Universidad del Bío-Bío
Facultad de Ciencias Empresariales
Departamento de Sistemas de Información

Profesor Guía: Pedro G. Campos Soto.

Estudio e Implementación de Componentes de Graficación y Operatoria en Aritmética Intervalar.

"Trabajo de Titulación presentado en conformidad a los requisitos para obtener el título de Ingeniero Civil en Informática"

Cristian A. Bravo Robin.
Claudio F. Garcés Canobbi.

Martes, 30 de Diciembre de 2008.

Agradecimientos

Agradezco a mi madre y a mi padre por darme la oportunidad de desarrollarme como profesional y como persona. A mis amigos que me ayudaron a llegar donde estoy, y agradezco también a Dios por ayudarme hasta el día de hoy en todas las decisiones que he tomado a lo largo de mi vida.

Agradezco también al profesor Pedro Campos por la confianza y apoyo que depositó en este proyecto, el cual con su ayuda concluyó satisfactoriamente.

Cristian

Agradecimientos

Quisiera agradecer a Dios, mi familia, amigos y todos aquellos que me apoyaron y en mi confiaron, en especial a mis padres y mi hermano Ricardo, por su innegable apoyo durante estos años de estudio. Quisiera agradecer además a mi tía Silvia, a quien guardo gran cariño.

Agradezco a Jaime, un muy buen amigo quien siempre me motivo para lograr mi sueño de ser Ingeniero.

Agradezco al profesor Pedro Campos quien en mi confió para realizar este trabajo, agradezco su tiempo y dedicación además agradezco al profesor Manuel Crisosto, quien a través de los años me confió su amistad.

Finalmente, y no por eso menos importante, quisiera agradecer a Cristian, gran amigo que confió en mí como compañero en este camino que hoy termino.

Claudio

Le dedico este trabajo a mis padres, por cuyo esfuerzo y apoyo estoy realizando mi sueño de convertirme en un ingeniero.

Cristian

Quisiera dedicar este trabajo a Bernardina, mi madre, quien siempre me motivó para que estudiara e hiciera realidad mis sueños. Espero no defraudarla.

A Juan, mi padre, quien siempre me dijo que la educación es la mejor herencia que se le puede dejar a un hijo y quien siempre me inspiró para ser una mejor persona.

Ojala siempre estén orgullosos de su hijo, como yo siempre estaré orgullosos de ellos.

Claudio

Resumen

La *Aritmética Intervalar* permite resolver el problema de optimización global, el cual consiste en encontrar el punto (o los puntos) en el cual una función es *óptima*, ósea, se busca el punto en el cual una función alcanza su valor mínimo (o máximo) al ser evaluada. Existen diversos métodos para obtener (o tratar al menos) óptimos globales, pero esta investigación esta centrada en el método de obtención de un mínimo (o máximo) global basado en la *Aritmética Intervalar*, pues ésta ofrece resultados garantizados.

Los estudios sobre aritmética intervalar, así como los campos en los que se pueden aplicar estos estudios se han ampliado más allá de las fronteras de las matemáticas en el último tiempo, por lo que se hace imperativo poseer una herramienta que permita trabajar con la aritmética intervalar como ayuda a la resolución del problema de optimización global.

Una de las principales carencias que presenta la librería de aritmética intervalar que ya se encuentra implementada, dentro del departamento de Sistemas de Información de la Universidad del Bío-Bío, es la ausencia de una forma de representar tanto los intervalos como las operaciones que se pueden hacer con ellos; pues bien, el objetivo principal de este trabajo es el estudio y la implementación, en el lenguaje de programación, C/C++ de una librería grafica que utilice la aritmética intervalar que ya se encuentra implementada.

Este trabajo se centra en estudiar los fundamentos de la aritmética intervalar, su aplicación en la optimización global, y la librería de aritmética intervalar que ya se encuentra implementada. Junto a esto, se abordan los conceptos de Desarrollo Basado en Componentes y la Orientación a Objetos, para a partir de esto diseñar los operadores gráficos que se necesiten.

Índice

Capítulo 1: Introducción

1.1 Origen del Proyecto

El concepto de *números intervalos* o *intervalos* simplemente, es un concepto conocido por los matemáticos hace ya bastante tiempo, pero no fue hasta la elaboración de la tesis de doctorado de Ramon E. Moore, en la década de 1960, en la Universidad de Stanford en la cual se mostró un nuevo punto de vista; dicha tesis se desarrolló en el contexto de la estimación de errores en el análisis numérico y su mayor aporte fue el planteamiento de una *aritmética intervalar*.

Los estudios sobre aritmética intervalar (Moore, 1959; Moore, 1966; Hansen, 1965; Nickel, 1966) han aumentado en el último tiempo, ya que los campos en los que se pueden aplicar estos estudios se han ampliado mas allá de las matemáticas (Sahinidis, 2004), aplicándose en campos tan diversos como son la ingeniería, la química, los sistemas financieros, la economía y la biotecnología (Muñoz y Peña, 2007).

Encontrar el punto (o los puntos) en el cual una función es *óptima*, es la razón de ser del llamado *problema de optimización global*, que consiste en buscar el valor mínimo (o máximo) que puede tomar una función al ser evaluada. Existen diversos métodos para obtener (o tratar al menos) óptimos globales, pero esta investigación muestra la *Aritmética Intervalar* y la ayuda que presta en la obtención de un mínimo (o máximo) global, pues dicha aritmética ofrece resultados garantizados.

Aparte del sistema de notación que poseen los números intervalos, éstos se pueden representar gráficamente, generalmente en un plano cartesiano. Dicha representación gráfica no es solo una forma de expresar y a la vez entender mas clara y rápidamente el concepto detrás de un intervalo, sino que además permite mostrar como se comportan

dichos intervalos al aplicar las operaciones que con ellos están definidas, tales como la suma, resta, multiplicación y división.

Este trabajo trata sobre el estudio de la aritmética intervalar y la implementación de una librería en lenguaje C++ que permita la representación gráfica de los intervalos, además realizar y mostrar operaciones intervalares de suma, resta, multiplicación, división, unión, intersección y también muestre los resultados de la optimización global por intervalos.

1.2 Formulación del Problema

1.2.1 El Problema

La aritmética intervalar es una potente herramienta para, por ejemplo, solucionar el problema de optimización global; el cual consiste en encontrar el o los puntos en el cual(es) una función es óptima, o sea, genera un valor mínimo (o máximo) al ser evaluada. Esta definición puede sonar un poco distante, pero si se piensa en la optimización global como la forma de encontrar la mejor respuesta posible frente a preguntas (*modelos*) que tienen numerosas soluciones posibles, tal vez esto aterrice el concepto.

Habitualmente, tanto ingenieros como investigadores están obligados a realizar costosos gastos extras de recursos, como lo son el tiempo y el dinero para obtener conclusiones o, mejor aún, soluciones que son consideradas cercanas al óptimo, pero en ningún caso lo representan. Esta necesidad de determinar óptimos globales se presenta en numerosas disciplinas que modelan sistemas del mundo real y en áreas tan diversas como el estudio de sistemas físicos y químicos, problemas de localización y transporte, sistemas de planificación de producción, asignación de recursos en sistemas financieros, y diseño en ingeniería (Sahinidis, 2004; Campos y Valdés-González, 2006). Por ejemplo, un corredor de acciones en la bolsa podría necesitar calcular el óptimo de su cartera de inversiones; un ingeniero aeronáutico que quiere construir un ala de avión desearía saber qué ángulos

deben llevar las piezas al unirlas entre sí para obtener la mayor resistencia del ala sin incrementar la resistencia al viento; un químico querría un *mejor ajuste* del espectro de un fotoelectrón como la suma de las curvas Gaussianas que representan los espectros individuales de varias sustancias (Muñoz y Peña, 2007; Leclerc, 1992).

