Mecánica Computacional Vol. XXIV
A. Larreteguy (Editor)
Buenos Aires, Argentina, Noviembre 2005

UNA FORMULACIÓN MATEMÁTICA ORIENTADA A OBJETOS PARA SIMULACIÓN CONTINUA

Gustavo A. Boroni[1], **Alejandro Clausse**[1,2]

[1]CNEA y Universidad Nacional del Centro,7000 Tandil, Argentina
email: gboroni@exa.unicen.edu.ar

[2]También CONICET
email: clausse@exa.unicen.edu.ar

Palabras Claves: Simulación continua, DAE, Orientación a objetos, Modelado.

Resumen: *En este trabajo se presenta un estudio de los problemas que surgen en la implementación de modelos de simulación continua usando programación orientada a objetos (O-O). El enfoque se dirige a encontrar un diseño O-O que resuelva problemas de valor inicial de simulación continua mediante DAE de manera generalizada. Para ello, se ha encontrado que es importante realizar un planteo matemático que sea compatible con la programación O-O, lo cual lleva naturalmente a esquemas numéricos implícitos. Dentro de este tipo de esquemas se ha propuesto una extensión del método BDF, que demostró tener una alta compatibilidad con los requerimientos de flexibilidad y modificabilidad. Como resultado final se realizaron una serie de experimentos donde se compararon aspectos numéricos como errores y velocidades de cálculo, y distintos factores de calidad de software tales como performance, modularidad, y extensibilidad, haciendo hincapié en las ventajas y desventajas del uso de distintos esquemas de cálculo.*

1 INTRODUCCIÓN

La programación orientada a objetos (O-O) y la utilización de frameworks han demostrado ser un medio eficiente para la construcción y mantenimiento de sistemas de software. La aplicación de esta metodología se ha extendido a muchas ramas de la computación aplicada, como bases de datos, agentes, sistemas expertos, computación gráfica, soportes de comunicación, entre otras. Esencialmente la metodología O-O consiste en un replanteo de la modularización de los programas [4]. Todo programa de computación tiene dos elementos fundamentales: datos y funciones. Los datos son los portadores de la información digital por medio de una codificación, mientras que las funciones (que en realidad también deben ser representadas mediante una codificación) son los procedimientos con los cuales se modifican los datos. En la metodología de programación procedural los programas se dividen en módulos definidos por las funciones (subrutinas, procedimientos, funciones, etc.). Esta es una manera natural de organización de un ingeniero industrial, que divide un proyecto en tareas, y los productos van pasando de una a otra tarea en forma secuencial. Esta manera de estructurar los programas, que es muy eficiente desde el punto de vista del proceso de cálculo, da lugar a enormes dificultades cuando se debe modificar o extender el programa. La raíz de este problema radica en que los datos son compartidos por todos los módulos funcionales, por lo que una modificación en un módulo implica revisar toda la estructura para asegurar compatibilidad y consistencia.

La metodología O-O cambia radicalmente el principio de estructuración de un programa: los elementos fundamentales son los datos. Las funciones (llamadas métodos) se definen e implementan de acuerdo a la modularización establecida por los datos. De esta manera, cada módulo del programa (llamado objeto) es responsable de sus propios datos, los cuales sólo pueden ser modificados por los métodos propios, y sólo pueden "observados" por los otros objetos a través de protocolos que resguardan el encapsulamiento.

Además de la técnica de encapsulamiento, la metodología O-O introduce otros conceptos de modularidad que ayudan al programador a estructurar los sistemas asegurando la extensibilidad y modificabilidad futura. Entre otros se mencionan la abstracción, herencia y el polimorfismo.

Para poder programar siguiendo la metodología O-O se desarrollaron lenguajes de alto nivel, como C++, Delphi y Java. Sin embargo, es importante notar que el uso de un lenguaje O-O es condición necesaria, pero no suficiente para asegurar las ventajas de la metodología O-O (*i.e.* modificabilidad y extensibilidad). En el marco de esta metodología cobra un rol crucial el diseño de la estructura básica del programa (llamada *framework*), a tal punto que hay un área de ingeniería de software dedicada específicamente al desarrollo de arquitecturas standards clásicas recomendadas para problemas típicos (llamados *patterns*).

En el área de computación científica, si bien ha habido avances importantes en el desarrollo de software O-O, la utilización de frameworks de diseño no ha alcanzado todavía el mismo grado que en otras áreas. Las mayoría de las herramientas de software actualmente disponibles para cálculo científico sigue todavía estructurándose en base a diseños modularizados en base a las funciones, resultando muy trabajoso (y a veces imposible)

mantener y extender los sistemas implementados, incluso para cambios menores. La razón principal de este hecho es que el software se implementa en programación procedural (*i.e.* FORTRAN) o se usan lenguajes O-O pero con diseños esencialmente procedurales. Una sintomatología típica de este tipo de inconvenientes es que a menudo se suele encarar el cálculo de problemas numéricos de gran escala empleando varias herramientas diferentes para generar resultados parciales, que luego se procesan con otras aplicaciones para obtener por fin un resultado final.

Existen algunos antecedentes de esfuerzos interesantes de implementaciones de métodos numéricos complejos en el paradigma O-O [2-7-8-9]. Todos ellos concluyen en que el papel que juega el planteo matemático de cada problema es crucial para lograr buenos diseños de software.

En este trabajo se presenta un estudio de los problemas que se presentan en la utilización del paradigma de O-O en el modelado y simulación continua mediante sistemas de ecuaciones diferenciales y algebraicas (DAE). El objetivo del análisis es encontrar un diseño O-O que resuelva problemas DAE de valor inicial de manera generalizada, con el propósito de facilitar la tarea del modelista que a menudo implica un proceso de sofisticación progresiva de los modelos, con la modificación, extensión y eliminación de ecuaciones y variables al sistema. Se mostrará que en este tipo de problemas —que involucran un fuerte peso de la funcionalidad respecto de la estructura de datos— es importante partir de planteos matemáticos compatibles con la metodología O-O. En el caso de simulación continua mediante DAE, esto lleva naturalmente a esquemas numéricos implícitos. Dentro de este tipo de esquemas se propone en este trabajo una extensión del método BDF (Backward Differentiation Formulas), que demostró tener una alta compatibilidad con los requerimientos de flexibilidad y modificabilidad.

Finalmente, se presenta una implementación de las ideas desarrolladas en una arquitectura O-O destinada a facilitar el diseño e implementación de problemas de simulación continua, estudiándose diferentes factores de calidad de software tales como performance, modularidad, y extensibilidad.

2 SIMULACIÓN CONTINUA

Se llama simulación continua a las representaciones numéricas de realidades cambiantes en el tiempo en base a procesos sincronizados; es decir que el tiempo es una única variable global que sirve de referencia para todas las otras variables. Este concepto, que puede parecer obvio para quienes están acostumbrados a las metodologías de simulación numérica en base a ecuaciones diferenciales, no es la única posibilidad para simular sistemas dinámicos. Existe también el concepto de tiempo local, usado en la llamada simulación discreta, mediante el cual cada hilo del proceso general tiene asociado una línea de tiempo diferente, las cuales deben ser sincronizadas de alguna manera.

