a la presencia de variables que no pueden ser despejadas.
- Modo de Fallo No. 4: ocurren fallos al despejar una variable debido a diversos motivos, como operaciones indefinidas, excepciones de punto flotante o la existencia de múltiples soluciones. Para evitar este tipo de fallo, las herramientas de MSOO solo permiten despejar variables que aparecen de forma lineal en las ecuaciones y con coeficientes que no pueden ser nulos. Sin embargo, esta restricción limita la reducción potencial del tamaño del sistema que se puede lograr mediante la rasgadura.

Los autores también propusieron ejemplos en Modelica para cada uno de los tres primeros modos de fallo y analizaron el comportamiento de dichos ejemplos utilizando dos de las herramientas que implementan Modelica: Dymola y OpenModelica. En el Apéndice A de este documento, se presenta la implementación y los resultados en EcosimPro de esos mismos ejemplos.

El problema de la segunda rasgadura en el ejemplo 1 podría considerarse como un caso del modo de fallo no. 4. Sin embargo, la causa subyacente no es la división por cero, sino la falta de una variación monótona de la función de residuos. Si el valor estimado inicialmente para la variable desconocida queda fuera de la zona de variación monótona, los algoritmos de solución de ecuaciones algebraicas experimentan dificultades para lograr la convergencia.

La forma en que se expresan las ecuaciones no lineales puede tener un impacto considerable en la convergencia. Considérese la siguiente ecuación:

$$c\,(a - x) - d\,(x - b) = 0\,, \tag{2-25}$$

donde x es la incógnita y los coeficientes a, b, c y d son positivos, con $a > b$. Suponiendo que x no puede ser despejada, dicha ecuación es fácilmente resoluble con las subrutinas de la librería Minpack-1 (Moré et al., 1980). Minpack-1 es una biblioteca de software ampliamente utilizada en herramientas de MSOO, ya que proporciona una colección de algoritmos numéricos para la resolución de ecuaciones no lineales y optimización.

En cambio, si expresamos la ecuación anterior en la siguiente forma alternativa:

$$c\,\frac{(a-x)}{(x-b)} - d\; = 0 \tag{2-26}$$

Minpack-1 y la mayor parte de los algoritmos de solución de sistemas de ecuaciones algebraicas fallarán cuando el valor inicial de x sea menor que b, pero convergerán fácilmente cuando x sea mayor que b.

Es importante tener en cuenta estas consideraciones al expresar ecuaciones no lineales en lenguajes de MSOO. Incluso en el caso de una única ecuación, la forma en que se exprese puede tener un impacto significativo sobre la convergencia de los algoritmos utilizados.

En modelos termo-fluido dinámicos con lazos algebraicos, se presenta un problema adicional: la existencia de ecuaciones con dominios de validez limitados. Por ejemplo, hay ecuaciones que calculan propiedades termo-fluido dinámicas, coeficientes de transferencia de calor o de masa, y velocidades de reacción. Estas ecuaciones tienen rangos de validez limitados, y fuera de esos rangos pueden generar valores incorrectos o producir excepciones, como operaciones inválidas, divisiones por cero, desbordamientos (*overflow*) o subdesbordamientos (*underflow*), lo que puede llevar a la terminación inesperada del programa.

La consecuencia práctica es que, aunque se afirma que los lenguajes de MSOO no consideran causalidad en las ecuaciones y que las ecuaciones pueden expresarse de cualquier forma equivalente, cuando se baja a la arena de las ecuaciones algebraicas, cobra importancia la forma en que se expresan las ecuaciones.

Un posible remedio a estos problemas de falta de robustez sería estimar la sensibilidad de las ecuaciones con respecto a las variables antes de realizar la rasgadura, con el objetivo de seleccionar las variables y ecuaciones más apropiadas desde un punto de vista numérico y así mejorar la robustez numérica. Sin embargo, es importante señalar que esta elección puede ser adecuada solo en os momentos iniciales de la integración, pudiendo volverse inadecuada a medida que el modelo evoluciona en el tiempo.".

Es interesante notar que la comunidad de desarrolladores de herramientas de simulación orientada a objetos ha invertido mucho esfuerzo en encontrar el algoritmo definitivo para la rasgadura de ecuaciones. Algunos ejemplos de desarrollos en esta dirección incluyen el

algoritmo descrito por Cellier y Kofman en 2006 (Capítulo 7) y su versión modificada por Täuber et al. en 2014. Posteriormente, el mismo autor reconoció los problemas numéricos asociados con la técnica de rasgadura y propuso la rasgadura dinámica en 2016.

Está claro que no existe una solución perfecta para lograr una rasgadura óptima, ya que dicha solución no puede existir debido a las limitaciones mencionadas de los algoritmos simbólicos. Aunque las limitaciones de la rasgadura se conocían desde los primeros artículos sobre el tema. Más recientemente, Baharev et al. en 2017 reconocieron las limitaciones inherentes de los algoritmos de rasgadura y propusieron el uso de una técnica llamada "*staircase sampling*".

2.6.7. Directivas para la Rotura de Lazos Algebraicos

En ciertos campos de simulación, la selección adecuada de variables y ecuaciones de rasgadura es bien conocida por los expertos en el campo. Un ejemplo de un campo en el que se da esta situación es la simulación del ciclo de turbinas de gas. Esto se identificó claramente durante el desarrollo de Proosis (Alexiou & Tsalavoutas, 2011), una herramienta para la simulación de turbinas de gas desarrollada a partir de EcosimPro, y posteriormente se ha vuelto a identificar durante el desarrollo de una librería de Modelica también para la simulación de turbinas de gas (Coïc et al., 2020).

La forma de incorporar en las librerías el conocimiento de expertos en el campo sobre la selección de las ecuaciones y variables de rasgadura, ha sido mediante directivas en el lenguaje en el caso de EcosimPro y mediante anotaciones en el lenguaje en el caso de herramientas basadas en Modelica (Oscar Kari, 2022).

Normalmente, las herramientas de MSOO que implementan directivas para la rotura de lazos algebraicos proporcionan las 3 directivas siguientes:

1. Indicar que una variable es de rasgadura.
2. Indicar que una ecuación es de rasgadura.
3. Indicar que una ecuación es de rasgadura y que una de las variables involucradas en esa ecuación es de rasgadura.

En el caso de la simulación de turbinas de gas, es posible lograr una definición completa de la rasgadura utilizando estas directivas en el código de los componentes. Esto se debe a que las turbinas de gas tienen arquitecturas muy definidas, en las cuales el flujo de gas progresa de un componente al siguiente y las recirculaciones son mínimas. Además, hay un número limitado de arquitecturas de las turbinas de gas, y existe un conocimiento bien establecido sobre cómo realizar la rasgadura de las ecuaciones que modelan cada una de estas arquitecturas. Este conocimiento se puede sistematizar utilizando las directivas mencionadas anteriormente.

Por ejemplo, los flujos de entrada de aire al motor son variables ideales de rasgadura, lo cual se especifica mediante directivas del tipo 1 en los componentes que representan las entradas de aire. Por otro lado, las ecuaciones de rasgadura asociadas a estos flujos de entrada de aire son las ecuaciones que calculan el flujo de salida en las toberas del motor. Para indicar que estas ecuaciones son de rasgadura se utilizan directivas del tipo 2. Es importante destacar que la variable y la ecuación de rasgadura correspondiente pueden pertenecer a componentes diferentes del modelo, como se ilustra en el ejemplo mencionado.

A continuación, se describe como es la rasgadura de lazos algebraicos en unas pocas de las herramientas de MSOO, a saber: una antigua versión de Dymola, J-Modelica, OpenModelica y EcosimPro.

Rotura de Lazos Algebraicos en Dymola (versión 1994)

Es interesante destacar que en las etapas iniciales del desarrollo de las herramientas de MSOO, Elmqvist & Otter, (1994) introdujeron el operador $\boldsymbol{residue}()$ en una versión de Dymola anterior a la creación o definición de Modelica. Este nuevo operador era en cierto modo similar al operador $\boldsymbol{der}()$ utilizado para indicar derivada en Dymola, y su sintaxis de su uso era la siguiente:

$$\boldsymbol{residue}(x) = expression$$

El significado de este operador es indicar que el valor de $expression$ debe ser nulo en todo momento, y x es la variable de rasgadura que se debe variar para lograr anular el residuo.

Posteriormente, el operador $\boldsymbol{residue}()$ desapareció en la definición de Modelica. Surge la pregunta sobre el porqué de esa desaparición. El propio Elmqvist, (2014) proporcionó la siguiente explicación sobre este cambio:

> *"Faltaba una pieza en la tecnología para abordar el hecho de que los bloques correspondientes a sistemas de ecuaciones simultáneas a menudo se volvían grandes pero dispersos. El rasgado es una técnica estática utilizada para solucionar este problema. Nuestro primer intento fue ayudar al traductor introduciendo indicaciones en forma de un operador de residuo. Poco después, Martin Otter y yo desarrollamos un algoritmo de rasgado automático que se implementó en Dymola. Por ejemplo, en un modelo multicuerpo con estructura jerárquica, Dymola sería capaz de encontrar un sistema lineal de ecuaciones de gran tamaño que involucra aceleraciones, fuerzas y momentos, y descubriría que el problema se podría reducir a la inversión de una matriz lineal del tamaño igual al número de grados de libertad."*

En otras palabras, al encontrar un algoritmo heurístico para la rasgadura que funcionaba bien para modelos multicuerpo, desestimaron la solución inicial de introducir indicaciones para ayudar al traductor. El problema de los algoritmos de rasgadura heurísticos es que solo se basan en la estructura del sistema de ecuaciones y en la implementación del

código, pero no en la física del problema. Estos algoritmos generalmente funcionan bien para ciertos tipos de sistemas de ecuaciones algebraicas, generalmente de tamaño medio o pequeño, pero no funcionan para cualquier campo.

Rotura de Lazos Algebraicos en Modelica - JModelica

En J-Modelica, Oscar Kari (2022) ha implementado directivas para la rotura de lazos algebraicos, de los tipos 1 a 3, utilizando las anotaciones del lenguaje Modelica.

En Modelica, los componentes incluyen ecuaciones transitorias, utilizadas para el cálculo dinámico, y ecuaciones iniciales, utilizadas para el cálculo del punto inicial del sistema. Las directivas implementadas en J-Modelica permiten manejar los lazos algebraicos presentes en ambos conjuntos de ecuaciones, ya que las anotaciones de rasgadura tienen un campo para indicar si se aplican a las ecuaciones iniciales, a las transitorias o a ambas.

Coïc et al. (2020) han desarrollado una librería para la simulación estacionaria de turbinas de gas. En esta librería, se ha utilizado de manera efectiva las directivas implementadas en J-Modelica para guiar el rasgado de lazos algebraicos. Al hacer uso de estas directivas, los desarrolladores han logrado gestionar la rasgadura de manera más efectiva, incorporando conocimiento físico al proceso de simulación.

Rotura de Lazos Algebraicos en OpenModelica

En el caso de OpenModelica, los algoritmos de rasgadura que ha implementado son públicos. En principio ha implementado las siguientes opciones de rasgadura:

- Ninguna rasgadura o rasgadura limitada a las variables discretas
- Método de rasgadura de Cellier
- Método de rasgadura desarrollado por TU Dresden: Frenkel, Schubert.

En el caso sin rasgadura, ofrece la posibilidad de resolver los lazos algebraicos utilizando Kinsol, que es un software para la resolución de sistemas algebraicos no-lineales dispersos.

En cuanto a las directivas para guiar el rasgado, OpenModelica solo soporta las directivas de tipo 1 para indicar las variables de rasgado.

El concepto de no realizar ninguna rasgadura, si bien es interesante al permitir el uso de algoritmos de solución de sistemas de ecuaciones dispersos, tiene el inconveniente de que puede dar lugar a sistemas de ecuaciones muy grandes. El algoritmo de Tarjan incluye en los lazos muchas ecuaciones y variables que en realidad son auxiliares. Este problema no se presenta en los ejemplos eléctricos comúnmente utilizados para ilustrar el algoritmo de Tarjan en las herramientas de MSOO, ya que en dichos ejemplos se emplean pocas o ninguna variable auxiliar. Sin embargo, en campos de simulación que involucran flujo de fluidos, transmisión de calor y reacciones químicas, se presentan un conjunto reducido de

ecuaciones fundamentales, las cuales expresan conservación de masa, momento y energía, y múltiples ecuaciones auxiliares, que se utilizan para calcular variables termodinámicas y coeficientes en las ecuaciones fundamentales. Por lo general, la causalidad de las ecuaciones auxiliares está bien definida y es posible calcularlas de forma explícita a partir de unas variables fundamentales. Para ilustrar el problema, se consideran las siguientes ecuaciones para el cálculo de caída de presión en una tubería:

$$p_{in} - p_{out} - \left({}^{1}/_{2}\right) f(l/d)\ m\,|m|/(\rho\, A^2)\ = 0 \tag{2-27}$$

$$\rho = \rho(p_{in}, T_{in}) \tag{2-28}$$

$$\mu = \mu(p_{in}, T_{in}) \tag{2-29}$$

$$Re\ = m \cdot D/(\mu \cdot A) \tag{2-30}$$

$$f\ = f(Re, \varepsilon/D) \tag{2-31}$$

En este ejemplo, la ecuación (2-27) expresa la conservación del momento, donde p_{in} es la presión de entrada, p_{out} es la presión de salida, f es el factor de fricción, l es la longitud del tubo, d es el diámetro hidráulico, ρ es la densidad del fluido y A es el área de flujo. Todas las ecuaciones restantes son auxiliares. Las ecuaciones (2-28) y (2-29) calculan las propiedades del fluido a la entrada de la tubería en función de la presión y la temperatura, la ecuación (2-30) es la definición del número de Reynolds, y la ecuación (2-31) es una correlación que proporciona el factor de fricción.

Cuando un experto plantea el sistema de ecuaciones, solo considera como implícita la ecuación de conservación del momento. Las ecuaciones auxiliares, a pesar de formar parte de los lazos, se pueden calcular de forma explícita. En realidad, el experto aplica una rasgadura, aunque no se preocupa por optimizarla, ya que puede recurrir a un resolvedor numérico disperso si es necesario. En contraste, en una herramienta de MSOO, todas las ecuaciones, tanto fundamentales como auxiliares, se consideran igualmente y forman parte del lazo algebraico. Es importante destacar que, en este ejemplo, mientras el experto plantea una única ecuación implícita por tubería, la herramienta de MSOO plantea 5 ecuaciones en caso de no realizar ninguna rasgadura.

Rotura de Lazos Algebraicos en EcosimPro

EcosimPro utiliza un algoritmo heurístico de rasgadura que presenta los problemas mencionados anteriormente asociados a este tipo de enfoque. Sin embargo, para guiar el proceso de rasgadura, el lenguaje de modelado de EcosimPro proporciona directivas de los tipos 1 a 3. Estas directivas son de vital importancia en la simulación de turbinas de gas utilizando EcosimPro.

Conclusiones sobre la Rasgadura y Necesidad de una Nueva Directiva

La comunidad de desarrolladores de herramientas de simulación orientada a objetos ha invertido un esfuerzo considerable en la búsqueda del algoritmo ideal para la realización de la rasgadura de ecuaciones. Algunos desarrollos destacados en esta área incluyen el algoritmo descrito por Cellier & Kofman (2006) y su versión modificada propuesta por Täuber et al. (2014). Posteriormente, este último autor propuso la idea de una rasgadura dinámica como una respuesta a los problemas numéricos asociados con la rasgadura (Täuber et al., 2016).

Está claro que no existe una bala mágica para lograr una rasgadura óptima, y lo máximo que se puede lograr son algoritmos heurísticos que funcionen adecuadamente para ciertos tipos de modelos.

Una posible solución más sólida es el uso de directivas de rasgadura que guíen completamente el proceso, evitando la necesidad de hacer selecciones propias por parte de la herramienta. Esto implica eliminar las ecuaciones y variables de rasgadura especificadas mediante estas directivas, de modo que las ecuaciones restantes proporcionen una secuencia de cálculo explícita para todas las variables restantes. Este enfoque es más robusto, ya que permite la incorporación de conocimiento físico y es factible, al menos, en el diseño de librerías para cálculos estacionarios. Sin embargo, en el caso de librerías transitorias, los problemas de índice superior pueden dificultar la viabilidad de esta aproximación. Es importante destacar que una rasgadura total y sistemática del sistema de ecuaciones debe combinarse con resolvedores numéricos para ecuaciones dispersas cuando el tamaño del sistema sea grande.

Surge la pregunta de si los tipos de directivas mencionados anteriormente (tipos 1 a 3) son suficientes para guiar completamente la rasgadura. Generalmente, las herramientas de simulación de circuitos eléctricos y sistemas hidráulicos utilizan la aproximación nodal modificada para construir el modelo matemático del sistema. Con esta aproximación, las variables de las conexiones que representan voltaje, presión u otro tipo de esfuerzo se seleccionan como variables de rasgadura, mientras que las ecuaciones de suma nula en las conexiones de las variables que representan corriente eléctrica, flujo de fluido u otro tipo de flujo se seleccionan como ecuaciones de rasgadura. Aunque este enfoque no es óptimo en términos de minimizar el número de ecuaciones, es numéricamente robusto, ya que generalmente se observa una variación monótona en los residuos en función de las variables de rasgadura. Por ejemplo, un aumento en el voltaje en un nodo generalmente reduce las corrientes que llegan a ese nodo, mientras que una disminución del voltaje suele incrementar las corrientes, lo que resulta en una variación monótona en los residuos de las corrientes con respecto al voltaje.

Las directivas de guiado de la rasgadura deberían permitir el forzado de una rasgadura similar a la de la aproximación nodal modificada. Sin embargo, no es posible indicar

explícitamente que las ecuaciones de suma nula de las variables de tipo flujo en las conexiones deben considerarse como ecuaciones de rasgadura, ya que no aparecen de forma explícita en el modelo. Por lo tanto, sería recomendable proporcionar un cuarto tipo de directiva en la declaración de las variables de tipo flujo en las conexiones para indicar que las ecuaciones de suma nula generadas automáticamente deben ser tratadas como ecuaciones de rasgadura. Esta adición permitiría una mayor flexibilidad y control en el proceso de rasgadura.

Además, es crucial que estas directivas formen parte de los lenguajes de MSOO para fomentar la portabilidad de modelos entre diferentes herramientas.

2.7. Revisión Crítica de las Herramientas de MSOO

Las herramientas de MSOO han supuesto un avance significativo y una mejora con respecto a las herramientas basadas en el estándar CSSL, como ACSL, y las herramientas basadas en diagramas de bloques, como Simulink. Las herramientas de MSOO permiten acortar y abaratar el desarrollo de aplicaciones para la simulación de sistemas físicos, tales como circuitos eléctricos, neumáticos, hidráulicos, procesos químicos, entre otros.

Sin embargo, como ocurre con cualquier tecnología, las herramientas de MSOO también presentan problemas inherentes. Estos problemas están estrechamente relacionados con los dos niveles de uso de las herramientas MSOO: un nivel 1 que implica el desarrollo de librerías de componentes para campos específicos, y un nivel 2, que es el nivel del usuario final que crea modelos utilizando librerías predefinidas y realiza simulaciones.

A continuación, se resumen en una lista los problemas asociados con el MSOO:

- Encapsulamiento de la información limitado a la fase de modelado.
- Diseño enfocado en el creador de librerías en lugar del usuario final.
- Dificultad en el procesamiento simbólico.
- Problemas de convergencia de las ecuaciones algebraicas.
- Problemas de integración transitoria.
- Técnicas de solución inadecuadas para modelos de gran tamaño.
- Limitación en los métodos de solución a resolvedores de DAE’s (Ecuaciones Diferenciales Algebraicas) y ODE’s (Ecuaciones Diferenciales Ordinarias).

2.7.1. Encapsulamiento de la información limitado a la fase de modelado

La encapsulación y el ocultamiento de información proporcionado por la orientación a objeto aplican sólo a la fase de modelado, pero no a la de simulación. Durante la fase de simulación, es posible que se requiera que el usuario de nivel 2 tenga un conocimiento

profundo de los detalles de implementación de la librería y de la herramienta de MSOO. Ninguna de las herramientas de MSOO consideradas en esta tesis ofrece un lenguaje para definir experimentos que incorpore los principios de encapsulación y ocultamiento de información propios de la programación orientada a objeto.

2.7.2. Diseño enfocado en el creador de librerías en lugar del usuario final

Los lenguajes MSOO suelen resultar muy atractivos para los desarrolladores de librerías, los usuarios de nivel 1. Sin embargo, las aplicaciones resultantes suelen ser complicadas de utilizar para los usuarios finales de nivel 2.

El primer nivel de uso, que implica la creación de librerías para un campo específico, requiere un conocimiento profundo de la herramienta en varias áreas clave. Estas áreas incluyen la comprensión de la sintaxis y la semántica del lenguaje de modelado, el conocimiento del procesamiento simbólico utilizado por la herramienta para convertir sistemas de ecuaciones planas en sistemas resolubles, así como los métodos de solución de ecuaciones diferenciales ordinarias y ecuaciones algebraico-diferenciales implementados en la herramienta.

Por otro lado, el segundo nivel de uso implica la utilización de librerías predefinidas para construir modelos y realizar simulaciones. En este nivel, se espera que los usuarios tengan conocimientos específicos del campo de simulación, así como un conocimiento superficial de la herramienta de MSOO. Estos conocimientos superficiales incluyen la capacidad de crear modelos gráficamente y ejecutar simulaciones de manera interactiva.

La Figura 2-12 resume los conocimientos que se exigen a los usuarios de cada uno de estos niveles.

Nivel 1: Creación de Librerías para Campos Específicos

Requisitos del Usuario:

- Conocimiento profundo del Campo Específico de simulación
- Conocimiento profundo de la herramienta de MSOO:
  - Lenguaje de modelado
  - Procesado simbólico
  - Métodos de solución de ecuaciones diferenciales ordinarias y algebraico-diferenciales

Nivel 2: Creación de Modelos y Ejecución de Simulaciones

Requisitos del Usuario:

- Conocimiento del Campo Específico de simulación
- Conocimiento general de la librería
- Conocimiento de uso de la aplicación de MSOO:
  - Creación gráfica de modelos
  - Realización de simulaciones (experimentos) sobre los modelos
  - Visualización de los resultados y obtención de informes de resultados

Figura 2-12: Niveles de uso de una herramienta de MSOO – Requisitos sobre los usuarios

El principal problema con las aplicaciones de MSOO radica en la falta de una clara separación entre los dos niveles de uso. Es común que se requieran conocimientos del nivel 1 al usuario de nivel 2, lo que lleva a la percepción de que las herramientas de MSOO son excesivamente complicadas y solo pueden ser utilizadas por especialistas o gurús en la materia. Esta falta de separación adecuada entre el nivel de desarrollador y el nivel de usuario limita la adopción generalizada de estas herramientas, ya que muchos ingenieros consideran que su uso está fuera de su alcance debido a la complejidad asociada.

Por lo general, el usuario de nivel 2 no tiene dificultad en construir y rellenar los datos de un modelo. Los problemas cuya solución le exigen conocimientos profundos de la librería utilizada y de la herramienta de MSOO aparecen durante el procesamiento simbólico, la inicialización del modelo y la integración del mismo.

Las características de orientación a objeto suelen ser más atrayentes para el usuario de nivel 1 que para el usuario de nivel 2. La orientación a objeto permite una programación más elegante, pero no es una característica esencial para una herramienta de simulación genérica. En realidad, la característica fundamental de una herramienta de modelado físico es brindar al usuario la capacidad de definir nuevos componentes y nuevos tipos de conexiones para la construcción de modelos modulares. La Figura 2-13 muestra la definición en el lenguaje de EcosimPro de los componentes básicos de una librería eléctrica, tanto con el uso de herencia como sin él. Se puede observar que la herencia puede reducir un poco la cantidad de código necesario para definir los componentes, pero no es esencial para su creación.

Un ejemplo destacado de herramienta de modelado físico que no utiliza un lenguaje de MSOO es Amesim. En esta herramienta, el desarrollador de librerías de componentes reutilizables debe anticipar las posibles causalidades de un componente que pueden requerirse cuando éste se conecta a otros componentes en un modelo, y crear diferentes versiones del submodelo matemático del componente para cada una de las posibles causalidades de uso identificadas. Durante la construcción del modelo, el usuario agrega componentes, los cuales pueden tener varios sub-modelos matemáticos asociados con diferentes causalidades. Es al conectar el símbolo del componente cuando se establece la causalidad y el sistema selecciona la versión del submodelo matemático con causalidad adecuada según los componentes conectados.

En su versión original, AMESim no proporcionaba ningún lenguaje de simulación. Para crear un componente, era necesario escribir una subrutina o función en Fortran o C siguiendo ciertas convenciones descritas en el manual de usuario. A pesar de que AMESim puede parecer muy primitivo en comparación con las herramientas de MSOO, ha logrado un gran éxito comercial. Esto puede deberse en parte a su interfaz de usuario intuitiva y

fácil de usar, así como a su conjunto de librerías bien adaptadas a las necesidades de las industrias automotriz y aeroespacial. En resumen, AMESim se enfocó en el usuario de nivel 2 y esto contribuyó a su éxito comercial notable.

```
ABSTRACT COMPONENT OnePort
    PORTS
        IN elec e_p
        IN elec e_n
    DECLS
        REAL i
        REAL v
    CONTINUOUS
        v = e_p.v - e_n.v
        0 = e_p.i + e_n.i
        i = e_p.i
END COMPONENT

COMPONENT Ground
    PORTS
        IN elec e_p
    CONTINUOUS
        e_p.v = 0
END COMPONENT

COMPONENT Resistor IS_A OnePort
    DATA
        REAL R=1
    CONTINUOUS
        R*i = v
END COMPONENT

COMPONENT Capacitor IS_A OnePort
    DATA
        REAL C=1e-6
    CONTINUOUS
        i = C*v'
END COMPONENT

COMPONENT Inductor IS_A OnePort
    DATA
        REAL L=1 UNITS
    CONTINUOUS
        L*i' = v

END COMPONENT
```

Versión con Herencia Librería Eléctrica Básica

```
COMPONENT Ground
    "Ground of an electrical circuit"
    PORTS
        IN elec e_p
    CONTINUOUS
        e_p.v = 0
END COMPONENT

COMPONENT Resistor
    PORTS
        IN elec e p
        IN elec e_n
    DATA
        REAL R=1 UNITS
        REAL i
        REAL v
    CONTINUOUS
        v = e_p.v - e_n.v
        0 = e_p.i + e_n.i
        i = e_p.i
        R*i = v
END COMPONENT

COMPONENT Capacitor
    PORTS
        IN elec e_p
        IN elec e_n
    DATA
        REAL C=1e-6
    DECLS
        REAL i
        REAL v
    CONTINUOUS
        v = e_p.v - e_n.v
        0 = e_p.i + e_n.i
        i = e_p.i
        i = C*v'
END COMPONENT

COMPONENT Inductor IS_A OnePort
    "Inductor"
    PORTS
        IN elec e p
        IN elec e_n
    DATA
        REAL L=1
    DECLS
        REAL i
        REAL v
    CONTINUOUS
        v = e_p.v - e_n.v
        0 = e_p.i + e_n.i
        i = e_p.i
        L*i' = v
END COMPONENT
```

Versión sin Herencia de una Librería Eléctrica Básica

Figura 2-13: Listados de Componentes Básicos de una Librería Eléctrica en EcosimPro con Herencia (lado izquierdo) y sin Herencia (lado derecho)

2.7.3. Dificultad en el Procesamiento Simbólico

Durante la fase de procesamiento simbólico, el usuario puede recibir mensajes de error incomprensibles, y la herramienta puede requerir su intervención para la selección de variables de contorno, variables de estado en caso de índice superior y variables de rasgadura.

Durante el procesamiento simbólico de las ecuaciones, el usuario puede recibir los siguientes tipos de mensajes:

- El sistema de ecuaciones es estructuralmente singular.
- El sistema de ecuaciones tiene restricciones insuficientes, y se solicita al usuario que seleccione variables adicionales como condiciones de contorno.
- El sistema de ecuaciones es sobre-restringido y se deben eliminar ecuaciones.
- El sistema de ecuaciones incluye problemas de alto índice y se solicita al usuario seleccionar las variables dinámicas a considerar.
- El problema contiene lazos algebraicos, y se solicita al usuario que seleccione las variables y ecuaciones de rasgadura.

Estos mensajes pueden resultar confusos o inoportunos para un usuario no experto. El motivo es que no indican claramente la causa del problema de una forma comprensible. En un modelo construido utilizando una librería verificada y siguiendo una metodología establecida, los problemas con las ecuaciones suelen derivar de la topología, y algunas posibles causas pueden ser: puertos sin conectar, definición de conexiones incompatibles entre componentes o falta de algún componente esencial. Por ejemplo, en una herramienta de simulación eléctrica como Spice, los mensajes de error apuntan claramente a la topología. El Listado 2-1 proporciona algunos ejemplos de mensajes de error de Spice.

Listado 2-1-Mensajes Típicos de Error de una Herramientas de Simulación Eléctrica

```
• This circuit does not have a conduction path to ground. Please
  flag a node as ground.
• Node Nxxx is floating
• Less than two connections to node Nxxx
```

2.7.4. Dificultad de Convergencia de las Ecuaciones Algebraicas

La dificultad de convergencia de los lazos algebraicos se debe principalmente al uso de la técnica de rasgadura, cuya falta de robustez se ha discutido ampliamente en la sección 2.5. Esta dificultad afecta a la inicialización de transitorios y el cálculo de estacionarios.

Resolver los lazos algebraicos puede depender de la capacidad del usuario para proporcionar una estimación cercana a la solución real de los valores de las variables de rasgadura.

2.7.5. Problemas de Integración Transitoria

Durante la integración numérica del modelo, es común encontrarse con situaciones en las que el avance de la solución se vuelve extremadamente lento o la integración finaliza de manera inesperada con mensajes de error difíciles de interpretar o, en algunos casos, sin ningún mensaje. Estos problemas pueden ser causados por diversas razones, entre las cuales se destacan:

- Selecciones inapropiadas por el usuario o por la herramienta en la etapa de análisis simbólico de las condiciones de contorno en el caso de problemas indeterminados, de las variables estado en el caso de problemas índice alto, y de las variables y ecuaciones de rasgadura en el caso de lazos algebraicos.
- Discontinuidades en la formulación que no se representan de manera explícita, lo cual requiere reducciones significativas en el incremento de tiempo. Estas discontinuidades a menudo están encapsuladas dentro de funciones.
- Claqueteo (*chattering*), que consiste en un número grande de eventos ocurriendo en un intervalo de tiempo muy corto.
- Ecuaciones altamente no lineales. La resolución de un DAE es equivalente a resolver un sistema de ecuaciones algebraicas, y los métodos de solución de ecuaciones algebraicas se basan en la linealización del sistema de ecuaciones. Por lo tanto, ecuaciones altamente no lineales pueden causar dificultades en los algoritmos de integración de DAE's. Por ejemplo, la siguiente ecuación diferencial ordinaria requiere decenas de miles de pasos de integración para su solución numérica en el intervalo [0, 1] cuando se utiliza un integrador implícito, como DASSL. Esto se debe a que la raíz cuadrada es una función altamente no lineal con una pendiente infinita en las proximidades de $x = 0$.

$$\begin{aligned} &\frac{dx}{dt} = \begin{cases} +\sqrt{x} & \text{si } x \geq 0 \\ -\sqrt{|x|} & \text{si } x < 0 \end{cases} \\ &x(t=0) = 0.1 \end{aligned} \tag{2-32}$$

Ciertamente, los problemas arriba citados no son exclusiva responsabilidad de la herramienta de MSOO utilizada. El diseño de las librerías de componentes también influye en la aparición de dichas dificultades, y un diseño conforme a ciertas reglas evita o ayuda a aliviar la aparición de este tipo de problemas.

Identificar la causa de cualquiera de estos problemas suele requerir un buen conocimiento de los algoritmos de integración implementados en la herramienta y del procesamiento simbólico efectuado por la misma.

2.7.6. Técnicas de Solución Inadecuadas para Modelos de Gran Tamaño

Las herramientas de MSOO son inadecuadas cuando se aplican a modelos de gran tamaño, con decenas de miles de ecuaciones e incluso millones de ecuaciones. Existen al menos tres cuellos de botella en su aplicación a modelos grandes:

1. El procesamiento simbólico de modelos muy grandes requiere un aumento significativo en los recursos computacionales necesarios, tanto en términos de memoria como en el tiempo de computación.
2. Los códigos fuente generados a partir de modelos de gran tamaño pueden tener un número considerable de líneas, ya que cada ecuación generalmente se traduce en al menos una línea de código en la función de cálculo de residuos. Estos códigos fuente extensos pueden generar dificultades durante la compilación, como tiempos de compilación prolongados y fallos del compilador debido a exceder los límites establecidos. Las herramientas de MSOO funcionan de forma muy diferente a los programas de simulación convencionales donde se tienen bucles de código por cada tipo de componente. En MSOO, cada componente da lugar a sentencias repetidas.
3. Falta de resolvedores dispersos: Para abordar correctamente los modelos, se necesitan resolvedores especializados en diferentes tipos de sistemas de ecuaciones dispersas. Se requieren al menos tres tipos de resolvedores dispersos:
   a) Resolvedores dispersos para ecuaciones diferenciales ordinarias y ecuaciones algebraico/diferenciales dispersas.
   b) Resolvedores para sistemas de ecuaciones lineales dispersas.
   c) Resolvedores para sistemas de ecuaciones algebraicas no lineales dispersas.

   Aunque algunas herramientas de MSOO están incorporando resolvedores dispersos, no todas proporcionan los tres tipos mencionados.

En resumen, las herramientas de MSOO no son adecuadas para modelos de gran tamaño debido a las limitaciones en el procesamiento simbólico, la generación de código fuente extenso y la falta de resolvedores específicos para sistemas de ecuaciones dispersas. Es importante considerar estas limitaciones al seleccionar las herramientas de modelado y simulación más apropiadas para proyectos que involucren modelos complejos y de gran envergadura.

2.7.7. Métodos de solución limitados a resolvedores de DAE’s y ODE’s

Los métodos de solución numérica en las herramientas de MSOO son inflexibles y predefinidos, limitándose a resolvedores de ecuaciones algebraico-diferenciales (DAE’s) y ecuaciones diferenciales ordinarias (ODE’s). Esto implica que los usuarios de nivel 1 deben formular los componentes utilizando ecuaciones algebraico-diferenciales, aunque existan formulaciones alternativas más eficientes y precisas, como las ecuaciones en diferencias finitas.

Un ejemplo de esta limitación se observa en herramientas de simulación transitoria de flujos en redes de tuberías con componentes 1D, donde los métodos de diferencias finitas son más adecuados que las representaciones matemáticas basadas en DAE’s u ODE’s.

En resumen, las herramientas de MSOO limitan a los usuarios de nivel 1 a formular modelos como DAE’s híbridos, lo que restringe la flexibilidad en los métodos de solución. Esto puede ser problemático, ya que los usuarios están obligados a utilizar ecuaciones algebraico-diferenciales en lugar de formulaciones más eficientes y precisas, como las ecuaciones en diferencias finitas.

3. SIMULACIÓN ESTACIONARIA

En el presente capítulo, se exploran los fundamentos de la simulación estacionaria y su relevancia en el diseño y análisis de sistemas. Asimismo, se describen las técnicas y principales características de las herramientas habitualmente empleadas para este tipo de simulaciones. Para concluir, se realiza una comparación entre las herramientas desarrolladas específicamente para el cálculo de estacionarios y las herramientas de MSOO aplicadas a simulaciones en estado estacionario.

Entre los usuarios de herramientas de MSOO suele darse por sentado que las simulaciones son predominantemente dinámicas o transitorias, y que el cálculo del estacionario solo sirve como punto de partida para el transitorio. Sin embargo, es importante reconocer que las simulaciones puramente estacionarias desempeñan un papel fundamental y abarcan un amplio campo de aplicación en el diseño de sistemas. A continuación, se listan algunos ejemplos destacados de simulaciones estacionarias:

- El cálculo de balances térmicos en plantas de producción de potencia.
- El cálculo de actuaciones de turbinas de gas y motores de reacción.
- La simulación estacionaria de procesos químicos.
- El cálculo del punto de operación en corriente continua de circuitos eléctricos.

Es importante señalar que la simulación en el dominio de la frecuencia presenta notables similitudes con la simulación estacionaria. La simulación estacionaria conlleva la resolución de sistemas de ecuaciones algebraicas, y la simulación en el dominio de la frecuencia implica la resolución de sistemas de ecuaciones lineales que representan la respuesta de los equipos ante pequeñas variaciones o perturbaciones en el dominio de la frecuencia. Estas ecuaciones representativas del comportamiento de los equipos frente a pequeñas perturbaciones se obtienen aplicando el desarrollo en serie de Taylor a las ecuaciones transitorias (o ecuaciones en el dominio del tiempo) alrededor de un punto de operación dado. Aunque la simulación en frecuencia se asemeja a la simulación estacionaria, su complejidad es inferior debido a la naturaleza lineal de las ecuaciones que deben resolverse, lo que elimina la necesidad de iteraciones para alcanzar la solución.

Debido a la importancia de la simulación estacionaria, se ha evidenciado un creciente interés por expandir el campo de aplicación de las herramientas de MSOO hacia el análisis de sistemas en estado estacionario. Una empresa líder en la aplicación de Modelica ha lanzado una página web bajo el título "Steady States the Next Big Thing" (Sieleman, 2015), donde se describe claramente este objetivo.

La adaptación de las herramientas de MSOO para el análisis de sistemas estacionarios presenta desafíos de consideración. Estas herramientas se derivan de las herramientas de simulación continua del tipo CSSL, que originalmente estaban diseñadas para la simulación

continua y representaban el comportamiento del sistema mediante un conjunto de ecuaciones diferenciales ordinarias, con el tiempo como variable independiente. En este contexto, el estado estacionario se considera simplemente una condición inicial para la integración del modelo dinámico. Por lo tanto, es esperable que estas herramientas no ofrezcan de forma natural muchas de las capacidades propias de las herramientas de simulación estacionaria. Ampliar su aplicabilidad al análisis de sistemas en estado estacionario puede exigir la incorporación de ajustes y capacidades adicionales, con el fin de equiparar su eficacia con las herramientas diseñadas específicamente para este propósito.

3.1. Características Específicas de las Herramientas de Simulación Estacionaria

En esta sección se presentan las características distintivas de las herramientas de simulación estacionaria, las cuales son las siguientes:

- La representación matemática del modelo se basa en un sistema de ecuaciones algebraicas.
- El modelo se construye como un conjunto de componentes que se conectan bien mediante nodos (“nodes”) o bien mediante corrientes (“streams”).
- El método de solución puede ser del tipo modular secuencial o del tipo orientado a ecuaciones.
- En las herramientas orientadas a ecuaciones, la rasgadura del sistema de ecuaciones se realiza de forma sistemática.
- Facilitan la inclusión de ecuaciones condicionales, lo que significa que permiten modificar las ecuaciones consideradas dependiendo de la ubicación de la solución.
- Facilitan la solución de problemas de diseño, a través de diversas técnicas que incluyen:
    - La habilidad de cambiar entre dos formulaciones de los componentes: una para el *diseño* y otra para situaciones *fuera de diseño*.
    - La introducción de ecuaciones adicionales, tales como especificaciones de diseño que imponen valores exactos o límites inferiores/superiores en ciertos resultados para uno o varios puntos de diseño (diseño monopunto y diseño multipunto).

A continuación, se describen en detalle estas características y capacidades de las herramientas de simulación estacionaria.

3.1.1. Representación de los modelos mediante ecuaciones algebraicas

La representación matemática del modelo en una herramienta de simulación estacionaria es un sistema de ecuaciones algebraicas con igual número de incógnitas que de ecuaciones. El sistema de ecuaciones puede expresarse de la siguiente forma simplificada:

$$\boldsymbol{F}\,(\boldsymbol{x},\boldsymbol{p}) = \boldsymbol{0} \tag{3-1}$$

donde $\boldsymbol{F}$ es un vector de n ecuaciones, $\boldsymbol{x}$ es un vector de n incógnitas y $\boldsymbol{p}$ es un vector de m datos o parámetros.

Normalmente, todas las herramientas de simulación estacionaria realizan una rasgadura del sistema de ecuaciones que, si bien no busca una optimización completa, logra reducir de manera significativa la cantidad de ecuaciones implícitas.

El sistema de ecuaciones rasgado puede expresarse de la siguiente manera:

$$y_i = g_i(\boldsymbol{y}_{1:i-1}, \boldsymbol{z}, \boldsymbol{p}) \qquad \forall i \in \{1,2,\ldots.,n_y\} \tag{3-2}$$

$$\boldsymbol{H}\,(\boldsymbol{z},\boldsymbol{y},\boldsymbol{p}) = \boldsymbol{0} \tag{3-3}$$

donde y_i representa la i-ésima variable calculada de manera explícita, n_y es el número total de variables calculadas explícitamente, g_i es una función explícita obtenida al despejar y_i en la ecuación adecuada, $\boldsymbol{z}$ es el vector de variables de rasgadura de dimensión $n_z = n - n_y$, $\boldsymbol{H}$ son las ecuaciones de rasgadura también de dimensión n_z, y los subíndices $1{:}\,i-1$ se utilizan para indicar el conjunto $1, 2, \ldots, i-1$.

3.1.2. Conexionado mediante nodos o mediante corrientes

En las herramientas estacionarias, los modelos se construyen de forma modular, esto es como un conjunto de componentes interconectados. En este contexto, se distinguen dos elementos esenciales para conectar componentes entre sí: los nodos y las corrientes.

Nodos como piezas de unión

En las herramientas para simulación de circuitos eléctricos, y también en las herramientas para simulación de circuitos hidráulicos, el nodo es el elemento que sirve para conectar componentes entre sí. En el caso de los circuitos eléctricos, un nodo representa un punto de conexión de varios componentes que mantiene un voltaje común y en el que la suma de corrientes consideradas con su signo (positivas si entran al nodo y negativas si salen del nodo) debe ser nula. En los modelos de circuitos eléctricos, las conexiones son cables ideales con resistencia nula, por lo que un nodo consta de toda la zona de cable entre componentes, no se limita solo a un punto. La Figura 3-1 muestra un ejemplo de un circuito

eléctrico con tres nodos, donde cada zona coloreada (rojo, azul y verde) representa uno de los nodos.

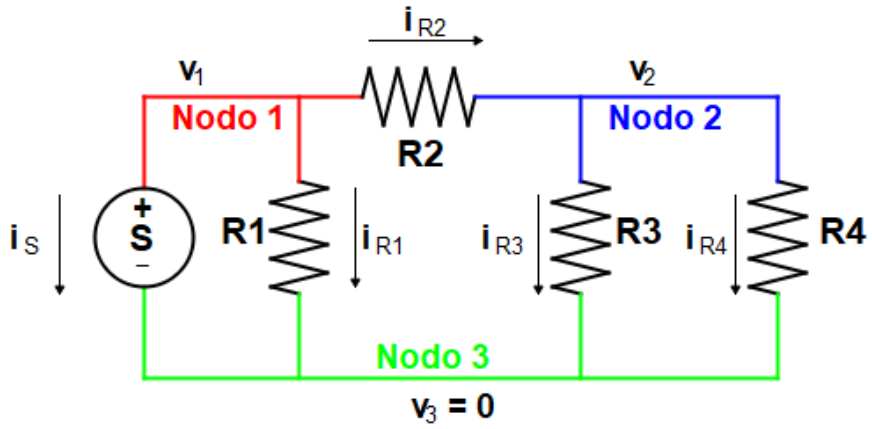


Figura 3-1: Nodos en un Modelo de Circuito Eléctrico.

En el caso de los circuitos hidráulicos, donde se analizan presiones y flujos, los modelos son completamente equivalentes a los de los circuitos eléctricos. Por consiguiente, también se emplea el conexionado mediante nodos en los circuitos hidráulicos. Similarmente a los circuitos eléctricos, los nodos de los circuitos hidráulicos representan puntos en los que la presión es común, y donde la suma de los flujos de entrada y salida en el nodo debe ser nula para cumplir con la ley de conservación de masa.

La adopción de nodos como elementos de conexión en los modelos de circuitos hidráulicos simplifica considerablemente su análisis, ya que permite la aplicación de conceptos y metodologías provenientes de la teoría de circuitos eléctricos.

Corrientes como piezas de unión

En el ámbito de la simulación de procesos químicos y sistemas de conversión de energía, el conexionado entre componentes se realiza mediante corrientes (streams) de diversos tipos: corrientes de masa, corrientes de energía y corrientes de información.

El modelo (flowsheet) de una planta de proceso o de un sistema de conversión de energía se compone de un conjunto de módulos o componentes interconectados mediante corrientes másicas, corrientes de energía y corrientes de información.

Una corriente másica representa un intercambio individual de flujo másico entre dos componentes. Las corrientes másicas se caracterizan por el flujo másico total, la composición del flujo másico intercambiado, la presión y el nivel de energía del fluido, que está representado por la temperatura o la entalpía.

Las corrientes de energía se utilizan para representar la transferencia de energía en sus diversas formas entre los componentes del sistema. En este contexto, es común encontrar

varios tipos de corrientes de energía que se corresponden con cada una de las formas de energía presentes, tales como conexiones de energía eléctrica, conexiones de energía mecánica rotacional y conexiones de energía térmica.

Además de las corrientes de masa y energía, algunas herramientas de simulación también incorporan corrientes de información. Estas corrientes representan la transmisión de datos, señales o información entre los distintos componentes del sistema. La información transmitida puede incluir parámetros de control, valores de variables, señales de activación y cualquier otro tipo de información necesaria para el funcionamiento del sistema.

Las corrientes siempre se representan mediante líneas rectas o quebradas con flechas superpuestas. En el caso de las corrientes de masa y energía, las flechas indican el sentido positivo del flujo, bien sea de masa o energía. En las corrientes de información, las flechas representan la dirección de transmisión de la información entre los diferentes elementos del sistema. En numerosas herramientas de simulación de procesos y sistemas de conversión de energía, se establece la restricción de que no se permiten flujos másicos negativos.

La Figura 3-2 muestra un modelo de un turborreactor donde las líneas azules representan corrientes de masa y las líneas de color rojo oscuro representan corrientes de energía mecánica transmitida a través de un eje.

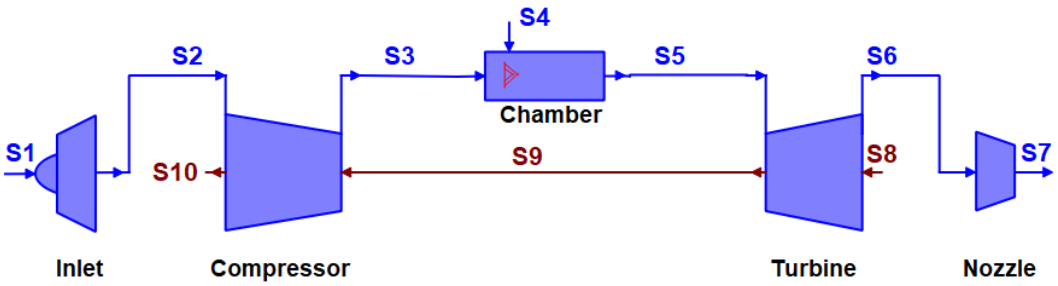


Figura 3-2: Corrientes (Streams) en un Modelo de Turborreactor.

El uso de “corrientes” en el software de simulación de procesos y de plantas de conversión de energía conlleva la necesidad de emplear componentes específicos para dividir las corrientes fluidas (divisores) y para reunirlas (mezcladores). Es interesante resaltar que estos componentes son innecesarios en el conexionado mediante nodos, salvo cuando un componente divisor o mezclador impone una relación entre los flujos o propiedades de las corrientes. Un ejemplo de esta situación se presenta cuando un mezclador o divisor establece que el flujo a través de una de las ramas es un porcentaje determinado del flujo total.

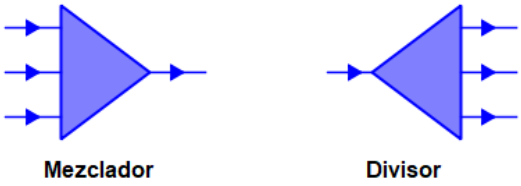


Figura 3-3: Componentes Mezclador y Divisor de Componentes

La selección de las piezas de conexión entre componentes es una cuestión fundamental en el desarrollo de herramientas de simulación de enfoque modular. En el caso de problemas termo-fluidos, la opción común es optar directamente por las corrientes como elementos de conexión para representar el flujo de masa y energía entre los componentes del sistema. Sin embargo, existen situaciones en las cuales la aproximación nodal podría haber sido una opción viable y beneficiosa, incluso para problemas termo-fluidos, especialmente en sistemas complejos con numerosos colectores. Algunas las ventajas de adoptar la aproximación nodal en estos casos son:

- *Simplificación del modelo*: Al conectar componentes a nodos específicos en lugar de trazar corrientes individuales, se reduce la cantidad de elementos en el modelo y se simplifica su estructura. Esto facilita la comprensión del sistema y mejora la eficiencia computacional en la simulación.
- *Reducción de componentes adicionales*: Al utilizar nodos como puntos de conexión, se elimina la necesidad de utilizar componentes adicionales como divisores y mezcladores, lo que reduce la complejidad del modelo.

3.1.3. Método de Solución: Modular Secuencial u Orientado a Ecuaciones

Dentro de los programas de cálculo de estados estacionarios, se encuentran dos tipos de métodos de solución: el modular secuencial (MS) y el orientado a ecuaciones (OE). El método modular secuencial se restringe a la simulación de procesos químicos o de producción de energía en estado estacionario. Por el contrario, el método orientado a ecuaciones es mucho más versátil, ya que puede aplicarse a cualquier tipo de sistema, tanto para simulaciones estacionarias como transitorias.

Aproximación modular secuencial

En la aproximación modular secuencial, los cálculos se realizan componente a componente siguiendo una secuencia de cálculo específica. Cada componente no es simplemente un conjunto de ecuaciones, sino que también incluye un procedimiento de solución integrado.

El uso de la aproximación modular secuencial se vincula al empleo de corrientes como elementos conectivos entre los componentes del sistema. En este enfoque, las variables asociadas a las corrientes que entran y salen de un componente se clasifican en “entradas”

y “salidas computacionales”. El procedimiento de solución integrado en cada componente calcula el valor de las variables de salida computacional en función de las entradas y de parámetros o datos del componente que definen sus actuaciones.

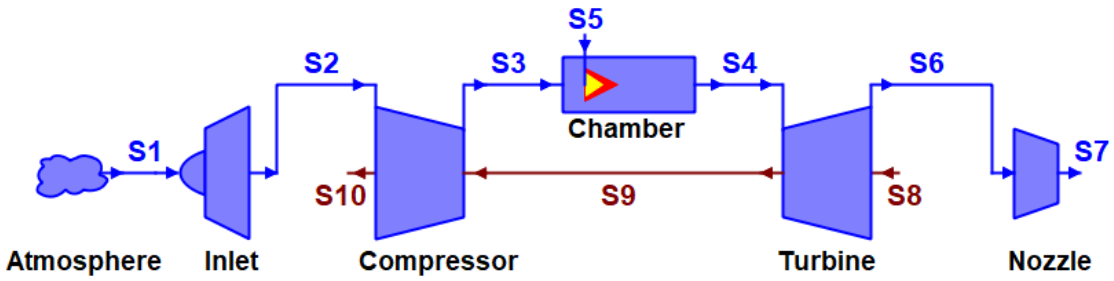
En general, en las corrientes de flujo másico, la causalidad computacional tiende a seguir la misma dirección que el flujo positivo. En consecuencia, las variables vinculadas a las corrientes fluidas que ingresan al componente son habitualmente "entradas computacionales", mientras que las variables asociadas a las corrientes que salen del componente suelen ser "salidas computacionales", dado que son calculadas por el componente y representan el resultado o respuesta para ciertas entradas específicas.

Es importante señalar que, aunque la causalidad computacional y la dirección de la corriente suelen coincidir, no siempre es así. En ciertos casos, el flujo de la información computacional puede ser en dirección contraria a la dirección de flujo positivo en la corriente.

La herramienta de simulación busca una ordenación para la solución individual de los componentes, de tal manera que los procedimientos de simulación de los componentes se vayan ejecutando una vez que sus entradas hayan sido calculadas por otros componentes. Por ejemplo, la Tabla 3-1 muestra la secuencia de cálculo para un modelo de un turborreactor, similar al mostrado en la Figura 3-2, cuando se utiliza una formulación correspondiente al diseño usando la Aproximación Modular Secuencial.

Tabla 3-1: Secuencia de Cálculo para un Turborreactor en Diseño usando la Aproximación Modular Secuencial

Componente	**Procedimiento Solución del Componente** (ver Nota 1)
Atmosphere	$[S1] = f1(m_{S1,D},\ Cond.Atm.\ Diseño,\ V_{vuelo,D})$
Inlet	$[S2] = f2(S1,\ \eta_{Inlet,D})$
Compressor	$[S3,\ W_{S9,D}] = f3(S2,\ \Pi_{comp,D},\ \eta_{comp,D})$
Chamber	$[S4,\ m_{S5,D}] = f4(S3,\ T_{S4,D})$
Turbine	$[S6] = f5(S4,\ W_{S9,D},\ \eta_{turb,D})$
Nozzle	$[S7,\ A_{nozzle,D}] = f6(S6,\ P_{Atm,D})$



Nota 1: En la tabla anterior, Sx denota el conjunto de variables asociadas con la corriente número x. En el caso de corrientes de flujo, esas variables son: flujo másico m_{Sx}, presión de remanso P_{Sx}, temperatura de remanso T_{Sx} y composición y_{Sx}. En el caso de corrientes de potencia mecánica rotatoria, las variables asociadas son la potencia mecánica W_{Sx} y la velocidad de giro n_{Sx}. El sufijo D en los subíndices indica condiciones de diseño, así $m_{S1,D}$ es el flujo másico por la corriente número para condiciones de diseño.

A continuación, se detallan los pasos de la secuencia de cálculo de la Tabla 3-1:

1. El componente **`Atmosphere`** calcula la corriente $S1$, donde se supone que el flujo másico $m_{S1,D}$ es una especificación de diseño. El estado termodinámico se obtiene a partir de las condiciones atmosféricas de diseño y de la velocidad de vuelo de diseño, $V_{vuelo,D}$.
2. El componente **`Inlet`** calcula la corriente $S2$, utilizando como entradas la corriente $S1$ y el rendimiento isentrópico de la toma en el punto de diseño, $\eta_{Inlet,D}$.
3. El componente **`Compressor`** calcula la corriente de salida $S3$ y la potencia mecánica del compresor en el punto de diseño, $W_{S9,D}$, a partir de la corriente $S2$, de la relación de compresión $\Pi_{comp,D}$ y el rendimiento isentrópico $\eta_{comp,D}$ en el punto de diseño.

4. El componente **`Chamber`** calcula la corriente $S4$ a partir de $S3$ y de la temperatura de entrada a la turbina, $T_{S4,D}$, que es parámetro de diseño; también calcula el flujo de combustible $m_{S5,D}$ necesario para conseguir esa temperatura.
5. El componente **`Turbine`** calcula la corriente de salida $S6$ a partir de la corriente $S5$, la potencia requerida por el compresor en la corriente $W_{S9,D}$ y el rendimiento de la turbina en el punto de diseño, $\eta_{turb,D}$.
6. Finalmente, el componente **`Nozzle`** calcula la corriente $S7$ y el área de salida de la tobera en diseño, $A_{nozzle,D}$, a partir de $S6$ y la presión atmosférica en diseño.

En este ejemplo, el flujo de información de las corrientes másicas es en el sentido del flujo positivo (salvo para la corriente de combustible $S5$, que es calculada por el componente **`Chamber`**). En el caso de la corriente de potencia mecánica $S9$, la dirección de la causalidad computacional es contraria al flujo de potencia (la turbina suministra potencia al compresor, pero es el compresor quien fija la potencia).

En la aplicación del método modular secuencial, es posible encontrar situaciones en las que existen bucles o ciclos en el flujo de información. Estos lazos imposibilitan encontrar una secuencia de solución de los componentes, de manera que las entradas a cada componente hayan sido calculadas antes de abordar la solución del mismo. En tales situaciones, se eligen corrientes para romper los lazos, estas corrientes se denominan corrientes de rasgadura (tearing), y se asume el valor de las variables asociadas a ellas. Seguidamente, se calculan secuencialmente todos los componentes hasta resolver todo el diagrama de flujo y obtener nuevos valores para las variables en las corrientes de rasgadura. En general, los valores asumidos y los valores calculados de las variables de las corrientes de rasgadura no coincidirán, por lo que será necesario estimar una corrección para dichas variables y repetir el cálculo hasta conseguir convergencia. La convergencia de las corrientes de rasgadura implica resolver el siguiente sistema de ecuaciones algebraicas:

$$\boldsymbol{R}\,(\boldsymbol{y}) = \boldsymbol{g}\,(\boldsymbol{y}) - \boldsymbol{y} = \boldsymbol{0} \tag{3-4}$$

donde $\boldsymbol{y}$ es el vector de valores asumidos para las variables de las corrientes de rasgadura y $\boldsymbol{g}\,(\boldsymbol{y})$ es el vector de valores calculados para las variables en esas corrientes tras completar un ciclo a través de los procedimientos de solución de los componentes en el diagrama de flujo.

Existen diversas estrategias para estimar las variables de rasgadura, desde una mera sustitución, como se indica en la siguiente expresión:

$$\boldsymbol{y}^{k+1} = \boldsymbol{g}\left(\boldsymbol{y}^{k}\right) \tag{3-5}$$

donde los superíndices denotan el número de iteración.

Otra estrategia es el método de Wegstein (Wegstein, 1958), que acelera la convergencia de las sustituciones utilizando resultados de iteraciones previas. También, es posible resolver directamente el sistema de ecuaciones (3-4) utilizando el método de Newton-Raphson o el de Broyden (Broyden, 1965).

La aproximación modular secuencial es sencilla de aplicar cuando los flujos de información coinciden con la dirección de las corrientes y el número de corrientes de reciclado es moderado.

Aproximación orientada a ecuaciones

La aproximación de simulación de procesos orientada a ecuaciones considera todo el diagrama de flujo del proceso como un sistema de ecuaciones simultáneas, Cada componente se representa por un conjunto de ecuaciones que definen relaciones sobre un conjunto de variables. En esta aproximación, todas las ecuaciones de modelado, que incluyen las ecuaciones de los componentes, las conexiones y las especificaciones de diseño, se combinan para formar un sistema de ecuaciones dispersas. En simulaciones estacionarias, este sistema adopta la forma de un sistema disperso de Ecuaciones Algebraicas No Lineales; en cambio, en simulaciones dinámicas, adopta la forma de un sistema disperso de Ecuaciones Algebraico-Diferenciales.

Todas las herramientas de MSOO utilizan la aproximación orientada a ecuaciones. Aunque esta aproximación tiene sus ventajas, existen problemas de simulación en los que la aproximación modular secuencial es más adecuada. Por tanto, la orientación exclusiva a ecuaciones de las herramientas de MSOO hace que las aplicaciones desarrolladas con ellas puedan resultar inferiores a herramientas específicas que siguen el enfoque modular secuencial.

Comparación de la aproximación orientada a ecuaciones y la aproximación modular secuencial

A continuación, se presentan las ventajas y desventajas de cada una de las dos aproximaciones: la modular secuencial y la orientada a ecuaciones. También, se indican los criterios para seleccionar la aproximación más adecuada para un determinado tipo de problemas.

Entre las ventajas de la aproximación Modular Secuencial, cabe mencionar:

- *Tamaño del problema*: Es adecuada para problemas grandes, ya que los componentes se resuelven de manera secuencial y no se requiere la solución simultánea de un gran sistema de ecuaciones.
- *Modularidad del código*: Introducir un nuevo componente implica desarrollar una subrutina que resuelva las ecuaciones que describen su comportamiento. Las ecuaciones de cada componente se resuelven de la forma más adecuada al

componente en cuestión. Esta estructura modular facilita la reutilización de código y el desarrollo incremental de la herramienta.

- *Programación y mantenimiento sencillos*: El enfoque modular del código hace que su programación y mantenimiento sean más fáciles de realizar.
- *Control de la convergencia*: Permite un control más sencillo de la convergencia tanto al nivel de los componentes como del diagrama de flujo completo (flow-sheet).
- *Corrección de errores y robustez*: Al resolver los componentes de manera secuencial, las primeras iteraciones suelen funcionar adecuadamente, lo que proporciona información útil para detectar y corregir errores en el modelo.

En cuanto a las desventajas de la aproximación modular secuencial, se tienen la siguientes:

- *Necesidad de análisis topológico*: Requiere un análisis topológico complejo ya que es necesario identificar los reciclajes y seleccionar las corrientes de rasgado (tear streams).
- *Lentitud de la convergencia*: Puede tener una velocidad de convergencia lenta. Esta limitación puede prolongar significativamente el tiempo de cálculo y afectar a la eficiencia del proceso de simulación.
- *Falta de adecuación a problemas de diseño y optimización*: No resulta adecuada para abordar problemas de diseño ni de optimización, debido a las dificultades para introducir especificaciones de diseño. Una especificación de diseño consiste en determinar un dato de diseño, que se considera independiente, como puede ser el flujo másico por un turborreactor, con el objetivo de lograr un resultado específico, como puede ser el empuje total del motor. Cuando la variable de diseño y la variable de resultado a conseguir se localizan en componentes diferentes, se generan corrientes de información adicionales que pueden ocasionar reciclajes y complicar la convergencia del modelo.
- *Falta de adecuación en caso de múltiples recirculaciones*: Esta metodología no resulta apropiada cuando se presentan numerosas recirculaciones, bien de flujo másico o bien de los flujos de información asociados a la causalidad computacional. Por ejemplo, en una red de tuberías, la información de presiones se propaga en dirección aguas arriba; por ende, la aproximación modular secuencial no resulta apropiada.
- *No es adecuada para simulación transitorias*: La aproximación modular secuencial se centra en resolver el estado estacionario del sistema, por lo que no es la mejor opción para problemas que implican simulación transitoria.

En cuanto a las ventajas de la aproximación Orientada a Ecuaciones (OE), se tienen las siguientes:

- *Adecuación a problemas de diseño y optimización*: Es adecuada para la resolución de problemas de diseño y optimización debido a su capacidad para imponer especificaciones de diseño en forma de ecuaciones adicionales.
- *Mejor tratamiento de reciclajes*: Es adecuada para problemas con muchas recirculaciones de flujo o de información.
- *Rápida convergencia*: Presenta muy buena velocidad de convergencia cuando la solución estimada se encuentra cercana a la solución real.

Ahora bien, la aproximación orientada a ecuaciones presenta las siguientes desventajas:

- *Mayor esfuerzo de programación*: Requiere más esfuerzo de programación en comparación con la aproximación modular secuencial.
- *Necesidad de recursos informáticos*: Puede requerir recursos informáticos sustanciales, aunque esta desventaja es cada vez menos problemática.
- *Dificultades en la depuración de modelos*: A diferencia de la aproximación modular secuencial, que siempre brinda resultados durante las primeras iteraciones, este no es el caso con la aproximación orientada a ecuaciones. En ocasiones, el programa puede finalizar de manera inesperada y los mensajes de error pueden ser insuficientes para identificar el problema de manera efectiva.
- *Requisito de una estimación inicial precisa de la solución*: La convergencia puede fallar si la estimación inicial de la solución no es lo suficientemente cercana a la solución real.

La Tabla 3-2 resume las ventajas y desventajas de ambas aproximaciones, tal como se han indicado anteriormente.

Tabla 3-2: Comparación de la Aproximación Modular secuencial (SM) y la Orientada a Ecuaciones (OE)

	Modular secuencial	**Orientada a Ecuaciones**
	Ventajas	**Desventajas**
Tamaño del problema	El problema de mayor tamaño que requiere solución simultánea es el del componente con el modelo matemático más extenso.	El sistema de ecuaciones aumenta con el tamaño del modelo, lo que demanda el uso de algoritmos para resolver ecuaciones dispersas en casos de modelos extensos.
Modularidad del Código	Desarrollo modular con subrutinas específicas para cada componente, resolviendo sus ecuaciones de manera independiente y adaptada.	Cada componente contribuye con ecuaciones al sistema global del modelo. Un solo error en una de estas ecuaciones puede afectar la obtención de resultados intermedios..
Programación y Mantenimiento	La programación y el mantenimiento son sencillos, pero requiere un análisis topológico para detectar los reciclaje y seleccionar las corrientes de rasgadura.	Programación compleja. Cada componente añade sus ecuaciones a un sistema global.
Control de la Convergencia	Es fácil el control de la convergencia al nivel de componente y al nivel del modelo (flow-sheet).	La convergencia depende de una estimación inicial precisa y cercana a la solución real. Si la estimación inicial no es lo suficientemente próxima, la convergencia puede fallar.
Corrección de Errores y Robustez	La aproximación es robusta y fiable. Las primeras iteraciones funcionan siempre y es posible obtener información para emprender la depuración del modelo.	Existe la posibilidad de que el programa termine de forma inesperada y que los mensajes obtenidos no resulten útiles para localizar el problema.
	Desventajas	**Ventajas**
Velocidad de Convergencia	Velocidad de convergencia lenta, que puede dar lugar iteraciones innecesarias.	Velocidad de convergencia muy buena cuando se está cerca de la solución.
Problemas de diseño y optimización	No es adecuada para problemas de diseño pues la introducción de las restricciones de diseño puede introducir reciclajes adicionales.	Facilidad para imponer restricciones de diseño. Es adecuada para la resolución de problemas de diseño y de optimización
Recirculaciones de flujo	Problemas con las recirculaciones de flujo y las recirculaciones de información	Tiene en cuenta sin problema las recirculaciones y las ecuaciones que describen el flujo en redes
Cálculo de presiones y flujos en redes de tuberías	En una red de tuberías existe un flujo de información en dirección aguas arriba de las presiones. Por tanto, la aproximación modular secuencial no es adecuada.	Resuelve problemas de presiones-flujos en redes de tuberías
Simulación Transitoria	No es adecuada para la simulación transitoria	Adecuada para simulación estacionaria y transitoria

Al emprender el diseño de un nuevo programa de simulación, es crucial considerar qué enfoque utilizar. Morton (2003) planteó que la elección de la aproximación más adecuada puede basarse en dos parámetros: (1) la complejidad de las corrientes de recirculación y (2) la no linealidad de las ecuaciones del modelo. Representando ambos parámetros en un gráfico con ejes, como se ilustra en la Figura 3-4, es posible clasificar los problemas en las cuatro categorías siguientes:

1. *Modelos con pequeñas no linealidades y pocas corrientes de recirculación*. En este caso, ambas aproximaciones, tanto la Modular Secuencial como la Orientada a Ecuaciones, son adecuadas.
2. *Modelos con ecuaciones muy no lineales y pocas recirculaciones*. En este caso, la aproximación Modular Secuencial es superior a la Orientada a Ecuaciones. Esto se debe a que permite dividir el problema en sub-problemas más manejables, lo cual resulta especialmente útil cuando algunos componentes son altamente no lineales y difíciles de resolver, pudiendo requerir soluciones iterativas internas. En cambio, la Orientada a Ecuaciones no permite soluciones personalizadas para componentes difíciles de resolver.
3. *Modelos con pequeñas no linealidades y numerosas recirculaciones* (de flujo o de información). La aproximación más adecuada es la Orientada a Ecuaciones, pues es capaz de considerar las recirculaciones y si la no linealidad es pequeña, la convergencia del método de Newton-Raphson es buena. Los modelos hidráulicos de redes de tuberías constituyen ejemplos de modelos de este tipo, donde las recirculaciones de información son totales, ya que un elemento cualquiera del circuito influye en los resultados de todo el circuito.
4. *Modelos con ecuaciones muy no lineales y con muchas recirculaciones*. Estos representan los problemas más difíciles de resolver, y la aproximación más indicada es la Orientada a Ecuaciones. Sin embargo, se requiere una buena estimación inicial de la solución real para lograr la convergencia. Un ejemplo de problema de este tipo es la simulación de una columna de destilación con múltiples bandejas.

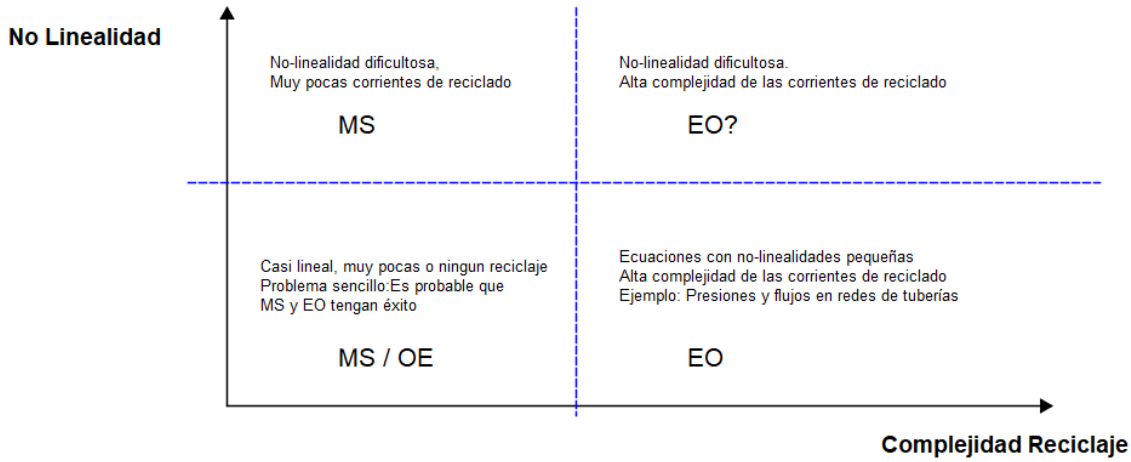


Figura 3-4: Clasificación de problemas por su adecuación a la Aproximación Modular secuencial o a la Orientada a Ecuaciones

Tanto la aproximación modular secuencial (MS) como la orientada a ecuaciones (OE) en la actualidad continúan siendo ampliamente utilizadas en la simulación de procesos. En la Tabla 3-2 se proporcionan ejemplos de programas de simulación que siguen cada una de estas aproximaciones:

Tabla 3-3: Ejemplos de Herramientas de Simulación Estacionaria que usan la Aproximación Modular Secuencial y la Aproximación Orientada a Ecuaciones

	Herramientas Modular Secuenciales	**Herramientas Orientadas a Ecuaciones**
SIMULACIÓN DE PROCESOS QUÍMICOS	Aspen Plus Aspen Hysys ChemCAD PRO II Prosim COCO	gPROMS Aspen Plus in EO mode Aspen Custom Modeler ASCEND
SISTEMAS DE PRODUCCIÓN DE ENERGÍA	Thermoflex GateCycle	PROOSIS

Existe una tendencia a subestimar la aproximación secuencial debido a que los primeros simuladores de procesos químicos fueron secuenciales, y la aproximación orientada a ecuaciones se considera más potente en comparación. No obstante, es importante destacar que la aproximación modular secuencial (MS) sigue siendo ampliamente aplicada en la simulación de procesos y ofrece ventajas significativas en ciertos contextos.

Un claro ejemplo que respalda esta afirmación es el programa Thermoflex (Thermoflow Inc., 2015), utilizado para el cálculo de balance térmico en plantas de producción de energía eléctrica, el cual se basa en la aproximación modular secuencial. Con el uso de esta aproximación, Thermoflex logra una mayor robustez y facilita la solución de un problema típico del usuario de programas de balances térmicos: cuadrar el número de ecuaciones con el de incógnitas.

Es común que los usuarios de programas de balance de masa y energía sufran dificultades para cuadrar o igualar el número de ecuaciones del modelo con el número de incógnitas. Cuando se utiliza la aproximación orientada a ecuaciones, el programa no inicia su funcionamiento hasta que el usuario cuadra el número de ecuaciones e incógnitas, lo cual puede ser complicado para aquellos con experiencia limitada en simulación. En contraste, las herramientas basadas en la aproximación secuencial, como Thermoflex, permiten que el programa funcione incluso en situaciones en que el número de ecuaciones y de incógnitas no coinciden, y proporcionan alertas en caso de que un componente recalcule el valor de una variable que ya ha sido calculada por otro componente, o que una variable no esté siendo calculada por ningún componente. Esto facilita que el usuario pueda identificar y corregir el problema de manera más sencilla y eficiente.

La combinación de ambas aproximaciones en la simulación de procesos es una tendencia prometedora. Por ejemplo, la herramienta comercial Aspen Plus ofrece ambos métodos, permitiendo abordar inicialmente los modelos mediante la aproximación secuencial y, posteriormente, cambiar sin dificultad a la aproximación orientada a ecuaciones. Esta versatilidad brinda lo mejor de ambos enfoques: la aproximación modular secuencial ofrece robustez y facilidad de depuración de errores en sistemas complejos, mientras que la aproximación orientada a ecuaciones proporciona una rápida velocidad de convergencia y facilita el planteamiento de problemas de diseño y optimización.

3.1.4. Rasgadura Sistemática del Sistema de Ecuaciones

Esta sección se refiere exclusivamente a las herramientas orientadas a ecuaciones. Por "rasgadura sistemática del sistema de ecuaciones" se hace referencia a que estas herramientas aplican un conjunto de reglas predefinidas para generar de manera ordenada y estructurada el sistema de ecuaciones que se utilizará para resolver un modelo específico. Estas reglas previamente establecidas determinan qué variables serán consideradas como incógnitas (variables de rasgadura) y qué ecuaciones serán tratadas como implícitas en el modelo (ecuaciones de rasgadura).

En lugar de depender de un análisis simbólico detallado de las ecuaciones, estas herramientas siguen un enfoque sistemático para rasgar y organizar las ecuaciones según las reglas establecidas. El propósito es simplificar el sistema original, reduciendo el número de ecuaciones, manteniendo al mismo tiempo la estructura dispersa del sistema de ecuaciones.

La "construcción sistemática del sistema de ecuaciones" garantiza una mayor eficiencia y robustez en la resolución, dado que evita intentar utilizar la dispersión del sistema en una etapa de manipulación simbólica previa, cuando aún no se conocen los valores numéricos de las variables y de los residuos de las ecuaciones. De esta manera, se logra una mayor robustez y se obtienen resultados más confiables y precisos para el problema en consideración.

Normalmente, la forma de realizar la rasgadura del sistema de ecuaciones está estrechamente relacionada con el tipo de elemento conectivo utilizado por la herramienta, ya sea nodo o corriente.

Rasgadura del sistema de ecuaciones en herramientas con nodos

Las herramientas de simulación que implementan nodos como elemento conectivo, como los simuladores eléctricos del tipo Spice, emplean un método llamado análisis nodal modificado para construir el sistema de ecuaciones.

En este método, las incógnitas son los voltajes en todos los nodos, excepto en uno que se selecciona como referencia (el nodo de tierra). También se consideran incógnitas las corrientes en aquellos elementos (componentes eléctricos) en los que la corriente no puede expresarse de forma explícita en función de los potenciales en los nodos.

Las ecuaciones implícitas que se consideran son las siguientes: para cada nodo (excepto el de referencia), la suma de las corrientes debe ser nula. Además, se incluyen las relaciones constitutivas de los componentes donde la corriente no puede expresarse explícitamente en función de los voltajes nodales. Por ejemplo, en algunos elementos, como las fuentes de voltaje o los elementos no lineales, la corriente no puede obtenerse explícitamente a partir de los voltajes en los nodos, y es necesario incluir las relaciones constitutivas de estos elementos en forma implícita.

A continuación, se muestra la aplicación del método nodal modificado al circuito mostrado en la Figura 3-5.

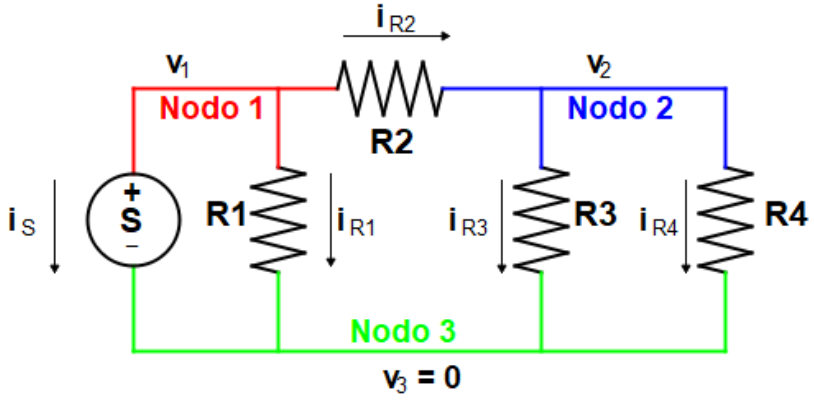


Figura 3-5: Ejemplo de Circuito Eléctrico para Ilustrar el Método Nodal Modificado

Para aplicar el método nodal modificado, primero se elige un nodo como referencia, por ejemplo, el nodo 3, y luego se aplica la ley de Kirchoff de suma de las corrientes a los 2 nodos restantes.

$$\begin{aligned} &\text{Nodo 1:} \quad i_S + i_{R1} + i_{R2} = 0 \\ &\text{Nodo 2:} \quad -i_{R2} + i_{R3} + i_{R4} = 0 \end{aligned} \tag{3-6}$$

Seguidamente, se expresan las corrientes en función de los voltajes en los nodos siempre que sea posible. En las resistencias se puede calcular la corriente en función de los voltajes en los nodos utilizando la ley de Ohm. Sin embargo, en la fuente de voltaje no es posible calcular la corriente, por lo que se añade la ecuación representativa de la fuente. Esta ecuación establece que el voltaje en el nodo 1 es igual a la diferencia de potencial de la fuente, E_S, que es un dato conocido.

$$\begin{aligned} &\text{Nodo 1:} \quad i_S + (v_1 - v_2)/R1 + v_2/R2 = 0 \\ &\text{Nodo 2:} \quad -v_2/R2 + v_2/R3 + v_2/R4 = 0 \\ &\text{Fuente:} \quad v_1 = E_S \end{aligned} \tag{3-7}$$

Finalmente, expresando estas ecuaciones en forma matricial se obtiene la formulación nodal modificada del circuito de la Figura 3-5.

$$\begin{bmatrix} 1/R1 & -1/R1 + 1/R2 & 1 \\ 0 & -1/R2 + 1/R3 + 1/R4 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ i_S \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ E_S \end{bmatrix} \tag{3-8}$$

Al examinar la ecuación (3-8) puede comprobarse que cada componente del circuito introduce contribuciones en la matriz que se suman con las de otros componentes. Las contribuciones aparecen en las filas y en las columnas de los nodos conectados por el componente, lo que refleja cómo cada elemento afecta a las ecuaciones del sistema en función de su conexión con otros elementos.

Esta característica facilita la generación del sistema de ecuaciones mediante un bucle que recorre todos los componentes y agrega la contribución de cada uno de ellos. A este proceso se le denomina "estampación". Así, al sumar todas las contribuciones de los componentes, se obtiene el sistema de ecuaciones completo que representa el comportamiento del circuito.

Rasgadura del sistema de ecuaciones en herramientas con corrientes

Una corriente fluida que contenga N_c componentes y que se encuentre en equilibrio termodinámico se caracteriza mediante un conjunto de variables. Estas variables se dividen en dos grupos: uno para definir los flujos másicos y molares, y otro para definir el estado termodinámico:

Tabla 3-4: Variables en una Corriente Fluida

Flujo	n_i, $1 \leq i \leq N_c$	Flujo molar de cada componente
	m_i, $1 \leq i \leq N_c$	Flujo másico de cada componente
	m	Flujo másico total
	n	Flujo molar total
Estado Termodinámico	y_i, $1 \leq i \leq N_c$	Fracción molar de cada componente
	x_i, $1 \leq i \leq N_c$	Fracción másica de cada componente
	T	Temperatura
	P	Presión
	ρ	Densidad
	H	Entalpía
	S	Entropía
	G	Energía libre de Gibbs

Es importante destacar que, en el caso de equilibrio termodinámico, todas estas variables no son independientes, ya que la corriente fluida tiene solo $N_c + 2$ grados de libertad. En consecuencia, es posible definir por completo una corriente fluida en equilibrio termodinámico empleando $N_c + 2$ variables independientes. Existen diversas selecciones posibles para las variables independientes, siendo las más comunes aquellas presentadas en la Tabla 3-5. Dichas selecciones incluyen la presión, la entalpía, así como los flujos molares o másicos de cada componente, o alternativamente, las fracciones molares o másicas para $N_c - 1$ componentes y el flujo molar o másico total.

Tabla 3-5: Posibles Selecciones de Variables Independientes Corriente Fluida

Selección 1	Selección 2	Selección 3	Selección 4
P	P	P	P
H	H	H	H
m_i, $1 \leq i \leq N_c$	n_i, $1 \leq i \leq N_c$	m	n
		x_i, $2 \leq i \leq N_c$	y_i, $2 \leq i \leq N_c$

El resto de variables de la corriente se puede calcular a partir de las variables independientes seleccionadas. Por ejemplo, en el caso de la selección 3, el resto de variables independientes pueden obtenerse con las ecuaciones siguientes:

$$x_1 = 1 - \sum_{k=2}^{N_c} x_k \tag{3-9}$$

$$m_i = m\, x_i \qquad \forall i \in \{1,2,\ldots.,N_c\} \tag{3-10}$$

$$n_i = m_i / MW_i \qquad \forall i \in \{1,2,\ldots.,N_c\} \tag{3-11}$$

$$y_i = \frac{(x_i/MW_i)}{\sum_{k=1}^{N_c}(x_k/MW_k)} \qquad \forall i \in \{1,2,\ldots,N_c\} \tag{3-12}$$

$$T = T(P,H,\boldsymbol{x}) \tag{3-13}$$

$$\rho = \rho(P,H,\boldsymbol{x}) \tag{3-14}$$

$$S = S(P,H,\boldsymbol{x}) \tag{3-15}$$

$$G = G(P,H,\boldsymbol{x}) \tag{3-16}$$

donde MW_i representa la masa molecular del i-ésimo constituyente químico, y las funciones $T(), \rho(), S()$ y $G()$ son funciones que proporcionan el estado termodinámico del fluido en función de la presión, la entalpía y la composición.