Los cálculos asociados para determinar un óptimo global son, muchas veces, complejos y extensos. Estos cálculos generan, habitualmente, un resultado decimal o se realizan mediante el uso de *números decimales* y es en este punto donde este problema se asocia a la computación y la informática. La forma en la que se manipulan y almacenan los números decimales en el procesador de un equipo está definida y es reducida, pues dependiendo de la *arquitectura* a la cual se rija, va a depender el tamaño de la parte decimal que podemos almacenar. Entonces si pensamos que un procesador almacena 20 dígitos de la parte decimal y el número con el cual se quiere trabajar tiene 25 dígitos en la parte decimal, entonces este número se trunca o se corta en el digito 20 o se aproxima, generando el llamado *Error de Redondeo*. Una forma de solucionar esto es con el uso de los números intervalos.

Dentro del departamento de Sistemas de Información de la Universidad del Bío-Bío, en el cual se encuentra adscrito este estudio, se han desarrollado ya varios estudios y/o proyectos sobre el problema de optimización global, bajo el alero del profesor Pedro Gerónimo Campos Soto, y la ayuda que presta la aritmética intervalar en la búsqueda de una solución. Estos estudios e implementaciones están basados en distintos métodos y algoritmos así como también en las diferentes formas de procesamiento de datos. Fruto de esto, se encuentra implementada ya una librería (Campos, 2004; Campos 2005) que manipula los números intervalos y su aritmética asociada, para la cual se desarrolló una forma de mostrar los resultados de forma gráfica.

Se dice que una imagen vale más que mil palabras, pues esta máxima es el problema y a la vez la motivación de esta investigación. Conceptos tan básicos, como por ejemplo ¿qué es un intervalo?, quedan mejor expresados o definidos con una simple representación

gráfica de los mismos, así también conceptos mas complejos, como lo son las diversas operaciones que se pueden realizar con los intervalos, se pueden explicar y a la vez absorber mejor si se representan gráficamente y lo mismo sucede con los óptimos globales que también se presentaran gráficamente.

Para definir mas precisamente y a la vez acotar el trabajo que persigue esta investigación, se plantean los objetivos a continuación.

1.3 Objetivos de Proyecto

1.3.1 Objetivo General

El objetivo principal de este trabajo es el estudio y la implementación en C++ de una librería de representación gráfica de números intervalos, que utilice la aritmética intervalar. Dicha librería será extendida con nuevos operadores y componentes de graficación que faciliten el uso y comprensión a cualquier usuario.

Este proyecto se enfocará en estudiar y ampliar la librería de aritmética intervalar que ya se encuentra implementada, añadiéndole operaciones no menos importantes como la unión, la intersección y la división por cero y a partir de ahí diseñar los operadores gráficos que se necesiten.

Posteriormente se construirán los componentes de una librería en C++ que permitan representar gráficamente la aritmética intervalar y sus operaciones.

1.3.2 Objetivos Específicos del Trabajo

- Realizar un estudio de la aritmética intervalar, considerando sus usos y funcionamiento.
- Realizar un estudio de la optimización global utilizando la aritmética intervalar, como aplicación práctica de la librería.
- Realizar un estudio de la librería de aritmética intervalar que se encuentra previamente implementada
- Diseñar los operadores que no fueron considerados en la implementación de la librería de aritmética intervalar original, específicamente la unión, la intersección y la división por cero.
- Realizar un estudio de las librerías gráficas disponibles para el lenguaje C++.
- Estudiar las bases de la ingeniería de componentes.

- Diseñar, implementar, integrar, probar y depurar tanto los componentes gráficos como los componentes aritméticos, en el lenguaje de programación C/C++, para la librería de Aritmética Intervalar ya implementada.

1.4 Conceptos Asociados

1.4.1 Número Intervalo

Concepto introducido en la década de los sesenta, que plantea la creación de un nuevo número, como los reales, naturales o complejos, que comprende a los infinitos números entre dos reales llamados extremos.

1.4.2 Aritmética de Intervalos

Ramón Moore desarrolló una lista de operaciones posibles de aplicar a los números intervalos mencionados anteriormente, tales como la suma, resta, multiplicación y división además de operaciones como la unión y la intersección.

1.4.3 Error de Redondeo

Problemática que soluciona la utilización de la aritmética intervalar. Corresponde a la lógica que siguen los computadores para los cálculos sobre variables de punto flotante de precisión fija, que hacen imposible la representación de ciertos números reales en binario y acaban finalmente por truncar o redondear los resultados, llevando a resultados erróneos.

1.4.4 Optimización Global

Corresponde a la búsqueda del valor(es) óptimo(s) de una función dada a través de un algoritmo, que corresponde al mínimo (o máximo) global de la función. Si existe más de un mínimo global, el óptimo global corresponde al conjunto de los mínimos globales hallados. También existe la denominada optimización local, que se preocupa de encontrar

el conjunto de valores mínimos (o máximos) locales de una función dada. Entre los métodos y formas que existen para dicha labor se destaca la aritmética intervalar, pues ofrece resultados garantizados. Matemáticamente hablando, la optimización global consiste en encontrar los puntos donde la derivada de la función es 0

1.4.5 SDL

Librería gráfica multiplataforma y de código libre que se utilizó para la implementación de los componentes gráficos del presente proyecto.

1.4.6 Programación Orientada a Objetos

Esta programación tiene su origen en la década de los ochenta y corresponde a unos de los paradigmas de la programación, así como también lo es la programación estructurada. Las propiedades que caracterizan a este estilo de programación son la herencia, polimorfismo, encapsulación, abstracción y ocultación de la información. El lenguaje C/C++ permite realizar programación orientada a objetos.

1.5 Presentación de capítulos

Este trabajo se ha dividido en ocho capítulos, los cuales se resumen a continuación:

El capítulo 1 presenta el origen de este proyecto, el porqué de su implementación y los objetivos que se persiguen con él.

El capítulo 2 explica todo lo concerniente al tema de la aritmética intervalar, su origen que se remonta a la década de los sesenta, ejemplos gráficos de los intervalos en las distintas dimensiones en las que se le puede representar. Detalla y ejemplifica también la operatoria intervalar, así como también otras propiedades importantes de conocer. Menciona asimismo las funciones por intervalos y el error de redondeo que la aritmética de intervalos permite solucionar.

El capítulo 3 aborda el tema de la optimización global, mencionando por que su utilización y continua búsqueda de más y mejores algoritmos que permitan encontrar la mejor solución a un problema dado. Presenta a continuación un algoritmo básico de optimización global, que es utilizado en la librería de aritmética intervalar disponible.

El capítulo 4 revisa la historia del lenguaje de programación C/C++ y hace mención de la historia de la computación gráfica, para luego introducir la librería gráfica SDL que se utilizó para la implementación del sistema, mencionando sus características y ventajas.

El capítulo 5 aborda el tema de la ingeniería de componentes, conceptos y definiciones que finalmente derivan a la elección de un lenguaje orientado a objetos como el paradigma de programación, que se eligió para el desarrollo del proyecto.

El capítulo 6 describe la plataforma de optimización global mediante la aritmética intervalar que se desarrolló de forma previa a este trabajo. También se explican los cambios y modificaciones que se hicieron a la librería para su mejor desempeño, tales como la implementación de la unión, la intersección y la división por cero.

El capitulo 7 hace mención a la librería gráfica y los componentes que se construyeron como parte de este trabajo, además se señalan detalles de su implementación.

El capítulo 8 menciona finalmente, algunas conclusiones que se obtuvieron durante el transcurso del trabajo realizado.

Capítulo 2: Aritmética Intervalar

2.1 Número Intervalo

Ramón Edgar Moore en la década de los sesenta plantea que un intervalo es un número nuevo que nace de la conjunción de un par de números reales. Estos números se representan con el extremo inferior o izquierdo y superior o derecho.