Los sistemas físicos que tienen variables dependientes del tiempo en general son representados mediante simulación continua por medio de ecuaciones diferenciales ordinarias. Los problemas que involucran el espacio como variable independiente —además del tiempo— son más complicados y se representan con ecuaciones a derivadas parciales. El

objeto del presente trabajo son la primera clase de problemas —*i.e.* una sola variable independiente—, aunque algunas de las conclusiones a las que se llegará pueden servir de base para el tratamiento de ecuaciones.

Considérese por ejemplo el caso simple de dos tanques con dos bombas que trasfieren agua de un tanque al otro como muestra la Figura 1.

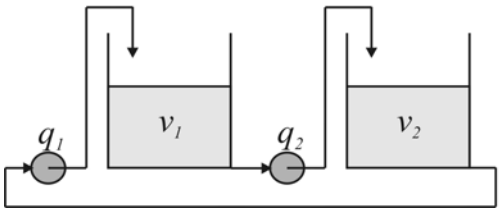

Figura 1: Tanques de regulación por bombas.

La dinámica de este sistema puede describirse en una primera aproximación con dos ecuaciones diferenciales para los volúmenes de agua y dos ecuaciones algebraicas para los caudales:

$$\begin{aligned} \dot{v}_1 &= q_1 - q_2 \\ \dot{v}_2 &= q_2 - q_1 \\ q_1 &= p_1\left(v_{ref} - v_1\right) \\ q_2 &= p_2\left(v_{ref} - v_2\right) \end{aligned} \tag{1}$$

Donde se ha adoptado un control proporcional de los volúmenes, buscando mantener un volumen de referencia fijo v_{ref}.

El procedimiento usual para resolver numéricamente un problema como este es utilizar un esquema de integración clásico (*i.e.* Runge-Kutta) para lo cual se lleva el sistema a la forma:

$$\dot{\vec{x}} = \vec{f}(\vec{x}) \tag{2}$$

Donde $\vec{x}$ es el vector de estado. En este caso:

$$\begin{aligned} \vec{x} &= (v_1, v_2) \\ f_1 &= (p_1 - p_2)v_{ref} - p_1 v_1 + p_2 v_2 \\ f_2 &= (p_2 - p_1)v_{ref} + p_1 v_1 - p_2 v_2 \end{aligned} \tag{3}$$

Generalmente, en un proceso de modelado ordenado, las ecuaciones se van complicando poco a poco a medida que se van agregando términos y variables para representar más detalles de la realidad. La tarea del modelista es ir experimentando con las progresivas sofisticaciones siguiendo una sucesión de hipótesis y comparaciones con la realidad. Las siguientes son ejemplos de las modificaciones típicas que se encuentran durante el desarrollo de modelos, complicando la tarea del modelista.

1) Cambiar una constante por una función de las variables de estado.

Durante el desarrollo de modelos dinámicos diferenciales es usual extender un modelo introduciendo una dependencia de las variables de estado en algunos parámetros de control. Este puede traer complicaciones de programación cuando la estructura de datos es muy rígida.

Por ejemplo, en el caso base se puede extender el modelo imponiendo una dependencia de los parámetros p_i con los volúmenes (*i.e.* introduciendo un control no lineal). Si se busca generalidad en el cambio, en lugar de cambiar la ecuación del caudal se debería cambiar la declaración de p_i de constante a variable auxiliar, y agregar en otra línea la dependencia con el volumen. A su vez, dependiendo de la manera en que cada tipo de dato (constante, variable auxiliar, variable de estado) es comunicado, debe asegurarse consistencia en la transferencia de datos entre módulos durante el cálculo.

2) Cambiar una constante por una variable de estado.

Esta modificación es parecida a la anterior, pero aquí se agrega una ecuación diferencial para dar cuenta de la variación del parámetro en cuestión. Los inconvenientes son análogos a la modificación 1.

3) Reemplazar una ecuación diferencial por una ecuación algebraica.

Este cambio es una transformación bastante común cuando se trabaja con ecuaciones diferenciales de balances del tipo:

$$\dot{E} = \phi_{in} - \phi_{out} \tag{4}$$

Donde las tasas de crecimiento y decrecimiento de la variable E, ϕ_{out} y ϕ_{in}, son funciones de las variables de estado del sistema.

Si la constante de tiempo de esta ecuación es mucho menor que las otras ecuaciones del sistema, se puede plantear una aproximación cuasiestática anulando la derivada, es decir:

$$\phi_{in} = \phi_{out} \tag{5}$$

La cual es una ecuación algebraica. Modificaciones como esta provocan cambios del orden del sistema de ecuaciones, lo cual puede afectar entre otras cosas el control del paso de tiempo. Sin embargo, el inconveniente principal no es ese, sino el hecho que obligan a reescribir completamente el sistema de ecuaciones para volver a llevarlo a la forma requerida

por los esquemas explícitos, esto es $\dot{\vec{x}} = \vec{f}(\vec{x})$. En efecto, si se elimina la ecuación (4), la variable E no será más variable de estado, aunque seguirá siendo requerida para calcular $\vec{f}(\vec{x})$ por lo que deberá calcularse a partir de las ecuaciones algebraicas auxiliares —incluyendo la nueva—. Este incordio implica la reprogramación del modelo, con su carga adicional de errores promedio de implementación.

4) Reemplazar una ecuación algebraica por una ecuación diferencial.

Este cambio es la transformación inversa a la anterior. También aquí se pueden generar inconvenientes en el control del paso de tiempo y en la reimplementación del modelo.

5) Agregar un término con una derivada temporal.

Este cambio típicamente aparece cuando se agrega un término de control derivativo. Por ejemplo en el ejemplo de los tanques:

$$\begin{aligned} q_1 &= p_1\left(v_{ref} - v_1\right) - d_1\dot{v}_1 \\ q_2 &= p_2\left(v_{ref} - v_2\right) - d_2\dot{v}_2 \end{aligned} \tag{6}$$

Este tipo de modificación es complicada porque requiere reescribir nuevamente el sistema $\dot{\vec{x}} = \vec{f}(\vec{x})$, lo cual implica como en el caso 3 la reprogramación del modelo.

6) Generar un sistema de N instancias.