En el caso de corrientes que representan energía mecánica transmitida por ejes, se utilizan las cuatro variables siguientes para caracterizar la corriente:

Tabla 3-6: Variables en una Corriente Mecánica Rotacional

W	Potencia mecánica
ω	Velocidad de giro en rad/s
N	Velocidad de giro en rpm
τ	Torque

En el caso de las corrientes de energía mecánica rotatoria, solo dos de las variables son independientes, pues se tienen las dos relaciones siguientes:

$$\omega = \frac{2\,\pi}{60}\,N \tag{3-17}$$

$$W = \tau\,\omega \tag{3-18}$$

En el caso de las corrientes de flujo información y de ciertos tipos de energía, la corriente se caracteriza mediante una única variable, que es, por tanto, independiente.

Es común que las herramientas de análisis estacionario basadas en ecuaciones y que emplean corrientes como elementos de conexión, consideren como variables de rasgadura todas las variables independientes de las corrientes. Esta opción resulta altamente ventajosa, ya que permite independizar totalmente la formulación de los componentes.

Al emplear esta aproximación de considerar las variables independientes de las corrientes como variables de rasgadura, no se debe explicitar el cálculo de ninguna de estas variables. Cualquier ecuación que relacione variables independientes de las corrientes

entre sí debe ser considerada en forma implícita (ecuación de rasgadura), y no se debe intentar despejar en ellas ninguna de las variables consideradas como independientes.

Con respecto a las variables internas del componente, es decir, aquellas no asociadas a ninguna corriente, es posible que existan ecuaciones asociadas que permitan calcularlas de manera explícita en función de las variables de las corrientes, así como de otras variables internas previamente calculadas. Si esto es el caso, dichas variables se calcularán explícitamente. No obstante, en situaciones contrarias, el desarrollador deberá analizar los lazos algebraicos internos al componente y proponer un esquema de rasgadura. Esto implica añadir algunas de las variables internas al conjunto de incógnitas o variables de rasgadura y considerar de manera implícita algunas de las ecuaciones que relacionan estas variables internas.

Para ilustrar lo indicado consideremos el caso del compresor mostrado en la Figura 3-6, que tiene dos corrientes de flujo de gas y dos corrientes de energía mecánica que representan cortes en el eje del compresor.

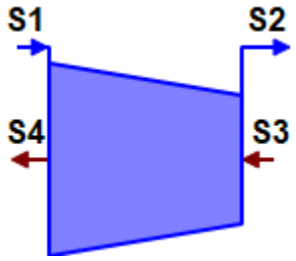


Figura 3-6: Símbolo de un Componente Compresor con Corrientes

A continuación, se indican las ecuaciones que se utilizan para representar el comportamiento del compresor, escribiendo en forma implícita todas aquellas que son de rasgadura.

La ecuación de conservación de la masa total:

$$m_{S1} - \quad m_{S2} = 0 \tag{3-19}$$

La ecuaciones de conservación de la masa de $N_c - 1$ componentes:

$$x_{S1,i} - \quad x_{S2,i} = 0 \qquad \forall i \ \in \{2, \ldots., N_c\} \tag{3-20}$$

La ecuación de la relación de compresión:

$$\pi_c \, P_{S1} - \quad P_{S2} = 0 \tag{3-21}$$

La ecuación del rendimiento isentrópico:

$$\eta_c (H_{S2} - H_{S1}) - \left(H_{S2,ideal} - H_{S1}\right) = 0 \tag{3-22}$$

donde η_c es el rendimiento isentrópico y $H_{S2,ideal}$ es la entalpía ideal de salida, esto es la entalpía si el proceso de compresión fuese isentrópico que viene dada por:

$$H_{S2,ideal} = H(P_{S2}, S_{S1}, x_{S2}) \tag{3-23}$$

donde a su vez S_{S1} es la entropía en el puerto S1 que es una de las variables dependientes del puerto. Es importante notar que tanto, S_{S1}, como $H_{S2,ideal}$ son variables calculables explícitamente a partir de las variables independientes de los puertos.

La ecuación de la energía puede expresarse como:

$$W_{S3} - W_{.S4} - m_{S1} \cdot (H_{S2} - H_{S1}) = 0 \tag{3-24}$$

siendo W_{S3} la potencia mecánica que entra por el puerto S3 y W_{S4} la potencia mecánica que sale por S4.

La continuidad mecánica del material del eje obliga a que las vueltas en ambos puertos mecánicos sean las mismas:

$$N_{S3} - N_{.S4} = 0 \tag{3-25}$$

Por último, se requieren dos ecuaciones para proporcionar la relación de compresión π_c y el rendimiento isentrópico η_c. Existen dos opciones para determinar el valor de estas variables internas del componente dependiendo del propósito del cálculo. En el diseño termodinámico del ciclo, la relación de compresión y el rendimiento isentrópico se fijan como datos de diseño, $\pi_{c,design}$ y $\eta_{c,design}$. En cambio, si se desea analizar el funcionamiento del compresor en condiciones de funcionamiento diferentes a las de diseño se utilizan las ecuaciones constitutivas (3-12) y (3-13), obtenidas mediante riguroso análisis dimensional. Estas ecuaciones, incorporan las funciones F_1 y F_2 que comúnmente se proporcionan en forma de un mapa, definido mediante dos tablas bidimensionales, a una presión y temperatura de referencia, denotadas como P_{REF} y T_{REF}. Un punto a resaltar es que estas dos variables internas, π_c y η_c , son calculables de forma explícita en función de las variables en las corrientes de entrada y salida del compresor, no hay por tanto lazos algebraicos internos del componente.

Diseño	$\pi_c = \pi_{c,design}$	(3-26)
	$\eta_c = \eta_{c,design}$	(3-27)
Fuera de diseño	$\pi_c = F_1\left(\frac{m_{S1}\sqrt{T_{S1}/T_{REF}}}{P_{S1}/P_{REF}}, \frac{N_{S3}}{\sqrt{T_{S1}/T_{REF}}}\right)$	(3-28)
	$\eta_c = F_2\left(\frac{m_{S1}\sqrt{T_{S1}/T_{REF}}}{P_{S1}/P_{REF}}, \frac{N_{S3}}{\sqrt{T_{S1}/T_{REF}}}\right)$	(3-29)

Obsérvese que la selección como desconocidas de todas las variables independientes de las corrientes, aunque incrementa el tamaño del sistema de ecuaciones permite independizar el análisis de los lazos algebraicos internos de los componentes. Es posible pre-analizar, las ecuaciones internas del componente para detectar lazos algebraicos internos, sin tener que formar el sistema de ecuaciones aplanado de todo el modelo.

Otra ventaja de seleccionar como variables desconocidas todas las variables independientes de las corrientes, es que conduce a una estructura muy dispersa del sistema de ecuaciones, los residuos de las ecuaciones implícitas que introduce un componente solo dependen de las variables independientes de las corrientes del componente y de las incógnitas implícitas internas. Es posible programar un módulo para el componente que calcule los residuos de las ecuaciones; así como el Jacobiano de dichas ecuaciones.

La aproximación orientada a ecuaciones puede dar lugar a sistemas de ecuaciones de gran tamaño, tanto si se usa un conexionado por nodos como si se usa un conexionado por corrientes. De forma que el uso de la aproximación orientada a ecuaciones suele ir siempre unida al uso de algoritmos de solución para ecuaciones dispersas, que reducen la memoria requerida y los tiempos de cálculo de forma muy significativa.

3.1.5. Ecuaciones Condicionales

En el cálculo de estados estacionarios, es común tener que emplear ecuaciones condicionales, es decir, considerar diferentes ecuaciones según la ubicación de la solución. Ejemplos típicos que demandan el uso de ecuaciones condicionales incluyen la presencia de cambios de fase en el fluido de trabajo, variaciones en el régimen del flujo (como la transición de laminar a turbulento o de subsónico a crítico), la existencia de componentes con comportamientos discontinuos (como válvulas anti-retorno) y la necesidad de imponer límites en la operación del sistema (como presiones máximas, temperaturas máximas y revoluciones máximas de la turbo-maquinaria) que no deben ser superadas.

Un sistema con ecuaciones condicionales se puede representar de la siguiente forma (Grossmann & Turkay, 1996; Zaher, 1995):

$$\boldsymbol{h}(\boldsymbol{x}) = 0$$

$$\bigvee_{i \in D_k} \begin{bmatrix} \boldsymbol{r}_{jk}^{i}(\boldsymbol{x}) = 0 \\ \boldsymbol{g}_{lk}^{i}(\boldsymbol{x}) \leq 0 \end{bmatrix} k \in K \qquad \begin{matrix} \forall\, j \in [1 \ldots \beta_k] \\ \forall\, l \in [1 \ldots \gamma_k] \end{matrix} \tag{3-30}$$

$$\boldsymbol{x} \in \mathbb{R}^n$$

donde:

$\boldsymbol{h}(\boldsymbol{x})$ es el vector con las ecuaciones invariantes del sistema, las cuales están definidas sobre toda la región de factibilidad, se supone de dimensión m.

K representa el conjunto de todas las disyunciones: el índice i se utiliza para indicar el i-ésimo término disyuntivo en cada una de las disyunciones D_k .

$\boldsymbol{r}^i_k(\boldsymbol{x})$ son los vectores con las ecuaciones variantes del sistema, con dimensión β_k para todo i en D_k .

$\boldsymbol{g}^i_k(\boldsymbol{x})$ son los vectores con las desigualdades que definen el dominio de validez de cada una de las variantes del conjunto de ecuaciones $\boldsymbol{r}^i_k(x)$, con dimensión γ_k para todo i en D_k . De esta manera, cada uno de los conjuntos de ecuaciones variantes está confinado a una subregión de la región de factibilidad.

La solución del sistema consistirá de un vector $\hat{\boldsymbol{x}}$ que satisfaga el conjunto de las ecuaciones invariantes y justamente uno de los conjuntos de ecuaciones variantes para cada una de las disyunciones, siempre y cuando se cumpla el conjunto correspondiente de desigualdades.

Por lo general, solo se consideran modelos condicionales en los que la solución mantiene continuidad al atravesar los límites establecidos por las desigualdades. En estos modelos, la continuidad se define de la siguiente manera: si un punto en un límite cumple con las ecuaciones de una subregión cercana al límite, también debe cumplir con las ecuaciones de todas las subregiones adyacentes para asegurar la continuidad en el modelo condicional.

A pesar de la demanda de continuidad, es importante señalar que la derivada primera es casi siempre discontinua en los límites. Esto impide que los métodos para resolución de sistemas de ecuaciones algebraicas del tipo Newton-Raphson, sean estrictamente aplicables a las ecuaciones condicionales, dado que estos métodos se fundamentan en funciones continuas y diferenciables. En ocasiones pueden lograr converger a la solución correcta, pero también es posible que la solución salte o fluctúe de manera constante entre diferentes subregiones sin lograr la convergencia deseada.

Zaher (1995) propuso un algoritmo de cruce de límites (boundary crossing algorithm) para la solución de problemas condicionales en los que existe continuidad de la solución al cruzar los límites. Este algoritmo utiliza el método de Newton-Raphson si el incremento en la solución no la desplaza fuera de la subregión asumida, y recurre a un método de minimización cuando la solución se aproxima a los límites de las subregiones.

Rico-Ramirez (1998) llevó a cabo un análisis del modelado condicional con el propósito de su implementación en ASCEND. En su estudio, en primer lugar, presenta los tipos de sentencias de un lenguaje de simulación necesarias para definir modelos condicionales. Luego, plantea una aproximación para llevar a cabo el análisis estructural de estos modelos. Finalmente, explora la implementación y evaluación de técnicas de solución. Específicamente, describe en detalle la implementación de un algoritmo de cruce de límites. También propone representar los modelos condicionales como una extensión de

los problemas de complementariedad y explora técnicas numéricas para resolver estos problemas de complementariedad extendidos.

Sin embargo, las técnicas de complementariedad que se describen en dicho trabajo, aunque prometedoras, adolecen de limitaciones. La extensión propuesta de la formulación de complementariedad es completamente particular y no permite utilizar los códigos y técnicas numéricas desarrollados para problemas estándar de complementariedad, como el problema de complementariedad mixta (MCP, mixed complementarity problem). Según indica el autor, todavía falta descubrir una reformulación sistemática del problema de complementariedad extendido propuesto en un problema de Complementariedad Mixta (MCP). Esta reformulación sistemática seria muy deseable porque haría viable la aplicación de una amplia cantidad de códigos y técnicas numéricas existentes, por lo que serían deseables trabajos futuros en esa dirección.

Ahora bien, a pesar de que no es posible una reformulación sistemática de un modelo condicional como un problema de complementariedad estándar, en la práctica, existen muchas situaciones en que dicha reformulación es posible. Por ejemplo, es viable expresar las limitaciones operativas de un turborreactor como un problema generalizado de complementariedad no lineal (GNPC). Dicho problema involucra dos funciones continuamente diferenciables que deben cumplir:

$$\begin{gathered} F(\boldsymbol{x}) \cdot G(\boldsymbol{x}) \;=\; 0 \\ F(\boldsymbol{x}) \;>=\; 0 \;\wedge\; G(\boldsymbol{x}) \;>=\; 0 \end{gathered} \tag{3-31}$$

Sea un turborreactor, en el cual el flujo de combustible a inyectar Wf viene determinado por dos límites que no deben ser excedidos, la temperatura final de combustión, variable denominada T4t, que debe ser menor de 1650 K y la velocidad de giro del eje del compresor y turbina, variable denominada NH que debe ser menor de 20000 rpm. La formulación de estas limitaciones como un problema complementario generalizado se expresa de la siguiente forma:

$$\begin{gathered} F1(Wf) \;=\; 1650 - T4t \qquad F2(Wf) \;=\; 20000. - NH \\ F1 \cdot F2 \;=\; 0 \\ F1 \;>=\; 0 \;\wedge\; F2 \;>=\; 0 \end{gathered} \tag{3-32}$$

Los problemas de complementariedad estándar tienen la ventaja de admitir una técnica de solución muy sencilla, que implica suavizar la formulación y emplear un resolvedor Newton-Raphson genérico. Dado que se hará aplicación de esta técnica en capítulos posteriores, se ha decidido incluir un Apéndice B describiendo los problemas estándar de complementariedad y su solución mediante el método Newton-Raphson suavizado.

3.1.6. Resolución de Problemas de Diseño

Generalmente, las herramientas de simulación estacionaria con orientación a ecuaciones dan soporte a la resolución de problemas de diseño. En los problemas de diseño, se desea fijar o imponer límites en el valor de ciertas variables que son resultados de la simulación, para así calcular datos que determinan las dimensiones y características de los componentes.

En esencia, existen dos enfoques o técnicas que las herramientas de simulación de estos sistemas pueden ofrecer para abordar problemas de diseño:

- *La habilidad de cambiar entre dos formulaciones de los componentes*: una para el diseño y otra para situaciones *fuera de diseño*.
- *La introducción de ecuaciones adicionales*, que representan especificaciones de diseño. Estas ecuaciones imponen valores precisos o límites superiores/inferiores en los resultados para un punto de diseño (diseño monopunto) o para varios puntos de diseño (diseño multipunto).

Formulación Diseño y Fuera de Diseño

Numerosas herramientas para la simulación de ciclos termodinámicos ofrecen dos modos de funcionamiento de los componentes, cada uno con formulaciones diferentes: un *modo de diseño* y otro modo para situaciones *fuera de diseño*. El ejemplo presentado en la sección 3.1.3, referente a la formulación de un compresor centrífugo, ilustra ambas formulaciones de manera explícita.

El *modo de diseño* se emplea para la definición del ciclo, es decir, para determinar el estado termodinámico del fluido de trabajo a lo largo del ciclo en un punto de operación conocido como el de diseño. En este modo, se calcula el estado termodinámico de las corrientes de salida de los componentes a partir del estado termodinámico de las corrientes de entrada. Esto se logra especificando o fijando ciertos parámetros de diseño que establecen relaciones sencillas entre las variables termodinámicas en las entradas y salidas de cada componente.

En el *modo de diseño*, el estado termodinámico en las salidas no depende de los flujos másicos, ya que los componentes no buscan determinar valores absolutos de flujo, sino establecer relaciones o ratios entre los flujos en las corrientes. Por ejemplo, en el caso de un compresor, los parámetros de diseño típicos incluyen la relación de compresión y el rendimiento isentrópico o politrópico. Es relevante señalar que, en este modo, el flujo a través del compresor queda indeterminado y debe ser calculado mediante ecuaciones externas al compresor.

Además de los datos de diseño de los componentes, la especificación o definición del punto de diseño debe incorporar las condiciones ambientales para las cuales se diseña el ciclo.

En el caso de sistemas abiertos que operan con aire, como las turbinas de gas, estas condiciones ambientales incluyen la presión, temperatura y humedad del aire ambiente, así como la velocidad relativa del aire en el caso de un sistema instalado en un vehículo. En los ciclos cerrados, las condiciones ambientales consisten básicamente en las temperaturas de los focos frío y caliente.

El *modo de diseño* siempre requiere al menos una ecuación adicional a las ecuaciones de los componentes para determinar el valor absoluto de los flujos por el ciclo. Esta ecuación adicional puede consistir en la fijación de la potencia deseada del ciclo, o la potencia calorífica aportada al ciclo, o el flujo másico en algún punto del ciclo, o la fuerza de empuje requerida, ésta última en el caso de un sistema de propulsión.

La importancia del *modo de diseño* radica en que permite establecer un punto de operación nominal de los equipos, lo cual resulta fundamental tanto para su adquisición, en el caso de comprarlos a proveedores o fabricantes externos, como para emprender su proceso de diseño en caso de un desarrollo interno. Se presupone que una vez se ha determinado el punto de diseño de un componente, es posible predecir cómo funcionará en condiciones de operación diferentes a las de diseño.

El *modo fuera de diseño* contesta a la pregunta de cómo se comporta un determinado sistema diseñado para un cierto punto de operación cuando opera en unas condiciones diferentes de las de diseño. Las condiciones de operación pueden variar con respecto a las de diseño debido a discrepancias en las condiciones ambientales en comparación con las previstas en el diseño, o debido a unas demandas de producción (que pueden incluir potencia eléctrica, potencia mecánica, empuje, calor, entre otros) que difieren de las establecidas para diseño, o por una combinación de ambos motivos.

En la mayoría de los casos, el modelo matemático *fuera de diseño* de un ciclo termodinámico tiene un número de grados de libertad. Dicho número de grados de libertad es igual al número de variables del modelo menos el número ecuaciones proporcionadas por los componentes del sistema con su formulación *fuera de diseño*. Por ejemplo, el modelo de una turbina de gas tiene al menos un grado de libertad, que corresponde a la cantidad de combustible inyectado, aunque puede haber grados de libertad adicionales, como la posición angular de los álabes guía en ciertos tipos de compresores y el área de salida en caso de toberas de geometría variable. Para llevar a cabo simulaciones *fuera de diseño*, el usuario deberá especificar un número de ecuaciones adicionales igual al de grados de libertad. En realidad, estas ecuaciones adicionales proporcionan una representación simplificada del sistema de control y pueden ser innecesarias en caso de que el control se represente de forma explícita en el modelo.

La elección entre el uso de la formulación para "*diseño*" o de la formulación para "*fuera de diseño*" debería ser una decisión individual para cada componente. Por lo general, en un sistema, todos los componentes se diseñan para funcionar en un mismo punto de

operación (diseño monopunto). El proceso de diseño de un ciclo termodinámico generalmente comienza con un cálculo en el que todos los componentes operan en *modo de diseño*, con el objetivo de obtener su dimensionamiento. Seguidamente, se procede a evaluar el comportamiento del sistema en condiciones de operación diferentes de las de diseño.

Sin embargo, no siempre es necesario seguir esta secuencia. Es bastante común reemplazar un componente en un sistema existente para mejorar su comportamiento, ya sea para conseguir un mayor rendimiento o una mayor potencia. En este escenario, los demás componentes ya están diseñados y, por ende, deben utilizar la formulación para "*fuera de diseño*", mientras que solo el nuevo componente debe emplear la formulación de "*diseño*" con el fin de permitir su dimensionamiento.

Introducción de Ecuaciones Adicionales con Especificaciones de Diseño

La aproximación de tener dos formulaciones para los componentes, una de *diseño* y otra para *fuera de diseño*, pudiendo elegir entre ambas formulaciones resulta sencilla, pero presenta limitaciones. Por un lado, solo se permite fijar en el diseño los parámetros de diseño predefinidos en los componentes. Por ejemplo, en el caso del compresor solo se permite fijar la relación de presiones y la eficiencia; cuando podría ser deseable fijar otros parámetros como la presión en una estación del ciclo situada fuera del compresor, o el trabajo específico (trabajo por unidad de masa de aire) proporcionado por el compresor. Por otro lado, esta aproximación obliga a diseñar todos los componentes en un mismo punto de diseño.

La manera más versátil en que las herramientas de simulación pueden soportar o ayudar el diseño de sistemas radica en utilizar formulaciones *fuera de diseño* que deben representar no solamente el comportamiento del componente fuera de diseño, también deben representar el efecto de unos datos que definen las dimensiones del componente sobre su comportamiento. Estando los componentes formulados de esta manera, el diseño se especifica introduciendo ecuaciones suplementarias, que se conocen como especificaciones de diseño. Estas ecuaciones posibilitan determinar el valor de ciertos datos que definen las dimensiones de los componentes y que deben ser declarados como incógnitas del problema de diseño.

Los datos p de un modelo se pueden clasificar en varias categorías.

p_{OpC} datos de las condiciones de operación, como condiciones ambientales y velocidad

p_{Cmps} datos de dimensionamiento de los componentes, que intervienen o aparecen solo en la formulación fuera de diseño de los componentes

p_{CnL} datos de control y limitaciones

Es importante destacar que los datos de diseño p_{CmpD} solo son necesarios cuando se utiliza la formulación de diseño del componente, y los datos de dimensionamiento p_{CmpS} cuando se utiliza la formulación *fuera de diseño*.

Los datos que definen las condiciones de operación del sistema, tales como presión atmosférica, temperatura del sumidero de calor, velocidad de vuelo en el caso de un aerorreactor, dichos datos los designaremos como p_{OpC}. Luego se tienen datos que son específicos de los componentes, hay dos subcategorías, los de diseño p_{cmpD} y los datos de dimensionamiento p_{cmpS} que requiere la formulación *fuera de diseño* los que se desea calcular. También, se tienen datos que definen puntos de referencia y limitaciones, que bien representan elementos control o bien limitaciones tecnológicas p_{CnL}.

El objetivo del diseño es determinar los datos que definen las dimensiones del componente p_{CmpS}. En el diseño monopunto, se fijan las condiciones de operación de diseño y se agregan al modelo ecuaciones que representan las especificaciones de diseño. El número de especificaciones de diseño debe ser igual al número de grados de libertad del modelo más el número de parámetros a determinar.

La Figura 3-7 muestra un esquema del sistema de ecuaciones e incógnitas para el diseño monopunto. En dicha figura se representan, las condiciones de operación correspondientes al punto de diseño del ciclo p_{OpC}. (que están impuestas), el sistema de ecuaciones constituido por las ecuaciones propias del modelo más las ecuaciones que representan especificaciones de diseño, así como las incógnitas, que incluyen las correspondientes al modelo fuera de diseño, que se designan como x, y los datos de dimensionado, p_{CmpS}, los cuales son incógnitas en el problema de diseño.

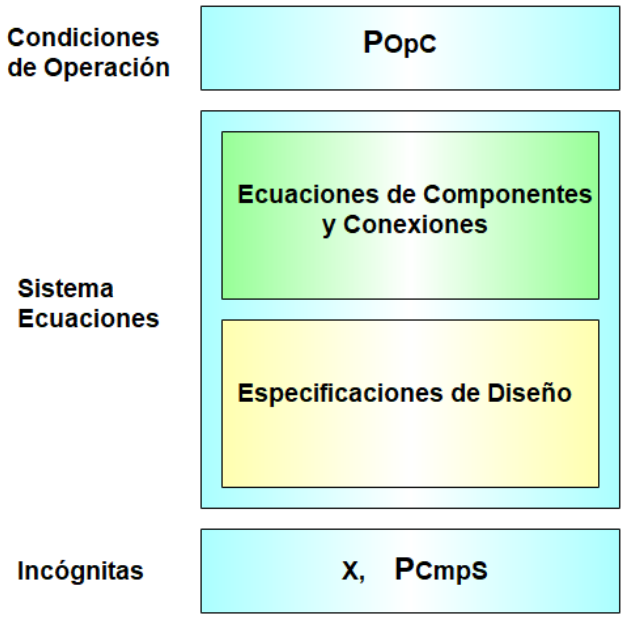


Figura 3-7: Esquema de las Ecuaciones e Incógnitas del Diseño Monopunto

El caso del diseño multipunto requiere una mayor complejidad. Considera directamente un número *k* de puntos de diseño con diferentes condiciones de operación y para cada uno de los puntos se introducen unas especificaciones de diseño (ecuaciones adicionales). El número total de especificaciones adicionales debe ser igual al número de grados de libertad del modelo por el número de puntos de diseño más el número total de parámetros de dimensionamiento a determinar. La Figura 3-8 ilustra el esquema de ecuaciones del diseño multipunto

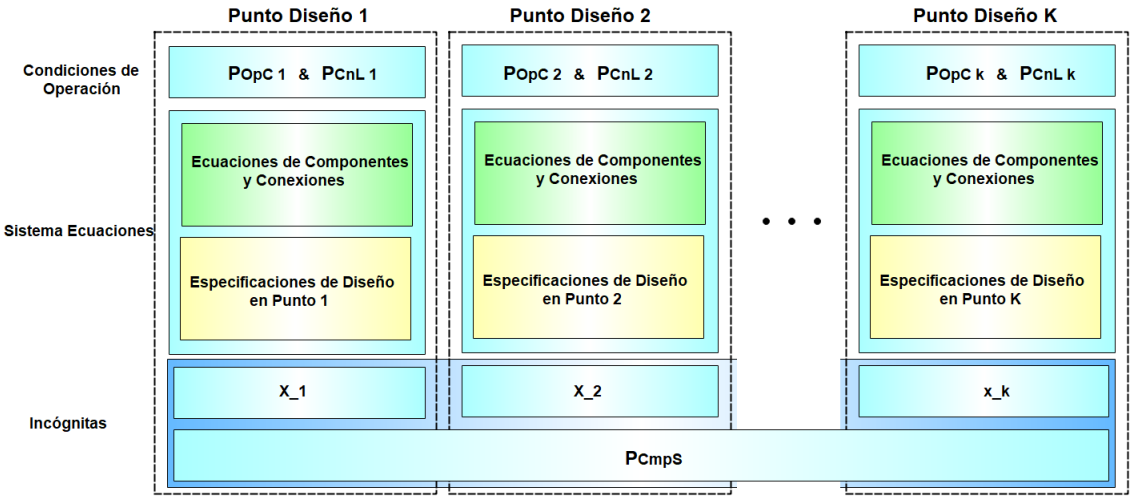


Figura 3-8: Esquema de las Ecuaciones e Incógnitas del Diseño Multipunto

3.2. Calculo de Estacionarios en Herramientas de MSOO

Después de haber explorado las características principales de las herramientas convencionales de simulación estacionaria en la § 3.1. Esta sección describe la forma en que las herramientas de MSOO han abordado el cálculo de estacionarios.

Las herramientas de MSOO siguen la aproximación orientada a ecuaciones dado que el paradigma subyacente implica la construcción de un sistema de ecuaciones representativo de todo el modelo. Este sistema de ecuaciones se obtiene ensamblando las ecuaciones de cada módulo o componente, junto con las ecuaciones de ligadura que establecen las conexiones entre componentes.

El tipo de sistema de ecuaciones en el que se basan las herramientas de MSOO es el sistema de ecuaciones algebraico-diferenciales híbrido. El término “híbrido” implica que el sistema de ecuaciones algebraico-diferenciales puede cambiar de forma discreta en ciertos momentos de tiempo, denominados eventos. Un sistema de ecuaciones algebraico-diferenciales puede representarse como:

$$\boldsymbol{F}(\boldsymbol{x}', \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p}, t\ ) = \boldsymbol{0} \tag{3-33}$$

$$\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p}, t\ ) = \mathbf{0} \tag{3-34}$$

donde $\boldsymbol{x}'$ son las derivadas de las variables de estado $\boldsymbol{x}, \boldsymbol{z}$ son las variables algebraicas, $\boldsymbol{p}$ son las variables discretas, t es el tiempo, $\boldsymbol{F}$ es un vector de ecuaciones diferenciales y $\boldsymbol{G}$ es un vector de ecuaciones algebraicas.

Las herramientas de MSOO ofrecen la capacidad de cálculo de estacionarios, básicamente imponiendo que en las derivadas de las variables de estado sean nulas, como se indica a continuación.

$$\boldsymbol{x}' = \mathbf{0} \rightarrow \boldsymbol{F}(\mathbf{0}, \boldsymbol{x}, \boldsymbol{z}, \boldsymbol{q}, t) = \mathbf{0} \tag{3-35}$$

$$\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{z}, \boldsymbol{p}, \boldsymbol{t}\ ) = \mathbf{0} \tag{3-36}$$

El sistema de ecuaciones (3-35) y (3-36) permite, en teoría, calcular los valores de las variables de estado $\boldsymbol{x}$ y las variables algebraicas $\boldsymbol{z}$ para los cuales se anulan las derivadas de las variables de estado ($\boldsymbol{x}' = \mathbf{0}$).

Es importante destacar que esta forma de cálculo de estacionarios está claramente diseñada con el propósito de obtener el punto inicial para una simulación transitoria. No está concebida para proporcionar las capacidades de una herramienta puramente estacionaria.

Un primer problema de esta forma de cálculo de estacionarios radica en que en los modelos de ciertos sistemas suele ocurrir que no todas las derivadas se anulan durante lo que se llama un cálculo estacionario. Por ejemplo, los modelos de sistemas de propulsión de aeronaves y otros vehículos suelen representar de forma explícita los tanques de combustible. Una de las ecuaciones dinámicas de los tanques es la de conservación de masa, la cual indica que la derivada de la masa de combustible en el tanque es igual a menos el flujo másico que sale del tanque. Lógicamente, un estacionario de la masa de combustible (derivada nula de la masa) en el tanque solo se puede producir con el sistema de propulsión completamente parado. El cálculo de estacionarios en sistemas de propulsión (con el sistema no parado) requiere congelar o fijar la masa en los tanques.

Otro caso típico de cálculo estacionario en el que no se anulan todas las derivadas, es la determinación del estado de un vehículo en una trayectoria rectilínea a velocidad constante. En dicho caso, la posición del vehículo está continuamente cambiando, y la derivada de la posición (esto es, la velocidad) es constante pero no nula.

Después de estas ideas generales sobre el cálculo de estacionarios, se presenta en detalle el cálculo de estacionarios con EcosimPro y con los entornos de simulación que implementan Modelica.

3.2.1. Estacionarios en EcosimPro

En el caso de EcosimPro, existe una función de experimento, denominada **STEADY()**, que intenta calcular el valor del vector de variables de estado $\boldsymbol{x}$ que anula las derivadas en el sistema de ecuaciones (3-35) y (3-36).

Para lidiar con el problema de derivadas que no se anulan durante el estacionario, EcosimPro proporciona otra función de experimento, la función **FREEZE**, cuya sintaxis:

FREEZE(*lista_de_variables*)

donde ***lista_de_variables*** es una lista de nombres de variables de estado separados por comas. El valor inicial de todas las variables de la lista permanece constante durante el cálculo del estacionario.

El problema de la sentencia **FREEZE** es que el uso eficaz de la misma requiere un buen conocimiento de las ecuaciones que se implementan en el modelo, lo cual va contra el principio de Encapsulamiento de la programación orientada a objeto. En los casos en que un modelo incluye variables que no se anulan durante el estacionario, se suele obtener mensajes de que el Jacobiano es singular, pero no se producen mensajes indicando que variables de estado deben ser congeladas.

Para ilustrar el funcionamiento del **STEADY()** en EcosimPro, considérese el siguiente modelo:

```
COMPONENT FirstOrder
   DATA
      REAL tau = 1    "Time constant"
   DECLS
      BOUND REAL x = 1
      REAL y = 0
   CONTINUOUS
      y' = (x-y) / tau
END COMPONENT
```

Dicho modelo implementa la siguiente ecuación

$$\frac{dy}{dt} = \frac{(x-y)}{\tau} \tag{3-37}$$

donde y es la variable de estado, x es una variable de contorno cuyo valor se proporciona como una función del tiempo en el experimento $(x = x(t))$ y τ es un dato que permanece constante durante el tiempo de simulación.

Una característica de la función **STEADY()** es que puede llamarse en cualquier tiempo de simulación. Es posible, llamar **STEADY()** antes del comienzo de una integración y también es posible llamar dicha función después de una integración. En caso de que en el

experimento, se especifique el cálculo de un estacionario llamando a la función **`STEADY()`** en el tiempo t_1, la solución será $y = x(t_1)$.

En el caso de EcosimPro, los componentes pueden incorporar un bloque **`INIT.`** Dentro de este bloque, se introduce código cuya ejecución es completamente secuencial y se lleva a cabo al inicio de la simulación. A diferencia de Modelica, donde en los bloques equivalentes (el bloque **`"initial equation"`** y el bloque **`"initial algorithm`**") se introducen ecuaciones que deben resolverse de forma simultánea, en EcosimPro es factible reasignar el valor de una misma variable tanto en bloques **`INIT`** de distintos componentes como incluso dentro del bloque **`INIT`** de un único componente.

Durante el procesamiento de un modelo de EcosimPro, los bloques **`INIT`** de todos los componentes se juntan en un extenso bloque de código secuencial dentro de una función C++, la cual se invoca al principio de la simulación y también en otros momentos que el usuario puede especificar mediante una llamada a la función del sistema **`EXEC_INIT()`**.

Cuando se juntan los bloques **`INIT`** de los componentes, las sentencias dentro de cada bloque se mantienen agrupadas, y el orden en que se disponen los bloques de los diferentes componentes de una agregación es indefinido. Ahora bien, el usuario puede especificar una prioridad del bloque **`INIT`** de cada componente. Esta prioridad se indica mediante la palabra clave **`PRIORITY`** después del **`INIT,`** seguida de número entero que representa la prioridad del bloque. En caso de utilizarse prioridades, los bloques con prioridad más alta se colocarán primeros. La prioridad de defecto, en caso de no especificarse, se establece en 0.

En el caso de componentes derivados por herencia de un componente base, las sentencias en el bloque **`INIT`** de la clase derivada se posicionan después de las sentencias presentes en el bloque **`INIT`** de la clase base. Por ejemplo, considerando el componente `son` cuya codificación se muestra en el lado izquierdo de la Figura 3-9, las sentencias **`INIT`** se ordenan en la forma indicada en el lado derecho. Los valores de las variables discretas `x` e y después de la ejecución del bloque serán los que se indican también en el lado derecho.

```
COMPONENT parent
   DECLS
      DISCR REAL x = 0
      DISCR REAL y = 0
   INIT
      y = x + 1
END COMPONENT
COMPONENT son IS_A parent
   INIT
      x = y + 1
      y = x + 1

END COMPONENT
```

Ordenación de las Sentencias **INIT** del componente ***son***

```
y = x + 1
x = y + 1
y = x + 1
```

Resultados: `x = 2` `y = 3`

Figura 3-9: Ejemplo de Ordenación de las Sentencias del Bloque INIT en el Caso de Herencia

En la Figura 3-10 se muestra este mismo ejemplo, pero cambiando el orden de ejecución de los bloques **INIT**, primero el de la clase derivada y luego el de la clase base. En consecuencia, los resultados para los valores de las variables `x` e y son diferentes a los del ejemplo anterior.

```
COMPONENT parent
   DECLS
      DISCR REAL x = 0
      DISCR REAL y = 0
   INIT  PRIORITY 0
      y = x + 1
END COMPONENT

COMPONENT son IS_A parent
   INIT  PRIORITY 100
      x = y + 1
      y = x + 1
END COMPONENT
```

Ordenación de las Sentencias **INIT** del componente ***son***

```
x = y + 1
y = x + 1
y = x + 1
```

Resultados: `x = 1` `y = 2`

Figura 3-10: Ejemplo de Ordenación de las Sentencias del Bloque INIT en el Caso de Herencia y Uso de Prioridades

Según el manual del usuario de EcosimPro, el bloque **INIT** de los componentes se emplea para asignar el valor inicial de ciertas variables que requieren un valor antes de iniciar la simulación. A nivel práctico, generalmente se asignan dos tipos de variables en el bloque **INIT**: variables de estado y variables discretas.

En relación con la inicialización de ***variables de estado***, algunas de ellas tienen un valor medible y se pueden asignar directamente con un dato. Ejemplos de estas variables, cuyo valor inicial se puede considerar conocido, son la presión, la temperatura y la posición. Sin embargo, también se emplean variables de estado no medibles, como la energía interna y la entalpía de un fluido. Aunque su valor inicial no se conoce, puede calcularse mediante funciones de estado termodinámicas a partir de la presión, temperatura y composición

iniciales, que sí son conocidas. En estos casos, la llamada a la función para el cálculo de la energía o entalpía iniciales se incluye en el bloque **`INIT`**.

Por otro lado, en el bloque **`INIT`** también suelen inicializarse ***variables discretas***. Un caso especial de variable discreta es aquella que permanece constante durante la simulación y es calculable en función de datos. Por ejemplo, en el caso de las tuberías circulares, el diámetro suele ser un dato conocido. Aunque la ecuación del área de flujo ($A = \pi D^2/4$) podría introducirse como una ecuación continua, esto sería ineficiente, ya que se recalcularía el área continuamente. Por lo general, la variable que representa el área se declara como discreta y se le asigna su valor en el bloque **`INIT`**.

Como se ha mencionado previamente, la función **`EXEC_INIT()`** permite invocar la ejecución de las sentencias de los bloques **`INIT`** en cualquier momento de la simulación. Esta función no solo puede ser utilizada el experimento, sino que también puede ser invocada desde el propio modelo, incluso desde el bloque **`INIT`** de un componente específico. Esta capacidad facilita la realización de llamadas recursivas a la función de inicialización, otorgando una considerable flexibilidad y potencia al lenguaje. En los capítulos 5 y 6, se explorará cómo los bloques **`INIT`** y la función **`EXEC_INIT()`** pueden emplearse para implementar algoritmos de cálculo de estacionarios, ya sea utilizando el método orientado a ecuaciones o el método modular secuencial.

3.2.2. Estacionarios en Modelica

La metodología para calcular estacionarios difiere notablemente entre Modelica y EcosimPro. En EcosimPro, el cálculo de estacionarios se realiza desde el experimento. Por el contrario, en Modelica, los modelos tienen la capacidad de incorporar ecuaciones iniciales adicionales para la inicialización, lo que permite forzar o especificar el cálculo de estacionarios mediante estas ecuaciones iniciales adicionales.

En Modelica, las secciones **`"initial equation"`** e **`"initial algorithm`**" permiten incluir ecuaciones que establecen restricciones algebraicas puras entre los valores iniciales de las variables y, posiblemente, sus derivadas. Estas ecuaciones iniciales, junto con las ecuaciones transitorias, componen el problema de inicialización del modelo. La solución del problema de inicialización se utiliza para asignar valores coherentes a todas las variables y derivadas.

Para calcular un estacionario en Modelica, es necesario especificar ecuaciones que igualen las derivadas a cero en cualquiera de las dos secciones "**`initial`**". Este enfoque permite al desarrollador de la librería anticipar las derivadas cuya anulación no se requiere para el cálculo del estacionario. Asimismo, permite evitar trasladar esta responsabilidad al usuario de la librería al definir los experimentos.

Otra capacidad de Modelica útil para el cálculo de estacionarios, es que las variables tienen el atributo **`fixed`** de tipo booleano, con dicho atributo es posible especificar que variables son las que se desean calcular en el problema de inicialización.

Es relevante destacar que ambas secciones iniciales en Modelica incorporan ecuaciones. A pesar de que pueda parecer que el **`"initial algorithm"`** de Modelica es muy similar al bloque **`INIT`** de EcosimPro, en realidad, el **`"initial algorithm"`** se comporta como un bloque de ecuaciones en Modelica.

Modelica se adhiere rigurosamente a un enfoque basado exclusivamente en ecuaciones. Según su especificación (Modelica Association, 2023):

> *Modelica se basa en el principio de flujo de datos síncrono y la regla de asignación única, que se definen de la siguiente manera:*
>
> 1) *Las variables de tiempo discreto mantienen sus valores hasta que estas variables sean cambiadas explícitamente. Las variables que aparecen derivadas se suponen continuas, excepto cuando se desencadena "reinit". Los valores de las variables se pueden acceder en cualquier instante de tiempo durante la integración continua y en instantes de eventos.*
> 2) *En cada instante de tiempo, durante la integración continua y en instantes de eventos, las ecuaciones expresan relaciones entre variables que deben cumplirse simultáneamente.*
> 3) *La computación y la comunicación en un instante de evento no llevan tiempo.*
> 4) *Debe existir una correspondencia perfecta entre las variables y las ecuaciones después del aplanamiento del modelo, donde una variable solo puede emparejarse con las ecuaciones que pueden contribuir a resolverla (regla de emparejamiento perfecto, anteriormente llamada regla de asignación única)*

El enfoque exclusivamente basado en ecuaciones de Modelica impide la asignación de una misma variable en las secciones **`"initial equation"`** de dos componentes diferentes. Esta restricción contrasta con la flexibilidad presente en EcosimPro, donde se permite la reasignación de variables en los bloques **`INIT`** de los componentes. de los componentes. Modelica carece de la potencia y flexibilidad que ofrece la programación secuencial. No obstante, es relevante señalar que algunos estudios han sugerido aumentar la flexibilidad en Modelica, asumiendo que no todos los eventos pueden ser síncronos, lo que permitiría el cálculo de una misma variable discreta en distintas sentencias **`when`** (Nikoukhah, 2007; Nikoukhah & Furic, 2008)

3.3. Problemas de los Estacionarios en Herramientas de MSOO

El problema de partida del cálculo estacionario en las herramientas de MSOO, ya sean basadas en Modelica o EcosimPro, radica en la suposición de que existe un modelo dinámico o transitorio y que se desea calcular un estacionario de dicho modelo dinámico o transitorio. A diferencia de una herramienta puramente estacionaria, que formula directamente un modelo estacionario sin variables de estado ni derivadas, introduciendo únicamente variables desconocidas algebraicas. La presunción de un modelo dinámico o transitorio plantea dos inconvenientes fundamentales.

En primer lugar, esta suposición obliga a que la formulación estacionaria sea idéntica a la formulación transitoria, sin aprovechar la posibilidad de soluciones analíticas existentes para el estado estacionario a nivel de componentes. Estas soluciones pueden simplificar y reducir significativamente la complejidad del problema de cálculo estacionario.

El segundo inconveniente radica en que la formulación transitoria puede requerir un procesamiento simbólico intensivo del modelo para abordar problemas de índice superior. Sin embargo, estos problemas son irrelevantes cuando solo se busca la solución estacionaria. A continuación, se explican y describen en detalle ambos problemas.

Formulación Estacionaria Igual a la Transitoria

El primer problema surge comúnmente al dividir el dominio de un componente en volúmenes de control discretos para calcular transitorios con precisión. En este caso, el problema transitorio se formula aplicando ecuaciones de conservación a cada uno de los elementos o volúmenes, generando variables de estado asociadas a cada uno de los volúmenes. Un ejemplo ilustrativo de este problema se presenta en el caso una pared con transmisión unidimensional del calor, dividida en nodos, tal y como se muestra en la Figura 3-11.

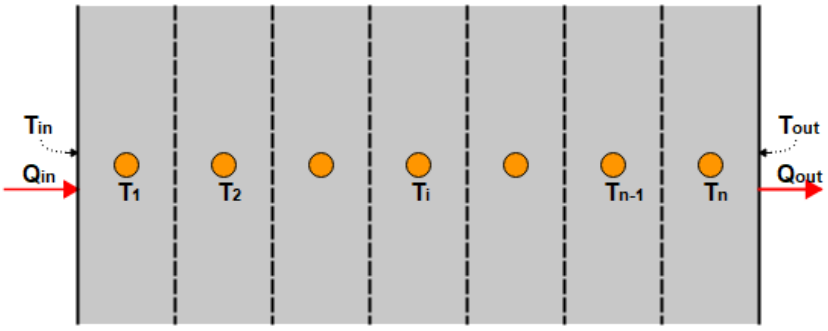


Figura 3-11: Modelo Térmico de Pared

Las ecuaciones para representar el comportamiento dinámico de los nodos de la pared son:

$$q_{in} = 2\frac{k\,A}{(l/n)}(T_{in} - T_1) \tag{3-38}$$

$$\frac{d\,T_1}{dt} = q_{in} - \frac{k\,A}{(l/n)}(T_1 - T_2) \tag{3-39}$$

$$\frac{d\,T_i}{dt} = \frac{k\,A}{(l/n)}(T_{i-1} - T_i) - \frac{k\,A}{(l/n)}(T_i - T_{i+1}) \quad \forall\, i \in 2, \dots, n-1 \tag{3-40}$$

$$q_{out} = 2\frac{k\,A}{(l/n)}(T_n - T_{out}) \tag{3-41}$$

$$\frac{d\,T_n}{dt} = \frac{k\,A}{(l/n)}(T_{n-1} - T_n) - q_{out} \tag{3-42}$$

donde:

T_i es la temperatura del nodo i-ésimo

T_{in} es la temperatura en la superficie del lado considerado como entrada

T_{out} es la temperatura en la superficie del lado considerado como salida

q_{in} es el flujo de calor en la superficie del lado considerado como entrada

q_{out} es el flujo de calor en la superficie del lado considerado como salida

k es la conductividad térmica del material de la pared

ρ es la densidad del material de la pared

c_p es el calor específico del material de la pared

A es el área transversal de la pared

L es el espesor del muro

n es el número de nodos equidistantes en que se divide la pared

Es fácil ver que en el caso de que la conductividad k sea constante, estas ecuaciones se pueden simplificar para un cáculo estacionario a las dos siguientes que relacionan solo 4 variables, los flujos de calor y temperaturas en las dos superficies :

$$q_{in} = q_{out} \tag{3-43}$$

$$q_{in} = \frac{k\,A}{L}(T_{in} - T_{out}) \tag{3-44}$$

En comparación, el sistema original tiene n+2 ecuaciones y n+4 variables (las temperaturas de los nodos anteriores, las dos temperaturas y flujos de calor en las superficies). Una herramienta convencional de estacionarios planteará directamente las ecuaciones para calcularlas temperaturas y flujos de calor en las superficies. Una vez conocidas las

temperaturas en las superficies, la inicialización de las temperaturas en los nodos interiores se puede efectuar con la siguiente formula.

$$T_i = T_{in} - \frac{\left(i - \frac{1}{2}\right)}{n} T_{out} \qquad \forall\, i \in 1, \dots, n \tag{3-45}$$

En modelos de termo-fluidos, las tuberías se suelen nodalizar a fin de tener en cuenta efectos dinámicos como retrasos térmicos y ondas de presión, siendo en ocasiones el número de nodos muy alto para calcular con precisión efectos que son muy no lineales. En estos casos, el número de variables de estado por nodo es bastante más alto, típicamente se tienen al menos 3 variables de estado por nodo: masa fluida, energía interna y flujo másico. Ahora bien, en muchas ocasiones existen soluciones analíticas para la distribución de presiones y temperaturas en la tubería, nótese que el flujo másico es siempre constante en estacionario. Por ejemplo, existe una solución analítica si el fluido es un líquido densidad constante y la transmisión de calor es despreciable, todos los programas comerciales de hidráulica hacen esta hipótesis. También existen soluciones analíticas en el caso de que el fluido sea un gas perfecto y el flujo pueda considerarse bien isotermo o bien adiabático. En todas esas situaciones es mucho más sencillo plantear un problema estacionario con ecuaciones diferentes de las del transitorio.

Sin embargo, las herramientas de MSOO no permiten prescindir de las ecuaciones nodales y obligan a tener que considerar el problema completo. En modelos complejos de redes de tuberías suele ser difícil conseguir el cálculo del estacionario, y muchas veces es necesario simular transitorios nulos durante tiempos bastante largos con el fin de conseguir un estacionario.

Puede criticarse que las soluciones analíticas indicadas, son solo aproximadas en muchos casos, porque no se cumplen de forma precisa todas las hipótesis requeridas. Pero aun en esos casos es útil plantearse una formulación simplificada del estacionario, que, aunque solo es aproximada, permita disminuir el tiempo de simulación requerido del transitorio nulo a fin de conseguir el estacionario.

Formulación Estacionaria Impactada por Problemas de Índice Superior

El segundo problema es que la utilización de una formulación transitoria introduce problemas de índice superior, los cuales son una complicación innecesaria para el usuario solo interesado en estacionario. En un cálculo puramente estacionario no existe el problema de índice superior, solo se tienen ecuaciones algebraicas. Por ejemplo, el circuito eléctrico que se muestra en la Figura 3-12 es un ejemplo de problema de índice alto, las corrientes por los tres inductores son variables de estado y tienen una ligadura, su suma debe ser nula.

Sin embargo, si consideramos ese mismo circuito en continua desaparece el problema de índice superior. En continua, un inductor se representa como un corto circuito. Es posible resolver de manera fácil el circuito en continua, sustituyendo los inductores por resistencias nulas.

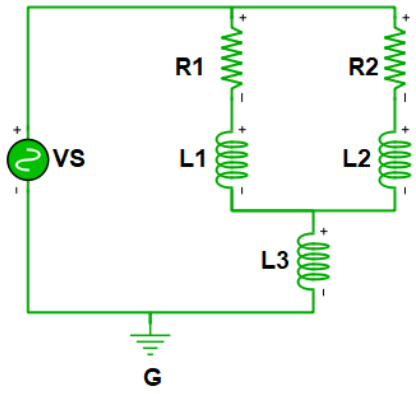


Figura 3-12: Ejemplo de Circuito Eléctrico de Índice Alto

El péndulo brinda otro ejemplo de problema de índice superior, que se desvanece o desaparece cuando consideramos el problema estacionario. Las ecuaciones que representan un péndulo en estacionario son las siguientes:

```
OMPONENT Pendulum
      DATA
            REAL m = 1       UNITS "kg"    "Pendulum mass"
            REAL L = 1       UNITS "m"     "Pendulum length"
            REAL g = 9.8     UNITS "m2/s"  "Gravity"
            REAL Fx_o = 9.8  UNITS "N"     "Initial force in x"
      DECLS
            REAL x, y
            REAL T
      CONTINUOUS
            x**2 + y**2 = L**2
            0 = Fx_o - T * x/L
            0 = -m*g - T * y/L
END COMPONENT
```

Estas ecuaciones son fácilmente resolubles, la única dificultad es la existencia de dos soluciones. La herramienta converge a la solución más próxima a la condición inicial proporcionada.

3.4. Evaluación de las Herramientas de MSOO para el Cálculo de Estacionarios

En principio, las herramientas de MSOO parecen perfectamente capaces de poder competir con las herramientas de simulación estacionaria. El formalismo matemático en el que están basada es el sistema hibrido de ecuaciones algebraico diferenciales, el cual engloba los sistemas de ecuaciones algebraicas utilizados en las herramientas

convencionales de simulación estacionaria. Ahora bien, las herramientas convencionales de simulación estacionaria tienen unas características y unas capacidades que se han discutido con detalle en § 3.1. Para evaluar la adecuación de las herramientas de MSOO en la simulación estacionaria, se han evaluado sus capacidades en relación a cada una de las características identificadas para las herramientas convencionales de simulación estacionaria.

3.4.1. Evaluación de Elementos de Conexionado en Herramientas de MSOO

Al nivel del lenguaje y también a nivel gráfico, puede considerarse que el elemento conectivo de las herramientas de MSOO es la corriente. La sentencia de definición de conexiones en EcosimPro es la siguiente:

```
CONNECT componente1.puertoA TO componente2.puertoB
```

Esta sentencia establece de manera clara una conexión de corriente que va del puerto A del componente 1 al puerto B del componente 2.

No obstante, es importante destacar que en las herramientas de MSOO se permite la posibilidad de tener conexiones múltiples en un mismo puerto, a diferencia de las herramientas convencionales de simulación de procesos y sistemas de conversión de energía, que generalmente no permiten múltiples conexiones en un solo puerto. En las herramientas convencionales, los componentes que requieren un número variable de conexiones suelen implementar un vector o arreglo de puertos, de manera que cada puerto puede tener una única corriente asociada.

Esta característica de las herramientas de MSOO, el permitir conexiones múltiples a los puertos, junto con la capacidad de definir las ligaduras y relaciones entre las variables de los puertos conectados (ecuaciones de igualdad y ecuaciones de suma nula), hace que las conexiones mediante corrientes de estas herramientas puedan dotarse de un comportamiento muy similar al de las conexiones basadas en nodos. Sin embargo, es importante destacar que formalmente no existe un objeto 'nodo' ni variables asociadas a dicho objeto.

3.4.2. Evaluación de Métodos de Solución Disponibles en Herramientas de MSOO

Las herramientas de MSOO consideran exclusivamente la aproximación orientada a ecuaciones. Sin embargo, existen múltiples campos en los que la aproximación modular secuencial o una combinación de dicha aproximación junto con la orientada a ecuaciones ofrece ventajas frente a la aproximación exclusivamente orientada a ecuaciones.

Sería interesante investigar la viabilidad de desarrollar aplicaciones utilizando herramientas de MSOO que implementen el método modular secuencial en conjunción con la aproximación orientada a ecuaciones. Esto podría conducir a soluciones más flexibles y eficientes en una variedad de campos. La combinación de ambas aproximaciones puede ayudar a abordar problemas complejos de manera más efectiva, permitiendo un enfoque modular para dividir el problema en partes más manejables y, al mismo tiempo, utilizar la orientación a ecuaciones cuando sea necesario para resolver interacciones críticas entre los componentes.

3.4.3. Evaluación de la Rasgadura en Herramientas de MSOO

En principio, las herramientas de MSOO no realizan una rasgadura sistemática del sistema de ecuaciones. La preocupación por encontrar una rasgadura cercana a la óptima ha conducido a que cada herramienta de MSOO haya implementado sus propios algoritmos heurísticos para la rasgadura. Sin embargo, estos algoritmos presentan múltiples inconvenientes, los cuales han sido analizados en el capítulo anterior.

Una manera de evitar los problemas asociados con los algoritmos heurísticos de rasgadura es implementar directivas de rasgadura en el lenguaje de modelado, que determinen de forma sistemática qué variables y ecuaciones deben considerarse para la rasgadura. El inconveniente radica en que, en el caso de Modelica, estas directivas no forman parte de la definición del lenguaje, por lo que algunas de las herramientas que utilizan Modelica han incorporado directivas para la rasgadura en las anotaciones. Un ejemplo concreto de esta práctica se encuentra en trabajo Coïc et al. (2020). En su investigación para el desarrollo de una herramienta destinada a la simulación estacionaria de turbinas de gas, se vieron en la necesidad imperativa de incorporar instrucciones directamente en los modelos de los componentes. Estas instrucciones tenían como objetivo guiar al compilador de Modelica utilizado en la selección precisa de variables de iteración y residuos. Aunque este trabajo no profundiza en la implementación detallada de dichas instrucciones, la tesis de master Kari (2022) aporta una perspectiva más detallada. En su investigación, Kari explora minuciosamente las instrucciones introducidas para optimizar la rasgadura en el compilador de Modelica OCT. Se presume que este compilador es similar al utilizado en el trabajo previo, dado que ambos estudios provienen de la misma compañía de simulación.

En el caso de EcosimPro, existen directivas, que forman parte del lenguaje, para indicar las variables y ecuaciones de rasgadura en los componentes.

Ahora bien, ni EcosimPro ni el compilador de Modelica OCT proporcionan construcciones suficientes para lograr una rasgadura sistemática similar a la realizada por las herramientas de estado estacionario que utilizan la aproximación nodal modificada, ya que se requerirían directivas adicionales en la declaración de los tipos de puerto o conexión, como se detallará a continuación.

Rasgadura Sistemática en Conexionado Mediante Nodos

Por ejemplo, en simulación de circuitos eléctricos con conexionado mediante nodos y utilizando la aproximación nodal modificada, los voltajes en los nodos deben ser variables de rasgadura y las ecuaciones de suma de corrientes en los nodos igual cero deben ser las ecuaciones de rasgadura correspondiente.

Tanto la variable de voltaje en los nodos o conexiones como la ecuación de suma de corrientes igual a cero se declaran en la definición del puerto eléctrico. La Figura 2-1 muestra la definición de puerto eléctrico que se utiliza en las librerías estándar de EcosimPro. Dentro de esta definición, el prefijo **`SUM`** en la declaración de la variable **`i`**, que representa la corriente eléctrica, instruye al procesador de modelos para generar automáticamente una ecuación de suma de corrientes igual a cero en todos los nodos.

```
-----------------------------------------------------------------------
-- Port elec (Electrical port)
-----------------------------------------------------------------------
PORT elec    "Electrical pin"
   EQUAL  REAL v      UNITS u_V       "Potential at pin"
   SUM    REAL i      UNITS u_A       "Current flowing into the pin"
END PORT
-----------------------------------------------------------------------
```

Figura 3-13: Declaración en EcosimPro del Puerto Eléctrico de las Librerías Estándar

Para lograr una rasgadura sistemática, sería necesario indicar que tanto la variable **`v`** como la ecuación de suma de corrientes igual a cero son de rasgadura. Sin embargo, en EcosimPro, esto presenta dos problemas. En primer lugar, para declarar que una variable es de rasgadura, se utiliza el prefijo **`ALG`** en su declaración, pero este prefijo solo está permitido para las variables declaradas dentro de los componentes, no en los puertos. El segundo problema es que la forma de indicar que una ecuación debe considerarse implícita es mediante el uso del operador **`IMPL()`** delante de la ecuación. Sin embargo, la ecuación de suma de corrientes igual a cero no se encuentra en ninguna parte del código, ya que se genera automáticamente.

Una propuesta de una posible sintaxis para lograr el comportamiento deseado sería extender el uso del prefijo **`ALG`** a las variables de puerto y permitir el uso del operador **`IMPL()`** delante de la palabra clave **`SUM`**, dado que **`SUM`** es la instrucción para construir la ecuación de suma de corrientes. La Figura 3-14 muestra las extensiones (en color rojo) que se proponen a la definición de puertos de EcosimPro con el fin de alcanzar el comportamiento deseado.

```
-----------------------------------------------------------------------
-- Port elec (Electrical port)
-----------------------------------------------------------------------
PORT elec    "Electrical pin"
   ALG    EQUAL REAL v   UNITS u_V       "Potential at pin"
   IMPL() SUM   REAL i   UNITS u_A       "Current flowing into the pin"
END PORT
-----------------------------------------------------------------------
```

Figura 3-14: Propuesta de Extensiones a la Definición de Puertos en EcosimPro Aplicada a un Puerto Eléctrico

Rasgadura Sistemática en Conexionado Mediante Corrientes

En cuanto a la definición de una rasgadura sistemática para el caso de conexionado mediante corrientes, se requiere poder declarar que las variables seleccionadas como independientes en las corrientes son variables de rasgadura. La Figura 3-15 muestra la definición en EcosimPro de un puerto fluido para agua, diseñado específicamente para su uso en librerías de balance térmico.

```
-----------------------------------------------------------------------
-- Port Water (for thermal balance calculation of steam power plants)
-----------------------------------------------------------------------
PORT Water SINGLE
   REAL W     UNITS “kg/s”     "Mass Flow"
   REAL P     UNITS “bar(a)”   "Pressure"
   REAL H     UNITS “kJ/kg”    "Enthalpy"
   REAL WH    UNITS “kW”       "Energy Flow"
   REAL T     UNITS “ºC”       "Temperature"
   REAL X     UNITS “nodim.”   "Quality"
   REAL rho   UNITS “kg/m3”    "Density"
   REAL S     UNITS “kJ/kg ºC” "Entropy"
CONTINUOUS
   WH  = W * H
   T   = H2O_Temp_vs_PH (P,H)
   X   = H2O_Qual_vs_PH (P,H)
   rho = H2O_Density_vs_PH (P,H)
   S   = H2O_Entropy_vs_PH (P,H)
END PORT
-----------------------------------------------------------------------
```

Figura 3-15: Declaración en EcosimPro de un Puerto para Flujos de Agua (en fase vapor, en fase líquida o en las dos fases)

Se ha mostrado previamente que en el caso de un puerto fluido con un único componente (agua en este caso), la corriente se puede caracterizar mediante solo tres variables independientes. En el caso del puerto de agua, las tres variables independientes serían el flujo másico `W`, la presión `P` y la entalpía `H`. Por tanto, estas deberían ser las variables de rasgadura. La propuesta de extender el uso del prefijo **`ALG`** también a las definiciones de los puertos permitiría definir que estas variables son de rasgadura, como se ilustra en la Figura 3-16.

```
-----------------------------------------------------------------------
-- Port Water (for thermal balance calculation of steam power plants)
-----------------------------------------------------------------------
PORT Water SINGLE
   ALG REAL W     UNITS “kg/s”     "Mass Flow"
   ALG REAL P     UNITS “bar(a)”   "Pressure"
   ALG REAL H     UNITS “kJ/kg”    "Enthalpy"
   REAL WH    UNITS “kW”       "Energy Flow"
   REAL T     UNITS “ºC”       "Temperature"
   REAL X     UNITS “nodim.”   "Quality"
   REAL rho   UNITS “kg/m3”    "Density"
   REAL S     UNITS “kJ/kg ºC” "Entropy"
CONTINUOUS
   WH  = W * H
   T   = H2O_Temp_vs_PH (P,H)
   X   = H2O_Qual_vs_PH (P,H)
   rho = H2O_Density_vs_PH (P,H)
   S   = H2O_Entropy_vs_PH (P,H)
END PORT
-----------------------------------------------------------------------
```

Figura 3-16: Propuesta de Extensiones a la Definición de Puertos en EcosimPro Aplicada a un Puerto Fluido para Agua

3.4.4. Evaluación de Ecuaciones Condicionales

Las herramientas de MSOO proporcionan ecuaciones condicionales.

En el caso de Modelica, se pueden utilizar **`if-expressions`** e **`if-equations`** para expresar este tipo de ecuaciones.

Sentencia `if-expression` de Modelica

La sintaxis de los **`if-expression`** de Modelica es la siguiente:

```
expression = if condition then expression1
             else expression2;
```

El **if-expression** se utiliza para calcular de manera condicional el lado derecho de una ecuación. Es posible anidar sentencias de este tipo, como se muestra en el siguiente ejemplo:

```
class ifExpression
   parameter Real uMax = 10;
   parameter Real uMin = 2;
   Real u;
   Real y;
equation
   y = if u > UMax then uMax
       else if u < uMin then uMin
       else u;
end ifExpression;
```

Sentencia `if-equation` de Modelica

La sentencia **`if-equation`** de Modelica es muy similar a la **`if-expression`**, pero en lugar de modificar el lado derecho de la ecuación, cambia la ecuación completa. Además, permite incluir varias ecuaciones para cada condición. La sintaxis de esta sentencia es la siguiente:

```
if condition1 then
   { equation ";" }
{ elseif condition2 then
   { equation ";" }
}
[ else
   { equation ";" }
]
end if ";"
```

A continuación, se muestra un ejemplo de uso de la sentencia **`if-equation`** de Modelica:

```
class ifEquation
   parameter Real uMax = 10;
   parameter Real uMin = 2;
   Real u;
   Real y;
   Real y2;
equation
   if u > UMax then
      y = uMax;
      y2 = uMax^2;
   elseif u < uMin then
      y = uMin;
      y2 = uMin^2;
   else
      y = u;
      y2 = u^2;
   end ifEquation;
```

Sentencia `ZONE` de Modelica

En el caso de EcosimPro, la sentencia **`ZONE`** permite introducir ecuaciones condicionales en función de diferentes estados o condiciones de funcionamiento. Esta sentencia es similar al **`if-expression`** de Modelica, aunque a diferencia de Modelica, no permite el anidamiento, pero en su lugar, admite un número arbitrario de ramas, de forma que resulta muy equivalente.

```
expression = ZONE "("condition")" expression
           { ZONE "("condition")" expression}
             OTHERS expression
```

A continuación, se presenta un ejemplo de una misma ecuación condicional programada en Modelica utilizando una **`if-equation`** y en EcosimPro utilizando la sentencia **`ZONE`**, donde se puede apreciar la similitud entre ambas.

MODELICA	ECOSIMPRO

`class ifEquation`
`  parameter Real uMax = 10;`
`  parameter Real uMin = 2;`
`  Real u;`
`  Real y;`
`equation`
`  if u > uMax then`
`  y = uMax;`
`  elseif u < uMin then`
`  y = uMin;`
`  else`
`  y = u;`
`  end ifEquation;`
	`COMPONENT ifEquation`
	`  DATA`
	`  REAL uMax = 10`
	`  REAL uMin = 2`
	`  DECLS`
	`  REAL u`
	`  REAL y`
	`  CONTINUOUS`
	`  y = ZONE (u > uMax) uMax`
	`  ZONE (u < uMin) uMin`
	`  OTHERS u`
	`  END COMPONENT`

Análisis Estructural de Ecuaciones Condicionales en Herramientas de MSOO

A pesar de que las herramientas de MSOO permiten expresar ecuaciones condicionales, el análisis de la estructura del sistema de ecuaciones con el fin de detectar problemas estructuralmente singulares es un desafío considerable. Al examinar un sistema de ecuaciones que incluye ecuaciones condicionales, es necesario evaluar cada una de las posibles combinaciones de estas ecuaciones para llegar a conclusiones correctas. Naturalmente, la complejidad de este análisis aumenta de manera exponencial. Por ejemplo, en un modelo con N ecuaciones condicionales binarias (es decir, con dos ecuaciones posibles para considerar), el número de combinaciones a evaluar es de 2^N.

Por lo general, las herramientas de MSOO simplifican el problema y tratan las diferentes ecuaciones de un grupo condicional como una única ecuación equivalente. Para realizar esta simplificación, es necesario hacer una suposición acerca de las variables que intervienen en dicha ecuación equivalente. Existen dos posibles suposiciones:

La primera suposición consiste en asumir que en dicha ecuación intervienen todas las variables que aparecen en cualquiera de las ramas. Sin embargo, esta suposición no es segura, ya que pueden no detectarse ecuaciones que son estructuralmente singulares. Por ejemplo, la siguiente ecuación condicional (expresada en lenguaje Modelica) que tiene una segunda rama que es estructuralmente singular, se consideraría correcta:

```
0 = if u > uMax then y - uMax
    else           0;
```

La segunda suposición consiste en considerar que las únicas variables que intervienen en la ecuación equivalente son aquellas que aparecen en todas las ramas. No obstante, esta suposición en muchas ocasiones resulta demasiado estricta y puede llevar a la detección de problemas de estructura singular donde no existen.

Las herramientas que implementan Modelica generalmente optan por la primera suposición, mientras que EcosimPro utiliza la segunda suposición. Por ejemplo, EcosimPro considera que en la siguiente ecuación

```
x = ZONE (u > uMax) y
    ZONE (u < uMin) w
    OTHERS          z
```

solo interviene la variable x, ya que las variables z, w e y no están presentes en todas las ramas. El problema es que en ocasiones, este criterio resulta ser demasiado estricto y puede llevar a que se indique que el modelo es estructuralmente singular, sin que lo sea en realidad. En tales casos, el desarrollador de EcosimPro puede introducir una dependencia ficticia en todas las ramas con respecto a todas las variables para que la comprobación de singularidad estructural se aplique de manera menos estricta, como se indica en el siguiente ejemplo.

```
zero = 0
x = ZONE (u > uMax) y + zero*(w + z)
    ZONE (u < uMin) w + zero*(y + z)
    OTHERS          z + zero*(w + y)
```

Funcionamiento Incorrecto de las Ecuaciones Condicionales en Estacionarios

Aparte de los problemas de comprobación del sistema de ecuaciones, la implementación de las ecuaciones condicionales en las herramientas de MSOO es deficiente para el cálculo de estacionarios. Este problema se deriva en gran medida de que los algoritmos utilizados para detectar cambios de zona están diseñados principalmente con un enfoque en la simulación transitoria, sin considerar adecuadamente la problemática asociada al cálculo de estados estacionarios.

Los algoritmos de detección de eventos en MSOO se enfocan en identificar si cambian las condiciones durante un paso de integración. Si se detecta un cambio en una o más condiciones, el sistema busca determinar el momento exacto en que ocurre la transición de la primera condición. Para lograrlo, se retrocede al inicio del paso de integración y se utiliza un algoritmo de búsqueda de raíces, como el algoritmo de Illinois, para estimar el momento de cambio de la condición. Este proceso se repite iterativamente hasta que se alcanza la precisión requerida en la determinación del tiempo de cambio de la condición (Mao & Petzold, 2002).

En general, las herramientas de MSOO no prestan atención al problema de detectar los cambios de zona de las ecuaciones condicionales en simulación estacionaria. Esta carencia puede resultar en problemas de convergencia cuando la solución inicial asumida no está ubicada en la zona adecuada.

Un ejemplo destacado de esta problemática se relaciona con la representación de los controladores en los modelos, que suele dar lugar a ecuaciones condicionales. Los

controladores siempre incorporan limitaciones en la acción de control, ya que existen límites inferiores y superiores para la variable de control. Cuando el controlador está activo y no se alcanzan los límites de saturación, el controlador opera en modo de modulación y puede describirse mediante la siguiente ecuación, en caso de que cuente con acción integral:

$$y - y_{sp} = 0 \tag{3-46}$$

donde y_{sp} es la señal de referencia es la variable controlada. Ahora bien, si se requiere una acción de control que esté fuera de los límites de saturación para alcanzar el punto de ajuste, entonces se activa la saturación, y la ecuación (3-46) ya no describe adecuadamente el comportamiento del sistema. En su lugar, se aplica las siguientes una de las dos siguientes ecuaciones:

$$u - u_{max} = 0 \tag{3-47}$$

$$u - u_{min} = 0 \tag{3-48}$$

Una de las dificultades que se menciona en la literatura (Casella et al., 2011) para el cálculo del estacionario de modelos termo-fluidos con herramientas de MSOO consiste en que es necesario especificar correctamente la zona en que operan los controladores del modelo. La dificultad principal proviene de la falta de algoritmos adecuados que permitan determinar durante el cálculo del estado estacionario cuáles de los controladores están en modo de modulación, cuáles están saturados en su valor máximo y cuáles están saturados en su valor mínimo. Dicho problema, la determinación de la zona de operación de los controladores, es altamente no lineal y tiene una complejidad potencialmente combinatoria, lo que puede generar serios problemas de convergencia a los algoritmos de solución de estacionarios.

Para abordar eficazmente esta cuestión, sería fundamental incorporar en las herramientas de MSOO algoritmos adecuados para la resolución de ecuaciones condicionales en estado estacionario. Un ejemplo de tal algoritmo es el algoritmo de cruce de límites propuesto por Zaher (1995).

3.4.5. Evaluación de Resolución de Problemas de Diseño

Conceptualmente, las herramientas de MSOO al estar Orientadas a Ecuaciones deberían ser capaces de abordar problemas de diseño sin dificultad, ya que permiten la introducción de especificaciones diseño en forma ecuaciones adicionales a las de los componentes del modelo.

En el ámbito del diseño las turbinas de gas para propulsión aeronáutica, es común enfrentarse a problemas de diseño bastante complejos de diseño multipunto, que involucran la introducción de requisitos o especificaciones de diseño en diferentes puntos

de operación, como el de máximo empuje de despegue (MTO - Maximum Take Off), el de crucero (CRZ - Cruise), el punto final de la subida (TOC - Top of Climb) y otros. Tanto, Proosis (Alexiou & Tsalavoutas, 2011) que es una herramienta para la simulación de turbinas de gas basada en EcosimPro, como la herramienta descrita por Coïc et al. (2020), que se basa en Modelica permiten abordar problemas de diseño con objetivos de diseño simultáneos en múltiples puntos de operación. No obstante, es importante destacar que el enfoque utilizado en ambas herramientas para resolver estos problemas de diseño en múltiples puntos ha sido notablemente diferente.

Desde un punto de vista conceptual, los problemas de diseño multipunto de motores se definen mediante una matriz de mapeado de puntos de diseño. Dicha matriz relaciona o vincula los puntos de diseño con las variables de diseño, las actuaciones requeridas del sistema, las estimaciones del rendimiento de los componentes y los límites tecnológicos (Schutte, 2009).

Solución Problemas de Diseño con EcosimPro/Proosis

En el caso de la resolución de problemas de diseño de turborreactores con EcosimPro/Proosis, el trabajo de Alexiou & Tsalavoutas (2011) describe de forma exhaustiva las técnicas disponibles, que básicamente son solo dos: la utilización de *particiones de diseño* y la aplicación de *estacionarios extendidos*.

Particiones de Diseño

Las particiones son un concepto específico de EcosimPro. El modelo matemático de un componente no necesariamente debe ser completo para llevar a cabo la simulación; las particiones permiten completar el modelo matemático, permitiendo así la ejecución de la simulación del mismo.

La definición de una partición consta de hasta cuatro pasos consecutivos, y la herramienta proporciona asistentes interactivos para cada uno de estos pasos. Estos cuatro pasos son:

Paso 1: Este paso es opcional y permite al usuario liberar datos, es decir, los datos seleccionados por el usuario pasan a ser considerados como incógnitas.

Paso 2: En caso de ligaduras entre variables de estado (problema de índice superior), se aplica el algoritmo de Pantelides. Este algoritmo introduce ecuaciones derivadas que facilitan el cálculo algebraico de algunas de las variables de estado. Es decir, permite transformar algunas de las variables de estado en variables algebraicas haciendo uso las nuevas ecuaciones derivadas. EcosimPro selecciona de forma automática las variables de estado que se transforman en algebraica. Ahora bien, la selección no es única y el usuario puede modificarla en base a su conocimiento del problema.

Paso 3: Consiste en la selección de variables como condiciones de contorno en el número necesario para igualar el número de variables y el número de ecuaciones. En este paso, se prohíbe seleccionar variables que puedan dar lugar a un problema de índice superior.

Paso 4: En caso de que se detecten lazos algebraicos, es necesario rasgar (*to tear*) los lazos algebraicos mediante la selección de las variables de rasgadura y las ecuaciones implícitas asociadas. La herramienta propone variables de rasgadura basándose en las directivas de rasgadura y en un algoritmo heurístico. El asistente para la rotura de lazos algebraicos permite que el usuario puede modificar las variables seleccionadas, pero no proporciona control sobre las ecuaciones consideradas implícitas.

En el caso de EcosimPro/Proosis, la formulación estándar de los componentes representa el comportamiento fuera de diseño y tiene en cuenta la dependencia del comportamiento del componente con respecto a datos o parámetros que representan el dimensionado del mismo. Las particiones permiten resolver fácilmente los problemas de diseño monopunto. Basta con emplear el asistente de particiones para liberar los datos relacionados con el dimensionado de los componentes. Seguidamente, con el asistente de condiciones de contorno, se establecen las especificaciones de diseño necesarias para cerrar el problema. Sin embargo, es importante destacar que el uso de particiones de diseño en EcosimPro solo posibilita resolver problemas de diseño monopunto.

Estacionarios Extendidos

La técnica más general de solución de problemas de diseño con EcosimPro es el uso de *estacionarios extendidos*, ya que permite realizar tanto diseños monopunto como diseños multipunto. El cálculo de *estacionarios extendido* es una capacidad que se ha incorporado dentro del lenguaje de experimentos de EcosimPro.

El nombre de *estacionarios extendidos* hace referencia a la posibilidad de ampliar el sistema de ecuaciones del modelo mediante la inclusión de ecuaciones adicionales. Estas nuevas ecuaciones representan tanto las especificaciones de diseño como las ecuaciones del modelo correspondientes a otros puntos de operación.

En el lenguaje de experimentos se han incorporado funcionalidades que permiten la construcción directa del sistema de ecuaciones del diseño, tanto monopunto como multipunto. Concretamente, se proporcionan las siguientes capacidades en el lenguaje de definición de experimentos:

- *Cálculo del sistema de ecuaciones del modelo con valores de entrada dados* mediante llamada a una función del sistema. Esto permite obtener los valores de los residuos de las ecuaciones implícitas del modelo. Es necesaria una llamada a esta función para cada punto de operación considerado en el diseño.

- *Acceso a los valores de los residuos del modelo*, calculados con la función anterior.
- *Definición de ecuaciones implícitas adicionales a las del modelo* para imponer las especificaciones de diseño.
- *Construcciones para permitir la introducción de ecuaciones condicionales*.
- *Acceso a los datos y variables del modelo* en cualquier punto de la ejecución, pudiendo cambiar los datos. Esta es una capacidad genérica del lenguaje de experimentos.

Empleando estas capacidades, es posible definir en el experimento una función que implemente un sistema de ecuaciones como el representado en la Figura 3-7 para el diseño monopunto o como el representado en la Figura 3-8 para el diseño multipunto. Una vez formado dicho sistema de ecuaciones, se puede invocar un resolvedor de ecuaciones algebraicas, que recibe como argumento de entrada un puntero a la función que implementa el sistema de ecuaciones, permitiendo así obtener la solución del problema de diseño.

En principio, el uso de estas capacidades puede parecer complejo, pero la herramienta proporciona asistentes que guían al usuario en la especificación del problema de diseño. Estos asistentes organizan la información de manera bastante similar a cómo se estructura en las matrices de mapeado de puntos de diseño, propuesta por Schutte (2009) como definición conceptual del problema de diseño.

Solución Problemas de Diseño con Modelica

En el caso de la librería JetPropulsion de Modelica, también se han implementado capacidades de diseño (Sielemann et al., 2020), pero siguiendo una estrategia completamente diferente a la de la librería Turbo de EcosimPro. Para el diseño monopunto, se ha optado por un conmutador a nivel de modelo que permite cambiar la formulación de todos los componentes entre diseño y fuera de diseño.

Para el caso del diseño multipunto, se ha optado por utilizar un modelo que incluye múltiples instancias del sistema, (esto es, el motor), tal y como se muestra en Figura 3-17. Cada una de las instancias representa el motor en una de las condiciones de operación predefinidas en las que se desean imponer especificaciones de diseño. Con el fin de imponer las especificaciones de diseño se tienen que emplear bloques que representan dichas ecuaciones, estos son las cajas rectangulares de color naranja.

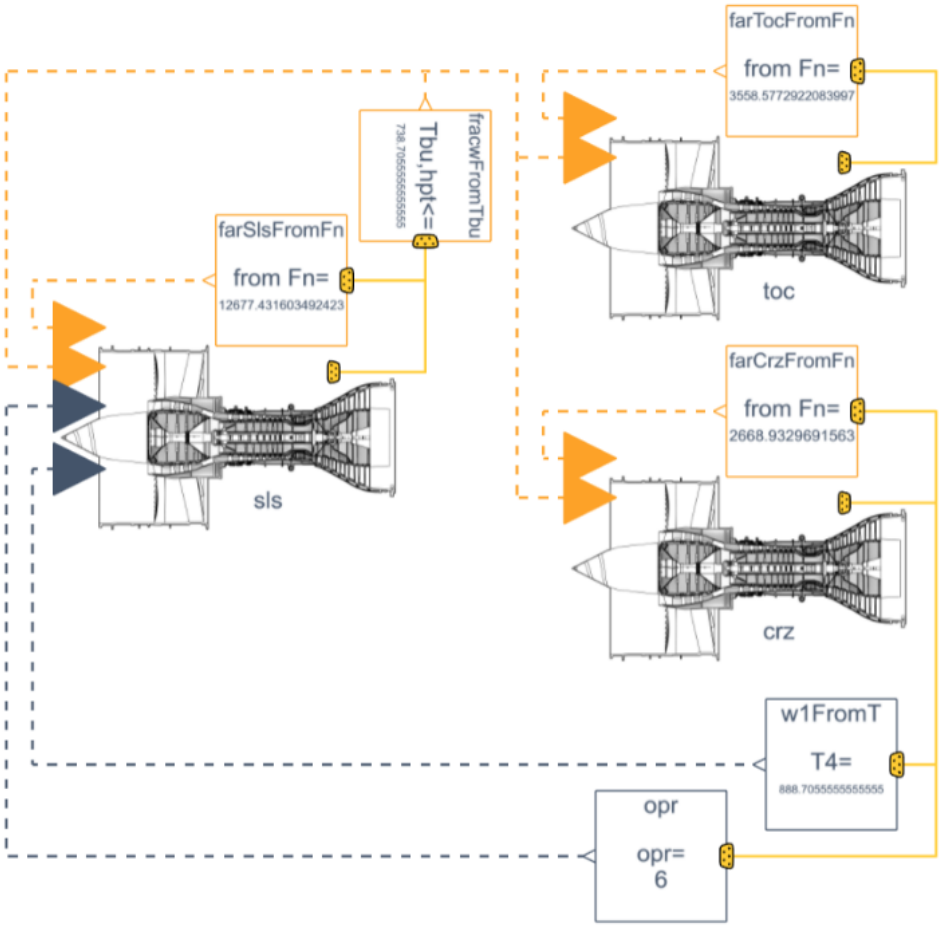


Figura 3-17: Modelo para Diseño Multipunto con Modelica -copiado de (*Enhanced Turbojet Multi-Point - Help Center*, n.d.)