Se puede entender un intervalo como el conjunto de números reales comprendidos entre dos números reales llamados cotas, así el intervalo [a,b] será el conjunto de números reales entre a y b, incluyéndolos.

Formalmente se puede expresar: I = [a,b], a,b $\in \mathbb{R}$ / $\exists x, x \in [a,b] \Rightarrow a \leq x \leq b$.

Si se considera a los números reales *a* y *b*, donde a es menor que b, entonces todos los números reales que se encuentran entre a y b, incluidos, conforman el *número intervalo* [a,b]. De ahora en adelante se usará indistintamente *intervalo* y *número intervalo* para referirse al mismo concepto.

Todo numero real “x” puede ser representado como intervalo de la forma [x,x]. Este intervalo se conoce como intervalo degenerado y posee ancho nulo. (Hansen, 1992).

Para representar un intervalo se utiliza una letra mayúscula y las variables reales con una letra minúscula, así el intervalo “A” se representaría de la siguiente manera:

A = [2,5], y expresado de otra forma seria: [2,5] = a: $2 \leq a \leq 5$; sin embargo si se quiere utilizar las mayúsculas para representar ambos casos, la que represente un intervalo deberá ir con el superíndice “I” de la siguiente manera: A^I.

Si se considera que cada cantidad que se analizará es un intervalo se puede omitir el superíndice “I”.

Los extremos, izquierdo y derecho de un intervalo se representarán con un subíndice “L” y “R” respectivamente (del inglés “Left” y “Right”). También existe una forma alternativa de representar un intervalo, utilizando una barra bajo el extremo izquierdo y una sobre el extremo derecho para referirse a los extremos izquierdo (o inferior) y derecho (o superior) respectivamente.

$X = [a,b]$

Extremo izquierdo = X^L (Notación alternativa: $\underline{a}$)

Extremo derecho = X^R (Notación alternativa: $\overline{b}$)

Un intervalo $X = [a,b]$ será: positivo (o no negativo) si $a \geq 0$, estrictamente positivo si $a > 0$, negativo (o no positivo) si $b \leq 0$ y estrictamente negativo si $b < 0$. (Hansen, 1992). Los intervalos $X = [a,b]$ y $Z = [c,d]$ serán iguales sí y solo sí $a = c$ y $b = d$.

Además Moore definió las distintas operaciones que se pueden realizar con dichos números como la suma y resta. Esto para poder manejar el error de redondeo (Moore, 1966).

2.2 Dimensiones de intervalos

La forma que adopta un intervalo, depende del número de dimensiones en las que se esté trabajando (1D, 2D, 3D). Esto a la vez, depende del tipo de problema que se está desarrollando. Un intervalo unidimensional, o sea de una sola dimensión, se representa sobre el eje X. Un intervalo bidimensional se representa sobre los ejes X e Y. De la misma manera, un intervalo tridimensional se representa sobre los ejes X, Y y Z. En otras palabras, un intervalo n-dimensional necesita n dimensiones para ser representado.

En el presente trabajo se presentan algunos ejemplos gráficos, la mayoría en una y dos dimensiones para su fácil entendimiento.

A continuación se pueden observar las formas de representar gráficamente los intervalos, unidimensionales y multidimensionales.

2.2.1 Intervalo unidimensional

Es el intervalo de más fácil interpretación y el más utilizado en el estudio de la aritmética intervalar. Si se considera el intervalo unidimensional [-4,3], su representación gráfica sería como la de la figura 2.1.

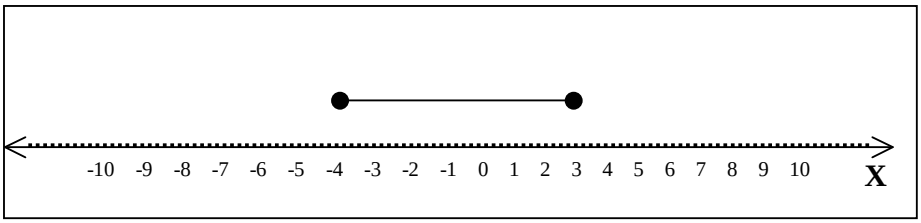

Figura 2.1: Intervalo de 1 dimensión.

2.2.2 Intervalo bidimensional

Este intervalo de dos dimensiones se representa en el plano cartesiano sobre los eje X e Y. Considérese como ejemplo el intervalo $\begin{bmatrix}[-3,6]\\[-4,4]\end{bmatrix}$ representado en la figura 2.2, donde [-3,6] es el intervalo sobre el eje X y [-4,4] es el intervalo sobre el eje Y.

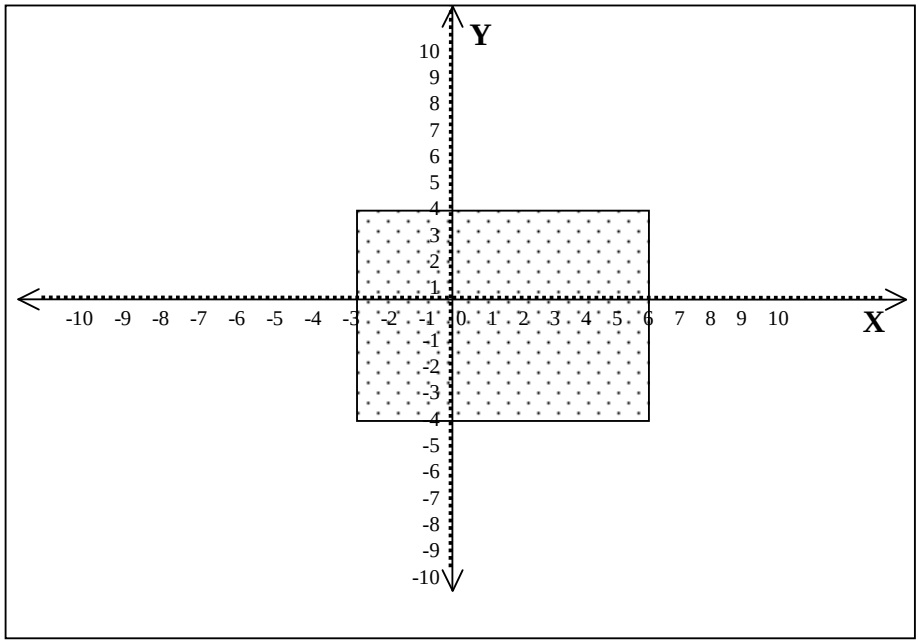

Figura 2.2: Intervalo de 2 dimensiones.

2.2.3 Intervalo tridimensional

Este intervalo se representa sobre un plano 3D como se muestra en la figura 2.3, representando el intervalo sobre los ejes X, Y y Z. Considérese como ejemplo el intervalo $\begin{bmatrix}[-3,5]\\[-4,3]\\[0,4]\end{bmatrix}$, donde [-3,5] es el intervalo sobre el eje X, [-4,3] es el intervalo sobre el eje Y y [0,4] es el intervalo sobre el eje Z.