Este es un cambio típico en el modelado de procesos con gran número de componentes similares. Ejemplos típicos son los procesos de destilerías, las tuberías, los modelos de poblaciones. De hecho la discretización de ecuaciones a derivadas parciales lleva generalmente a sistemas de muchas ecuaciones ordinarias similares acopladas (celdas). En el ejemplo de los tanques la extensión a N instancias es directa, quedando:

$$\begin{aligned} &\dot{v}_1 = q_1 - q_2 && q_1 = \rho_1\left(v_{ref} - v_1\right) \\ &\ldots && \ldots \\ &\dot{v}_i = q_i - q_{i+1} && q_i = \rho_i\left(v_{ref} - v_i\right) \\ &\ldots && \ldots \\ &\dot{v}_N = q_N - q_1 && q_N = \rho_N\left(v_{ref} - v_N\right) \end{aligned} \tag{7}$$

En este caso, es muy útil para el modelista disponer de la propiedad de herencia, que permite encapsular las partes comunes de los subsistemas, ahorrando trabajo de

programación. Es claro darse cuenta que los inconvenientes que traen asociadas las modificaciones anteriores se multiplican por N con esta extensión.

3 SISTEMAS DE ECUACIONES ALGEBRAICAS Y DIFERENCIALES (DAE)

La resolución numérica de problemas de condiciones iniciales modelados con sistemas DAE es un tema de importancia al cual se dedican esfuerzos considerables desde hace 30 años [1-2]. Muchos problemas de ingeniería y ciencia se modelan en forma natural con sistemas DAE, donde en general las ecuaciones diferenciales representan balances del tipo:

$$\vec{F}(\dot{\vec{x}}, \vec{x}, \vec{y}) = 0 \tag{8}$$

Y las ecuaciones algebraicas representan restricciones o relaciones constitutivas:

$$\vec{G}(\vec{x}, \vec{y}) = 0 \tag{9}$$

Para garantizar que puede resolverse problemas de valores iniciales, los sistemas de ecuaciones (8) y (9) deben asegurar que las derivadas temporales $\dot{\vec{x}}$ tengan solución única para todo conjunto válido de valores de $\vec{x}$ e $\vec{y}$. En este caso, siempre se puede convertir el sistema DAE en un sistema de ecuaciones diferenciales ordinarias (ODE) del tipo:

$$\dot{\vec{x}} = \vec{f}(\vec{x}) \tag{10}$$

El cual se puede resolver con alguno de los integradores numéricos clásicos (Runge-Kutta, multipaso, etc.).

La conversión al sistema de ecuaciones (10) puede ser problemática por varias razones. Por ejemplo, a menudo los sistemas DAE aumentan su rigidez (*stiffness*) al ser reescritos como ODE. De cualquier modo, aunque esto no suceda, nuestro interés en mantener la estructura DAE durante la resolución numérica, surge de la búsqueda de patrones de diseños de software eficientes, especialmente desde el punto de vista del usuario como modelista.

El objeto general de estudio puede plantearse entonces con una función no lineal totalmente implícita de la forma:

$$\vec{F}(\dot{\vec{x}}, \vec{x}) = 0 \tag{11}$$

Las funciones $\vec{F}(\dot{\vec{x}}, \vec{x})$ suelen llamarse funciones residuales. El conjunto de ecuaciones (11) es entonces una combinación de ecuaciones diferenciales y algebraicas. Un problema de condiciones iniciales (IVP) consiste en determinar las funciones temporales $\vec{x}(t)$ que satisfacen un sistema DAE dados los valores $\vec{x}_0$ en $t = 0$.

Para la solución numérica de IVP de DAEs actualmente se dispone de una variedad de métodos numéricos que pueden clasificarse en tres categorías: métodos de un paso, multipaso y de extrapolación. La eficiencia de cada tipo de aproximación depende del problema particular.

En este trabajo, se presentará un estudio de los aspectos de diseño de software para resolver DAE aplicando la subclase de métodos multipaso BDF (Backward Difference Formulas), desarrollado por Gear (1971) [3], el cual tiene propiedades interesantes desde el punto de vista de la flexibilidad.

En líneas generales, el procedimiento BDF para resolver sistemas DAE se basa en transformar las DAE en un sistema puramente algebraico mediante aproximaciones numéricas. Para ello se define una transformación reemplazando $\vec{x}$ y $\dot{\vec{x}}$ en las funciones residuales por funciones $\vec{X}(\vec{x}_n, \vec{x}_{n-1}, \vec{x}_{n-2}, ...)$ y $D\vec{X}(\vec{x}_n, \vec{x}_{n-1}, \vec{x}_{n-2}, ...)$, que representan estimadores multipaso del vector de estado y su derivada (por convención $\vec{x}_k = \vec{x}(t_k)$).

Con esta operación quedan definidas las funciones residuos estimadas:

$$\vec{G}(\vec{x}_n, \vec{x}_{n-1}, \vec{x}_{n-2}, ...) = 0 \tag{12}$$

Las Ecs. (12) son un sistema de ecuaciones algebraicas cuya incógnita es el vector de estado en el nuevo tiempo $\vec{x}_n$, el cual se calcula en función de la historia pasada.

Dentro de este procedimiento general, existen diferentes alternativas de acuerdo a la especificación de:

1) Las fórmulas $\vec{X}$ y $D\vec{X}$.

2) La estrategia de control del paso de tiempo.

3) El método de resolución de las raíces de las funciones residuales estimadas $\vec{G}(\vec{x}_n, \vec{x}_{n-1}, \vec{x}_{n-2}, ...) = 0$.

4) La estructura de datos y clases para manejar los distintos pasos del algoritmo de resolución.

El planteo BDF extendido resulta un esquema de cálculo ventajoso para el desarrollo de modelos, sobre todo porque resuelve naturalmente los problemas que surgen al introducir las modificaciones típicas del desarrollo de modelos dinámicos. En efecto, planteando el problema matemático desde el punto de vista implícito, la forma funcional básica $\vec{F}(\dot{\vec{x}}, \vec{x}) = 0$ es invariante a cualquiera de los cambios mencionados en la sección anterior. Desde este punto de vista se puede decir que los esquemas implícitos son una “matemática natural” para la programación orientada a objetos.

Los objetos naturales serán las funciones $\vec{F}(\dot{\vec{x}}, \vec{x})$. La implementación de la resolución puede ser encapsulada fácilmente, de manera que el modelista “vea” una “caja integradora”

en la cual introduce, agrega, cambia, elimina, funciones del tipo $\vec{F}(\vec{\dot{x}},\vec{x})$, sin preocuparse por reescribir las ecuaciones, ni por problemas de comunicación de datos.

4 ANÁLISIS Y DISEÑO O-O DE DAE

Las bibliotecas de subrutinas tradicionales disponibles para resolver ecuaciones diferenciales presentan dos debilidades principales desde el punto de vista de la programación: interfaces rígidas y complejidad de los algoritmos exponencialmente creciente con la variedad de los sistemas a resolver. La primera debilidad afecta a los usuarios ya que es relativamente dificultoso elaborar modelos usando estas bibliotecas, ya que el modelista carga con un doble trabajo: representar su modelo matemáticamente y adaptar la representación resultante a la interface requerida. La segunda debilidad afecta a los desarrolladores de bibliotecas ya que las tareas de programación y mantenimiento se multiplican.