En principio, esta aproximación puede parecer menos compleja que los estacionarios extendidos de EcosimPro. Sin embargo, no está claro que esto sea necesariamente así, ya que, como se ha indicado, la forma más estándar para representar problemas de diseño es mediante una tabla. Los asistentes para los estacionarios extendidos de EcosimPro son más cercanos a dicha representación tabular que el diagrama de flujo de los modelos utilizados en Modelica. Por otro lado, la aproximación de Modelica tiene una limitación, y es que obliga a que uno de los puntos de operación sea el punto de diseño de todos los componentes, así que no diferencia entre el punto de diseño del ciclo y el punto de diseño de los componentes.

4. COMPARACIÓN ENTRE HERRAMIENTAS DE SIMULACIÓN ESPECÍFICAS Y APLICACIONES DE MSOO

En este capítulo, se comparan algunas herramientas de simulación específicas con aplicaciones de MSOO adaptadas al mismo campo o ámbito de simulación. El propósito de estas comparaciones es identificar las causas que hacen que las aplicaciones de MSOO sean inferiores a los programas de simulación específicos y así poder proponer soluciones para mejorarlas. Dada la extensión del tema, la comparación se ha restringido a dos campos específicos: la simulación de circuitos eléctricos mediante programas del tipo SPICE y la simulación de transitorios hidráulicos, es decir, el golpe de ariete. A continuación, se presentan los resultados obtenidos en estas dos comparaciones.

4.1. Herramientas de Simulación de Circuitos Eléctricos vs Aplicaciones de MSOO

4.1.1. Herramienta Específica de Simulación de Circuitos Eléctricos: SPICE

SPICE (Simulation Program with Integrated Circuit Emphasis) es un software de simulación de circuitos eléctricos ampliamente reconocido y utilizado tanto en la industria como en el ámbito académico. Inicialmente desarrollado en la década de 1970 en la Universidad de California, Berkeley, SPICE ha experimentado un continuo desarrollo y se ha establecido como un estándar de facto en la simulación de circuitos eléctricos.

El propósito fundamental de SPICE es brindar a los diseñadores de circuitos electrónicos la capacidad de evaluar el comportamiento y las características de funcionamiento de estos previamente a su implementación física.

El usuario describe el circuito como una red de componentes eléctricos conectados a nodos eléctricos, que representan puntos de voltaje común. Los componentes eléctricos, como resistencias, condensadores, inductores y transistores se representan mediante ecuaciones matemáticas que describen su comportamiento eléctrico, conocidas como ecuaciones constitutivas de rama (branch constitutive equations). Las conexiones a los nodos establecen relaciones entre las variables de los componentes: el voltaje en un nodo es compartido por todos los terminales de los componentes conectados a ese nodo, y la suma de las corrientes que entran y salen del nodo debe ser igual a cero, en cumplimiento de la ley de corrientes de Kirchoff.

SPICE es capaz de realizar tres tipos análisis de básicos:

1) *Análisis de corriente continua (DC):* Este análisis se utiliza para estudiar el comportamiento de los circuitos en estado estacionario. Calcula las corrientes y voltajes en el circuito cuando todas las fuentes de señal son constantes.
2) *Análisis de pequeñas señales (AC):* Este tipo de análisis se realiza en el dominio de la frecuencia y permite estudiar el comportamiento de los circuitos en respuesta a señales de corriente alterna. Calcula ganancia y fase entre señales de entrada y salida en función de la frecuencia.
3) *Análisis transitorio (TRAN)*: Este tipo de análisis evalúa cómo evolucionan las corrientes y voltajes en función del tiempo después de una perturbación.

Adicionalmente, se ofrecen otros tipos de análisis que son derivados de los tipos básicos. Estos tipos de análisis derivados pueden variar entre las diferentes versiones de SPICE. Dado que el código de SPICE es de acceso público, esto ha dado lugar a la creación de numerosas versiones y variantes. La Figura 4-1 muestra los tipos básicos de análisis en los simuladores de la familia de SPICE y algunos de los tipos derivados.

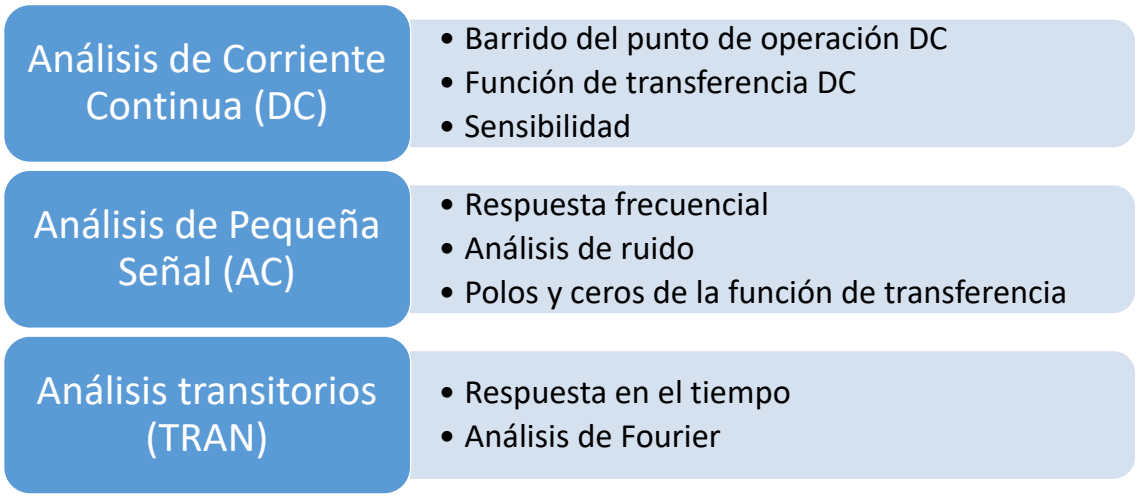


Figura 4-1: Tipos de Análisis en SPICE

SPICE aplica el método del análisis nodal modificado (MNA, Modified Nodal Approach) para generar directamente sistemas de ecuaciones que sean resolubles. A diferencia de las herramientas de MSOO, en las cuales se genera primero un sistema de ecuaciones aplanado y luego se procesa simbólicamente dicho sistema para adaptarlo a los algoritmos de solución de DAE's proporcionados por la herramienta. SPICE genera directamente sistemas de ecuaciones, que son adecuados a los algoritmos de solución proporcionados por la herramienta.

Es interesante señalar un par de diferencias entre el diseño de SPICE y las aplicaciones desarrolladas con herramientas de MSOO para simulación de circuitos eléctricos:

1. SPICE no incluye modelos de elementos completamente ideales, como interruptores con resistencia infinita en estado abierto y resistencia nula en estado cerrado, diodos con resistencia nula en estado de conducción y resistencia infinita en estado de corte, o amplificadores operacionales con ganancia infinita, ancho de banda infinito, impedancia de entrada infinita e impedancia de salida nula. De hecho, en la literatura sobre SPICE (Kielkowski, 1993) se menciona que, para evitar problemas numéricos, todos los componentes tienen una conductancia mínima y que los componentes que pueden tener una conductancia excesivamente alta, consideran la inclusión de una resistencia muy pequeña en serie.
2. En la literatura relacionada con SPICE, se observa que hay una escasa mención a los problemas de índice superior. El método de resolución de transitorios de SPICE está diseñado para abordar los problemas de índice 2, que son comunes en circuitos eléctricos. En teoría, es posible que se presenten problemas de índice superior a 2 e incluso problemas con estructura variable en el caso de incluir elementos ideales en los circuitos eléctricos. Sin embargo, SPICE evita tales problemas al no considerar la inclusión de elementos ideales en sus modelos.

4.1.2. Aplicaciones de MSOO para la Simulación de Circuitos Eléctricos

Desde los comienzos del modelado y simulación orientados a objeto se han desarrollado múltiples librerías de componentes para la simulación de circuitos electrónicos. Dichas librerías se han inspirado en SPICE y han intentado proporcionar componentes similares a los de SPICE, y también capacidades de análisis similares, aunque esto último de forma limitada.

En el caso de Modelica se han desarrollado al menos tres librerías para la simulación de circuitos eléctricos (Cellier et al., 2007): la librería estándar de Modelica (Clauß et al., 2000), SPICELib (Urquia et al., 2005) y BondLib (Cellier & Nebot, 2005). La Tabla 4-1 muestra las características principales de dichas librerías; se observa que solo SPICELib ofrece los tres tipos de análisis de SPICE, y que se considera que dicha librería es la más conveniente para usuarios avanzados de SPICE. La forma en que SPICELib consigue proporcionar los tres tipos de análisis de SPICE es que cada componente incluye variables y ecuaciones para cada uno de los tipos de análisis. Además, los conectores también requieren variables de voltaje y corriente para los tres tipos de análisis. Este enfoque introduce una considerable complejidad en el desarrollo del software. Si bien las ecuaciones y variables para los tres tipos de análisis siempre están presentes, la programación se encarga de garantizar que las ecuaciones correspondientes a los tipos de análisis no activos tengan una solución trivial.

Es interesante señalar que en una comparación de estas tres librerías (Cellier et al., 2007) se critica que la aproximación adoptada por SPICELib es la menos orientada a objeto. Se propone una forma alternativa para realizar los análisis de pequeña señal AC, que consiste en exportar el modelo linealizado desde el entorno de MSOO a Matlab. Sin embargo, al mismo tiempo que se hace esta propuesta, se reconoce que SPICELib puede ser la librería más adecuada para un usuario de SPICE. Esto refleja una cierta contradicción, ya que se está reconociendo implícitamente que la aproximación orientada a objeto puede no ser la mejor opción para cumplir con las expectativas del usuario típico de SPICE.

Tabla 4-1: Características Principales de las Librerías de Modelica para Simulación Eléctrica

	Librería Modelica Estándar	SPICELib	BondLib
Tipo de Análisis	TRAN	TRAN, AC y DC	TRAN y DC
Lista de Componentes	Propios[1] y subconjunto de SPICE3 [2]	Subconjunto de PSPICE	Subconjunto de HSPICE
Complejidad Componentes	Más simples que SPICE	Similar a PSPICE	Similar a HSPICE
Definición Componentes	Ecuaciones	Ecuaciones	Bond Graph
Compatibilidad con Librería estándar	Compatible	Incompatible	Compatible
Comentarios	Puede modelar la disipación térmica	Más adecuada para usuarios de SPICE	

4.2. Herramientas de Simulación de Transitorios Hidráulicos vs Aplicaciones de MSOO

El flujo transitorio de líquidos en tuberías puede generar ondas de presión que se propagan a través del cuerpo del fluido a la velocidad local del sonido, dando lugar a sobrepresiones

[1] La sección eléctrica de la biblioteca estándar de Modelica ofrece componentes propios, distintos de los utilizados en SPICE. Incluye elementos pasivos fundamentales como resistencias, condensadores e inductores, así como modelos sencillos de transistores y diodos. También incorpora fuentes de voltaje y corriente.

[2] Desde la versión 3.2 de la biblioteca estándar de Modelica, se ha agregado una subsección en la sección eléctrica que incorpora un subconjunto de los componentes de Spice 3 (Majetta et al., 2011). Esta subsección incluye modelos básicos, fuentes y dispositivos semiconductores esenciales, como diodos, BJT, MOSFET de nivel 1 y resistencias.

y otros fenómenos asociados. Estos fenómenos pueden incluir el desplazamiento de las tuberías en sus soportes, dañar dichos soportes, causar fugas e incluso la posibilidad de que las tuberías y los recipientes revienten. Comúnmente, a los transitorios hidráulicos se les denomina "golpe de ariete" debido a que los transitorios hidráulicos rápidos suelen generar o ir acompañados de un fuerte sonido metálico.

Los transitorios hidráulicos, o el golpe de ariete, pueden manifestarse en una amplia variedad de situaciones, tales como en sistemas de propulsión aeroespacial, estaciones hidroeléctricas, sistemas de generación de vapor y de enfriamiento en plantas de energía, sistemas de calefacción urbana, plantas industriales, sistemas de suministro de agua, redes de riego, plomería doméstica y oleoductos para transporte de productos químicos y petróleo.

El golpe de ariete es un fenómeno que puede tener consecuencias graves en sistemas de tuberías. Estas ondas de presión súbitas pueden causar daños costosos, pérdida de producción y, en casos extremos, representar un peligro para la seguridad. Por lo tanto, es fundamental disponer de herramientas especializadas para su análisis y el diseño de dispositivos de mitigación.

Estas herramientas permiten a los ingenieros y diseñadores prever y comprender cómo se comportará un sistema de tuberías en diferentes situaciones operativas. Al anticipar y evaluar posibles escenarios de golpe de ariete, es posible tomar medidas preventivas para evitar daños costosos y tiempo de inactividad no planificado.

4.2.1. Herramientas Específicas de Simulación de Transitorios Hidráulicos

En esta sección dedicada a las herramientas específicas para la simulación de transitorios hidráulicos en sistemas de tuberías, primero se presentan las ecuaciones de conservación de masa y momento que describen el flujo transitorio unidimensional de líquidos. Luego, se explica cómo estas ecuaciones se simplifican para el caso de flujos estacionarios, ya que el cálculo de condiciones estacionarias es un paso previo crucial antes de abordar el análisis de los transitorios. Por último, se proporcionan detalles sobre los métodos de cálculo empleados por un conjunto representativo de programas comerciales diseñados específicamente para llevar a cabo el cálculo de transitorios hidráulicos.

Ecuaciones del Golpe de Ariete

Los libros clásicos sobre golpe de ariete de Chaudhry (2014) y de Wylie et al. (1993) ofrecen una derivación detallada de las ecuaciones que describen los transitorios hidráulicos en tuberías. A continuación, se presentan dichas ecuaciones, en las que se ha incluido un término de fricción transitoria propuesto por Vítkovský et al. (2006):

$$\frac{a^2}{g\,A}\frac{\partial Q}{\partial x}+\frac{\partial H}{\partial t}=0 \tag{4-1}$$

$$\frac{1}{g\,A}\frac{\partial Q}{\partial t}+\frac{\partial H}{\partial x}+\frac{f}{2g\,D}\frac{Q|Q|}{A^2}+\frac{k}{g\,A}\left(\frac{\partial Q}{\partial t}+a\,\mathrm{sign}(Q)\left|\frac{\partial Q}{\partial x}\right|\right)=0 \tag{4-2}$$

donde Q es el flujo (m3/s), H es la altura piezométrica (m), a es la velocidad de onda, g es la aceleración de la gravedad (m^2/s), D es el diámetro de la tubería (m), A es el área de flujo (m2), f es el factor de fricción de Darcy, k es el coeficiente de decaimiento para la fricción transitoria, x es la distancia a lo largo de la tubería (m), y t es el tiempo (s).

Ecuaciones Estacionarias

Una característica importante de cualquier herramienta de análisis de transitorios hidráulicos es su capacidad para calcular estacionarios, que deben servir como punto de comienzo a los estudios transitorios. Eliminando en las ecuaciones (4-1) y (4-2) la dependencia con respecto al tiempo, resultan las siguientes ecuaciones que describen el flujo en estado estacionario:

$$\frac{dQ}{dx}=0 \tag{4-3}$$

$$\frac{dH}{dx}+\frac{f}{2g\,D}\frac{Q|Q|}{A^2}=0 \tag{4-4}$$

Integrando las ecuaciones en estado estacionario a lo largo de la longitud de la tubería entre las secciones de entrada y salida (los valores de las variables en las secciones entrada salida se designan respectivamente con los sufijos “in” y “out”), se obtienen las siguientes relaciones:

$$Masa:\ \ Q_{in}=Q_{out} \tag{4-5}$$

$$Momento: H_{in}-H_{out}+f\frac{L}{D}\frac{Q_{in}|Q_{in}|}{2g\,A^2}=0 \tag{4-6}$$

Cada tubería proporciona dos ecuaciones con cuatro incógnitas: $H_{in},\ H_{out}, Q_{in}, Q_{out}$. Para formar el sistema de ecuaciones y determinar todas las incógnitas, es necesario considerar las ecuaciones de todas las tuberías y agregar las ecuaciones adicionales que describan las condiciones en los extremos de las tuberías. Los componentes ubicados en los extremos de las tuberías proporcionan las ecuaciones adicionales necesarias para igualar el número de incógnitas y el de ecuaciones.

Una vez resuelto el sistema de ecuaciones en estado estacionario, se obtienen los valores de las variables hidráulicas en los extremos de todas las tuberías. Utilizando esta información y asumiendo una distribución uniforme de la fricción, es posible determinar la distribución de alturas piezométricas y flujos a lo largo de cada tubería. Los métodos transitorios subdividen las tuberías en volúmenes de control (nodos) o en secciones. A partir de las distribuciones estacionarias, es factible calcular los valores iniciales de las variables hidráulicas en los nodos o secciones de cálculo, que se emplean para la solución transitoria. Por ejemplo, en el caso de considerar n secciones equidistantes en la tubería, con la primera sección justo en la entrada y la sección n justo en la salida, las alturas y flujos en las secciones pueden calcularse con las formulas siguientes:

$$Q_i = Q_{in} \qquad \forall\, i \in 1,2,\dots,n \tag{4-7}$$

$$H_i = H_{in} - (H_{in} - H_{out})\frac{(i-1)}{(n-1)} \qquad \forall\, i \in 1,2,\dots,n \tag{4-8}$$

Software Comercial de Golpe de Ariete

Ghidaoui et al. (2005) realizaron una revisión y descripción de un conjunto representativo de programas para el análisis del fenómeno de golpe de ariete. En la Tabla 4-2 se detallan, para cada uno de los programas revisados, el método numérico utilizado para resolver las ecuaciones del flujo transitorio, así como la empresa o institución responsable de su desarrollo y comercialización.

Tabla 4-2: Algunos Programas Comerciales de Golpe de Ariete

Institución o Compañía	Software	Método
Universidad de Cambridge	Pipenet	Método de las Características (MOC)
Bentley Systems, Inc.	HAMMER	Método de las Características (MOC)
Universidad de Auckland	HYTRAN	Método de las Características (MOC)
DHI	HYPRESS	Diferencias finitas implícitas
Applied Flow Technology	IMPULSE	Método de las Características (MOC)
WL / Delft Hydraulics	WANDA	Método de las Características (MOC)
BHR Group	FlowMaster	Método de las Características (MOC)
Universidad de Kentucky	SURGE	Método de la Onda Plana (plane-wave)

Institución o Compañía	Software	Método
Stoner Associates, Inc.	LIQT	Método de las Características (MOC)
US Army Corps Engineers	WHAMO	Diferencias finitas implícitas
Universidad de Toronto	TRANSAM	Método de las Características (MOC)

En la tabla anterior, se observa que ocho de los once programas revisados utilizan el método de las características. Este método es altamente popular debido a que combina alta precisión, simplicidad y eficiencia matemática. Se trata de un método explícito, lo que significa que su incremento de tiempo está sujeto a la restricción de que el número de Courant[1] debe ser menor o igual que 1.

El método de la onda plana, que guarda mucha similitud con el MOC, también es un método explícito en el que el paso de integración está limitado por el número de Courant. Este método incorpora de manera explícita las trayectorias de las ondas en el procedimiento de solución y agrupa la fricción en el centro de cada tubería. En particular, la fricción se representa utilizando la analogía de un orificio.

Dos de los programas revisados utilizan esquemas implícitos en diferencias finitas. La principal ventaja de los métodos implícitos es su estabilidad para intervalos de tiempo grandes, lo que significa que permiten que el número de Courant sea mayor que 1. Sin embargo, desde el punto de vista computacional, los esquemas implícitos aumentan tanto el tiempo de ejecución como los requisitos de almacenamiento, y requieren una biblioteca de software especializada para la resolución de sistemas de ecuaciones lineales dispersas, ya que en el caso de modelos grandes se debe resolver un gran sistema de ecuaciones dispersas.

Desde una perspectiva matemática, los métodos implícitos no son la elección más adecuada para problemas de propagación de ondas, ya que pueden distorsionar la trayectoria de propagación de la información, lo que resulta en una representación incorrecta del fenómeno físico. Además, en cualquier caso, se requiere un intervalo de tiempo pequeño para garantizar la precisión en los problemas de transitorios rápidos. Por

[1] El número de Courant se define como el cociente entre la distancia que viaje la onda en un paso de tiempo del algoritmo de integración y la longitud de los nodos o volúmenes de control, expresado mediante la formula $Cr = a\,\Delta t/\Delta x$. Por lo general, los algoritmos de integración explícitos imponen que el número Courant deba ser menor o igual que 1 para garantizar la estabilidad del proceso de integración.

estos motivos, la mayor parte de los programas de transitorios hidráulicos utilizan métodos explícitos como el método de las características.

4.2.2. Aplicaciones de MSOO para la Simulación de Transitorios Hidráulicos

Trabajos Previos de Simulación de Transitorios Hidráulicos Usando Herramientas de MSOO

Las herramientas de MSOO se han desarrollado con base en resolvedores de ecuaciones diferenciales ordinarias y ecuaciones diferenciales algebraicas. Sin embargo, las ecuaciones que describen el golpe de ariete son ecuaciones en derivadas parciales de tipo hiperbólico. En el campo del MSOO, la aproximación convencional para resolver ecuaciones en derivadas parciales es mediante la aplicación del método de las líneas.

El método de las líneas (MOL, por sus siglas en inglés, Method of Lines) es una técnica numérica ampliamente utilizada para resolver ecuaciones en derivadas parciales. Se considera un método "semidiscreto" porque discretiza las ecuaciones en derivadas parciales en una dirección (el espacio) y mantiene la dimensión temporal como continua. Esto transforma las ecuaciones en un sistema de ecuaciones diferenciales ordinarias que puede ser resuelto utilizando las capacidades de las herramientas de MSOO.

La discretización en el espacio se puede realizar con diversas técnicas. Por ejemplo, es posible reemplazar las derivadas espaciales que aparecen en las ecuaciones en derivadas parciales por aproximaciones utilizando diferencias finitas. Otra opción es aplicar principios de conservación en volúmenes finitos o recurrir a métodos de elementos finitos para conseguir la discretización espacial.

En el capítulo 6 del libro de Cellier & Kofman (2006), titulado “Continuous System Simulation”, ilustra la aplicación del método de las líneas a ecuaciones en derivadas parciales de tipo parabólico, como la ecuación de la difusión de calor, y de tipo hiperbólico, como la ecuación de onda y las ecuaciones que describen el flujo unidimensional de un gas ideal. Además, se realiza un análisis exhaustivo de las características del método en términos de precisión y eficiencia. A continuación, se presentan las principales conclusiones.

- Características de las ecuaciones parabólicas:
  - La aplicación del método de las líneas en ecuaciones parabólicas conduce a ecuaciones diferenciales "rígidas".
  - La rigidez aumenta a medida que se refina la discretización espacial.
  - Las ecuaciones rígidas presentan escalas de tiempo muy dispares en su respuesta, pero las partes más rápidas de la respuesta se amortiguan rápidamente.

- Características de las ecuaciones hiperbólicas:
  - La aplicación del método de las líneas en ecuaciones hiperbólicas también da como resultado ecuaciones diferenciales ordinarias "rígidas".
  - Al igual que en las ecuaciones parabólicas, la rigidez aumenta con una discretización más refinada.
  - A diferencia de las ecuaciones parabólicas, en las ecuaciones hiperbólicas, las partes más rápidas de la respuesta no necesariamente se amortiguan rápidamente debido a la ubicación de los autovalores cerca del eje imaginario.
  - Su solución consiste en la superposición de varias ondas. Dichas ondas pueden propagar discontinuidades, e incluso pueden aparecer o surgir discontinuidades, como ondas de choque, a partir de respuestas inicialmente continuas.
- Rigidez de las ecuaciones diferenciales:
  - La rigidez en ecuaciones diferenciales se refiere a la existencia de escalas de tiempo significativamente diferentes en la respuesta del sistema.
  - Se calcula evaluando los autovalores del sistema linealizado que surge de las ecuaciones diferenciales en cuestión.
  - La relación entre el autovalor más grande y el más pequeño se utiliza para cuantificar la rigidez. Cuanto mayor sea esta relación, mayor será la rigidez del sistema.
  - En el caso de ecuaciones parabólicas resueltas con el método de las líneas, todos los autovalores tienden a ubicarse en la parte negativa del eje real, indicando una respuesta altamente amortiguada.
  - Sin embargo, en ecuaciones hiperbólicas, los autovalores pueden estar cerca del eje imaginario, lo que conlleva respuestas oscilatorias con una amortiguación pequeña.

Es importante subrayar que las herramientas de MSOO incluyen resolvedores especialmente diseñados para abordar ecuaciones diferenciales ordinarias con alta rigidez. Esto es esencial debido a que los sistemas de ecuaciones que resultantes de aplicar metodologías orientadas a objeto suelen ser bastante rígidos. Estos resolvedores permiten abordar de manera eficaz los sistemas de ecuaciones diferenciales ordinarias que surgen al aplicar el método de las líneas a ecuaciones parabólicas. La razón detrás de esta eficacia radica en que las escalas de tiempo más rápidas en la respuesta se amortiguan rápidamente, lo que facilita la obtención de soluciones precisas y estables.

Sin embargo, los resolvedores de ecuaciones diferenciales ordinarias rígidas no son igualmente adecuados para los sistemas de ecuaciones diferenciales ordinarias que

surgen de la aplicación del método de las líneas a ecuaciones hiperbólicas. En estos casos, la parte de la respuesta correspondiente a las escalas de tiempo más rápidas se amortigua de manera muy lenta. Como resultado, aunque el método de las líneas puede aplicarse a ecuaciones de tipo hiperbólico, no es la opción más idónea. Por lo general, los métodos completamente discretos diseñados específicamente para ecuaciones hiperbólicas suelen ofrecer una mayor precisión y requieren menos tiempo de cálculo.

El flujo transitorio de líquido en tuberías se describe mediante las ecuaciones en derivadas parciales (4-1) y (4-2), que son de tipo hiperbólico. En consecuencia, el método de las líneas, que es el enfoque natural al utilizar una herramienta de MSOO, se ve impactado por las limitaciones previamente mencionadas cuando se aplica a ecuaciones hiperbólicas.

Aplicaciones Previas de Herramientas de MSOO a Transitorios Hidráulicos

En esta investigación, se ha realizado una búsqueda selectiva de trabajos previos de la aplicación de herramientas de Modelado y Simulación de Sistemas Orientado a Objetos (MSOO), en particular Modelica y EcosimPro, para la simulación de transitorios hidráulicos.

En términos generales, la casi totalidad de las aplicaciones de Modelica y EcosimPro para la simulación de transitorios hidráulicos, salvo algunas excepciones que se discutirán más adelante, han empleado métodos semidiscretos. Estos métodos discretizan únicamente la coordenada espacial, transformando así el sistema de ecuaciones en derivadas parciales en un sistema de ecuaciones diferenciales ordinarias.

Estas aplicaciones han utilizado dos tipos de métodos semidiscretos: las *mallas desplazadas* (staggered grids) y los *esquemas de corriente arriba* (upwind schemes), que se describen brevemente a continuación.

Mallas desplazadas (staggered grids)

En el método de discretización centrada con mallas desplazadas, la tubería se divide en n volúmenes, y se determina el estado termodinámico en cada uno de ellos mediante la aplicación del principio de conservación de la masa y de la energía. Esto se lleva a cabo asumiendo que se conocen los flujos de masa en los límites entre estos volúmenes. Para calcular estos flujos de masa entre los volúmenes de la malla anterior, se emplea el principio de conservación del momento en una segunda malla desplazada en medio volumen con respecto a la primera, como se muestra en la Figura 4-2.

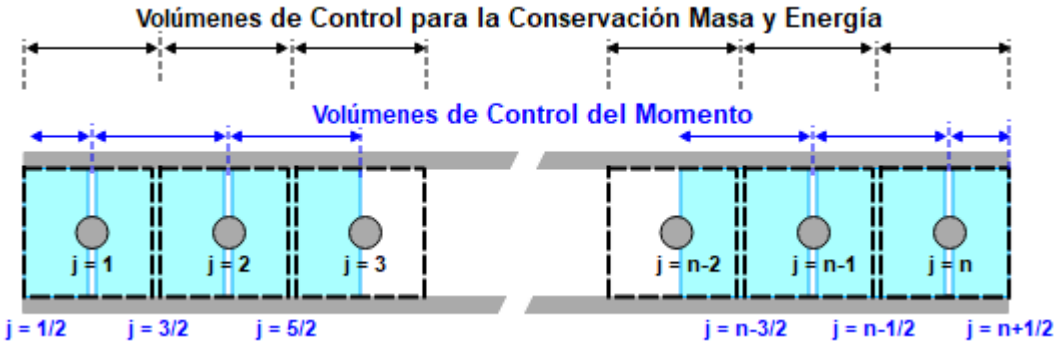


Figura 4-2: Discretización Unidimensional con Mallas Desplazadas

El esquema centrado con mallas desplazadas ofrece una aproximación de primer orden, lo que significa que no es un esquema de alta precisión. Sin embargo, presenta la ventaja de requerir menos tiempo de cálculo en comparación con otros esquemas de orden superior. Esta eficiencia en el tiempo de cálculo permite la utilización de nodalizaciones más detalladas, lo que a su vez compensa la limitación inherente al bajo orden de este método.

Esquemas de Corriente Arriba (Upwind scheme)

Los esquemas de corriente arriba son un conjunto de métodos utilizados para la resolución numérica de ecuaciones en derivadas parciales de tipo hiperbólico. Estos esquemas requieren expresar las ecuaciones en derivadas parciales de tipo hiperbólico en la forma conservativa que se muestra a continuación:

$$\frac{\partial \vec{U}}{\partial t} + \frac{\partial \vec{F}}{\partial x} = \vec{S} \tag{4-9}$$

Donde $\vec{U}$ son las variables de estado, $\vec{F}$ son los flujos y $\vec{S}$ son los términos fuente.

Por ejemplo, las ecuaciones (4-1) y (4-2) que describen el fenómeno del golpe de ariete, pueden reformularse en la forma conservativa que se muestra a continuación:

$$\frac{\partial}{\partial t}\begin{bmatrix} H \\ Q \end{bmatrix} + \frac{\partial}{\partial x}\begin{bmatrix} \frac{a^2}{g} Q \\ gA\ H \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{f}{2D}\frac{Q|Q|}{A} - k\left(\frac{\partial Q}{\partial t} + a\ \mathrm{sign}(Q)\left|\frac{\partial Q}{\partial x}\right|\right) \end{bmatrix} \tag{4-10}$$

Donde se ha supuesto que el área de la sección transversal (A) y la velocidad de onda (a) son constantes a lo largo de la tubería.

En los esquemas aguas arriba se calculan valores medios para las variables de estado en los nodos o volúmenes de control de una malla única definida según se indica en la Figura 4-3.

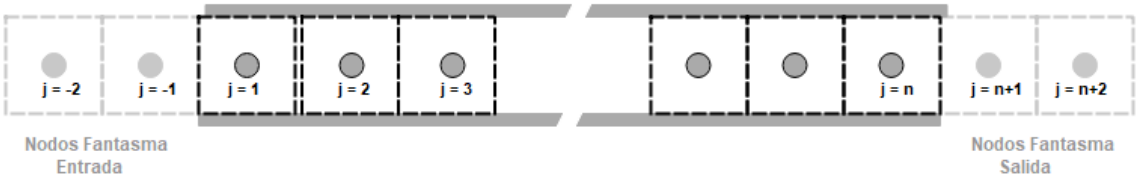


Figura 4-3: Discretización Unidimensional de un Esquema Aguas Arriba

Los esquemas de corriente arriba requieren la reconstrucción de la distribución espacial de las variables de estado en función de los valores promedio calculados en los nodos. La reconstrucción más sencilla es suponer que las variables de estado son constantes dentro de cada volumen y toman el valor promedio calculado en dicho volumen. Ahora bien, esta reconstrucción constante a tramos proporciona una aproximación de solo primer orden. Generalmente, los esquemas de corriente arriba utilizan métodos de reconstrucción más sofisticados, como la reconstrucción lineal, la parabólica y la discontinua. Estos métodos de reconstrucción más sofisticados conducen a métodos de orden mayor que uno, por tanto, ofrecen una mayor precisión, pero son más complejos y requieren mayor tiempo de cálculo.

Además, las reconstrucciones de orden elevado, requieren además el conocimiento de los valores de las variables de estado en los nodos anteriores y posteriores a aquel en que se realiza la reconstrucción, pero en los extremos de la tubería, dichos nodos no existen. La técnica habitual para eludir esta dificultad es considerar un par de nodos ficticios en cada extremo de la tubería, denominados nodos fantasma (ver Figura 4-3), y rellenar las propiedades en dichos nodos extrapolando las propiedades en los nodos interiores. Lógicamente, el orden del método puede sufrir debido a esta extrapolación en las zonas próximas a los extremos.

Clasificación de las Aplicaciones de Herramientas de MSOO a Golpe de Ariete

En los últimos años, se ha observado un aumento significativo en la cantidad de trabajos relacionados con la aplicación de herramientas de MSOO a la simulación de transitorios hidráulicos. En la Tabla 4-3, se presentan los trabajos que se han considerado más relevantes. En el caso de autores que han publicado múltiples trabajos sobre un mismo tema, se han seleccionado los más destacados. Esta tabla también proporciona información sobre la herramienta de MSOO utilizada, el método empleado y las principales características de la librería de componentes utilizada. Aunque esta tabla no pretende ser exhaustiva, ya que algunos de los autores mencionados han contribuido con varios trabajos, la inclusión de uno de ellos sirve como ilustración de las aplicaciones típicas desarrolladas hasta el momento."

Tabla 4-3: Aplicaciones de Herramientas de MSOO a la simulación de Transitorios Hidráulicos

Referencias	Herramienta	Librería	Método Semidiscreto	Principales Características de la Librería
Lindblom, 2015	Modelica Dymola	Modelica.Fluid	Mallas desplazadas (1er orden)	Librería genérica para sistemas termo-fluidos Soporta cualquier fluido de trabajo en fase líquida o gaseosa o ambas (bifásico)
Vytvytskyi and Lie 2019	Modelica OpenModelica	OpenHPL	Esquema aguas arriba (2º orden): • Kurganov Petrova	Librería diseñada para la simulación de centrales hidro-eléctricas. El fluido de trabajo es agua. Tipicamente, solo considera una tuberia discretizada por modelo, la que representa la tubería forzada (penstock) de una central hidro-eléctica
Koppel et al. 2011 Bombardieri, Traudt, and Manfletti 2019 Lema 2013	EcosimPro	ESPSS	Mallas desplazadas (1er orden) y Esquemas aguas arriba: • Roe, • Roe con límites, y • AUSM	Diseñada para simular motores cohete de combustible líquido. Soporta cualquier fluido de trabajo en fase líquida o gaseosa o ambas (bifásico)
Ren, Li, and Chen 2020	Modelica	Propia	Mallas desplazadas (1er orden)	Librería propia para la simulación de ciclos de motores cohete de combustión escalonada El fluido de trabajos es metano

Tal y como se demostrará en el siguiente apartado, los métodos semidiscretos resultan bastante inferiores al método de las características para el caso del golpe de ariete clásico representado por las ecuaciones (4.1) y (4.2). Esto plantea la pregunta de si ha habido intentos de desarrollar aplicaciones de MSOO para la simulación de transitorios hidráulicos que empleen el método de las características.

En esencia, se han identificado dos trabajos de máster (García-Alvárez, 2017; Hinna, 2021) que implementaron modelos de tuberías utilizando Modelica y aplicando el método de las características. Sin embargo, es relevante señalar que estos modelos consideraron exclusivamente una única tubería, sin abordar la complejidad de resolver redes de tuberías hidráulicas.

Además, se encontró una librería pública para la simulación de centrales termo-eléctricas (El-Hefni et al., 2011), que incluye un componente tubería que implementa el método de las características. El código fuente de dicha librería, disponible en el repositorio (*ThermoSysPro Repository*, n.d.), incluye la definición del componente mencionado, así como un caso de prueba bastante sencillo. Sin embargo, los intentos del autor por ejecutar este caso de prueba han resultado en tiempos de CPU excepcionalmente prolongados, a pesar de que el MOC se caracteriza por su eficiencia computacional. Esta inusual carga de tiempo de cómputo parece derivar de la falta de adecuada resolución de los problemas numéricos inherentes al emplear un solucionador discreto en una herramienta de Modelado y Simulación de Sistemas Orientado a Objetos (MSOO). Esta problemática se manifiesta incluso en el caso de prueba simple proporcionado con la librería.

Comparación entre una Aplicación de MSOO y una Herramienta Específica

Para llevar a cabo una comparación entre una simulación de transitorios hidráulicos utilizando una herramienta de MSOO y una herramienta específica de transitorios hidráulicos, es necesario, en el caso de la herramienta de MSOO, seleccionar una biblioteca hidráulica transitoria desarrollada por otros o desarrollar una propia. Con el objetivo de realizar una comparación autónoma y comprensible, se ha optado por desarrollar una pequeña biblioteca hidráulica utilizando EcosimPro y el método de mallas desplazadas centradas. El proceso de desarrollo de esta biblioteca se describe en el Apéndice C y también sirve para ilustrar los desafíos que implica aplicar una herramienta de MSOO a un problema de ecuaciones en derivadas parciales de tipo hiperbólico.

Como se mencionó anteriormente, existen dos enfoques para discretizar las ecuaciones del flujo transitorio: el centrado con mallas desplazadas y los esquemas aguas arriba. Surge la pregunta de si las conclusiones de esta comparación serían diferentes si se hubiera utilizado un esquema aguas arriba de mayor orden. La elección de utilizar el método centrado con mallas desplazadas se basa en comparaciones previas realizadas por Lema (2013) utilizando EcosimPro/ESPSS. En estas comparaciones, se analizaron casos de golpe de ariete particularmente complejos, como el llenado de líneas, y se

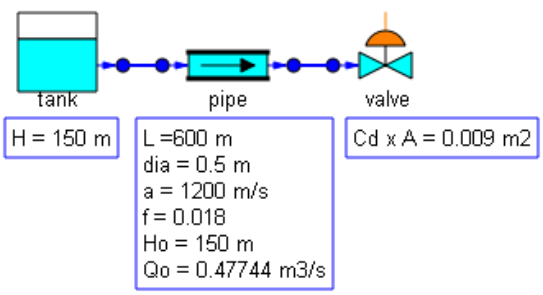
concluyó que el esquema centrado es más recomendable. Esto se debe a que el tiempo de CPU requerido por los esquemas aguas arriba implementados en EcosimPro/ESPSS es aproximadamente un orden de magnitud mayor que el del esquema centrado. Por lo tanto, para mejorar la precisión de los cálculos, resulta más eficiente aumentar el número de nodos en el esquema centrado que emplear un esquema aguas arriba de mayor orden.

A continuación, se presentan una serie de casos de prueba y se resuelven utilizando una herramienta específica de transitorios hidráulicos, AFT Impulse, una herramienta específica para transitorios hidráulicos que implementa el Método de las Características (MOC), y EcosimPro en conjunto con la librería descrita en el Apéndice C, que se utilizará con la herramienta de MSOO. Por razones de simplicidad, las soluciones obtenidas se denominan como 'MOC" y "MSOO", respectivamente.

Dichos casos de prueba se han ejecutado en un PC con un procesador Intel I7-10510U @ 1.80 GHz y con 16 GB de RAM. La versión de EcosimPro utilizada es la 6.2 junto con el compilador de C++ de GNU en su versión 4.9 de 64 bits.

Caso de Prueba 1

La Figura 4-4 muestra el caso de prueba número 1. El modelo consta de una tubería conectada a un depósito en el extremo aguas arriba y a una válvula de salida en el extremo aguas abajo. Inicialmente, la válvula está en posición abierta y cierra gradualmente según una ley de cierre se muestra en la figura. En esta ley, una posición igual a 1 indica que la válvula está completamente abierta, mientras que una posición igual a 0 indica que la válvula está completamente cerrada. En este caso, la velocidad de cierre de la válvula no es muy rápida, el cierre completo ocurre un poco después de que el primer frente de onda reflejado desde el depósito (2·L/a = 2s) haya alcanzado la válvula. Por lo tanto, dicha ley de cierre se podría designar como semirrápida.



Posición Válvula

$$pos = \begin{cases} 1 - (Time/2.1)^{\frac{3}{4}} & Time \leq 2.1\ s \\ 0 & Time > 2.1\ s \end{cases}$$

Duración = 6 s

Figura 4-4: Caso de Prueba 1

Para resolver este ejemplo, se han considerado 6 secciones en la tubería en el caso del MOC y 20 volúmenes de control o nodos en el caso del MSOO.

La Figura 4-5 muestra los resultados obtenidos para el caso de prueba no 1. Es evidente que los resultados calculados por la herramienta específica de transitorios y por la herramienta de MSOO son muy similares. La única diferencia perceptible es que el primer pico de presión es ligeramente menor con la aplicación de MSOO en comparación con la herramienta específica utilizando el MOC. La razón de esta diferencia radica en que en el caso del MOC, se representa la presión justo al final de la tubería en la sección de entrada de la válvula, mientras que en el caso del MSOO, la presión representada corresponde al último nodo. Esta presión no coincide exactamente con la de la sección de salida, ya que el centro del último nodo está ubicado a una distancia de 1/40 de la longitud total desde el punto final.

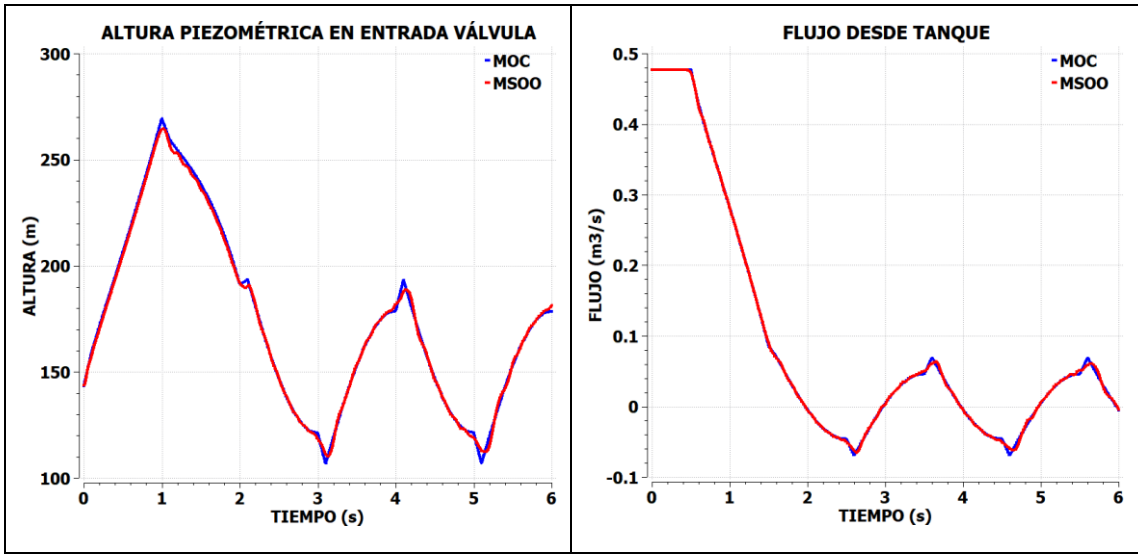
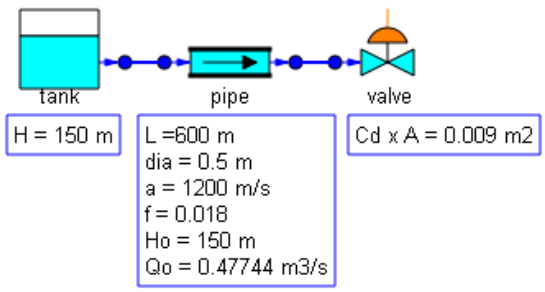


Figura 4-5: Resultados del Caso de Prueba 1

Caso de Prueba 2

El segundo caso de prueba considera un modelo idéntico al del primer caso, pero con un cierre instantáneo de la válvula. La válvula está inicialmente abierta al 30% y cierra instantáneamente en el tiempo 0.001 ms. Un cierre en un tiempo menor que el de ida y vuelta de una onda en la tubería puede designarse como rápido.



Posición Válvula

$$pos = \begin{cases} 0.30 & Time \leq 10^{-3}\ s \\ 0. & Time > 10^{-3}\ s \end{cases}$$

Duración: 12 s

Figura 4-6: Caso de Prueba 2

Los resultados para la altura piezométrica en el nodo o sección más cercano a la válvula calculadas con una herramienta específica de transitorios utilizando el MOC y con una aplicación de MSOO se muestran en la Figura 4-7. En esta figura, se observa que la solución obtenida mediante la aplicación de MSOO con un método semidiscreto no logra seguir de manera adecuada los cambios en forma de escalón de la presión. Se evidencia la dispersión de las ondas de presión y la aparición fluctuaciones u oscilaciones no físicas después del paso de la onda. Una solución común a este tipo de problemas numéricos es introducir un término de viscosidad artificial, cuya magnitud se controla mediante un coeficiente de amortiguamiento proporcionado como dato.

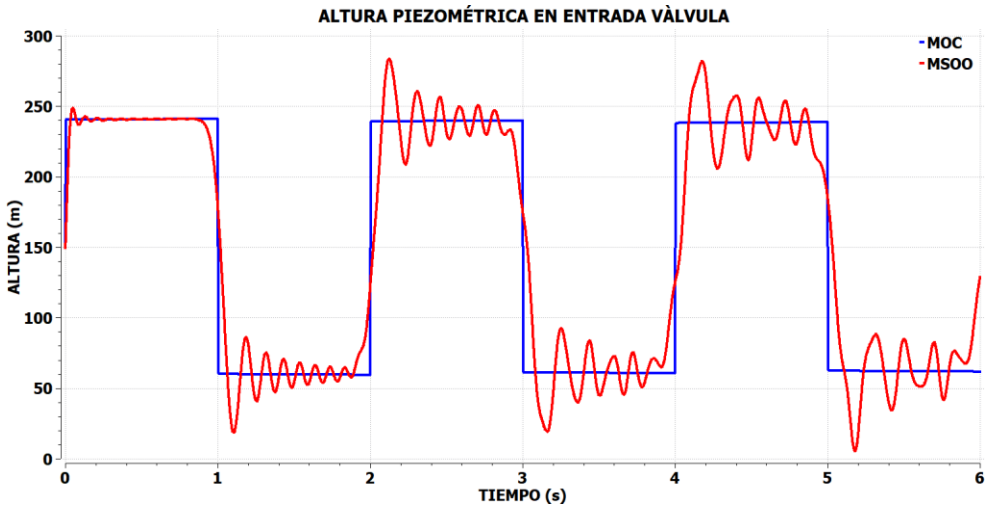


Figura 4-7: Resultados del Caso de Prueba 2 (sin viscosidad artificial)

La Figura 4-8 presenta los resultados obtenidos para el caso 2 utilizando distintos valores del factor de amortiguamiento (`kdamp`), el cual multiplica el término de viscosidad artificial. Se puede observar que con `kdamp` igual a `0.05` y `0.01`, la sobreoscilación no se elimina por completo. Para este caso específico, se determina que el valor más adecuado de `kdamp` es aproximadamente `0.15`, ya que es el que logra eliminar la sobreoscilación de manera efectiva, mientras que con `kdamp` igual a `0.20`, se introduce un nivel de amortiguamiento mayor de lo estrictamente necesario.

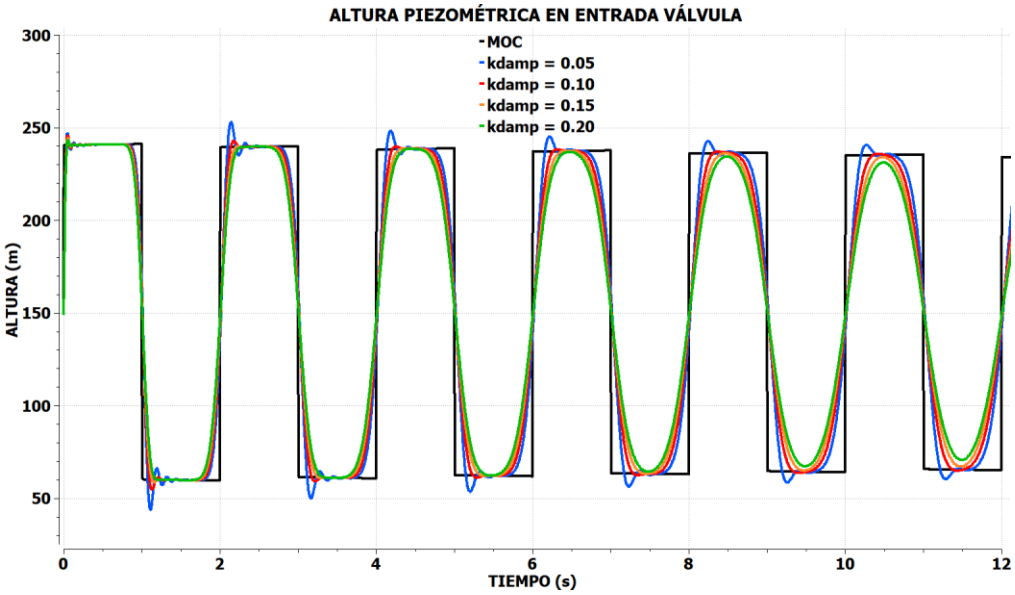


Figura 4-8: Resultados del Caso de Prueba 2 - Comparación de la Viscosidad Artificial

En referencia a la presentación de la Figura 4-5 , que exhibe los resultados del caso de prueba 1, se ha señalado que la variable representada como altura piezométrica en V6 para el caso del MSOO corresponde a la altura del último nodo. Esta altura es representativa para el punto central del último nodo de los 20 en que se divide la tubería. Esta observación plantea una pregunta relevante: ¿por qué no se consideró la altura piezométrica en el extremo final de la tubería?

Con el fin de responder a la cuestión anterior, la Figura 4-9 muestra la alturas piezométricas calculadas mediante MSOO, con 20 nodos y coeficiente de amortiguamiento (kdamp) igual a 0.15, en dos puntos: el último nodo de la tubería (variable p1.H[20]), y el extremo final de la tubería (variable p1.h_out.H). Se observa claramente que la altura piezométrica en el extremo final muestra un pico no físico en el momento inicial. Este fenómeno se origina debido a la concentración de la inercia distribuida del último volumen de control para la ecuación del momento en un único elemento. La ecuación del momento para este último elemento o tramo, que se justifica en el Apéndice C, es la siguiente:

$$\frac{dQ_{n+1}}{dt} = -\frac{A\,g * (H_{out} - H_n)}{(\Delta x/2)} - f\frac{Q_{n+1}|Q_{n+1}|}{2\,D\,A} \tag{4-11}$$

donde Q_{n+1} es el flujo volumétrico de salida, H_n es la altura del ultimo nodo, que es variable de estado y por tanto no puede cambiar de forma instantánea, y H_{out} es la altura en el extremo de salida.

En el escenario en el que Q_{n+1} está impuesto por la condición de contorno, es decir, por el elemento situado al final de la tubería, su derivada puede alcanzar valores extremadamente

altos durante maniobras rápidas. Despejando en la ecuación anterior, la altura piezométrica en el extremo de la tubería se obtiene:

$$H_{out} = H_n - \frac{\left[\left(\frac{\Delta x}{2}\right)\frac{dQ_{n+1}}{dt} + f\frac{Q_{n+1}|Q_{n+1}|}{2\,D\,A}\right]}{A\,g} \tag{4-12}$$

Como puede observarse, en situaciones de una detención instantánea del flujo, dQ_{n+1}/dt tiende a $-\infty$, y la presión o altura en el final de la tubería H_{out} tiende a ∞. Sin embargo, es sabido que el máximo incremento de presión debido a una maniobra instantánea es finito y puede calcularse con la fórmula de Joukowsky (Wylie et al., 1993). (Chaudhry, 2014; Wylie et al., 1993). Esto conduce a la conclusión de que el MSOO junto con el método semidiscreto considerado, no puede proporcionar un valor físicamente realista para la altura piezométrica o la presión en situaciones de maniobras rápidas que ocurran en un intervalo de tiempo igual o menor que el intervalo de tiempo correspondiente al número de Courant igual a 1.

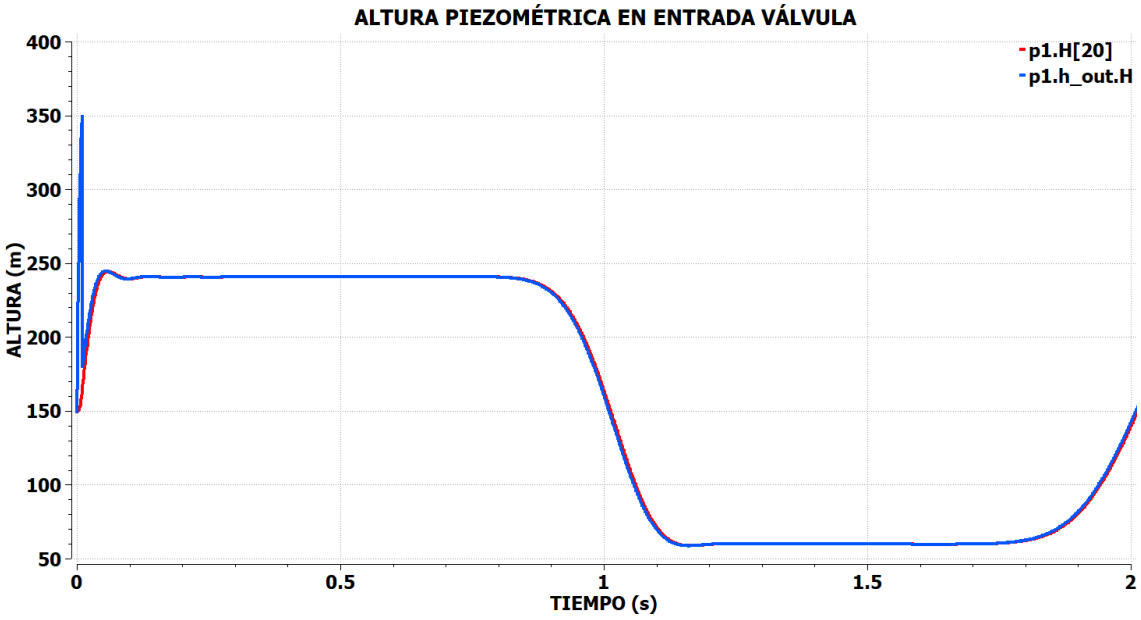
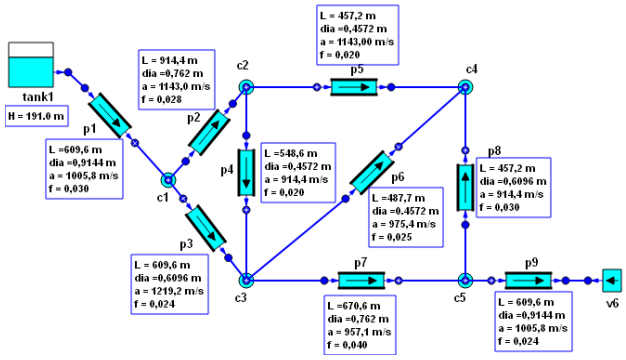


Figura 4-9: Resultados del Caso de Prueba 2 en Último Nodo y en Extremo Final

Caso de Prueba 3

El tercer caso de prueba se ha tomado de Merilo (1992) y consiste en una red de 9 tuberías con un depósito a un lado (`tank1`) y una salida de caudal impuesta (`v6`). la Figura 4-10, se muestra el modelo, la tabla con los datos de las tuberías, la condición de caudal impuesta y la duración de la simulación.



Tubo No	Longitud *m*	Diámetro *m*	Factor Fricción	Velocidad Onda *m/s*
p1	609.60	0.91440	0.030	1005.84
p2	914.40	0.76200	0.028	1143.00
p3	609.60	0.60960	0.024	1219.20
p4	548.64	0.45720	0.020	914.40
p5	457.20	0.45720	0.020	1143.00
p6	487.68	0.45720	0.025	975.36
p7	670.56	0.76200	0.040	957.07
p8	457.20	0.60960	0.030	914.40
p9	609.60	0.91440	0.024	1005.84

$$Q_{V6} = \begin{cases} 0.8495\ m^3/s & Time \leq 10^{-3}\ s \\ 0.\ \ m^3/s & Time > 10^{-3}\ s \end{cases}$$

Duración = 50 s

Figura 4-10: Caso de Prueba 3

Este caso es más representativo de las e aplicaciones del mundo real que los dos casos anteriores, ya que involucra una red de tuberías en lugar de una única tubería. Por lo tanto, se ha utilizado este caso para llevar a cabo una comparación más exhaustiva de la precisión y los tiempos de cálculo entre ambas categorías de herramientas: las específicas para transitorios hidráulicos y las basadas en MSOO.

Cuando se construye el modelo de este caso con una herramienta específica para transitorios hidráulicos, el usuario solo tiene que especificar el número de segmentos en que desea dividir la tubería con el tiempo de transito de onda más corto, y el programa se encarga de calcular el número de nodos en cada tubería y el incremento de tiempo. Esta división automática de las tuberías busca que los tiempos de transito de onda en todos los segmentos en que se dividen las tuberías sean iguales, lo que garantiza la máxima precisión.

En la aplicación MSOO usando el método semidiscreto, el número de nodos en los que se divide cada tubería es un dato intrínseco que desempeña un papel crucial en la precisión de los cálculos. La selección de una nodalización muy detallada en algunas tuberías y menos detallada en otras tiene un impacto significativo en la precisión general de los resultados. En última instancia, las tuberías con una nodalización menos detallada serán las que limiten la precisión del cálculo global. Por lo tanto, es esencial reconocer que la selección del número de nodos en la aplicación MSOO puede dar lugar a la introducción de errores.

Con el objetivo de permitir una comparación equitativa entre ambas herramientas, se llevó a cabo un cálculo inicial del número de nodos en cada tubería utilizando una herramienta especializada, de las que emplean del Método de las Características (MOC). En el cálculo realizado con la aplicación de MSOO utilizando un método semidiscreto, se emplearon mismas discretizaciones que las calculadas previamente para el MOC.

Los resultados de este cálculo inicial de la nodalización con una herramienta que utiliza el MOC, se detallan en la Tabla 4-4, que muestra el número de nodos en cada tubería y el intervalo de integración en función del número de nodos especificados en la tubería con el tiempo de tránsito de onda más corto o tubería de control, es decir, la tubería número 5.

Tabla 4-4: Nodalización de las Tuberías del Caso 3 usando el MOC

No nodos tubería control	No. Secciones									Incremento Tiempo (s)
	Tub. 1	Tub. 2	Tub. 3	Tub. 4	Tub. 5	Tub. 6	Tub. 7	Tub. 8	Tub. 9	
4	6	8	5	6	4	5	7	5	6	0.10050
8	12	16	10	12	8	10	14	10	12	0.05025
16	24	32	20	24	16	20	28	20	24	0.02513
32	48	64	40	48	32	40	56	40	48	0.01256
64	97	128	80	96	64	80	112	80	97	0.00625
128	194	256	160	192	128	160	224	160	194	0.00313

El motivo por el que en la tabla anterior se consideran únicamente nodalizaciones múltiples de cuatro en la tubería de control, dado que es sencillo comprobar que, para este modelo, el método de las características requiere ajustes mínimos en la velocidad de onda de las tuberías bajo dicha condición.

Los resultados de las alturas piezométricas calculadas en el punto V6, tanto utilizando el MOC como el MSOO, se presentan en la Figura 4-11 y en la Figura 4-12, La Figura 4.11 detalla los resultados para los primeros 6 segundos. Para el MOC, solo se muestran los resultados con una nodalización de 4 nodos en la tubería de control, dada su alta precisión. Las curvas adicionales representan las alturas piezométricas calculadas con EcosimPro utilizando el método semidiscreto con 2, 4, 8, 16, 32, 64 y 128 nodos en la tubería de control.

Es destacable observar que los resultados obtenidos mediante MSOO muestran una amortiguación significativa de los picos debidos a reflexiones, por ejemplo, los picos que ocurren a 3 y 5 s. A medida que aumenta el número de nodos en la tubería de control, el nivel de amortiguamiento disminuye, aunque incluso con 128 nodos, aún se observa un amortiguamiento muy notable.

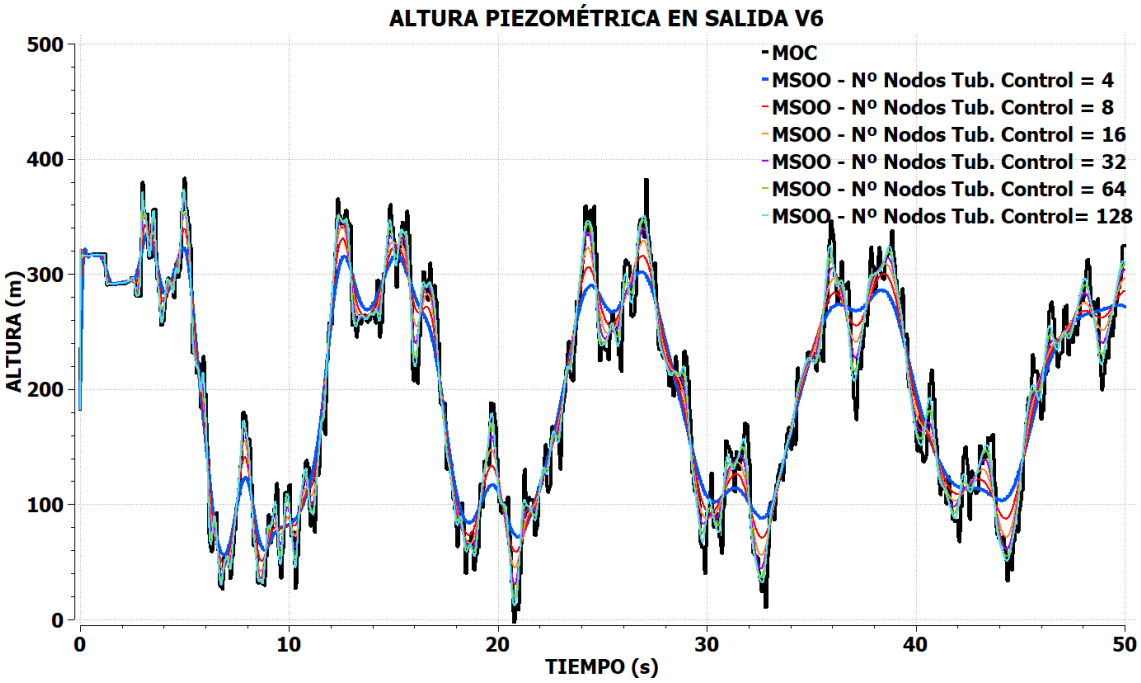


Figura 4-11: Resultados del Caso de Prueba 3 – Altura Piezométrica en V6

Tercer Caso de Prueba

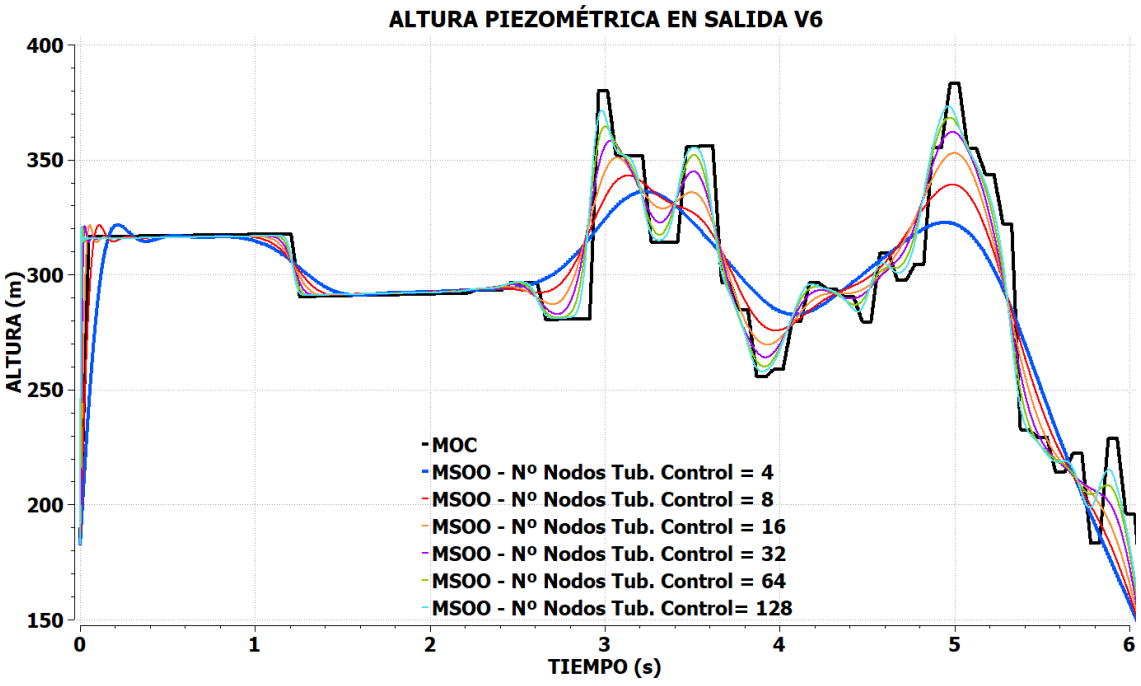


Figura 4-12: Detalle de Resultados del Caso de Prueba 3 - Altura Piezométrica en V6

La Tabla 4-5 muestra los valores de los tiempos de computador, la máxima altura piezométrica en la válvula y el error en dicha altura que se obtienen con ambos métodos, el MOC y el MSOO, para las diversas nodalizaciones. El error de la máxima altura en V6 se mide utilizando siguiente ecuación:

$$error = \frac{H_{V6,\ max} - H_{tank}}{H_{V6,\ ref} - H_{tank}} - 1 \qquad (4\text{-}13)$$

donde:

$H_{V6,\ max}$ es la máxima altura calculada en la válvula.

H_{tank} es la altura del tanque, igual a 191 m.

$H_{V6,\ ref}$ es la máxima altura de referencia en la válvula V6, estimada a partir de un cálculo muy preciso con 512 nodos en la tubería de control, y se establece en 382.198 m.

Es relevante señalar que en el caso del método MOC, el cálculo inicial del estado estacionario es prácticamente instantáneo debido a que involucra solamente 3 variables desconocidas en las tuberías (las dos alturas piezométricas en los extremos y el flujo). En contraste, el método MSOO presenta un aumento en el número de incógnitas en cada tubería es $2n + 4$, donde n representa el número de nodos internos. Este incremento del número de incógnitas, combinado con la falta de un solucionador de estacionarios para ecuaciones dispersas en EcosimPro, resulta en un incremento significativo en los tiempos de cálculo del estado estacionario. Por esta razón, en el caso del MSOO, se reportan tanto el tiempo de cálculo del estado estacionario como el tiempo de integración.

Tabla 4-5: Precisión y Tiempos de Cálculo del MOC y del MSOO para el Caso 3

	AFT Impulse Herramienta Específica utilizando MOC			**PipeliqTran Aplicación MSOO utilizando método Semidiscreto**			
No. Nodos Tubería Control	**Tiempo CPU** (s)	**Max. Altura en V6** (m)	**Error** (%)	**Tiempo CPU Estacion.** (s)	**Tiempo CPU Integración** (s)	**Max. Altura en V6** (m)	**Error** (%)
4	0.095	383.468	0.66%	0.041	0.490	336.123	-24.10%
8	0.19	383.187	0.52%	0.149	0.753	343.038	-20.48%
16	0.38	383.161	0.50%	1.059	1.549	352.907	-15.32%
32	0.93	383.150	0.50%	5.551	8.475	362.091	-10.52%
64	2.90	382.129	-0.04%	34.81	24.80	368.244	-7.30%
128	9.65	382.125	-0.04%	279.20	81.55	373.296	-4.66%

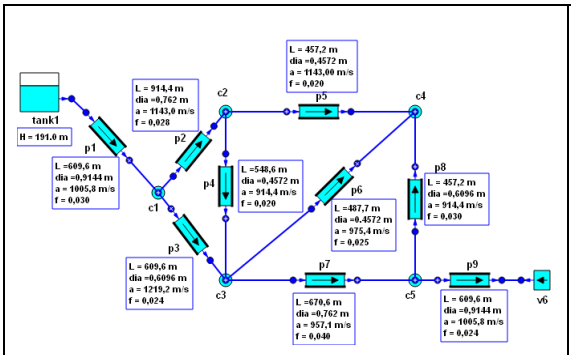
A la vista de los valores que se presentan en la tabla se pueden dibujar las siguientes conclusiones:

- El MOC muestra una precisión notable en la estimación de la altura en la válvula V6, con errores muy bajos. Incluso con solo 4 nodos en la tubería de control, el error es de solo el 0.66%.
- El MSOO proporciona resultados menos precisos, con errores en el rango de -24.10% a -4.66%, y tiende a subestimar la altura en comparación con la referencia, lo que se refleja en valores negativos de error.
- El MOC muestra tiempos de cálculo relativamente bajos, en el orden de milisegundos. A igualdad en el número de nodos, el MSOO tarda varias veces más que el MOC.
- El tiempo de cálculo del estacionario con el MSOO se dispara al aumentar el número de nodos.
- En el caso de transitorios rápidos como el analizado, el método semidiscreto utilizado con la herramienta de MSOO es mucho menos preciso y requiere mucho más tiempo de computación en el caso de transitorios rápidos que el MOC. Es relevante señalar, que incluso con un tiempo de computación de 279.2 segundos, el método semidiscreto no consigue la precisión que obtiene el MOC con tan solo 0.095 segundos de cálculo.

Caso de Prueba 4

El cuarto caso considera el mismo sistema que se estudió en el caso 3. Sin embargo, en lugar de abordar una detención rápida del caudal, se analiza una extracción de caudal durante un período de dos horas, la cual se repite de manera periódica cada 6 horas a lo largo de un día completo. Es importante destacar que las maniobras de cambio de flujo, tanto la subida como la detención, son relativamente lentas y tienen una duración de aproximadamente 100 segundos. Esto implica una simulación de larga duración que permite evaluar el comportamiento de los métodos con transitorios de larga duración y maniobras lentas.

Las fórmulas que definen la ley de extracción de flujo durante el primer periodo se muestran en la Figura 4-13, y la gráfica de la extracción de flujo a lo largo del día simulado se muestra en Figura 4-14.



$$Q_{V6} = \begin{cases} 0.8495\frac{1}{2}\left(1-\cos\left(\pi\frac{Time}{100}\right)\right) & 0 < Time < 100 \\ 0.8495 & 100 < Time < 7200 \\ 0.8495\frac{1}{2}\left(1+\cos\left(\pi\frac{(Time-1800)}{100}\right)\right) & 7200 < Time < 7300 \\ 0 & 7300 < Time < 6\times 3600 \end{cases}$$

Duración = 86400 s

Figura 4-13: Caso de Prueba 4

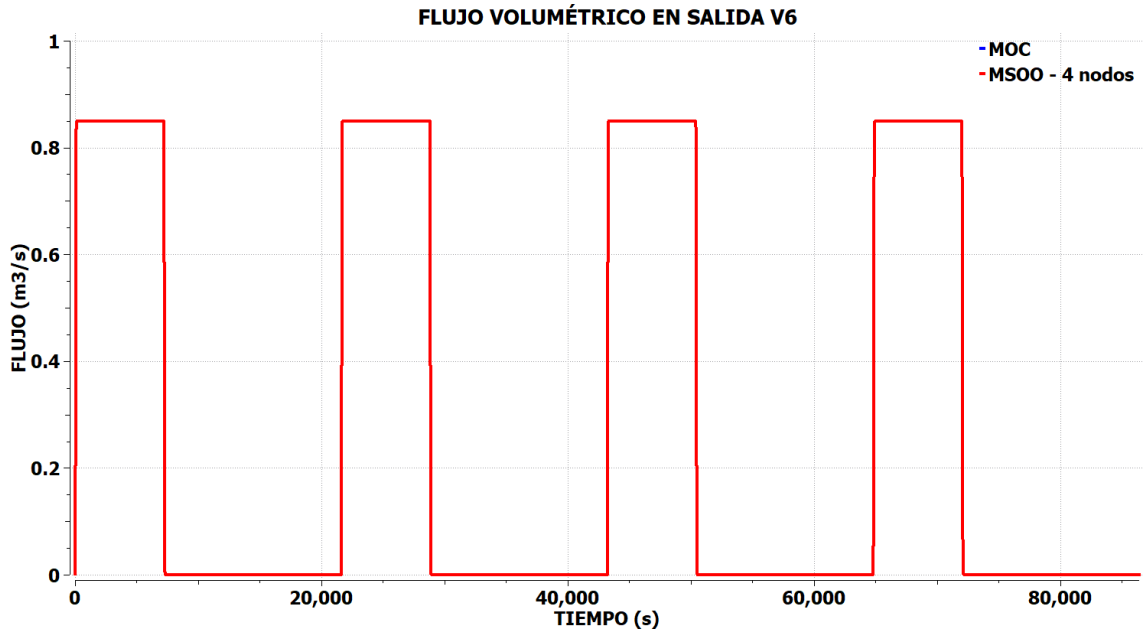


Figura 4-14: Ley de Flujo Impuesta en el Caso de Prueba 4

Los resultados para la altura piezométrica en V6 se presentan en la Figura 4-15. En esta figura, se puede observar que tanto el MOC como el MSOO son capaces de calcular de manera bastante precisa los picos de presión. Dichos picos son de pequeña magnitud para este caso, el máximo es aproximadamente 197 m y el mínimo 180 m. Esto se debe a que la duración de las maniobras de cambio de flujo, que es de100 segundos, es lenta para este sistema, lo que a su vez contribuye a la generación de picos de presión de menor magnitud.

La figura también muestra que el sistema presenta oscilaciones de larga duración (debido a su amortiguamiento muy pequeño) después de ser sometido a una maniobra. Es relevante destacar que el MSOO calcula una amortiguación significativamente mayor en comparación con el MOC. No obstante, es necesario subrayar que el MOC exhibe una mayor precisión en la resolución de las ecuaciones, así que esta disparidad implica que el MSOO introduce un efecto de amortiguamiento numérico.

La Figura 4-16 muestra los tiempos de computación requeridos por ambos métodos en función del tiempo de simulación. Es evidente que, para transitorios de este tipo, el MSOO puede requerir tiempos de computación mucho menores que los del MOC. Esto se debe a que los resolvedores para ecuaciones diferenciales rígidas implementados en las herramientas de MSOO permiten el uso de pasos de integración mayores que el paso correspondiente al número de Courant igual a 1, que es típico de los métodos explícitos como el MOC.

En definitiva, para transitorios de larga duración y con maniobras lentas, el MSOO requiere menos tiempo que el MOC, y calcula los picos de forma precias, pero muestra un amortiguamiento numérico bastante alto. Por ejemplo, el utilizar el MSOO para el diseño de un control hidráulico pudiese ser peligroso, ya que el método puede aumentar de forma artificial el amortiguamiento del sistema.

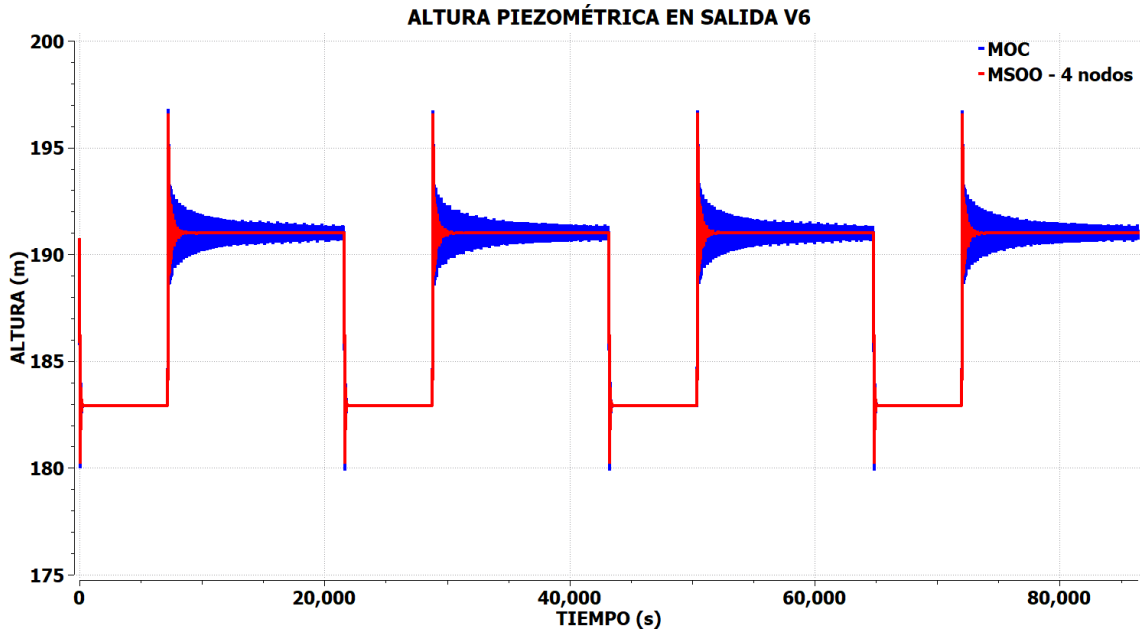


Figura 4-15: Resultados del Caso de Prueba 4 – Altura Piezométrica en V6

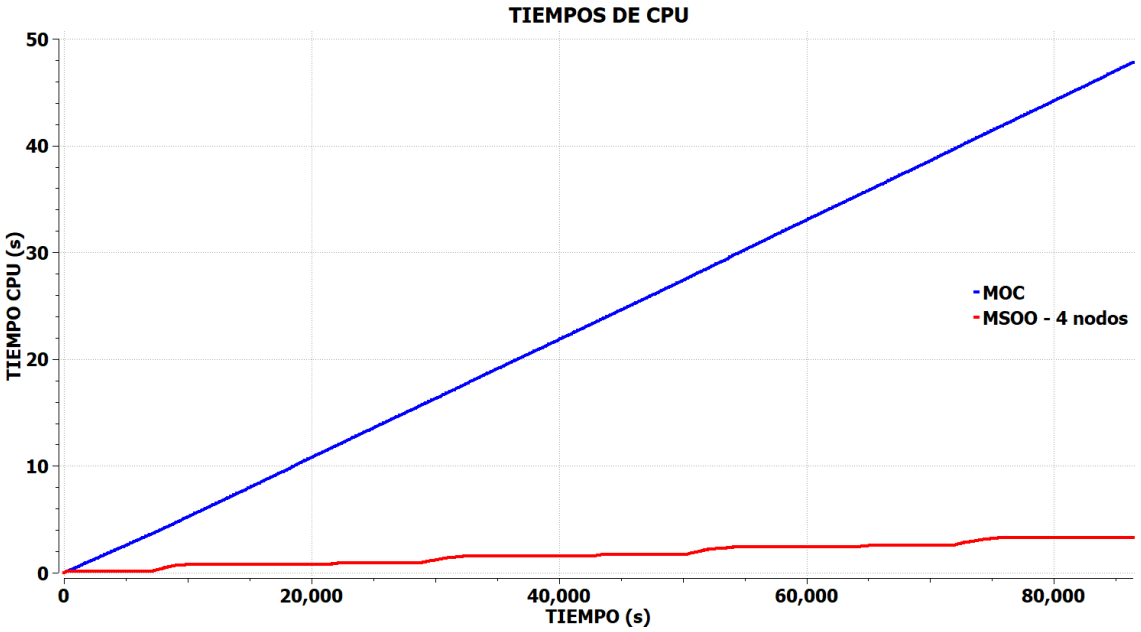


Figura 4-16: Resultados del Caso de Prueba 4 – Tiempos de CPU

Resumen de Comparación entre MSOO y Herramientas de Transitorios Hidráulicos

En el análisis comparativo entre el MSOO y las herramientas específicas de transitorios hidráulicos, se destacan las siguientes ventajas y desventajas:

- Precisión:
    - El MOC se caracteriza por su alta precisión y eficiencia, especialmente en maniobras rápidas con transitorios de corta o media duración.
    - El MSOO muestra precisión en el caso de maniobras lentas, pero ofrece menos precisión en maniobras rápidas. Sin embargo, se destaca por su eficiencia en transitorios de larga duración con maniobras lentas, lo que se traduce en un menor consumo de tiempo computacional.
- Nodalización no automática.
    - El MOC automatiza la discretización de las tuberías, simplificando el proceso para el usuario,
    - En contraste, el MSOO requiere que el usuario especifique el número de nodos a considerar en cada tubería del modelo. Esto abre la posibilidad de que se empleen nodalizaciones inconsistentes, nodos muy pequeños en algunas zonas y demasiado grandes en otras.
- Aparición de Picos no físicos.
    - El MOC tiene la capacidad de analizar maniobras instantáneas y proporciona valores que tienen una base física sólida. El incremento de presión de una maniobra instantánea viene dado por la fórmula de Joukowsky.
    - Por otro lado, el MSOO puede generar picos de presión sin fundamento físico cuando se aplican maniobras (cambios de flujo) en intervalos de tiempo más cortos que el de Courant.
- Calculo de Estacionarios:
    - Las herramientas específicas de transitorios hidráulicos calculan eficientemente el estacionario al considerar solo tres variables desconocidas por tubería y emplear resolvedores de tipo disperso.
    - En contraste, el MSOO enfrenta dificultades para calcular estados estacionarios en modelos hidráulicos con un gran número de nodos, dado que involucra dos variables desconocidas por nodo. Además, no todas las herramientas de MSOO incorporan resolvedores dispersos para el cálculo del estacionario. En ocasiones, la única alternativa para alcanzar el estado estacionario es ejecutar un transitorio ficticio manteniendo condiciones de contorno constantes.