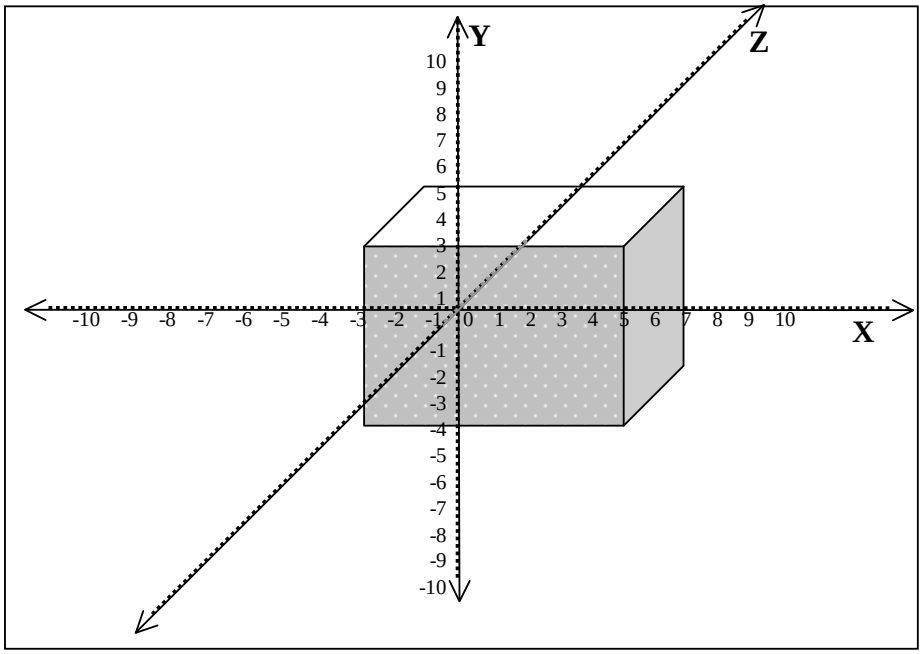

Figura 2.3: Intervalo de 3 dimensiones.

2.3 Propiedades de un Intervalo

El ancho de un intervalo X= [a,b] es un número real no negativo, se representará como *w*(X) (de ”ancho” en ingles: “width”) (Vivallos, 2007) y será definido como:

$$w(X) = b - a$$

Supóngase el intervalo X= [2, 8], su ancho sería entonces: 8 - 2 = 6, como se puede observar en la figura 2.4.

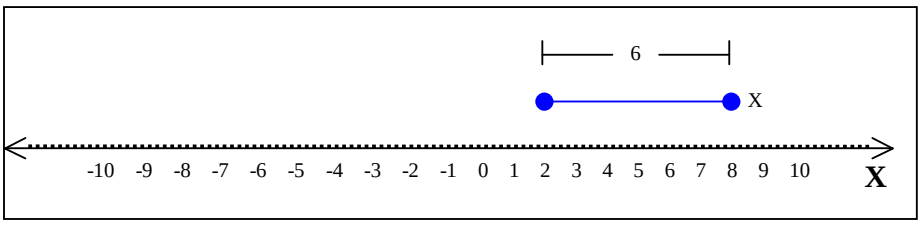

Figura 2.4: Ancho de un intervalo.

De la definición anterior podemos corroborar lo que se dijo en el capitulo 2.1 sobre los intervalos degenerados y su ancho igual a cero.

El centro de un intervalo X= [a,b] está definido como el punto medio entre sus extremos y se representa por:

$$c(X) = \frac{a+b}{2}$$

Supóngase el intervalo X= [-4,4], su centro sería entonces: $\frac{-4+4}{2} = 0$, como se puede observar en la figura 2.5.

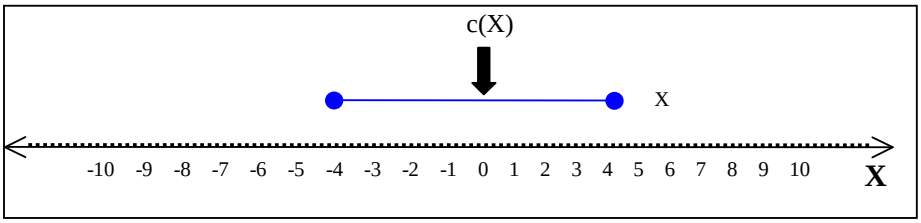

Figura 2.5: Centro de un intervalo.

Supóngase ahora el intervalo bidimensional X= $\begin{bmatrix}[0,5]\\[3,9]\end{bmatrix}$, su centro corresponde al centro de cada intervalo que lo compone, dando como resultado el vector $\begin{bmatrix}2,5\\6\end{bmatrix}$ como se muestra en la figura 2.6. Donde 2,5 es el centro del intervalo [0,5] y 6 es el centro del intervalo [3,9]. La misma lógica se utiliza para cualquier intervalo de n-dimensiones.

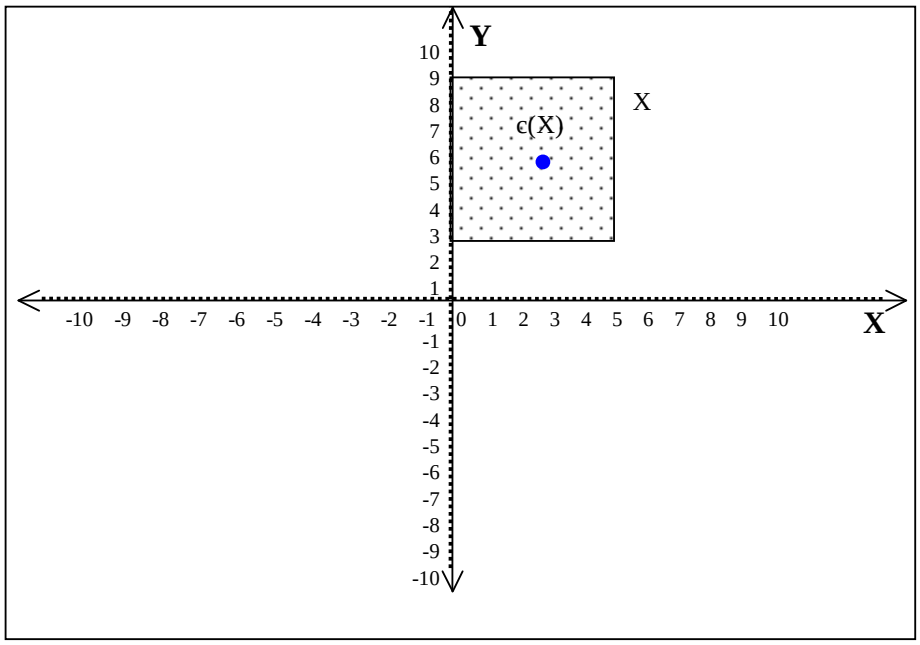

Figura 2.6: Centro de un intervalo bidimensional.

Ya teniendo tanto el ancho como el centro de un intervalo X = [a,b], podemos definir sus extremos como:

$$a = c(X) - \frac{w(X)}{2}$$

$$b = c(X) + \frac{w(X)}{2}$$

Considérese para lo anterior, el intervalo X= [-4,4]. Sus extremos serían entonces:

$$a = c([-4,4]) - \frac{w([-4,4])}{2} = 0 - 4 = -4$$

$$b = c([-4,4]) + \frac{w([-4,4])}{2} = 0 + 4 = 4$$

Así se puede observar en la figura 2.7.

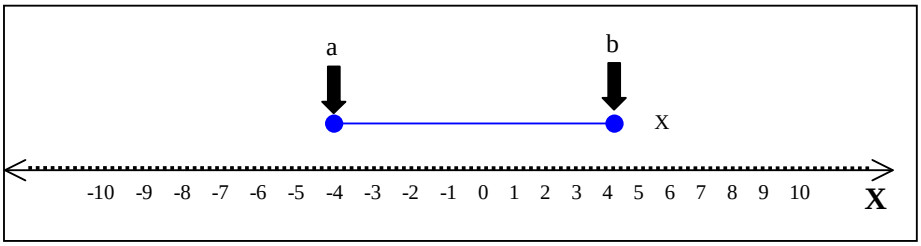

Figura 2.7: Extremos de un intervalo.

La distancia entre dos intervalos esta definida como:

$$d([\underline{x}, \overline{x}],[\underline{y}, \overline{y}]) = \max(|\underline{x} - \underline{y}|, |\overline{x} - \overline{y}|)$$

Para poder darle un orden parcial a los intervalos, se debe considerar:

$$[a,b] < [c,d] \leftrightarrow b<c$$

Considérese como ejemplo los intervalos [-2,1] y [4,8]. La distancia entre ambos sería entonces:

$$d([-2,1],[4,8]) = \max(6 , 7) = 7$$

2.4 Aritmética de Intervalos

La aritmética que creó Moore (Moore, 1966) es y ha sido utilizada tanto para resolver ecuaciones diferenciales ordinarias y sistemas lineales, como para verificación de caos y optimización global (Leclerc, 1992), (Moore, 1966), (Vivallos, 2007).