Ambos problemas son consecuencia de diseños con abstracciones deficientes. Por un lado, para optimizar la performance numérica, el acceso a las subrutinas suele estar forzado a estructuras de datos rígidas, de manera que cuando alguna de esas estructuras es substituida durante el proceso de modelado, se requiere la implementación de nuevas subrutinas. Otro inconveniente es que las subrutinas suelen incluir variables que describen las estructuras de datos.

Mediante la programación O-O podrían producirse implementaciones mejoradas de métodos numéricos, pero debe tenerse en cuenta que habrá siempre una penalidad en la performance de cálculo. Por ello, como la simulación computacional a menudo requiere códigos complicados y altamente eficientes, la performance de los lenguajes de alto nivel siempre compite con las mejoras logradas.

En esta sección se presenta un estudio de estos tópicos, y los resultados obtenidos durante el diseño e implementación de una herramienta general de resolución numérica de sistemas DAE para simulación continua.

4.1 Análisis

Esencialmente, un sistema DAE es un conjunto de funciones escalares $F=\{F_1(A,B),F_2(A,B),...,F_N(A,B)\}$, donde A es un conjunto de N funciones del parámetro tiempo (real e independiente), y B son las correspondientes derivadas temporales de A. La metodología BDF consiste en transformar el conjunto F en un sistema de N ecuaciones algebraicas $G\{G_1([X]),G_2([X]),...,G_N([X])\}=0$ donde $[X]$ es un arreglo de N elementos que representan el valor calculado del conjunto de funciones de A evaluadas en el tiempo actual. La transformación $F\rightarrow G$ se efectúa asignando a cada elemento de A y B una función estimadora multipaso $f([[X]])$, donde $[[X]]$ denota una lista de arreglos $[X]$ creciente con el tiempo (i.e. $[[X]]$ es una lista donde cada elemento es el valor de las funciones *A* calculada en cada tiempo).

4.2 Diseño

El objetivo de los sistemas a implementar es la resolución numérica de sistemas DAE usando esquemas explícitos e implícitos, buscando mantener la flexibilidad para el desarrollo de modelos de simulación continua. En particular se aplicará el método de Runge-Kutta de cuarto orden, y el método BDF generalizado. A continuación se describen los diseños correspondientes a cada esquema.

Para que las clases y los diagramas resulten más claros se omitirán los siguientes aspectos: constructores, destructores y métodos de consulta de atributos. La sintaxis de declaración de atributos, métodos y pseudocodigos es una sintaxis relajada en C++.

4.2.1 Clases para la resolución de esquemas implícitos

"Sistema DAE"

Esta clase define de manera genérica a los sistemas de ecuaciones DAE del tipo (11). Para representar las variables temporales $\vec{x}$ y $\dot{\vec{x}}$ se utilizan matrices dinámicas (x y $dxdt$), donde las columnas están asociadas a las variables del sistema y las filas representan los valores históricos temporales de las variables (ver Figura 2). Las $\vec{F}$ se declaran sobre el método "Calcular_F()", al cual se le pasa por parámetro las estimaciones de $\vec{x}$ y $\dot{\vec{x}}$, y devuelve como resultado el valor de las $\vec{F}$ evaluadas en dichos puntos. Dado que en la resolución de un problema puede haber varios sistemas de ecuaciones vinculados a través de las variables, se agregó una relación de asociación de $0..*$ entre objetos de la misma clase.

Para completar la funcionalidad de esta clase se definieron además los siguientes métodos:

- "Guess_Inicial()": se utiliza para obtener $\vec{x}_{guess}(t)$ asociado a un tiempo t.
- "Posible_Solucion()": permite guardar un $\vec{x}_{probable}(t)$ para un tiempo t.
- "Actualizar_Histórico()": efectiviza un $\vec{x}_{probable}(t)$ como solución para un tiempo t.
- "Obtener_Histórico()": como su nombre lo indica se utiliza para obtener los valores históricos de las variables.

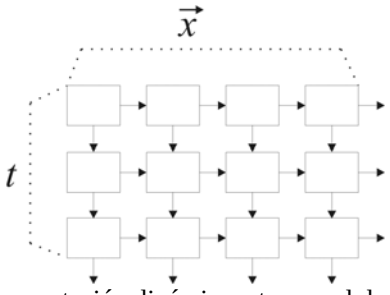

Figura 2: Representación dinámica y temporal de las variables.

"Controlador"

Esta clase representa un controlador maestro que sincroniza y actualiza todo el proceso de cálculo. Las dos variables principales que administra son el tiempo actual t y el paso de

tiempo DT . El método "Avanzar()" se utiliza para calcular los valores de las variables para un tiempo dado por parámetro. Durante la ejecución de este método se actualizarán los valores históricos de los resultados obtenidos para los sistemas de ecuaciones DAE, y se recalculará DT (método "Calcular_Nuevo_DT()") en función de una estimación del error de los resultados (método "Verificar_Paso_Tiempo()").

"Estimador"

Es una clase abstracta que define de manera generalizada el cálculo de valores estimados ($X\_estimado$) de las variables o de los resultados de las ecuaciones. Dichos estimadores podrán ser ingresados por los usuarios (implementación del método "Estimar()"), posibilitando además el uso funciones estimadoras preprogramadas (*i.e.* estimadores semi-implícitos, fórmulas de interpolación multipaso, etc.). Dado que es común que en las funciones estimadoras aparezca la variable paso de tiempo, se agregó DT como propiedad de la clase.

"EstimadorDXDT"

Es subclase de "Estimador" y en la misma se define la función estimadora de la derivada:

$$\dot{\vec{x}}(t) = \sum_{i=0}^{k} \frac{\alpha_i x_{n-i}}{\Delta t} \tag{13}$$

"EstimadorX"

Es subclase de "Estimador" y en la misma se define las funciones estimadoras implícitas, explícitas y semi-implícitas de las variables $\vec{x}$.

"Constructor Sistema Implícito"

Es subclase de "Estimador" y define el método G() para representar las ecuaciones algebraicas asociadas a las funciones $\vec{F}$, utilizando para tal fin las funciones estimadoras de $\vec{x}$ y $\dot{\vec{x}}$.

"Resolvedor de Raíces"

Clase abstracta que define genéricamente los métodos numéricos que resuelven sistemas DAE del tipo (12). Todas las clases que hereden de "Resolvedor de Raíces" deberán implementar el método "Calcular_Raíces()", el cual tiene como parámetros de entrada una función algebraica $G()$ y los valores de variables $\vec{x}_{guess}$, devolviendo como resultado los valores de las raíces $\vec{x}_{raíces}$.

"Resolvedor Sistema Implícito"

Esta clase permite armar la secuencia necesaria para pasar del sistema definido por el usuario a la resolución de un sistema algebraico implícito. Dicho transformación la realiza a

través del método “Calcular()” el cual recibe como parámetro una instancia de la clase “Sistema DAE”.