- Falta de modelos especiales:
    - Por último, un problema notable del MSOO radica en la carencia de modelos específicos para algunos fenómenos particulares que aparecen en los transitorios hidráulicos, como la cavitación transitoria con separación de columna y la fricción transitoria. Estos fenómenos requieren modelos u algoritmos adicionales más allá del método de integración básica.
    - Muchos de los algoritmos desarrollados para abordar estos fenómenos específicos se basan en la premisa de que el método básico de solución es el MOC y de que existe un “quanto” de tiempo, que es el tiempo que una onda tarda en atravesar un nodo. Esto elimina la necesidad de subdividir el tiempo en intervalos más pequeños.
    - En cambio, cuando se utiliza MSOO con métodos semidiscretos, el tiempo no puede ser considerado de forma granular, y la conversión o adaptación de los algoritmos para abordar los fenómenos indicados suele requerir la detección de eventos o cruces. En este contexto, existe la posibilidad de que ocurran múltiples eventos de manera consecutiva, lo que ralentiza el proceso de integración y, en algunos casos, puede dar lugar a fenómenos de "chattering" que suponen un drástico obstáculo para el avance de la simulación.

En conclusión, los modelos semidiscretos de los MSOO son adecuados a simulaciones de larga duración con maniobras lentas, pero no son competitivos con los métodos tradicionales en caso de maniobras rápidas y hace falta un conocimiento muy bueno del fenómeno para construir el modelo e interpretar los resultados debido a que la nodalización es responsabilidad del usuario y a la posible aparición en la solución de picos no físicos.

5. UN NUEVO PARADIGMA PARA DESARROLLAR HERRAMIENTAS DE SIMULACIÓN

5.1. INTRODUCCIÓN

En este capítulo se presenta un nuevo paradigma para el desarrollo de aplicaciones utilizando EcosimPro, una herramienta de MSOO, con el propósito de ofrecer una respuesta afirmativa a la siguiente pregunta:

> ¿Es factible construir aplicaciones de simulación para campos específicos utilizando herramientas de Modelado y Simulación Orientadas a Objeto que sean igual de eficaces y amigables que los programas de simulación para campos específicos desarrollados utilizando lenguajes de programación genéricos?

En lugar de abordar un nuevo problema de simulación intentando formularlo mediante ecuaciones algebraico-diferenciales, el nuevo paradigma propone utilizar formulaciones y algoritmos de solución idóneos para el problema específico que se trata de resolver. Este enfoque implica llevar a cabo un análisis exhaustivo del problema con el objetivo de identificar las formulaciones y métodos más apropiados para la resolución del problema a simular.

Dadas las marcadas diferencias entre las características de la simulación estacionaria y la transitoria, se aborda de manera independiente la aplicación de la nueva aproximación o paradigma a cada uno de estos tipos de simulación. La aproximación propuesta es completamente general en lo que respecta a la simulación estacionaria. En cambio, en el caso de la simulación transitoria, se restringe deliberadamente al ámbito de la simulación hidráulica. Aunque a primera vista, esta elección de campo podría parecer excesivamente limitante, es importante destacar que una simulación hidráulica eficaz es aplicable a la simulación de cualquier tipo de proceso que involucre el flujo de líquidos en tuberías.

5.2. NUEVA APROXIMACIÓN PARA ESTACIONARIOS

En el caso de la simulación estacionaria, la aproximación propuesta implica el desarrollo de un código secuencial que implementa métodos de solución completamente "ad hoc" para el campo de simulación considerado. Como se detallará más adelante, el bloque **`INIT`** de los componentes facilita la aplicación o incorporación de prácticamente cualquier método de solución, ofreciendo así una flexibilidad considerable en la elección de enfoques específicos para abordar el problema planteado.

Este enfoque contrasta con la aproximación habitual para formular estacionarios en herramientas de MSOO, que implica considerar de forma simultánea todas las ecuaciones transitorias y un conjunto de ecuaciones adicionales para imponer que las derivadas son

nulas en el estacionario. Los inconvenientes de la aproximación convencional, previamente analizados en los capítulos anteriores, motivan el desarrollo de esta nueva estrategia.

La herramienta EcosimPro destaca por la capacidad de integrar código secuencial en el bloque **`INIT`** de los componentes, proporcionando así una libertad y flexibilidad significativas al desarrollador. Conforme al manual del usuario de EcosimPro, este bloque se utiliza principalmente para asignar valores iniciales a variables que requieren un valor antes del comienzo de la simulación. A pesar de esta función principal, cabe destacar que el bloque **`INIT`** de los componentes no se limita a esta tarea básica; de hecho, ofrece la posibilidad de llevar a cabo operaciones más complejas, como el cálculo de estados estacionarios.

5.2.1. Visión General de la Nueva Aproximación de Estacionarios

La nueva aproximación parte del supuesto de que el modelo se compone de una malla de componentes interconectados, tal como se ilustra en la Figura 3-16. Asimismo, en esta aproximación se asume la necesidad de incorporar en cada modelo un componente que actúa como resolvedor y que no representa a ningún elemento físico.

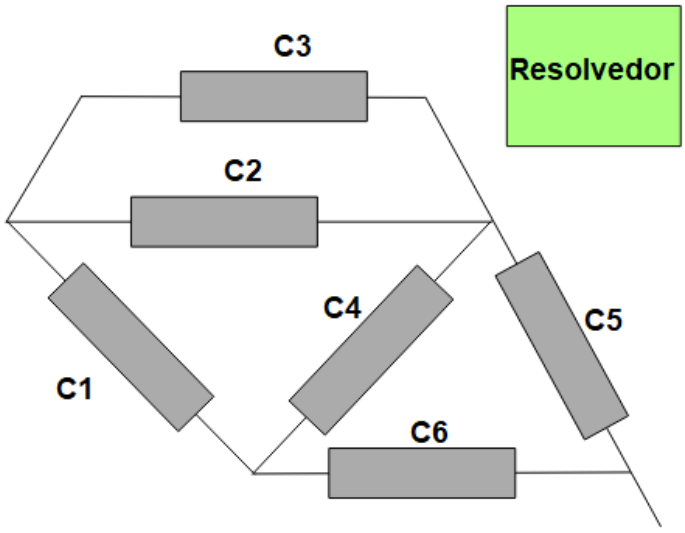


Figura 5-1: Ejemplo de Modelo Estacionario (Malla Componentes + Resolvedor)

La aproximación propuesta para implementar un estacionario completamente independiente de los estacionarios proporcionado por EcosimPro, consta de tres pasos, que se ilustran en la Figura 5-2.

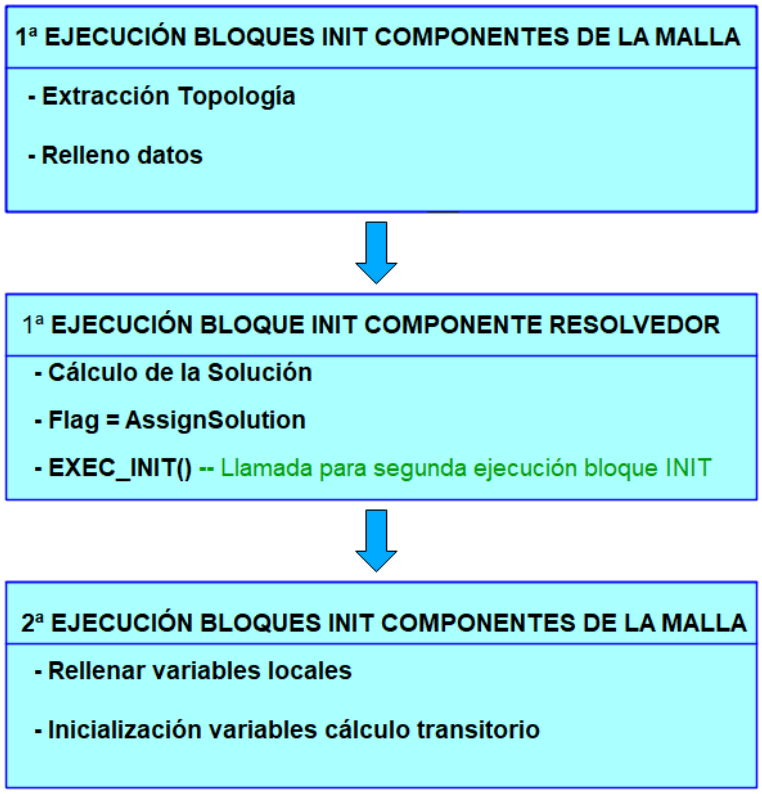


Figura 5-2: Pasos de la Nueva Aproximación Estacionaria

Descripción de los Pasos del Procedimiento

A continuación, se describen estos tres pasos de manera conceptual. Los detalles de las técnicas de programación requeridas para la realización de los pasos del procedimiento se describen en el Apéndice D.

Paso primero:

En una primera ejecución del bloque **`INIT`** de los componentes que constituyen la malla, se extrae su topología y se rellenan estructuras de datos globales con dicha topología y con los datos de todos y cada uno de los componentes.

La topología del modelo es la definición de qué componentes constituyen el modelo y cómo están interconectados. En principio, esta información solo es conocida por el compilador del lenguaje de simulación, pero la ejecución de los bloques **`INIT`** de los componentes permite acceder a la misma.

Es fundamental destacar que la metodología propuesta requiere ejecuciones adicionales del bloque **`INIT`**. Con el objetivo de distinguir esta primera ejecución, destinada a obtener la topología del modelo, de las ejecuciones posteriores con propósitos diferentes, se propone el uso de una bandera global, ya sea de tipo enumerado o booleano.

Paso segundo:

Se asume que en cada modelo debe incluirse un componente representativo del "*Resolvedor*", que no representa ningún elemento físico. La prioridad de ejecución del bloque **`INIT`** de este componente debe ser menor que la prioridad de ejecución del bloque **`INIT`** de los componentes reales de la malla. Esto implica que el bloque **`INIT`** de este componente se ejecuta después de la finalización de los bloques **`INIT`** de los componentes que representan elementos físicos reales de la malla. La gestión de la prioridad de ejecución de los bloques **`INIT`** de los componentes en EcosimPro se explica en la sección 3.2.1. El bloque **`INIT`** del componente "*Resolvedor*" se encarga de ejecutar el algoritmo de solución seleccionado, que puede basarse en ecuaciones o en la aproximación modular secuencial. Para el cálculo de la solución se utilizan las estructuras de datos globales.

Una vez finalizado el algoritmo de solución, es necesario copiar los resultados obtenidos en las estructuras de datos globales a las correspondientes variables locales de los componentes. Este proceso se realiza mediante una nueva ejecución del bloque **`INIT`** de los componentes, desencadenada desde el propio bloque **`INIT`** del componente "*Resolvedor*" mediante la función **`EXEC_INIT()`**, cuyo funcionamiento se explica en la sección 3.2.1. En este segundo paso, solo se desencadena el copiado, siendo importante destacar que el copiado propiamente dicho se considera que ocurre en el tercer paso del proceso general.

Distinguir entre esta segunda ejecución de los bloques **`INIT`** de los componentes y la primera es esencial, ya que requiere acciones diferentes. En el caso de los componentes reales de la malla, durante la segunda llamada, deben obtener los resultados almacenados en las estructuras globales. Por otro lado, en el caso del componente "*Resolvedor*", no es necesario repetir la llamada al algoritmo de solución. En su lugar, se debe detectar el final del segundo paso. Para realizar esta distinción, se utiliza una bandera global, que puede ser una variable de tipo enumerado o de tipo booleano, configurada para indicar de manera inequívoca el propósito específico de la actual ejecución de los bloques **`INIT`**.

Paso tercero:

En el tercer paso, se procede a la segunda ejecución del bloque **`INIT`** de los componentes que integran la malla. El objetivo de esta segunda ejecución es rellenar las variables locales con los resultados obtenidos en las estructuras de datos globales. Los bloques **`INIT`** de los componentes utilizan un semáforo o bandera global para llevar a cabo la acción correspondiente. Durante la primera ejecución del bloque **`INIT`** de los componentes de la malla, se extrae la topología y se llenan los datos en una estructura de datos global, mientras que en la segunda ejecución se copian los resultados del cálculo desde variables globales a locales del componente. Por lo general, en esta segunda ejecución del bloque **`INIT`**, no se ejecuta ninguna sentencia del bloque **`INIT`** del componente "*Resolvedor*".

En el caso de que se tenga la intención de realizar una simulación transitoria después de la estacionaria, en este paso también se procede a la inicialización de las variables de estado de la formulación transitoria con los resultados de la simulación estacionaria. No obstante, esta operación es opcional, ya que no en todos los casos se desea o requiere llevar a cabo una simulación transitoria después de la fase estacionaria.

Aplicabilidad del Procedimiento a Análisis en el Dominio de la Frecuencia

Conviene observar que la técnica propuesta para el cálculo del estado estacionario, también puede utilizarse para obtener soluciones en el dominio de la frecuencia. De hecho, la simulación en el dominio de la frecuencia es más sencilla que la simulación estacionaria, pues se consideran ecuaciones linealizadas y no hay necesidad de iterar para obtener la solución.

Una vez implementada dicho tipo de la solución es posible llevar a cabo barridos en la frecuencia y realizar los siguientes tipos de análisis:

- *Análisis de respuesta a pequeñas señales*.
- *Análisis de transferencia*, esto es calcular la función de transferencia entre una condición de contorno y una salida.
- *Dibujar diagramas de Bode y Nyquist* de las funciones de transferencia.
- *Análisis de la respuesta a ruido*.

Cabe resaltar que la metodología convencional para realizar análisis en el dominio de la frecuencia con herramientas de MSOO implica la conversión del sistema de ecuaciones a una representación lineal (proceso conocido como linealización), seguido por la exportación del sistema de ecuaciones linealizado a una herramienta como Matlab o similar, para en dicha herramienta realizar el análisis en el dominio de la frecuencia. El uso de dos herramientas tiene desventajas notables. En primer lugar, implica la necesidad de obtener licencias para ambas herramientas, generando costos adicionales. Además, aprender a usar dos entornos diferentes puede llevar tiempo y causar errores al cambiar de uno a otro. La exportación del modelo resulta en la pérdida de la estructura original y puede dificultar el acceso a las variables. Por último, la gestión de modelos en dos entornos separados complica la mantenibilidad del análisis a largo plazo.

Optimización del Procedimiento

Con el objetivo de mejorar el procedimiento previamente delineado, se propone una optimización destinada a reducir el tamaño del código generado. Esta mejora permite la resolución eficiente de modelos con miles de componentes. La optimización consiste en encapsular el código del bloque **`INIT`** de cada tipo de componente en una función de inicialización asociada con el tipo correspondiente. De esta manera el bloque **`INIT`** de cada

componente generará una única sentencia, que es la invocación de la función de inicialización, disminuyendo así el número de líneas del código generado.

5.2.2. Aplicación de la Nueva Aproximación para Cálculo de Estacionarios

La aplicación de esta nueva aproximación al cálculo de estacionarios es factible, pero requiere estrategias específicas de programación para obtener la topología y realizar la transferencia entre variables locales y globales. Estas estrategias se explican en detalle en el Apéndice D.

Además, la creación de una herramienta de simulación estacionaria orientada a ecuaciones y eficaz para grandes modelos implica el desarrollo de un resolvedor eficaz para sistemas de ecuaciones algebraicas dispersas. En el contexto de esta tesis, se ha diseñado un resolvedor de este tipo, documentado en el Apéndice E.

Finalmente, con el objetivo de explicar pormenorizadamente la nueva aproximación, así como para evaluarla y contrastarla con la aproximación clásica, se ha desarrollada una librería para estacionarios hidráulicos. En el capítulo 6 se describe el diseño de dicha librería y se comparan los resultados obtenidos con esta frente a los de una librería equivalente desarrollada de la forma clásica.

5.3. NUEVA APROXIMACIÓN PARA TRANSITORIOS

La nueva aproximación se centra en utilizar métodos de solución eficientes para abordar los problemas de simulación transitoria, con el objetivo de lograr que aplicaciones resultantes sean tan fáciles de utilizar y al mismo tiempo tan eficaces en términos de precisión y tiempo de cálculo como las herramientas de simulación específicas.

Las herramientas de MSOO proporcionan resolvedores genéricos para sistemas de ecuaciones diferenciales ordinarias (ODE's) y sistemas de ecuaciones algebraico-diferenciales (DAE's). Por tanto, comúnmente se considera que la formulación natural de un problema de simulación, para adaptarse a estas herramientas, es de tipo semidiscreto.

En ciertos campos de simulación, la formulación semidiscreta del problema es adecuada y proporciona una simulación eficiente. Sin embargo, existen otros muchos campos de simulación para los cuales los métodos discretos ofrecen una simulación más adecuada y eficiente. La nueva aproximación persigue la implementación de métodos discretos con herramientas de MSOO en aquellos casos en que resultan más adecuados. Para llevar a cabo la implementación de dichos métodos, se aprovechará la flexibilidad intrínseca que ofrecen los lenguajes de simulación de las herramientas de MSOO.

5.3.1. Visión General de la Nueva Aproximación de Transitorios

En general, se proponen cuatro ideas conductoras para el desarrollo de aplicaciones transitorias más eficientes:

1. **Consideración Métodos Discretos de Integración:**

   Se sugiere la adopción de métodos discretos de integración cuando estos sean los más adecuados para el campo específico de simulación.

2. **Evitar las Discontinuidades de los Métodos Discretos de Integración:**

   La propuesta es evitar las discontinuidades inherentes a los métodos discretos de integración. Esto implica implementar ecuaciones que aseguren transiciones suaves.

3. **Ocultar las Ecuaciones y el Estado Interno del Componente:**

   Se aboga por ocultar las ecuaciones y el estado interno del componente mediante el uso de objetos que funcionen como cajas negras. Estos objetos calculan y actualizan el estado interno en momentos específicos.

4. **Representación del Componente a través de Variables de Puertos:**

   La propuesta es representar el componente a través de ecuaciones que involucren exclusivamente las variables de los puertos. Esta idea junto con la anterior busca poner límite a la complejidad del modelo matemático que maneja la herramienta de MSOO.

A continuación, se desarrollan y se explican en detalle cada una de estas ideas.

Consideración de Métodos Discretos de Integración

Tal y como se ha visto en el Capítulo 4, las formulaciones de tipo semidiscreto no son las más adecuadas para la simulación hidráulica. Esta situación no se limita exclusivamente al ámbito de la simulación hidráulica. En general, la mayoría de las herramientas de simulación termofluida emplean métodos discretos. Estos métodos consideran una discretización temporal diferente para los diversos términos de las ecuaciones a resolver, aprovechando el conocimiento teórico que se tiene sobre la evolución y magnitud relativa de dichos términos basados en una discretización temporal de los diferentes términos de las ecuaciones adaptada al tipo de ecuaciones a resolver. En otras palabras, estos métodos aprovechan el conocimiento específico sobre el comportamiento de las soluciones de las ecuaciones para implementar enfoques de resolución más eficientes que los métodos genéricos proporcionados por las herramientas de MSOO.

Es lógico preguntarse cómo emplear los lenguajes de MSOO para crear algoritmos de integración ad-hoc, similares o idénticos a los métodos de solución discreta utilizados por herramientas de simulación específicas. Dado que estos métodos son de naturaleza discreta y considerando que las herramientas de MSOO cuentan con capacidades de

simulación discreta, se vislumbra la posibilidad de aprovechar estas capacidades inherentes para implementar de manera efectiva métodos de integración discretos.

A continuación, se presenta un ejemplo de integración de una ecuación simple utilizando las capacidades de simulación discreta de la herramienta de MSOO. Sea la siguiente ecuación diferencial ordinaría:

$$\frac{dx}{dt} = a\ x \tag{5-1}$$

Aplicando el método trapezoidal, se obtiene la siguiente expresión:

$$\frac{x^{t+\Delta t} - x^{t}}{\Delta t} = a\ \frac{1}{2}\left(x^{t+\Delta t} + x^{t}\right) \tag{5-2}$$

Despejando, el valor de la variable en el nuevo incremento de tiempo, se obtiene:

$$x^{t+\Delta t} = x^{t}\ \frac{\left(1 + \frac{1}{2}\ a\ \Delta t\right)}{\left(1 - \frac{1}{2}\ a\ \Delta t\right)} \tag{5-3}$$

Este sencillo algoritmo se puede implementar en el bloque discreto de un componente de EcosimPro. El Listado 5-1 muestra la codificación del componente, donde se utiliza la asignación retardada (sentencia con **`AFTER`**) de la variable booleana `IsTimeStep` para forzar los eventos de cálculo, y donde la actualización de la variable integrada se efectúa en el bloque discreto.

Listado 5-1: : Ejemplo de Componente con Integración Discreta

```
COMPONENT Inline1 "Component to solve x' = a*x by inline integration \
                   Discrete Version"
   DATA
      REAL a = -1.  "Data, equation to be solved x' = a*x"
      REAL h = 0.25 "Integration step"
      REAL xo = 10. "Initial value"
   DECLS
      DISCR REAL x                "Unknown Variable"
      DISCR REAL x_old            "Previous stored value"
      BOOLEAN IsTimeStep = FALSE "Flag, true at end of time step"
   INIT
      x_old = xo
      x = xo
      IsTimeStep = TRUE AFTER h
   DISCRETE
      WHEN(IsTimeStep) THEN
         IsTimeStep = FALSE
         IsTimeStep = TRUE AFTER h
         x_old = x
         x = x_old * (1 + 0.5*a*h) / (1. - 0.5*a*h)
      END WHEN
END COMPONENT
```

La Figura 5-3 muestra el resultado de la simulación transitoria de este componente. En la figura, la solución numérica se compara con la solución analítica. Es evidente que la solución numérica es muy próxima a la solución analítica, aunque exhibe discontinuidades. Estas discontinuidades son una consecuencia de cómo las herramientas de MSOO consideran los eventos durante la simulación.

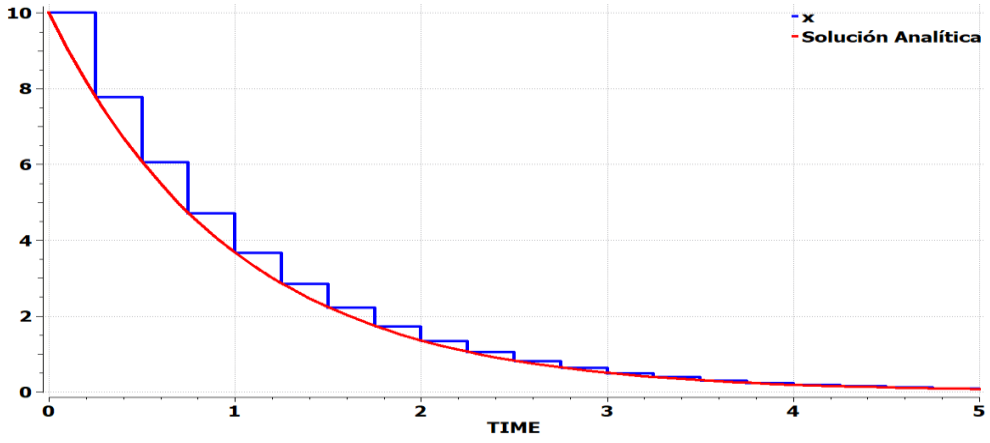


Figura 5-3: Solución Calculada por el Componente Ejemplo con Integración Discreta

Evitación las Discontinuidades de los Métodos Discretos de Integración.

El problema de una implementación como la anterior, utilizando una actualización puramente discreta de las variables, radica en que produce una solución discontinua. En

el contexto de una aplicación autónoma que no está vinculada a otras bibliotecas desarrolladas con la aproximación clásica, esto puede no representar un problema significativo. Sin embargo, cuando se busca conectar el modelo con otros componentes que emplean ecuaciones diferenciales ordinarias y ecuaciones algebraicas, las discontinuidades generadas por la parte discreta de este componente pueden complicar la integración de la parte continua.

Por tanto, es preciso evitar estas discontinuidades para posibilitar la integración de este componente con componentes de otras librerías que utilicen Ecuaciones Algebraicas Diferenciales (DAE´s) en su formulación, como es el caso de la librería de bloques de Control. Esta evitación de discontinuidades en la solución puede lograrse moviendo la sentencia que calcula la variable integrada del bloque discreto al bloque continuo. La ecuación de actualización del valor de la variable integrada sigue siendo idéntica, pero considera un incremento de tiempo variable, igual a la diferencia entre el tiempo actual y el último tiempo de actualización del estado discreto (t_old):

$$x^t = x^{t\_old} \frac{\left(1 + \frac{1}{2}\, a\, (t - t\_old)\right)}{\left(1 - \frac{1}{2}\, a(t - t\_old)\right)} \tag{5-4}$$

El Listado 5-2 muestra el componente mejorado, que integra la ecuación utilizando el mismo algoritmo, pero que evita las discontinuidades generadas por la versión anterior. Cabe resaltar que el bloque discreto únicamente se utiliza para generar los eventos y almacenar el valor anterior de la variable.

Listado 5-2: Ejemplo de Componente con Integración Discreta/Continua

```
COMPONENT Inline2 "Component to solve x' = a*x by inline integration \
                   Continuous Version"
   DATA
      REAL a = -1.      "Data, equation to be solved x' = a*x"
      REAL h = 0.25     "Integration step"
      REAL xo = 10.     "Initial value"
   DECLS
      REAL dt               "Integration step"
      REAL x                "Unknown Variable"
      DISCR REAL x_old      "Previous stored value"
      DISCR REAL TIME_old
      BOOLEAN IsTimeStep = FALSE "Flag, true at end of time step"
   INIT
      x_old = xo
      TIME_old = TIME
      x = xo
      IsTimeStep = TRUE
   DISCRETE
      WHEN(IsTimeStep) THEN
         IsTimeStep = FALSE
         IsTimeStep = TRUE AFTER h
         x_old = x
         TIME_old = TIME
      END WHEN
   CONTINUOUS
      dt = (TIME-TIME_old)
      x = x_old * (1 + 0.5*a*dt) / (1. - 0.5*a*dt)
END COMPONENT
```

A continuación, la Figura 5-4 muestra los resultados del componente mejorado comparados con la solución analítica, donde puede comprobarse la completa evitación de las discontinuidades.

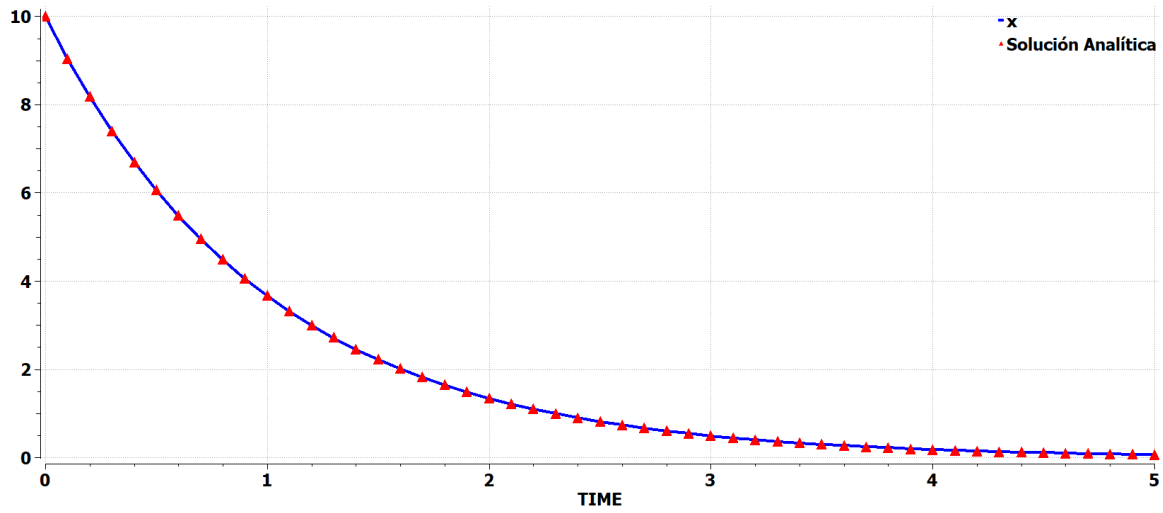


Figura 5-4: Solución Calculada por el Componente Ejemplo con Integración Discreta/Continua

El sencillo ejemplo anterior muestra el tipo de técnicas que se propone para evitar discontinuidades al emplear soluciones a nivel de componente.

Ocultación de las Ecuaciones Internas del Componente

La realización de simulaciones a gran escala con herramientas de MSOO presenta una dificultad significativa. Para realizar simulaciones transitorias que capturen fenómenos como el retraso térmico y la transmisión de ondas de presión en las tuberías del sistema, es necesario subdividir los componentes en volúmenes de control relativamente pequeños, y esto da lugar a sistemas de ecuaciones extraordinariamente grandes. Por ejemplo, en un modelo de planta industrial o de producción de energía, se pueden encontrar cientos o incluso miles de tuberías, cada una dividida en múltiples volúmenes de control. La formulación de cada volumen de control implica a su vez una cantidad considerable de variables. Como resultado, un modelo de complejidad industrial puede requerir cientos de miles o incluso millones de variables e igual número de ecuaciones.

Evidentemente, un modelo con una cantidad tan considerable de variables tiene el potencial de sobrecargar la herramienta de simulación. En primer lugar, en la etapa de procesamiento simbólico, donde se deben analizar cientos de miles de ecuaciones y variables. A continuación, el resolvedor también enfrenta una carga considerable. Por último, la herramienta para monitorización de los resultados de simulación, también resulta sobrecargadas al tener que gestionar la selección de cualquier variable en un conjunto tan grande.

Con el fin de evitar este crecimiento de la complejidad, se propone que los componentes oculten sus variables internas, y que el cálculo y actualización de dichas variables se encapsule en un objeto.

Continuando con el ejemplo de la integración de la ecuación anterior con el fin de ilustrar la técnica. En primer lugar, se define una clase llamada ***SimpleEqn*** que incluye como atributo el estado almacenado de la variable integrada. Esta clase proporciona dos métodos: `updateState` para almacenar el estado e `increment` para calcular el incremento de la variable. El Listado 5-3 muestra la definición de dicha clase al principio.

En la implementación del componente, se elimina la declaración de la variable `x_old`, que se utilizaba para el estado almacenar el estado, y en su lugar se declara un objeto de la clase ***SimpleEqn*** con el identificador `eqn`. Finalmente, la sentencia en el bloque discreto que actualizaba el estado se reemplaza por una llamada al método de `updateState`, y el lado derecho de la sentencia del bloque continuo que integraba el valor de `x`, se reemplaza por una llamada al método `increment`.

El Listado 5-3 muestra la definición de dicha clase y del componente.

Listado 5-3: Ejemplo de Componente con Integración Discreta/Continua y Ocultación del Estado

```
CLASS SimpleEqn
   DECLS
      REAL x_old   "Stored state"
   METHODS
      METHOD NO_TYPE updateState(IN REAL x)
         BODY
            x_old = x
      END METHOD
      METHOD REAL increment(IN REAL a, IN REAL dt)
         BODY
            RETURN  x_old * (1 + 0.5*a*dt) / (1. - 0.5*a*dt)
      END METHOD
END CLASS
COMPONENT Inline3
"Component to solve x' = a*x by inline integration - \
       Continuous Version & State Hiding"
   DATA
      REAL a = -1.  "Data, equation to be solved x' = a*x"
      REAL h = 0.25 "Integration step"
      REAL xo = 10. "Initial value"
   DECLS
      REAL dt
      REAL x
      DISCR REAL TIME_old
      BOOLEAN IsTimeStep
   OBJECTS
      SimpleEqn eqn
   INIT
      eqn.updateState(xo)
      TIME_old = TIME
      IsTimeStep = TRUE
   DISCRETE
      WHEN(IsTimeStep) THEN
         IsTimeStep = FALSE
         IsTimeStep = TRUE AFTER h
         eqn.updateState(x)
         TIME_old = TIME
      END WHEN
   CONTINUOUS
      dt = (TIME-TIME_old)
      x = eqn.increment(a, dt)
END COMPONENT
```

El funcionamiento de esta tercera versión del componente es idéntico al de la versión anterior. Como se puede observar, el uso de un objeto ha permitido ocultar las variables que representan el estado, así como el cálculo de la actualización de la variable integrada. En un componente tan simple como éste, el beneficio es bastante limitado. Sin embargo, si esta idea se aplica a un componente con decenas o cientos de volúmenes de control, el uso de un objeto permite ocultar el estado y las ecuaciones para actualizar dicho estado, reduciendo el número de variables y ecuaciones que tiene que gestionar la herramienta de MSOO.

Un problema de la aproximación es la gestión de cada cuanto tiempo debe almacenarse el estado de los nodos o volúmenes de control del componente. En el caso de transitorios rápidos, la frecuencia de actualización del estado interno del componente debe estar directamente vinculada al tiempo característico en el cual se producen cambios sustanciales en su estado. Ahora bien, en el caso de transitorios lentos, dicho incremento de tiempo puede ser mayor que el tiempo característico mencionado.

Representación del Componente a través de Variables de Puertos:

Para los componentes que se dividen en múltiples volúmenes control, y que tienen unas pocas variables de interfaz (estas variables de interfaz son las de puerto), se propone representarlos a nivel del modelo global mediante ecuaciones continuas que impongan relaciones entre las variables de los puertos.

Un ejemplo sencillo, puede servir para ilustrar esta idea. Sea una pared sólida en la que desea analizar la conducción transitoria de calor en una dimensión, e imaginemos que el componente se encarga de gestionar su estado interno, y que lo ha almacenado en un tiempo anterior. En tanto que el tiempo transcurrido desde el momento en que se almacenó el estado, sea pequeño, la pared puede ser considerada como un conductor, cuyos flujos de calor en la cara de entrada y en la cara de salida dependen solamente de la temperatura aplicada en dichas caras, y de la distribución de temperaturas en el tiempo previo. Una vez que la pared puede ser considerada como un conductor, es posible aplicar el método nodal modificado, esto es, en los nodos sin capacidad calorífica, la temperatura se obtiene a partir de que la suma de flujos caloríficos debe ser nula.

Sin embargo, surge un problema en la aplicación de esta idea: la gestión del tiempo entre cada actualización del estado de los nodos o volúmenes de control del componente. En el caso de la pared y con transitorios rápidos, como cambios bruscos de temperatura, el incremento de tiempo para actualizar el estado interno del componente debe estar relacionado con el tiempo característico de la variación de temperatura en los nodos. En cambio, en el caso de transitorios lentos, dicho incremento de tiempo se puede aumentar y se pueden utilizar valores grandes cuando el sistema se acerca al estado estacionario. La selección adecuada de este parámetro es crucial para garantizar la precisión y eficiencia de la simulación en diferentes condiciones de operación del sistema. En el caso de métodos explícitos como el método de las características, el propio método de solución dicta un incremento de tiempo apropiado para transitorios rápidos, eliminando la necesidad de preocuparse por su selección. No obstante, existen métodos implícitos que permiten el uso de incrementos de tiempo mayores que el característico; en estos casos, podría considerarse la implementación de un control de errores para la selección del incremento de tiempo.

5.3.2. Aplicación del Nuevo Paradigma al Cálculo Térmico de Paredes

En esta sección se continua con el ejemplo mencionado anteriormente, la conducción transitoria de calor unidimensional en una pared. Se explora como aplicar las ideas de la nueva aproximación hasta conseguir una implementación concreta y efectiva.

Primera Idea: Utilización de un Método Discreto

La primera idea de la nueva aproximación consiste en seleccionar un método discreto que sea apropiado para resolver el problema específico en consideración. Para la ecuación de conducción transitoria del calor en una dimensión, el método de diferencias finitas de Crank-Nicolson es comúnmente considerado como uno de los mejores métodos para la solución numérica de dicha ecuación. Este método ofrece una combinación de precisión y estabilidad numérica que lo hace especialmente adecuado para resolver problemas de difusión térmica en sistemas unidimensionales. Es incondicionalmente estable y de segundo orden tanto en el espacio como en el tiempo.

La ecuación transitoria del calor en una dimensión se expresa típicamente como:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \tag{5-5}$$

donde $T(x,t)$ es la temperatura en función de la posición x y el tiempo t, y α es la difusividad térmica.

El método de Crank-Nicolson aproxima tanto la derivada temporal como la derivada espacial de manera central en el tiempo y el espacio, respectivamente. La discretización del método de Crank-Nicolson lleva a un esquema de diferencias finitas que se puede expresar de la siguiente manera:

$$\frac{T_i^{k+1} - T_i^k}{\Delta t} = \frac{\alpha}{2}\left(\frac{T_{i+1}^{k+1} - 2T_i^{k+1} + T_{i-1}^{k+1} + T_{i+1}^{k} - 2T_i^{k} + T_{i-1}^{k}}{(\Delta x)^2}\right) \tag{5-6}$$

donde:

T_i^k es la temperatura en el nodo i en el tiempo k.

T_i^{k+1} es la temperatura en el nodo i en el próximo paso de tiempo.

Δx es el tamaño del paso espacial

Δt es el tamaño del paso temporal

Después de reorganizar la ecuación, se obtiene:

$$-r\,T_{i-1}^{k+1} + (1+2r)\,T_i^{k+1} - r\,T_{i+1}^{k+1} = r\,T_{i-1}^{k} + (1-2r)\,T_i^{k} + r\,T_{i+1}^{k} \tag{5-7}$$

definiendo $r = \frac{\alpha\,\Delta t}{2\,(\Delta x)^2}$

Esta ecuación se aplica para cada punto de la malla espacial, resultando un sistema de ecuaciones lineales. Suponiendo que las temperatura en el primer y último nodo, T_1^{k+1} y T_N^{k+1}, son conocidas, resulta un sistema de ecuaciones que proporciona las temperaturas de los nodos interiores. Organizando ese sistema de ecuaciones en forma matricial, resulta el siguiente sistema tridiagonal, que puede resolverse fácilmente usando el algoritmo de Thomas. Dicho algoritmo es una forma muy eficiente de resolver sistemas de ecuaciones lineales tridiagonales.

$$\begin{bmatrix} 1+2r & -r & 0 & 0 & \cdots & 0 \\ -r & 1+2r & -r & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -r & 1+2r \end{bmatrix} \begin{bmatrix} T_2^{k+1} \\ T_3^{k+1} \\ T_4^{k+1} \\ \vdots \\ T_{N-1}^{k+1} \end{bmatrix} = \\ = \begin{bmatrix} T_2^k + r\left(T_1^k - 2T_2^k + T_3^k\right) + rT_1^{k+1} \\ T_2^k + r\left(T_1^k - 2T_2^k + T_3^k\right) \\ \vdots \\ T_{N-1}^k + r\left(T_{N-2}^k - 2T_{N-1}^k + T_N^k\right) + rT_N^{k+1} \end{bmatrix} \tag{5-8}$$

Una vez calculadas las temperaturas en los nodos interiores, es posible obtener el resultado buscado, los flujos de calor en los contornos:

$$q_1^{n+1} = \frac{k\,A}{\Delta x}(T_1^{n+1} - T_2^{n+1}) \tag{5-9}$$

$$q_N^{n+1} = \frac{k\,A}{\Delta x}(T_{N-1}^{n+1} - T_N^{n+1}) \tag{5-10}$$

Segunda Idea: Evitación de las Discontinuidades

Con el objetivo de evitar las discontinuidades, se implementará el cálculo de diferencias finitas de forma continua, es decir, se llamará dicho cálculo desde el bloque continuo.

Tercera Idea: Ocultación de las Ecuaciones Internas del Componente

Con la finalidad de ocultar variables y la complejidad del cálculo, se define la clase ***"WallClass"*** para representar la pared. Esta clase cuenta con los métodos:

- **`Initialize()`**: Inicializa los objetos de la clase

- **`Update()`**: Actualiza el estado de la distribución de temperaturas en momentos específicos de la integración.
- **`InnerPoints()`**: Aplica el método de Crank-Nicolson y calcula la nueva distribución de temperaturas en el interior de la pared.
- **`q_in()`**: Calcula el flujo de calor en el contorno de entrada.
- **`q_out()`:** Calcula el flujo de calor en el contorno de salida.

El Listado 5-4 muestra el código de esta clase, así como de la función `tridiag`, la cual implementa el algoritmo de Thomas y es esencial para el funcionamiento del método **`InnerPoints()`**. Es importante destacar que el método **`InnerPoints()`** debe ser invocado por cualquiera de los dos métodos encargados de calcular el flujo de calor en los contornos del modelo. Sin embargo, dado que no hay garantía sobre cuál de estos dos métodos se ejecutará primero una vez que las ecuaciones del modelo estén ordenadas, se ha implementado un contador de llamadas para ambos métodos. Haciendo uso de dicho contador, cualquiera de los dos métodos llamará a **`InnerPoints()`** únicamente cuando el contador sea impar, asegurando así una ejecución coherente y no repetida del método en el momento adecuado.

Otra característica interesante de la clase es que incluye un parámetro de dimensionamiento llamado `MAX_NODES`, que representa el número máximo de nodos considerados. Es importante destacar que el número de nodos utilizados actualmente puede ser igual o inferior a este valor máximo de nodos. Esto proporciona flexibilidad, ya que permite cambiar el número de nodos considerados sin tener que recompilar el modelo, y facilita comprobar la influencia de la nodalización en los resultados obtenidos.

*Listado 5-4: Función para Sistemas de Ecuaciones Lineales Tridiagonales y Clase "**`WallClass`**" Encapsulando la Aplicación del Método de Crank-Nicolson*

```
FUNCTION NO_TYPE tridiag(
--Function implementing the Thomas algorithm to solve tridiagonal linear equation
systems
      IN REAL a[],
      IN REAL b[],
      IN REAL c[],
      IN REAL r[],
      OUT REAL u[],
      IN INTEGER n)
   DECLS
      REAL bet
      REAL gam[n]
      INTEGER j
   BODY
      bet=b[1]
      ASSERT (abs(bet) > 0) FATAL  "Tridiag function fails"
      u[1]=r[1]/bet
      FOR (j IN 2,n) --Decomposition and forward substitution.
         gam[j]=c[j-1]/bet
         bet=b[j]-a[j]*gam[j]
         ASSERT (abs(bet) > 0) FATAL  "Tridiag function fails"
         u[j]=(r[j]-a[j]*u[j-1])/bet
      END FOR
      FOR(k IN 1, n-1)
```

```
25          j = n - k
26          u[j]=u[j]-gam[j+1]*u[j+1]
27       END FOR
28       RETURN
29 END FUNCTION
30
31 CLASS WallClass (INTEGER MAX_NODES)
32    DECLS
33       INTEGER icount = 0
34       INTEGER nodes
35       REAL k
36       REAL rho
37       REAL cp
38       REAL e
39       REAL A
40       REAL dx
41       REAL Time0
42       REAL T0[MAX_NODES]
43       REAL T[MAX_NODES]
44       REAL alpha
45       CONST REAL theta = 0.5
46       REAL a[MAX_NODES]
47       REAL b[MAX_NODES]
48       REAL c[MAX_NODES]
49       REAL d[MAX_NODES]
50    METHODS
51       METHOD NO_TYPE Initialize(IN INTEGER nodes_cmp, IN REAL k_cmp,
52                     IN REAL rho_cmp, IN REAL cp_cmp, IN REAL e_cmp,
53                     IN REAL A_cmp, IN REAL To, IN REAL Time)
54          BODY
55             icount = 0
56             nodes = nodes_cmp
57             k = k cmp
58             rho = rho_cmp
59             cp = cp_cmp
60             e = e_cmp
61             A = A_cmp
62             dx = e/(nodes-1)
63             Time0 = Time
64             FOR (i IN 1, nodes)
65                T0[i] = To
66                T[i] = To
67             END FOR
68             RETURN
69       END METHOD
70       METHOD BOOLEAN Update(IN REAL Time)
71          BODY
72             icount = 0
73             FOR (i IN 1, nodes)
74                T0[i] = T[i]
75             END FOR
76             Time0 = Time
77             RETURN TRUE
78       END METHOD
79       METHOD NO_TYPE InnerPoints(IN REAL Time, IN REAL Tk_in, IN REAL Tk_out)
80          DECLS
81             REAL r
82          BODY
83             alpha = k /(rho*cp)
84             r = theta * (Time-Time0) * alpha / dx**2
85             a[1] = 0
86             b[1] = 1
87             c[1] = 0
88             d[1] = Tk_in
89             FOR(j IN 2, nodes-1)
90                a[j] = -r
91                b[j] = (1. + 2.*r)
92                c[j] = -r
93                d[j] = T0[j] + (1-theta)*((Time-Time0) * alpha  / dx**2)\
94                      *(T0[j-1] - 2*T0[j] + T0[j+1])
95             END FOR
96             a[nodes] = 0
97             b[nodes] = 1
```

```
            c[nodes] = 0
            d[nodes] = Tk_out
            tridiag(a,b,c,d,T,nodes)
         RETURN
      END METHOD
      METHOD REAL q_in(IN REAL Time, IN REAL Tk_in, IN REAL Tk_out)
         DECLS
            REAL q_in
         BODY
            icount = icount + 1
            IF(icount - icount/2*2 == 1) THEN
               InnerPoints(Time, Tk_in, Tk_out)
            END IF
               q_in = k*A*(Tk_in - T[2])/dx
               RETURN q_in
      END METHOD
      METHOD REAL q_out(IN REAL Time, IN REAL Tk_in, IN REAL Tk_out)
         DECLS
            REAL q_out
         BODY
            icount = icount + 1
            IF(icount - icount/2*2 == 1) THEN
               InnerPoints(Time, Tk_in, Tk_out)
            END IF
            q_out = k*A*(T[nodes-1] - Tk_out)/dx
            RETURN q_out
      END METHOD
END CLASS
```

Cuarta Idea: Representación del Componente a través de Variables de Puertos

Por último, se aplica la idea de representar el componente a través de ecuaciones que relacionan los valores en los contornos. A nivel de modelo, el componente va a consistir en dos ecuaciones que proporcionan el flujo de calor en cada uno de los contornos en función de la temperatura en dichos contornos. Aplicando esta idea y haciendo uso de un objeto de la clase ***"WallClass"*** se llega al código mostrado en el Listado 5-5.

*Listado 5-5: Ejemplo de Componente "**Wall_NA**" Utilizando la Nueva Aproximación*

```
USE PORTS LIB
COMPONENT Wall_NA "Thermal 1D transient wall using the new approach"
   PORTS
      IN thermal(n=1) tp_in
      OUT thermal (n=1) tp_out
   DATA
      INTEGER nodes = 20  "Number of nodes"
      REAL A = 1          "Wall Area"
      REAL e = 0.1        "Wall Thickness"
      REAL rho = 1000.    "Densiy of wall material"
      REAL cp = 500       "Specific heat of wall material"
      REAL k = 50.        "Thermal conductivity of wall material"
      REAL To = 290.      "Initial Wall Temperature"
   OBJECTS
      WallClass(MAX_NODES=1000) wall
   INIT
      wall.Initialize(nodes, k, rho, cp, e, A, To, TIME)
   DISCRETE
      ASSERT(wall.Update(TIME)) WARNING "This message should never appear"
   CONTINUOUS
      tp_in.q[1] = wall.q_in(TIME, tp_in.Tk[1], tp_out.Tk[1])
      tp_out.q[1] =  wall.q_out(TIME, tp_in.Tk[1], tp_out.Tk[1])
END COMPONENT
```

A continuación, se proporcionan algunas explicaciones y detalles sobre la implementación del componente "***Wall_NA***" (modelo de pared usando la nueva aproximación):

- Gracias al uso del objeto, las temperaturas internas del componente y otras variables auxiliares, como la difusividad y el espaciado internodal, quedan ocultas dentro del objeto `wall` de tipo ***"WallClass"***.
- La clase ***"WallClass"*** requiere un parámetro de dimensionado (`MAX_NODES`), que representa el número máximo de nodos que puede manejar. El número actual de nodos (`nodes`) es un dato variable y solo necesita ser menor o igual que `MAX_NODES`, lo que permite ajustarlo de forma interactiva durante la ejecución de la simulación sin necesidad de recompilar el modelo.
- Se realiza una llamada al método de inicialización de la clase en el bloque **INIT** del componente con el propósito de pasar los datos del componente a la clase.
- En el bloque **DISCRETE**, es necesario reinicializar el estado del componente cada cierto tiempo. Por simplicidad, se ha decidido actualizar el estado del objeto `wall` en cada incremento de tiempo del algoritmo de integración utilizado en EcosimPro. Esto se logra llamando al método `wall.Update()` desde la condición de un **ASSERT** en el bloque **INIT**, ya que las condiciones de los **ASSERT** se verifican al final de cada paso de integración.
- A nivel de ecuaciones continuas, el componente consiste únicamente de dos ecuaciones que calculan los flujos en los contornos. El número de nodos internos considerados no afecta al número de ecuaciones que ve la herramienta de simulación. Es posible considerar 1000 nodos y el número de ecuaciones sigue siendo dos. Esto representa una simplificación significativa en comparación con un componente programado de forma convencional.

Una crítica que se puede hacer a este enfoque es que los resultados pueden depender de los incrementos de tiempo utilizados. En el caso de transitorios rápidos, los incrementos de tiempo considerados por el integrador podrían ser demasiado altos. Los defensores de formulaciones semidiscretas podrían argumentar que este problema no existe al utilizar un resolvedor moderno de ODE's o DAE's, que incorporan un control del paso de integración para mantener el error de integración dentro de límites. Sin embargo, esta sensación de seguridad es engañosa, ya que existe otra fuente de error: la discretización espacial. El efecto de la discretización espacial es difícil de estimar, a menos que se realicen pruebas paramétricas cambiando la nodalización. Con la nueva aproximación, tanto el número de nodos como el incremento de tiempo pueden ser cambiados sin necesidad de recompilar el modelo, lo que facilita la exploración del efecto de la discretización espacial y del tiempo en los resultados.

Por último, es importante señalar que el nuevo componente produce resultados casi idénticos a los del clásico salvo que se le fuerce a funcionar con incrementos de tiempo de actualización altos, y que los consumos de tiempo de computación son menores o similares a los de la aproximación clásica. En cualquier caso, el propósito del componente es ilustrar la nueva aproximación. Es ampliamente conocido que el enfoque clásico funciona bastante bien en el caso de ecuaciones de tipo parabólico como la de transferencia de calor, por lo que no existe una presión a utilizar el nuevo paradigma en dicho tipo de problemas. Donde la nueva aproximación puede proporcionar beneficios significativos es en su aplicación a problemas de tipo hiperbólico. Es sabido que los métodos semidiscretos, como el método de las líneas, no son demasiado eficaces cuando se aplican a este tipo de ecuaciones.

5.3.3. Aplicación del Nuevo Paradigma en Cálculos Termo-hidráulicos

En esta sección se proporciona una visión general de cómo aplicar las ideas del nuevo paradigma a la simulación termo-hidráulica en redes de tuberías. Un componente que represente una tubería en una la simulación termofluida tiene dos conexiones fluidas (una de entrada y otra de salida) a respectivos nodos termo-hidráulicos. Se asume que la presión y la temperatura en dichos nodos son conocidas, las ecuaciones para simular la tubería de forma discreta podrían expresarse conceptualmente de la siguiente forma:

$$m_{p.in}^{t} = m_{p.in}^{t}\left(t, P_{nd.in}^{t}, P_{nd.out}^{t}, T_{nd.in}^{t}, T_{nd.out}^{t}, \boldsymbol{P}_{1:n}^{told}, \boldsymbol{m}_{1:n}^{told}, \boldsymbol{T}_{1:n}^{told}\right) \tag{5-11}$$

$$m_{p.out}^{t} = m_{p.in}^{t}\left(t, P_{nd.in}^{t}, P_{nd.out}^{t}, T_{nd.in}^{t}, T_{nd.out}^{t}, \boldsymbol{P}_{1:n}^{told}, \boldsymbol{m}_{1:n}^{told}, \boldsymbol{T}_{1:n}^{told}\right) \tag{5-12}$$

$$T_{p.in}^{t} = \begin{cases} T_{p.in}^{t}\ \left(t, P_{nd.in}^{t}, P_{nd.out}^{t}, T_{nd.in}^{t}, T_{nd.out}^{t}, \boldsymbol{P}_{1:n}^{told}, \boldsymbol{m}_{1:n}^{told}, \boldsymbol{T}_{1:n}^{told}\right) & \textbf{if } m_{p.in}^{t} < 0 \\ T_{nd.in}^{t} & \textbf{if } m_{p.in}^{t} \geq 0 \end{cases} \tag{5-13}$$

$$T_{p.out}^{t} = \begin{cases} T_{p.out}^{t}\left(t, P_{nd.in}^{t}, P_{nd.out}^{t}, T_{nd.in}^{t}, T_{nd.out}^{t}, \boldsymbol{P}_{1:n}^{told}, \boldsymbol{m}_{1:n}^{told}, \boldsymbol{T}_{1:n}^{told}\right) & \textbf{if } m_{p.out}^{t} > 0 \\ T_{nd.out}^{t} & \textbf{if } m_{p.in}^{t} \leq 0 \end{cases} \tag{5-14}$$

donde:

$m_{p.in}^{t}$ es el flujo másico de entrada a la tubería en el tiempo t

$m_{p.out}^{t}$ es el flujo másico de salida de la tubería en el tiempo t

$P_{nd.in}^{t}$ es la presión en el nodo de entrada a la tubería

$P_{nd.out}^{t}$ es la presión en el nodo de salida de la tubería

$T_{nd.in}^{t}$ es la temperatura en el nodo de entrada a la tubería, dicha temperatura es el resultado del mezclado de los flujos que entran a dicha nodo.

$T^t_{nd.out}$ es la temperatura en el nodo de salida de la tubería, dicha temperatura es el resultado del mezclado de los flujos que entran a dicha nodo.

$T^t_{p.in}$ es la temperatura en la sección de entrada de la tubería, que es igual a la temperatura del nodo si el flujo de entrada $m^t_{p.in}$ es positivo, pero debe ser calculada por la tubería si dicho flujo es negativo.

$T^t_{p.out}$ es la temperatura en la sección de salida de la tubería, que es igual a la temperatura del nodo si el flujo de salida $m^t_{p.out}$ es negativo, pero debe ser calculada por la tubería si dicho flujo es positivo

$\boldsymbol{P}^{told}_{1:n}$ son las presiones en las secciones 1 a n en que se divide la tubería, almacenada en el tiempo previo $told$

$\boldsymbol{m}^{told}_{1:n}$ son las flujos másicos en las secciones 1 a n en que se divide la tubería, almacenada en el tiempo previo $told$

$\boldsymbol{T}^{told}_{1:n}$ son las temperatura en las secciones 1 a n en que se divide la tubería, almacenada en el tiempo previo $told$

Si se aplica el principio o idea anterior de ocultar el estado interno del componente en un objeto, las ecuaciones anteriores pueden simplificarse a:

$$m^t_{p.in} = m^t_{p.in}\left(t, P^t_{nd.in}, P^t_{nd.out}, T^t_{nd.in}, T^t_{nd.out}\right) \tag{5-15}$$

$$m^t_{p.out} = m^t_{p.in}\left(t, P^t_{nd.in}, P^t_{nd.out}, T^t_{nd.in}, T^t_{nd.out}\right) \tag{5-16}$$

$$T^t_{p.in} = \begin{cases} T^t_{p.in}\ \left(t, P^t_{nd.in}, P^t_{nd.out}, T^t_{nd.in}, T^t_{nd.out}\right) & \textbf{if } m^t_{p.in} < 0 \\ T^t_{nd.in} & \textbf{if } m^t_{p.in} \geq 0 \end{cases} \tag{5-17}$$

$$T^t_{p.out} = \begin{cases} T^t_{p.out}\left(t, P^t_{nd.in}, P^t_{nd.out}, T^t_{nd.in}, T^t_{nd.out}\right) & \textbf{if } m^t_{p.out} > 0 \\ T^t_{nd.in} & \textbf{if } m^t_{p.in} \leq 0 \end{cases} \tag{5-18}$$

Las ecuaciones anteriores especifican las funciones necesarias para adoptar la nueva aproximación. Es importante destacar que el desarrollador puede optar por funciones con diferentes grados de sofisticación, ya sea muy sencillas o muy complejas, dependiendo del nivel de precisión deseado en la simulación.

Considerando el caso puramente hidráulico y dejando de lado el cálculo de las temperaturas, se presentan varias posibilidades para las funciones de cálculo de los flujos másicos.

Una primera opción sería aplicar una formulación puramente estacionaria, como la siguiente:

$$m_{in}^{t} = m_{out}^{t} = \sqrt{\frac{2\,\rho\,\left(P_{in}^{t} - P_{out}^{t}\right)}{f(L/D)}} \tag{5-19}$$

Otra posible opción para el cálculo de los flujos másicos, es aplicar una formulación transitoria basada en el método de las características, la cual como se aplica con incrementos de tiempo menores que el tiempo de viaje de una onda de presión entre entrada y salida, independiza completamente la entrada de la salida. En ese caso, las ecuaciones para relacionar el flujo y las presiones (ver sección G.3 del Apéndice G con descripción general del método de las características) son:

$$m_{in}^{t} = \rho\left[\frac{CM - (P_{in}^{t}/\rho g + z_{in})}{BM}\right] \tag{5-20}$$

$$m_{out}^{t} = \rho\left[\frac{(P_{out}^{t}/\rho g + z_{out}) - CP}{BP}\right] \tag{5-21}$$

Una tercera opción sería utilizar el método implícito de cuatro puntos para obtener los flujos de entrada salida. El método implícito de 4 puntos plantea el siguiente par de ecuaciones para cada espacio entre nodos (ver sección G.4 del Apéndice G) y es un método estable para cualquier incremento de tiempo.

$$d_1\,\frac{m_{i+1}^{t+\Delta t}}{\rho} + d_2\,\frac{m_{i}^{t+\Delta t}}{\rho} - d_3\left(\frac{P_{i}^{t+\Delta t}}{\rho g} + z_i\right) + d_3\left(\frac{P_{i+1}^{t+\Delta t}}{\rho g} + z_{i+1}\right) + d_4 = 0 \tag{5-22}$$

$$\forall\, i \in \{1, \dots, N-1\}$$

$$-c_1\,\frac{m_{i+1}^{t\Delta t}}{\rho} + c_1\,\frac{m_{i}^{t+\Delta t}}{\rho} - c_2\left(\frac{P_{i}^{t+\Delta t}}{\rho g} + z_i\right) + c_3\left(\frac{P_{i+1}^{t+\Delta t}}{\rho g} + z_{i+1}\right) + c_4 = 0 \tag{5-23}$$

$$\forall\, i \in \{1,2, \dots, N-1\}$$

Teniendo en cuenta que el número de incógnitas es justamente el número de nodos, si se suponen conocidas la presión en el primer nodo o sección $P_1^{t+\Delta t}$ y la presión en la última sección, es posible resolver el sistema de ecuaciones y obtener los flujos másicos en el nodo 1 y el nodo N (último nodo), así como las presiones y flujos en todos los nodos interiores.

Una característica interesante del método es que proporciona una abstracción total del sobre método de solución, que se manipula dentro de un objeto. Sería posible implementar los 3 métodos y saltar de uno a otro a conveniencia. Por ejemplo, utilizando la ecuación (5-19) justamente cuando se realiza el cálculo del estacionario, pasando luego a utilizar las ecuaciones (5-20) y (5-21) correspondientes al método de las características durante

las partes rápidas del transitorio, y las ecuaciones (5-22) y (5-23) durante las partes lentas del transitorio.

5.3.4. Aplicación de la Nueva Aproximación para Cálculo de Transitorios

Con el objetivo de demostrar la viabilidad de aplicar los conceptos expuestos en esta sección, se ha llevado a cabo el desarrollo de una aplicación o librería en EcosimPro destinada al cálculo de transitorios hidráulicos mediante el uso del método de las características.

Este código, cuyo diseño se detalla en el capítulo 7, representa un paso significativo hacia la implementación práctica de las ideas discutidas. En dicho capítulo, no solo se expone la estructura y funcionamiento de la aplicación, sino que también se lleva a cabo una evaluación exhaustiva, comparándola con herramientas específicas diseñadas para abordar el fenómeno del golpe de ariete.

6. ESTACIONARIOS HIDRÁULICOS USANDO EL NUEVO PARADIGMA

En este capítulo, se presenta la librería ***HYDRAULIC_ST***, diseñada específicamente para el cálculo de estacionarios hidráulicos, haciendo uso de la nueva aproximación presentada en la sección 5.2. Asimismo, se comparan los resultados obtenidos con esta librería y los generados mediante una librería desarrollada para el mismo tipo de cálculo, siguiendo el paradigma clásico del modelado y simulación orientado a objetos.

Es importante señalar que la librería ***HYDRAULIC_ST*** se presenta como un ejemplo académico e incluye únicamente tres tipos de componentes:

- *"**Pipe**"*: para representar tuberías y conductos con la ecuación de caída de presión de Hazen-William,
- *"**Tank**"*: para representar tanques y depósitos, y
- *"**Outflow**"*: para representar salidas de flujo.

Una librería destinada a uso profesional debe incluir una amplia variedad de componentes, tales como bombas, reductores, válvulas, válvulas de retención, válvulas de control, filtros, cambiadores de calor, calentadores, enfriadores y boquillas de aspersión. Además, la formulación de los componentes debe ofrecer la posibilidad de emplear diferentes correlaciones para el cálculo de las caídas de presión. En el caso del componente tubería, la formulación debe permitir el cálculo de la caída de presión utilizando la fórmula de Darcy, la de Hazen-Williams o la de Manning. Asimismo, debería ser capaz de considerar la caída de presión adicional debida a accesorios de tubería como codos y curvas.

Adicionalmente, una librería profesional debe contar con una extensa base de datos que permita el cálculo de las propiedades fluidas necesarias para los fluidos de trabajo más comunes, considerando la temperatura y la presión como variables independientes. En un enfoque puramente hidráulico, las propiedades fluidas esenciales son la densidad, viscosidad y presión de saturación. No obstante, en cálculos termohidráulicos, que buscan determinar las temperaturas del fluido mediante la ecuación de la energía, se vuelve esencial incorporar el calor específico.

6.1. Requisitos de Usuario del Desarrollo

Antes de iniciar el desarrollo de cualquier aplicación de simulación, es fundamental definir los requisitos de usuario específicos para la aplicación. En el caso del ejemplo que estamos considerando, los requisitos son inicialmente limitados, ya que su propósito principal es ilustrar la aproximación propuesta. A continuación, se detallan los requisitos de usuario seleccionados para el desarrollo de este ejemplo:

Requisito 1: El propósito de la aplicación es realizar cálculos de estacionarios hidráulicos en redes de tuberías. Las variables a obtener son alturas piezométricas y flujos volumétricos. Se supone que la densidad del líquido es constante.

Requisito 2: La librería deberá incluir tres tipos de componentes: tuberías, tanques o depósitos y flujos volumétricos que se imponen en los nodos.

Requisito 3: Se utilizará la formulación Hazen-William para el cálculo de la caída de presión en las tuberías.

Requisito 4: Los depósitos o tanques impondrán una altura constante en el nodo conectado. En consecuencia, está prohibido conectar dos depósitos al mismo nodo. En este componente, se adopta el criterio de considerar los flujos como positivos cuando salen del tanque y entran en la red de tuberías.

Requisito 5: En este componente, se adopta el criterio de considerar los flujos como positivos cuando salen del tanque y entran en la red de tuberías.

Requisito 6: El fluido se define mediante una densidad y una viscosidad, ambas se consideran constantes.

6.2. Diseño de la Librería *HYDRAULIC_ST*

En esta sección se describen el diseño de la Librería ***HYDRAULIC_ST***, proporcionando una visión general de su estructura y de los elementos clave de la misma. El listado completo de la librería se encuentra disponible en la sección F.1 del Apéndice F. Las técnicas de programación generales utilizadas para obtener de la topología y facilitar la comunicación entre los componentes de la malla y el componente resolvedor se han explicado previamente en el Apéndice D. Además, el resolvedor hace uso de una clase para la resolución de sistemas algebraicos dispersos, que se documenta en el Apéndice E.

A continuación, se comentan los aspectos del diseño que son específicos de esta librería, tales como los datos y variables de los componentes, las estructuras de datos globales, y la formación del sistema de ecuaciones representativo de la malla.

Datos y Variables de los Componentes

Es fundamental identificar los datos y variables necesarias para representar cada uno los componentes conforme a la formulación deseada. En el lenguaje de EcosimPro, los datos se declaran en un bloque denominado **`DATA`** y las variables en un bloque denominado **`DECLS`**. El Listado 6-1 muestra los fragmentos de código donde se declaran los datos y variables de los componentes hidráulicos de la librería. Es relevante observar que las variables resultantes del cálculo estacionario se declaran como discretas, ya que su valor se establece únicamente en el momento de cálculo del estado estacionario.

Datos y Variables del Componente "`Pipe`"

Es requisito de diseño la utilización de la ecuación de Hazen-Williams como método para calcular la caída de presión en las tuberías. Con esta formulación, los datos requeridos para las tuberías incluyen el diámetro de la tubería, su longitud, el coeficiente de rugosidad de Hazen-Williams (que mide la rugosidad de la tubería). También se incluye un dato con la estimación inicial del flujo. Esta estimación, en caso de ser proporcionada, puede contribuir a lograr una convergencia más rápida, pero no es estrictamente necesaria, y se puede dejar el valor por defecto sin afectar la funcionalidad del sistema.

Las variables calculadas en las tuberías con dicha formulación son las alturas piezométricas en la entrada y salida de la tubería, así como el flujo volumétrico a través de la misma.

Datos y Variables del Componente "Tank"

Para el componente tanque, el dato esencial es la altura piezométrica en el mismo. Existe la opción de proporcionar una estimación del flujo inicial que sale del tanque, lo cual puede mejorar la convergencia, pero esta estimación no es estrictamente necesaria y se puede utilizar el valor por defecto sin comprometer la funcionalidad del sistema. La variable calculada en el tanque es el flujo saliente del mismo.

Datos y Variables del Componente "Outflow"

En el caso del componente "Outflow", se requiere un único dato: el flujo que se extrae del sistema. La única variable que se calcula para este componente es la altura piezométrica en el nodo de conexión.

Listado 6-1: Declaración de Datos y Variables Locales de los Componentes Hidráulicos

```
COMPONENT Pipe
   ...
   DATA
      REAL L = 600       UNITS "m"     "Pipe length"
      REAL D = 0.5       UNITS "m"     "Pipe inside diametert"
      REAL CHW = 120.    UNITS "-"     "Hazen Williams Coefficient"
      REAL Qo = 0.1      UNITS "m3/s"  "Initial flow"
   DECLS
      INTEGER ipipe = 0                "Pipe identifier"
      DISCR REAL H1      UNITS "m"     "Piezometric head at inlet"
      DISCR REAL H2      UNITS "m"     "Piezometric head at outlet"
      DISCR REAL Q       UNITS "m3/s"  "Volume flow"
   ...
END COMPONENT

COMPONENT Tank
   ...
   DATA
      REAL H = 100.      UNITS "m"     "Tank piezometric head"
      REAL Qo = 0.1      UNITS "m3/s"  "Initial guess of the tank outflow"
   DECLS
```

```
      INTEGER itank = 0               "Tank identifier"
      DISCR REAL Q      UNITS "m3/s"  "Tank outflow"
   ...
END COMPONENT

COMPONENT Outflow
   ...
   DATA
      REAL Q = 0.         UNITS "m3/s"  "Outflow"
   DECLS
      INTEGER iflow = 0                 "Outflow identifier"
      DISCR REAL H        UNITS         "Piezometric head at connected node"
    ...
END COMPONENT
```

Estructuras de Datos Globales

Para la comunicación con el resolvedor, por cada tipo de componente se define una clase y un vector unidimensional de objetos de dicha clase. El objetivo es proporcionar de estas estructuras de datos es proporcionar un mecanismo de para la a comunicación entre los componentes reales de la malla y el resolvedor. El siguiente fragmento de código muestra la declaración de estas clases y de los vectores de objetos de dichas clases. Es importante notar que este fragmento de código se encuentra fuera del alcance (*scope*) de cualquier componente y, por lo tanto, se considera que los vectores de objetos de estas clases son globales y, en consecuencia, visible desde cualquier otro componente o función.

Listado 6-2: Listado 6-3: Clases y Vectores de Objetos Globales

```
CONST INTEGER MAX_PIPE = 1000
CONST INTEGER MAX_TANK = 100
CONST INTEGER MAX_FLOW = 100

CLASS PipeClass
   DECLS
      INTEGER nd1              "Node at inlet"
      INTEGER nd2              "Node at outlet"
      REAL L      UNITS "m"    "Pipe length"
      REAL D      UNITS "m"    "Pipe inside diametert"
      REAL CHW    UNITS "-"    "Hazen Williams Coefficient"
      REAL Qo     UNITS "m3/s" "Initial guess flow"
      REAL H1     UNITS "m"    "Piezometric head at inlet"
      REAL H2     UNITS "m"    "Piezometric head at outlet"
      REAL Q      UNITS "m3/s" "Flow through pipe"
END CLASS
PipeClass P[MAX_PIPE]

CLASS TankClass
   DECLS
      INTEGER nod              "Node at tank outlet"
      REAL H      UNITS "m"    "Tank Piezometriz Head"
      REAL Q      UNITS "m3/s" "Tank outflow"
      REAL Qo     UNITS "m3/s" "Initial guess for pipe flow"
END CLASS
TankClass T[MAX_TANK]

CLASS OutflowClass
   DECLS
```

```
      INTEGER nod              "Node at tank outlet"
      REAL H      UNITS "m"    "Tank Piezometriz Head"
      REAL Q      UNITS "m3/s" "Initial guess for pipe flow"
END CLASS
OutflowClass OF[MAX_FLOW]
```

Componente Tipo Resolvedor y Algoritmo de Solución

Tras la finalización de la ejecución de los bloques **`INIT`** de los componentes de la malla, se encuentran almacenados en estructuras de datos globales tanto la topología del modelo como los datos de los componentes. Con esta información, es factible implementar cualquier método de solución estacionaria, ya sea orientado a ecuaciones o siguiendo la aproximación modular secuencial.

En el caso de la librería hidráulica ejemplo, se ha optado por aplicar la aproximación orientada a ecuaciones utilizando un método de Newton-Raphson adaptado a la solución de sistemas de ecuaciones dispersas.

6.2.1. Formación del Sistema de Ecuaciones

Existen diversas maneras de generar el sistema de ecuaciones para representar el comportamiento de una red hidráulica en estado estacionario. Por ejemplo, es posible considerar un sistema de ecuaciones en el cual solo son desconocidas las alturas piezométricas en los nodos, pero esto requiere que los flujos volumétricos en los componentes entre nodos se puedan despejar en función de las alturas en los nodos. El problema es que muchas de las formulaciones para el cálculo de caídas de presión en tuberías no permiten obtener de forma explícita el flujo en función de las alturas en los nodos.

También es posible considerar que solo son desconocidos los flujos en los lazos independientes en que se puede descomponer el circuito. Ahora bien, esta selección requiere que las caídas de presión en todos los componentes sean expresables en función del caudal, lo cual no es siempre posible. Por ejemplo, para una válvula reguladora de presión no existe una relación directa entre el flujo y la caída de presión en la misma, y aunque el desarrollo dicho componente no está entre los objetivos del desarrollo, seleccionar solo los flujos en los lazos independientes como desconocidos obstaculizaría la implementación de dicho componente en el futuro.

Por tanto, en este ejemplo y aprovechando que se dispone de un resolvedor para sistemas de ecuaciones dispersos se ha decidido considerar desconocidos tanto las alturas en los nodos, como los flujos a través de las tuberías y los proporcionados desde los tanques.

El número de variables desconocidas que se consideran es:

$$\textit{Nº variables desconocidas} = \textit{Nº nodos} + \textit{Nº tuberías} + \textit{Nº tanques}$$

En cuanto a las ecuaciones, habrá que considerar igual número de ecuaciones que de variables desconocidas. En el ejemplo planteado, se han considerado las siguientes ecuaciones: ecuaciones de continuidad en los nodos, ecuaciones de caída presión en las tuberías y ecuaciones de altura en los nodos directamente conectados a los tanques. De manera que el número de ecuaciones es también justamente igual al de variables desconocidas.

Nº ecuaciones = Nº nodos + Nº tuberías + Nº tanques = Nº variables desconocidas

A continuación, se presentan las ecuaciones correspondientes a cada uno de estos elementos: nodos, tuberías y tanques. Además, se presentan las derivadas respecto a las incógnitas con el propósito de facilitar la formulación del Jacobiano. Este último resulta fundamental para la implementación del método de Newton-Raphson.

Ecuaciones de Continuidad en los Nodos

En cada nodo, la suma de los caudales, considerados positivos si entran al nodo y negativos si salen del nodo, debe ser nula. Esta ecuación se expresa como:

$$F_{NODE\,n} = \sum_{i \in S_n} Q_{P,i} - \sum_{j \in E_n} Q_{P,j} + Q_{T,k} - Q_{O,l} = 0 \qquad (6\text{-}1)$$

donde:

$F_{NODE\,n}$ es la función o residuo a anular en el nodo *n-ésimo*.

S_n es el conjunto de todas las tuberías conectadas al nodo *n-ésimo* por su extremo de salida.

E_n es el conjunto de todas las tuberías conectadas al nodo n-ésimo por su extremo de entrada.

$Q_{P,i}$ es el caudal por la tubería i-ésima, que entra al nodo *n-ésimo*, (incógnita).

$Q_{P,k}$ es el caudal por la tubería *j-ésima*, que sale del nodo *n-ésimo*, (incógnita).

$Q_{T,k}$ es el caudal desde el tanque *k*-ésimo, que se supone conectado al nodo *n-ésimo*, (incógnita). Nótese que un nodo no puede conectarse a más de un tanque, y si no hay ningún tanque conectado, este término es nulo.

$Q_{O,l}$ es el caudal del componente flujo de salida (Outflow) l-ésimo, el cual se supone conectado al nodo *n-ésimo*, (dato).

Las incógnitas en la ecuación anterior son los caudales por las tuberías y el caudal desde el tanque. Los caudales de los componentes flujo de salida son siempre datos. Por tanto, las derivadas de esta ecuación con respecto a las incógnitas son:

$$\frac{d\,F_{NODE\,n}}{d\,Q_{P,i}} = 1 \qquad \forall\, i \;\in\; S_n \tag{6-2}$$

$$\frac{d\,F_{node\,n}}{d\,Q_{P,j}} = 1 \qquad \forall\, j \;\in\; E_n \tag{6-3}$$

$$\frac{d\,F_{NODE\,n}}{d\,Q_{T,k}} = 1 \qquad si\ el\ tanque\ k\text{-}ésimo\ está\ conectato\ al\ nodo\ n\text{-}ésimo \tag{6-4}$$

Ecuaciones de Caída de Presión en los Nodos

La ecuación de Hazen Williams para la caída de presión en un nodo es:

$$F_{PIPE\,i} = H_{in,i} - H_{out,i} - 10.67\,\frac{L_i}{C_{HW,i}^{1.852}\,D_i^{4.8704}}\,Q_{P,i}\,\left|Q_{P,i}\right|^{0.852} \tag{6-5}$$

donde:

$F_{PIPE\,i}$ es la función o residuo a anular en la tubería *i-ésima*.

$H_{in,i}$ es la altura piezométrica en el nodo de entrada a la tubería *i-ésima*, (incógnita).

$H_{out,i}$ es la altura piezométrica en el nodo de salida de la tubería *i-ésima*, (incógnita).

L_i es la longitud de la tubería *i-ésima*, (dato).

D_i es el diámetro de la tubería *i-ésima*, (dato).

$C_{HW,i}$ es el coeficiente Hazen-Williams de la tubería *i-ésima*, (dato).

$Q_{P,i}$ es el caudal desde por la tubería *i-ésima,* (incógnita).

Esta ecuación tiene un inconveniente y es que cuando el flujo es cero, la derivada de la caída de presión con respecto al flujo es nula, lo cual poder dar lugar a Jacobianos singulares. Por tanto, se modifica ligeramente la ecuación (6-5) para conseguir que no tenga derivada nula si el flujo es cero, esto se hace sumando una constante muy pequeña, Q_{min}, al valor absoluto del flujo:

$$F_{PIPE\,i} = H_{in,i} - H_{out,i} - 10.67\,\frac{L_i}{C_{HW,i}^{1.852}\,D_i^{4.8704}}\,Q_{P,i}\,\left(\left|Q_{P,i}\right| + Q_{min}\right)^{0.852} \tag{6-6}$$

Las derivadas de esta ecuación con respecto a las incógnitas que aparecen en la misma son:

$$\frac{\partial F_{PIPE\,i}}{\partial H_{in,i}} = 1 \tag{6-7}$$

$$\frac{\partial F_{PIPE\ i}}{\partial H_{out,i}} = -1 \tag{6-8}$$

$$\frac{\partial F_{PIPE\ i}}{\partial Q_{P,i}} = -10.67\ \frac{L}{C_{HW,i}^{1.852}\ D_i^{4.8704}}\ \left[\left(\left|Q_{P,i}\right| + Q_{min}\right)^{0.852} + \left|Q_{P,i}\right|\left(\left|Q_{P,i}\right| + Q_{min}\right)^{0.852-1}\right] \tag{6-9}$$

Ecuaciones de Altura Piezométrica en los nodos conectados a tanques

$$F_{TANK\ k} = H_{n,k} - H_{TANK\ k} \tag{6-10}$$

donde:

$F_{Tank\ k}$ es la función o residuo a anular en el tanque *k-ésimo*.

$H_{n,k}$ es la altura piezométrica del nodo conectado al tanque *k-ésimo*, (incógnita).

$H_{TANK\ k}$ es la altura piezométrica en el tanque *k-ésimo*, (dato).

La derivada de esta ecuación con respecto a la única incógnita que aparece en la misma es:

$$\frac{\partial F_{TANK\ K}}{\partial H_{n,k}} = 1 \tag{6-11}$$

Asignación de los Resultados a Variables Locales

Finalmente, una vez calculada la solución es preciso volcar los resultados de la misma en variables locales de los componentes. Los resultados de la simulación se rellenan en variables globales, que no son visibles desde la herramienta de monitorización del entorno de simulación. La herramienta de monitorización no puede acceder a las variables de las Clases, las cuales se consideran privadas. La solución a este problema es volver a invocar la ejecución del bloque **`INIT`** de los componentes, y utilizar una bandera global para diferenciar entre la primera ejecución, utilizada para extraer la topología y rellenar los datos globales, y esta segunda ejecución, empleada para transferir los resultados desde variables globales a variables locales.

6.3. Resultados del Desarrollo y Comparación con la Aproximación Clásica

6.3.1. Caso de Prueba 1

La Figura 6-1 muestra el esquemático del modelo de prueba 1 de la Librería de Hydraulic_ST.

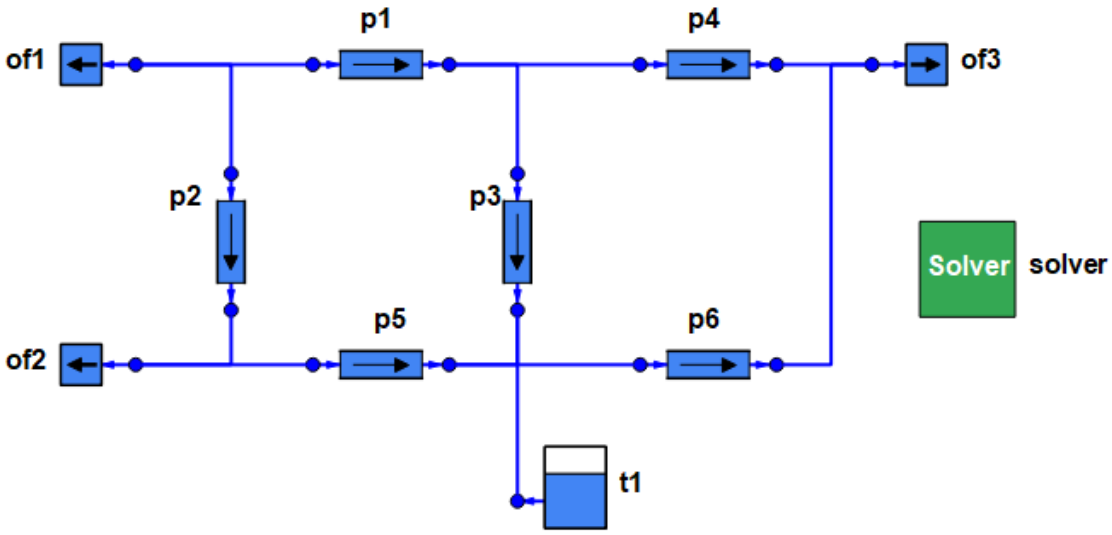


Figura 6-1: Diagrama del Modelo de Prueba 1

Este modelo tiene 6 tuberías, 1 tanque, 3 flujos salidas y 5 nodos, así que el número de incógnitas es:

Nº variables desconocidas = Nº nodos + Nº tuberías + Nº tanques = 5+6+1 = 12

Las cuatro tablas siguientes muestran los datos del modelo de prueba 1:

Tabla 6-1: Datos de Componentes "Pipe" del Modelo de Prueba 1

Component Name	**Pipe Length** (m)	**Pipe inside diameter** (m)	**Hazen Williams Coefficient**	**Initial flow** (m3/s)
p1	45.72	0.1016	130	0.01
p2	30.48	0.0762	130	0.01
p3	30.48	0.0762	130	0.01
p4	30.48	0.1016	130	0.01
p5	45.72	0.0762	130	0.01
p6	42.9768	0.0762	130	0.01

Tabla 6-2: Datos de Componentes "Tank" del Modelo de Prueba 1

Component Name	**Tank Piezometric Head (m)**	**Initial guess of Tank flow (m3/s)**
t1	30.48	0.01

Tabla 6-3: Datos de Componentes "`Outflow`" del Modelo de Prueba 1

Component Name	**Outflow** (m3/s)
of1	-0.028317
of2	0.007079
of3	0.014158

Tabla 6-4: Datos del Componente "solver" del Modelo de Prueba 1

Data	**Value**	**Units**	**Description**
Ho	30	m	Initial guess for the node piezometric head
tol	1.e-7	-	Tolerance of the NR
iter_damp	2	-	Number of initial damped iteration in NR algorithm
FRAC_damp	0.1	-	Damping factor for the first iter_damp iterations
iter_max	20	-	Maximum number of iterations in NR algorithm
NR_debug	1	-	Debug level for Newton-Raphson
STAMP_debug	0	-	Debug level for Stamping process

Resultados de la Simulación

Los resultados de la simulación caso de prueba 1 se muestran en la Figura 6-2.