Esta aritmética define una operación cualquiera Θ, en donde Θ puede ser, +, -, *, /, y X e Y como intervalos cualesquiera que se deben calcular de la siguiente forma (Trafalis y Kasap, 2002), (Vivallos, 2007):

$$X \Theta Y = [\ min\ x\Theta y,\ max\ x\Theta y] \quad con\ x \in X\ ,\ y \in Y$$

Con esto se establece que el intervalo resultante de X Θ Y contiene cada número que resulta de xΘy para cada x∈ X y cada y∈ Y.

2.4.1 Suma

Dicho esto, se puede pensar que la aritmética intervalar es simplemente una aritmética de desigualdades. Por ejemplo, si se definen los intervalos: X= [a,b] e Y=[c,d], y $a \leq x \leq b$ y $c \leq y \leq d$, entonces se puede asegurar que $a+c \leq x+y \leq b+d$, así se puede deducir que la adición de intervalos es igual a:

$$X + Y = [a+c, b+d]$$

La figura 2.8 representa de forma gráfica la suma de los intervalos X=[-8,-3] e Y=[1,5], ambos intervalos unidimensionales, cuyo resultado es el intervalo [-7,2].

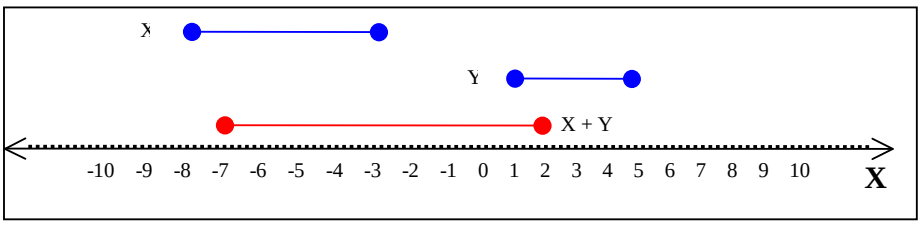

Figura 2.8: Suma de intervalos.

En el caso de intervalos multidimensionales, los cálculos anteriores se aplican a cada dimensión, como el siguiente ejemplo así lo muestra.

Considérese los intervalos $X=\begin{bmatrix}[-6,2]\\[-1,6]\end{bmatrix}$ e $Y=\begin{bmatrix}[1,8]\\[-6,2]\end{bmatrix}$, cuya suma debe interpretarse como $\begin{bmatrix}[-6,2]+[1,8]\\[-1,6]+[6,2]\end{bmatrix}$ cuyo resultado es el intervalo $\begin{bmatrix}[-5,10]\\[-7,8]\end{bmatrix}$. Lo anterior se puede observar en la figura 2.9, que corresponde a una captura de pantalla de la aplicación que se realizó en este trabajo, al igual que las imágenes en dos dimensiones que le siguen.

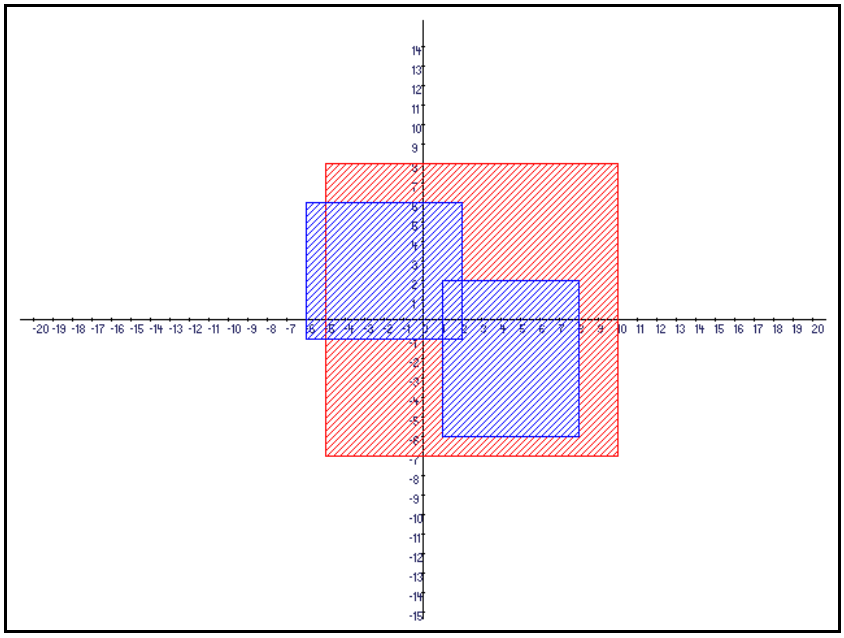

Figura 2.9: Suma de intervalos bidimensionales.

Como se aprecia en la figura 2.9 el intervalo X+Y, en color rojo, representa el resultado de todas las posibles combinaciones de suma que se pueden realizar con los intervalos X e Y, ambos de color azul, por eso este intervalo no necesariamente debe contener a los intervalos involucrados. Dichos colores se utilizaran a lo largo de este trabajo para representar tanto el resultado (rojo) como los intervalos involucrados en la operación (azul).

De la misma manera se pueden definir las demás operaciones básicas de la aritmética intervalar.

2.4.2 Resta de Intervalos

Tal como se definió para la suma y así para el resto de operaciones que se revisaran, dado los intervalos: X= [a,b] e Y=[c,d],entonces se define X-Y como:

$$X - Y = [a-d, b-c]$$

Considérese los intervalos X= [-3,2] e Y= [0,5], la resta de ellos sería:

$$X - Y = [-3-5, 2-0] = [-8,2]$$

Este ejemplo se puede observar en la figura 2.10 a continuación.

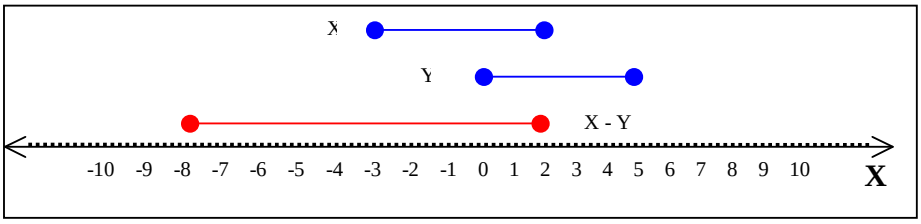

Figura 2.10: Resta de intervalos.

Considérese ahora los intervalos bidimensionales $X=\begin{bmatrix}[-3,2]\\[-1,6]\end{bmatrix}$ e $Y=\begin{bmatrix}[1,4]\\[-2,2]\end{bmatrix}$ cuya resta debe interpretarse como $\begin{bmatrix}[-3,2]-[1,4]\\[-1,6]-[-2,2]\end{bmatrix}$, cuyo resultado es el intervalo $\begin{bmatrix}[-7,1]\\[-3,8]\end{bmatrix}$. Lo anterior se puede observar en la figura 2.11 a continuación.