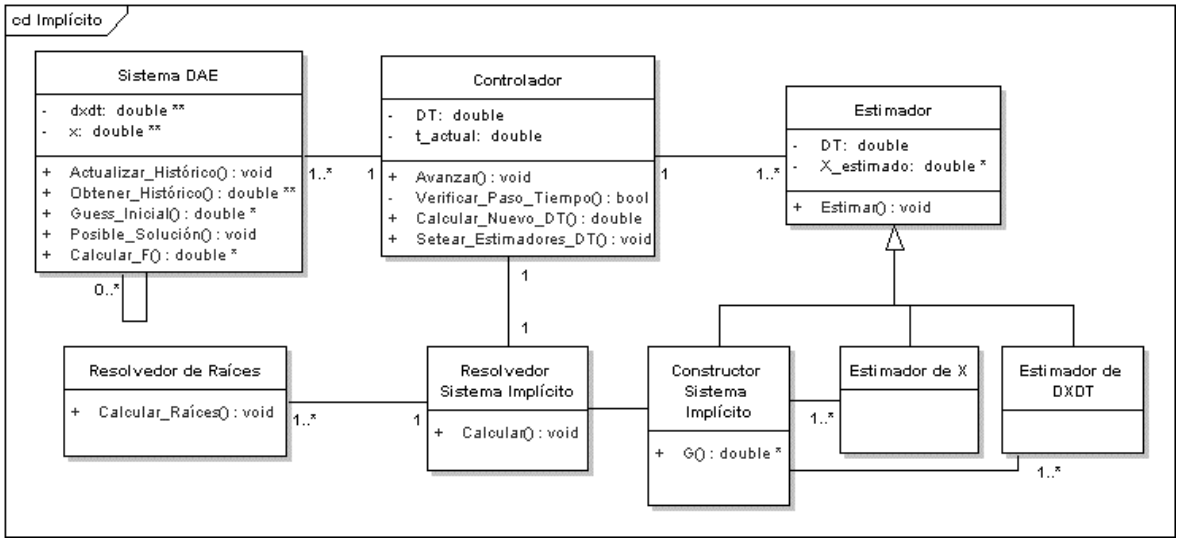

Figura 3: Diagrama de clases para esquemas implícitos.

4.2.2 Clases para la resolución de esquemas explícitos

La mayoría de las clases diseñadas para la resolución de sistemas explícitos son similares a las detalladas para la resolución de sistemas implícitos. La principal diferencia radica en que no se utilizan estimadores para las derivadas. Se mencionarán sólo aquellas clases que poseen diferencias significativas con respecto al diseño de clases anterior.

“Sistema Explícito”

Clase que especifica genéricamente a los sistemas de ecuaciones con derivadas explícitas del tipo (10). Las ecuaciones algebraicas y las funciones de las derivadas se declaran sobre el método “Calcular_DXDT()”, al cual se le pasa por parámetro las estimaciones de $\vec{x}$ y devuelve como resultado el valor de $\dot{\vec{x}}$ evaluadas en dichas estimaciones.

“Constructor Sistema Explícito”

Es subclase de “Estimador” y define el método H() para representar las ecuaciones algebraicas asociadas a las funciones de $\dot{\vec{x}}$.

“Resolvedor Integrador”

Clase abstracta que define genéricamente los métodos numéricos que resuelven sistemas de ecuaciones de derivadas explícitas del tipo (10). Todas las subclases de “Resolvedor Integrador” deberán implementar el método “Calcular_Integración()”, el cual tendrá como

parámetros de entrada una función algebraica H() y los valores de variables $\vec{x}_{guess}$, retornando como resultado los valores de $\vec{x}$.

"Resolvedor Sistema Explícito"

Esta clase permite armar la secuencia necesaria para pasar del sistema definido por el usuario a la resolución de un sistema algebraico explícito. Dicho proceso lo realiza a través del método "Calcular()" el cual recibe como parámetro una instancia de la clase "Sistema Explícito".

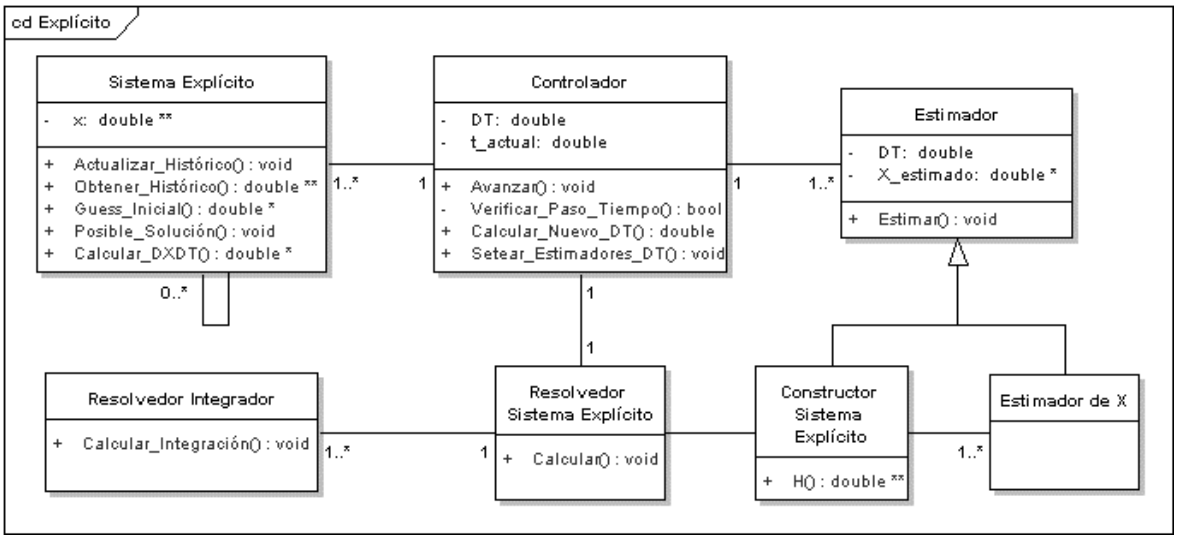

Figura 4: Diagrama de clases para esquemas explícitos.

5 ESTUDIOS NUMÉRICOS

Con el propósito de estudiar la performance del método BDF extendido para resolución de sistemas DAE y de las ventajas comparativas de su implementación en el paradigma O-O, se realizaron una serie de experimentos. Desde el punto de vista numérico, el objetivo fue comparar errores y velocidades de cálculo respecto de métodos explícitos clásicos. Desde el punto de vista de la programación, se estudiaron algunas métricas de extensibilidad y modificabilidad del software resultante con cada esquema de cálculo.

Para el estudio se eligieron tres casos de prueba: un sistema de ecuaciones diferenciales no lineales con solución analítica conocida, un sistema oscilatorio conservativo no lineal, y un sistema de control de fácil extensibilidad y modificabilidad.