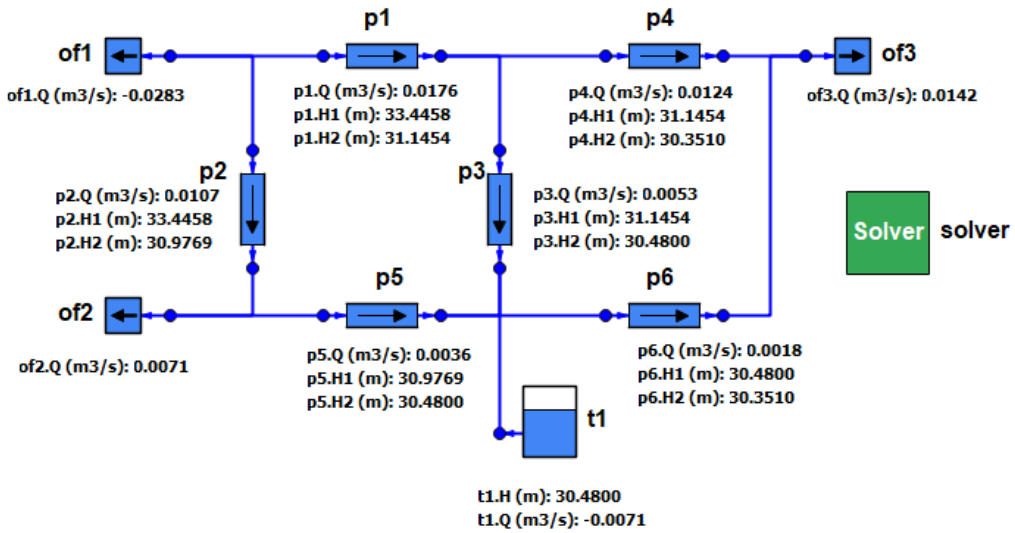


Figura 6-2: Resultados del Modelo de Prueba 1

Variable	**Unidades**	**Valor**
of1.H	m	33.4457879
of2.H	m	30.9769181
of3.H	m	30.3510085
p1.Q	m3/s	0.0176224217
p2.Q	m3/s	0.0106944253

Variable	Unidades	Valor
p3.Q	m3/s	0.00526848199
p4.Q	m3/s	0.0123539397
p5.Q	m3/s	0.00361521367
p6.Q	m3/s	0.0018044836

6.3.2. Caso de Prueba 2

La Figura 6-3 muestra el esquemático del modelo de prueba 2 de la Librería de ***HYDRAULIC_ST***. Este modelo es un modelo ficticio, que no pretende representar ningún sistema real. Se ha construido mediante repetición de partes y tiene un tamaño que se puede considerar medio.

Figura 6-3: Diagrama del Modelo de Prueba 2

El modelo de prueba 2 tiene 716 tuberías, 4 tanques, 12 flujos de salida y 396 nodos hidráulicos. Es un modelo de tamaño medio, con un número de tuberías que se aproxima al millar. El número de incógnitas, es:

Nº variables desconocidas = Nº nodos + Nº tuberías + Nº tanques = 396+716+4 = 1116

Las cuatro tablas siguientes muestran los datos de los componentes de este modelo. Por sencillez, todos los componentes de un tipo dado usan los mismos datos:

Tabla 6-5: Datos de Componentes "`Pipe`" del Modelo de Prueba 2

Component Name	**Pipe Length** (m)	**Pipe inside diameter** (m)	**Hazen Williams Coefficient**	**Initial flow** (m3/s)
Todos (1 a 796)	30	0.1	130	0.01

Tabla 6-6: Datos de Componentes "`Tank`" del Modelo de Prueba 2

Component Name	**Tank Piezometric Head (m)**	**Initial guess of Tank flow (m3/s)**
Todos (1 a 4)	100	0.10

Tabla 6-7: Datos de Componentes "`Outflow`" del Modelo de Prueba 2

Component Name	**Outflow** (m3/s)
Todos (1 a 12)	0.025

Tabla 6-8: Datos de Componentes "solver" del Modelo de Prueba 2

Data	**Value**	**Units**	**Description**
Ho	100	m	Initial guess for the node piezometric head
tol	1.e-7	-	Tolerance of the NR
iter_damp	1	-	Number of initial damped iteration in NR algorithm
FRAC_damp	0.1	-	Damping factor for the first iter_damp iterations
iter_max	20	-	Maximum number of iterations in NR algorithm
NR_debug	1	-	Debug level for Newton-Raphson
STAMP_debug	0	-	Debug level for Stamping process

Resultados de la Simulación

La Tabla 6-9 muestra un sumario de resultados del Caso 2, en particular, muestra el caudal de salida de cada uno de los 4 tanques (t1, t1_1, t1_2, t1_3) y las alturas piezométricas en los nodos a los que se conectan las 12 salidas de flujo (of1, of1_1, of1_2, of1_3,..).

Tabla 6-9: Sumario de Resultados del Modelo de Prueba 2

Variable	Unidades	Valor
t1.Q	m3/s	0.0753286744
t1_1.Q	m3/s	0.0718523504
t1_2.Q	m3/s	0.0765868553
t1_3.Q	m3/s	0.0762321198
of1.H	m	93.8066416
of1_1.H	m	94.1636727
of1_2.H	m	93.8441143
of1_3.H	m	93.7348364
of2.H	m	93.8625276
of2_1.H	m	94.3601702
of2_2.H	m	93.9253989
of2_3.H	m	94.0021503
of3.H	m	94.2558122
of3_1.H	m	95.0371641
of3_2.H	m	94.0609025
of3_3.H	m	94.0911371

6.3.3. Comparación con la Aproximación Clásica

Para comparar la nueva aproximación con la aproximación clásica, se ha desarrollado una librería completamente equivalente a la presentada en la sección anterior, pero utilizando la aproximación clásica en lugar de la nueva aproximación propuesta en esta tesis. En la sección F.2 del Apéndice F se proporciona el código fuente de esta librería, denominada ***HYDRAULIC_ST_CLASSIC***.

La librería ***HYDRAULIC_ST***, que implementa la nueva aproximación y cuyo código fuente completo se muestra en la sección F1, tiene 285 líneas de código. En cambio, la librería ***HYDRAULIC_ST_CLASSIC***, que implementa la aproximación clásica, tiene solo 56 líneas de código. Lógicamente, producir código de simulación más eficaz requiere mayor esfuerzo de programación. En este caso de una librería hidráulica simplificada, la nueva aproximación requiere aproximadamente 5 veces más líneas de código.

Comparación Caso de Prueba 1

Los resultados obtenidos con la nueva aproximación y con la aproximación clásica para un conjunto de variables representativas del caso de prueba 1 se muestran en la Tabla 6-13. Se puede observar que los resultados son totalmente idénticos, incluso la última cifra decimal es idéntica para todas las variables mostradas.

En cuanto al número de ecuaciones y el tiempo de computación (CPU), la Tabla 6-11 muestra los resultados de la comparación. La aproximación clásica consigue reducir el número de ecuaciones a la cuarta parte debido a que aplica técnicas de rasgadura. En cuanto a los tiempos de CPU, son muy pequeños, menos del milisegundo para ambas

aproximaciones, aunque la nueva aproximación consume aproximadamente el doble de tiempo que la clásica. Este mayor consumo de CPU solo ocurre con modelos muy pequeños. El siguiente caso mostrará que para modelos de tamaño medio la nueva aproximación consume mucho menos tiempo de computación.

Tabla 6-10: Comparación Resultados para Modelo Prueba 1 con Aproximación Nueva y Clásica

Variable	Unidades	Aproximación Nueva	Aproximación Clásica
of1.H	m	33.4457878	33.4457878
of2.H	m	30.9769181	30.9769181
of3.H	m	30.3510085	30.3510085
p1.Q	m3/s	0.0176224214	0.0176224214
p2.Q	m3/s	0.0106944252	0.0106944252
p3.Q	m3/s	0.00526848185	0.00526848185
p4.Q	m3/s	0.0123539396	0.0123539396
p5.Q	m3/s	0.00361521353	0.00361521353
p6.Q	m3/s	0.00180448373	0.00180448373

Tabla 6-11: Comparación Nº. Ecuaciones y Tiempos de CPU para Modelo Prueba 1

	Aproximación Clásica	Aproximación Nueva	Ratio Nueva a Clásica
Número Ecuaciones Implícitas	3	12	4
Tiempo de Computación (ms)	0.082	0.168	2.05

Tabla 6-12: Comparación Nº. Ecuaciones y Tiempos de CPU para Modelo Prueba 1

	Aproximación Clásica	Aproximación Nueva	Ratio Nueva a Clásica
Número Ecuaciones Implícitas	3	12	4
Tiempo de Computación (ms)	0.082	0.168	2.05

Ahora bien, otro tema importante a considerar es la robustez del método de solución. Una forma de poner a prueba la robustez es introducir valores muy grandes o muy pequeños para algunos de los datos y observar si el algoritmo sigue convergiendo, así como cuántas iteraciones son necesarias para lograrlo. Aunque esta prueba pueda parecer artificial, refleja situaciones reales donde el usuario de la simulación puede no conocer ciertos datos de un componente y tiene que hacer suposiciones sobre sus valores. Es común que, en tales situaciones, la herramienta de simulación no converja, dejando al usuario bloqueado. Por tanto, es importante que la herramienta sea capaz de calcular soluciones en condiciones donde el dimensionado de algunos componentes no corresponda al punto de operación deseado del sistema, permitiendo al usuario corregir los valores de los datos en función de la solución obtenida.

La prueba de robustez que se ha efectuado ha sido cambiar el valor del dato del diámetro del "***Pipe*** **p3** a un valor muy pequeño (`p3.D=1.e-6`) e intentar calcular el estacionario. La nueva aproximación calcula el estacionario sin dificultad en 25 iteraciones, mientras que la aproximación clásica no converge en 172 iteraciones. Esto demuestra que la nueva aproximación es mucho más robusta que la clásica.

Comparación Caso de Prueba 2

Los resultados obtenidos con la nueva aproximación y con la aproximación clásica para un conjunto de variables representativas se muestran en la Tabla 6-13. Se puede observar que los resultados son totalmente idénticos, incluso en la última cifra decimal.

Tabla 6-13: Comparación Resultados para Modelo Prueba 2

Variable	Unidades	Aproximación Nueva	Aproximación Clásica
t1.Q	m3/s	0.0753286744	0.0753286744
t1_1.Q	m3/s	0.0718523504	0.0718523504
t1_2.Q	m3/s	0.0765868553	0.0765868553
t1_3.Q	m3/s	0.0762321198	0.0762321198
of1.H	m	93.8066416	93.8066416
of1_1.H	m	94.1636727	94.1636727
of1_2.H	m	93.8441143	93.8441143
of1_3.H	m	93.7348364	93.7348364
of2.H	m	93.8625276	93.8625276
of2_1.H	m	94.3601702	94.3601702
of2_2.H	m	93.9253989	93.9253989
of2_3.H	m	94.0021503	94.0021503
of3.H	m	94.2558122	94.2558122
of3_1.H	m	95.0371641	95.0371641
of3_2.H	m	94.0609025	94.0609025
of3_3.H	m	94.0911371	94.0911371

La Tabla 6-14 muestra el número de ecuaciones implícitas y los tiempos de computación de ambas aproximaciones para este segundo caso. Con la nueva aproximación el número de ecuaciones es 1116. Utilizando la aproximación clásica, se aplica el algoritmo de rasgadura de la herramienta que reduce el número de ecuaciones a 328, lo que equivale a una reducción de 3.4 veces. Sin embargo y a pesar de esta reducción, el tiempo de cálculo de la aproximación clásica es considerablemente superior al de la nueva aproximación. La nueva aproximación utiliza técnicas numéricas para los sistemas de ecuaciones dispersos, las cuales son mucho más eficaces que aplicar la rasgadura (la rasgadura puede considerarse una técnica simbólica para sistemas de ecuaciones dispersos) y, continuación, efectuar una resolución numérica que no hace uso de la dispersión.

Tabla 6-14: Comparación Nº. Ecuaciones y Tiempos de CPU para Modelo Prueba 2

	Aproximación Clásica	Aproximación Nueva	Ratio Nueva a Clásica
Número Ecuaciones Implícitas	328	1116	3.402
Tiempo de Computación (ms)	369	24.1	0.065

6.4. Mejora de la Librería Hidráulica para Cálculo de Temperaturas

En el desarrollo de la librería hidráulica ***HYDRAULIC_ST***, se ha utilizado la formulación conocida Aproximación Nodal Modificada para calcular alturas piezométricas y flujos volumétricos.

En esta sección, se presenta la extensión de la aproximación nodal modificada, la cual hasta ahora se ha empleado para calcular alturas y flujos, al cálculo de variables como la temperatura y la composición. Estas variables se transportan por advección junto con el fluido. El propósito de esta explicación es proporcionar un punto de partida para transformar la librería hidráulica, cuyo desarrollo se ha presentado en este capítulo, en una librería termohidráulica.

Al aplicar la aproximación nodal modificada a una malla termohidráulica, es crucial comprender que se entiende por un nodo. Considérese el modelo de circuito eléctrico y el modelo termofluido dinámico mostrados en la Figura 6-4. En el modelo del circuito eléctrico, las líneas marcadas en rojo son cables con resistencia nula, por lo tanto, toda la zona coloreada en rojo constituye un nodo con un voltaje común.

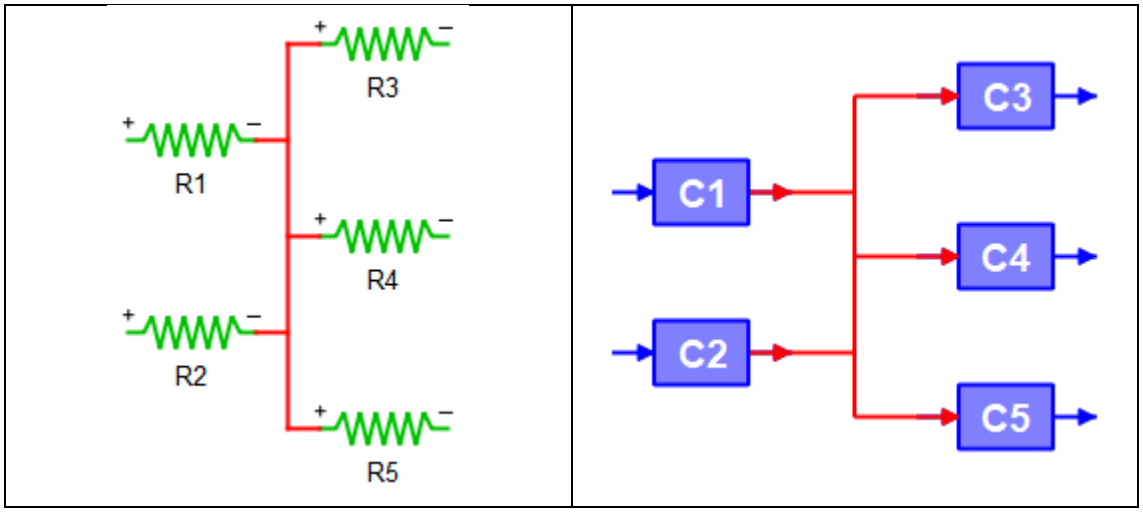


Figura 6-4: Comparación Conexiones en Modelo de Circuito Eléctrico (izquierda) y en Modelo Termofluido Dinámico (derecha)

Supóngase ahora que en el modelo termohidráulico de la Figura 6-4, las líneas marcadas en rojo son tuberías de caída presión despreciable, y que los puntos donde concurren dos líneas son conexiones en "T", también con una caída de presión insignificante. ¿Constituye un nodo la zona marcada en rojo? A pesar de las caídas de presión nulas, la zona resaltada en rojo en no se puede considerar como un nodo. En un nodo, se debe producir un mezclado perfecto de las corrientes que entran. Sin embargo, esto no se cumple en la configuración mostrada. Por ejemplo, si desde el componente **C1** entra una corriente de agua de 20 kg/s a 100 ºC, y desde el componente **C2** entra una corriente de agua de 10 kg/s a 50 ºC, y salen corrientes de 10 kg/s hacia los componentes **C3**, **C4** y **C5**, las temperaturas resultantes para las corrientes de salida a **C3**, **C4** y **C5** serían (suponiendo calor específico constante) 100 ºC, 75 ºC y 50 ºC, respectivamente, según se muestra en la Figura 6-5.

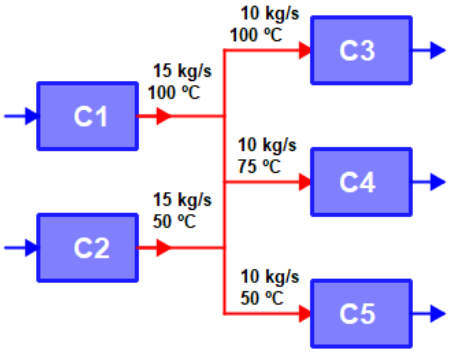


Figura 6-5: Ejemplo de Temperaturas en Conexiones Termofluido Dinámicas

Para que unas conexiones termofluido dinámicas funcionen como un nodo, tiene que existir un punto de mezclado perfecto. Por ejemplo, la configuración que se muestra en la Figura 6-6, el punto marcado con un circulo si constituye un nodo termo-fluido dinámico supuesto que representa un volumen en el que ocurre un mezclado perfecto.

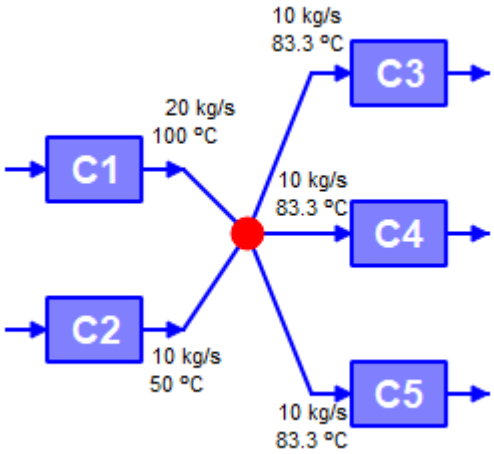


Figura 6-6: Ejemplo de Nodo Termofluido Dinámico

Aunque el concepto de nodo debe aplicarse con cuidado en el caso de problemas termofluido dinámicos, es un concepto bastante más potente que el de conexionado mediante corrientes, ya que permite el conexionado de múltiples corrientes termofluidas sin tener que implementar equipos mezcladores y divisores de corrientes. Además, desarrollar una formulación para calcular con precisión las temperaturas de los nodos en casos de flujos inversos no resulta complicado.

Para que una librería termo-hidráulica pueda manejar flujos inversos, se requieren dos condiciones: 1) que la formulación de los componentes admita flujo inverso, y 2) que la formulación para representar el mecanismo de conexión entre componentes, es decir, los nodos, también debe soportar flujo inverso.

En relación con el flujo inverso en los componentes, existen componentes en los que el flujo inverso es algo natural, como en ciertas tuberías de una red que pueden transportar flujo en ambos sentidos. Sin embargo, existen componentes en los que un flujo inverso solo puede ocurrir en condiciones excepcionales, como en una caldera de ebullición, donde los flujos de entrada de agua y de salida de vapor solo pueden invertirse en condiciones de operativas muy excepcionales.

A continuación, se presenta la formulación para describir un nodo fluido con cálculo de temperaturas y flujo inverso. En dicho nodo fluido se cumplen las siguientes ecuaciones:

Conservación de la Masa Total

Sea M el conjunto de las n conexiones al nodo: $M = \{1,2,3,\dots,n\}$. La ecuación de conservación de masa en el nodo puede expresarse como:

$$\sum_{i \in M} m_i = 0 \tag{6-12}$$

donde m_i es el flujo másico por la conexión i, considerado con signo positivo si entra al nodo y con signo negativo si sale del nodo.

Conservación del Momento o Igualdad de Presiones de Remanso

Típicamente en un nodo se asume igualdad de presiones de remanso. Las ecuaciones de igualdad de presiones de remanso de las distintas conexiones pueden expresarse como:

$$P_i = P_k \qquad \forall i,k \in M \tag{6-13}$$

donde P_i es la presión de remanso en la conexión i. Adviértase que solo hay $n-1$ ecuaciones independientes de igualdad de presión.

Conservación de la Energía en el Nodo

Para expresar la conservación de la energía es necesario separar las conexiones al nodo en 2 subconjuntos. Por un lado, el subconjunto de todas las conexiones al nodo con flujo másico positivo o nulo, $M^+ = \{i | m_i \geq 0\}$, y por otro lado, el subconjunto de todas las conexiones con flujo másico negativo, $M^- = \{j | m_j < 0\}$. En el caso de una corriente cuyo flujo entra al nodo, su entalpía viene fijada por el componente situado aguas arriba, y en el caso de una corriente cuyo flujo sale del nodo, su entalpía es la resultante de mezclar las corrientes de entrada. En consecuencia, la ecuación de conservación de la energía en un nodo, puede expresarse como:

$$\sum_{i \in M^+} m_i \, h_{up,i} - h_{mix} \sum_{j \in M^-} m_j = 0 \tag{6-14}$$

donde:

$h_{up,i}$ es la entalpía de remanso en la conexión i (supuesta con flujo positivo), que viene fijada por el componente situado aguas arriba.

h_{mix} es la entalpía de remanso resultante de la mezcla de las corrientes de entrada, se supone que todas las corrientes que salen del nodo poseen dicha entalpía.

Mientras que un método del tipo Newton-Raphson resuelve sin dificultad las ecuaciones de conservación de masa y de igualdad de presiones en los nodos, pues se trata de ecuaciones lineales y el método Newton-Raphson está precisamente basado en linealizar las ecuaciones, la ecuación de la energía presenta un desafío mayor. Esto se debe a que se trata de una ecuación no lineal, donde se tiene un sumatorio de productos de dos incógnitas: los flujos másicos por las entalpías correspondientes.

Al aplicar el procedimiento de Newton-Raphson, es común que durante las primeras iteraciones no se satisfagan ni la ecuación de conservación de masa ni la ecuación de conservación de energía en los nodos. Al aplicar la ecuación de la energía con flujos másicos erróneos durante las primeras iteraciones, se pueden obtener variaciones muy grandes de la entalpía en el nodo. Por ejemplo, si antes de la primera iteración, los flujos iniciales en el colector son todos positivos o entrantes al nodo, en ese caso el conjunto M^+ es el conjunto vacío, y la única opción para intentar satisfacer la ecuación de la energía es o anular todos los flujos o incrementar tremendamente h_{mix}.

Para evitar este problema, lo más conveniente es asumir se satisface la ecuación de conservación de masa antes de aplicar la ecuación de conservación de la energía. De manera que la suma de los flujos másicos que salen del nodo es igual a la suma de flujos másicos entrantes:

$$\sum_{j \in M^-} m_j = \sum_{i \in M^+} m_i \tag{6-15}$$

Reescribiendo la ecuación de la energía con esta suposición, queda:

$$\sum_{i \in M^+} m_i \left(h_{i,inlet} - h_{mix}\right) = 0 \tag{6-16}$$

Sin embargo, incluso con esta modificación, la ecuación de la energía puede seguir presentando problema si todos los flujos son negativos o nulos y el M^+ es el conjunto vacío, lo que resultaría en una ecuación de la forma $0 = 0$, que no proporciona ninguna información útil.

A fin de evitar este problema, existen al menos tres tipos de solución.

1. *Asignar una temperatura media en caso de flujo nulo:* Una primera solución consiste en considerar que, en caso de flujo nulo, la temperatura en el nodo es una media de las temperaturas de los fluidos en los componentes conectados.
2. *Agregar un término de intercambio de calor con el ambiente:* Una segunda opción, que puede considerarse más representativa de la física real del problema, es añadir un pequeño término que represente, al menos de forma cualitativa, el intercambio de calor con el ambiente.
3. *Considerar conducción de calor a lo largo de la tubería*: La tercera solución implica considerar una conducción de calor a lo largo de la tubería basada en la diferencia de temperatura de los nodos conectados.

Para la segunda opción, la ecuación de la energía con el nuevo término es:

$$\sum_{i \in M^+} m_i \left(h_{i,inlet} - h_{mix}\right) - K\left(T_{mix}(P, h_{mix}) - T_{amb}\right) = 0 \tag{6-17}$$

donde el coeficiente K es un coeficiente que sirve para calcular el intercambio de calor con el ambiente, y que puede ser muy pequeño, pues su propósito es evitar la indeterminación de las temperaturas en caso de flujos nulos. Con el nuevo término, en caso de que todos los flujos de entrada sean nulos, resulta $T_{mix} = T_{amb}$ y $h_{mix} = h_{mix}(P, T_{amb})$.

En el caso de las concentraciones, normalmente no tiene sentido físico considerar un intercambio de masa con el ambiente, y la opción más física sería considerar una conducción de masa a lo largo de las tuberías basada en las diferencias de concentración en los nodos de entrada y salida.

6.5. Conclusiones

La nueva aproximación para el cálculo de estacionarios ofrece una serie de ventajas que se describen a continuación:

- Permite desarrollar aplicaciones mucho más eficaces que usando la aproximación clásica, pero con un esfuerzo de programación mayor.
- Permite utilizar formulaciones diferentes para el estacionario y el transitorio. Esto es un inconveniente de la aproximación convencional, pues hay casos en que la formulación estacionaria puede simplificarse. En el capítulo anterior se vio que una limitación de las herramientas de MSOO era que obligaban a una formulación igual para el transitorio y el estacionario, y que esto conduce en ocasiones a sistemas de ecuaciones innecesariamente grandes para el cálculo estacionario.
- Permite también obtener soluciones en el dominio de la frecuencia.
- Permite el uso de cualquier método de solución, ya sea orientado a ecuaciones o modular secuencial

Ahora bien, la nueva aproximación también presenta una serie dificultades y de desventajas:

- Requiere un mayor esfuerzo de desarrollo.
- Es difícil de implementar en el caso de uso de herencia en la codificación de los componentes.
- Solo está soportada por las librerías diseñadas con este tipo de cálculo estacionario, así que la simulación multidisciplinar junto con librerías desarrolladas de la forma convencional no está soportada.

7. DESARROLLO DE UNA APLICACIÓN PARA TRANSITORIOS HIDRÁULICOS

En este capítulo, se presenta la librería ***LiqHammer_ST***, diseñada específicamente para el análisis de transitorios hidráulicos, haciendo uso de la nueva aproximación presentada en la sección 5.3. Asimismo, se comparan los resultados obtenidos con esta librería y los obtenidos con una herramienta específica de análisis de transitorios hidráulicos, AFT Impulse. Ambas herramientas, ***LiqHammer_ST*** y AFT Impulse, utilizan el método de las características. En el Apéndice G se presentan las ecuaciones del golpe de ariete y dos métodos para su solución: el método de las características y el método de diferencias finitas implícitas.

Es importante señalar que la librería ***LiqHammer_ST*** se presenta como un ejemplo académico y le faltan muchas de las características requeridas por un código dedicado a uso profesional. Por ejemplo, no incluye un cálculo de estacionarios y su paleta de componentes es bastante limitada.

7.1. Requisitos de Usuario

El primer paso antes de emprender el desarrollo de cualquier aplicación de simulación es definir unos requisitos de usuario de la aplicación. Los requisitos de la aplicación que se desarrolla como ejemplo, son bastante limitados, ya que su propósito es solo ilustrar la aproximación propuesta:

Requisito 1: El propósito de la aplicación es el cálculo de transitorios hidráulicos en redes de tuberías. Las variables cuya evolución transitoria se desea obtener son alturas piezométricas y flujos volumétricos. Se supone que la densidad del líquido es constante.

Requisito 2: El método de cálculo será el método de las características, y la implementación del mismo debe ser eficaz, ofreciendo tiempos de cálculo similares a los de las herramientas comerciales.

Requisito 3: La formulación debe ser capaz de representar la cavitación transitoria utilizando el modelo DVCM (Discrete Vapor Cavity Model).

Requisito 4: La librería deberá incluir dos categorías de componentes: tuberías y condiciones de contorno.

Requisito 5: Las condiciones de contorno incluirán componentes para representar: tanques, entradas de flujo, salidas de flujo, colectores, válvulas de salida, válvulas de dos vías, válvulas antirretorno y bombas

7.2. Diseño de la Librería y Formulación

El método de las características aplicado a la resolución de las ecuaciones del golpe de ariete es un método que es fácilmente programable utilizando lenguajes de programación de propósito general.

La Figura 7-1 muestra la malla rectangular en el espacio y en el tiempo que el método de las características considera en cada una de las tuberías del modelo. Una breve descripción del método de las características puede verse en el Apéndice G.

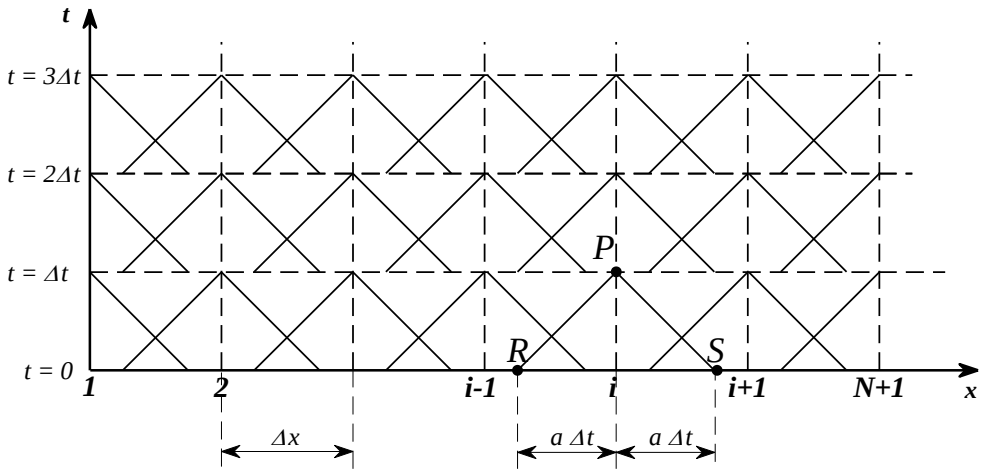


Figura 7-1: Malla Rectangular para la Aplicación del MOC a una tubería

El principal desafío para aplicar el método de las características (MOC) en una herramienta de MSOO radica en lograr una implementación eficiente. Dado que el MOC es un método discreto, tiene el potencial de introducir discontinuidades, y su interacción con los resolvedores de ecuaciones diferenciales rígidas de las herramientas MSOO puede obstaculizar la eficiencia de la simulación. En esta sección, se presentan las técnicas y trucos de codificación utilizados para obtener una implementación eficiente del MOC en una herramienta de MSOO.

7.2.1. Implementación del Cálculo de Secciones Internas

El cálculo mediante el MOC de las secciones internas de una tubería se puede realizar en una herramienta MSOO mediante ecuaciones discretas activadas por un generador de eventos con un período de muestreo igual al paso de tiempo del MOC. Es posible utilizar la técnica indicada en la sección 5.3 y encapsular la gestión del cálculo de las secciones internas en un objeto, de forma que las secciones internas queden ocultas y la herramienta de MSOO no tenga que procesar las ecuaciones correspondientes a los nodos internos.

7.2.2. Implementación del Cálculo de Condiciones de Contorno

En la implementación tradicional del Método de Características Móviles (MOC) utilizando un lenguaje de programación de propósito general, las condiciones de contorno se abordan de manera discontinua, resolviéndose únicamente en momentos de tiempo discretos. Sin embargo, Turpin (2004) ofreció una solución innovadora a esta limitación al proponer la interpolación continua en el tiempo de las ecuaciones características en los bordes o límites de las tuberías. Esta técnica, como se ilustra en la Figura 7-2, permite una actualización fluida de las condiciones de contorno a lo largo del tiempo, superando la restricción de la resolución discreta en momentos específicos. La aplicación de esta metodología se ha incorporado de manera efectiva en la librería ***LiqHammer_ST***, posibilitando así una solución continua y precisa de las condiciones de contorno en los modelos de sistemas de fluidos.

Este enfoque permite una representación más fiel de los fenómenos transitorios en dichos sistemas, ya que elimina las discontinuidades inherentes a la implementación tradicional. Al posibilitar la resolución continua de las condiciones de contorno a lo largo del tiempo, se logra una integración más precisa de las variables del sistema.

Además, este método ofrece esta mejora sin un sobrecoste computacional significativo.

Por ejemplo, en comparación con la transmisión de calor unidimensional en una pared, donde se necesita resolver una matriz tridiagonal en cada paso intermedio de tiempo (véase la sección 5.3.2), en este caso solo se requiere una interpolación lineal de las propiedades fluidas en el punto S y el cálculo de los coeficientes de la ecuación característica C^- que pasa por dicho punto. Esta eficiencia computacional lo convierte en una solución eficiente y viable para aplicaciones prácticas.

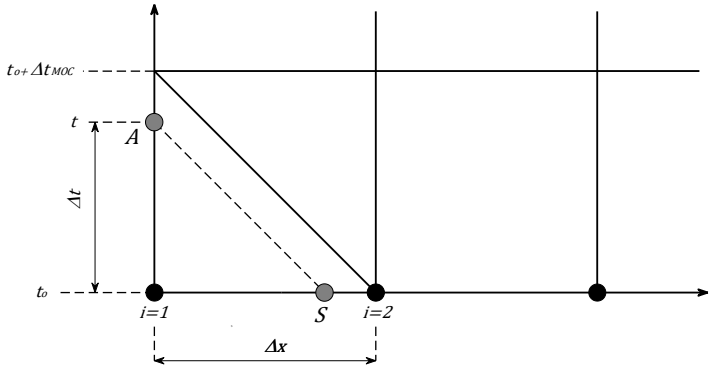


Figura 7-2: Interpolación Continua de las Características en los Nodos Extremos

Un aspecto que conviene aclarar acerca de la implementación de las condiciones de contorno, es que en la implementación convencional del método de las características utilizando un lenguaje de programación general, se encuentran soluciones analíticas para

la resolución de muchas de las condiciones de contorno. Por ejemplo, las válvulas ya sean de salida o interiores y los depósitos se reducen a ecuaciones de segundo grado con solución analítica, y los colectores se reducen a una ecuación lineal también resoluble de forma analítica. En cambio, en la implementación aquí presentada, la herramienta de MSOO es capaz de identificar los diferentes lazos algebraicos independientes en cada una de las condiciones de contorno. Incluso puede identificar los lazos lineales y resolverlos utilizando funciones para la resolución de sistemas de ecuaciones lineales. Sin embargo, cuando se trata de lazos no lineales, que pueden transformarse en ecuaciones algebraicas de segundo orden, el procesamiento simbólico de la herramienta no posee la suficiente inteligencia para lograr dicha transformación. En consecuencia, la herramienta resuelve las condiciones de contorno que generan lazos algebraicos no lineales mediante un proceso que comienza con la realización de una rasgadura, con el propósito de reducir el tamaño del sistema de ecuaciones; seguidamente, se aplica un resolvedor numérico de sistemas de ecuaciones no lineales para obtener la solución deseada. Es conveniente utilizar las directivas de guiado de la rasgadura disponibles en EcosimPro para identificar claramente las variables y ecuaciones de rasgadura.

7.2.3. Discretización y Sincronización de Todas las Tuberías del Modelo

En la implementación del Método de Características Móviles (MOC) con un lenguaje de programación de propósito general, es práctica habitual emplear el mismo paso de tiempo para integrar las ecuaciones en todas las tuberías del modelo. Esto se debe a que mantener un paso de tiempo común simplifica el proceso de cálculo y garantiza una integración coherente en todo el sistema.

Aunque la implementación del MOC utilizando una herramienta MSOO no necesariamente requiere todas las tuberías tengan el mismo paso de tiempo, dado que teóricamente la herramienta puede gestionar los eventos provenientes de todas las tuberías del modelo, ***LiqHammer_ST*** está diseñada para considerar un paso de tiempo común en todas las tuberías. Esta decisión de diseño se realiza con el objetivo de alcanzar una eficiencia similar a la del software comercial de golpe de ariete.

Si la implementación del MOC con una herramienta de MSOO se diseñara de manera que cada tubería utilizara su propio paso de integración, el número de eventos durante la integración aumentaría proporcionalmente con el número de tuberías en el modelo. Este incremento en la cantidad de eventos puede ser problemático, ya que los resolvedores de paso variable típicamente utilizados en las herramientas MSOO tienden a volverse muy ineficientes cuando se enfrentan a múltiples eventos (Casella, 2015). Cada evento requiere una re-inicialización del proceso de integración, lo que puede generar una sobrecarga significativa en términos de tiempo de cálculo. Por el contrario, al emplear el mismo paso de tiempo para todas las tuberías, ***LiqHammer_ST*** consigue que el número de eventos

discretos sea independiente del número de tuberías en el modelo. Esta disminución en el número de eventos contribuye a una integración más eficiente.

La selección de un paso de tiempo común está estrechamente relacionada con la elección del número de tramos en cada tubería, debido a la restricción impuesta por el número de Courant (Cr), que tiene la siguiente forma:

$$C_{rj} = \frac{a_j \, \Delta t}{L_j / \, N_j} \leq 1 \qquad (7\text{-}1)$$

donde C_{rj} es el número de Courant en la *j-ésima* tubería, a_j es la velocidad de onda en la *j-ésima* tubería, Δt el incremento de tiempo común, L_j es la longitud de la *j-ésima* tubería y N_j es el número de intervalos entre secciones (o nodos de cálculo) en la *j-ésima* tubería.

El MOC es altamente preciso cuando el número de Courant es igual a uno en todas las tuberías del modelo. Sin embargo, para la mayoría de los sistemas de tuberías, resulta imposible satisfacer esta condición exactamente en cada tubería del modelo con unos valores razonables del incremento de tiempo (Δt) y del número de nodos (N_j) en cada tubería. Para abordar este problema y lograr que el paso de integración de todas las tuberías sea idéntico, existen dos opciones prácticas.

La primera opción es el llamado "método de ajuste de la velocidad de onda", que modifica la velocidad de la onda para cumplir exactamente con la condición de Courant igual a uno. Esta opción ajusta ligeramente la velocidad de la onda con el factor de ajuste Ψ_J para encontrar un número entero de intervalos y se expresa mediante la ecuación:

$$\Delta t = \frac{L_J}{a_J\left(1 \pm \Psi_J\right) N_J} \qquad \rightarrow \quad N_J = \frac{L_J}{a_J \Delta t\left(1 \pm \Psi_J\right)} \qquad (7\text{-}2)$$

La segunda opción es permitir que el número de Courant sea menor que uno e interpolar las características entre puntos de la malla conocidos. Los métodos de interpolación más comunes son la interpolación lineal en el espacio en un nivel de tiempo fijo, y la interpolación lineal en el tiempo en ubicaciones fijas. Sin embargo, las interpolaciones producen la dispersión de las ondas e introducen un amortiguamiento numérico artificial que suaviza los frentes de onda.

En ***LiqHammer_ST*** se han implementado ambas opciones mediante un algoritmo de discretización descrito por Karney & Ghidaoui (1997). Este algoritmo admite la combinación de las dos opciones anteriores y constituye una herramienta flexible para investigar la importancia de los errores de discretización en sistemas de tuberías.

Desde el punto de vista de la programación con un lenguaje de MSOO, la principal dificultad de implementar este algoritmo de discretización radica en la necesidad de definir bucles sobre todos los componentes del tipo “tubería” en un modelo. Los lenguajes MSOO

proporcionan ocultamiento de información, lo que significa que cada componente solo comparte sus variables de puerto con otros componentes. Por lo tanto, se requiere rellenar arreglos de variables globales con los datos de los componentes del tipo “tubería” y asignar números enteros consecutivos a todos los componentes de este tipo en el modelo. Estos números enteros son identificadores que permiten asociar elementos de los arreglos globales de variables a instancias específicas de los componentes del tipo tubería. La técnica de programación a utilizar es similar a la explicada en la sección D.2 del Apéndice D.

7.3. Descripción General de la Librería *LIQHAMMER_ST* para Usuarios

La Figura 7-3 muestra la paleta de componentes de la librería ***LiqHammer_ST***. La paleta se divide en tres grupos de componentes: tuberías, condiciones de contorno y resolvedor. El grupo de tuberías incluye dos tipos de componentes: el ***"Pipe"***, que es el componente estándar de tubería que utiliza el MOC, y el ***"PipeImpl"***, que está diseñada para representar tuberías muy cortas.

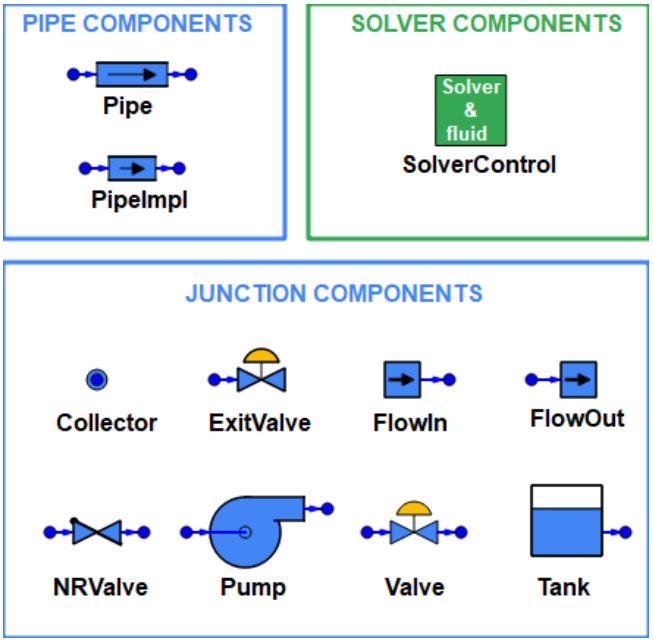


*Figura 7-3: Paleta de Componentes de la Librería **LiqHammer_ST***

El componente ***"Pipe"*** implementa dos métodos opcionales: el MOC y el método AWH (Wylie et al. 1993). El método AWH es una variante del MOC que aplica las ecuaciones características entre las secciones inicial y final de las tuberías evitando el cálculo de los nodos internos. Aunque tiene la desventaja de que su cálculo de fricción es menos preciso, es adecuado para el análisis de sistemas de distribución de agua con miles de tuberías. La

opción MOC también implementa fricción transitoria y el modelo para cavitación transitoria DVCM (Discrete Vapor Cavity Model).

Como se ha indicado en el apartado anterior, la implementación efectiva del MOC y el AWH requiere un paso de tiempo igual en toda la red. Ocurre que cada tubería debe tener al menos una sección, y el paso de tiempo está limitado por el número de Courant. En el caso de un sistema que incluye tuberías con tiempos de transito de onda muy diferentes, la simulación se vuelve extremadamente costosa porque la tubería con el tiempo de transito de onda más corto limita el paso de tiempo global y la distancia entre nodos para todas las demás tuberías. A fin de evitar este tipo de problema, se ha incluido un componente denominado ***"PipeImpl"***, que utiliza una formulación de diferencias finitas implícita propuesta por Karney (1984) cuyo paso de tiempo no está limitado por el Courant. Este componente se puede utilizar para representar tuberías con un tiempo de transito de onda muy breve.

Un modelo de golpe de ariete se construye como una red de tuberías interconectadas por elementos de contorno donde se encuentran dispositivos de flujo y accesorios. Las condiciones de contorno incluyen tanques, colectores, válvulas de salida y válvulas internas, válvulas de retención, extremos muertos, cambios de área, bombas, válvulas de entrada/salida de aire, acumuladores de gas y tanques de sobrepresión.

Se debe incluir un componente especial de tipo ***"SolverControl"*** en cada modelo de ***LiqHammer_ST***, discretizar el modelo utilizando el algoritmo de Karney & Ghidaoui (1997) y seleccionar el solucionador transitorio (MOC o AWH) y sus parámetros operativos.

Es importante destacar que gracias al uso de un lenguaje de MSOO, ***LiqHammer_ST*** requiere solo alrededor de 1500 líneas de código. Un código equivalente escrito en Fortran para el análisis de golpe de ariete utilizando el MOC, requiere más de 5000 líneas sin proporcionar ninguna capacidad gráfica interactiva.

7.4. Casos de Ejemplo y Comparación con Otro Software de Golpe de Ariete

Se han definido casos de prueba cuidadosamente seleccionados de la literatura sobre golpe de ariete para verificar y validar los diversos modelos de componentes en el código de ***LiqHammer_ST***. Los resultados para un subconjunto relevante de tres casos de prueba se presentan en esta sección. Estos casos también se utilizaron para comparar la eficiencia y precisión del nuevo código con otro software de golpe de ariete.

7.4.1. Caso de Ejemplo 1: Red de Tuberías con Cierre Instantáneo de Válvula

El primer ejemplo es un caso de referencia propuesto por Merilo (1992). La Figura 7-4 muestra el modelo, que incluye un tanque atmosférico, nueve tuberías y una condición de flujo variable que representa una válvula. La condición inicial es un estado estacionario calculado para una descarga de salida de 0.8495 m³/s, y el transitorio se genera mediante el cierre instantáneo del flujo de salida en el tiempo $t = 0$. El propósito del caso es verificar el componente de tubería y las diferentes opciones de solucionador: MOC o AWH, esquema de interpolación y fricción transitoria o estacionaria.

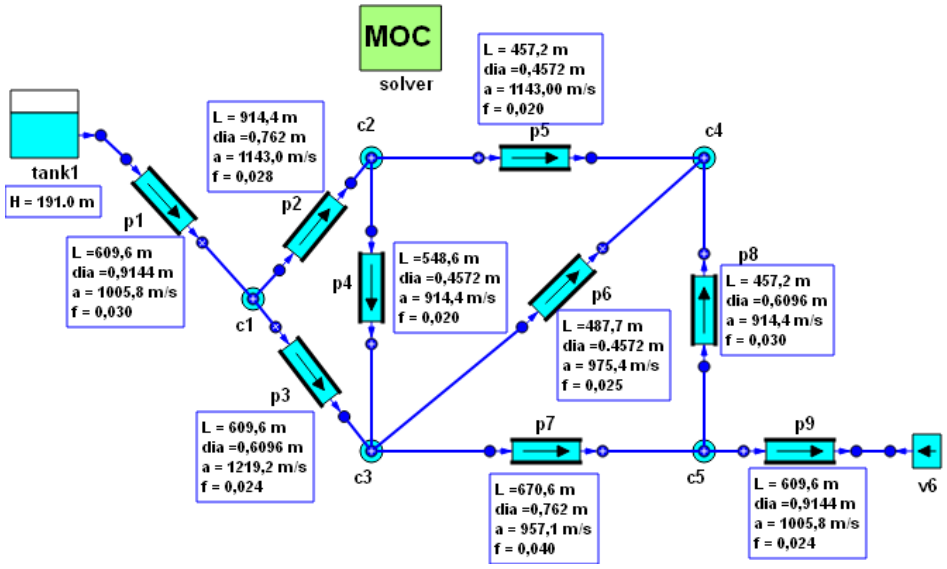


Figura 7-4: Modelo del Caso de Ejemplo 1

La Figura 7-5 presenta una comparación entre los resultados para la altura piezométrica en el componente que representa la condición de flujo (v6), obtenidos mediante el uso de ***LiqHammer_ST*** y con AFT Impulse. Esta comparación muestra una consistencia satisfactoria entre ambos resultados, lo que confirma la adecuada implementación del Método de las Características (MOC) para los tipos de componentes empleados en el modelo.

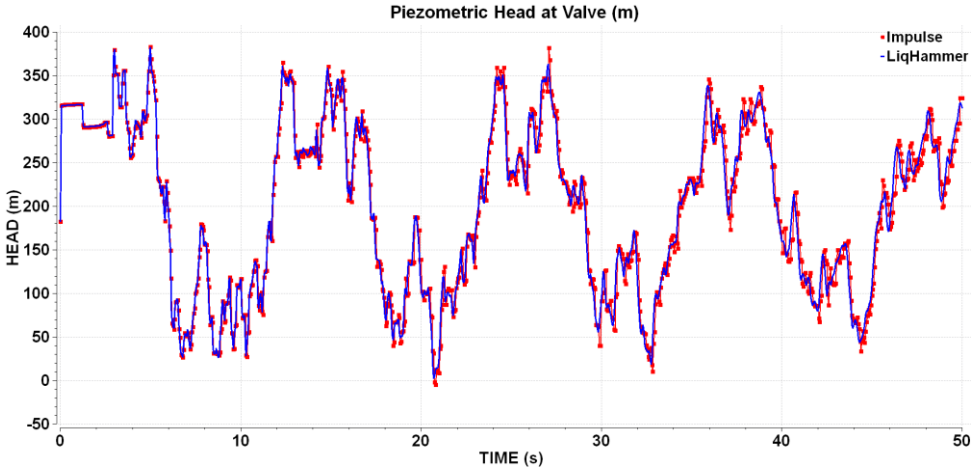


Figura 7-5: Resultados del Caso de Ejemplo 1

7.4.2. Caso de Ejemplo 2: Parada de Bomba

El segundo ejemplo, mostrado en la Figura 7-6, se toma del libro de Chaudhry (2014). Representa un sistema de bombeo y su objetivo es verificar el componente de la bomba. Inicialmente, la bomba está operando en condiciones nominales y el transitorio se inicia con la parada de la bomba debido a una pérdida de energía.

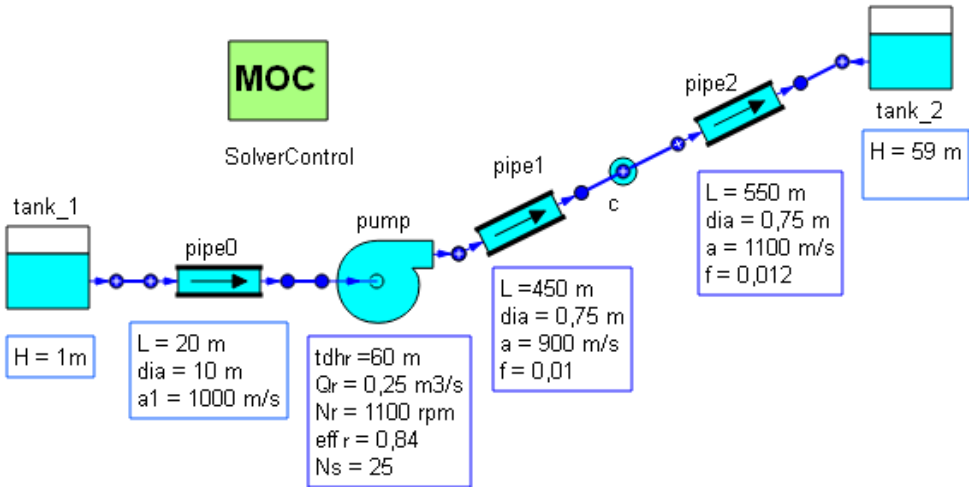


Figura 7-6: Modelo del Caso de Ejemplo 2

La Figura 7-7 muestra los resultados calculados para la velocidad de giro adimensional (fracción de la velocidad de giro respecto a la velocidad de giro nominal) y el caudal adimensional (caudal dividido por el caudal nominal durante el transitorio). La Figura 7-8 muestra la altura piezométrica en la descarga de la bomba y en la unión entre las dos tuberías de descarga. Ambas figuras también muestran las predicciones proporcionadas por Chaudhry (2014) para este caso. Existe una buena concordancia entre los resultados

de la nueva aproximación y los de la literatura. ***LiqHammer_ST*** reproduce con gran precisión las fases sucesivas del transitorio: la rápida reducción de la velocidad de la bomba, la inversión del flujo, el funcionamiento de la bomba como turbina y el aumento final de la presión.

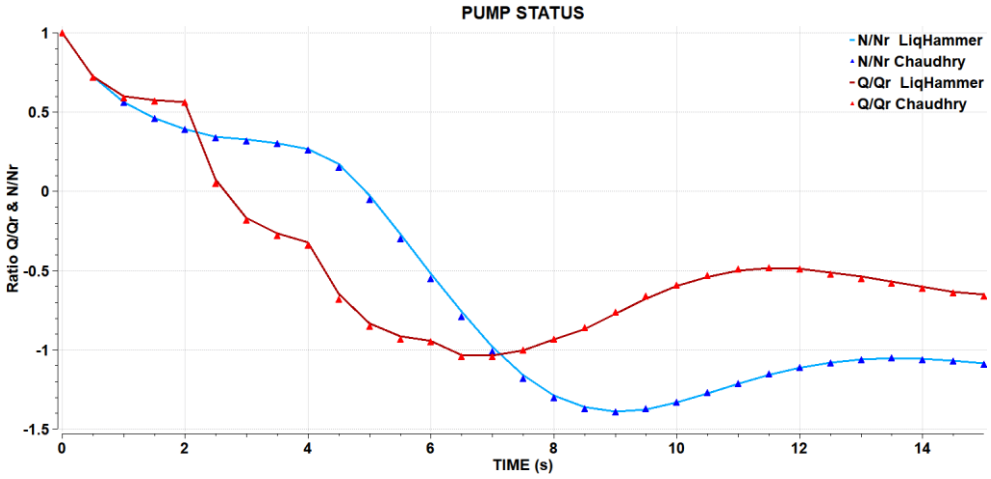


Figura 7-7: Resultados del Caso de Ejemplo 2 - Velocidad y Flujo Adimensionales

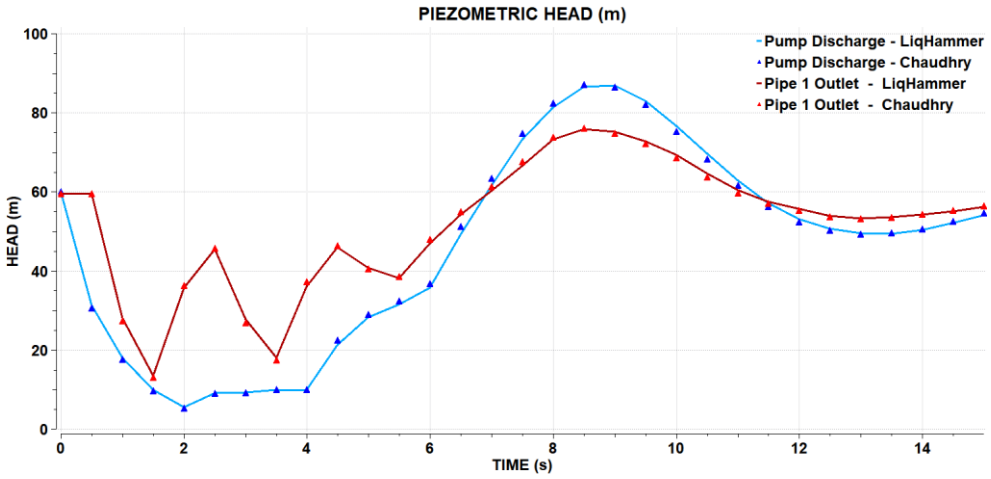


Figura 7-8: Resultados del Caso de Ejemplo 2 - Alturas Piezométricas

7.4.3. Caso de Ejemplo 3: Cavitación Transitoria

Este caso, tomado de (Wylie et al. 1993), se seleccionó para verificar la implementación del modelo DVCM que representa la cavitación transitoria. La Figura 7-9 muestra el modelo, que consta de una tubería con pendiente ascendente en la dirección del flujo. El flujo de entrada se impone en el límite aguas arriba y la tubería está conectada a un depósito aguas abajo cuyo nivel es de 15 m. El flujo inicial en la tubería es de 0.4 m³/s y se detiene repentinamente para iniciar el transitorio en el tiempo cero.

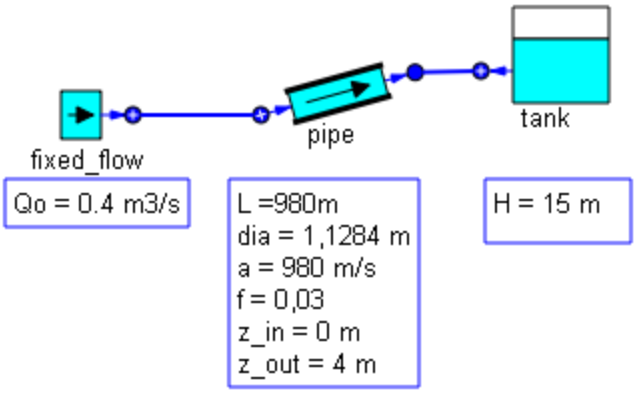


Figura 7-9: Modelo del Caso de Ejemplo 3

Figura 7-10 compara los resultados del nuevo código con los resultados calculados con AFT Impulse. La altura piezométrica en la entrada de la tubería es prácticamente la misma y solo aparecen algunas diferencias menores después del tercer pulso. Estas diferencias menores pueden atribuirse al hecho de que el DVCM puede generar pulsos de presión (picos) poco realistas debido al colapso de multicavidades.

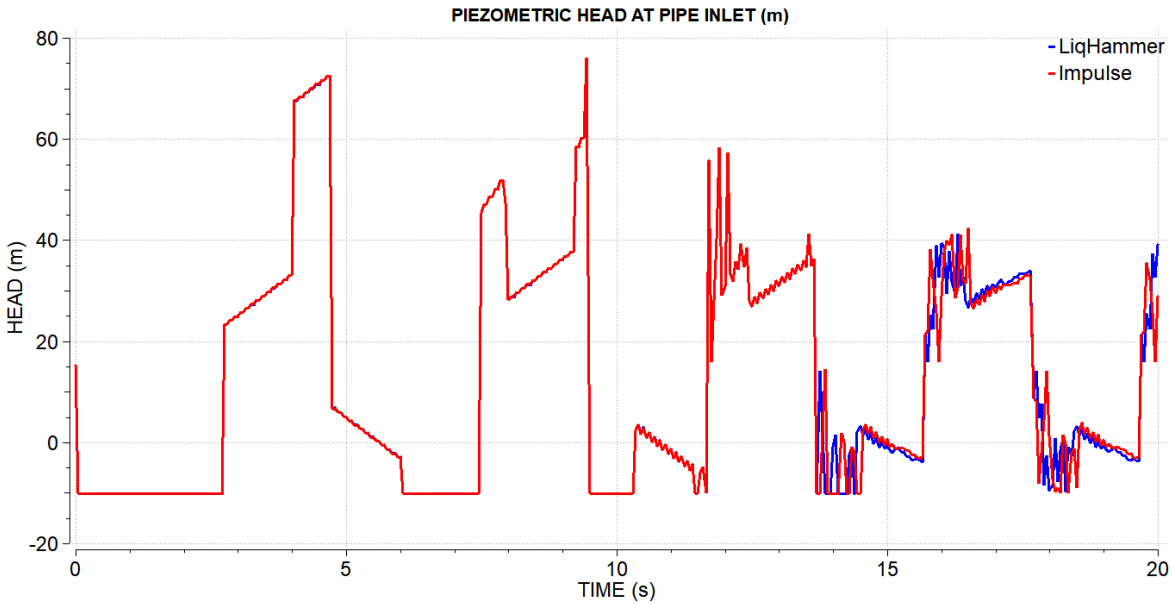


Figura 7-10: Resultados del Caso de Ejemplo 3

7.5. Comparación de la Nueva Aproximación contra Software Existente de Golpe de Ariete

En el capítulo 4 se ha proporcionado una comparación entre la aproximación clásica del MSOO aplicada la simulación de transitorios hidráulicos y un programa específico para el cálculo de transitorios hidráulicos que emplea el método de las características. Allí se

concluyó, que la aproximación clásica del MSOO aplicada a los transitorios hidráulicos es altamente ineficaz en el caso de transitorios rápidos.

Por tanto, en esta sección, solo se proporciona una comparación entre, ***LiqHammer_ST***, la aplicación de golpe de ariete desarrollada con una herramienta de MSOO y que aplica el método de las características, y un programa comercial de golpe de ariete, AFT Impulse, desarrollada con un lenguaje de programación general.

Se ha analizado el caso ejemplo 1 con ambas herramientas, ***LiqHammer_ST*** y AFT Impulse, y se ha realizado un estudio paramétrico modificando la nodalización, que se define a través de un único dato, observando su efecto en la precisión y el tiempo de cálculo.

El error en la sobrealtura calculada en la válvula se mide utilizando la ecuación (7-3), donde $H_{valve\,max,\ comp}$ es la máxima altura piezométrica calculado en la válvula, H_{tank} es la altura piezométrica en el tanque, igual a 191 m, y $H_{valve\,max,\ ref}$ es el valor de referencia para la máxima altura piezométrica en la válvula obtenido a partir de un cálculo con 400 nodos en la tubería de control, es igual a 285. m.

$$error = 1 - \frac{H_{valve\,max,\ comp} - H_{tank}}{H_{valve\,max,\ ref} - H_{tank}} \tag{7-3}$$

La computadora utilizada para ejecutar las pruebas es un Intel i5-4570 CPU @ 3.20 GHz, con 8.00 GB de RAM. La versión de EcosimPro utilizada es la 5.10.2 con la versión de GNU C++ 4.7.

Los resultados obtenidos para el ejemplo 1 en función de la discretización se resumen en la Tabla 7-1 y se muestran gráficamente en la Figura 7-11 y la Figura 7-12. La primera columna de la tabla muestra el número de tramos o intervalos entre nodos en la tubería de control, que es el valor que define el refinamiento de la discretización. Las columnas siguientes muestran el tiempo de cálculo, la altura máxima en la válvula de salida y el error en las sobrealtura máxima en la válvula, primero con la nueva aproximación y luego con un código comercial.

*Tabla 7-1: Comparación de los Resultados de **LiqHammer_ST** y AFT Impulse para el Caso Ejemplo 1*

	Nueva Aproximación *LiqHammer_ST*			**Herramienta Específica AFT Impulse**		
No. de tramos en tubería de control	**Tiempo de CPU (s)** (s)	**Max. Altura en Válvula** (m)	**Error** (%)	**CPU Time** (s)	**Max. Altura en Válvula** (m)	**Error** (%)
4	0.17	383.36	0.58%	0.09	383.51	0.66%
8	0.14	383.14	0.47%	0.20	383.23	0.51%
12	0.20	383.21	0.50%	0.28	383.29	0.54%
16	0.25	383.14	0.47%	0.39	383.21	0.50%
20	0.34	383.18	0.49%	0.52	383.25	0.52%
24	0.41	383.14	0.47%	0.66	383.20	0.50%
28	0.50	383.17	0.48%	0.82	383.23	0.51%
32	0.59	383.14	0.47%	1.00	383.20	0.50%
36	0.71	381.50	-0.39%	1.20	381.50	-0.39%
40	0.83	381.62	-0.33%	1.46	381.63	-0.32%
60	1.49	382.07	-0.10%	3.00	382.13	-0.06%
80	2.14	382.37	0.06%	4.37	382.44	0.10%
100	3.03	382.49	0.13%	6.29	382.6	0.18%

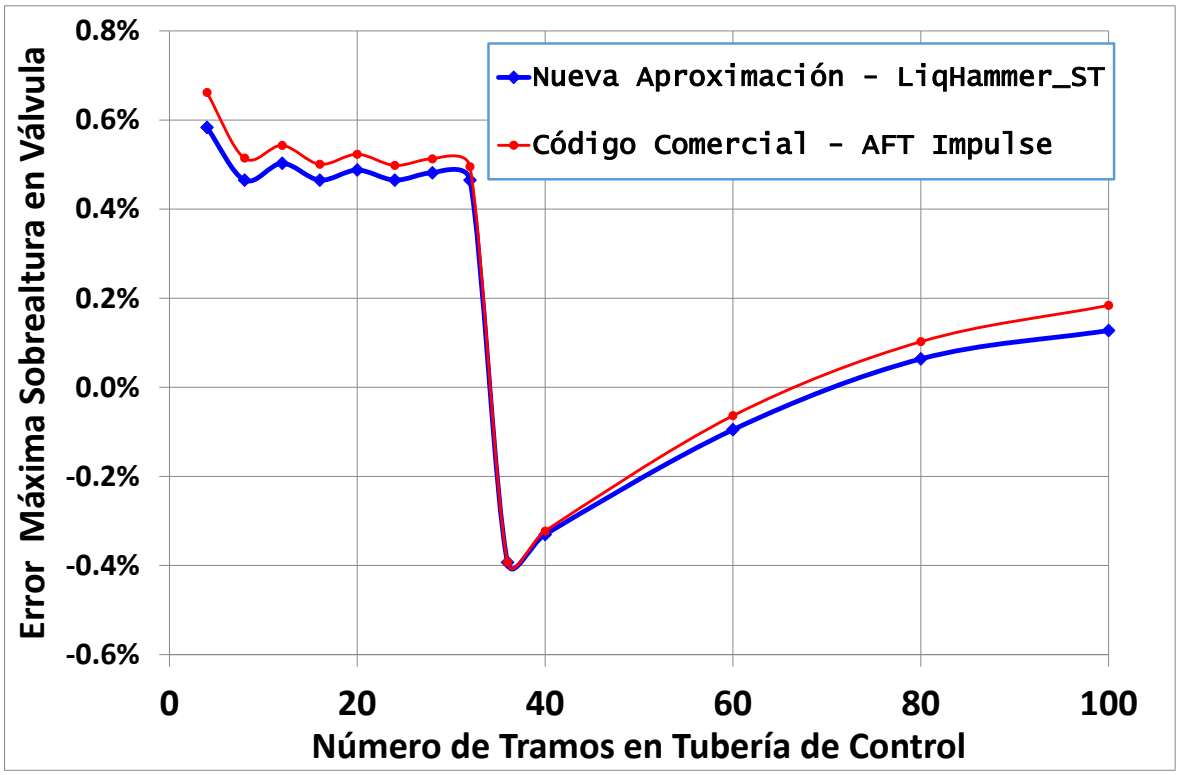


Figura 7-11: Error en la Máxima Altura Calculada en la Válvula para el Ejemplo 1 en Función de la Discretización

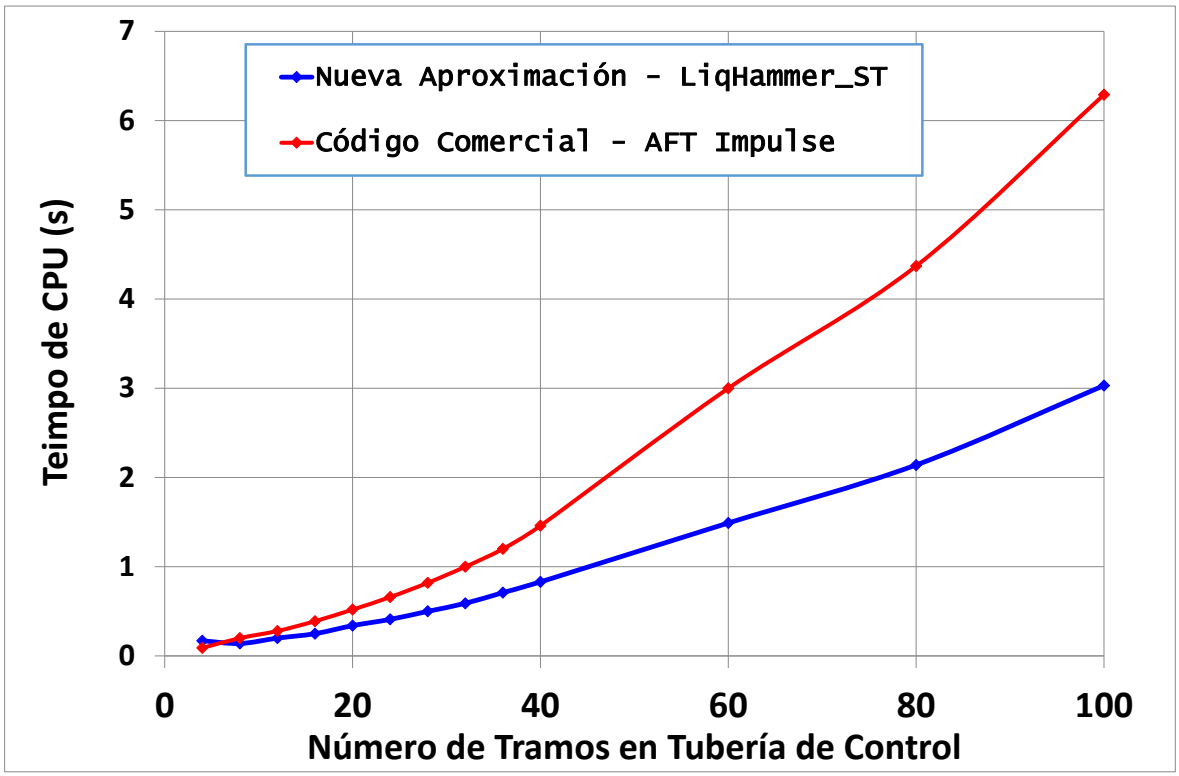


Figura 7-12: Tiempos de Cálculo para el Ejemplo 1 en Función de la Discretización

La Figura 7-11 muestra el error de la sobre altura piezométrica máxima calculada en la válvula versus el número de tramos en la tubería de control. ***LiqHammer_ST*** y AFT Impulse ofrecen una precisión casi idéntica, como se esperaba, ya que ambos utilizan el mismo método. Además, los códigos MOC logran una precisión muy alta incluso con un pequeño número de tramos; el error de la máxima sobrealtura en la válvula es inferior al 0.6% con solo 4 tramos en la tubería de control. En realidad, el refinar la discretización produce bastante poco beneficio en la precisión en los resultados.

La Figura 7-12 muestra los tiempos de cálculo de LiqHammer_ST y de AFT Impulse. El. Los tiempos de cálculo de la nueva aproximación, aunque inferiores, tienen el mismo orden de magnitud que los del código comercial. Sorprendentemente, el nuevo código es ligeramente más rápido que el código comercial para este caso.

7.6. Conclusiones

La conclusión fundamental es que la nueva aproximación ha permitido diseñar una librería que utiliza un método mucho más eficiente que los métodos semidiscretos que se hubieran seleccionado utilizando la aproximación convencional (ver capítulo 4). La eficacia de la nueva aplicación, en cuanto a precisión y tiempos de cálculo, es comparable a la de los códigos comerciales de golpe de ariete.

8. CONCLUSIONES Y LÍNEAS DE TRABAJO FUTURAS

8.1. CONCLUSIONES

La pregunta básica de esta investigación ha sido la siguiente:

> ¿Sería posible construir *aplicaciones de MSOO* para campos específicos que fuesen igual de eficaces y amigables que los programas de simulación para campos específicos desarrollados utilizando lenguajes de programación genéricos?

Los análisis y ejemplos presentados confirman que esa pregunta tiene respuesta afirmativa. Sin embargo, se plantea la interrogante sobre por qué optar por una herramienta de MSOO en lugar de un lenguaje de programación de propósito general. La respuesta radica en que la herramienta de MSOO proporciona los beneficios de utilizar un marco de aplicación genérico para el desarrollo de programas científicos. En particular, se consiguen los siguientes beneficios.

- **Desarrollo más económico**: A pesar de que el desarrollo de aplicaciones con el nuevo paradigma puede ser más costoso que con el paradigma clásico, es considerablemente más breve que el desarrollo de una formulación similar utilizando un lenguaje de programación general. La herramienta de MSOO ofrece una serie de capacidades integradas en su núcleo, sin ningún coste adicional. Estas capacidades abarcan un lenguaje de modelado que facilita la definición jerárquica de modelos, un lenguaje para la definición de experimentos, una interfaz gráfica que permite la representación visual de la topología de los modelos y la entrada de datos de los componentes, así como una interfaz gráfica para el análisis de resultados.
- **Consistencia**: Todas las aplicaciones de simulación desarrolladas con una herramienta MSOO comparten el mismo lenguaje y la misma interfaz gráfica de usuario. Esto acorta la curva de aprendizaje, ya que los usuarios pueden aplicar la experiencia adquirida con una aplicación a cualquier otra aplicación desarrollada con la misma herramienta de MSOO.
- **Transparencia**: El código fuente de las aplicaciones es más fácil de comprender que utilizando un lenguaje de programación general debido a que el lenguaje de modelado de la herramienta de MSOO oculta muchos de los detalles de implementación.
- **Flexibilidad**: A diferencia de las herramientas comerciales que son una caja negra, los usuarios avanzados pueden modificar el código y agregar nuevos componentes sin restricciones.

Una vez realizada esta puntualización, a continuación, se detallan las conclusiones alcanzadas en relación con la aplicación de la nueva aproximación al desarrollo de aplicaciones estacionarias y transitorias.

8.1.1. Conclusiones para la Simulación de Estacionarios

En este trabajo, se ha desarrollado una aproximación totalmente general para la creación con EcosimPro de aplicaciones de simulación estacionaria en campos específicos. La aproximación propuesta ofrece la flexibilidad necesaria para programar cualquier tipo de método de solución, abarcando tanto métodos orientados a ecuaciones como métodos de tipo modular secuencial. Esta versatilidad en la elección de métodos de solución otorga a las aplicaciones resultantes la capacidad de rivalizar en eficacia con herramientas de simulación especializadas.

Es destacable que la misma aproximación utilizada para la creación de aplicaciones de simulación estacionaria puede ser utilizada para el cálculo de soluciones en el dominio de la frecuencia. Por ende, sería posible desarrollar con EcosimPro aplicaciones de simulación que ofrezcan varios tipos de análisis, tales como estacionario, transitorio en el dominio de la frecuencia y transitorio en el dominio del tiempo, de manera similar a los tipos de análisis ofrecidos por Spice. Es importante señalar que la metodología convencional para realizar análisis en el dominio de la frecuencia utilizando herramientas de MSOO implica dos pasos. En primer lugar, se lleva a cabo la linealización del sistema de ecuaciones dentro de la propia herramienta de MSOO. A continuación, es necesario exportar el sistema linealizado a Matlab para efectuar el análisis en el dominio de la frecuencia con esta otra herramienta. Esta metodología tiene la desventaja de requerir el uso de dos herramientas separadas e independientes.

Es posible combinar la nueva aproximación estacionaria con formulaciones transitorias Estas formulaciones transitorias pueden haber sido desarrolladas de la manera convencional, utilizando DAE’s y siguiendo el paradigma del MSOO, o bien mediante la nueva aproximación propuesta para el desarrollo de aplicaciones transitorias.

El principal inconveniente o limitación de la aproximación propuesta para el cálculo de estacionarios es que no permite el cálculo de estacionarios en modelos multidisciplinares que incluyan componentes de otras librerías desarrolladas de la forma convencional.

8.1.2. Conclusiones para la Simulación de Transitorios

En el ámbito del desarrollo de aplicaciones de simulación transitoria, se propone una nueva aproximación basada en el empleo de métodos semidiscretos y en el encapsulamiento en un objeto de las ecuaciones que determinan el estado interno del componente. A nivel de las ecuaciones continuas utilizadas para representar los componentes, estos se simplifican a unas pocas ecuaciones o relaciones algebraicas entre las variables de los puertos

correspondientes. Esta estrategia contribuye significativamente a reducir la complejidad del análisis simbólico dentro de la herramienta de MSOO.

Un aspecto fundamental de la nueva aproximación es el control del incremento de tiempos en la actualización del estado interno de los componentes. En el caso de métodos implícitos, que son estables con cualquier incremento de tiempo, existe el riesgo de imponer incrementos demasiado grandes, lo que podría comprometer la precisión de la solución. En tal situación, sería necesario implementar alguna estrategia de control del incremento de tiempos o, al menos, llevar a cabo estudios de sensibilidad de los casos analizados utilizando diferentes incrementos de tiempo. Sin embargo, en el caso de métodos explícitos, como el método de las características, el paso de integración viene fijado por el método mismo, lo que facilita el control del incremento de tiempo.

8.2. LÍNEAS DE TRABAJO FUTURAS

Los ejemplos de librerías presentadas en los capítulos 6 y 7 de esta tesis, una para estacionarios hidráulicos y otra para transitorios hidráulicos, están destinados a ilustrar el nuevo paradigma. Estos ejemplos son de índole académica y su principal finalidad es explicar la nueva aproximación y las técnicas de programación propuestas. Sin embargo, es importante señalar que estas aplicaciones no han sido diseñadas para satisfacer las exigencias de uso comercial, práctico o en situaciones del mundo real. Presentan limitaciones, como un conjunto de componentes bastante reducido y ciertas simplificaciones en la formulación para facilitar su comprensión y enseñanza. Las terminaciones ***ST*** en el nombre de ambas librerías (***HYDRAULIC_ST*** y ***LIQHAMMER_ST***) indican claramente que se trata de versiones académicas o para estudiantes.

No obstante, estas librerías cuentan con una estructura clara y ordenada, lo que las convierte en un valioso punto de partida para el desarrollo de aplicaciones más avanzadas y profesionales en el futuro.

8.2.1. Aumento de Capacidades y Mejora de las Librerías Desarrolladas

Como líneas de trabajo futuras, se plantea la necesidad de ampliar el conjunto de componentes ofrecido por ambas librerías, así como de incluir formulaciones más sofisticadas para algunos efectos, con el objetivo de que permitan abordar problemas de mayor complejidad y relevancia industrial. Además, es necesaria la realización de planes de validación y verificación más exhaustivos para garantizar la fiabilidad y precisión de las modelos desarrolladas con estas aplicaciones.

En el caso de la aplicación de estacionarios hidráulicos se plantean las siguientes mejoras:

- Inclusión de nuevos componentes para representar bombas, colectores, conexiones en T, y equipos como filtros

- Introducción de la ecuación de la energía para permitir el cálculo de temperaturas
- Inclusión de componentes para representar cambiadores de calor y calentadores.
- Introducción de válvulas de control, que requieren el uso de las técnicas para la solución de problemas de complementariedad descritas en el Apéndice B. Las válvulas de control tienen como objetivo mantener una variable en un valor de referencia específico, sin exceder el cierre total ni la apertura máxima (problema de complementariedad mixta).
- Introducción de una base de datos con las propiedades de los líquidos más comunes: agua, agua de mar, gasoil, sal solar, entre otros.

En el caso de la aplicación de estacionarios hidráulicos se plantean las siguientes mejoras:

- Inclusión de nuevos componentes para representar bombas, válvulas de aire, chimeneas de equilibrio, y acumuladores.
- Introducción de una base de datos con las propiedades de los líquidos más comunes: agua, agua de mar, gasoil, sal solar, entre otros.
- Introducción del cálculo del estacionario previamente al cálculo del transitorio.
- Inclusión del método de diferencias finitas implícitas de 4 puntos cuando el incremento de tiempo exceda el Courant con cambio automático entre el método de las características y el método de diferencias finitas implícitas dependiendo del incremento de tiempo requerido.

8.2.2. Desarrollo de Nuevas Librerías

De manera similar a como se ha desarrollado un par de aplicaciones para la simulación de flujo de líquidos en redes de tuberías: una para simulación estacionaria y otra para simulación transitoria, se propone el desarrollo de un par de aplicaciones para la simulación del flujo de gases en redes de tuberías: una estacionaria (***GASSTEADY***) y otra transitoria (***GASHAMMER***). Estas aplicaciones tendrían el objetivo de proporcionar herramientas robustas y eficientes para la modelización y análisis de sistemas de tuberías que transportan gases, tanto en condiciones estacionarias como en situaciones transitorias.

Otro posible desarrollo sería una librería dedicada a la simulación del flujo transitorio en canales abiertos.

Además, se identifica una clara necesidad de desarrollar una librería para la realización de balances de masa y energía en ciclos de producción de potencia, particularmente en ciclos de vapor. Las aplicaciones actuales para estos fines, desarrolladas con herramientas de Modelado y Simulación Orientados a Objetos (MSOO), se enfrentan a dificultades de uso, principalmente debido a la técnica de rasgadura utilizada en las herramientas de MSOO.

Por otro lado, las librerías existentes pueden aprovechar la modelización transitoria de tuberías presentada en el capítulo 7. Por ejemplo, la simulación de plantas hidroeléctricas y, en general, la simulación de cualquier sistema que involucre tuberías para el transporte de líquidos podrían beneficiarse de los modelos desarrollados en dicho capítulo.

.