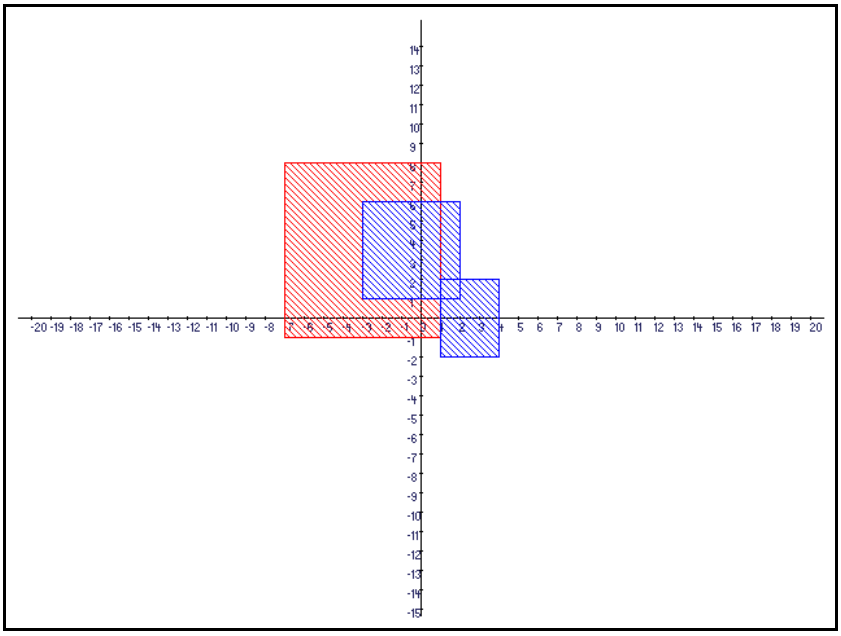

Figura 2.11: Resta de intervalos bidimensionales.

2.4.3 Multiplicación de Intervalos

$$X * Y = [min\{ac,ad,bc,bd\}, max\{ac,ad,bc,bd\}]$$

Considérese los intervalos X= [-1,3] e Y= [1,3], su multiplicación se expresa de la siguiente forma:

$$X * Y = [min\{-1,-3,3,9\}, max\{-1,-3,3,9\}] = [-3,9]$$

Lo anterior se observa en la figura 2.12 a continuación.

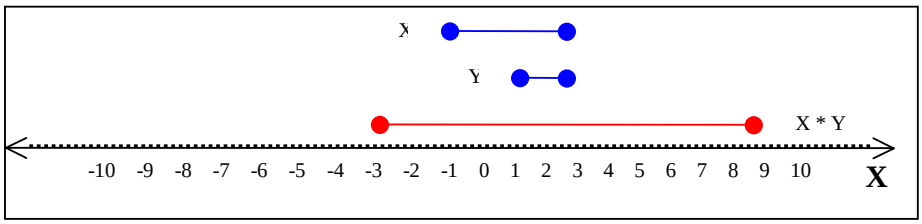

Figura 2.12: Multiplicación de intervalos.

Considérese ahora los intervalos bidimensionales $X=\begin{bmatrix}[2,4]\\[-1,3]\end{bmatrix}$ e $Y=\begin{bmatrix}[-1,2]\\[-2,3]\end{bmatrix}$, el resultado de su multiplicación, como se observa en la figura 2.13, es $\begin{bmatrix}[-4,8]\\[-6,9]\end{bmatrix}$.

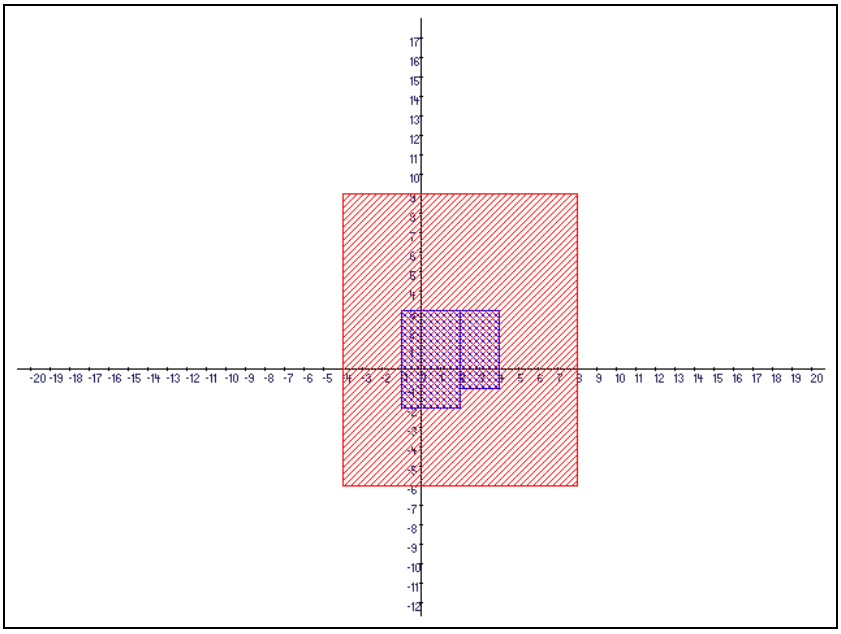

Figura 2.13: Multiplicación de intervalos bidimensionales.

2.4.4. División de Intervalos

$$X/Y = [a,b] * \left[\frac{1}{d}, \frac{1}{c}\right] si\ 0 \notin [c,d]$$

Considérese los intervalos unidimensionales X=[-2,6] e Y=[1,4], su multiplicación se expresa de la siguiente forma:

$$X/Y = [-2,6] * \left[\frac{1}{4}, \frac{1}{1}\right] = [-2,6]$$

Lo anterior se observa en la figura 2.14 a continuación.

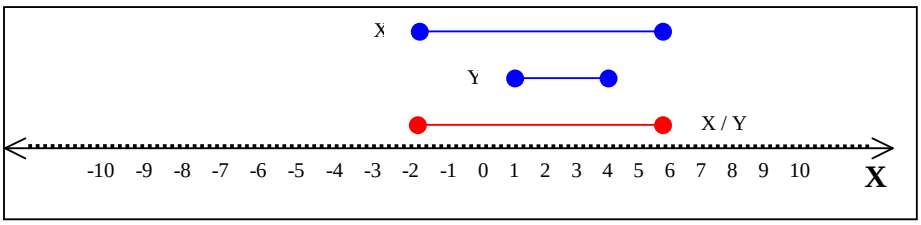

Figura 2.14: División de intervalos.

Considérese ahora los intervalos bidimensionales $X=\begin{bmatrix}[2,4]\\[6,9]\end{bmatrix}$ e $Y=\begin{bmatrix}[2,4]\\[1,3]\end{bmatrix}$, el resultado de la división es $\begin{bmatrix}[0.5,2]\\[2,9]\end{bmatrix}$ como se puede observar en la figura 2.15 a continuación.

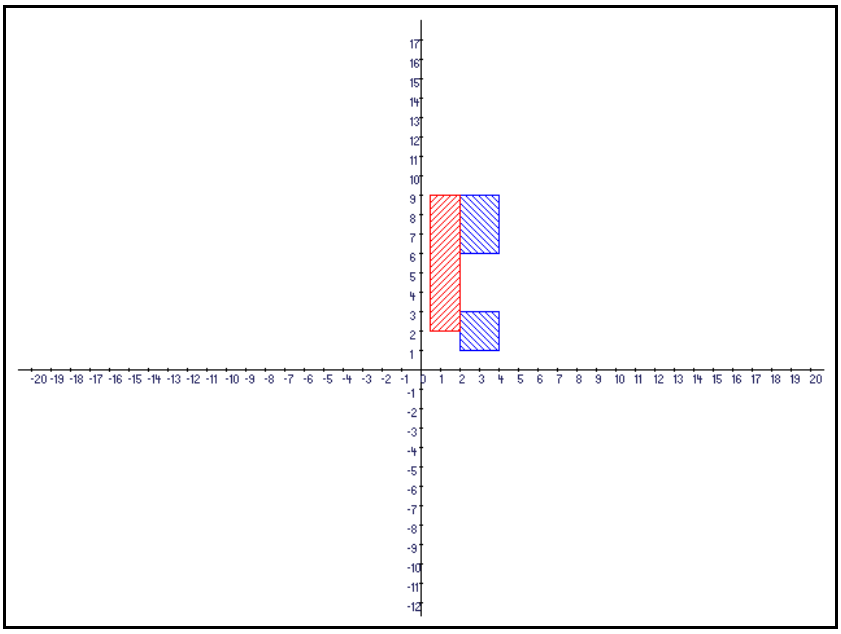

Figura 2.15: División de intervalos bidimensionales.