Caso 1: Solución analítica conocida

Se consideró primeramente el siguiente sistema de ecuaciones diferenciales ordinarias de segundo grado:

$$\begin{aligned} \dot{x} &= 2y\left(x - y^2\right) - y \\ \dot{y} &= x - y^2 \end{aligned} \tag{14}$$

Con las condiciones iniciales $x(0) = 1,\ y(0) = 0$. La solución exacta es:

$$\begin{aligned} x &= \sin^2(t) + \cos(t) \\ y &= \sin(t) \end{aligned} \tag{15}$$

Lo cual sirve de patrón para comparar la performance numérica de distintos esquemas de resolución. Se implementó el método BDF con un estimador implícito en las variables de estado y un estimador multipaso lineal clásico en las derivadas, es decir:

$$\begin{aligned} \vec{x} &= \vec{x}(t) \\ \dot{\vec{x}} &= \sum_{i=0}^{k} \frac{\alpha_i \vec{x}(t - i\Delta t)}{\Delta t} \end{aligned} \tag{16}$$

Donde k es el orden del estimador multipaso, Δt es el paso de tiempo, y α_i son coeficientes apropiados del método multipaso [5].

Las soluciones obtenidas con el método BDF fueron comparadas con las obtenidas con un esquema clásico de Runge-Kutta de cuatro orden y con el paquete de rutinas FORTRAN LSODE [6].

En la Figura 5 se compara la dependencia del error cuadrático acumulado promediado entre $t = 0$ y $t = 20$:

$$\varepsilon = \frac{1}{t_{\max}} \sum_{t=0}^{t_{\max}} \left| \vec{x}_{real}(t) - \vec{x}_{calc}(t) \right|^2 \tag{17}$$

Con el paso de tiempo, para los distintos esquemas. Se puede observar que a partir del orden $k = 2$, el esquema BDF obtiene menores errores que los dos esquemas patrones.

La performance de cálculo se estudió calculando el cociente entre el tiempo de cálculo y el tiempo real. En la Figura 6 se compara este indicador para los distintos esquemas, observándose que cualquiera de los métodos BDF estudiados requieren por lo menos un orden de magnitud mayor de tiempo de cálculo que los esquemas explícitos. Esto es esperable ya que los esquemas implícitos cargan con una iteración interna en cada paso de tiempo para resolver las raíces del sistema algebraico asociado. Esta penalidad debe tenerse en cuenta en el balance general de flexibilidad y performance.

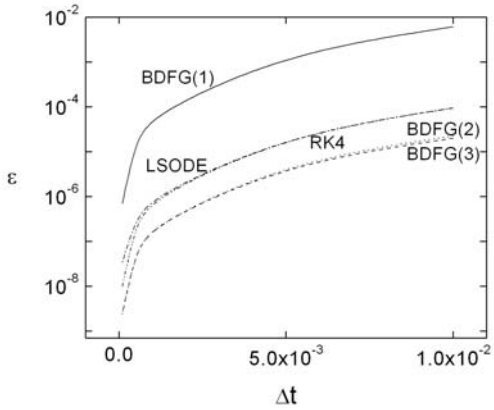

Figura 5: Error cuadrático.

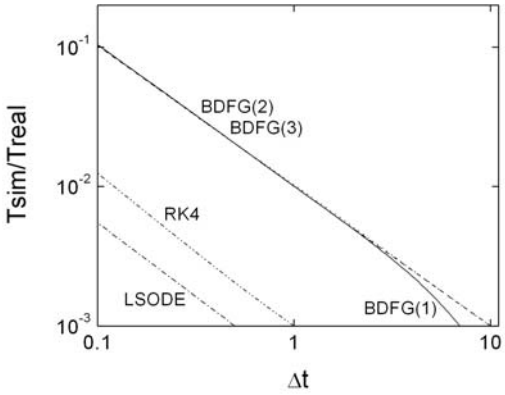

Figura 6: Eficiencia de cálculo.

Caso 2: Sistema no lineal conservativo

El segundo caso de estudio es el péndulo elástico Figura 7. Este es un sistema de ecuaciones diferenciales no lineales de cuarto orden, cuyas variables de estado natural son la longitud del brazo elástico, el ángulo de inclinación respecto a la vertical, y sus respectivas derivadas temporales; esto es:

$$\begin{aligned} z &= \dot{r} \\ w &= \dot{\theta} \\ \dot{z} - rw^2 &= -\frac{k}{m}(r - L) + g\ \cos(\theta) \\ r\dot{w} + 2zw &= -g\ sen(\theta) \end{aligned} \tag{18}$$

Dado que en este caso no existe solución analítica, se compararon las soluciones obtenidas con los esquemas numéricos utilizados en el caso 1. En la Figura 8 se muestra la evolución de la posición de la masa para los siguientes parámetros y las siguientes condiciones iniciales:
$r(0) = 1,\ \theta(0) = \pi / 2,\ \dot{r}(0) = 0,\ \dot{\theta}(0) = 0,\ k = 7,\ L = 1,\ m = 0.1,\ g = 9.8$.

En la Figura 9 se muestra la diferencia cuadrática de la solución obtenida con BDF con multipaso de tercer orden con las obtenidas con Runge-Kutta de cuarto order y con el paquete LSODE. Puede verse que hay una pequeña deriva creciente pero los valores son muy bajos. En la Figura 10 se muestra el cociente de los tiempos de cálculo y el tiempo simulado. Puede observarse nuevamente que al igual que en el caso 1, los métodos BDF tienen un costo computacional de un orden de magnitud mayor que los métodos explícitos.

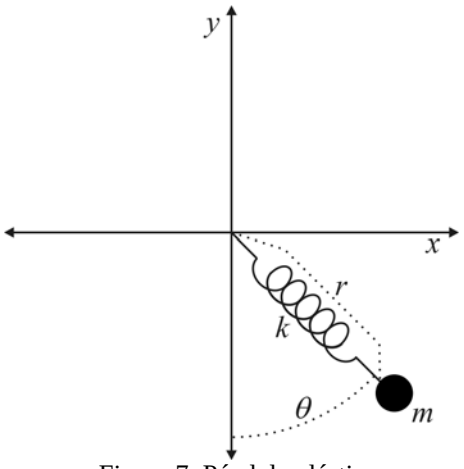

Figura 7: Péndulo elástico.

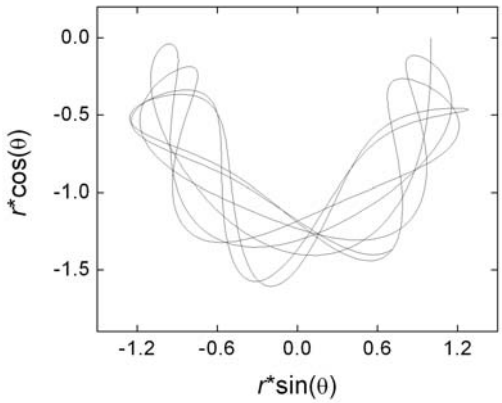

Figura 8: Posición de la masa.