APÉNDICE A EJEMPLOS DE MODOS DE FALLO DE LA RASGADURA (TEARING) DE ECUACIONES

A continuación, se presentan tres ejemplos de sistemas de ecuaciones algebraicas tomados de (Baharev et al., 2017), en los cuales los algoritmos de rasgadura presentan fallas por diversas razones. Además, se analiza el comportamiento de EcosimPro en relación a estos ejemplos y se realiza una comparación con otras herramientas de MSOO. Los ejemplos considerados son los siguientes:

- Ejemplo 1: Residuo muy sensible con respecto a la variable de rasgadura.
- Ejemplo 2: Residuo muy poco sensible respecto a la variable de rasgadura.
- Ejemplo 3: Incapacidad de la técnica de rasgadura para reducir el tamaño del sistema.

A.1. Ejemplo 1: Residuo Muy Sensible Respecto Variable Rasgadura

A.1.1. Definición del Ejemplo 1

El sistema de ecuaciones del primer ejemplo es el siguiente:

$$x_{i-1} + k\,x_i + x_{i+1} = (k+2)\cdot 0.1 \qquad i = 1{:}20 \tag{A-1}$$

donde $x_o = 0.1$ y $x_{21} = 0.1$ y cuya solución es $x_i = 0.1$ para $i = 1{:}20$.

Al aplicar manualmente la técnica de rasgadura, se determina que la variable de rasgadura más adecuada, desde un punto de vista simbólico, es x_1. El conocimiento de x_1 permite obtener de forma explícita las 19 variables restantes usando las 19 primeras ecuaciones:

$$x_2 = (k+2)\cdot 0.1 - k\,x_1 - x_o \tag{A-2}$$

$$x_3 = (k+2)\cdot 0.1 - k\,x_2 - x_1 \tag{A-3}$$

…

$$x_{20} = (k+2)\cdot 0.1 - k\,x_{19} - x_{18} \tag{A-4}$$

Finalmente, la ecuación (A-1) para $i = 20$ proporciona el residuo que, en teoría, permite calcular el valor de la variable de rasgadura:

$$residuo(x_1) = (k+2)\cdot 0.1 - x_{21} - k\,x_{20} - x_{19} \tag{A-5}$$

En este ejemplo, la técnica de rasgadura falla debido a que el residuo de la ecuación de rasgadura es muy sensible con respecto a variable de rasgadura. Es fácil comprobar por substituciones sucesivas de las variables despejadas que el residuo es proporcional a $(-k)^{20}x_1$, y, por tanto, es muy sensible respecto x_1 cuando el parámetro $k > 1$.

A.1.2. Implementación y Comportamiento en EcosimPro del Ejemplo 1

El Listado A-1 muestra la codificación del modelo del Ejemplo 1 en EcosimPro. La partición de defecto de dicho modelo es capaz de encontrar la rasgadura óptima del mismo, que se ha descrito anteriormente.

Listado A-1: Modelo EcosimPro del Ejemplo 1 de Problemas en la Rasgadura

```
FUNCTION REAL f(IN REAL x)
   BODY
      RETURN x
END FUNCTION
COMPONENT BaharevExample1(INTEGER N = 20)
   DATA
      REAL x_o = 0.1
      REAL x_N1 = 0.1
   DECLS
      BOUND REAL k = 10.
      REAL x[N]
   CONTINUOUS
      f(x_o) + k * f(x[1]) + x[2] =  (k+2)* 0.1

      EXPAND (i IN 2, N-1)
         f(x[i-1]) + k * f(x[i]) + x[i+1] = (k+2) * 0.1

      f(x[N-1]) + k * f(x[N]) + x_N1 = (k+2)* 0.1
END COMPONENT
```

Por último, al ejecutar un experimento consistente en el cálculo de un estacionario fijando el valor de la condición de contorno `k` igual a `10`, la solución calculada por EcosimPro para los últimos elementos del vector `x` es totalmente errónea (por ejemplo, `x[20]=-118.18`), y se obtiene un mensaje de error indicando que no ha habido convergencia. Según Baharev et al. (2017) el comportamiento de dos de los entornos de simulación de Modelica con este problema es el siguiente: Dymola y OpenModelica calculan respectivamente `x[20]=32.03` y `x[20]=85.82`. Cabe destacar que ninguno de estos entornos proporciona un mensaje de error en este caso.

A.2. Ejemplo 2: Residuo Insensible Respecto de la Variable de Rasgadura

A.2.1. Definición del Ejemplo 2

El ejemplo 2 consiste en el siguiente sistema de ecuaciones:

$$x_{i-1} + x_i + k\,x_{i+1} = k+2 \qquad i = 1{:}20 \tag{A-6}$$

donde $x_o = 1$ y $x_{21} = 1$. Es fácil comprobar que la solución de este sistema es $x_i = 1\ (i = 1{:}20)$ y que también en este caso, la variable de rasgadura más adecuada desde un punto de vista simbólico es x_1, ya que permite obtener las restantes variables de forma explícita:

$$x_2 = (k+2 - x_1 - x_o)/k \tag{A-7}$$

$$x_3 = (k+2 - x_2 - x_1)/k \tag{A-8}$$

…

$$x_{20} = (k+2 - x_{19} - x_{18})/k \tag{A-9}$$

y utilizar la ecuación para $i = 20$ como ecuación de rasgadura:

$$residuo(x_1) = (k+2-x_{21} - x_{20})/k \tag{A-10}$$

Es fácil comprobar por substituciones sucesivas de las variables despejadas, que el residuo es proporcional a $(-k)^{-20}x_1$, por tanto el residuo puede ser muy poco sensible en el caso de que el parámetro $k > 1$.

A.2.2. Implementación y Comportamiento en EcosimPro del Ejemplo 2

El Listado A-2 muestra la codificación en EcosimPro del Ejemplo 2.

Listado A-2: Modelo EcosimPro del Ejemplo 2 de Problemas en la Rasgadura

```
COMPONENT BaharevExample2(INTEGER N = 20)
   DATA
      REAL x_o = 1
      REAL x_N1 = 1
   DECLS
      BOUND REAL k = 1
      REAL x[N]
   CONTINUOUS
      f(x_o) + f(x[1]) + k*x[2] = k+2

      EXPAND (i IN 2, N-1)
            f(x[i-1]) + f(x[i]) + k*x[i+1] = k+2

      f(x[N-1]) + f(x[N]) + k*x_N1 = k+2
END COMPONENT
```

De nuevo con la partición de defecto, el algoritmo de rasgadura de EcosimPro es capaz de encontrar la rasgadura óptima. Seguidamente, la ejecución de un experimento para

calcular un estacionario proporciona una solución errónea, `x[1]=0.05`, cuando la solución correcta es `x[1]=1`. A pesar de que la solución es errónea, el programa indica que ha conseguido convergencia.

En cuanto a los entornos de simulación de Modelica analizados por Baharev et al. (2017), ninguno de ellos calcula la solución correcta. Dymola arroja un mensaje de error que ofrece poca ayuda, mientras que OpenModelica proporciona avisos intermedios (*warnings*) y reporta al final que la simulación ha terminado correctamente, aunque el resultado obtenido es incorrecto, ya que el valor de `x[1]` es igual al valor supuesto inicialmente.

A.3. Ejemplo 3: La Rasgadura es Incapaz de Reducir el Tamaño del Sistema

A.3.1. Definición del Ejemplo 3

El ejemplo 3 es el sistema de ecuaciones de tamaño $N \times N$ siguiente:

$$x_i + x_N = 2 \qquad i = 1{:}N-1 \tag{A-11}$$

$$\sum_{i=1}^{N} x_i = N \tag{A-12}$$

donde se supone que la única variable que se puede despejar de forma explícita es x_N en la ecuación (A-12). La dificultad de este sistema radica en que, aunque es disperso, no es posible seleccionar un conjunto reducido de variables de rasgadura. La única solución posible para la rasgadura es considerar como variables de rasgadura las $n\text{-}1$ primeras variables y despejar x_N en la última ecuación. En este caso la rasgadura solo permite reducir el tamaño del sistema de ecuaciones de N a $N-1$.

A.3.2. Implementación y Comportamiento en EcosimPro del Ejemplo 3

El Listado A-3 muestra la codificación en EcosimPro del Ejemplo 3. Con `N=300`, la partición por defecto de EcosimPro tarda menos de 2 segundos y no reduce el tamaño del Sistema de Ecuaciones, esto es, considera las 300 ecuaciones como implícitas. Seguidamente, la ejecución de un experimento para calcular un estacionario tarda menos de 17 s.

Según Baharev et al. (2017), en el entorno Dymola se requieren 74 segundos para realizar la rasgadura con `N=300`, mientras que en OpenModelica se requieren 37 segundos. Es importante tener en cuenta que estos tiempos deben interpretarse con precaución debido a la utilización de diferentes ordenadores y compiladores. No obstante, estos resultados sugieren que los dos entornos de Modelica considerados dedican un tiempo excesivo en intentar encontrar una rasgadura optima en un caso en el que la rasgadura no proporciona beneficios apreciables.

Listado A-3: Modelo EcosimPro del Ejemplo 3 de Problemas en la Rasgadura

```
COMPONENT BaharevExample3(INTEGER N =  300)
   DECLS
      REAL x[N]
   CONTINUOUS
      EXPAND (i IN 1, N-1)
            f(x[i]) + f(x[N]) = 2

      SUM(i IN 1, N-1; f(x[i])) + x[N] = N

END COMPONENT
```

A.4. Conclusiones de los Ejemplos de Fallo de la Rasgadura

De los tres ejemplos de problemas en la rasgadura que se presentan, dos de ellos, el ejemplo uno y el ejemplo dos, ponen de manifiesto la falta de robustez numérica de la rasgadura. Estos ejemplos ilustran situaciones que, aunque pueden parecer artificiales, pueden presentarse en modelos de interés práctico.

Por ejemplo, consideremos un modelo hidráulico como el mostrado en la Figura A-1. En este modelo, se tiene una línea con varias válvulas muy estranguladas en serie. Al realizar la rasgadura, la herramienta puede asumir como conocidos el flujo y la presión de entrada a la línea, e ir calculando de forma sucesiva las presiones a la salida de cada válvula.

Sin embargo, debido a la posición muy estrangulada de las válvulas, las caídas de presión serán sumamente sensibles al flujo asumido, acumulándose a lo largo de las diversas válvulas en serie. Esta sensibilidad, puede generar inestabilidades numéricas, comprometiendo la precisión y confiabilidad de los resultados obtenidos.

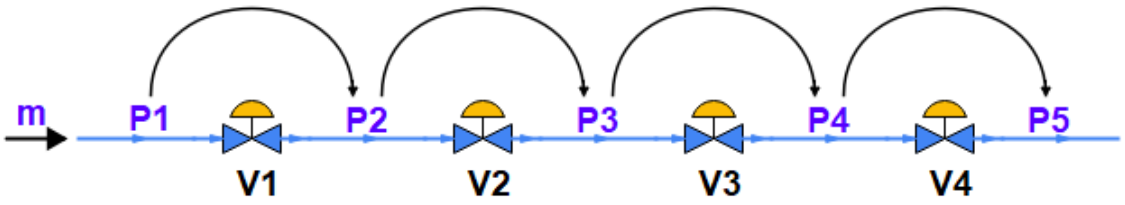


Figura A-1: Ejemplo de rasgadura en una serie de válvulas

En cuanto al ejemplo tres, se trata de un sistema de ecuaciones disperso, pero para el que no es posible encontrar un subconjunto reducido de variables de rasgadura.

Es importante notar que si en lugar de aplicar una técnica dispersa de tipo simbólico, como es la rasgadura, se aplica una técnica dispersa de tipo numérico, esto es, un programa para la solución de sistemas de ecuaciones dispersas, la solución se calcula correctamente y en un tiempo muy breve para los tres casos ejemplo..

APÉNDICE B VISIÓN GENERAL DE PROBLEMAS DE COMPLEMENTARIEDAD

B.1. Introducción

B.2. Continuación

En este apéndice se proporciona una visión general de los problemas de complementariedad y de la solución de los mismos mediante métodos suavizados de Newton. Estos problemas revisten gran interés debido a que, en muchas situaciones prácticas, es viable expresar las ecuaciones condicionales como problemas de complementariedad. Estos, a su vez, pueden ser resueltos de manera eficaz mediante la aplicación de métodos suavizados de Newton.

B.3. Definición de los Problemas de Complementariedad

Los problemas de complementariedad son sistemas de restricciones no lineales donde las variables del sistema están ligadas por restricciones en forma de condiciones de complementariedad. Existen tres problemas estándar de complementariedad: el problema de complementariedad no lineal (Yong, 2010), el problema generalizado de complementariedad (M. Li et al., 2012) y el problema de complementariedad mixta (D. Li & Fukushima, 2000).

B.3.1. Problema de Complementariedad No Lineal

Dada una función continuamente diferenciable $\boldsymbol{F}(\boldsymbol{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}^n$, $\boldsymbol{F} = (f_1, f_2, \dots, f_n)$, el problema de complementariedad no lineal (NCP, nonlinear complementarity problem) consiste en encontrar un punto $\boldsymbol{x} \in \mathbb{R}^n_+$ que satisfaga el siguiente sistema de ecuaciones y desigualdades:

$$\boldsymbol{x} \geq 0, \qquad \boldsymbol{F}(\boldsymbol{x}) \geq 0, \qquad x^T \cdot \boldsymbol{F}(\boldsymbol{x}) = x_1 f_1 + x_2 f_2 + \cdots + x_n f_n = 0 \tag{B-1}$$

B.3.2. Problema Generalizado de Complementariedad No Lineal

Dadas dos funciones continuamente diferenciables $\boldsymbol{G}(\boldsymbol{x})$ y $\boldsymbol{F}(\boldsymbol{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}^n$, el problema generalizado de complementariedad no lineal (GNCP, generalized nonlinear complementarity problem) consiste en encontrar un punto $\boldsymbol{x} \in \mathbb{R}^n$ que satisfaga el siguiente sistema de ecuaciones y desigualdades:

$$\boldsymbol{G}(\boldsymbol{x}) \geq 0, \qquad \boldsymbol{F}(\boldsymbol{x}) \geq 0, \qquad \boldsymbol{G}(\boldsymbol{x})^T \cdot \boldsymbol{F}(\boldsymbol{x}) = g_1 f_1 + g_2 f_2 + \cdots + g_n f_n = 0 \tag{B-2}$$

El problema generalizado de complementariedad no lineal (GNCP) incluye como caso especial el problema de complementariedad no lineal, el cual corresponde al caso en que $\boldsymbol{G}(\boldsymbol{x}) = \boldsymbol{x}$.

B.3.3. Problema de Complementariedad Mixta

Dada una función continuamente diferenciable asociada $\boldsymbol{F}(\boldsymbol{x})$: $\mathbb{R}^n \rightarrow \mathbb{R}^n$, $\boldsymbol{F} = (f_1, f_2, \dots, f_n)$, y unos límites inferior y superior

$$\boldsymbol{l} \in \{\mathbb{R} \cup \{-\infty\}\}^n$$

$$\boldsymbol{u} \in \{\mathbb{R} \cup \{+\infty\}\}^n$$

El problema de complementariedad mixta (MCP, mixed complementarity problem) consiste en encontrar un punto o vector $\boldsymbol{x} \in \mathbb{R}^n$ tal que para cada índice $i \in \{1, \dots, n\}$ se cumpla una de la siguientes alternativas:

$$\begin{aligned} x_i &= l_i, && f_i(\underline{x}) \geq 0 \\ l_i &< x_i < u_i, && f_i(\underline{x}) = 0 \\ x_i &= u_i, && f_i(\underline{x}) \leq 0 \end{aligned} \qquad \text{(B-3)}$$

La condición de complementariedad mixta también se puede expresar con la siguiente ecuación:

$$\text{mid}\{x_i - l_i, f_i(\underline{x}),\ x_i - u_i\} = 0\ , \qquad \text{(B-4)}$$

Donde el operador $\text{mid}\{a, b, c\}$ representa la mediana de 3 escalares $a, b, c\ \mathbb{R} \cup \{\pm\infty\}$

El problema de complementariedad mixta incluye como caso especial el problema de complementariedad no lineal (NCP), el cual corresponde al caso en que $l_i = 0$ y $u_i = +\infty$ para todo índice $i \in \{1, \dots, n\}$.

B.4. Métodos de Newton Suavizados

Es factible reformular los problemas convencionales de complementariedad como sistemas de ecuaciones no lineales, con el propósito de aplicar métodos de solución de sistemas ecuaciones, como el método de Newton. La transformación de un problema de complementariedad en un sistema de ecuaciones se lleva a cabo mediante el uso de funciones denominadas NCP para el caso el problema de complementariedad no lineal y el problema generalizado, así como funciones denominadas MCP para el caso del problema de complementariedad mixta.

Definición función NCP: Una función ϕ: $\mathbb{R}^2 \rightarrow \mathbb{R}$ es llamada una función NCP si satisface:

$$\phi(a,b) = 0 \iff a \geq 0, b \geq 0, ab = 0. \quad \text{(B-5)}$$

Las siguientes funciones son ejemplos de funciones NCP obtenidas en las referencias (Chen & Mangasarian, 1996; Fischer, 1992; M. Li et al., 2012; Yong, 2010) :

$$\phi(a,b) = min(a,b) \quad \text{(B-6)}$$

$$\phi(a,b) = (a+b) - \sqrt{a^2+b^2} \quad \text{(B-7)}$$

$$\phi(a,b) = (a+b) - \sqrt{\theta(a-b)^2 + (1-\theta)(a^2+b^2)} - \textbf{ donde } \theta \in [0,1] \quad \text{(B-8)}$$

$$\phi(a,b) = (a+b) - \sqrt[p]{|a|^p + |b|^p} \textbf{ donde } p \in \mathbb{R} \mid 1 < p < \infty \quad \text{(B-9)}$$

$$\phi(a,b) = (a+b) - \sqrt{a^2+b^2} + \alpha a_+ b_+ \textbf{ donde } \alpha \in \mathbb{R} \textbf{ y } t_+ = \max\{0,t\} \quad \text{(B-10)}$$

$$\phi(a,b) = a\,b - \tfrac{1}{2}[\min(0, a+b)]^2 \quad \text{(B-11)}$$

$$\phi(a,b) = a - \max(a-b, 0) \quad \text{(B-12)}$$

La función (B-7) se denota como función Fischer-Burmeister y se designa con el símbolo ϕ^{FB} y la función (B-10) se denota como función Fischer-Burmeister penalizada.

Todas estas funciones NCP se han definido de manera que están orientadas positivamente, lo cual significa que para todo $a, b \in \mathbb{R}$ se cumple que:

$$\text{sign}\left(\phi(a,b)\right) = \text{sign}\big(\min(a,b)\big) \quad \text{(B-13)}$$

Mediante el empleo de cualquier función NCP ϕ, es factible transformar el problema de complementariedad en un sistema de ecuaciones. Esto se logra definiendo una función $\Phi: \mathbb{R}^n \to \mathbb{R}^n$ de la siguiente manera:

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \Big(\phi\big(x_1\,,\, f_1(\boldsymbol{x})\big), \ldots, \phi\big(x_n\,,\, f_n(\boldsymbol{x})\big)\Big)^T \quad \text{(B-14)}$$

Si la función $\boldsymbol{F}$ y la función NCP son continuamente diferenciables, la función $\boldsymbol{\Phi}$ también es continuamente diferenciable. En este caso, es posible aplicar el método de Newton clásico a la ecuación $\boldsymbol{\Phi}(\boldsymbol{x}) = \boldsymbol{0}$ con el fin de resolver el problema NCP. Esto conduce a la siguiente iteración:

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \left[\boldsymbol{\Phi}'\big(\boldsymbol{x}^k\big)\right]^{-1} \boldsymbol{\Phi}\big(\boldsymbol{x}^k\big),\ k = 0, 1, 2, \ldots \quad \text{(B-15)}$$

dado un punto inicial $\boldsymbol{x}^0 \in \mathbb{R}^n$. Es esperable una convergencia local rápida si se cumplen las suposiciones estándar del método de Newton; en particular, la matriz Jacobiana $\boldsymbol{\Phi}'(\boldsymbol{x}^*)$ tiene que ser no singular en el punto de solución $\boldsymbol{x}^*$ del problema NCP. Lamentablemente, se demuestra que la matriz $\boldsymbol{\Phi}'$ es singular para cualquier solución degenerada ($x_i = 0$ y

$f_i(\boldsymbol{x})$= 0). Una técnica para resolver el problema de singularidad consiste en emplear una función NCP suavizada.

Los métodos suavizados para problemas NCP aproximan la función NCP ϕ no suavizada con una función suavizada ϕ_ε que implementa un parámetro de suavización $\varepsilon > 0$. A continuación se proporcionan algunos ejemplos de funciones NCP suavizadas.

$$\phi_\varepsilon(a,b) = (a+b) - \sqrt{a^2+b^2+\ \varepsilon^2} \tag{B-16}$$

$$\phi_\varepsilon(a,b) = (a+b) - \sqrt{\theta(a-b)^2 + (1-\theta)(a^2+b^2) + 4\varepsilon^2}\ \ \textbf{donde}\ \theta \in [0,1] \tag{B-17}$$

$$\phi_\varepsilon(a,b) = (a+b) - \sqrt[p]{|a|^p + |b|^p + \varepsilon^p}\ \ \textbf{donde}\ \ p \in\ \mathbb{R} \mid 1 < p < \infty \tag{B-18}$$

$$\phi_\varepsilon\,(a,b) = (a+b) - \sqrt{a^2+b^2+\ \varepsilon^2} + \frac{\alpha}{4}\left(a + \sqrt{a^2+2\varepsilon^2}\,\right)\left(b + \sqrt{b^2+2\varepsilon^2}\,\right) \tag{B-19}$$

Nota: la función (B-19) es la versión suavizada de la función Fischer-Burmeister penalizada (B-10).

Inicialmente, en las primeras iteraciones del método de Newton se utiliza un parámetro de suavización alto. Luego, se disminuye a valores casi nulos conforme se consigue convergencia. Un punto crítico de estos métodos es como actualizar el valor ε en cada iteración.

Definición Funciones MCP: Denotando $\overline{\mathbb{R}}^3 = (\mathbb{R} \cup \{-\infty\}) \times \mathbb{R} \times (\mathrm{R} \cup \{+\infty\})$; una función ψ: $\overline{\mathbb{R}}^3 \to \mathbb{R}$ es una función MCP si cumple:

$$\psi(a,b,c) = 0\ , a > c\ \ \Leftrightarrow\ mid(a,b,c) = 0, a > c \tag{B-20}$$

Las siguientes funciones son ejemplos de funciones MCP obtenidas en las referencias (Billups, 2002; D. Li & Fukushima, 2000):

$$\psi(a,b,c) = a + c - \sqrt{(a-b)^2}\ + \sqrt{(b-c)^2}\,, \qquad a > c \tag{B-21}$$

$$\psi(a,b,c) = \phi^{FB}\left(a, -\phi^{FB}(-b,-c)\right) \tag{B-22}$$

Utilizando cualquier función MCP ψ , es factible definir la siguiente función $\boldsymbol{\Phi}(\boldsymbol{x}) = \left(\psi_1(\boldsymbol{x}), \psi_2(\boldsymbol{x}), \dots, \psi_n(\boldsymbol{x})\right)^T$, donde para cada $i = 1,2,\dots,n$

$$\psi_i(\boldsymbol{x}) = \psi(\ x_i - l_i,\ F_i(\boldsymbol{x})\ , x_i - u_i) \tag{B-23}$$

El problema MCP se reformula entonces como el siguiente sistema de ecuaciones no suavizadas:

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \mathbf{0} \tag{B-24}$$

Al igual que con las funciones NCP, es conveniente el uso de funciones suavizadas para evitar Jacobianos singulares.

$$\psi_\epsilon(a,b,c) = a + c - \sqrt{(a-b)^2 + \varepsilon^2} \; + \sqrt{(b-c)^2 + \varepsilon^2} \tag{B-25}$$

$$\psi_\epsilon(a,b,c) = \phi_\varepsilon^{FB}\left(a, -\phi_\varepsilon^{FB}(-b,-c)\right) \tag{B-26}$$

APÉNDICE C DERIVACIÓN DE UNA LIBRERIA DE ECOSIMPRO PARA SIMULACIÓN DE TRANSITORIOS HIDRÁULICOS CON EL MÉTODO SEMIDISCRETO

C.1. Introducción

En este apéndice se presenta el desarrollo de TRANLIQ, una librería de EcosimPro, diseñada para la simulación de transitorios hidráulicos. Aunque existen diversas librerías comerciales de EcosimPro para abordar este tipo de simulaciones, como FluidaPro y PipeLiqTran, es importante destacar que estas librerías son de naturaleza comercial y no proporcionan acceso a su código fuente.

El desarrollo de la librería presentada en este apéndice persigue un doble propósito. En primer lugar, se busca ilustrar el proceso de desarrollo de una librería hidráulica, lo que permite exponer los desafíos típicos que enfrenta un desarrollador durante este proceso. En segundo lugar, esta librería se utiliza para resolver los casos o problemas de prueba que se han utilizado para llevar a cabo las comparaciones entre los métodos empleados por las herramientas específicas de transitorios hidráulicos y los métodos semidiscretos utilizados en el MSOO.

C.2. Formulación de los Componentes

C.2.1. Formulación Componente ***Pipe***

Ecuaciones en Derivadas Parciales

El componente tubería es el elemento fundamental de esta librería. La descripción del comportamiento de este componente se basa en las dos ecuaciones en derivadas parciales siguientes:

$$\frac{a^2}{g\,A}\frac{\partial Q}{\partial x}+\frac{\partial H}{\partial t}=0 \qquad (-1)$$

$$\frac{1}{g\,A}\frac{\partial Q}{\partial t}+\frac{\partial H}{\partial x}+\frac{f}{2g\,D}\frac{Q|Q|}{A^2}=0 \qquad (C\text{-}2)$$

La ecuación (-1) describe la conservación de la masa y la ecuación (C-2) es la ecuación del momento. Es importante destacar que, con el propósito de simplificar estas ecuaciones, no se ha considerado ningún término que represente la fricción transitoria en la ecuación del momento.

Discretización Espacial

Para aplicar un método semidiscreto, se requiere realizar una discretización de la coordenada espacial, convirtiendo así las ecuaciones en derivadas parciales en ecuaciones diferenciales ordinarias en puntos discretos de una malla. Para llevar a cabo esta discretización, se ha optado por el método de discretización centrada con mallas desplazadas.

En dicho tipo de discretización, la tubería se divide en n volúmenes, como se muestra en la Figura A-1. En estos volúmenes, se aplica el principio de conservación de la masa para calcular la altura piezométrica. A continuación, se introduce una segunda malla desplazada en medio volumen con respecto a la primera, tal como se ilustra en la Figura C-2. Esta segunda malla se emplea para aplicar la ecuación del momento y calcular los flujos que interconectan los volúmenes de la primera malla.

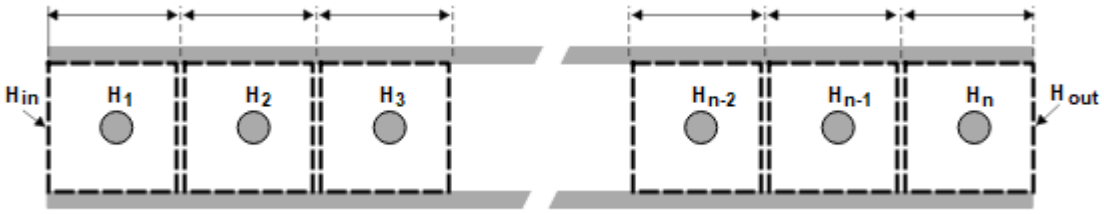


Figura C-1: Volúmenes de Control para la Conservación de la Masa

Figura C-2: Volúmenes de Control para la Conservación del Momento

Aproximando las derivadas espaciales en la ecuación de conservación de masa mediante expresiones en diferencias finitas para todos los volúmenes de control de la primera malla, se obtiene:

$$\frac{a^2}{g\,A}\frac{\left(Q_{j+1}-Q_j\right)}{\Delta x}+\frac{dH_j}{dt}=0 \qquad \forall j \in 1,2,\dots,n \qquad \text{(C-1)}$$

Aproximando las derivadas espaciales en la ecuación del momento mediante expresiones en diferencias finitas para todos los volúmenes de control interiores de la segunda malla, se obtiene:

$$\frac{1}{g\,A}\frac{dQ_j}{dt}+\frac{\left(H_{j+1}-H_j\right)}{\Delta x}+\frac{f}{2g\,D}\frac{Q_j|Q_j|}{A^2}=0 \qquad \forall j\in 2,\dots,n \tag{C-2}$$

Asimismo, aplicando la ecuación del momento a los volúmenes de control en los dos extremos, que tienen la mitad de longitud, se obtiene las dos ecuaciones siguientes:

$$\frac{1}{g\,A}\frac{dQ_1}{dt}+\frac{(H_1-H_{in})}{\Delta x/2}+\frac{f}{2g\,D}\frac{Q_1|Q_1|}{A^2}=0 \tag{C-3}$$

$$\frac{1}{g\,A}\frac{dQ_{n+1}}{dt}+\frac{(H_{out}-H_n)}{\Delta x/2}+\frac{f}{2g\,D}\frac{Q_{n+1}|Q_{n+1}|}{A^2}=0 \tag{C-4}$$

En estas dos últimas ecuaciones, los sufijos "*in*" y "*out*" se utilizan para denotar el valor de las variables (altura piezométrica y flujo) en las secciones de entrada y salida de la tubería. Se asume que los flujos en las secciones de entrada y salida son iguales respectivamente a los flujos en el primer y último volumen control del momento:

$$Q_{in}=Q_1 \tag{C-5}$$

$$Q_{out}=Q_{n+1} \tag{C-6}$$

Una dificultad inherente en las ecuaciones del momento en los volúmenes de control de los extremos, es decir, las ecuaciones (C-3) y (C-4), radica en que requieren que los flujos volumétricos de entrada y salida son variables de estado. Esta aproximación puede ser adecuada cuando el componente conectado a la tubería es un tanque o depósito que fija la presión en las secciones de entrada o salida. Sin embargo, se vuelve inapropiada cuando se trata de elementos que controlan el flujo, como las válvulas.

Para superar esta dificultad y lograr una causalidad adecuada en los extremos, se recurre a una técnica consistente en despreciar la diferencia de presión en los volúmenes de control del momento en los extremos. Con esta técnica, las ecuación (C-3) o la ecuación (C-4) o ambas, son sustituidas por las siguientes ecuaciones respectivas:

$$H_{in}=H_1 \tag{C-7}$$

$$H_{out}=H_n \tag{C-8}$$

Existen cuatro combinaciones posibles para la causalidad del componente pipe, que se denominarán de la siguiente manera:

- **Resistiva-Resistiva (RR)**, que considera las ecuaciones (C-3) y (C-4),
- **Resistiva-Capacitiva (RC)**, que incluye las ecuaciones (C-3) y (C-8),

- **Capacitiva-Resistiva (CR)**, que considera las ecuaciones(C-4) y (C-7), y
- **Capacitiva-Capacitiva (CC)**, que contempla las ecuaciones (C-7) y (C-8).

Un inconveniente de las aproximaciones **CR**, **RC** y **CC** es que omiten la aplicación de la ecuación del momento en uno o en los dos volúmenes de control ubicados en los extremos de la segunda malla. La magnitud del error que se comete es inversamente proporcional al número de nodos en los que se divide la tubería. Si el número de nodos es considerable, el error es despreciable; no obstante, si el número de nodos es menor de una decena, la magnitud del error resulta significativa. Para corregir parcialmente estos errores, se puede introducir un multiplicador en los términos de fricción y de inercia en la ecuación del momento, de manera que se compense la inercia del flujo y la fricción no considerad en los extremos.

En el caso de las causalidades **RC** o **CR**, no se está considerando la diferencia de presión en un volumen de control con un tamaño igual a la mitad de uno de los volúmenes de control interiores, y en el caso de la causalidad **CC**, no se está considerando la diferencia de presión en dos volúmenes de control con un tamaño total igual al de uno de los volúmenes de control interiores. Por lo tanto, el coeficiente multiplicador para compensar el tramo de tubería no considerado, viene dado por:

$$km = 1 \qquad \textbf{para la causalidad RR} \tag{C-9}$$

$$km = \frac{n}{n - 1/2} \qquad \textbf{para las causalidades CR y RC} \tag{C-10}$$

$$km = \frac{n}{n - 1} \qquad \textbf{para la causalidad CC} \tag{C-11}$$

Las ecuaciones del momento, teniendo en cuenta este coeficiente corrector, quedan de la siguiente manera:

$$km\frac{1}{g\,A}\frac{dQ_j}{dt} + \frac{\left(H_{j+1} - H_j\right)}{\Delta x} + km\frac{f}{2g\,D}\frac{Q_j|Q_j|}{A^2} = 0 \qquad \forall j \in 2, \dots, n \tag{C-12}$$

$$km\frac{1}{g\,A}\frac{dQ_1}{dt} + \frac{(H_1 - H_{in})}{\Delta x/2} + km\frac{f}{2g\,D}\frac{Q_1|Q_1|}{A^2} = 0 \tag{C-13}$$

$$km\frac{1}{g\,A}\frac{dQ_{n+1}}{dt} + \frac{(H_{out} - H_n)}{\Delta x/2} + km\frac{f}{2g\,D}\frac{Q_{n+1}|Q_{n+1}|}{A^2} = 0 \tag{C-14}$$

Sin embargo, es relevante destacar que, a pesar de la inclusión del coeficiente multiplicador, persiste un error, ya que se altera la distribución de capacitancias e inductancias. Esto se manifiesta en pequeñas discrepancias entre el incremento de altura

calculado con esta formulación después de una detención instantánea del flujo y el incremento teórico calculado con la ecuación de Joukowsky.

Viscosidad Artificial

La formulación que se ha presentado para la tubería, a pesar de ser de primer orden, e introducir un amortiguamiento numérico significativo, conlleva a la aparición de oscilaciones sin una correspondencia en la realidad física después del paso de una onda de presión abrupta, como se evidencia en el segundo caso de prueba del capítulo 4. El remedio para evitar dichas oscilaciones es la introducción de una viscosidad artificial numérica.

La viscosidad artificial se introduce como una altura adicional, que se añade a la altura piezométrica en la ecuación del momento, y que viene dada por la siguiente formula debida a Landshoff (1955):

$$W = k_{damp} \frac{a}{gA} \frac{\partial Q}{\partial x} \tag{C-1}$$

Donde k_{damp} es un coeficiente que sirve para controlar la cantidad de viscosidad artificial introducida.

El valor de la altura adicional W se calcula por diferencias finitas en todos los puntos de la primera malla:

$$W_j = k_{damp} \frac{a}{g\,A} \frac{(Q_{j+1} - Q_j)}{\Delta x} \qquad \forall j \in 1,2,\dots,n \tag{C-2}$$

Introduciendo esta altura adicional en las ecuaciones del momento, quedan las siguientes ecuaciones finalmente implementadas en el componente.

$$km \frac{1}{g\,A} \frac{dQ_j}{dt} + \frac{(H_{j+1} + W_{j+1} - H_j - W_j)}{\Delta x} + km \frac{f}{2g\,D} \frac{Q_j|Q_j|}{A^2} = 0 \qquad \forall j \in 2,\dots,n \tag{C-3}$$

$$km \frac{1}{g\,A} \frac{dQ_1}{dt} + \frac{(H_1 + W_1 - H_{in})}{\Delta x/2} + km \frac{f}{2g\,D} \frac{Q_1|Q_1|}{A^2} = 0 \tag{C-4}$$

$$km \frac{1}{g\,A} \frac{dQ_{n+1}}{dt} + \frac{(H_{out} - H_n - W_n)}{\Delta x/2} + km \frac{f}{2g\,D} \frac{Q_{n+1}|Q_{n+1}|}{A^2} = 0 \tag{C-5}$$

C.2.2. Formulación del Componente ***Tank***

La formulación del componente Tank, utilizado para representar tanques o depósitos, es bastante sencilla, ya que fija el valor de la altura piezométrica en el puerto de salida, es decir, en el extremo de tubería conectado al tanque.

$$H_{out} = Dato \tag{C-6}$$

Es evidente que la causalidad del tanque permite su conexión a tuberías cuyo extremo conectado sea resistivo sin que surja ningún problema. Sin embargo, su conexión a una tubería con el extremo capacitivo conlleva a un índice superior.

C.2.3. Formulación del Componente ***ValveExit***

La ecuación representativa de una válvula es la siguiente.

$$Q = C_d\, A\, x \sqrt{2\, g\, H_{in}} \tag{C-7}$$

Donde A es el área de garganta de la válvula en posición completamente abierta, C_d es el coeficiente de descarga, x es el grado de apertura de la válvula medido entre 0 y 1, y H_{in} es la altura piezométrica a la entrada de la válvula.

En principio, este componente se puede conectar al extremo resistivo de una tubería. Suponiendo que este extremo es el de salida, el procesamiento simbólico de EcosimPro combina la ecuación del momento en el último volumen de control (C-14) y la ecuación (C-7) que representa la válvula. El resultado es la siguiente ecuación diferencial ordinaria:

$$\frac{dQ_{n+1}}{dt} = -\frac{g\, A}{km}\frac{\left(\frac{1}{2\, g}\left(\frac{Q_{n+1}}{C_d\, A\, x}\right)^2 - H_n\right)}{\Delta x/2} - \frac{f}{2D}\frac{Q_{n+1}|Q_{n+1}|}{A} = 0 \tag{C-8}$$

Sin embargo, es importante destacar que esta ecuación diferencial ordinaria presenta un problema de índice superior. En el caso de un cierre completo de la válvula, donde el caudal se forzará a ser nulo, esta ecuación no es aplicable debido a la división por cero.

A pesar de esto, es posible utilizar la válvula conectada al extremo resistivo de una tubería, siempre y cuando no se imponga un cierre completo y absoluto de la válvula ($x = 0$). La apertura puede ser muy pequeña pero no nula. No obstante, una aproximación más robusta es conectar la válvula a extremos capacitivos, lo que permite el cierre completo de la válvula.

C.2.4. Formulación del Componente ***Flow***

Este componente se emplea para imponer un flujo en el extremo de una tubería, pero cuando la causalidad del extremo conectado de la tubería es resistiva, se origina un problema de índice superior al intentar imponer el flujo.

Existen dos soluciones alternativas al problema. La primera consiste en permitir un pequeño retraso del caudal impuesto con una ecuación del siguiente tipo:

$$\frac{dQ_{n+1}}{dt} = \frac{Q_{fixed} - Q_{n+1}}{\tau} = 0 \qquad \text{(C-9)}$$

Donde Q_{fixed} es el flujo que se impone, que normalmente será función del tiempo. y τ es una constante de tiempo para proporcionar un pequeño filtrado. Con esta solución, la ecuación del momento del último volumen de control de momento se utiliza para calcular la presión en la sección de salida:

$$H_{out} = H_n - km\frac{\Delta x}{2\,g\,A}\frac{dQ_{n+1}}{dt} - km\frac{f\Delta x}{4g\,D}\frac{Q_{n+1}|Q_{n+1}|}{A^2} = 0 \qquad \text{(C-10)}$$

Es fundamental notar que debido al segundo término de la ecuación, H_{out} puede alcanzar valores enormemente altos o bajos si Q_{fixed} experimenta cambios rápidos y la constante de tiempo τ es mucho menor que el intervalo de tiempo correspondiente a Courant igual a 1. No obstante, la física del problema dice que los incrementos de altura piezométrica son finitos incluso en el caso de maniobras instantáneas y vienen dados por la ecuación de Joukowsky:

$$\Delta H = \pm\frac{a}{gA}\Delta Q \qquad \text{(C-11)}$$

La segunda solución consiste en obligar a que la causalidad del extremo conectado al componente ***Flow*** sea capacitiva; en ese caso, es posible imponer maniobras instantáneas y obtener incrementos de presión no muy alejados del de Joukowsky.

C.2.5. Formulación del Componente ***Collector***

El componente ***Collector*** se utiliza para representar un punto en el que confluyen varias tuberías. La ecuación que representa el comportamiento del colector es que la suma de caudales o flujo de las tuberías conectadas, considerando positivos los de entrada y negativos los de salida, debe ser nula. Esta ecuación se puede expresar como

$$\sum_{j\,\in JN} Q_{j,n+1} \qquad \sum_{k\,\in KN} Q_{k,1} = 0 \qquad \text{(C-12)}$$

donde

JN es el conjunto de todas las tuberías conectadas al colector por su último nodo

KN es el conjunto de todas las tuberías conectadas al colector por su primer nodo

Cuando los extremos de las tuberías conectadas son de tipo resistivo, se genera un problema de índice superior debido a la restricción entre las variables de estado, que en este caso son los flujos en los extremos de las tuberías. EcosimPro aborda de manera eficiente este problema de índice superior aplicando la metodología de Pantelides. Al utilizar Pantelides, se deriva la ecuación de suma de flujos igual a 0, lo que conduce a una ecuación que establece que la suma de las derivadas de los flujos también debe ser nula. A continuación, la herramienta selecciona una de las tuberías que convergen en el colector y considera como variables algebraicas tanto el valor del flujo desde dicha tubería como su derivada. Los flujos restantes en el colector continúan siendo variables de estado y se calculan mediante integración, pero el flujo seleccionado como variable algebraica se calcula a partir de la condición de suma de flujos igual a 0, y su derivada se calcula a partir de la ecuación de suma de derivadas de flujos igual a 0. Por último, la ecuación del momento en el último volumen de control de la tubería seleccionada permite calcular la altura piezométrica en dicha tubería.

En resumen, es apropiado considerar causalidad resistiva en los extremos de las tuberías conectados a colectores.

C.3. Código Fuente de la Librería **TRANLIQ**

```
LIBRARY TRANLIQ
USE MATH
CONST REAL g = 9.806
PORT Hydraulic
   SUM REAL Q     UNITS "m3/s"
   EQUAL REAL H   UNITS "m"
END PORT

ENUM PipeType = {RR, CR, RC, CC}

COMPONENT Pipe(INTEGER n = 100, ENUM PipeType type =RR)
   PORTS
      IN Hydraulic h_in
      OUT Hydraulic h_out
   DATA
      REAL L = 600      UNITS "m"     "Pipe length"
      REAL D = 0.5      UNITS "m"     "Pipe inside diametert"
      REAL a = 1200.    UNITS "m/s"   "Wave speed"
      REAL f = 0.018    UNITS "-"     "Friction factor"
      REAL kdamp = 0.15 UNITS "-"     "Damping factor for artificial viscosity"
      REAL Ho = 150     UNITS "m"     "Initial elevation at inlet section"
      REAL Qo = 0.47744 UNITS "m3/s" "Initial flow"
   DECLS
      DISCR REAL A      UNITS "m"     "Flow area"
      DISCR REAL dx     UNITS "m"     "Nodal length"
      REAL H[n]         UNITS "m"     "Array of piezometric heads"
      REAL Q[n+1]       UNITS "m3/s" "Array of flows"
      REAL W[n]         UNITS "m"     "Array of artificial viscosity head"
      DISCR REAL km     UNITS "-"     "Multiplier of momentum terms"
   INIT
      A = 0.25 * PI * D**2
      dx = L/n
      IF(type == RR) THEN
         km = 1.
      ELSEIF(type == CR OR type == RC) THEN
         km = n/(n-0.5)
      ELSEIF(type == CC) THEN
         km = n/(n-1)
```

```
      END IF
      FOR(i IN 1, n)
         IF(type == RR) THEN
            H[i] = Ho - ((i-1+0.5)/n) * f * (L/D) * Qo*abs(Qo)/(2*g*A**2)
         ELSEIF(type == CC) THEN
            H[i] = Ho - ((i-1)/(n-1)) * f * (L/D) * Qo*abs(Qo)/(2*g*A**2)
         ELSEIF(type == RC) THEN
            H[i] = Ho - ((i-1+0.5)/(n-0.5))* f * (L/D) * Qo*abs(Qo)/(2*g*A**2)
         ELSEIF(type == CR) THEN
            H[i] = Ho - ((i-1)/(n-0.5)) * f * (L/D) * Qo*abs(Qo)/(2*g*A**2)
         END IF
      END FOR
      FOR(i IN 1, n+1)
         Q[i] = Qo
      END FOR
      h_in.Q = Qo
      h_out.Q = Qo
   CONTINUOUS
      h_in.Q = Q[1]
      h_out.Q = Q[n+1]
      EXPAND_BLOCK(type == RR)
         km*Q[1]' =   - A * g * (H[1] + W[1] - h_in.H)/ (dx/2) \
                      - km*f*Q[1]*abs(Q[1])/(2*D*A)
         km*Q[n+1]' = - A * g * (h_out.H - H[n] - W[n])/ (dx/2) \
                      - km*f*Q[n+1]*abs(Q[n+1])/(2*D*A)
      END EXPAND BLOCK
      EXPAND_BLOCK(type == CR)
         h_in.H = H[1]
         km*Q[n+1]' =  - A * g * (h_out.H - H[n] - W[n])/ (dx/2) \
                       - km*f*Q[n+1]*abs(Q[n+1])/(2*D*A)
      END EXPAND BLOCK
      EXPAND_BLOCK(type == RC)
         h out.H = H[n]
         km*Q[1]' = - A * g * (H[1] + W[1] - h_in.H)/ (dx/2) \
                    - km*f*Q[1]*abs(Q[1])/(2*D*A)
      END EXPAND BLOCK
      EXPAND_BLOCK(type==CC)
         h in.H = H[1]
         h_out.H = H[n]
      END EXPAND BLOCK
      EXPAND (i IN 1, n)
         H[i]' = -a**2/(g*A) * (Q[i+1] - Q[i])/dx
      EXPAND (i IN 2, n)
         km*Q[i]' = - A * g * (H[i]+W[i]-H[i-1]-W[i-1])/dx \
                    - km * f*Q[i]*abs(Q[i])/(2*D*A)
      EXPAND (i IN 1, n)
         W[i] = - kdamp*a*(Q[i+1]-Q[i])/(g*A)
END COMPONENT

COMPONENT Tank
   PORTS
      OUT Hydraulic h_out
   DATA
      REAL H = 150. UNITS "m" "Tank Level"
   CONTINUOUS
      h_out.H = H
END COMPONENT

COMPONENT Collector
   PORTS
      IN Hydraulic h_in
   CONTINUOUS
      h_in.Q = 0
END COMPONENT

COMPONENT Flow
   PORTS
      IN Hydraulic h_in
   DATA
   DECLS
      BOUND REAL Q UNITS "m3/s" "Specified flow"
   CONTINUOUS
      h_in.Q = Q
```

```
112 END COMPONENT
113
114 COMPONENT ValveExit
115    PORTS
116       IN Hydraulic h_in
117    DATA
118       REAL Cd = 1     UNITS "-"  "Discharge coefficient"
119       REAL A = 0.009 UNITS "m2" "Nozzle area"
120    DECLS
121       CONST REAL dHlam = 0.001 UNITS "m" "Pressure diff. for laminar flow"
122       BOUND REAL pos UNITS "-"  "Valve Position between 0 & 1"
123    CONTINUOUS
124       h_in.Q = Cd*A*pos*fsqrt(2*g*h_in.H, dHlam)
125 END COMPONENT
```

APÉNDICE D Técnicas de Programación para la Nueva Aproximación de Estacionarios

En este Apéndice, se describen las técnicas de programación necesarias para materializar la nueva aproximación de estacionarios, descrita en la sección 5.2.

D.1. Obtención de la Topología

El primer paso para obtener la topología por parte del modelo es la numeración automática de los componentes. Esta fase implica asignar un número entero consecutivo a cada componente de un tipo específico. Estos identificadores enteros posibilitarán la vinculación de arreglos de datos y variables a los componentes, así como la formulación de bucles para operar con estos arreglos.

En EcosimPro, es factible realizar la numeración automática de los componentes, utilizando el bloque **INIT** de los mismos. Para ilustrar esta técnica, se tomará como ejemplo una librería hidráulica con tres tipos de componentes: *"**Pipe**"*, *"**Tank**"* y *"**Outflow**"*. Los componentes de tipo *"**Pipe**"* tienen dos puertos hidráulicos (uno de entrada y otro de salida), mientras que los componentes de tipo *"**Tank**"* disponen de un único puerto, que es de salida, y los componentes de tipo *"**Outflow**"* tienen un único puerto, el cual se considera es de entrada al componente. El Listado D-1 muestra los fragmentos de código que realizan la numeración automática de estos componentes en la librería hidráulica ejemplo.

Con el propósito de contabilizar los componentes de cada tipo, se declaran tres variables globales de tipo entero: `npipe`, `ntank` y `nflow`, la cuales se inicializan a cero. Estas variables funcionan como contadores que, tras la primera ejecución del bloque **INIT,** reflejan con exactitud el número total de componentes del tipo correspondiente en el modelo.

Es importante resaltar que, en EcosimPro, un identificador declarado en una librería fuera de una función, puerto, componente o clase, como es el caso de `npipe npipe`, `ntank` y `nflow,` puede ser utilizado desde cualquier lugar de esa librería o incluso desde fuera de ella, especificando el nombre de la librería; en otras palabras, posee un alcance global.

En la definición del tipo de componente *"**Pipe**"*, se introduce una variable local de tipo entero llamada `ipipe`. El propósito de esta variable es representar el número específico de cada instancia del componente *"**Pipe**"*. En el bloque **INIT** del componente *"**Pipe**"*, se realiza una comprobación inicial para asegurar que el identificador no haya sido previamente asignado. Si el identificador no está asignado previamente, se procede a

incrementar el contador de instancias `npipe` y se asigna el valor del contador a la variable local `ipipe`.

La definición de los componentes "***Tank***" y "***Outflow***" sigue un enfoque completamente similar, definen respectivamente las identificadores locales `itank` e `iflow`, y gestionan en el bloque **`INIT`** los contadores globales respectivos y la asignación de los identificadores locales.

*Listado D-1: Fragmento de Código de EcosimPro con Ejemplo de Numeración Automática de Componentes de los Tipos "**`Pipe`**", "**`Tank`**" y "**`Outflow`**"*

```
INTEGER npipe = 0    "Counter for the number of Pipes
INTEGER ntank = 0    "Counter for the number of Tanks
INTEGER nflow = 0    "Counter for the number of Outflows
COMPONENT Pipe
    DECLS
        INTEGER ipipe  --pipe internal number
    INIT
        IF (ipipe == 0) THEN
            npipe = npipe + 1
            ipipe = npipe
        END IF
        ...
END COMPONENT

COMPONENT Tank
    DECLS
        INTEGER itank  --tank internal number
    INIT
        IF (itank == 0) THEN
            ntank = ntank + 1
            itank = ntank
        END IF
        ...
END COMPONENT

COMPONENT Outflow
    DECLS
        INTEGER iflow  --outflow internal number
    INIT
        IF (iflow == 0) THEN
            nflow = nflow + 1
            iflow = nflow
        END IF
        ...
END COMPONENT
```

La numeración de componentes es el primer paso para extraer la topología del modelo. El segundo paso consiste en identificar cómo se interconectan los componentes. La metodología para esta identificación depende del método de conexión empleado. Tal y como se discute en el capítulo 3, existen dos métodos de conexionado: mediante nodos y mediante corrientes.

D.1.1. Detección de la Topología con Conexionado Mediante Nodos

En el caso de un conexionado mediante nodos, donde un nodo sirve como el punto de conexión para uno o más puertos de componentes, es esencial asignar números enteros secuenciales a los nodos de cada tipo de puerto para su identificación. Asimismo, se debe establecer una correspondencia entre los nodos, identificados por sus números, y los puertos de los componentes.

Para llevar a cabo esta numeración, se requiere una variable global para cada tipo de puerto. Esta variable funciona como un contador, registrando el número total de nodos asociados al tipo de puerto correspondiente. Adicionalmente, en la definición de cada tipo de puerto, se declara una variable de tipo entero cuyo comportamiento se define mediante el prefijo **EQUAL**. El propósito de esta variable es identificar a que nodo está conectado cada uno de los puertos del tipo considerado en los componentes. Esta estructura permite establecer una correspondencia clara entre los nodos, definidos por sus identificadores, y los puertos de los componentes.

Para el ejemplo de librería hidráulica considerado, el Listado D-2 muestra un fragmento de código. En este fragmento, se declara la variable global `node`, utilizada como contador para el número de nodos hidráulicos, y se define el puerto `Hydraulic` que incorpora la variable `inode` para la identificación de los nodos hidráulicos. La inclusión del prefijo **EQUAL** en la declaración de la variable `inode` implica que todos los puertos que están interconectados entre sí comparten la misma variable. Es importante notar que, en este fragmento de código, el puerto incluye únicamente la variable relevante para la detección de la topología, aunque también podría contener otras variables.

*Listado D-2: Declaración de Contador de Nodos y de Puerto "**Hydraulic**" para Conexionado Mediante Nodos*

```
INTEGER node = 0     "Counter for the number of hydraulic nodes"
PORT Hydraulic
    EQUAL INTEGER inode = 0 --hydraulic nodes identifier
END PORT
```

Durante la ejecución del bloque **INIT** del componente *"**Pipe**"*, se comprueba si la variable `inode` en alguno de los dos puertos aún mantiene su valor inicial de cero; esto indica que el nodo correspondiente aún no ha sido identificado. En esta circunstancia, se procede incrementando en uno el contador `node` que registra el número total de nodos hidráulicos y asignando el número actual de nodos a la variable `inode` del puerto en cuestión.

La identificación de los nodos a los que se conectan los componentes de tipo *"**Tank**"* y *"**Outflow**"* se hace de forma completamente similar, con la salvedad de que estos componentes poseen un único puerto hidráulico en lugar de dos. En el caso del componente *"**Tank**"*, este puerto es de salida, mientras que en el caso del componente *"**Outflow**"*, es de entrada.

El Listado D-3 muestra los fragmento de código de los componente *"**Pipe**"*, *"**Tank**"* y *"**Outflow**"* que llevan a cabo la asignación de números de nodo a los puertos.

*Listado D-3: Identificación Nodos "**Hydraulic**" en Comp. "**Pipe**", "**Tank**" y "**Outflow**"*

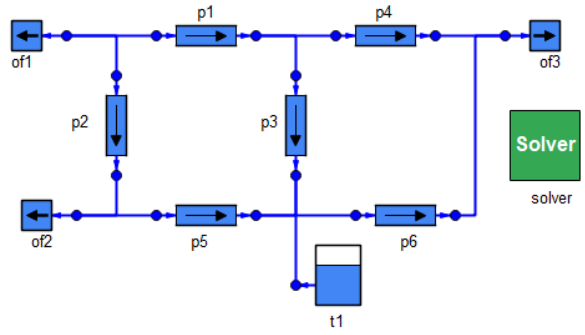
```
INTEGER npipe = 0    "Counter for the number of pipes"
INTEGER ntank = 0    "Counter for the number of Tanks
INTEGER nflow = 0    "Counter for the number of Outflows
INTEGER node = 0     "Counter for the number of hydraulic nodes"
PORT Hydraulic
    EQUAL INTEGER inode  --hydraulic nodes identifier
END PORT
COMPONENT Pipe
   PORTS
       IN Hydraulic h_in     "Hydraulic inlet port"
       OUT Hydraulic h_out   "Hydraulic outlet port"
    DECLS
        INTEGER ipipe = 0  --pipe identifier
    INIT PRIORITY 100
        IF (ipipe == 0) THEN
            npipe = npipe + 1
            ipipe = npipe
            IF(h_in.inode == 0) THEN
                node = node + 1
                h_in.inode = node
            END IF
            IF(h_out.inode == 0) THEN
                node = node + 1
                h_out.inode = node
            END IF
        END IF
        ...
END COMPONENT

COMPONENT Tank
   PORTS
       OUT Hydraulic h_out   "Hydraulic outlet port"
    DECLS
        INTEGER itank  --tank identifier
    INIT PRIORITY 90
        IF (itank == 0) THEN
            ntank = ntank + 1
            itank = ntank
            IF(h_out.inode == 0) THEN
                node = node + 1
                h_out.inode = node
           END IF
        END IF
        ...
END COMPONENT

COMPONENT Outflow
   PORTS
       OUT Hydraulic h_in   "Hydraulic inlet port"
    DECLS
        INTEGER iflow  --outflow internal number
    INIT PRIORITY 800
        IF (iflow == 0) THEN
            nflow = nflow + 1
            iflow = nflow
            IF(h_in.inode == 0) THEN
                node = node + 1
                h_in.inode = node
           END IF
        END IF
        ...
END COMPONENT
```

En EcosimPro, es posible asignar una prioridad, representada por un número entero positivo o negativo, al bloque **INIT** de cada clase o tipo de componentes (el funcionamiento y efecto de las prioridades se explica en la sección 3.2.1 de este documento). Se puede observar que en las definiciones anteriores de los componentes tipo *"**Pipe**"* y *"**Tank**"*, se ha establecido una prioridad de **100** para el bloque **INIT** del *"**Pipe**"*, y una prioridad de **90** para el bloque **INIT** del *"**Tank**"*. Esto significa que el bloque **INIT** de los componentes tipo *"**Pipe**"* se ejecutará antes que el bloque **INIT** de los componentes tipo "***Tank***".

En el caso específico de este ejemplo, el orden de ejecución de los bloques **INIT** de los componentes de tipo *"**Pipe**"* y de los componentes de tipo "***Tank***" carece de relevancia. No obstante, es importante resaltar que existen situaciones en las cuales es apropiado otorgar prioridad a la inicialización de un tipo específico de componente. Un ejemplo ilustrativo de esto se encuentra en el modelado de circuitos eléctricos, donde es común seguir la convención de que el nodo correspondiente al componente *"**tierra**"* (***ground***) debe ser designado como el nodo número **0**. Este objetivo se alcanza asignando una prioridad superior al bloque **INIT** del componente *"**tierra**"* en comparación con cualquier otro componente e iniciando el contador de nodos en **-1**.

Es relevante destacar que la obtención de la topología produce lo que en terminología de SPICE se conoce como NETLIST; esto es, una lista de componentes con la identificación de los nodos a los que se conecta cada uno de ellos.

Con el fin de ilustrar el funcionamiento de la obtención de la topología, la Figura D-1 muestra un ejemplo de circuito hidráulico que consta de 6 componentes de tipo *"**Pipe**"*, 1 componente del tipo *"**Tank**"*, y 3 componentes del tipo *"**Outflow**"*. En el lado derecho de la figura, se muestra una tabla que presenta los resultados obtenidos para las variables que representan la topología después de la ejecución de las sentencias iniciales (bloques **INIT** de los componentes):



Pipe Name	ipipe	h_in node	h_out node
p1	1	1	2
p2	2	1	3
p3	3	2	4
p4	4	2	5
p5	5	3	4
p6	6	4	5
Tank Name	**itank**		**h_out node**
t1	1		4
Outflow Name	**iflow**	**h_in node**	
of1	1	1	
of2	2	3	
of3	3	5	

Figura D-1: Ejemplo de extracción de la topología con EcosimPro de un modelo de circuito hidráulico

D.1.2. Detección de la Topología con Conexionado Mediante Corrientes

En el caso de utilizar las corrientes como elemento de conexionado, las diferencias fundamentales con el conexionado mediante nodos son que se prohíben las conexiones múltiples a los puertos, es decir, las conexiones entre puertos deben ser uno a uno, y un puerto de salida debe conectarse siempre a un puerto de entrada.

Casos típicos de conexionado mediante corrientes se dan en los modelos de balance térmico de plantas térmicas, en los modelos de turbinas de gas y en los modelos de procesos químicos.

La técnica a emplear para extraer la topología es muy similar a la del conexionado mediante nodos. La diferencia es que la declaración de un tipo de puerto diseñado para el conexionado mediante corrientes se le añade la palabra clave **`SINGLE`** detrás del nombre del tipo de puerto para indica que los puertos de ese tipo solo aceptan una conexión

El siguiente trozo de código muestra la declaración de la variable global para contar las corrientes del tipo `Gas`, y la declaración del tipo de puerto `Gas`. La palabra clave **`SINGLE`**. La variable `istream` se utiliza para identificar el número de la corriente, y lógicamente es de tipo **`EQUAL`**.

Listado D-4: Declaración de Contador de Corrientes y de Puerto "`Gas`" para Conexionado Mediante Corrientes

```
INTEGER nStream = 0     "Counter for the number of Gas Streams"
PORT Gas SINGLE
    EQUAL INTEGER iStream = 0 --hydraulic nodes identifier
END PORT
```

D.2. Relleno Estructuras Globales con Topología y Datos de Componentes

Como se ha señalado anteriormente, en el lenguaje EcosimPro los identificadores declarados fuera de funciones, componentes, puertos y clases poseen un alcance global. Esta característica permite utilizar arreglos de variables globales para almacenar la topología y los datos de los componentes. Considérese como ejemplo el caso de un componente *"**`Pipe`**"* con los tres datos siguientes: longitud, diámetro, y coeficiente de rugosidad Hazen-Williams. El Listado D-5 ilustra la técnica utilizada para rellenar estructuras de datos globales con la topología y los datos de todas las tuberías del modelo.

Para transferir la información topológica se definen dos arreglos de variables enteras (`nd1` y `nd2`) que contienen los números de nodo de entrada y salida, respectivamente. A su vez, para transferir los datos, se definen tres arreglos globales de variables reales (`L_pipe`, `D_pipe` y `CHW_pipe`), que actúan como contenedores para almacenar los datos de todas las tuberías del modelo. Estos arreglos están dimensionados a un número máximo de tuberías, declarado con la constante `MAX_PIPE`. Aunque EcosimPro no permite un

dimensionado dinámico de estos arreglos, es fácil recompilar la librería ajustando el valor de la constante `MAX_PIPE`, según sea necesario. Es importante destacar que los arreglos de variables globales reales se declaran como discretos mediante la palabra clave **`DISCR`**, con el propósito de evitar que variables, que son exclusivamente datos o que solo se utilizan inicialmente en el estado estacionario, sean clasificadas como variables desconocidas por el compilador de modelos.

Finalmente, en el bloque **`INIT`** del componente, después de asignar el identificador entero `ipipe` a la tubería, se copian la información topológica y los datos locales en la posición correspondiente (`ipipe`) de los arreglos globales mencionados anteriormente.

Este procedimiento asegura que los datos específicos de cada tubería se almacenen de manera estructurada en arreglos globales, facilitando así el acceso y la manipulación global de la información sobre todas las tuberías en el sistema del modelo.

*Listado D-5: Relleno Estructuras Globales de Componentes "**`Pipe`**" Usando Variables*

```
INTEGER npipe = 0   "Counter for the number of pipes"
...
--       Global Variables
CONST INTEGER MAX_PIPE = 500   "Maximum number of pipes in the model"
      INTEGER nd1[MAX_PIPE]    "Number of node at pipe inlet"
      INTEGER nd2[MAX_PIPE]    "Number of node at pipe outlet"
DISCR REAL L_pipe[MAX_PIPE]    "Global array for pipe lenghs"
DISCR REAL D_pipe[MAX_PIPE]    "Global array for pipe diameters"
DISCR REAL CHW_pipe[MAX_PIPE]  "Global array for pipe Hazen-Williams coeff."
...
COMPONENT Pipe
   PORTS
       IN Hydraulic h_in     "Hydraulic inlet port"
      OUT Hydraulic h_out    "Hydraulic outlet port"
    DATA
        REAL L = 600      UNITS "m"    "Pipe length"
        REAL D = 0.5      UNITS "m"    "Pipe inside diametert"
        REAL CHW = 120.   UNITS "-"    "Hazen Williams Coefficient"
    DECLS
        INTEGER ipipe = 0  --pipe identifier
    INIT PRIORITY 100
        IF (ipipe == 0) THEN
            npipe = npipe + 1
            ipipe = npipe
            IF(h_in.inode == 0) THEN
                node = node + 1
                h_in.inode = node
            END IF
            IF(h_out.inode == 0) THEN
                node = node + 1
                h_out.inode = node
            END IF
            nd1[ipipe] = h_in.inode
            nd2[ipipe] = h_out.inode
            L_pipe[ipipe] = L
            D_pipe[ipipe] = D
            CHW_pipe[ipipe] = CHW
        END IF
        ...
END COMPONENT
```

Es interesante mencionar que existe otra posibilidad para almacenar los datos de los componentes. EcosimPro ofrece la posibilidad de utilizar Clases, lo que permite estructurar los datos y variables globales de manera clara y eficiente. Las Clases de EcosimPro se asemejan a las clases en lenguajes de programación orientados a objetos, como C++. Cada clase consta de una estructura de datos y una serie de métodos para la manipulación de la información contenida en la clase.

En lugar de utilizar varios arreglos de variables, se declara una Clase por cada tipo de componente, dicha clase debe incluir como atributos los datos y variables resultado del cálculo. Luego, se define un arreglo de objetos de la clase dimensionado al número máximo de tuberías. El Listado D-6 muestra el ejemplo considerado anteriormente, pero utilizando un vector de objetos para rellenar los datos globales.

*Listado D-6: Relleno Estructuras Globales de Componentes "**Pipe**" Usando Objetos*

```
INTEGER npipe = 0   "Counter for the number of pipes"
...
CONST INTEGER MAX_PIPE = 500
CLASS PipeClass
   DECLS
      INTEGER nd1              "Node at inlet"
      INTEGER nd2              "Node at outlet"
      REAL L      UNITS "m"    "Pipe length"
      REAL D      UNITS "m"    "Pipe inside diametert"
      REAL CHW    UNITS "-"    "Friction factor"
END CLASS
PipeClass P[MAX_PIPE]
...
COMPONENT Pipe
   PORTS
       IN Hydraulic h_in     "Hydraulic inlet port"
      OUT Hydraulic h_out    "Hydraulic outlet port"
    DATA
        REAL L = 600      UNITS "m"    "Pipe length"
        REAL D = 0.5      UNITS "m"    "Pipe inside diametert"
        REAL CHW = 120.   UNITS "-"    "Hazen Williams Coefficient"
    DECLS
        INTEGER ipipe = 0  --pipe identifier
    INIT PRIORITY 100
        IF (ipipe == 0) THEN
            npipe = npipe + 1
            ipipe = npipe
            IF(h_in.inode == 0) THEN
                node = node + 1
                h_in.inode = node
            END IF
            IF(h_out.inode == 0) THEN
                node = node + 1
                h_out.inode = node
            END IF
            P[ipipe].nd1 = h_in.inode
            P[ipipe].nd2 = h_out.inode
            P[ipipe].L = L
            P[ipipe].D = D
            P[ipipe].CHW = CHW
        END IF
        ...
END COMPONENT
```

D.3. Desarrollo de Resolvedores

El nuevo paradigma o aproximación para el cálculo de estacionarios con herramientas de MSOO brinda total libertad al desarrollador para emplear tanto el método orientado a ecuaciones como el método modular secuencial. En contraste, la aproximación tradicional de MSOO obliga a utilizar exclusivamente el método orientado a ecuaciones.

Para implementar el algoritmo de solución, se puede utilizar el bloque **`INIT`** de un componente que represente al "*`Resolvedor`*". La prioridad de este bloque **`INIT`** debe ser la más baja entre todos los componentes, asegurando así que la obtención de la topología y la copia de los datos locales a estructuras de datos globales se hayan completado antes de proceder a la solución.

La implementación del algoritmo de solución deseado puede verse facilitada mediante la definición previa de clases que implementen los algoritmos de solución. Cabe concebir al menos dos clases, una clase para resolver sistemas de ecuaciones algebraicos y otra clase para la aplicación del método modular secuencial. En el contexto de esta tesis se ha desarrollado una biblioteca de clases destinada a la resolución de sistemas de ecuaciones algebraicas dispersas, que es aplicable al desarrollo de otras aplicaciones simulación estacionaria que utilicen el método orientado a ecuaciones. Dicha biblioteca o librería se presenta en el Apéndice E.

De manera análoga, si se desea aplicar el método modular secuencial, se requeriría la creación de una biblioteca de clases con métodos para identificar los lazos de reciclaje a partir de la topología, seleccionar las corrientes de rotura de dichos lazos de reciclaje, lanzar la ejecución ordenada de los bloques **`INIT`** de los componentes (donde se supone que se incluiría el procedimiento de solución del componente), e iterar hasta lograr la convergencia.

D.4. Volcado de la Solución en Variables Locales de los Componentes

El componente *"**`Resolvedor`**"* realiza el cálculo de la solución del sistema de ecuaciones, pero es preciso volcar los resultados de la solución en variables locales de los componentes de la malla. La forma de comunicar o transferir dicha solución a los componentes de la malla, es hacer uso de variables globales u objetos globales. La opción de transmitir esta información mediante conexiones no físicas se descarta debido a la complejidad asociada con múltiples conexiones no físicas.

Este proceso es inverso al de Relleno de los Datos Globales. Ahora se busca copiar variables globales a variables locales de los componentes. Una manera de lograr este objetivo es incluir un bloque de código con dicho copiado en el bloque **`INIT`** de los componentes y lanzar una nueva ejecución de los bloques **`INIT`**. La forma de desencadenar esta nueva ejecución del bloque **`INIT`** de los componentes es mediante una

llamada a la función **EXEC_INIT()**, cuyo funcionamiento se explica en la sección 3.2.1, desde el propio bloque **INIT** del componente *"**Resolvedor**"*.

utilizar una bandera global para diferenciar entre la primera ejecución, utilizada para extraer la topología y rellenar los datos globales, y esta segunda ejecución, empleada para transferir los resultados desde variables globales a variables locales.

Ahora bien, es necesario incorporar estructuras de control flujo en los bloques **INIT** de los componentes. Estos bloques contienen dos secciones de código distintas: la primera se encarga de obtener la topología y copiar los datos a variables globales, mientras que la segunda se dedica al copiado de la solución en variables locales. La capacidad de elegir entre la ejecución de la primera sección de código o la segunda es esencial. Dicha selección se realiza mediante una estructura de control de flujo que consulta el valor de una variable global que puede ser de tipo booleano o enumerado. El componente *"**Resolvedor**"* modifica el valor de esta variable global justo antes de la llamada a la función **EXEC_INIT()**, lo que permite distinguir entre la primera y la segunda ejecución de los bloques **INIT**.

En el fragmento de código del Listado D-7, se presenta la declaración de la variable global `AssignSolution`, la cual permite distinguir entre la primera y la segunda ejecución del bloque **INIT**, así como las dos secciones del bloque **INIT** del Componente "***Pipe***". Además, se muestra cómo el "***Resolvedor***" modifica el valor de `AssignSolution` para forzar la nueva ejecución de los bloques **INIT**.

*Listado D-7: Volcado de la Solución en Variables Locales del Componente "**Pipe**"*

```
BOOLEAN AssignSolution = FALSE
COMPONENT Pipe
   ...
    DECLS
        REAL H1 = 600    UNITS "m"    "Pipe length"
        REAL H2 = 0.5    UNITS "m"    "Pipe inside diametert"
        REAL Q = 120.    UNITS "m3/s" "Volume flow"
   INIT PRIORITY 100
      IF (ipipe == 0) THEN
         npipe = npipe + 1
         ipipe = npipe
         ...
      END IF
      IF(NOT AssignSolution) THEN
         --
         H1 = Xunk[nd1]
         H2 = Xunk[nd2]
         Q = Xunk[node+ipipe]
      END IF
      ...
END COMPONENT

COMPONENT Resolvedor
   INIT PRIORITY  0
```

```
      IF(NOT AssignSolution) THEN
         --Calculation of the solution
         ...
         --End of calculation of the solution
         AssignSolution = TRUE
         EXEC_INIT()
      END IF

END COMPONENT
```

APÉNDICE E LIBRERÍA SPARSE

La librería SPARSE es una librería de clases de EcosimPro, que ofrece componentes para facilitar la construcción de resolvedores de sistemas de ecuaciones algebraicas dispersas en el lenguaje EcosimPro. SPARSE incluye las siguientes partes:

- Interfaz a la librería `SuperLU` (X. Li et al., 2011)
- Clase `SPARSE_MATRIX` para la representación de matrices dispersas
- Clase `NLEQN_SYSTEM` para resolver sistemas de ecuaciones algebraicos dispersos:

E.1. Interfaz a la Librería SuperLU

SuperLU (Li et al., 2011) también conocido como Super Sparse LU, es una colección de tres bibliotecas en ANSI C diseñadas específicamente para abordar la resolución de sistemas de ecuaciones lineales dispersas de la forma $AX = B$. En esta formulación, A representa una matriz cuadrada no singular de tamaño $n \times n$, mientras que X y B son matrices densas de tamaño $n \times nrhs$, donde $nrhs$ denota el número de lados derechos y de vectores de solución.

Las bibliotecas de SuperLU están diseñadas para optimizar el proceso de eliminación gaussiana, aprovechando la dispersión de las matrices y sacando ventaja de ciertas características de la arquitectura del equipo informático, tales como jerarquías de memoria (cachés) y paralelismo.

SuperLU se compone de tres bibliotecas, cada una de las cuales es adecuada para diferentes entornos computacionales:

1. **Sequential SuperLU**: Diseñada para procesadores secuenciales, esta versión se adapta a sistemas con jerarquías de memoria (cachés) y es eficiente en entornos con un solo procesador.
2. **Multithreaded SuperLU (SuperLU MT)**: Orientada a sistemas multiprocesadores de memoria compartida, esta versión puede aprovechar efectivamente hasta 16 o 32 procesadores en matrices extensas.
3. **Distributed SuperLU (SuperLU DIST)**: Diseñada para sistemas multiprocesadores de memoria distribuida que utilizan. SuperLU DIST tiene la capacidad de aprovechar cientos de procesadores paralelos en matrices de gran tamaño.

SuperLU está implementada en lenguaje ANSI-C. Para utilizar las funciones de esta biblioteca desde otros lenguajes, como EcosimPro, es necesario construir funciones de interfaz. En el código de las librerías SuperLU, se ofrecen ejemplos de funciones de interfaz para permitir su llamada desde otros lenguajes. En particular, se proporciona la función **`c_fortran_dgssv_()`** escrita en C, diseñada para ser llamada desde FORTRAN,

actuando como una interfaz para resolver sistemas de ecuaciones lineales en doble precisión mediante la versión secuencial de SuperLU. Tomando esta función como ejemplo, se ha desarrollado la función **`d_ecosimpro_superlu`** para posibilitar la utilización de SuperLU secuencial desde el entorno de EcosimPro.

*Listado E-1: Función "C" **`d_ecosimpro_superlu`** para llamar SuperLU desde EcosimPro*

```
/*
 * -- Interface from EcosimPro to SuperLU routine (version 5.1) --
 */

#include "slu_ddefs.h"

#define HANDLE_SIZE  8

/* kind of integer to hold a pointer.  Use 64-bit. */
typedef long long int fptr;

typedef struct {
    SuperMatrix *L;
    SuperMatrix *U;
    int *perm_c;
    int *perm_r;
} factors_t;

void d_ecosim_superlu(int iopt, int n, int nnz, int nrhs,
       double *values, int *rowind, int *colptr,
       double *b, int ldb,
       int *f_factors, /* a handle containing the address
                pointing to the factored matrices */
       int *info)

{
/*
 * This routine can be called from EcosimPro.
 *
 * iopt (input) int
 *      Specifies the operation:
 *      = 1, performs LU decomposition for the first time
 *      = 2, performs triangular solve
 *      = 3, free all the storage in the end
 *
 * f_factors (input/output) fptr*
 *      If iopt == 1, it is an output and contains the pointer pointing to
 *                    the structure of the factored matrices.
 *      Otherwise, it it an input.
 *
 */

    SuperMatrix A, AC, B;
    SuperMatrix *L, *U;
    int *perm_r; /* row permutations from partial pivoting */
    int *perm_c; /* column permutation vector */
    int *etree;  /* column elimination tree */
    SCformat *Lstore;
    NCformat *Ustore;
    int      i, panel_size, permc_spec, relax;
    trans_t  trans;
    mem_usage_t   mem_usage;
    superlu_options_t options;
    SuperLUStat_t stat;
    factors_t *LUfactors;
    GlobalLU_t Glu;   /* Not needed on return. */
    int    *rowind0;  /* counter 1-based indexing from Frotran arrays. */
    int    *colptr0;

    trans = NOTRANS;

```

```
62      if ( iopt == 1 ) { /* LU decomposition */
63
64          /* Set the default input options. */
65          set_default_options(&options);
66
67      /* Initialize the statistics variables. */
68      StatInit(&stat);
69
70
71      /* Adjust to 0-based indexing */
72      if ( !(rowind0 = intMalloc(nnz)) ) ABORT("Malloc fails for rowind0[].");
73      if ( !(colptr0 = intMalloc(n+1)) ) ABORT("Malloc fails for colptr0[].");
74      for (i = 0; i < nnz; ++i) rowind0[i] = rowind[i] - 1;
75      for (i = 0; i <= n; ++i) colptr0[i] = colptr[i] - 1;
76
77      dCreate_CompCol_Matrix(&A, n, n, nnz, values, rowind0, colptr0,
78                             SLU_NC, SLU_D, SLU_GE);
79      L = (SuperMatrix *) SUPERLU_MALLOC( sizeof(SuperMatrix) );
80      U = (SuperMatrix *) SUPERLU_MALLOC( sizeof(SuperMatrix) );
81      if ( !(perm_r = intMalloc(n)) ) ABORT("Malloc fails for perm_r[].");
82      if ( !(perm_c = intMalloc(n)) ) ABORT("Malloc fails for perm_c[].");
83      if ( !(etree = intMalloc(n)) ) ABORT("Malloc fails for etree[].");
84
85      /*
86       * Get column permutation vector perm_c[], according to permc_spec:
87       *   permc_spec = 0: natural ordering
88       *   permc_spec = 1: minimum degree on structure of A'*A
89       *   permc_spec = 2: minimum degree on structure of A'+A
90       *   permc_spec = 3: approximate minimum degree for unsymmetric matrices
91       */
92      permc_spec = options.ColPerm;
93      get_perm_c(permc_spec, &A, perm_c);
94
95      sp_preorder(&options, &A, perm_c, etree, &AC);
96
97      panel_size = sp_ienv(1);
98      relax = sp_ienv(2);
99
100     dgstrf(&options, &AC, relax, panel_size, etree,
101                 NULL, 0, perm_c, perm_r, L, U, &Glu, &stat, info);
102
103     if ( *info == 0 ) {
104         Lstore = (SCformat *) L->Store;
105         Ustore = (NCformat *) U->Store;
106         printf("No of nonzeros in factor L = %d\n", Lstore->nnz);
107         printf("No of nonzeros in factor U = %d\n", Ustore->nnz);
108         printf("No of nonzeros in L+U = %d\n", Lstore->nnz + Ustore->nnz);
109         dQuerySpace(L, U, &mem_usage);
110         printf("L\\U MB %.3f\ttotal MB needed %.3f\n",
111           mem_usage.for_lu/1e6, mem_usage.total_needed/1e6);
112     } else {
113         printf("dgstrf() error returns INFO= %d\n", *info);
114         if ( *info <= n ) { /* factorization completes */
115        dQuerySpace(L, U, &mem_usage);
116        printf("L\\U MB %.3f\ttotal MB needed %.3f\n",
117               mem_usage.for_lu/1e6, mem_usage.total_needed/1e6);
118         }
119     }
120
121     /* Save the LU factors in the factors handle */
122     LUfactors = (factors_t*) SUPERLU_MALLOC(sizeof(factors_t));
123     LUfactors->L = L;
124     LUfactors->U = U;
125     LUfactors->perm_c = perm_c;
126     LUfactors->perm_r = perm_r;
127     *f_factors = (fptr) LUfactors;
128
129     /* Free un-wanted storage */
130     SUPERLU_FREE(etree);
131     Destroy_SuperMatrix_Store(&A);
132     Destroy_CompCol_Permuted(&AC);
133     SUPERLU_FREE(rowind0);
134     SUPERLU_FREE(colptr0);
```

```
135     StatFree(&stat);
136
137      } else if ( iopt == 2 ) { /* Triangular solve */
138     /* Initialize the statistics variables. */
139     StatInit(&stat);
140
141     /* Extract the LU factors in the factors handle */
142     LUfactors = (factors_t*) *f_factors;
143     L = LUfactors->L;
144     U = LUfactors->U;
145     perm_c = LUfactors->perm_c;
146     perm_r = LUfactors->perm_r;
147     dPrint_Dense_Matrix("Before SuperMatrix B\n", &B);
148     dCreate_Dense_Matrix(&B, n, nrhs, b, ldb, SLU_DN, SLU_D, SLU_GE);
149     dPrint_Dense_Matrix("After SuperMatrix B\n", &B);
150
151          /* Solve the system A*X=B, overwriting B with X. */
152          dgstrs (trans, L, U, perm_c, perm_r, &B, &stat, info);
153
154     Destroy_SuperMatrix_Store(&B);
155     StatFree(&stat);
156
157      } else if ( iopt == 3 ) { /* Free storage */
158     /* Free the LU factors in the factors handle */
159     LUfactors = (factors_t*) *f_factors;
160     SUPERLU_FREE (LUfactors->perm_r);
161     SUPERLU_FREE (LUfactors->perm_c);
162     Destroy_SuperNode_Matrix(LUfactors->L);
163     Destroy_CompCol_Matrix(LUfactors->U);
164          SUPERLU_FREE (LUfactors->L);
165          SUPERLU_FREE (LUfactors->U);
166     SUPERLU_FREE (LUfactors);
167      } else {
168     fprintf(stderr,"Invalid iopt=%d passed to c_ecosimpro_superlu()\n",iopt);
169     exit(-1);
170      }
171 }
```

La declaración en lenguaje EcosimPro de la función `d_ecosimpro_superlu` se muestra a continuación en el Listado E-2.