Cabe señalar que así como se mencionó que un número real podía ser representado como intervalo degenerado, ósea su ancho es igual a cero, se puede decir también que si se trabaja las operaciones antes mencionadas con dichos intervalos degenerados, se estaría utilizando aritmética real ordinaria (Vivallos, 2007).

Mención aparte merece la división por cero, que en realidad se refiere a la división en la cual el intervalo divisor contiene al cero, esto se explicara en el capitulo 6.1.5 (Modificaciones y Correcciones a la Librería de Aritmética Intervalar).

2.5 Otras propiedades

Como se definió a un intervalo como un conjunto de números, se puede aplicar también las operaciones que se encuentran definidas en la teoría de conjuntos, como lo son la unión y la intersección entre otras. Por lo cual se puede decir, $x \in [a,b]$, lo que significa que x es un numero real que está contenido dentro del intervalo [a,b], ósea, $a \leq x \leq b$. De la misma forma, $[a,b] \subset [c,d]$ significa que el intervalo [a,b] esta contenido como un conjunto dentro del intervalo [c,d], ósea, $c \leq a \leq b \leq d$. (Hansen, 1992).

Supóngase por ejemplo los intervalos X= [-4,3] e Y= [1,6], si se les aplica la operación de unión, como se observa en la figura 2.16 se obtiene el intervalo [-4,6]. Y si se les aplica la operación de intersección se obtiene el intervalo [1,3] como se puede observar en la figura 2.17.

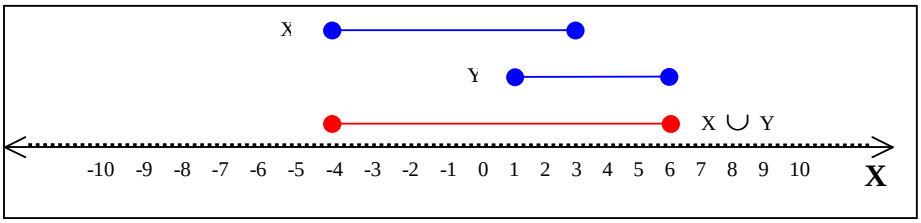

Figura 2.16: Unión de intervalos.

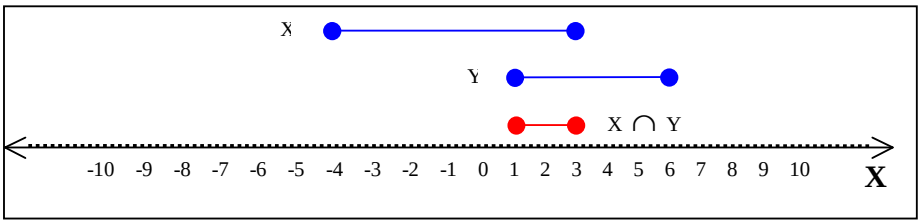

Figura 2.17: Intersección de intervalos.

En el caso que se consideren intervalos en dos dimensiones la situación es similar. Si se tiene a $X=\begin{bmatrix}[-4,3]\\[-1,2]\end{bmatrix}$ e $Y=\begin{bmatrix}[1,6]\\[4,8]\end{bmatrix}$, ósea, ampliamos intervalos anteriores en una dimensión se obtiene el intervalo $\begin{bmatrix}[-4,6]\\[-1,8]\end{bmatrix}$, tal como lo muestra la figura 2.18.

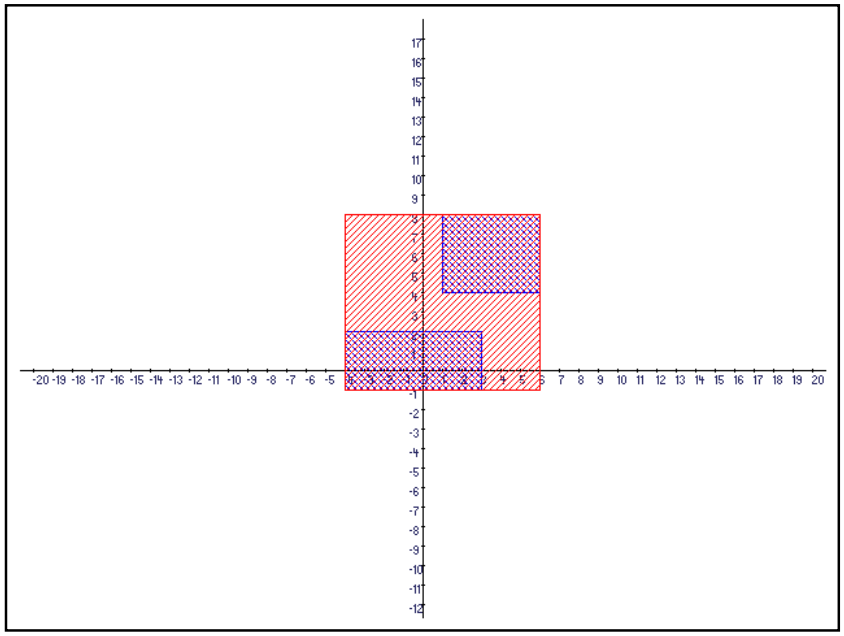

Figura 2.18: Unión de intervalos en 2 Dimensiones.

Lo mismo ocurre con la intersección, si se considera a $X=\begin{bmatrix}[-4,3]\\[-1,2]\end{bmatrix}$ e $Y=\begin{bmatrix}[1,6]\\[-4,6]\end{bmatrix}$, se obtiene el intervalo $\begin{bmatrix}[1,3]\\[-1,2]\end{bmatrix}$, tal como lo muestra la figura 2.19.

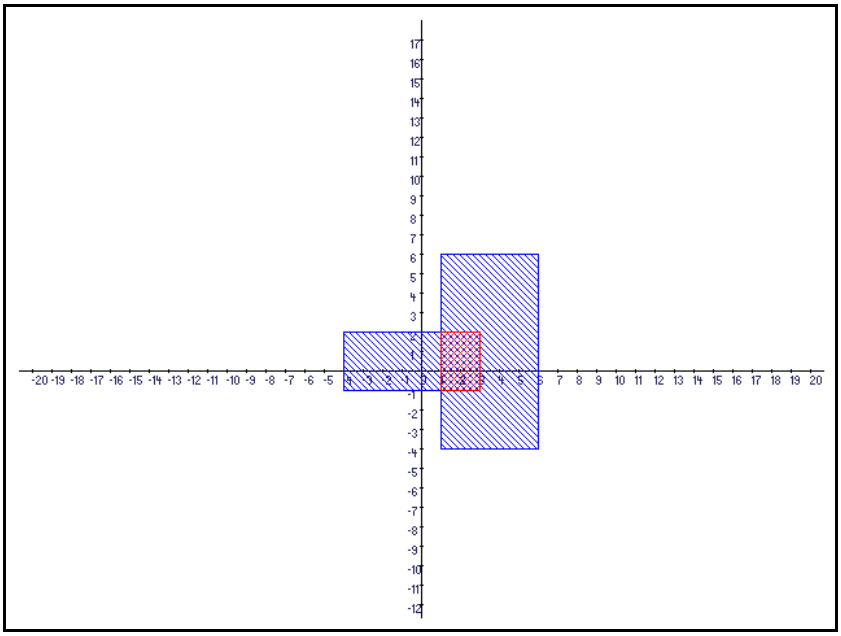

Figura 2.19: Intersección de intervalos en 2 Dimensiones.