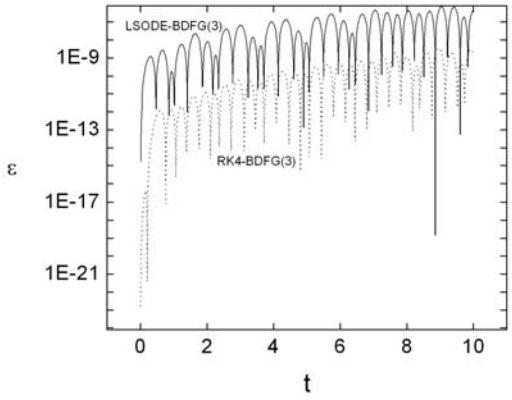

Figura 9: Error cuadrático.

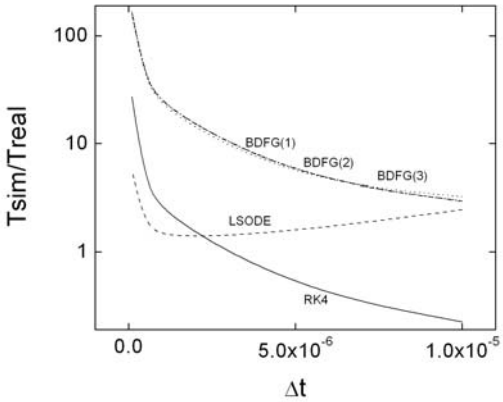

Figura 10: Eficiencia de cálculo.

Caso 3: Control de nivel de un sistema de tanques.

Finalmente se estudió un sistema de control de nivel de tanques en serie, como muestra la Figura 1. Este caso es apropiado para analizar las ventajas de implementación del esquema BDF en un proceso de modelado, en el que se parte de un modelo base simple (control proporcional en este caso) y se va elevando la complejidad progresivamente en un proceso circular de prueba y error. Para este estudio se implementaron dos esquemas siguiendo el paradigma de O-O, uno utilizando el método BDF y otro utilizando un esquema explícito Runge-Kutta. Usando como herramientas las dos implementaciones, se fueron aplicando distintas modificaciones al modelo, evaluando en cada caso el grado de flexibilidad del software.

La importancia de cada modificación fue caracterizada por los siguientes parámetros:

- Cantidad de ecuaciones del modelo afectadas a una modificación (r1).
- Cantidad de ecuaciones a agregar al modelo luego de aplicar los pasos algebraicos necesarios (r2).

La adaptabilidad del software ante cada modificación del modelo se caracterizó con los siguientes indicadores:

- Cantidad de líneas de código agregadas (r3).
- Pasos algebraicos intermedios necesarios para implementar la modificación al modelo (r4).

Como modelo base se tomó un sistema de dos tanques y dos bombas con un control proporcional clásico. La dinámica de este sistema puede describirse con dos ecuaciones diferenciales para los volúmenes de agua y dos ecuaciones algebraicas para los caudales:

$$\begin{aligned} \dot{v}_1 &= q_1 - q_2 \\ \dot{v}_2 &= q_2 - q_1 \\ q_1 &= p_1\left(v_{ref} - v_1\right) \\ q_2 &= p_2\left(v_{ref} - v_2\right) \end{aligned} \tag{19}$$

Donde se ha adoptado un control proporcional de los volúmenes, buscando mantener un nivel de referencia fijo v_{ref} .

Este modelo base se extendió a sistemas de 3 y 4 tanques, y a cada extensión se impusieron las siguientes modificaciones:

a) Agregar términos de control derivativo.
b) Introducir una dependencia de los coeficientes en función de las variables de estado.
c) Transformación de ecuaciones algebraicas en ecuaciones diferenciales.

Las condiciones iniciales y los valores de los parámetros fijos en cada caso particular se detallan en la Tabla 1. En cada caso se calcularon los indicadores r_i, los cuales se detallan en la Tabla 2.

Puede observarse que en general la implementación BDF reduce significativamente el número de líneas de código y la cantidad de pasos algebraicos intermedios. Sin embargo, como se encontró en los casos anteriores, esta ventaja tiene un costo de cálculo, lo cual hace necesario ponderar la importancia de ambos esquemas (*i.e.* modificabilidad-extensibilidad vs tiempo de ejecución). El punto de equilibrio de este balance está determinado por el tamaño del problema a resolver. Si la cantidad de ecuaciones del modelo es grande la modificabilidad y extensibilidad son el factor preponderante, mientras que en modelos de pocas ecuaciones no es significativa la ganancia en tiempo de desarrollo de software frente a la pérdida de velocidad de cálculo. Se proponen dos métricas para modelar la competencia entre estos dos elementos de calidad, métrica de software, S, y métrica de performance de cálculo, C, las cuales se definen a continuación:

S =Cantidad de líneas de código + Cantidad de pasos algebraicos

C = Tiempo simulado / Tiempo de cálculo

Modificaciones	Ecuaciones por tanque	*Condiciones iniciales*
a) Agregar términos de control derivativo	$q_i = p_i(v_{ref} - v_i) - \delta_i \dot{v}_i$ $\dot{v}_i = q_i - q_{out}$	$v_{ref} = 10,\ p_1 = 0.2,\ p_2 = 0.1, \delta_1 = 0.5$ $\delta_2 = 0.5, v_1(0) = 10.01, v_2(0) = 10$
b) Coeficientes en función de las variables de estado	$q_i = p_i(v_{ref} - v_i) - \delta_i \dot{v}_i$ $\dot{v}_i = q_i - q_{out}$ $p_i(v_1,..,v_n) = k_i(v_1 + .. + v_n)$ $\delta_i(v_1,..,v_n) = c_i(v_1 + .. + v_n)$	$v_{ref} = 10, k_1 = 50, k_2 = 100$ $c_1 = 20, c_2 = 20, v_1 = 10.01$ $v_2 = 10$
c) Transformación de ecuaciones algebraicas en ecuaciones diferenciales	$I_i \dot{q}_i + K_i q_i \lvert q_i \rvert = \Delta p_{A,i}$ $\Delta p_{A,i} = f(v_i) - g(\dot{v}_i) + \Delta p_{B,i}$ $f(v_i) = p_i(v_{ref} - v_i)$ $g(\dot{v}_i) = \delta_i \dot{v}_i$ $\Delta p_{B,i} = \Delta p_B(\dot{v}_i)$ $\dot{v}_i = q_i - q_{out}$	$v_{ref} = 10,\ p_1 = 0.2,\ p_2 = 0.1$ $\delta_1 = 0.5, \delta_2 = 0.5, K_1 = 1$ $K_2 = 1, I_1 = 2, I_2 = 2$ $\Delta p_B = 0.05, v_1 = 10.01, v_2 = 10$

Tabla 1

En la Tabla 2 se muestran los resultados de las métricas S y C para los distintos casos del sistema de tanques. La Figura 11 muestra la dependencia de los indicadores con el número de ecuaciones involucradas en el modelo, lo cual se toma como una medida del tamaño del problema. Se puede ver que el esquema BDF tiene menor C que el esquema Runge-Kutta, mientras que lo opuesto ocurre con S.