Listado E-2: Declaración en Lenguaje EcosimPro de la Función `d_ecosim_superlu`

```
/* ============================================================================
* FUNCTION: d ecosim superlu
* This function is a wrapper for the SuperLU solver in EcosimPro Language.
* ============================================================================
*
* iopt              = int specifies the operation:
*                       1, performs LU decomposition for the first time
*                       2, performs triangular solve
*                       3, free all the storage in the end
* n                 = dimension of the square sparse matrix
* nnz               = number of nonzeros in the sparse matrix
* nrhs              = number of right-hand sides
* val               = double array containing the nonzero entries
* colind            = int array containing the column indices of the entries
* rowptr            = int array containing the row start
* b                 = double array containing the right-hand side vector (gets
                      overwritten with solution)
* factors           = pointer to LU factors. (If iopt == 1, it is an output and
*                       contains the pointer pointing to the structure of
*                       the factored matrices. Otherwise, it is an input.)
* info              = info flag from SuperLU
*                   = 0: successful exit
*                   < 0: if info = -i, the i-th argument had an illegal value
*                   > 0: if info = i, and i is
*                   <= A->ncol: U(i,i) is exactly zero. The factorization has
*                       been completed, but the factor U is exactly
```

```
*                       singular, so the solution and error bounds
*                       could not be computed.
*                  = A->ncol+1: U is nonsingular, but RCOND is less than machine
*                       precision, meaning that the matrix is singular to
*                       working precision. Nevertheless, the solution and
*                       error bounds are computed because there are a number
*                       of situations where the computed solution can be more
*                       accurate than the value of RCOND would suggest.
*                  > A->ncol+1: number of bytes allocated when memory allocation
*                       failure occurred, plus A->ncol.
* pivot_threshold  = pivot threshold
* equil            = 1/0 equilibrate /no equilibrate matrix (not implemented yet)
* refine           = 1/0 refine solution/ no refine solution (not implemented yet)
* debug            = 1/0 for debug&performance info/no debug&performance info
* ============================================================================   */
"C" FUNCTION  NO_TYPE d_ecosim_superlu(
   IN INTEGER iopt          "integer that specifies the operation to be performed",
   IN INTEGER n             "dimension of the square sparse matrix",
   IN INTEGER nnz           "number of non-zeros entries in the sparse matrix",
   IN INTEGER nrhs          "number of righ hand side terms",
   IN REAL val[]            "values of the non zero items in row compressed order",
   IN INTEGER colwind[]     "column indexes in row compressed format",
   IN INTEGER rowptr[]      "row pointers in row compressed format",
   OUT REAL b[]             "double array containing the right hand side",
   OUT INTEGER factors[4]   "handle pointing to the LU factors",
   OUT INTEGER info         "return flag from superlu, if =0 successful exit",
   IN REAL pivot_threshold "pivot threshold",
   IN INTEGER equil         "1/0 equilibrate matrix/no equilibrate matrix \
                             (not implemented yet)",
   IN INTEGER refine        "1/0 refine solution/ no refine solution
                             (not implemented yet)",
   IN INTEGER debug         "1/0 for debug&performance info/no debug&performance info"
   ) IN  "libsuperlu_5.1.a"
```

E.2. Clase SPARSE_MATRIX

La clase `SPARSE_MATRIX` se emplea para manipular matrices dispersas, las cuales se caracteriza, caracterizadas por la predominancia de elementos nulos. Diseñada como una utilidad auxiliar para resolver sistemas de ecuaciones no lineales dispersos, esta clase se distingue por su eficaz generación de Jacobianos mediante estampación y su capacidad para calcular de manera eficiente el paso o corrección en métodos del tipo Newton.

Esta clase proporciona métodos específicos para llevar a cabo diversas operaciones, entre las que se incluyen:

- Rellenar elementos no nulos mediante estampación.
- Construir la representación de la matriz en formato comprimido por filas (CSR, compressed sparse row format) a partir de las estampaciones.
- Resolver sistemas de ecuaciones lineales.
- Realizar distintas operaciones con la matriz, tales como:
    - Establecer a cero todos los elementos de la matriz.
    - Obtener un vector con el valor absoluto máximo de cada fila.
    - Pre-multiplicar la matriz por un vector de fila.

- Post-multiplicar la matriz por un vector de columna.
- Imprimir la matriz.

En el proceso de estampación, se parte de una matriz nula, y se añaden contribuciones en posiciones específicas, utilizando un método de la clase denominado **`stampItem`**. Este método recibe como argumentos el número de fila, el número de columna y el valor a estampar. Por ejemplo, las estampaciones que se indican a continuación:

```
stampItem(1, 1, 8.)
stampItem(2, 2, 1.)
stampItem(2, 2, 2.)
stampItem(3, 3, 3.)
stampItem(4, 4, 3.)
stampItem(5, 5, 2.)
stampItem(5, 5,-6.)
stampItem(1, 5, 1.)
stampItem(2, 4, 4.)
stampItem(2, 4, 1.)
stampItem(3, 1, 1.)
stampItem(4, 2, 1.)
stampItem(5, 1, 3.)
```

generarían la matriz $\begin{bmatrix} 8 & 0 & 0 & 0 & 1 \\ 0 & 1+2 & 0 & 4+1 & 0 \\ 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 \\ 3 & 0 & 1 & 0 & 2-6 \end{bmatrix}$

Un elemento de la matriz puede recibir cero, una o varias contribuciones. Durante el proceso de estampación, se generan cuatro vectores unidimensionales: `stampRow`, `stampCol`, `stampValue` y `stampIpos`, que registran las estampaciones realizadas.

Los vectores `stampRow[i]`, `stampCol[i]`, `stampValue[i]` almacenan respectivamente la fila, la columna y el valor añadido por la i-ésima estampación. Adicionalmente, el vector `stampIpos` se inicializa con el número de la estampación, es decir, `stampIpos[i] = i`.

Después de lleva a cabo las estampaciones según el ejemplo proporcionado, los vectores resultantes son los siguientes.

```
stampIrow  = {  1,  2,  2,  3,  4,  5,  5,  1,  2,  2,  3,  4,  5,  5}
stampIcol  = {  1,  2,  2,  3,  4,  5,  5,  5,  4,  4,  1,  2,  1,  3}
stampIpos  = {  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14}
stampValue = { 8., 1., 2., 3., 3., 2.,-6., 1., 4., 1., 1., 1., 3., 1.}
```

Antes de poder utilizar la librería SuperLU para la resolución de sistemas de ecuaciones lineales dispersas, es preciso organizar la matriz dispersa en formato comprimido por filas (CSR) o en formato comprimido columnas (CSC). El formato CSR almacena una matriz dispersa de tamaño `nrow × ncol` utilizando tres vectores unidimensionales: `csrValue`, `csrColInd`, `csrRowPtr`.

El vector `csrValue` contiene los valores no nulos de la matriz, mientras que `csrColInd` almacena los índices de las columnas correspondientes a estos valores. La longitud de estos dos vectores es igual al número de elementos no nulos de la matriz, valor designado como `nnzeros`.

El vector `csrRowPtr` tiene una longitud de `nrow +1` y almacena el índice o posición en `csrValue` y `csrColInd` donde comienza cada fila de la matriz. Esto se traduce en que `csrRowPtr[j]` es igual a la cantidad total de elementos no nulos por encima de la fila `j` más uno. El último elemento en `csrRowPtr`, esto es `csrRowPtr[nrow+1]` es igual a `nnzeros+1` y corresponde a un índice ficticio en `csrValue`, inmediatamente después del último índice válido de `nnzeros`.

Por ejemplo, la matriz del ejemplo anterior:

$$\begin{bmatrix} 8 & 0 & 0 & 0 & 1 \\ 0 & 3 & 0 & 5 & 0 \\ 1 & 0 & 3 & 0 & 0 \\ 0 & 1 & 0 & 3 & 0 \\ 3 & 0 & 1 & 0 & -4 \end{bmatrix}$$

es una matriz 5x5 con 10 elementos nulos, y la representación de la misma en formato CSR es:

```
 csrValue = {8., 1., 3., 5., 1., 3., 1., 3., 3., 1.,-4.}

csrColInd = { 1,  5,  2,  4,  1,  3,  2,  4,  1,  3,  5}

csrRowPtr = { 1,  3,  5,  7,  9, 11}
```

A fin de facilitar la obtención de la representación de la matriz en formato CSR, sería posible obligar a que las estampaciones tuviesen que llegar en orden lexícográfico por filas y columnas. Ahora bien, esto restaría flexibilidad a los posibles usos de la clase. Ha sido decisión de diseño permitir que las estampaciones puedan producirse en cualquier orden. Para abordar esta decisión de diseño, se ha implementado en la clase un método específico denominado **`buildCSR`**. Este método, primeramente, se encarga de ordenar las estampaciones aplicando el algoritmo Quicksort y obtener la representación de la misma en formato CSR. Durante el proceso de ordenamiento, los elementos se organizan según los números de fila; y en caso de empate, se utiliza el número de columna para establecer el orden.

```
stampIrow  = { 1,  1,  2,  2,   ,  2,  3,  3,  4,  4,  5,  5,  5,  5},

stampIcol  = { 1,  5,  2,  2,  4,  4,  1,  3,  2,  4,  1,  3,  5,  5},

stampIpos  = { 1,  8,  2,  3, 10,  9,  1,  4, 12,  5, 13, 14,  6,  7}

stampValue = {8., 1., 1., 2., 1., 4., 1., 3., 1., 3., 3., 1., 2.,-6.}
```

Una vez ordenadas las estampaciones, el algoritmo presentado en el Listado E-3 facilita la obtención de los vectores que definen la matriz en formato CSR. Este algoritmo realiza las siguientes acciones:

- Inicia el contador de elementos no nulos (nnzeros) en cero.
- Recorre las estampaciones, fusionando contribuciones repetidas en una misma posición de la matriz,
- Determina los índices y valores de las columnas no nulas, y Actualiza los punteros de fila.

Listado E-3: Algoritmo para Generar el Formato CSR de la Matriz a partir de Estampaciones Ordenadas

```
FOR (i IN 1, nstamp)
--It detects if the position of the actual stamp is equal to the position
--of the previous stamp
   IF(i > 1 AND (stampIrow[i] == stampIrow[i-1]) AND \
                    (stampIcol[i] == stampIcol[i-1])) THEN
      csrValue[nnzeros] = csrValue[nnzeros] + stampValue[i]
   ELSE
      nnzeros = nnzeros + 1
      csrColInd[nnzeros] = stampIcol[i]
      csrValue[nnzeros] = stampValue[i]
      IF(i == 1) THEN
         i1 = 1
         i2 = stampIrow[i]
         ASSERT(i1 == i2) ERROR "Empty row in sparse matrix"
         FOR(k IN i1, i2)
             csrRowPtr[k]= nnzeros
         END FOR
      ELSEIF(stampIrow[i] != stampIrow[i-1]) THEN
         i1 = stampIrow[i-1]+1
         i2 = stampIrow[i]
         ASSERT(i1 == i2) ERROR "Empty row in sparse matrix"
         FOR(k IN i1, i2)
            csrRowPtr[k]= nnzeros
         END FOR
      END IF
   END IF
   k = stampIpos[i]
   stampCsrIpos[k] = nnzeros
END FOR
csrRowPtr[nrow+1] = nnzeros + 1
isCSRBuilt = TRUE
```

El algoritmo garantiza la correcta organización de la matriz en vectores CSR (`csrValue`, `csrColInd`, `csrRowPtr`). Además, garantiza una gestión efectiva de filas vacías y verifica la coherencia del proceso.

Es frecuente que, al generar un Jacobiano mediante estampaciones, estas mantengan el mismo orden en sucesivas iteraciones del método Newton-Raphson. Con el propósito de mejorar la eficiencia en la generación de Jacobianos subsiguientes al inicial, el algoritmo utiliza el vector `stampToCSR`. Este vector registra la posición específica en la matriz CSR

(csrValue) a la que contribuye cada estampación no ordenada. Al mismo tiempo, se marca como TRUE el atributo isCsrBuilt.

En consecuencia, si después de haber construido inicialmente el formato CSR, se regenerar la matriz mediante estampaciones (lo cual implica su previa re-inicialización a cero), se asume que cada estampación contribuirá al elemento de csrValue indicado por stampToCSR. Ahora bien, se comprueba que tanto la fila como la columna de la estampación coincidan con la fila y columna del elemento señalado por stampToCSR en el formato CSR. En caso de falta de coincidencia, se infiere que las estampaciones han cambiado su orden. Para abordar este escenario, se establece el atributo isCsrBuilt como FALSE, y se procede de manera análoga al caso de la primera estampación: se ordenan las estampaciones y se aplica el algoritmo descrito en el Listado E-3.

E.2.1. Parámetros de Construcción

Nombre	Tipo	Descripción
MAX_NROW	INTEGER	Maximum number of rows and columns"
MAX_STAMP	INTEGER	Maximum number of stamps to build the Jacobian matrix

E.2.2. Atributos

La clase SPARSE_MATRIX consta de los siguientes atributos:

Nombre	Tipo	Defecto	Descripción
isCSRBuilt	BOOLEAN	FALSE	Boolean indicating if the sparse matrix has been built in CSR format or not
isSetToZero	BOOLEAN	TRUE	Boolean indicating if the sparse matrix has been reset to zero or not
nstamp	INTEGER	0	Total number of stamps
nrow	INTEGER	0	Total number of rows
ncol	INTEGER	0	Total number of columns
stampIrow[MAX_STAMP]	INTEGER[]	0	Number of row of the stamp
stampIcol[MAX_STAMP]	INTEGER[]	0	Number of column of the stamp
stampIpos[MAX_STAMP]	INTEGER	0	Position of the stamp before sorting
stampValue[MAX_STAMP]	INTEGER[]	0	Real value of the stamp
stampToCSR[MAX_STAMP]	INTEGER[]	0	Mapping of stamp contributions in CSR format
STAMP_debug	INTEGER	0	Debug level of the stamping process
nnzeros	INTEGER	0	Total number of non zero items
csrColInd[MAX_STAMP]	INTEGER[]	0	Columns of the matrix items
csrValue[MAX_STAMP]	INTEGER[]	0	Columns of the matrix items
csrRowPptr[MAX_ROW+1]	INTEGER[]	0	Pointers to the beginning of each row

E.2.3. Métodos

1. SetToZero()

- Descripción: Este método establece que todos los elementos de la matriz dispersa son cero. Es necesario llamar a este método siempre que se desee volver a generar la matriz por estampación.
- Tipo Retornado: NO_TYPE

2. stampItem(irow, icol, value)

- Descripción: Este método añade un valor a cualquier elemento de la matriz dispersa en cualquier orden.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción
`irow`	IN	INTEGER	Row index of the stamp
`icol`	IN	INTEGER	Column index of the stamp
`value`	IN	INTEGER	Additive value contribution of the stamp

- Tipo Retornado: NO_TYPE

3. buildCSR()

- Descripción: Este método se llama una vez rellenados los elementos de la matriz con el método `stampItem()`. Rellena las estructuras de datos que definen la matriz en formato comprimido por filas o CSR.

  Este método, primero, ordena las estampaciones en el orden requerido por el formato CSR utilizando el método de ordenación Quicksort. Posteriormente, genera la matriz en formato CSR a partir de las estampaciones ordenadas.

  En el caso de que la matriz se restableza a cero con el método `setToZero()` y se vuelva a rellenar por estampación. Se comprueba de forma automática si las estampaciones se producen en el mismo orden que en el rellenado previa. En caso afirmativo, se utiliza el mapeo generado previamente entre las estampaciones y los elementos no nulos en formato CSR. Si difiere, se ordenan las estampaciones, y se construye la matriz en formato CSR.
- Tipo Retornado: NO_TYPE

4. LinearEqnSolve(b, x, pivot_threshold, debug)

- Descripción: Este método resuelve un sistema de ecuaciones lineales utilizando la factorización LU de la matriz dispersa.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción

b[]	IN	REAL	Vector de términos constantes
x[]	OUT	REAL	Vector de solución
pivot_threshold	IN	REAL	Pivot threshold
debug	IN	INTEGER	0/1 Debug level for SuperLU

- Tipo Retornado: NO_TYPE

5. MaxAbsRow(rmax)

- Descripción: Este método encuentra el valor máximo absoluto en cada fila de la matriz dispersa y lo almacena en un vector.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción
rmax[]	OUT	REAL	Vector with the absolute maximum of each row

- Tipo Retornado: NO_TYPE

6. postMultiplyByColumn(X, Y)

- Descripción: Este método post-multiplica la matriz dispersa por un vector columna y devuelve el resultado en otro vector.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción
X[]	IN	REAL	Input column vector
Y[]	OUT	REAL	Results vector

- Tipo Retornado: NO_TYPE

7. preMultiplyByRow(X, Y)

- Descripción: Este método pre- multiplicación de fila de la matriz dispersa por un vector de entrada y devuelve el resultado en otro vector.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción
X[]	IN	REAL	Input row vector
Y[]	OUT	REAL	Results vector

- Tipo Retornado: NO_TYPE

8. Print()

- Descripción: Este método imprime información sobre la matriz dispersa, incluyendo el número de elementos no cero, el número de filas, el número de columnas y detalles de la estructura de la matriz.

- Tipo Retornado: NO_TYPE

E.2.4. Ejemplo de Uso de la Clase SPARSE_MATRIX

Listado E-4: Ejemplo de Uso de la Clase `Sparse_Matrix`

```
--Example of Usage of the  CLASS SPARSE MATRIX
USE SPARSE
COMPONENT TestSparseMatrix01
   DATA
      REAL b[5] = { 10.0,  0.25,  4.50, -0.25, 13.5}
      --independent terms of the linear equation system
      REAL c[5] = {  3.0,  0.50, -0.50,  2.50, -1.0}
      --1D array
   DECLS
      DISCR REAL x[5] --Solution of the linear equation system
      DISCR REAL m[5] --Maximum absolute values per row
      DISCR REAL d[5] --Result of postmultiplication
      DISCR REAL e[5] --Result of premultiplication
      DISCR REAL error
   OBJECTS
      --Declaration of a sparse matrix with up to 5 rows
      --and up to 20 possible stampings
      SPARSE_MATRIX(MAX_ROW=10, MAX_STAMP=20)  A
   INIT
      A.setToZero()
      -- Let the matrix be 5x5 with the values shown below:
      -- A = [
      --         8,   0,   0,   0,   1;
      --         0, 1+2,   0, 4+1,   0;
      --         1,   0,   3,   0,   0;
      --         0,   1,   0,   3,   0;
      --         3,   0,   1,   0, 2-6;
      --     ];
      --Fill the matrix by stamping as follows:
      A.stampItem(1, 1, 8.)
      A.stampItem(2, 2, 1.)
      A.stampItem(2, 2, 2.)
      A.stampItem(3, 3, 3.)
      A.stampItem(4, 4, 3.)
      A.stampItem(5, 5, 2.)
      A.stampItem(5, 5,-6.)
      A.stampItem(1, 5, 1.)
      A.stampItem(2, 4, 4.)
      A.stampItem(2, 4, 1.)
      A.stampItem(3, 1, 1.)
      A.stampItem(4, 2, 1.)
      A.stampItem(5, 1, 3.)
      A.stampItem(5, 3, 1.)
      --
      --After stamping
      --nstamps    = 14
      --stampIrow  = {  1,  2,  2,  3,  4,  5,  5,  1,  2,  2,  3,  4,  5,  5}
      --stampIcol  = {  1,  2,  2,  3,  4,  5,  5,  5,  4,  4,  1,  2,  1,  3}
      --stampIpos  = {  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14}
      --stampValue = { 8., 1., 2., 3., 3., 2.,-6., 1., 4., 1., 1., 1., 3., 1.}
      A.Print()
      ------------------------------------------------------------
      --Construcción de la matriz en formato CSR
      A.buildCSR()
      --After this method, the stamps are sorted
      --stampIrow    = { 1,  1,  2,  2,  2,  2,  3,  3,  4,  4,  5,  5,  5,  5},
      --stampIcol    = { 1,  5,  2,  2,  4,  4,  1,  3,  2,  4,  1,  3,  5,  5},
      --stampIpos    = { 1,  8, 2,   3, 10,  9, 11,  4, 12,  5, 13,  4,  6,  7}
      --stampValue   = {8., 1., 1., 2., 1., 4., 1., 3., 1., 3., 3., 1., 2.,-6.}
      --The CSR format is built
      --nnzeros = 11
      --csrColInd = {1,   5,  2,  4,  1,  3,  2,  4,  1,  3,  5}
      --csrValue  = {8., 1., 3., 5., 1., 3., 1., 3., 3., 1.,-4.}
      --csrRowPtr = {1,   3,  5,  7,  9, 12}
```

```
      --The following array indicates the destination position in CSR
      --format of the unsorted stamps
      --stampCsrIpos = {  1, 3, 3, 6, 8, 11, 11, 2, 4, 4, 5, 7, 9, 10}
      A.Print()
      ------------------------------------------------------------
      --Solving the system of linear equations A·x = b
      --where the vector of independent terms
      ------------------------------------------------------------
      --b = [
      --      10.0;
      --      0.25;
      --      4.50;
      --     -0.25;
      --      13.5;
      --    ]
      A.LinearEqnSolve(b, x, 1., 0)
      WRITE("\n\nSolution:\n")
      WRITE("    x[5]   = { %lg , %lg , %lg , %lg , %lg }\n",
                           x[1], x[2], x[3], x[4], x[5])
      ------------------------------------------------------------
      --Find the maximum absolute value of each row
      A.MaxAbsRow(m)
      WRITE("\n\nMax.Abs.Value of each row:\n")
      WRITE("    m[5]   = { %lg , %lg , %lg , %lg , %lg }\n",
                           m[1], m[2], m[3], m[4], m[5])
      ------------------------------------------------------------
      --Pre-multiplication by a row vector
      A.preMultiplyByRow(c, e)
      WRITE("\n\nResult of Premultiplicacion:\n")
      WRITE("    e[5]   = { %lg , %lg , %lg , %lg , %lg }\n",
                           e[1], e[2], e[3], e[4], e[5])
      ------------------------------------------------------------
      ---Postmulitplicacion by column vector
      A.postMultiplyByColumn(c, d)
      WRITE("\n\nResult of Post-multiplicacion:\n")
      WRITE("    d[5]   = { %lg , %lg , %lg , %lg , %lg }\n",

                           d[1], d[2], d[3], d[4], d[5])
```

E.3. CLASE NLEQN_SYSTEM

Es una clase para resolver sistemas de ecuaciones algebraicas no lineales dispersos por métodos del tipo Newton-Raphson (NR).

Es especialmente adecuada para simulaciones a gran escala, permitiendo abordar sistemas de ecuaciones dispersas de gran tamaño, con miles de ecuaciones.

El método más importante que ofrece esta clase es el método `Solve`, un método para resolver sistemas de ecuaciones no lineales utilizando el algoritmo de Newton-Raphson. Dicho método requiere como argumento principal un puntero a una función que define el sistema de ecuaciones.

El Listado E-5 proporciona declaración del tipo de función al que debe apuntar dicho puntero.

Listado E-5: Declaración del Puntero a Función `func_Fsystem`

```
TYPEDEF FUNCTION NO_TYPE func_Fsystem(
/*=========================================================================
--Declaration of the function pointer to calculate the residues and
```

```
--the Jacobian from the unknowns and the data
=============================================================================*/
      IN INTEGER iopt      "int that specifies the operation",
      IN REAL X[]          "vector of unknowns",
      IN REAL RDATA[]      "Auxiliary vector for communication",
      OUT REAL F[]         "Residues",
      OUT NLEQN_SYSTEM_BASE nleqn_system  "Base of the non linear equation system"
      )
/*
* iopt             = Integer specifying the operation:
*                      1, to set the number of equations, create unknowns and equations
*                      2, to calculate the residuals
*                      3, to calculate residuals and functions
* X[]              = Real array containing a vector of unknowns
* RDATA            = Real parameter arrays used for communication
*                    with the calling and current function
* nleqn_system     = Object of type NLEQN_SYSTEM_BASE, a parent class
*                    for NLEQN_SYSTEM. NLEQN_SYSTEM_BASE includes all attributes
*                    and methods of NLEQN SYSTEM except the SOLVE method.
*                    The SOLVE method calls this function using the "this"
*                    pointer, allowing the function to access the class to define
*                    the number of equations, create unknowns and equations,
*                    and fill the Jacobian by stamping
*/
```

Uno de los argumentos de la función `func_Fsystem` es un objeto de la clase padre `NLEQN_SYSTEM_BASE`. Dicha clase incluye todos los atributos y métodos de `NLEQN_SYSTEM` excepto `Solve`. El método `Solve` llama a esta función usando el puntero "`this`", para así permitir que la función tenga acceso a la clase para definir el número de ecuaciones, crear incógnitas y ecuaciones, y rellenar el Jacobiano por estampación.

La clase NLEQN_SYSTEM también proporciona métodos para:

- Crear variables desconocidas, lo cual implica proporcionar un valor inicial para la variable, establecer un rango de valores (tanto máximo como mínimo) que la variable no puede exceder, proporcionar una descripción de la función de la variable y, finalmente, fijar el atributo `freezeIterUnk`. Este atributo le indica al resolvedor que la variable permanecerá inalterada durante las primeras `freezeIterUnk` iteraciones.
- Rellenar el Jacobiano por estampación.

E.3.1. Parámetros de Construcción

Nombre	Tipo	Descripción
MAX_EQN	INTEGER	Maximum number of equations
MAX_STAMP	INTEGER	Maximum number of stampings to fill the Jacobian

E.3.2. Atributos

La clase NLEQN_SYSTEM consta de los siguientes atributos:

Nombre	Tipo	Defecto	Descripción
`neqn`	INTEGER	0	Number of equations
`iter`	INTEGER	0	Iteration number of the NR

Nombre	Tipo	Defecto	Descripción
NR_debug	INTEGER	0	Debug level of the Newton Raphson - range 0/1/2
STAMP_debug	INTEGER	0	Debug level of the stamping process - range 0/1
superLU_debug	INTEGER		Debug level of the superLU solver - range 0/1
Fun[MAX_EQN]	REAL[]	0	Vector of new residues
FunOld[MAX_EQN]	REAL[]	FALSE	Vector of old residues
Xunk[MAX_EQN]	REAL[]	0	Vector of new unknown values
XunkOld[MAX_EQN]	REAL[]	-1	Vector of old values of the unknowns
XunkMax[MAX_EQN]	REAL[]	0	Vector of maximum limits for the unknowns
XunkMin[MAX_EQN]	REAL[]	0	Vector of minimum limits for the unknowns
freezeIterUnk[MAX_EQN]	INTEGER[]	0	Number of freezed iterations for the unknowns
descrUnk[MAX_EQN]	STRING[]	0	String with a description of the unknowns
descrEqn[MAX_EQN]	STRING[]	0	String with a description of the equations
EqnType[MAX_EQN]	INTEGER[]	0	Equation type according to some conventions
pivot_threshold	REAL	1.	Threshold used for a diagonal entry to be an acceptable pivot in SuperLU
tol	REAL	1.e-7	Tolerance of the NR
converged	BOOLEAN	FALSE	Boolean, if TRUE, the solver has converged
Jacob (MAX_ROW = MAX_EQN, MAX_STAMP=MAX_STAMP)	SPARSE_MATRIX		Jacobian matrix

E.3.3. Métodos:

1. Solve(fsys, iter_max, iter_damp, FRAC_damp, tol_arg, rdata)

- Descripción: Resuelve un sistema de ecuaciones utilizando el método de Newton-Raphson.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción
fsys	IN	FUNC_PTR <func_Fsystem>	Pointer to the function with the equations
Iter_max	IN	REAL	Maximum number of iterations
Iter_damp	IN	REAL	Number of initial damped iterations
FRAC_damp	IN	INTEGER	Multiplier of the NR step during the first iter_damp iterations

`tol_arg`	IN	REAL	Tolerance of the NR
`rdata[]`	IN	REAL[]	Auxiliary vector for communication

- Tipo Retornado: NO_TYPE

- Algoritmo: El método `Solve` utiliza el método de Newton-Raphson para resolver el sistema de ecuaciones

2. CreateUnknown(i, Xo, Xmin, Xmax, unkn_descrip, unkn_type)

- Descripción: Esteo método permite proporcionar los siguientes datos para la i-ésima variable deconocida:
  - Valor inicial.
  - Rango de valores, tanto máximo como mínimo.
  - Descripción de la función de la variable.
  - Fijar el atributo `freezeIterUnk` de la variable. Este atributo le indica al método `Solve` que la variable permanecerá inalterada durante las primeras `freezeIterUnk` iteraciones.

- Argumentos:

Nombre	Causalidad	Tipo	Descripción
`i`	IN	INTEGER	Number of the unknown variable
`Xo`	IN	REAL	Initial value of the unknown
`Xmin`	IN	REAL	Minimum allowable value for the unknown
`Xmax`	IN	REAL	Maximum allowable value for the unknown
`unk_descrip`	IN	STRING	Description of the unknown variable
`freezeIter`	IN	INTEGER	Number of iterations to freeze the unknown

- Tipo Retornado: NO_TYPE

- Uso del argumento `freezeIter`: El argumento `freezeIter` se asigna al atributo `freezeIterUnk[i]` de la i-ésima variable desconocida.

  La capacidad de congelar una variable resulta útil para facilitar la convergencia en ciertos tipos de simulaciones con ecuaciones fuertemente desacopladas. Un ejemplo destacado es la simulación del flujo de líquidos en redes de tuberías, donde las ecuaciones de conservación de masa y momento están desacopladas de la ecuación de conservación de energía.

En este tipo de casos, es preferible que las primeras iteraciones modifiquen únicamente las presiones y flujos, manteniendo las temperaturas congeladas. La ecuación de energía se aplica una vez que se han obtenido presiones y flujos más consistentes, evitando oscilaciones innecesarias de la temperatura. Este enfoque se puede lograr estableciendo un valor adecuado para el atributo `freezeIterUnk` asociado a las temperaturas, generalmente en el rango de 5 a 10.

Al fijar este atributo con un valor específico, se indica al método Solve que la variable de temperatura permanecerá inalterada durante las primeras iteraciones, permitiendo que las otras variables ajusten sus valores de manera más efectiva. Esta estrategia contribuye a un proceso de convergencia más eficiente y estable en situaciones, en situaciones en las que se pueden identificar ecuaciones desacopladas.

3. CreateEqn(i, eqn_descrip)

- Descripción: Asocia una descripción a la i-ésima ecuación.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción
i	IN	INTEGER	Number of the equation
eqn_descrip	IN	STRING	Description of the Equation

- Tipo Retornado: NO_TYPE
- **CreateEqn(i, eqn_descrip)**

4. Jacob.stamp(ieq, iunk, value)

- Descripción: Este método es en realidad un método de la Clase SPARSE_MATRIX, pero como uno de los atributos de la clase NLEQN_SYSTEM es el objeto Jacob de la clase SPARSE_MATRIX, el método stamp puede ser usado sobre dicho objeto. Ahora bien, el número de fila de corresponde con el número de ecuación y el número de columna coincide con el número de la desconocida.

- Argumentos:

Nombre	Causalidad	Tipo	Descripción
ieq	IN	INTEGER	Equation number
iunk	IN	INTEGER	Unknown number
value	IN	INTEGER	Additive value contribution of the Jacobian term

- Tipo Retornado: NO_TYPE

5. GetUnknown(X)

- Descripción: Obtiene los valores del vector de incógnitas.
- Argumentos:

Nombre	Causalidad	Tipo	Descripción
X[]	OUT	REAL	Unknowns vector

- Tipo Retornado: NO_TYPE

E.3.4. Ejemplo de Uso de la Clase NLEQN_SYSTEM

Listado E-6: Ejemplo de Uso de la Clase NLEQN_SYSTEM

```
--Example of Usage of the  CLASS NLEQN_SYSTEM:
USE SPARSE
FUNCTION NO_TYPE fsystem(IN INTEGER iopt, IN REAL x[], IN REAL rdata[], OUT REAL F[],
OUT NLEQN_SYSTEM_BASE nleqn_system)
   DECLS
   BODY
      nleqn_system.neqn = 4
      IF(iopt == 0) THEN
            FOR(j IN 1, nleqn_system.neqn)
               nleqn_system.CreateUnk(j, 0.1, -1e6, 1e6, \
                     concatStrings(concatStrings("x[", integerToString(j)),"]"),0)
            END FOR
            nleqn_system.CreateEqn(1,"-x[1]**2 - x[2]**2 - x[3]**2 + x[4]    = 0")
            nleqn_system.CreateEqn(2," x[1]**2 + x[2]**2 + x[3]**2 + x[4]**2 = 0")
            nleqn_system.CreateEqn(3," x[1] - x[2] = 0")
            nleqn_system.CreateEqn(4," x[2] - x[3] = 0")
      ELSEIF(iopt == 1 OR iopt == 2) THEN
         nleqn_system.Jacob.setToZero()
         F[1] =  - x[1]**2 - x[2]**2 - x[3]**2 + x[4]
         F[2] =    x[1]**2 + x[2]**2 + x[3]**2 + x[4]**2 - 1
         F[3] =    x[1] - x[2]
         F[4] =    x[2] - x[3]
         IF(iopt == 2) THEN
            nleqn_system.Jacob.setToZero()
            nleqn_system.Jacob.stampItem(1, 1, -2.*x[1])
            nleqn_system.Jacob.stampItem(1, 2, -2.*x[2])
            nleqn_system.Jacob.stampItem(1, 3, -2.*x[3])
            nleqn_system.Jacob.stampItem(1, 4, 1.0)
            nleqn_system.Jacob.stampItem(2, 1, 2.*x[1])
            nleqn_system.Jacob.stampItem(2, 2, 2.*x[2])
            nleqn_system.Jacob.stampItem(2, 3, 2.*x[3])
            nleqn_system.Jacob.stampItem(2, 4, 2.*x[4])
            nleqn_system.Jacob.stampItem(3, 1, 1.0)
            nleqn_system.Jacob.stampItem(3, 2, -1.0)
            nleqn_system.Jacob.stampItem(4, 2, 1.0)
            nleqn_system.Jacob.stampItem(4, 3, -1.0)
            nleqn_system.Jacob.buildCSR()
         END IF
      END IF
      RETURN
END FUNCTION
COMPONENT TestNleqnSystem01
   DATA
      INTEGER iter_damp = 2  "Number of initial damped iterations"
      INTEGER iter_max = 20  "Maximum number of iterations"
      REAL FRAC_damp = 0.1
      REAL tol = 1e-6        "Tolerance"
      INTEGER NR_debug = 2    "Debugging level of the Newton-Raphson"
      INTEGER STAMP_debug = 0
   DECLS
      DISCR REAL FRAC = 1
      DISCR REAL errorx
      DISCR REAL errorf
      DISCR REAL x[20]
      DISCR REAL rdata[10]
   OBJECTS
      NLEQN_SYSTEM(MAX_EQN=20, MAX_STAMP = 100) nleqn_system
   INIT
      nleqn_system.NR_debug = NR_debug
      nleqn_system.Jacob.STAMP_debug = STAMP_debug
      nleqn_system.Solve(fsystem, iter_max, iter_damp, FRAC_damp, tol, rdata)
      nleqn_system.GetUnknown(x)
END COMPONENT
```

E.4. Listado Completo del Código de la Librería SPARSE

```
/*-------------------------------------------------------------------------------------------
 LIBRARY: SPARSE
 FILE: sparse.el
 CREATION DATE: 30/01/2020
-------------------------------------------------------------------------------------------*/
USE MATH
"C" FUNCTION  NO_TYPE d_ecosim_superlu(
/* ==========================================================================
* FUNCTION: d_ecosim_superlu
* This function is a wrapper for the SuperLU solver in EcosimPro Language.
* ==========================================================================
*
* iopt             = int specifies the operation:
*                      1, performs LU decomposition for the first time
*                      2, performs triangular solve
*                      3, free all the storage in the end
* n                = dimension of the square sparse matrix
* nnz              = number of nonzeros in the sparse matrix
* nrhs             = number of right-hand sides
* val              = double array containing the nonzero entries
* colind           = int array containing the column indices of the entries
* rowptr           = int array containing the row start
* b                = double array containing the right-hand side vector (gets overwritten with
solution)
* factors          = pointer to LU factors. (If iopt == 1, it is an output and contains the
pointer pointing
*                    to the structure of the factored matrices. Otherwise, it is an input.)
* info             = info flag from SuperLU
*                  = 0: successful exit
*                  < 0: if info = -i, the i-th argument had an illegal value
*                  > 0: if info = i, and i is
*                  <= A->ncol: U(i,i) is exactly zero. The factorization has
*                       been completed, but the factor U is exactly
*                       singular, so the solution and error bounds
*                       could not be computed.
*                  = A->ncol+1: U is nonsingular, but RCOND is less than machine
*                       precision, meaning that the matrix is singular to
*                       working precision. Nevertheless, the solution and
*                       error bounds are computed because there are a number
*                       of situations where the computed solution can be more
*                       accurate than the value of RCOND would suggest.
*                  > A->ncol+1: number of bytes allocated when memory allocation
*                       failure occurred, plus A->ncol.
* pivot_threshold  = pivot threshold
* equil            = 1/0 equilibrate matrix/no equilibrate matrix (not implemented yet)
* refine           = 1/0 refine solution/ no refine solution (not implemented yet)
* debug            = 1/0 for debug&performance info/no debug&performance info
* ========================================================================   */
   IN INTEGER iopt          "integer that specifies the operation to be performed",
   IN INTEGER n             "dimension of the square sparse matrix",
   IN INTEGER nnz           "number of non-zeros entries in the sparse matrix",
   IN INTEGER nrhs          "number of righ hand side terms",
   IN REAL val[]            "values of the non zero items in row compressed order",
   IN INTEGER colwind[]     "column indexes in row compressed format",
   IN INTEGER rowptr[]      "row pointers in row compressed format",
   OUT REAL b[]             "double array containing the right hand side",
   OUT INTEGER factors[4]   "handle pointing to the LU factors",
   OUT INTEGER info         "return flag from superlu, if =0 successful exit",
   IN REAL pivot_threshold "pivot threshold",
   IN INTEGER equil         "1/0 equilibrate matrix/no equilibrate matrix (not implemented yet)",
   IN INTEGER refine        "1/0 refine solution/ no refine solution (not implemented yet)",
   IN INTEGER debug         "1/0 for debug&performance info/no debug&performance info"

   ) IN  "libsuperlu_5.1.a"


CLASS SPARSE_MATRIX(
      INTEGER MAX_ROW      "Maximum number of rows and columns",
      INTEGER MAX_STAMP    "Maximum number of stamps to build the Jacobian matrix"
      )
/* ==========================================================================
* CLASS: SPARSE_MATRIX
* Purpose: Filling non-zero items by stampation,
*          Building the CSR representation of the matrix from the stamps
*          Solving linear equation systems
*          Performing diverse operations with the matrix:
*              -Set to zero all the elements of the matrix
*              -Get an array with maximum absolute value of each row
```

```
*              -Pre-multiply the matrix by a row vector
*              -Post-multiply the matrix by a column vector
*              -Print the matrix
* ========================================================================
--Example of Usage of the  CLASS SPARSE MATRIX:
USE SPARSE
COMPONENT TestSparseMatrix01
   DATA
      REAL b[5] = { 10.0,  0.25,  4.50, -0.25, 13.5}
      --independent terms of the linear equation system
      REAL c[5] = {  3.0,  0.50, -0.50,  2.50, -1.0}
      --1D array
   DECLS
      DISCR REAL x[5] --Solution of the linear equation system
      DISCR REAL m[5] --Maximum absolute values per row
      DISCR REAL d[5] --Result of postmultiplication
      DISCR REAL e[5] --Result of premultiplication
      DISCR REAL error
   OBJECTS
      --Declaration of a sparse matrix with up to 5 rows
      --and up to 20 possible stampings
      SPARSE_MATRIX(MAX_ROW=10, MAX_STAMP=20)  A
   INIT
      A.setToZero()
      -- Let the matrix be 5x5 with the values shown below:
      -- A = [
      --        8,   0,   0,   0,   1;
      --        0, 1+2,   0, 4+1,   0;
      --        1,   0,   3,   0,   0;
      --        0,   1,   0,   3,   0;
      --        3,   0,   1,   0, 2-6;
      --     ];
      --Fill the matrix by stamping as follows:
      A.stampItem(1, 1, 8.)
      A.stampItem(2, 2, 1.)
      A.stampItem(2, 2, 2.)
      A.stampItem(3, 3, 3.)
      A.stampItem(4, 4, 3.)
      A.stampItem(5, 5, 2.)
      A.stampItem(5, 5,-6.)
      A.stampItem(1, 5, 1.)
      A.stampItem(2, 4, 4.)
      A.stampItem(2, 4, 1.)
      A.stampItem(3, 1, 1.)
      A.stampItem(4, 2, 1.)
      A.stampItem(5, 1, 3.)
      A.stampItem(5, 3, 1.)
      --
      --After stamping
      --nstamps    = 14
      --stampIrow  = {  1,  2,  2,  3,  4,  5,  5,  1,  2,  2,  3,  4,  5,  5}
      --stampIcol  = {  1,  2,  2,  3,  4,  5,  5,  5,  4,  4,  1,  2,  1,  3}
      --stampIpos  = {  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14}
      --stampValue = { 8., 1., 2., 3., 3., 2.,-6., 1., 4., 1., 1., 1., 3., 1.}
      A.Print()
      ----------------------------------------------------------
      --Construcción de la matriz en formato CSR
      A.buildCSR()
      --After this method, the stamps are sorted
      --stampIrow    = { 1,   1,   2,   2,   2,   2,   3,   3,   4,   4,   5,   5,   5,   5},
      --stampIcol    = { 1,   5,   2,   2,   4,   4,   1,   3,   2,   4,   1,   3,   5,   5},
      --stampIpos    = { 1,   8,   2,   3,  10,   9,  11,   4,  12,   5,  13,  14,   6,   7}
      --stampValue   = {8.,  1.,  1.,  2.,  1.,  4.,  1.,  3.,  1.,  3.,  3.,  1.,  2., -6.}
      --The CSR format is built
      --nnzeros = 11
      --csrColInd = {1,    5,   2,   4,   1,   3,   2,   4,   1,   3,   5}
      --csrValue  = {8.,  1.,  3.,  5.,  1.,  3.,  1.,  3.,  3.,  1., -4.}
      --csrRowPtr = {1,    3,   5,   7,   9,  12}
       --The following array indicates the destination position in CSR format of the unsorted
stamps
       --stampToCSR = {  1,  3,   3,   6,   8,  11,  11,   2,   4,   4,   5,   7,   9,  10}
      A.Print()
      ----------------------------------------------------------
      --Solving the system of linear equations A·x = b
      --where the vector of independent terms
      ----------------------------------------------------------
      --b = [
      --      10.0;
      --      0.25;
      --      4.50;
      --     -0.25;
      --      13.5;
      --    ]
```

```
      A.LinearEqnSolve(b, x, 1., 0)
      WRITE("\n\nSolution:\n")
      WRITE("    x[5]   = { %lg , %lg , %lg , %lg , %lg }\n",
                           x[1], x[2], x[3], x[4], x[5])
      ------------------------------------------------------
      --Find the maximum absolute value of each row
      A.MaxAbsRow(m)
      WRITE("\n\nMax.Abs.Value of each row:\n")
      WRITE("    m[5]   = { %lg , %lg , %lg , %lg , %lg }\n",
                           m[1], m[2], m[3], m[4], m[5])
      ------------------------------------------------------
      --Pre-multiplication by a row vector
      A.preMultiplyByRow(c, e)
      WRITE("\n\nResult of Premultiplicacion:\n")
      WRITE("    e[5]   = { %lg , %lg , %lg , %lg , %lg }\n",
                           e[1], e[2], e[3], e[4], e[5])
      ------------------------------------------------------
      ---Postmulitplicacion by column vector
      A.postMultiplyByColumn(c, d)
      WRITE("\n\nResult of Post-multiplicacion:\n")
      WRITE("    d[5]   = { %lg , %lg , %lg , %lg , %lg }\n",
                           d[1], d[2], d[3], d[4], d[5])

END COMPONENT*/
   DECLS
      BOOLEAN isCSRBuilt = FALSE  "boolean indicating whether the sparse matrix \
                                   has been built in CSR format or not"
      BOOLEAN isSetToZero = TRUE  "boolean indicating if the sparse matrix \
                                   has been reset to zero or not"
      --Stamps representation
      INTEGER nstamp = 0                    "Total number of stamps"
      INTEGER nrow = 0                      "Total number of rows"
      INTEGER ncol = 0                      "Total number of columns"
      INTEGER stampIrow[MAX_STAMP]  = 0     "Number of row of the stamp"
      INTEGER stampIcol[MAX_STAMP]  = 0     "Number of column of the stamp"
      INTEGER stampIpos[MAX_STAMP]  = 0     "Position of the stamp before sorting"
      REAL    stampValue[MAX_STAMP] = 0.    "Real value of the stamp"
      INTEGER stampToCSR[MAX_STAMP] = 0     "Mapping of stamp contributions in CSR format"
      INTEGER STAMP_debug                   "Debug level of the stamping process (0/1)"
      --CSR representation of the matrix
      INTEGER nnzeros = 0                   "Total number of non zero items"
      INTEGER csrColInd[MAX_STAMP]  = 0     "Columns of the matrix items in CSR format"
      INTEGER csrRowPtr[MAX_ROW+1]  = 0     "Pointers to beginning of each row
      REAL    csrValue[MAX_STAMP]   = 0.    "Real Values in CSR format"
   METHODS
      ------------------------------------------------------------------------------
      --Method to set to zero the sparse matrix
      ------------------------------------------------------------------------------
      METHOD NO_TYPE setToZero()
         BODY
            isSetToZero = TRUE
            nstamp = 0
            FOR(i IN 1, nnzeros)
               csrValue[i] = 0
            END FOR
            RETURN
      END METHOD
      ------------------------------------------------------------------------------
      --Method to swap two stamps - Auxiliary method for the Quicksort
      ------------------------------------------------------------------------------
      METHOD NO_TYPE SwapItems(
         IN INTEGER i "Position of the first stamp",
         IN INTEGER j "Position of the second stamp")
         DECLS
            INTEGER swap[3]
            REAL v
         BODY
            swap[1] = stampIrow[i]
            swap[2] = stampIcol[i]
            swap[3] = stampIpos[i]
            v = stampValue[i]
            --
            stampIrow[i] = stampIrow[j]
            stampIcol[i] = stampIcol[j]
            stampIpos[i] = stampIpos[j]
            stampValue[i] = stampValue[j]
            --
            stampIrow[j] = swap[1]
            stampIcol[j] = swap[2]
            stampIpos[j] = swap[3]
            stampValue[j] = v
            RETURN
      END METHOD
```

```
246         ------------------------------------------------------------------------------
247         --Method two compare two stamp positions of the matrix to find which one
248         --is first in the CSR format
249         ------------------------------------------------------------------------------
250         METHOD BOOLEAN Compare(
251            IN INTEGER irow1 "row index of first stamp",
252            IN INTEGER icol1 "column index of first stamp",
253            IN INTEGER irow2 "row index of second stamp",
254            IN INTEGER icol2 "column index of second stamp"
255            )
256            DECLS
257               BOOLEAN bval = FALSE
258            BODY
259               IF (irow1 < irow2  OR (irow1 == irow2 AND icol1 < icol2)) THEN
260                  bval = TRUE
261               END IF
262               RETURN bval
263         END METHOD
264         ------------------------------------------------------------------------------
265         --Partition method called by the Quicksort method
266         ------------------------------------------------------------------------------
267         METHOD INTEGER partition(
268            IN INTEGER lo "Start index of the partition",
269            IN INTEGER hi "Ending index of the partition"
270            )
271            DECLS
272               INTEGER pivot[2]
273               INTEGER i
274            BODY
275               pivot[1] = stampIrow[hi]
276               pivot[2] = stampIcol[hi]
277               i = lo
278               FOR (j IN lo, hi)
279                  IF Compare(stampIrow[j], stampIcol[j], pivot[1], pivot[2]) THEN
280                     SwapItems(i, j)
281                     i = i + 1
282                  END IF
283               END FOR
284               SwapItems(i, hi)
285               RETURN i
286         END METHOD
287         ------------------------------------------------------------------------------
288         --QuickSort Algorithm to sort the sparse matrix stamps in CSR format
289         ------------------------------------------------------------------------------
290         METHOD NO_TYPE QuickSort(
291               IN INTEGER lo  "the starting index of the stamp array or the current sub-array",
292               IN INTEGER hi  "the ending index of the stamp array or the current sub-array"
293               )
294            DECLS
295               INTEGER p
296            BODY
297               IF(lo < hi) THEN
298                  p = partition(lo, hi)
299                  QuickSort(lo,  p - 1)
300                  QuickSort(p + 1, hi)
301               END IF
302               RETURN
303         END METHOD
304         ------------------------------------------------------------------------------
305         --Method to build the matrix in CSR format from the unsorted stamps
306         ------------------------------------------------------------------------------
307         METHOD NO_TYPE buildCSR()
308            DECLS
309               INTEGER i
310               INTEGER k
311               INTEGER i1, i2
312            BODY
313               isSetToZero = FALSE
314               --It checks if the CSR sorting is still valid
315               IF(NOT isCSRBuilt) THEN
316                  ASSERT (nrow == ncol) ERROR  "Non square sparse matrix"
317                  --Sorting of the stamps for CSR format using QuickSort
318                  QuickSort(1, nstamp)
319                  --Once the stamps have been sorted, the CSR representation is built
320                  nnzeros = 0
321                  --
322                  FOR (i IN 1, nstamp)
323                     --It detects if the position of the actual stamp is equal to the position
324                     --of the previous stamp
325                     IF(i > 1 AND (stampIrow[i] == stampIrow[i-1]) AND (stampIcol[i] ==
326 stampIcol[i-1])) THEN
327                        csrValue[nnzeros] = csrValue[nnzeros] + stampValue[i]
328                     ELSE
```

```
                    nnzeros = nnzeros + 1
                    csrColInd[nnzeros] = stampIcol[i]
                    csrValue[nnzeros] = stampValue[i]
                    IF(i == 1) THEN
                       i1 = 1
                       i2 = stampIrow[i]
                       ASSERT(i1 == i2) ERROR "Empty row in sparse matrix"
                       FOR(k IN i1, i2)
                          csrRowPtr[k]= nnzeros
                       END FOR
                    ELSEIF(stampIrow[i] != stampIrow[i-1]) THEN
                       i1 = stampIrow[i-1]+1
                       i2 = stampIrow[i]
                       ASSERT(i1 == i2) ERROR "Empty row in sparse matrix"
                       FOR(k IN i1, i2)
                          csrRowPtr[k]= nnzeros
                       END FOR
                    END IF
                 END IF
                 k = stampIpos[i]
                 stampToCSR[k] = nnzeros
              END FOR
              csrRowPtr[nrow+1] = nnzeros + 1
              isCSRBuilt = TRUE
           END IF
           RETURN
     END METHOD
     ------------------------------------------------------------------------
     --Method to stamp values in any order
     ------------------------------------------------------------------------
     METHOD NO_TYPE stampItem(
        IN INTEGER ir     "Row index of the stamp",
        IN INTEGER ic     "Column index of the stamp",
        IN REAL rval       "Additive value contribution of the stamp"
        )
        DECLS
           INTEGER izero
        BODY
           ASSERT isSetToZero FATAL "Before stamping items the matrix has to be zeroed"
           nstamp = nstamp + 1
           ASSERT (nstamp <= MAX_STAMP) FATAL "Maximum number of SPARSE_MATRIX Stamps exceeded.
\
                                       Create the SPARSE_MATRIX object with increased
MAX_STAMP"

           IF(ir > nrow) THEN
                 ASSERT (ir <= MAX_ROW) FATAL "Maximum number of SPARSE_MATRIX rows exceeded. \
                                       Create the SPARSE_MATRIX object with increased MAX_ROW"
                 nrow = ir
           END IF
           IF(ic > ncol) THEN
              ASSERT (ic <= MAX_ROW) FATAL "Maximum number of SPARSE_MATRIX columns exceeded. \
                                    Create the SPARSE_MATRIX object with increased MAX_ROW"
                 ncol = ic
           END IF
           stampIpos[nstamp] = nstamp
           stampIrow[nstamp] = ir
           stampIcol[nstamp] = ic
           stampValue[nstamp] = rval
           IF(STAMP_debug > 0) THEN
              WRITE("Stamp %d into Matrix Position[%d,%d] = %lg \n", nstamp, ir, ic, rval)
           END IF
           --Tty to use previous sorting if the CSR format has been already built
           IF(isCSRBuilt) THEN
              izero = stampToCSR[nstamp]
              IF(ic == csrColInd[izero] AND izero >= csrRowPtr[ir] AND izero < csrRowPtr[ir+1])
THEN
                 csrValue[izero] = csrValue[izero] + rval
              ELSE
                 isCSRBuilt = FALSE
              END IF
           END IF
           RETURN
     END METHOD
     ------------------------------------------------------------------------
     --Method to solve linear equation system
     ------------------------------------------------------------------------
     METHOD NO_TYPE LinearEqnSolve(
        IN REAL b[]   "independent terms",
        OUT REAL x[]  "solution vector",
        IN REAL pivot_threshold "pivot threshold",
        IN INTEGER superlu_debug    "0/1 Debug level for SuperLU"
        )
```

```
412          DECLS
413             INTEGER factors[4]
414             INTEGER info
415             INTEGER superlu_equil =  1
416             INTEGER superlu_refine = 0
417          BODY
418             ASSERT(isCSRBuilt) FATAL "Method LinearEqnSolve of class SPARSE_MATRIX \
419                                       can be called only if CSR format is built"
420             FOR(i IN 1, nrow)
421                x[i] = b[i]
422             END FOR
423             --Factorize Matrix into L & U
424             d_ecosim_superlu(1, nrow, nnzeros, 1, csrValue, csrColInd, csrRowPtr, x, factors,
425                    info, pivot_threshold, superlu_equil, superlu_refine, superlu_debug)
426
427             IF (info != 0) THEN
428                WRITE("\n****Error in SUPERLU during Factorization info = %d\n\n", info)
429             END IF
430             --Solve Linear System
431             d_ecosim_superlu(2, nrow, nnzeros, 1, csrValue, csrColInd, csrRowPtr, x, factors,
432                    info, pivot_threshold, superlu_equil, superlu_refine, superlu_debug)
433             IF (info != 0) THEN
434                WRITE("\n****Error in SUPERLU during Solve info = %d\n\n", info)
435             END IF
436             --Destroy Matrix
437             d_ecosim_superlu(3, nrow, nnzeros, 1, csrValue, csrColInd, csrRowPtr, x, factors,
438                    info, pivot_threshold, superlu_equil, superlu_refine, superlu_debug)
439             IF (info != 0) THEN
440                WRITE("\n****Error in SUPERLU during Memory deallocation info = %d\n\n", info)
441             END IF
442             RETURN
443       END METHOD
444       ------------------------------------------------------------------------
445       --Get an array with maximum absolute value of each row
446       ------------------------------------------------------------------------
447       METHOD NO_TYPE MaxAbsRow(
448          OUT REAL rmax[] "Vector with the absolute maximum of each row"
449          )
450          DECLS
451             INTEGER i
452             INTEGER j
453             INTEGER k, k1, k2
454          BODY
455             ASSERT(isCSRBuilt) FATAL "Method MaxAbsRow of class SPARSE_MATRIX \
456                                       can be called only if CSR format is built"
457             FOR(i IN 1, nrow)
458                k1 = csrRowPtr[i]
459                k2 = csrRowPtr[i+1]-1
460                rmax[i] = 0.
461                FOR(k IN k1, k2)
462                   rmax[i] = max(rmax[i], abs(csrValue[k]))
463                END FOR
464                IF(rmax[i] == 0) THEN
465                   WRITE("****WARNING - Singular or Almost Singular Jacobian\n")
466                END IF
467             END FOR
468             RETURN
469       END METHOD
470       ------------------------------------------------------------------------
471       -- Method to pre-multiply a row vector by the matrix
472       ------------------------------------------------------------------------
473       METHOD NO_TYPE preMultiplyByRow(
474          IN REAL X[] "Input Row vector",
475          OUT REAL Y[] "Results vector")
476          DECLS
477             INTEGER i "Loop index for rows"
478             INTEGER j "Loop index for columns"
479             INTEGER k, k1, k2 "Loop indices for matrix elements"
480          BODY
481          -- Check if CSR format is built before proceeding
482          ASSERT(isCSRBuilt) FATAL "Method preMultiplyByRow of class SPARSE_MATRIX \
483                                    can be called only if CSR format is built"
484          -- Initialize the result vector to zero
485          FOR(j IN 1, ncol)
486             Y[j] = 0.
487          END FOR
488          -- Perform matrix-vector multiplication
489          FOR(i IN 1, nrow)
490             k1 = csrRowPtr[i]
491             k2 = csrRowPtr[i+1]-1
492             FOR(k IN k1, k2)
493                 j = csrColInd[k]
494                 -- Accumulate the product of matrix element and vector element
```

```
                Y[j] = Y[j] + csrValue[k] * X[i]
             END FOR
          END FOR
          -- Return the result vector
          RETURN
       END METHOD
       ------------------------------------------------------------------------
       --Method to multiply the matrix by a column vector
       ------------------------------------------------------------------------
       METHOD NO_TYPE postMultiplyByColumn(
             IN REAL X[] "Input Column vector",
             OUT REAL Y[] "Results vector")
          DECLS
             INTEGER i  "Loop index for rows"
             INTEGER j  "Column index"
             INTEGER k, k1, k2 "Loop indices for matrix elements"
          BODY
             -- Check if CSR format is built before proceeding
             ASSERT(isCSRBuilt) FATAL "Method postMultiplyByColumn of class SPARSE_MATRIX \
                                       can be called only if CSR format is built"
             -- Initialize the result vector to zero
             FOR(i IN 1, nrow)
                Y[i] = 0.
             END FOR
             -- Perform matrix-vector multiplication
             FOR(i IN 1, nrow)
                k1 = csrRowPtr[i]
                k2 = csrRowPtr[i+1]-1
                FOR(k IN k1, k2)
                   j = csrColInd[k]
                   -- Accumulate the product of matrix element and vector element
                   Y[i] = Y[i] + csrValue[k]*X[j]
                END FOR
             END FOR
             -- Return the result vector
             RETURN
       END METHOD
       ------------------------------------------------------------------------
       --Method to print the sparse matrix
       ------------------------------------------------------------------------
       METHOD NO_TYPE Print()
          DECLS
          BODY
             IF(isCSRBuilt) THEN
                WRITE("\n\nnnzeros = %10d\n", nnzeros)
                WRITE("nrow     = %10d\n", nrow)
                WRITE("ncol = %10d\n\n", ncol)
                FOR(j IN 1, nnzeros)
                   WRITE("csrColInd[%d]= %d \t val[%d] = %14.7g\n", j, csrColInd[j], j,
csrValue[j])
                END FOR
                WRITE("\n")
                FOR(j IN 1, nrow+1)
                   WRITE("csrRowPtr[%d] = %d \n", j, csrRowPtr[j])
                END FOR
             ELSE
                WRITE("\nnstamp = %10d\n", nstamp)
                WRITE("   Row      Column         Value         Ipos    Ipos CSR\n")
                WRITE("  ======    ======    ==============    ======    ======\n")
                FOR(j IN 1, nstamp)
                   WRITE("  %6d    %6d    %14.7lg    %6d    %6d\n", stampIrow[j], stampIcol[j], \
                                          stampValue[j], stampIpos[j], stampToCSR[j])
                END FOR
             END IF
             RETURN
       END METHOD
END CLASS

CLASS NLEQN_SYSTEM_BASE (
       INTEGER MAX_EQN      "Maximum number of equations",
       INTEGER MAX_STAMP    "Maximum number of stamps to build the Jacobian matrix"
   )
   DECLS
       INTEGER neqn = 0                    "Number of equations"
       INTEGER iter = 0                    "Iteration number of the NR"
       INTEGER NR_debug = 1                "Debug level of the Newton Raphson- range 0/1/2"
       INTEGER STAMP_debug = 0             "Debug level of the stamping process range 0/1"
       INTEGER superLU_debug = 0           "Debug level of the superLU solver 0/1"
       REAL Fun[MAX_EQN] = 0.              "Vector of new residues"
       REAL FunOld[MAX_EQN] = 0.           "Vector of old residues"
       REAL Xunk[MAX_EQN] = 0.             "Vector of new values of the unknowns"
       REAL XunkOld[MAX_EQN] = 0.          "Vector of old values of the unknowns"
       REAL XunkMax[MAX_EQN] =  1.e10      "Vector of maximum limits for the unknowns"
```

```
578         REAL XunkMin[MAX_EQN] = -1.e10      "Vector of minimum limits for the unknowns"
579         INTEGER freezeIterUnk[MAX_EQN]=0    "Number of freezed iterations for the unknowns"
580         STRING descrUnk[MAX_EQN]            "String with a description of the unknowns"
581         STRING descrEqn[MAX_EQN]            "String with a description of the equation"
582         REAL pivot_threshold = 1.           "Specifies the threshold used for a diagontal \
583                                              entry to be an acceptable pivot"
584         REAL tol = 1.e-7                    "Tolerance"
585         BOOLEAN converged = FALSE           "Boolean, if TRUE, the solver has converged"
586         CONST REAL MinFracLimit = 0.1
587      OBJECTS
588         SPARSE_MATRIX (MAX_ROW = MAX_EQN, MAX_STAMP = MAX_STAMP) Jacob    "Jacobian matrix"
589      METHODS
590         -----------------------------------------------------------------------------
591         -- Method to set to zero the residues of the functions and the Jacobian
592         -----------------------------------------------------------------------------
593         METHOD NO_TYPE setToZero()
594            BODY
595               --nstamp = 0
596               FOR (i IN 1, neqn)
597                  Fun[i] = 0
598               END FOR
599               Jacob.setToZero()
600               neqn = 0
601            RETURN
602         END METHOD
603         -----------------------------------------------------------------------------
604         --Method to create an unknown
605         -----------------------------------------------------------------------------
606         METHOD NO_TYPE CreateUnk(
607               IN INTEGER i "Unknown number",
608               IN REAL Xo   "Initial value of the unknown",
609               IN REAL Xmin "Minimum allowable value for the unknown",
610               IN REAL Xmax "Maximum allowable value for the unknown",
611               IN STRING unkn_descrip "Description of the unknown variable",
612               IN INTEGER freezeIter "Number of iterations to freeze the unknown "
613               )
614            BODY
615               Xunk[i] = Xo
616               IF(Xmin != 0. OR Xmax != 0.) THEN
617                  XunkMax[i] = Xmax
618                  XunkMin[i] = Xmin
619               ELSE
620                  XunkMax[i] =  1e10
621                  XunkMin[i] = -1e10
622               END IF
623               ASSERT(Xunk[i] >= XunkMin[i] AND Xunk[i] <= XunkMax[i]) ERROR \
624                                "Initial value of unknown is out of allowable range"
625               descrUnk[i] = unkn_descrip
626               freezeIterUnk[i] = freezeIter
627               RETURN
628         END METHOD
629         -----------------------------------------------------------------------------
630         --Method to create the comment or description of an equation
631         -----------------------------------------------------------------------------
632         METHOD NO_TYPE CreateEqn(
633               IN INTEGER i "Equation number" ,
634               IN STRING eqn_descrip "Description of the equation")
635            BODY
636               descrEqn[i] = eqn_descrip
637               RETURN
638         END METHOD
639
640         METHOD NO_TYPE UpdateUnknownVec(
641            IN REAL FRAC,
642            IN REAL XunkOld[],
643            IN REAL dx_nr[],
644            --IN REAL dx_sp[],
645            OUT REAL Xunk[],
646            OUT REAL errorx[],
647            OUT REAL errorx_tot,
648            OUT INTEGER iworstx
649            )
650            DECLS
651               REAL fracLimit
652               REAL errorx max
653               REAL MinFracLimit = 0.5
654            BODY
655               --Calculate the fractional limit of the Newton-Raphson step
656               FOR(j IN 1, neqn)
657                  fracLimit = FRAC
658                  -- Adjust the limit to ensure that neither the maximum nor the minimum
659                  -- limits are exceeded
660                  IF(dx_nr[j] > 0.) THEN
```

```
                  fracLimit = 0.99 * (XunkMax[j] - XunkOld[j])/dx_nr[j]
               ELSEIF(dx_nr[j] < 0.) THEN
                  fracLimit = 0.99 * (XunkMin[j] - XunkOld[j])/dx_nr[j]
               END IF
               IF(fracLimit < MinFracLimit) THEN
                  fracLimit = MinFracLimit
               END IF
               IF(fracLimit < FRAC) THEN
                  FRAC = fracLimit
               END IF
            END FOR

            FOR(j IN 1, neqn)
               IF(iter >= freezeIterUnk[j]) THEN
                  Xunk[j] = min(max(XunkOld[j] + FRAC * dx_nr[j], XunkMin[j]), XunkMax[j])
               ELSE
                  Xunk[j] = Xunk[j]
               END IF
            END FOR
            errorx_tot = 0.
            errorx_max = 0.
            iworstx = 0
            FOR (j IN 1, neqn)
               IF(abs(Xunk[j]) > 10*tol) THEN
                  errorx[j] = (Xunk[j]-XunkOld[j])/abs(Xunk[j])
               ELSE
                  errorx[j] =  Xunk[j]-XunkOld[j]
               END IF
               IF(abs(errorx[j]) > errorx_max) THEN
                  iworstx = j
                  errorx_max = abs(errorx[j])
               END IF
               --Accumulate the total variable error
               errorx_tot = errorx[j]**2 + errorx_tot
            END FOR
            errorx_tot = errorx_tot**0.5
            RETURN
      END METHOD
      ------------------------------------------------------------------------------
      -- Method to evaluate the error of the residues
      -- Errors are scaled by dividing each by the maximum absolute value
      -- in the corresponding row of the Jacobian matrix.
      -- A total error is calculated by composing individual errors
      -- using the Euclidean norm.
      -- Additionally, the worst-scaled residue is identified.
      ------------------------------------------------------------------------------
      METHOD NO_TYPE EvalFunError(
            IN REAL f[],
            IN REAL df[],
            OUT REAL errorf[],
            OUT REAL errorf_tot,
            OUT INTEGER iworstf)
         DECLS
            REAL errorf_max
         BODY
            errorf_tot = 0.
            errorf_max = 0.
            iworstf = 0
            FOR(j IN 1, neqn)
               -- Calculate the scaled error for each equation
               errorf[j] = f[j]/df[j]
               -- Update the worst-scaled residue if needed
               IF(abs(errorf[j]) > errorf_max) THEN
                  iworstf = j
                  errorf_max = abs(errorf[j])
               END IF
               -- Accumulate the total error
               errorf_tot = errorf_tot + errorf[j]**2
            END FOR
            -- Calculate the Euclidean norm of the total error
            errorf_tot = errorf_tot**0.5
            RETURN
      END METHOD
      METHOD NO_TYPE GetUnknown(OUT REAL X[])
         BODY
            FOR (i IN 1, neqn)
               X[i] = Xunk[i]
            END FOR
            RETURN
      END METHOD
END CLASS

```

```
TYPEDEF FUNCTION NO_TYPE func_Fsystem(
/*=========================================================================
--Declaration of the function pointer to calculate the residues and
--the Jacobian from the unknowns and the data
=========================================================================*/
      IN INTEGER iopt      "int that specifies the operation",
      IN REAL X[]          "vector of unknowns",
      IN REAL RDATA[]      "Auxiliary vector for communication",
      OUT REAL F[]         "Residues",
      OUT NLEQN_SYSTEM_BASE nleqn_system  "Base of the non linear equation system"
      )
/*
* iopt             = Integer specifying the operation:
*                      1, to set the number of equations, create unknowns and equations
*                      2, to calculate the residuals
*                      3, to calculate residuals and functions
* X[]              = Real array containing a vector of unknowns
* RDATA            = Real parameter arrays used for communication
*                    with the calling and current function
* nleqn_system     = Object of type NLEQN_SYSTEM_BASE, a parent class
*                    for NLEQN_SYSTEM. NLEQN_SYSTEM_BASE includes all attributes
*                    and methods of NLEQN_SYSTEM except the SOLVE method.
*                    The SOLVE method calls this function using the "this"
*                    pointer, allowing the function to access the class to define
*                    the number of equations, create unknowns and equations,
*                    and fill the Jacobian by stamping
*/

CLASS NLEQN_SYSTEM IS_A NLEQN_SYSTEM_BASE
/* ========================================================================
* CLASS: NLEQN SYSTEM
* Purpose: This class is designed for the resolution of
*          sparse non-linear equation systems within the EcosimPro language,
*          utilizing the Newton-Raphson Method.
*          It is particularly well-suited for large-scale simulations,
*          specifically handling sparse systems of equations with a large size,
*          often reaching thousands of equations.
* ========================================================================
--Example of Usage of the  CLASS SPARSE MATRIX:
USE SPARSE
FUNCTION NO_TYPE fsystem(IN INTEGER iopt, IN REAL x[], IN REAL rdata[], OUT REAL F[], OUT
NLEQN_SYSTEM_BASE nleqn_system)
   DECLS
   BODY
      nleqn_system.neqn = 4
      IF(iopt == 0) THEN
            FOR(j IN 1, nleqn_system.neqn)
               nleqn_system.CreateUnk(j, 0.1, -1e6, 1e6, concatStrings(concatStrings("x[",
integerToString(j)),"]"),0)
            END FOR
            nleqn_system.CreateEqn(1,"-x[1]**2 - x[2]**2 - x[3]**2 + x[4]    = 0")
            nleqn_system.CreateEqn(2," x[1]**2 + x[2]**2 + x[3]**2 + x[4]**2 = 0")
            nleqn_system.CreateEqn(3," x[1] - x[2] = 0")
            nleqn_system.CreateEqn(4," x[2] - x[3] = 0")
      ELSEIF(iopt == 1 OR iopt == 2) THEN
         nleqn_system.Jacob.setToZero()
         F[1] =  - x[1]**2 - x[2]**2 - x[3]**2 + x[4]
         F[2] =    x[1]**2 + x[2]**2 + x[3]**2 + x[4]**2 - 1
         F[3] =    x[1] - x[2]
         F[4] =    x[2] - x[3]
         IF(iopt == 2) THEN
            nleqn_system.Jacob.stampItem(1, 1, -2.*x[1])
            nleqn_system.Jacob.stampItem(1, 2, -2.*x[2])
            nleqn_system.Jacob.stampItem(1, 3, -2.*x[3])
            nleqn_system.Jacob.stampItem(1, 4, 1.0)
            nleqn_system.Jacob.stampItem(2, 1, 2.*x[1])
            nleqn_system.Jacob.stampItem(2, 2, 2.*x[2])
            nleqn_system.Jacob.stampItem(2, 3, 2.*x[3])
            nleqn_system.Jacob.stampItem(2, 4, 2.*x[4])
            nleqn_system.Jacob.stampItem(3, 1, 1.0)
            nleqn_system.Jacob.stampItem(3, 2, -1.0)
            nleqn_system.Jacob.stampItem(4, 2, 1.0)
            nleqn_system.Jacob.stampItem(4, 3, -1.0)
         END IF
      END IF
      RETURN
END FUNCTION
COMPONENT TestNleqnSystem01
   DATA
      INTEGER iter_damp = 2  "Number of initial damped iterations"
      INTEGER iter_max = 20  "Maximum number of iterations"
      REAL FRAC damp = 0.1
      REAL tol = 1e-6        "Tolerance"
```

```
827         INTEGER NR_debug = 2    "Debugging level of the Newton-Raphson"
828         INTEGER STAMP_debug = 0
829      DECLS
830         DISCR REAL FRAC = 1
831         DISCR REAL errorx
832         DISCR REAL errorf
833         DISCR REAL x[20]
834         DISCR REAL rdata[10]
835      OBJECTS
836         NLEQN_SYSTEM(MAX_EQN=20, MAX_STAMP = 100) nleqn_system
837      INIT
838         nleqn system.NR debug = NR debug
839         nleqn_system.Jacob.STAMP_debug = STAMP_debug
840         nleqn_system.Solve(fsystem, iter_max, iter_damp, FRAC_damp, tol, rdata)
841         nleqn_system.GetUnknown(x)
842   END COMPONENT*/
843
844      METHODS
845   ------------------------------------------------------------------------------
846   -- Method to apply the Newton-Raphson (NR) method on a system of non-linear equations
847   --  Some details of the algorithm are based on two papers:
848   --  1) Numerical Solution of Constrained Non-Linear Algebraic Equations
849   --     by Mordechai Shacham
850   --  2) A class of Methods for Solving Non Linear Simultaneous Equations by C.G.Broyden
851   --
852   ------------------------------------------------------------------------------
853      METHOD NO_TYPE Solve(
854         IN FUNC_PTR <func_Fsystem> fsys "Pointer to the function with the equations",
855         IN INTEGER iter_max            "Maximum number of iterations",
856         IN INTEGER iter_damp           "Number of initial damped iterations",
857         IN REAL FRAC_damp              "Multiplier of NR step during first iter_damp iterations",
858         IN REAL tol_arg                "Tolerance of the NR",
859         IN REAL rdata[]                "Auxiliary vector for communication"
860         )
861         DECLS
862            REAL FRAC                   "Fractional value for updating the Newton step"
863            REAL df_max[MAX_EQN]        "Maximum absolute values of Jacobian rows"
864            REAL dx_nr[MAX_EQN]         "Newton step for unknowns"
865            REAL dx_sp[MAX_EQN]         "Gradient step for unknowns"
866            REAL J_dx_sp[MAX_EQN]       "Jacobian times the gradient step"
867            REAL nFun[MAX_EQN]          "Negative values of the system of equations"
868            REAL fracLimit              "Limiting value for fractional update"
869            REAL eta                    "Correction factor for reducing the Newton step"
870            INTEGER i, j, k, k1, k2     "Loop indices and counters"
871            REAL errorx[MAX_EQN]        "Error in unknowns at each iteration"
872            REAL errorx_max             "Maximum error in unknowns"
873            REAL errorx_tot             "Total error in unknowns"
874            REAL errorf[MAX_EQN]        "Error in equations at each iteration"
875            REAL errorf max             "Maximum error in equations"
876            REAL errorf tot             "Total error in equations"
877            REAL errorf_tot_old         "Total error in equations at the previous iteration"
878            INTEGER iworstx, iworstf    "Indices of the worst errors in unknowns and equations"
879         BODY
880            iter = 1
881            tol = tol_arg
882            converged = FALSE
883            fsys(0, Xunk, rdata, Fun, this)
884            WHILE((iter <= iter_max) AND ((NOT converged)))
885               IF(iter == 1) THEN
886                  IF(NR_debug > 1) THEN
887                     WRITE("*************************************************************************\
888                           **************************************************\n")
889                     WRITE("****************************   List of Unknowns  ******************\
890                           **************************************************\n")
891                     FOR(j IN 1, neqn)
892                        WRITE("Unknown  %4d \t %s\n", j, descrUnk[j])
893                     END FOR
894                     WRITE("****************************   List of Equations  *****************\
895                            **************************************************\n")
896                     FOR(j IN 1, neqn)
897                        WRITE("Equation %4d \t %s\n", j, descrEqn[j])
898                     END FOR
899                     WRITE("*************************************************************************\
900                           **************************************************\n")
901                  END IF
902                  Jacob.setToZero()
903                  fsys(2, Xunk, rdata, Fun, this)
904                  Jacob.buildCSR()
905                  Jacob.MaxAbsRow(df_max)
906                  EvalFunError(Fun, df_max, errorf, errorf_tot, iworstf)
907               END IF
908               FOR(j IN 1, neqn)
909                  XunkOld[j] = Xunk[j]
```

```
910                FunOld[j] = Fun[j]
911                nFun[j] =  -Fun[j]
912                errorf_tot_old = errorf_tot
913             END FOR
914             --Calculation of the Newton step
915             Jacob.LinearEqnSolve(nFun, dx_nr, 1., superLU_debug)
916             --Calculation of the gradient step
917             Jacob.preMultiplyByRow(nFun, dx_sp)
918             Jacob.postMultiplyByColumn(dx_sp, J_dx_sp)
919             FRAC=1
920             IF(iter <= iter_damp) THEN
921                FRAC = FRAC_damp
922             END IF
923             UpdateUnknownVec(FRAC, XunkOld, dx_nr, Xunk, errorx, errorx_tot, iworstx)
924             Jacob.setToZero()
925             fsys(2, Xunk, rdata, Fun, this)
926             Jacob.buildCSR()
927             EvalFunError(Fun, df_max, errorf, errorf_tot, iworstf)
928             --Reduction of the Newton step in case of increasing residues
929             IF(errorf_tot > errorf_tot_old) THEN
930                eta = (errorf_tot / errorf_tot_old)**2
931                FRAC = FRAC*((1+6.*eta)**0.5 - 1)/(3.*eta)
932                UpdateUnknownVec(FRAC, XunkOld, dx_nr, Xunk, errorx, errorx_tot, iworstx)
933                Jacob.setToZero()
934                fsys(2, Xunk, rdata, Fun, this)
935                Jacob.buildCSR()
936                EvalFunError(Fun, df_max, errorf, errorf_tot, iworstf)
937             END IF
938             IF(NR_debug > 1) THEN
939                WRITE("*************************************************************\
940                   **************************************************\n")
941                WRITE("  No     Xunk. Old       Xunk. New     Dx Increment     Funct. Old      \
942                   Funct. New    Scal.Unk.Error  Scal.Eqn.Error\n")
943                WRITE("===== ============== ============== ============== ============== \
944                   ============== ============== ==============\n")
945                FOR(j IN 1, neqn)
946                   WRITE(" %4d  %14.6lg  %14.6lg  %14.6lg  %14.6lg  %14.6lg  %14.6lg  %14.6lg\n"\
947                    , j, XunkOld[j], Xunk[j], FRAC*dx_nr[j], FunOld[j], Fun[j], errorx[j],\
948                     errorf[j])
949                END FOR
950                WRITE("*************************************************************\
951                      **************************************************\n")
952             END IF
953             IF(NR_debug > 0)THEN
954                WRITE("*** iter = %3d  Fraction NR= %-6.4lf  Unk.Error= %-10.3lg  Fun.Error= %-
955 10.3lg  \
956                Worst Unk.= %-5d  Worst Eqn= %-5d\n", iter, FRAC, errorx_tot, errorf_tot,
957 iworstx, iworstf)
958             END IF
959             ASSERT(neqn == Jacob.nrow AND neqn == Jacob.ncol) FATAL  "Inconsistent dimension of
960 \
961                                                                  Equation System and Jacobian"
962             converged = (iter > 0) AND (errorx_tot < tol) AND (errorf_tot < 10 * tol)
963             iter = iter + 1
964          END WHILE
965       END METHOD
966 END CLASS
967
```

APÉNDICE F CÓDIGO DE DOS LIBRERÍAS PARA CÁLCULO DE ESTACIONARIOS HIDRÁULICOS

En este apéndice, se presentan los listados completos de las dos librerías para simulación de estacionarios hidráulicos, que se describen y evalúan en el capítulo 6 de esta tesis.

Primero, se presenta la librería designada ***HYDRAULIC_ST_CLASSIC***, cuyo desarrollo ha seguido el paradigma del MSOO. Seguidamente, se presenta la librería ***HYDRAULIC_ST***, que se ha desarrollado siguiendo la nueva aproximación. Ambas librerías son totalmente equivalentes y ambas se han diseñado en base a los requisitos de usuario descritos en la sección 6.1.

F.1. Librería Estacionarios Hidráulicos con la Nueva Aproximación

La librería ***HYDRAULIC_ST*** está diseñada para ser altamente eficiente y soportar la simulación estacionaria de modelos con miles de tuberías.

Esta librería hace a su vez uso de una librería de clases, denominada ***SPARSE***, que incluye una clase para matrices dispersas y otra clase para resolver sistemas de ecuaciones no lineales dispersas.

Aunque solo ofrece tres componentes: (1) Tubería ("***Pipe***"), (2) Tanque ("***Tank***") y (3) Salida de Flujo ("***Outflow***"), estos componentes pueden emplearse como modelos base para la definición de nuevos componentes, tales como bombas, válvulas de aislamiento y válvulas de control.

La programación de la nueva aproximación ha requerido 2 líneas de código. En este caso específico de la simulación hidráulica estacionaria con los tres componentes mencionados, la nueva aproximación demanda cinco veces más código en comparación con la aproximación clásica. Sin embargo, como se explica en el Capítulo 6, la nueva aproximación produce modelos más eficaces y con una mayor robustez numérica.

*Listado F-1: Listado de la Librería **HYDRAULIC_ST***

```
USE SPARSE
--Flag to control the part of the INIT block executed
BOOLEAN AssignSolution = FALSE

--Maximum Dimensions
CONST INTEGER MAX_PIPE = 500
CONST INTEGER MAX_TANK = 100
CONST INTEGER MAX_FLOW = 100

CONST INTEGER MAX_EQN = 2 * MAX_PIPE + 1 + MAX_TANK
CONST INTEGER MAX_STAMP = 5 * MAX_PIPE + 2 * MAX_TANK

--Counters
INTEGER npipe = 0 --Counter for the number of pipes
```

```
INTEGER ntank = 0 --Counter for the number of tanks
INTEGER nflow = 0 --Counter for the number of outlet flow conditions
INTEGER node  = 0 --Counter for the number of nodes
INTEGER neq   = 0 --Counter for the number of equations

PORT Hydraulic
   EQUAL INTEGER inode
END PORT

CLASS PipeClass
   DECLS
      INTEGER nd1              "Node at inlet"
      INTEGER nd2              "Node at outlet"
      REAL L       UNITS "m"    "Pipe length"
      REAL D       UNITS "m"    "Pipe inside diametert"
      REAL CHW     UNITS "-"    "Hazen Williams Coefficient"
      REAL Qo      UNITS "m3/s" "Initial guess flow"
      REAL H1      UNITS "m"    "Piezometric head at inlet"
      REAL H2      UNITS "m"    "Piezometric head at outlet"
      REAL Q       UNITS "m3/s" "Flow through pipe"
END CLASS

PipeClass P[MAX_PIPE]

CLASS TankClass
   DECLS
      INTEGER nod           "Node at tank outlet"
      REAL H   UNITS "m"    "Tank Piezometriz Head"
      REAL Q   UNITS "m3/s" "Tank outflow"
      REAL Qo  UNITS "m3/s" "Initial guess for pipe flow"
END CLASS

TankClass T[MAX_TANK]
CLASS OutflowClass
   DECLS
      INTEGER nod           "Node at tank outlet"
      REAL H   UNITS "m"    "Tank Piezometriz Head"
      REAL Q   UNITS "m3/s" "Initial guess for pipe flow"
END CLASS

OutflowClass OF[MAX_FLOW]

FUNCTION NO_TYPE InitPipe(
      IN REAL L,
      IN REAL D,
      IN REAL CHW,
      IN REAL Qo,
      OUT INTEGER ipipe,
      OUT INTEGER inode1,
      OUT INTEGER inode2,
      OUT REAL H1,
      OUT REAL H2,
      OUT REAL Q)
   BODY
      --Obtain topology and fill global data
      IF(ipipe == 0) THEN
         npipe = npipe+1
         ASSERT (npipe <= MAX PIPE) FATAL "Maximum number of pipe exceeded. \
                          Recompile library with increased Maximum Dimensions"
         ipipe = npipe
         IF(inode1 == 0) THEN
            node = node + 1
            inode1 = node
         END IF
         IF(inode2 == 0) THEN
            node = node + 1
            inode2 = node
         END IF
         P[ipipe].nd1 = inode1
         P[ipipe].nd2 = inode2
         P[ipipe].L =  L
         P[ipipe].D  = D
```

```
87          P[ipipe].CHW = CHW
88          P[ipipe].Qo = Qo
89       END IF
90       --Assign Solution
91       IF(AssignSolution==TRUE) THEN
92          H1 = P[ipipe].H1
93          H2 = P[ipipe].H2
94          Q =  P[ipipe].Q
95       END IF
96 END FUNCTION
97 COMPONENT Pipe
98    PORTS
99       IN Hydraulic h_in
100      OUT Hydraulic h_out
101   DATA
102      REAL L = 600      UNITS "m"    "Pipe length"
103      REAL D = 0.5      UNITS "m"    "Pipe inside diametert"
104      REAL CHW = 120.   UNITS "-"    "Hazen Williams Coefficient"
105      REAL Qo = 0.47744 UNITS "m3/s" "Initial flow"
106   DECLS
107      INTEGER ipipe = 0
108      DISCR REAL H1     UNITS "m"    "Piezometric head at inlet"
109      DISCR REAL H2     UNITS "m"    "Piezometric head at outlet"
110      DISCR REAL Q      UNITS "m3/s" "Volume flow"
111   INIT PRIORITY 100
112      InitPipe(L, D, CHW, Qo, ipipe, h_in.inode, h_out.inode, H1, H2, Q)
113 END COMPONENT
114
115 FUNCTION NO TYPE InitTank(
116       IN REAL H,
117       IN REAL Qo,
118       OUT INTEGER itank,
119       OUT INTEGER inode,
120       OUT REAL Q
121       )
122    BODY
123       --Obtain topology and fill global data
124       IF(itank == 0) THEN
125          ntank = ntank + 1
126          ASSERT (ntank <= MAX TANK) FATAL "Maximum number of tanks exceeded. \
127                                  Recompile library with increased Maximum Dimensions"
128          itank = ntank
129          IF(inode == 0) THEN
130             node = node + 1
131             inode = node
132          END IF
133          T[itank].nod = inode
134          T[itank].H = H
135          T[itank].Qo = Qo
136       END IF
137       --Assign Solution
138       IF(AssignSolution) THEN
139          Q = T[itank].Q
140       END IF
141       RETURN
142 END FUNCTION
143
144 COMPONENT Tank
145    PORTS
146       IN Hydraulic h_out
147    DATA
148       REAL H = 100.  UNITS "m" "Tank piezometric head"
149       REAL Qo = 0.1  UNITS "m3/s" "Initial guess of the tank outflow"
150    DECLS
151       INTEGER itank = 0
152       DISCR REAL Q
153    INIT PRIORITY 90
154       InitTank(H, Qo, itank, h_out.inode, Q)
155 END COMPONENT
156
157 FUNCTION NO_TYPE InitOutflow(
158       IN REAL Q,
```

```
159         OUT INTEGER node,
160         OUT INTEGER inode,
161         OUT REAL H
162      )
163      BODY
164         --Obtain topology and fill global data
165         IF(iflow == 0) THEN
166            nflow = nflow + 1
167            ASSERT (nflow <= MAX FLOW) FATAL "Maximum number of flows exceeded. \
168                                 Recompile library with increased Maximum Dimensions"
169            iflow = nflow
170            IF(inode == 0) THEN
171               node = node + 1
172               inode = node
173            END IF
174            OF[iflow].nod = inode
175            OF[iflow].Q = Q
176         END IF
177         --Assign solution
178         IF(AssignSolution) THEN
179            H = OF[iflow].H
180         END IF
181         RETURN
182   END FUNCTION
183
184   COMPONENT Outflow
185      PORTS
186         IN Hydraulic h_in
187      DATA
188         REAL Q = 0.     UNITS "m3/s" "Outflow"
189      DECLS
190         INTEGER iflow = 0
191         DISCR REAL H    UNITS "Piezometric head at the connected node"
192      INIT PRIORITY 90
193         InitOutflow(Q, iflow, h_in.inode, H)
194   END COMPONENT
195   FUNCTION NO TYPE FJacob(IN REAL Xunk[], IN REAL rdata[], OUT REAL F[], OUT
196           SPARSE_MATRIX Jacob)
197      DECLS
198         INTEGER nd1
199         INTEGER nd2
200         INTEGER iunkq
201         REAL H1, H2, Q
202         REAL Qmin
203         CONST REAL Vmin = 1e-4 "Min. velocity to avoid singular Jacobian at zero flow"
204      BODY
205         neq = node+npipe+ntank
206         FOR(j IN 1, neq)
207            F[j] = 0
208         END FOR
209         --Pipe Equations
210         FOR(j IN 1, npipe)
211            nd1 = P[j].nd1
212            nd2 = P[j].nd2
213            iunkq = node+j
214            P[j].H1 = Xunk[nd1]
215            P[j].H2 = Xunk[nd2]
216            P[j].Q  = Xunk[j+node]
217
218            F[nd1] = F[nd1] - P[j].Q
219            Jacob.stampItem(nd1, iunkq, -1)
220            --
221            F[nd2] = F[nd2] + P[j].Q
222            Jacob.stampItem(nd2, iunkq,  1)
223            --
224            Qmin = 0.785398163*P[j].D**2 * Vmin
225            F[node+j] = P[j].H1 - P[j].H2 - 10.67 * (P[j].L /(P[j].CHW**1.852*\
226                        P[j].D**4.8704))* P[j].Q * (abs(P[j].Q)+Qmin)**0.852
227            Jacob.stampItem(node+j, nd1,  1.)
228            Jacob.stampItem(node+j, nd2, -1.)
229            Jacob.stampItem(node+j, iunkq, -  10.67 * (P[j].L /(P[j].CHW**1.852*\
230                        P[j].D**4.8704))*((abs(P[j].Q)+Qmin)**0.852 \
```

```
                        +0.852*abs(P[j].Q)*(abs(P[j].Q)+Qmin)**(0.852-1)))
      END FOR
      --Tank Equations
      FOR(j IN 1, ntank)
         nd2 = T[j].nod
         T[j].Q = Xunk[node+npipe+j]
         F[nd2] = F[nd2] + T[j].Q
         Jacob.stampItem(nd2, node+npipe+j, 1)
         F[node+npipe+j] = Xunk[nd2] - T[j].H
         Jacob.stampItem(node+npipe+j, nd2, 1.)
      END FOR
      --Flow Equations
      FOR(j IN 1, nflow)
         nd1 = OF[j].nod
         OF[j].H = Xunk[nd1]
         F[nd1] = F[nd1] - OF[j].Q
      END FOR
END FUNCTION

COMPONENT Solver
   DATA
      REAL Ho = 100. UNITS "m" "Initial guess for the node piezometric head"
      INTEGER iter damp = 5    "Number of initial damped iterations"
      REAL FRAC damp = 0.1     "Damping factor for the damped iterations"
      INTEGER iter max = 20    "Maximum number of iterations in NR algorithm"
      INTEGER NR_debug = 1     "Debug level for Newton-Raphson"
      INTEGER STAMP_debug = 0  "Debug level for Stamping process"
   DECLS
      REAL tol = 1e-7
      HIDDEN DISCR REAL rdata[5]
   OBJECTS
      NLEQN_SYSTEM(MAX_EQN= MAX_EQN, MAX_STAMP = MAX_STAMP,
                                    MAX_NNZEROS = MAX_EQN*10) nleqn_system
   INIT PRIORITY  0
      IF(NOT AssignSolution) THEN
         nleqn system.NR debug = NR debug
         nleqn system.Jacob.STAMP_debug = STAMP_debug
         --number of equations
         nleqn_system.neqn = node + npipe + ntank
         --Initialization of the unknowns
         FOR(j IN 1, node)
            nleqn_system.Xunk[j] = Ho
         END FOR
         FOR(j IN 1, npipe)
            nleqn_system.Xunk[j+node] = P[j].Qo
         END FOR
         FOR(j IN 1, ntank)
            nleqn_system.Xunk[j+node+npipe] = T[j].Qo
         END FOR
         --Call to the solution procedure
         nleqn system.Solve(FJacob, iter_max, iter_damp, FRAC_damp, tol, rdata)
         AssignSolution = TRUE
         EXEC_INIT()
      END IF
END COMPONENT
```

F.2. Librería Estacionarios Hidráulicos con la Aproximación Clásica

Esta librería sigue enteramente el paradigma del MSOO. Los componentes implementan directamente las ecuaciones en el bloque **CONTINUOUS**, y el sistema de ecuaciones se construye combinando las ecuaciones de los componentes y del conexionado. La codificación de la librería solo ha requerido 57 líneas de código. La principal ventaja de la aproximación clásica es que el código es claro y conciso.

*Listado F-2: Listado de la Librería **HYDRAULIC_ST_CLASSIC***

```
LIBRARY HYDRAULIC_ST_CLASSIC
PORT Hydraulic
   SUM REAL Q
   EQUAL REAL H
END PORT

COMPONENT Pipe
   PORTS
      IN Hydraulic h_in
      OUT Hydraulic h_out
   DATA
      REAL L = 600       UNITS "m"    "Pipe length"
      REAL D = 0.5       UNITS "m"    "Pipe inside diametert"
      REAL CHW = 120.    UNITS "-"    "Hazen Williams Coefficient"
      REAL Qo = 0.47744 UNITS "m3/s" "Initial flow"
   DECLS
      REAL H1      UNITS "m"    "Piezometric head at inlet"
      REAL H2      UNITS "m"    "Piezometric head at outlet"
      REAL Q       UNITS "m3/s" "Volume flow"
      REAL Qmin    UNITS "m3/s" "Volume flow at Vmin"
      CONST REAL Vmin = 1.e-4 UNITS "m/s"  "Small velocity for DP linearization"
   INIT
       Q = Qo
   CONTINUOUS
       Qmin = 0.785398163*D**2 * Vmin
       0 = h_in.H - h_out.H - 10.67 * (L /(CHW**1.852 * D**4.8704))* Q * \
            (abs(Q)+Qmin)**0.852
       H1 = h_in.H
       H2 = h_out.H
       Q = h_in.Q
       Q = h_out.Q
END COMPONENT

COMPONENT Tank
   PORTS
      OUT Hydraulic h_out
   DATA
      REAL H = 100.  UNITS "m" "Tank piezometric head"
   DECLS
      REAL Q
   CONTINUOUS
       h_out.H = H
       Q = h_out.Q
END COMPONENT

COMPONENT Outflow
   PORTS
      IN Hydraulic h_in
   DATA
      REAL Q = 0. UNITS "m3/s" "Outflow"
   DECLS
      REAL H      UNITS "m" "Piezometric head at the connected node"
   CONTINUOUS
      h_in.Q = Q
      H = h_in.H
END COMPONENT
```

APÉNDICE G MÉTODO DE SOLUCIÓN DE TRANSITORIOS HIDRÁULICOS

Los libros de Wylie et al. (1993) y de Chaudhry (2014) presentan en detalle la derivación de las ecuaciones que describen los transitorios hidráulicos, así los métodos de solución más adecuados para dichas ecuaciones, estos son el método de las características aplicado a la solución de transitorios hidráulicos y el método implícito de cuatro puntos. En este apéndice, se presenta un resumen de las partes relevantes de estos libros, centrándose en las ecuaciones fundamentales del golpe de ariete y los dos métodos de solución mencionados.

G.1. Nomenclatura

Símbolo	Descripción	Unidades
A	área transversal de la tubería	m^2
A_v	área de salida o de referencia de la válvula	m^2
a	velocidad de propagación de las ondas de golpe de ariete	m/s
B	constante de compresibilidad de la tubería	s/m^2
BM	coeficiente en la ecuación de compatibilidad de las características C^-	s/m^2
BP	coeficiente en la ecuación de compatibilidad de las características C^+	s/m^2
CM	coeficiente en la ecuación de compatibilidad de las características C^-	m
CP	coeficiente en la ecuación de compatibilidad de las características C^+	m
C_d	coeficiente de descarga en la válvula de salida	*adim.*
C_r	Número de Courant	*adim.*
D	diámetro interno de la tubería	m
f	factor de fricción de Darcy	*adim.*
g	aceleración gravitacional	m/s^2
H	altura piezométrica ($= z + p\,/\rho g$)	m
H_v	altura de vapor	m
k	coeficiente de decaimiento de fricción para el cálculo de la fricción transitoria adimensional	*adim.*
L	Longitud de la tubería	m

p	presión estática	*N/m²*
Q	caudal volumétrico	*m³/s*
Q_U	caudal volumétrico en la entrada al nodo o sección	*m³/s*
Q_D	caudal volumétrico en la salida del nodo o sección	*m³/s*
R	Coeficiente de resistencia	*s²/ m⁵*
x	distancia a lo largo de la tubería	*m*
t	tiempo	*s*
V_v	Volumen de la cavidad de vapor	*m³*
z	elevación	*m*
ϵ	constante de linealización para el término de fricción	*adim.*
ξ	coeficiente de pérdida de presión	*adim.*
Ψ_J	variación permisible en la velocidad de propagación de la tubería J	*adim.*
ρ	densidad del líquido	*kg/m³*
τ	apertura relativa de la válvula	*adim.*

Subíndices

i	*i* –ésima sección o nodo
k	índice para tuberías conectadas por el extremo de entrada
j	índice para tuberías conectadas por el extremo de salida
Nj	última sección o número de tramos (dependiendo del contexto) de la j-ésima tubería

G.2. Ecuaciones del Golpe de Ariete

Las dos ecuaciones que describen el flujo transitorio unidimensional de un líquido en una tubería son la conservación de la masa y la conservación del momento:

$$Masa: \qquad L_1 = \frac{a^2}{g\,A}\frac{\partial Q}{\partial x} + \frac{\partial H}{\partial t} = 0 \tag{G-1}$$

$$Momento: \quad L_2 = \frac{1}{g\,A}\frac{\partial Q}{\partial t} + \frac{\partial H}{\partial x} + \frac{f}{2g\,D}\frac{Q|Q|}{A^2} + \frac{k}{g\,A}\left(\frac{\partial Q}{\partial t} + a\,\mathrm{sign}(Q)\left|\frac{\partial Q}{\partial x}\right|\right) = 0 \tag{G-2}$$

Los dos últimos términos en el lado izquierdo de la ecuación del momento son, respectivamente, el término de fricción estacionaria y el término de fricción transitoria. El término de fricción transitoria se calcula con la fórmula de Darcy. El término de fricción transitoria sigue el modelo de aceleración instantánea modificado de Vítkovský et al.

(2006), el cual fue seleccionado entre los diferentes modelos de fricción transitoria disponibles en la literatura debido a su simplicidad y amplio uso.

En la ecuación de continuidad, a es la velocidad de la onda del golpe de ariete, la cual puede calcularse mediante la siguiente fórmula:

$$a = \frac{\sqrt{\frac{K}{\rho}}}{\sqrt{1 + \left[\frac{C(\mu) + 2\frac{e}{D}(1+\mu)(1+e/D)}{1+e/D}\right]\frac{K\,D}{E\,e}}} \tag{G-3}$$

donde $C(\mu)$ es un coeficiente que depende del coeficiente de Poisson del material, μ, y las condiciones de soporte de la tubería: $C(\mu) = 1 - \mu/2$ para una tubería solo anclada en su extremo aguas arriba, $C(\mu) = 1 - \mu^2$ para una tubería anclada contra cualquier movimiento axial, y $C(\mu) = 1$ para una tubería anclada con juntas de expansión en todo su recorrido.

G.3. Método de las Características

El método de características transforma los sistemas originales de ecuaciones diferenciales en derivadas parciales en cuatro ecuaciones diferenciales ordinarias:

$$C^+ \begin{cases} \frac{dH}{dt} + \frac{a}{g\,A}\frac{dQ}{dt} + \frac{a}{g}\left[\frac{f}{2D}\frac{Q|Q|}{A^2} + \frac{k}{A}\left(\frac{\partial Q}{\partial t} + a\,\text{sign}(Q)\left|\frac{\partial Q}{\partial x}\right|\right)\right] = 0 & \text{(G-4)} \\ \frac{dx}{dt} = a & \text{(G-5)} \end{cases}$$

$$C^- \begin{cases} \frac{dH}{dt} - \frac{a}{g\,A}\frac{dQ}{dt} - \frac{a}{g}\left[\frac{f}{2D}\frac{Q|Q|}{A^2} + \frac{k}{A}\left(\frac{\partial Q}{\partial t} + a\,\text{sign}(Q)\left|\frac{\partial Q}{\partial x}\right|\right)\right] = 0 & \text{(G-6)} \\ \frac{dx}{dt} = -a & \text{(G-7)} \end{cases}$$

Las ecuaciones (G-4) y (G-6) son las llamadas ecuaciones de compatibilidad y son válidas, respectivamente, a lo largo de las líneas características definidas por las ecuaciones (G-5) y (G-7). La Figura G-1 muestra estas líneas características, denotadas como C^+ con una pendiente $+1/a$ $(dx/dt = +a)$ y C^- con una pendiente $-1/a$ $(dx/dt = -a)$, en el plano de variables independientes (es decir, el plano $x - t$). Las líneas características son rectas con pendiente $\pm 1/a$.

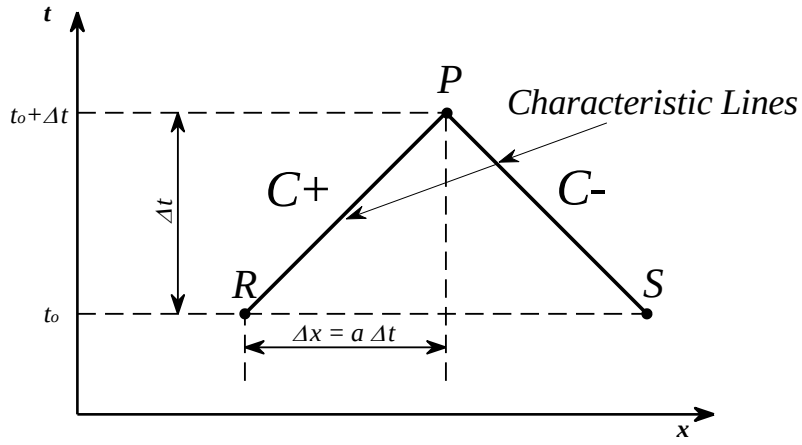


Figura G-1: Líneas Características en el Plano x-t Figura A 1:

Integrando la ecuación (G-4) entre los puntos R and P, e integrando de forma similar la ecuación (G-6) entre los puntos S y P, se obtienen las ecuaciones de compatibilidad en la siguiente forma:

$$H_P^{t+\Delta t} = CP - BP \cdot Q_P^{t+\Delta t} \quad \text{(G-8)}$$

$$H_P^{t+\Delta t} = CM + BM \cdot Q_P^{t+\Delta t} \quad \text{(G-9)}$$

Donde:

$$CP = H_R^t + Q_R^t\,[B - R\,|Q_R^t|\,(1-\varepsilon)] - \frac{Bk}{2}\left(-Q_P^{t-\Delta t} + \text{sign}(Q_R^t)|Q_S^t - Q_R^t|\right)$$

$$CM = H_S^t + Q_S^t\,[B - R\,|Q_S^t|\,(1-\varepsilon)] - \frac{Bk}{2}\left(-Q_P^{t-\Delta t} + \text{sign}(Q_S^t)|Q_S^t - Q_R^t|\right)$$

$$BP = B\left(1+\frac{k}{2}\right) + \varepsilon\, R\,|Q_R^t|\,, \qquad BM = B\left(1+\frac{k}{2}\right) + \varepsilon\, R\,|Q_S^t|$$

$$B = a/(gA)\,,\ R = f\,a\,\Delta t/(2g\,DA^2),$$

ε es una constante para la linealización del termino de fricción introducida por Karney and McInnis (1990). Debe estar en el rango $(0 \le \varepsilon \le 1)$, y el valor recomendado es 0.81.

Suponiendo que se conocen la altura piezométrica y el flujo en los puntos R y S de la Figura G-1, es posible resolver las ecuaciones (G-8) y (G-9) para obtener la altura piezométrica $H_P^{t+\Delta t}$ y el flujo $Q_P^{t+\Delta t}$ en el nuevo paso de tiempo en el punto P, obteniéndose:

$$Q_P^{t+\Delta t} = \frac{CP - CM}{BP + BM} \quad \text{(G-10)}$$

$$H_P^{t+\Delta t} = \frac{CP \cdot BM + CM \cdot BP}{BP + BM} \qquad \text{(G-11)}$$

Para aplicar el MOC a una tubería, es necesario considerar una malla rectangular tal y como la que se muestra en la Figura G-2. La longitud de la tubería se divide en *N* intervalos de igual longitud, cada uno de longitud $x = L/N$, y el tiempo se divide en intervalos de tiempo iguales, Δt. Los bordes de los intervalos de longitud se suelen designar en la literatura como "nodos" o "secciones". En el tiempo inicial, $t = 0$, se asume que la altura piezométrica y el flujo en los nodos son conocidos a partir de una solución en estado estacionario. El proceso de solución consiste en encontrar H y Q para los puntos de la malla en $t = \Delta t$, luego en $t = 2\Delta t$ y así sucesivamente, hasta alcanzar la duración deseada. Las líneas características que pasan por el punto P, un punto típico de la malla en el próximo paso de tiempo, intersectan el eje x en R y S. Aunque la altura piezométrica y el flujo solo se conocen en los puntos de la malla, es posible interpolar sus valores en R y S mediante interpolación lineal o de orden superior.

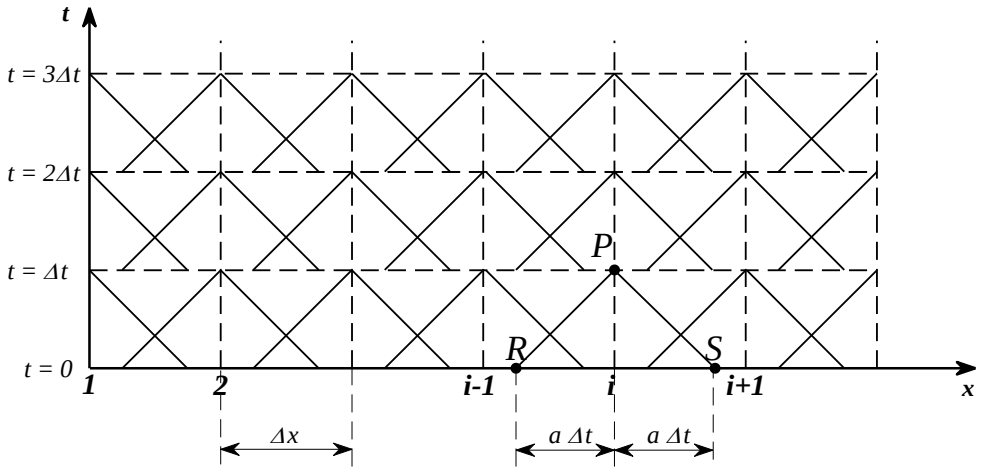


Figura G-2: Malla Rectangular para la Aplicación del MOC a una tubería

Para garantizar la estabilidad del método, el paso de tiempo debe cumplir la condición de que el número de Courant sea menor o igual que 1.

$$C_r = a\ \Delta t\ /\Delta x \leq 1 \qquad \text{(G-12)}$$

La selección más obvia para el paso de tiempo es obligar a que el número de Courant sea igual a 1, de esta manera los puntos R y S coinciden con los puntos de la cuadrícula en el paso de tiempo anterior y el MOC es muy preciso. No obstante, cumplir con $C_r = 1$ para

cada tubería de un modelo plantea presenta un desafío, ya que se requiere que el paso de tiempo sea el mismo para todas las tuberías común.

Las tuberías individuales tienen propiedades diferentes y, en principio, solo se puede modificar el número de nodos en cada tubería, que es un valor entero. Esto solo permitiría igualar los incrementos de tiempo cuando las relaciones entre los tiempos que tarda una onda en viajar de un lado a otro de una tubería puedan expresarse mediante relaciones entre enteros.

Este problema de discretización puede abordarse mediante ajustes artificiales de la velocidad o la longitud de la onda, o mediante técnicas de interpolación, ya sea en el espacio, en el tiempo o en ambos. Karney y Ghidaoui (1997) desarrollaron un algoritmo de discretización que permite la combinación de varios de estos métodos, incluyendo el ajuste de la velocidad de onda, el ajuste de la longitud y varios tipos de interpolación lineal.

G.3.1. Condiciones de Contorno:

En un extremo de una tubería, se tienen dos incógnitas (H, Q), pero solo se dispone de una ecuación de compatibilidad. En el extremo de entrada, la ecuación (G-6) se cumple a lo largo de la línea característica C^-, y en el extremo de salida, la ecuación (G-4) se cumple a lo largo de la característica C^+. Por lo tanto, se hace necesario contar con ecuaciones adicionales proporcionadas por los componentes conectados a los extremos de las tuberías para calcular ambas incógnitas en cada tubería conectada. Debido a su función en la solución del problema matemático, a los componentes que no son tuberías y están conectados en los extremos de las tuberías se les denomina "*Condiciones de Contorno*" o simplemente "*Contornos*".

Tanques o Depósitos

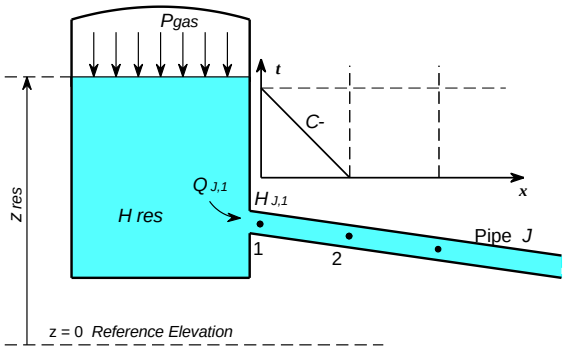


Figura G-3: Condición de Contorno Tanque o Depósito

En el caso de un tanque o depósito conectado al extremo de entrada de una tubería, es decir, conectado por el nodo 1 como se muestra en la Figura G-3, se tienen dos variables

desconocidas: la altura y caudal en el nodo 1, denotadas como $H_{j,1}^{t+\Delta t}$ y $Q_{j,1}^{t+\Delta t}$, respectivamente. La altura piezométrica dentro del tanque, H_{res}, puede calcularse a partir de la presión absoluta en la zona de gas, P_{gas}, y el nivel dentro del mismo, z_{res}, utilizando la siguiente ecuación:

$$H_{res} = \frac{\left(P_{gas} - P_{atmospheric}\right)}{\rho\, g} + z_{res} \tag{G-13}$$

En el caso de los tanques o depósitos, se suele suponer que la altura piezométrica dentro de los mismos permanece constante durante el tiempo de duración del transitorio, lo cual es válido cuando el volumen del tanque o depósito es grande, como suele ser el caso habitual.

Al aplicar el principio Bernouilli entre un punto dentro de la zona líquida del tanque y el nodo 1 de la tubería, considerando tanto la presión dinámica como las caídas de presión en la conexión con la tubería, se obtiene la siguiente ecuación:

$$H_{res} = H_{j,1}^{t+\Delta t} + \frac{\left(Q_{j,1}^{t+\Delta t}\right)^2}{2\, g\, A_j^2} + \xi\, \frac{Q_{j,1}^{t+\Delta t}\left|Q_{j,1}^{t+\Delta t}\right|}{2\, g\, A_j^2} \tag{G-14}$$

donde ξ es el coeficiente de caída de presión, que puede depender de la dirección del flujo, $\xi = \xi_+$ para flujo entrante en la tubería ($Q_{j,1}^{t+\Delta t} \geq 0$), y $\xi = \xi_-$ para flujo saliente de la tubería al tanque o depósito ($Q_{j,1}^{t+\Delta t} < 0$).

Al haber dos incógnitas, se requiere una ecuación adicional, que es la ecuación compatibilidad a lo largo de la característica negativa C^-:

$$H_{j,1}^{t+\Delta t} = CM_{j,1} + BM_{j,1}\, Q_{j,1}^{t+\Delta t} \tag{G-15}$$

Para resolver estas dos ecuaciones, (G-14) y (G-15), se elimina $H_{j,1}^{t+\Delta t}$, y simplificando resulta:

$$\frac{1+\xi_+}{2\, g\, A_j^2\, BM_{j,1}}\left(Q_{j,1}^{t+\Delta t}\right)^2 + Q_{j,1}^{t+\Delta t} + \left(\frac{CM_{j,1} - H_{res}}{BM_{j,1}}\right) = 0 \quad \boldsymbol{si}\ Q_{j,1}^{t+\Delta t} \geq 0 \tag{G-16}$$

$$\frac{1-\xi_-}{2\, g\, A_j^2\, BM_{j,1}}\left(Q_{j,1}^{t+\Delta t}\right)^2 + Q_{j,1}^{t+\Delta t} + \left(\frac{CM_{j,1} - H_{res}}{BM_{j,1}}\right) = 0 \quad \boldsymbol{si}\ Q_{j,1}^{t+\Delta t} < 0 \tag{G-17}$$

Es evidente que el signo del caudal depende del signo de la diferencia $\mathrm{CM_{j,1}} - H_{res}$, lo que resulta en las siguientes soluciones para $Q_{\mathrm{j,1}}^{t+\Delta t}$:

$$Q_{j,1}^{t+\Delta t} = \frac{-1 + \sqrt{1 - 4\left(\frac{1+\xi_+}{2\,g\,A_j^2\,BM_{j,1}}\right)\left(\frac{CM_{j,1} - H_{res}}{BM_{j,1}}\right)}}{\frac{1+\xi_+}{2\,g\,A_j^2\,BM_{j,1}}} \qquad \boldsymbol{para}\ H_{res} \geq CM_{j,1} \tag{G-18}$$

$$Q_{j,1}^{t+\Delta t} = \frac{-1 + \sqrt{1 - 4\left(\frac{1-\xi_-}{2\,g\,A_j^2\,BM_{j,1}}\right)\left(\frac{CM_{j,1} - H_{res}}{BM_{j,1}}\right)}}{\frac{1-\xi_-}{2\,g\,A_j^2\,BM_{j,1}}} \qquad \boldsymbol{para}\ H_{res} < CM_{j,1} \tag{G-19}$$

Una vez conocido el caudal, es inmediato obtener la altura usando la ecuación (G-15).

Flujos Especificado

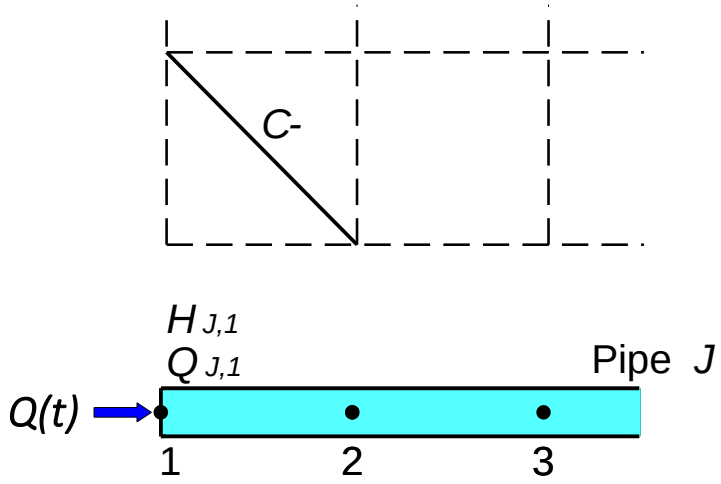


Figura G-4: Condición de Contorno de Flujo

En el caso de una condición de contorno que impone el flujo en la entrada de una tubería, tal como se muestra en la Figura G-4, se tiene:

$$Q_{j,1}^{t+\Delta t} = Q(t) \tag{G-20}$$

En ese caso, la ecuación de compatibilidad a lo largo de la característica negativa C^-, permite obtener la altura piezométrica de forma directa:

$$H_{j,1}^{t+\Delta t} = CM_{j,1} + BM_{j,1}\, Q_{j,1}^{t+\Delta t} \qquad \text{(G-21)}$$

Válvulas de Salida

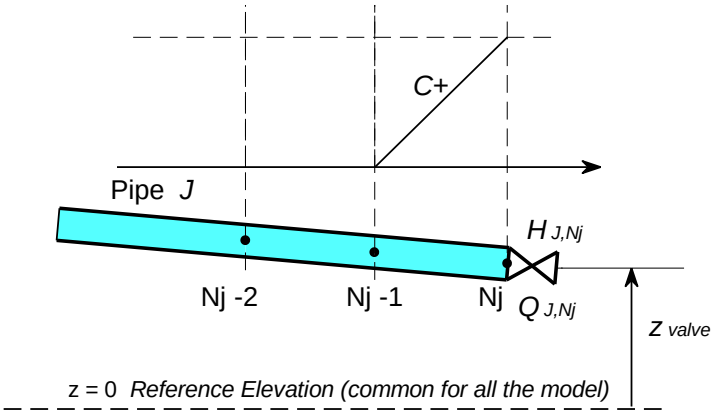


Figura G-5: Condición de Contorno Válvula de Salida

Considerando, una válvula de salida colocada en el extremo de salida de la tubería, esto es conectada al último nodo Nj, como se muestra en la Figura G-5, y con un área de tobera equivalente $A_g(t)$, que puede ser variable en el tiempo. Aplicando el principio de Bernouilli entre la entrada y salida de la válvula, se obtiene:

$$H_{j,Nj}^{t+\Delta t} + \frac{\left(Q_{j,Nj}^{t+\Delta t}\right)^2}{2\, g\, A_j^2} = \frac{P_g}{\rho\, g} + z_{Nj} + \frac{\left(Q_{j,Nj}^{t+\Delta t}\right)^2}{2\, g\, A_g^2} \qquad \text{(G-22)}$$

donde $\mathrm{P_g}$ es la presión en la garganta de la válvula relativa a la atmosférica. Normalmente, esta presión es nula, pero puede ser no nula en el caso de agua caliente con una presión de vapor mayor que la atmosférica. En esta situación, el agua se evapora rápidamente al salir por la válvula, un proceso conocido como "*flashing*", y en la garganta se mantiene una presión próxima a la de vapor.

La segunda ecuación a considerar es la ecuación de compatibilidad a lo largo de la característica C^+:

$$H_{j,Nj}^{t+\Delta t} = CP_{j,Nj} - BP_{j,Nj}\, Q_{j,Nj}^{t+\Delta t} \qquad \text{(G-23)}$$

Utilizando la ecuación (G-23) para eliminar $H_{j,Nj}^{t+\Delta t}$ en (G-22), se obtiene:

$$\frac{1}{2\,g}\left(1\ -\frac{A_g^2}{A_j^2}\right)\left(Q_{j,Nj}^{t+\Delta t}\right)^2+BP_{j,Nj}A_g^2\,Q_{j,Nj}^{t+\Delta t}+A_g^2\left(\frac{P_g}{\rho\,g}+\ z_{Nj}-CP_{j,Nj}\right)=0 \qquad \text{(G-24)}$$

Resolviendo la ecuación de segundo grado anterior, y considerando la única raíz con sentido físico, se obtiene:

$$Q_{j,1}^{t+\Delta t}=\frac{-BP_{j,Nj}A_g^2+\sqrt{\left(BP_{j,Nj}A_g^2\right)^2-\frac{2}{g}\left(1\ -\frac{A_g^2}{A_j^2}\right)A_g^2\left(\frac{P_g}{\rho\,g}+\ z_{Nj}-CP_{j,Nj}\right)}}{\frac{1}{g}\ \left(1\ -\frac{A_g^2}{A_j^2}\right)} \qquad \text{(G-25)}$$

Obsérvese que se debe cumplir $CP_{j,Nj}-\frac{P_g}{\rho\,g}-\ z_{Nj}>0$ para que el flujo por la válvula sea saliente. En caso de no cumplirse la condición, entraría aire en la tubería y la formulación dejaría de ser válida.

Válvulas de Dos Vías

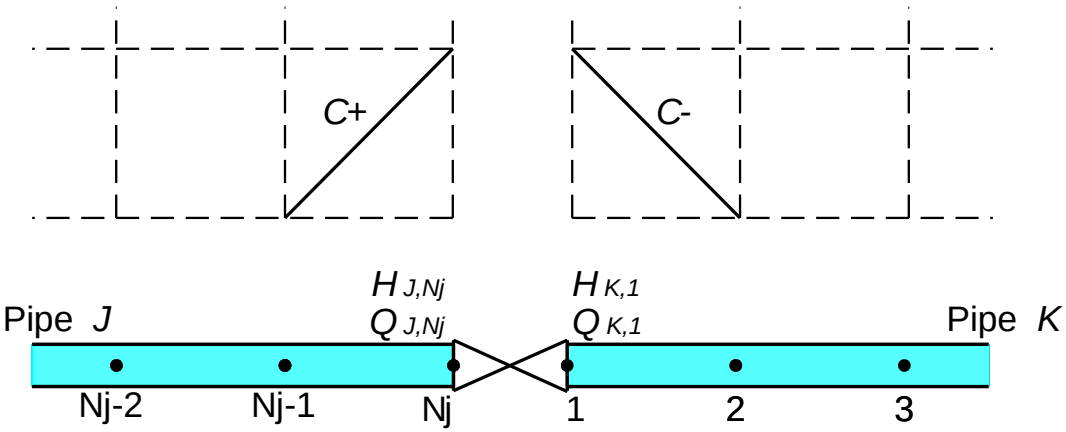


Figura G-6: Condición de Contorno Válvula de Dos Vías

La Figura G-6 representa una válvula ubicada entre la última sección o nodo de la tubería *j-ésima* y el primer nodo de la tubería *k-ésima*. En la válvula se considera que no puede acumularse agua, por lo que la ecuación de continuidad en la misma es:

$$Q_{j,N_j}\ =\ Q_{k,1} \qquad \text{(G-26)}$$

La ecuación de caída de presión en la válvula es la siguiente:

$$H_{j,Nj}^{t+\Delta t} + \frac{\left(Q_{j,Nj}^{t+\Delta t}\right)^2}{2\ g\ A_j^2} = H_{k,1}^{t+\Delta t} + \frac{\left(Q_{k,1}^{t+\Delta t}\right)^2}{2\ g\ A_k^2} + \xi_v \frac{Q_{k,1}^{t+\Delta t}\left|Q_{k,1}^{t+\Delta t}\right|}{2\ g\ A_v^2} \quad \text{(G-27)}$$

donde ξ_v es el coeficiente de caída de presión en la misma referido a un área de referencia de la válvula A_v.

También se tienen las ecuaciones características en las tuberías de entrada y salida:

$$H_{j,Nj}^{t+\Delta t} = CP_{j,Nj} - BP_{j,Nj}\ Q_{j,Nj}^{t+\Delta t} \quad \text{(G-28)}$$

$$H_{k,1}^{t+\Delta t} = CM_{k,1} + BM_{k,1}\ Q_{k,1}^{t+\Delta t} \quad \text{(G-29)}$$

En total hay 4 ecuaciones y 4 incógnitas. Para resolver las ecuaciones se eliminan las alturas en (G-27) utilizando las ecuaciones características (G-28) y (G-29), resultando la siguiente ecuación de segundo grado para el caudal:

$$\frac{Q^2}{2\ g}\left(\frac{1}{A_j^2} - \frac{1}{A_k^2}\right) - \xi_v \frac{Q\ |Q|}{2\ g\ A_v^2} - \left(BP_{j,Nj} + BM_{k,1}\right) \mathrm{Q} + \left(CP_{j,Nj} - CM_{k,1}\right) = 0 \quad \text{(G-30)}$$

Despejando el caudal se obtiene:

$$Q = \frac{\left(BP_{j,Nj} + BM_{k,1}\right) - \sqrt{\left(BP_{j,Nj} + BM_{k,1}\right)^2 - \frac{2}{g}\left(\frac{1}{A_j^2} - \frac{1}{A_k^2} - \frac{\xi_v}{A_v^2}\right)\left(CP_{j,Nj} - CM_{k,1}\right)}}{\frac{1}{g}\left(\frac{1}{A_j^2} - \frac{1}{A_k^2} - \frac{\xi_v}{A_v^2}\right)} \quad \text{(G-31)}$$

Colector

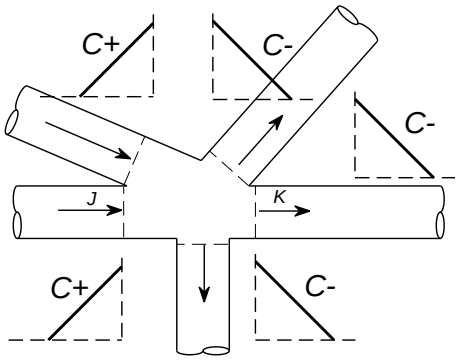


Figura G-7: Condición de Contorno Colector

La Figura G-7 representa un colector, un punto de convergencia de varias tuberías donde se asume que no puede acumularse líquido. SJ es el conjunto de todas las tuberías conectadas por su último nodo y SK es el conjunto de todas las tuberías conectadas por el nodo 1.

Para todas las tuberías pertenecientes al conjunto SJ, se aplica la ecuación característica de compatibilidad C^+, mientras que para todas las tuberías del conjunto SK se aplica la ecuación característica de compatibilidad C^-:

$$H_{j,Nj}^{t+\Delta t} = CP_{j,Nj} - BP_{j,Nj}\, Q_{j,Nj}^{t+\Delta t} \qquad \forall\ j \in SJ \tag{G-32}$$

$$H_{k,1}^{t+\Delta t} = CM_{k,1} + BM_{k,1}\, Q_{k,1}^{t+\Delta t} \qquad \forall\ k \in SK \tag{G-33}$$

Si n es el número total de tuberías conectadas, hay $2n$ incógnitas (el flujo y la altura piezométrica al final de cada tubería conectada). Las ecuaciones características de compatibilidad anteriores proporcionan n ecuaciones. Por lo tanto, el colector debe proporcionar las n ecuaciones restantes. Estas ecuaciones restantes son la ecuación de continuidad, que requiere que la suma de los flujos que entran al nodo sea igual a la suma de los flujos que salen del nodo:

$$\sum_{j \in SJ} Q_{j,Nj} - \sum_{k \in SK} Q_{k,1} = 0 \tag{G-34}$$

Y las $n-1$ ecuaciones de igualdad de la altura piezométrica total en el colector, H_{col}:

$$\begin{aligned} H_{col} &= H_{j,Nj}^{t+\Delta t} + \frac{\left(Q_{j,Nj}^{t+\Delta t}\right)^2}{2\,g\,A_j^2} + \xi_j \frac{Q_{j,Nj}^{t+\Delta t}\left|Q_{j,Nj}^{t+\Delta t}\right|}{2\,g\,A_j^2} \qquad \forall\ j \in SJ \\ H_{col} &= H_{k,1}^{t+\Delta t} + \frac{\left(Q_{k,1}^{t+\Delta t}\right)^2}{2\,g\,A_k^2} + \xi_k \frac{Q_{k,1}^{t+\Delta t}\left|Q_{k,1}^{t+\Delta t}\right|}{2\,g\,A_k^2} \qquad \forall\ k \in SK \end{aligned} \tag{G-35}$$

En el caso del colector, es muy común asumir que las pérdidas de carga en las conexiones son despreciables y que las alturas dinámicas son en todos los conductos son también despreciables. Con estas suposiciones, todas las ecuaciones pueden resolverse fácilmente, ya que son lineales. La ecuación obtenida para la altura en el colector es:

$$H_{col} = \left[\sum_{j \in SJ}\left(\frac{CP_{j,Nj}}{B_{j,Nj}}\right) + \sum_{k \in SK}\left(\frac{CM_{k,1}}{BM_{k,1}}\right)\right] \Big/ \sum_{j \in SJ}\left(\frac{1}{B_{j,Nj}}\right) + \sum_{k \in SK}\left(\frac{1}{BM_{k,1}}\right) \tag{G-36}$$

Una vez obtenida la altura en el colector, las ecuaciones de compatibilidad permiten obtener el caudal en cada una de las tuberías conectadas.

Bomba

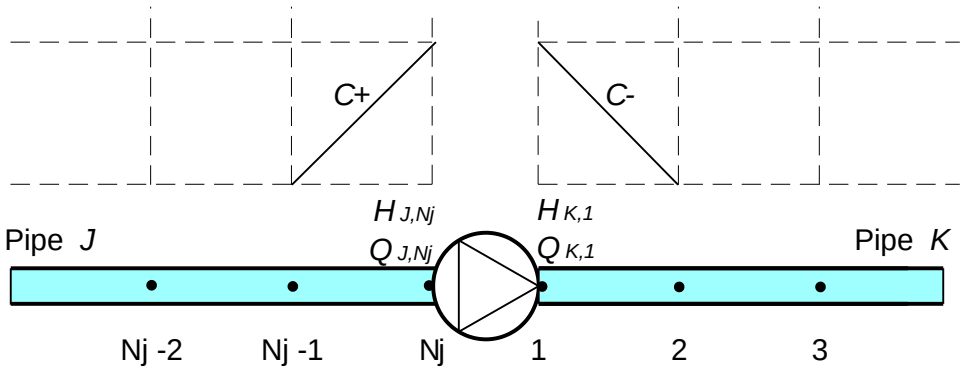


Figura G-8: Condición de Contorno Bomba

La Figura G-8 representa una bomba de velocidad variable ubicada entre la última sección o nodo de la tubería *j-ésima* y el primer nodo de la tubería *k-ésima*.

En la actualidad, la información sobre el comportamiento dinámico de las bombas es muy escasa, por lo tanto, se asume habitualmente que las curvas de operación en estado estacionario de las bombas son también aplicables durante la operación transitoria. Hay cuatro cantidades que describen el rendimiento en estado estacionario de la bomba: el caudal, Q; la altura de bombeo o altura dinámica total, ΔH; la velocidad de rotación, N; y el par de torsión del eje, T. Solo dos de las cuatro variables son independientes, Q y N. Por lo tanto, ΔH y T pueden expresarse como funciones de dos variables.

$$\Delta H = F(Q, N) \tag{G-37}$$

$$T = G(Q, N) \tag{G-38}$$

Los valores de Q, ΔH, N y T en el punto de máxima eficiencia se denominan condiciones nominales ("*rated*", en inglés) y se denotan con el subíndice R. Es conveniente trabajar con parámetros adimensionales referidos a este punto de máxima eficiencia:

$$v = \frac{Q}{Q_R}\ ; \quad h = \frac{\Delta H}{\Delta H_R}\ ; \quad \alpha = \frac{N}{N_R}\ ; \quad \beta = \frac{T}{T_R} \tag{G-39}$$

La aplicación del análisis dimensional a las ecuaciones permite ver que las curvas de funcionamiento de la bomba se pueden expresar mediante las siguientes relaciones homogéneas adimensionales:

$$\frac{h}{\alpha^2} = f\left(\frac{v}{\alpha}\right) \tag{G-40}$$

$$\frac{h}{\alpha^2} = g\left(\frac{v}{\alpha}\right) \tag{G-41}$$

Una bomba opera dentro de su zona de funcionamiento normal cuando $Q > 0$, $\Delta H > 0$, $N > 0$ y $T > 0$. Sin embargo, ocasionalmente las bombas pueden operar en zonas atípicas. Por ejemplo, después de una parada o detención de la bomba, puede ocurrir la inversión del flujo ($Q \leq 0$) y de la velocidad ($N \leq 0$), mientras no se cierren las válvulas de aislamiento situadas en la descarga de la bomba. También se pueden producir situaciones en las que una bomba gira en molinete ($Q > 0$, $\Delta H < 0$, $N > 0$ y $T \leq 0$) en un sistema con bombas en serie.

Las ecuaciones (G-40) y (G-30) no son adecuadas para el cálculo con programas de ordenador. Tanto v como α pueden cambiar de signo y cruzar el punto cero durante un transitorio, lo que provoca que v/α pueda tomar valores desde $-\infty$ hasta $+\infty$. Para abordar esta dificultad, se introduce una representación polar, que introduce una nueva variable angular θ y dos parámetros adimensionales $W_H\,(\theta)$ y $W_B\,(\theta)$:

$$\theta = \pi + tan^{-1}\left(\frac{v}{\alpha}\right) \tag{G-42}$$

$$\frac{h}{\alpha^2 + v^2} = W_H(\theta) \tag{G-43}$$

$$\frac{\beta}{\alpha^2 + v^2} = W_B(\theta) \tag{G-44}$$

La variable θ varía en el intervalo de 0 a $2\,\pi$, y los parámetros $W_H\,(\theta)$ y $W_B\,(\theta)$ son singulares solo cuando α y v son simultáneamente iguales a cero. En consecuencia, resulta sencillo definir las funciones $W_H\,(\theta)$ y $W_B\,(\theta)$ mediante tablas o curvas.

Los fabricantes de bombas solo suelen proporcionar las curvas de $\Delta H(Q)$ y $T(Q)$ en la zona de funcionamiento normal y a velocidad constante. Dicha zona de operación normal es aproximadamente equivalente al rango de θ entre π y 1.3π. En la mayoría de las situaciones prácticas, no se suministran curvas de funcionamiento fuera de la operación normal. Para superar esta dificultad, se asume que bombas con las mismas velocidades de rotación específicas $N\sqrt{Q}/H^{3/4}$ tienen curvas $W_H\,(\theta)$ y $W_B\,(\theta)$ muy similares. Por lo tanto, se recurre a extender las curvas de la bomba más allá de la zona de operación normal utilizando las curvas de una bomba de velocidad específica similar para la cual se hayan proporcionado las curvas completas de operación en la literatura.

En cualquier caso, una vez que se conocen las curvas homólogas de la bomba, hay 5 incógnitas en el nuevo paso de tiempo: la altura piezométrica y el caudal en la entrada y

salida de la bomba, y la velocidad de la bomba, que es una variable dinámica cuya derivada depende del desequilibrio entre el par del motor impulsor y el par de la bomba.

$$I \frac{2\pi}{60} \frac{dN}{dt} = T_{Motor} - T \quad \text{(G-45)}$$

Por lo general, el par del motor es una variable conocida. Por ejemplo, en el caso de una parada de la bomba, se supone que el par del motor es nulo después de la parada de la bomba.

Las otras ecuaciones necesarias para resolver la bomba son la ecuación de balance de altura:

$$H_{j,Nj} + \frac{Q_{j,NS}^2}{2\,g\,A_j^2} + \Delta H = H_{K,1} + \frac{Q_{k,1}^2}{2\,g\,A_k^2} \quad \text{(G-46)}$$

La continuidad del flujo a través de la bomba:

$$Q_{j,Nj} = Q_{k,1} \quad \text{(G-47)}$$

Y las dos ecuaciones características en la entrada y en la salida de la bomba:

$$H_{j,Nj}^{t+\Delta t} = CP_{j,Nj} - BP_{j,Nj}\, Q_{j,Nj}^{t+\Delta t} \quad \text{(G-48)}$$

$$H_{k,1}^{t+\Delta t} = CM_{k,1} + BM_{k,1}\, Q_{k,1}^{t+\Delta t} \quad \text{(G-49)}$$

Entonces, hay 5 ecuaciones para calcular las 5 incógnitas. Reemplazando la derivada de la velocidad de la bomba por una expresión de diferencia finita, se transforman en un conjunto algebraico que puede resolverse mediante el método de Newton-Raphson.

Otras Condiciones de Contorno

En esta sección se han abordado las formulaciones de las condiciones de contorno que corresponden a los elementos hidráulicos más usuales. No obstante, es importante señalar que existen diversas condiciones de contorno adicionales, especialmente aquellas asociadas a dispositivos diseñados para mitigar el golpe de ariete, como válvulas de entrada y salida de aire, chimeneas de equilibrio y acumuladores cargados con gas. Se recomienda al lector consultar la literatura sobre el golpe de ariete mencionada al inicio de este apéndice para comprender mejor su funcionamiento y formulación.

G.3.2. Separación de Columnas y Modelo de Cavitación Discreta

Las ecuaciones del golpe de ariete (G-1) y (G-2) son válidas mientras la presión es mayor que la presión de vapor. Si la presión cae por debajo de la presión de vapor, ocurre la formación de vapor. Las burbujas de vapor pueden dispersarse físicamente de manera

homogénea, agruparse en grandes bolsas o manifestarse como una combinación de ambos fenómenos. Dependiendo de la geometría del sistema y de la caída de presión (magnitud y gradiente), las cavidades pueden llegar a ser tan grandes como para llenar toda la sección transversal del tubo y así dividir el líquido en dos columnas. Este fenómeno se llama separación de columnas. Cuando la presión se recupera, el agua rellena la cavidad o cavidades causadas por el vapor y termina chocando con los elementos que confinan las cavidades, como otra masa de agua o la pared de la tubería, lo que resulta en un aumento de presión, que puede ser bastante severo. Durante una separación de columnas, se presenta tanto vacío como fuertes aumentos de presión, una combinación que puede provocar daños importantes.

Durante una separación de columna, el flujo que inicialmente es de una fase se convierte en un flujo de dos fases (líquido/vapor). Por lo tanto, las ecuaciones estándar de golpe de ariete, (G-1) y (G-2), dejan de ser aplicables. A pesar de esta dificultad, es necesario simular la separación de columnas transitorias debido a sus posibles consecuencias graves.

Aunque el fenómeno de la separación de columnas no se comprende completamente, se han desarrollado una variedad de algoritmos numéricos para modelarlo.

Modelo de Cavitación Discreta (DVCM) para la Separación de Columnas

El modelo DVCM (Discrete Vapor Cavity Model) es el modelo más comúnmente utilizado para la separación de columnas y la cavitación distribuida en la actualidad. Una ventaja significativa del DVCM es que se puede implementar fácilmente y que reproduce muchas características de los eventos físicos de separación de columnas en tuberías

Se permite que se empiezan a formar cavidades o burbujas en cualquiera de las secciones (o nodos) cuando la presión en la sección baja por debajo de la presión de vapor del líquido. El DVCM no diferencia específicamente entre cavidades de vapor localizadas y cavidades de vapor distribuidas vaporosa distribuida. Las cavidades de vapor están confinadas a las secciones computacionales, y se asume la velocidad de las ondas de presión es constante para el líquido entre secciones computacionales.

Cuando la presión en una sección computacional cae por debajo de la presión de vapor del líquido, la sección se trata como una condición de contorno interna fija. La presión se fija igual a la presión de vapor del líquido hasta que la cavidad se colapsa. El flujo volumétrico aguas arriba, Q_U se calculan utilizando la ecuación características C^+, (G-8) y C^- (G-9) con $H = H_{vap}$, siendo H_{vap}, la altura piezométrica en la sección que corresponde a la presión de vapor. Una vez obtenidos, los caudales de entrada y salida a la sección, la ecuación de continuidad permite calcular el volumen de la cavidad de vapor:

$$V_{cav} = \int_0^t (Q_D - Q_U)\, dt \qquad \text{(G-50)}$$

G.4. Método de Diferencias Finitas Implícito

Otro método comúnmente utilizado para la solución de transitorios hidráulicos es el método de diferencias finitas implícitas, también conocido como el esquema de Preissmann o el esquema de cuatro puntos o el esquema de caja (Twyman, 2004). Dicho método considera dos coeficientes de ponderación que pueden variar entre 0. y 1. Cuando tales coeficientes son iguales a 0.5, se tiene el esquema implícito de cuatro puntos centrado, el cual es de segundo orden en el tiempo y en el espacio y es incondicionalmente estable con cualquier incremento de tiempo. Por tanto, es un método adecuado para transitorios de larga duración en comparación con los tiempos de transito de onda.

En el método de diferencias finitas implícitas, el caudal y la altura desconocidos en un punto de la cuadrícula al final del paso de tiempo ($t + \Delta t$) se expresan en términos de los valores desconocidos de estas variables en las secciones vecinas. Por lo tanto, las ecuaciones para todo el sistema deben resolverse simultáneamente.

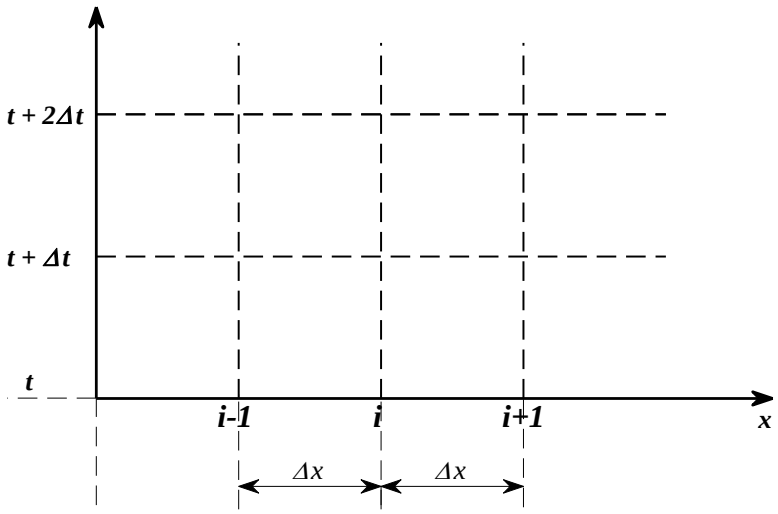


Figura G-9: Malla Espacio Tiempo para el Método de Diferencias Finitas Implícitas

De acuerdo a la Figura G-9, las variables de estado en el tiempo t han sido calculadas, siendo necesario calcular nuevos valores en el tiempo $t = t + \Delta t$. Para cumplir con este propósito, las derivadas parciales en las ecuaciones (G-1) y (G-2) se aproximan con las siguientes expresiones en diferencias finitas que incluyen los dos coeficientes de ponderación arriba mencionados.

$$\frac{\partial H}{\partial x} = \frac{\left[\theta_2 H_{i+1}^{t+\Delta t} + (1-\theta_2) H_{i+1}^{t}\right] - \left[\theta_2 H_{i}^{t+\Delta t} + (1-\theta_2) H_{i}^{t}\right]}{\Delta x} \quad \text{(G-51)}$$

$$\frac{\partial H}{\partial t} = \frac{\left[\theta_1 H_{i+1}^{t+\Delta t} + (1-\theta_1) H_{i}^{t+\Delta t}\right] - \left[\theta_1 H_{i+1}^{t} + (1-\theta_1) H_{i}^{t}\right]}{\Delta t} \quad \text{(G-52)}$$

$$\frac{\partial Q}{\partial x} = \frac{\left[\theta_2 Q_{i+1}^{t+\Delta t} + (1-\theta_2) Q_{i+1}^{t}\right] - \left[\theta_2 Q_{i}^{t+\Delta t} + (1-\theta_2) Q_{i}^{t}\right]}{\Delta x} \quad \text{(G-53)}$$

$$\frac{\partial H}{\partial t} = \frac{\left[\theta_1 Q_{i+1}^{t+\Delta t} + (1-\theta_1) Q_{i}^{t+\Delta t}\right] - \left[\theta_1 Q_{i+1}^{t} + (1-\theta_1) Q_{i}^{t}\right]}{\Delta t} \quad \text{(G-54)}$$

Además, el término $fQ|Q|/(2DA^2)$ se aproxima como:

$$\frac{fQ|Q|}{2DA^2} = \frac{f}{8DA^2}\left((1-\varepsilon)\left(Q_i^t + Q_{i+1}^t\right) + \varepsilon\left(Q_i^{t+\Delta t} + Q_{i+1}^{t+\Delta t}\right)\right)\left|Q_i^t + Q_{i+1}^t\right|$$

siendo ε la constante para la linealización del termino de fricción introducida por Karney and McInnis (1990), ya mencionada anteriormente.

Después de las sustituciones, las ecuaciones (G-1) y (G-2) se transforman en las dos ecuaciones siguientes:

$$d_1\, Q_{i+1}^{t+\Delta t} + d_2\, Q_i^{t+\Delta t} - d_3\, H_i^{t+\Delta t} + d_3\, H_{i+1}^{t+\Delta t} + d_4 = 0 \quad \text{(G-55)}$$

$$-c_1\, Q_{i+1}^{t\Delta t} + c_1\, Q_i^{t+\Delta t} - c_2\, H_i^{t+\Delta t} + c_3\, H_{i+1}^{t+\Delta t} + c_4 = 0 \quad \text{(G-56)}$$

Donde los coeficientes vienen dados por las siguientes expresiones:

$$d_1 = 2\,(1-\theta_1) - \frac{\theta_2 \Delta t\left(Q_i^t + Q_{i+1}^t\right)}{A\,\Delta x} + f_{av}\Delta t\,\varepsilon\,\frac{\left|Q_i^t + Q_{i+1}^t\right|}{4\,D\,A} + \frac{k}{2}\left(1 - sign(Q)\,\frac{a\,\Delta t}{\Delta x}\right) \quad \text{(G-57)}$$

$$d_2 = 2\,\theta_1 + \frac{\theta_2 \Delta t\left(Q_i^t + Q_{i+1}^t\right)}{A\,\Delta x} + f_{av}\Delta t\,\varepsilon\,\frac{\left|Q_i^t + Q_{i+1}^t\right|}{4\,D\,A} + \frac{k}{2}\left(1 - sign(Q)\,\frac{a\,\Delta t}{\Delta x}\right) \quad \text{(G-58)}$$

$$d_3 = \frac{2\,\theta_2\, g\, A\,\Delta t}{\Delta x} \quad \text{(G-59)}$$

$$
\begin{aligned}
d_4 = & \frac{2\,(1-\theta_2) g\,A\,\Delta t\left(H_{i+1}^t - H_i^t\right)}{\Delta x} - 2\left(\theta_1 Q_{i+1}^t + (1-\theta_1) Q_i^t\right) \\
& + \frac{\Delta t}{\Delta x}\left(Q_i^t + Q_{i+1}^t\right)\left[(1-\theta_2) Q_{i+1}^t - (1-\theta_2) Q_i^t\right] \\
& + f_{av}\Delta t\,(1-\varepsilon)\frac{\left(Q_i^t + Q_{i+1}^t\right)\left|Q_i^t + Q_{i+1}^t\right|}{4\,D\,A} \\
& + \frac{k}{2}\left[sign(Q)\left(\left|Q_{i+1}^t\right| - \left|Q_i^t\right|\right) - \left(Q_{i+1}^t + Q_i^t\right)\right]
\end{aligned}
\tag{G-60}
$$

$$
c_1 = \frac{a^2\,\theta_2}{\Delta x} \tag{G-61}
$$

$$
c_2 = \frac{(1-\theta_1) g\,A}{\Delta t} - \frac{g\theta_2\left(Q_i^t + Q_{i+1}^t\right)}{2\,\Delta x} \tag{G-62}
$$

$$
c_3 = \frac{\theta_1 g\,A}{\Delta t} + \frac{g\theta_2\left(Q_i^t + Q_{i+1}^t\right)}{2\,\Delta x} \tag{G-63}
$$

$$
\begin{aligned}
c_4 = & -\frac{g\,A}{\Delta t}\left(\theta_1 H_{i+1}^t + (1-\theta_1) H_i^t\right) \\
& + \frac{g}{2\,\Delta x}\left(Q_i^t + Q_{i+1}^t\right)\left[(1-\theta_2) H_{i+1}^t - (1-\theta_2) H_i^t\right] \\
& + \frac{a^2}{\Delta x}\left[(1-\theta_2) Q_{i+1}^t - (1-\theta_2) Q_i^t\right]
\end{aligned}
\tag{G-64}
$$

Donde θ_1 y θ_2 son los coeficientes de ponderación, ε es la constante de linealización de la fricción y f_{av} corresponde al coeficiente de fricción medio $\left(f_i^t + f_{i+1}^t\right)/2$

En resultado, se tiene un par de ecuaciones como la (G-55) y la (G-56) por cada espacio intermodal, así que si n es el número de nodos, el número de espacios entre nodos es $n-1$ y el número de incógnitas es $2(n-1)$. El número de incógnitas en las tubería es dos por el número de nodos o secciones, $n-1$. De manera que si se supone conocida la altura en el primer y último nodo es posible calcular todas las variables. Específicamente, es posible calcular los flujos de entrada y salida de la tubería , así como el estado de todos los nodos interiores.

REFERENCIAS

Åkesson, J., Gäfvert, M., & Tummescheit, H. (2009). Jmodelica—an open source platform for optimization of modelica models. Proceedings of MATHMOD.

Alexiou, A., & Tsalavoutas, A. (2011). Introduction to Gas Turbine Modelling With PROOSIS.

Andersson, M. (1990). Omola : An Object-Oriented Language for Model Representation.

Applied Flow Technology. (2017). AFT Impulse 6.0 Waterhammer user guide (http://www).

Aspen Technology. (2001). Aspen Plus 11.1 Users Guide. Aspen Technology. Cambridge, USA.

Astrom, K. J., Elmqvist, H., & Mattsson, S. E. (1998). Evolution of Continuous-Time Modeling and Simulation. ESM, 9–18.

Baharev, A., Neumaier, A., & Schichl, H. (2017). Failure Modes of Tearing and a Novel Robust Approach. 353–362. https://doi.org/10.3384/ecp17132353

Billups, S. C. (2002). A homotopy-based algorithm for mixed complementarity problems. SIAM Journal on Optimization, 12(3), 583–605.

Bombardieri, C., Traudt, T., & Manfletti, C. (2019). Experimental study of water hammer pressure surge. Progress in Propulsion Physics, 11, 555–570.

Broyden, C. G. (1965). A Class of Methods for Solving Nonlinear Simultaneous Equations. Mathematics of Computation, 19(92), 577. https://doi.org/10.2307/2003941

Carpanzano, E. (2000). Order reduction of general nonlinear DAE systems by automatic tearing. Mathematical and Computer Modelling of Dynamical Systems, 6(2), 145–168.

Casella, F. (2015). Simulation of large-scale models in modelica: State of the art and future perspectives. Proceedings of the 11th International Modelica Conference, Versailles, France, September 21-23, 2015, 118, 459–468.

Casella, F., Sielemann, M., Savoldelli, L., & others. (2011). Steady-state initialization of object-oriented thermo-fluid models by homotopy methods. Proceedings of the 8th International Modelica Conference, 86–96.

Cellier, F. E., Clauß, C., & Urqu\'\ia, A. (2007). Electronic circuit modeling and simulation in Modelica. Proceedings of the Sixth Eurosim Congress on Modelling and Simulation, 1–10.

Cellier, F. E., & Kofman, E. (2006). Continuous system simulation. Springer Science+Business Media.

http://www.loc.gov/catdir/enhancements/fy0663/2005936516-d.html

Cellier, F. E., & Nebot, À. (2005). The Modelica Bond Graph Library. Paper Presented at the 4th International Modelica Conference.

Chaudhry, M. H. (2014). Applied hydraulic transients (Third). Springer.

Chen, C., & Mangasarian, O. L. (1996). A class of smoothing functions for nonlinear and mixed complementarity problems. Computational Optimization and Applications, 5(2), 97–138.

Clauß, C., Leitner, T., Schneider, A., & Schwarz, P. (2000). Modelling of electronic circuits with Modelica. Proc. Modelica Workshop, Lund, Sweden, 3–11.

Coïc, C., Hübel, M., & Thorade, M. (2020). Enhanced Steady-State in Modelon Jet Propulsion Library, an Enabler for Industrial Design Workflows. American Modelica Conference; Boulder; Colorado.

Duff, I. S. (1981). On Algorithms for Obtaining a Maximum Transversal. ACM Transactions on Mathematical Software, 7(3), 315–330. https://doi.org/10.1145/355958.355963

El-Hefni, B., Bouskela, D., & Lebreton, G. (2011). Dynamic Modelling of a Combined Cycle Power Plant with ThermoSysPro. Proceedings of the 9th Modelica Conference, 365–375.

Elmqvist, H. (1978). A Structured Model Language for Large Continuous Systems.

Elmqvist, H. (2014). Modelica Evolution – From My Perspective Dynamic Modeling Language. Presented at the the 10th International Modelica Conference, March 10-12, 2014, Lund, Sweden, 17–26. https://doi.org/10.3384/ECP1409617

Elmqvist, H., & Otter, M. (1994). Methods for tearing systems of equations in object-oriented modeling. Proceedings of the European Simulation Multiconference, 94(Mah), 1–3. http://people.inf.ethz.ch/~fcellier/Lect/MMPS/Refs/tearing.pdf

Empresarios Agrupados International. (2015). EcosimPro User Manual. https://www.ecosimpro.com

Enhanced Turbojet Multi-point - Help Center. (n.d.). Retrieved February 20, 2024, from https://help.modelon.com/latest/tutorials/enhanced_turbojet_multipoint/

Figueiredo, J. R., Santos, R. G., Favaro, C., Silva, A. F. S., & Sbravati, A. (2002). Substitution-Newton-Raphson method applied to the modeling of a vapour compression refrigeration system using different representations of the thermodynamic properties of R-134a. Journal of the Brazilian Society of Mechanical Sciences, 24, 158–168.

Fischer, A. (1992). A special Newton-type optimization method. Optimization, 24(3–4),

269–284.

Fritzson, P. (2014). Principles of object-oriented modeling and simulation with Modelica 3.3: a cyber-physical approach. John Wiley \& Sons.

Fritzson, P., Aronsson, P., Lundvall, H., Nyström, K., Pop, A., Saldamli, L., & Broman, D. (2005). The OpenModelica modeling, simulation, and development environment. 46th Conference on Simulation and Modelling of the Scandinavian Simulation Society (SIMS2005), Trondheim, Norway, October 13-14, 2005.

Fritzson, P., & Engelson, V. (1998). Modelica—A unified object-oriented language for system modeling and simulation. European Conference on Object-Oriented Programming, 67–90.

García-Alvárez, A. (2017). WATERHAMMER_MOC: Una librería en Modelica para el modelado del análisis de transitorios en redes de tuberías mediante el método de las características. Máster Universitario en Ingeniería de Sistemas y de Control. Universidad Nacional de Educación a Distancia.

Ghidaoui, M. S., Zhao, M., McInnis, D. A., & Axworthy, D. H. (2005). A review of water hammer theory and practice. Applied Mechanics Reviews, 58(1/6), 49.

Grace, A. C. W. (1991). SIMULAB, an integrated environment for simulation and control. Proc. 1991 American Control Conference, 1015–1020.

Grossmann, I. E., & Turkay, M. (1996). Solution of algebraic systems of disjunctive equations. Computers \& Chemical Engineering, 20.

Hinna, J. T. (2021). Modelica model of transient pipe flow in hydraulic laboratory systems using the method of characteristics. NTNU.

HYSYS, A. (n.d.). V7. 0, 2009, Aspen HYSYS Simulation Basis; Aspen HYSYS User's Guide. AspenONEV7. 0 Documentation, AspenTechnology, Inc., Burlington, Massachusetts.

Jones, S. M. (2007). An introduction to thermodynamic performance analysis of aircraft gas turbine engine cycles using the numerical propulsion system simulation code. March. https://doi.org/NASA TM 2007-214690

Kari, O. (2022). Improving Tearing in a Modelica Compiler (Master´s Thesis). Lund University.

Karney, Bryan W., & McInnis, D. (1990). Transient Analysis of Water Distribution Systems. Journal - American Water Works Association, 82(7), 62–70. https://doi.org/10.1002/j.1551-8833.1990.tb06992.x

Karney, Bryan W, & Ghidaoui, M. S. (1997). Flexible discretization algorithm for fixed-grid MOC in pipelines. Journal of Hydraulic Engineering, 123(11), 1004–1011.

Karney, Bryan William. (1984). Analysis of fluid transients in large distribution networks. University of British Columbia.

Kielkowski, R. M. (1993). Inside SPICE: Overcoming the obstacles of circuit simulation. McGraw-Hill, Inc.

Koppel, C., Moral, J., De Rosa, M., Pérez, R., & Omaly, P. (2011). Satellite propulsion modeling with Ecosimpro: Comparison between simulation and ground tests. Progress in Propulsion Physics, 2, 743–764.

Kurzke, J. (1995). Advanced user-friendly gas turbine performance calculations on a personal computer. ASME 1995 International Gas Turbine and Aeroengine Congress and Exposition, V005T16A003-V005T16A003.

Landshoff, R. (1955). A numerical method for treating fluid flow in the presence of shocks.

Lema, M. (2013). Multyphase fluid hammer: modeling, experiments and simulations.

Li, D., & Fukushima, M. (2000). Smoothing Newton and quasi-Newton methods for mixed complementarity problems. Computational Optimization and Applications, 17, 203–230.

Li, M., Che, H., & others. (2012). A smoothing inexact Newton method for generalized nonlinear complementarity problem. Mathematical Problems in Engineering, 2012.

Li, X., Demmel, J., & Gilbert, J. (2011). iL. Grigori, M. Shao, and I. Yamazaki,“SuperLU Users’ Guide,” Lawrence Berkeley National Laboratory.

Lindblom, A. (2015). Modelling of Dynamic Pressure Conditions in District Heating Systems [Chalmers University of Technology]. https://hdl.handle.net/20.500.12380/225153

LLC., G. E. E. S. (n.d.). GateCycle Version 5.40. Manual.

Maciel, R. S., Padilha-Feltrin, A., & Righeto, E. (2006). Substitution-Newton-Raphson Method for the Solution of Electric Network Equations. 2006 IEEE/PES Transmission & Distribution Conference and Exposition: Latin America, 1–6. https://doi.org/10.1109/TDCLA.2006.311391

Majetta, K., Böhme, S., Clauß, C., & Schneider, P. (2011). Msl electrical spice3-status and further development. 8th International Modelica Conference, Dresden, Germany.

Mao, G., & Petzold, L. R. (2002). Efficient integration over discontinuities for differential-algebraic systems. Computers \& Mathematics with Applications, 43(1–2), 65–79.

Mattsson, S. E., & Elmqvist, H. (1997). Modelica-An international effort to design the next generation modeling language. IFAC Proceedings Volumes, 30(4), 151–155.

Mattsson, S. E., & Söderlind, G. (1993). Index reduction in differential-algebraic equations using dummy derivatives. SIAM Journal on Scientific Computing, 14(3), 677–692.

Mentor Graphics Corporation. (2016). FloMASTER (Flowmaster) V8.

Merilo, M. (1992). Water Hammer Prevention, Mitigation, and Accommodation. Volume 4: Review of Analytic Models and Computer Codes. Part 1: Sample Problems and Comparisons. Epri Np-6766.

Mitchell, E. E., & Gauthier, J. S. (1976). Advanced continuous simulation language (ACSL). Simulation, 26(3), 72–78.

Modelica Association. (2023). Modelica Specification, Version 3.6. http://www.modelica.org

Moré, J. J., Garbow, B. S., & Hillstrom, K. E. (1980). User guide for MINPACK-1.

Morton, W. (2003). Equation-oriented simulation and optimization. Proceedings Indian National Science Academy Part A, 69(3/4), 317–358.

Nagel, L. W., & Pederson, D. O. (1973). SPICE (Simulation Program with Integrated Circuit Emphasis), Memorandum ERL-M382, University of California. Electronics Research Laboratory.

Nikoukhah, R. (2007). Hybrid Dynamics in Modelica: Should all Events be Considered Synchronous. EOOLT, 37–48.

Nikoukhah, R., & Furic, S. (2008). Synchronous and asynchronous events in Modelica: proposal for an im proved hybrid model.

Oscar Kari. (2022). Improving Tearing in a Modelica Compiler. Lund University.

Otter, M., & Cellier, F. (2019). Software for Modeling and Simulating Control Systems (pp. 419–432). https://doi.org/10.1201/9781315214030-22

Pantelides, C. C. (1988). The consistent initialization of differential-algebraic systems. SIAM Journal on Scientific and Statistical Computing, 9(2), 213–231.

Reissig, G., Martinson, W., & Barton, P. (2000). Differential-Algebraic Equations of Index 1 May Have an Arbitrarily High Structural Index. SIAM Journal on Scientific Computing, 21, 1987–1990. https://doi.org/10.1137/S1064827599353853

Ren, X., Li, P., & Chen, H. (2020). Numerical simulation of water hammer in liquid methane feedline based on two compressible fluid models. Cryogenics, 109, 103122.

Rico-Ramirez, V. (1998). Representation, analysis and solution of conditional models in an equation-based environment. Carnegie Mellon University.

Schutte, J. S. (2009). Simultaneous multi-design point approach to gas turbine on-design cycle analysis for aircraft engines. Georgia Institute of Technology.

Sieleman, M. (2015). Steady state: The next big thing? - Modelon. https://www.modelon.com/steady-state-the-next-big-thing/

Sielemann, M., Co\"\ic, C., Hübel, M., Zhao, X., & Kyprianidis, K. (2020). Introduction to multi-point design strategies for aero engines. Turbo Expo: Power for Land, Sea, and Air, 84157, V006T08A005.

Strauss (ed.), J. C. (1967). The SCi Continuous System Simulation Language (CSSL). Simulation, 9(6), 281–303. https://doi.org/10.1177/003754976700900601

Sunrise. (2016). PIPENET VISION 1.8.0 Transient Module User and Reference Manual Software (http://www).

Tarjan, R. (1972). Depth-First Search and Linear Graph Algorithms. SIAM Journal on Computing, 1(2), 146–160. https://doi.org/10.1137/0201010

Täuber, P., Ochel, L., & Bachmann, B. (2016). Dynamic tearing: efficiency enhancement of tearing methods by consideration of solvability criteria during runtime. Proceedings of the 7th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, 61–68.

Täuber, P., Ochel, L., Braun, W., & Bachmann, B. (2014). Practical realization and adaptation of Cellier's tearing method. Proceedings of the 6th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools, 11–19.

Thermoflow Inc. (2015). ThermoFlex - Fully-flexible design and simulation of combined cycles, cogeneration systems, and other thermal power systems. Sudbury, MA, USA. http://www.thermoflow.com

ThermoSysPro Repository. (n.d.). Retrieved September 29, 2023, from https://github.com/ThermoSysPro/ThermoSysPr

Turpin, J. B. (2004). Variable step integration coupled with the method of characteristics solution for water-hammer analysis, a case study. Proceedings of the 52th Jannaf Propulsion Meeting.

Twyman, J. A. (2004). Decoupled Hybrid methods for unsteady flow analysis in pipe networks. La Cáfila.

Urquia, A., Martin, C., & Dormido, S. (2005). Design of SPICE Lib: A Modelica library for modeling and analysis of electric circuits. Mathematical and Computer Modelling of Dynamical Systems, 11(1), 43–60. https://doi.org/10.1080/13873950500052488

Visser, W. P. J., & Broomhead, M. J. (2000). GSP, a generic object-oriented gas turbine simulation environment. ASME Turbo Expo 2000: Power for Land, Sea, and Air, V001T01A002-V001T01A002.

Vítkovský, J. P., Bergant, A., Simpson, A. R., & Lambert, M. F. (2006). Systematic evaluation of one-dimensional unsteady friction models in simple pipelines. Journal of Hydraulic Engineering, 132(7), 696–708.

Vytvytskyi, L., & Lie, B. (2019). OpenHPL for Modelling the Trollheim Hydropower Plant. Energies, 12(12), 2303.

Wegstein, J. H. (1958). Accelerating convergence of iterative processes. Communications of the ACM, 1(6), 9–13.

Wylie, E. B., Streeter, V. L., & Suo, L. (1993). Fluid transients in systems (Vol. 1). Prentice Hall Englewood Cliffs, NJ.

Yong, L. (2010). Nonlinear complementarity problem and solution methods. Artificial Intelligence and Computational Intelligence: International Conference, AICI 2010, Sanya, China, October 23-24, 2010, Proceedings, Part I 2, 461–469.

Zaher, J. J. (1995). Conditional Modeling (Issue November). PhD thesis, Department of Chemical Engineering, Carnegie Mellon University.