El esquema óptimo dependerá de la importancia relativa que se le da a las métricas C y S. Este tipo de balance suele resolverse tradicionalmente con una función utilidad, que establece un nivel de subjetivo de utilidad percibido por los usuarios dado un par de valores C y S. Para el presente análisis se propone usar una función utilidad del mismo tipo que las utilizadas en microeconomía, esto es:

$$U = \exp(-S - \alpha C) \tag{20}$$

Donde α representa la importancia relativa de la performance de cálculo frente a la flexibilidad del software. Para cada valor de α el esquema que tenga mayor utilidad será la mejor alternativa. En la Figura 12 se muestra el mapa de decisión óptima calculado para el sistema de tanques, en el plano formado por α y el tamaño del sistema. Puede verse que valores de α mayores que 10 convendrá implementar esquemas explícitos en modelos limitados a pocas ecuaciones, mientras que para modelos con un número de ecuaciones mayor que cierto límite lo más conveniente será la implementación de esquemas implícitos. Puede

observarse que el valor crítico que separa ambas alternativas es menor cuanto menor es el valor de α. Es decir, cuanta más importancia se le da a la flexibilidad y modificabilidad del software, menor será el tamaño mínimo de los modelos que convendrá implementar con esquemas implícitos que son más naturales para la programación O-O. Para valores de α menores que 10 siempre convendrá una implementación O-O con esquemas implícitos.

#Tanques	Modificaciones	BDFG						RK4					
		r1	r2	r3	r4	S	C	r1	r2	r3	r4	S	C
1	a)	2	0	2	0	2	0.196	2	0	2	3	5	0.005
	b)	4	2	4	0	4	0.198	4	2	4	3	7	0.006
	c)	6	3	6	0	6	0.21	6	3	6	4	10	0.007
2	a)	4	0	4	0	4	0.198	4	0	4	6	10	0.006
	b)	8	4	8	0	8	0.212	8	4	8	6	14	0.007
	c)	12	6	12	0	12	0.213	12	6	12	8	20	0.007
3	a)	6	0	6	0	6	0.196	6	0	6	9	15	0.006
	b)	12	6	12	0	12	0.208	12	6	12	9	21	0.008
	c)	18	9	18	0	18	0.21	18	9	18	12	30	0.008
4	a)	8	0	8	0	8	0.212	8	0	8	12	20	0.007
	b)	16	8	16	0	16	0.211	16	8	16	12	28	0.008
	c)	24	12	24	0	24	0.214	24	12	24	16	40	0.010

Tabla 2.

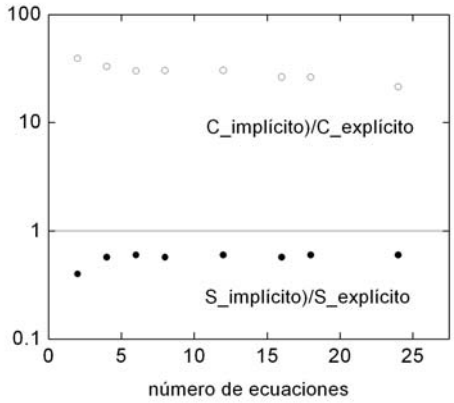

Figura 11: Relaciones entre S y C.

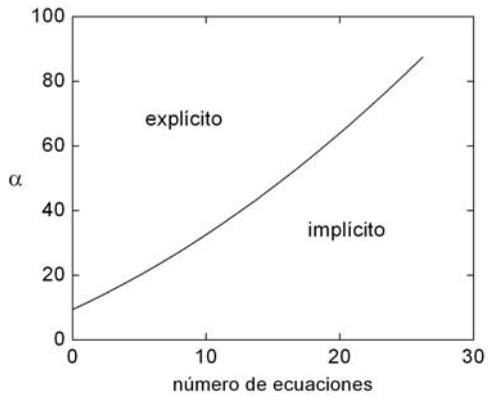

Figura 12: Mapa de decisión.

Al igual que en economía, el valor de α en cada caso puede estimarse por medio del impacto relativo de cada factor (C y S) en la utilidad. En este contexto, α es el número de unidades de S equivalentes a una unidad de C, respecto de la utilidad del sistema para desarrollar modelos.

6 CONCLUSIONES

Se estudió la aplicación de diseños O-O como una forma de mejorar el desarrollo de programas para problemas de simulación continua. El análisis y diseño ha considerado las dificultades que afectan a los usuarios modelistas y los desarrolladores de bibliotecas, tanto en los aspectos numéricos como de software.

Se diseñó una estructura de clases que provee una solución rápida para la implementación de problemas de simulación continua, poniendo énfasis en su representación natural a través de sistemas DAE, los cuales en su mayoría emplean ecuaciones diferenciales de balances.

Aunque el documento ha enfocado más a la solución de problemas de modelado que a la eficiencia pura, sabemos que esta última puede llegar a ser importante cuando el tamaño del problema aumenta. Referente a esto, vimos que un cierto grado de pérdida de eficiencia es necesario para alcanzar la flexibilidad necesaria. Sin embargo, los resultados obtenidos hasta ahora demuestran que uno puede negociar fácilmente flexibilidad y la generalidad de un diseño orientado al modelado por la eficacia de cálculo. Los conceptos de herencia y encapsulación desempeñan un papel importante, haciendo que el programa sea mucho más legible y modular que en una programación procedural, promoviendo definitivamente la modificación, extensión, mantenimiento, y reutilización del código.

7 REFERENCIAS

[1] Brenan K., Campbell S., and Petzold L. 1996. Numerical Solution of Initial Value Problems in Differential-Algebraic Equations. SIAM, Philadelphia, PA.

[2] Kees C. and Miller C. 1999. C11 Implementations of Numerical Methods for Solving Differential-Algebraic Equations: Design and Optimization Considerations. ACM Transactions on Mathematical Software, 25, 377–403.

[3] Gear C. 1971. The Simultaneous Numerical Solution of Differential-Algebraic Equations. IEEE Trans. Circuit Theory CT-18, 89–95.

[4] Meyer B. Object-Oriented Software Construction. Prentice Hall, 2000.

[5] Jackson R. and Sacks-Davis R. 1980. An Alternative Implementation of Variable Step-Size Multistep Formulas for Stiff ODEs. ACM Transactions on Mathematical Software, 6, 295–318.

[6] LSODE, the Livermore Solver for Ordinary Differential Equations, California.

[7] Langtangen H. and Munthe O. 2001. Solving Systems of Partial Differential Equations Using Object-Oriented Programming Techniques with Coupled Heat and Fluid Flow as Example. ACM Transactions on Mathematical Software 27, 1–26.

[8] Machiels L. and Deville M. 1997. Fortran 90: An Entry to Object-Oriented Programming for the Solution of Partial Differential Equations. ACM Transactions on Mathematical Software, 23, 32–49.

[9] Jinn-Liang L., Ing-Jer L., Miin-Zhih S., Ren-Chuen C, Mao-Chung H. 1996. Object-oriented programming of adaptive finite element and finite volume methods. Applied Numerical Mathematics 21, 439-467.