Supóngase, nuevamente, los intervalos X= [-4,-1] e Y= [1,6] y las mismas operaciones. Se observará que en el caso de la intersección se obtiene un conjunto vacío como en la figura 2.20. Pero en caso de la unión hay básicamente dos formas de resolver el problema: a) la unión de los intervalos son ambos intervalos por separado (figura 2.21a) y b) la unión de los intervalos son ambos intervalos y el espacio que hay entre ellos (figura 2.21b).

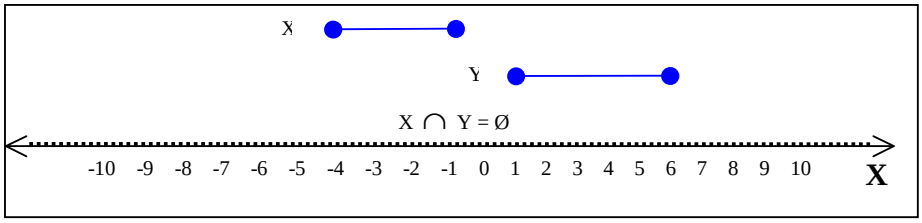

Figura 2.20: Intersección vacía de intervalos.

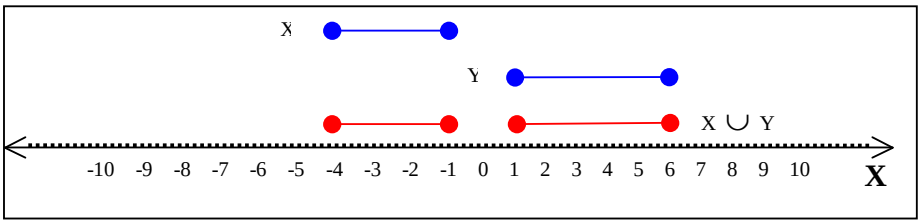

Figura 2.21a: Unión de intervalos alejados.

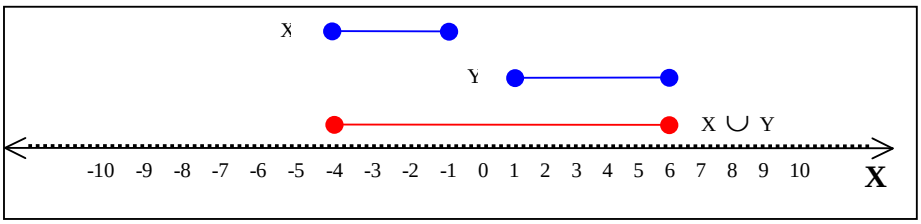

Figura 2.21b: Unión alternativa de intervalos alejados.

Es importante resaltar, que para efectos del proyecto se utilizará la segunda alternativa de unión de intervalos cuando se necesite, tal como se observó en la figura 2.21b, esto también considera el caso de dos dimensiones, tal como se muestra en la figura 2.21c.

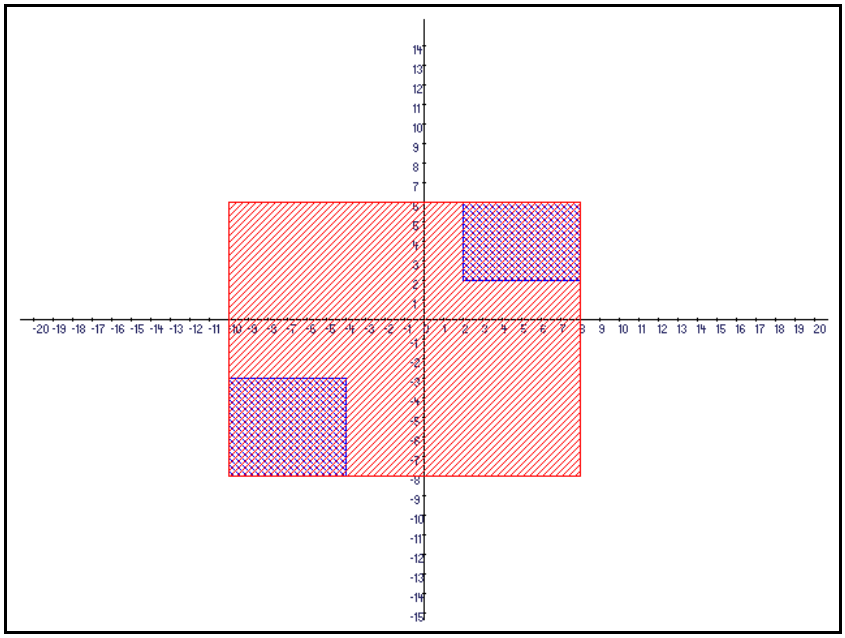

Figura 2.21c: Unión alternativa de intervalos alejados en 2 dimensiones.

Dentro de las propiedades que se mantienen de la aritmética real ordinaria a la aritmética intervalar se encuentran la conmutatividad y la asociatividad. Así por ejemplo si se define a X, Y, Z como intervalos, se puede decir que:

$$X + Y = Y + X$$

$$X * Y = Y * X$$

$$X + (Y + Z) = (X + Y) + Z$$

$$X * (Y * Z) = (X* Y) * Z$$

Si se define el intervalo A, se puede considerar la propiedad de identidad del cero en la adición y del uno en la multiplicación:

$$A+0 = 0+A = A$$

$$A*1 = 1*A = A$$

La última de las propiedades que se definirá será la de inclusión monótona. Es decir, si X, Y, W y Z son intervalos y $W \subset X$ e $Y \subset Z$, entonces:

$$W+Y \subset X+Z$$

$$W-Y \subset X-Z$$

$$W*Y \subset X*Z$$

$$\frac{W}{Y} \subset \frac{X}{Z} \text{ si } 0 \notin Y \text{ y } 0 \notin Z$$

2.6 Funciones por Intervalos

La aritmética por intervalos permite generalizar la función real $f(\chi_1, \chi_2, ..., \chi_n)$ a una función correspondiente por intervalos $F(X_1, X_2, ..., X_n)$, a través de la extensión de $f(\chi_1, \chi_2, ..., \chi_n)$ (Hansen, 1992):

$$F(X_1, X_2, ..., X_n) = \{ f(\chi_1, \chi_2, ..., \chi_n) \mid \forall \chi_i \in X_i, i=1,...,n \}$$

Entonces si $f(x)$ es una función real, con la variable real x, la consideramos como intervalo con X como argumento si se escribe como $f(X)$. La aparición de un argumento intervalo, da a entender que el valor que toma f(X) es también un intervalo. (Vivallos, 2007)

Para trabajar con intervalos se deben reemplazar las variables reales por las correspondientes variables por intervalos. De la misma forma, se debe cambiar la aritmética real por la aritmética por intervalos. Una vez hecho lo anterior, se está en condiciones de evaluar una función. Hay que tener presente que el cálculo de una función por intervalos es delicado, pues la aritmética de intervalos difiere de la aritmética real en sus fundamentos algebraicos. Como la propiedad distributiva: I x (J + K) = I x J + I x K que no se cumple, pero sí lo hace la propiedad de sub-distribuidad: I x (J + K) $\subset$ I x J + I x K. En otras palabras, el resultado de una función por intervalos depende del orden en que las operaciones son efectuadas.

Para aclarar esta situación, veamos el siguiente ejemplo: (Campos, 2004)

Sean $f_1(x_1, x_2, x_3) = x_1 \times (x_2 - x_3)$ y $f_2(x_1, x_2, x_3) = x_1 \times x_2 - x_1 \times x_3$; las funciones por intervalos correspondientes son $F_1(X_1, X_2, X_3) = X_1 \times (X_2 - X_3)$ y $F_2(X_1, X_2, X_3) = X_1 \times X_2 - X_1 \times X_3$. Si tenemos que $X_1 = [2,4]$, $X_2 = [1,4]$ y $X_3 = [3,6]$, la evaluación de F_1 es:

$$F_1(X_1, X_2, X_3) = [2,4] \text{ x } ([1,4] - [3,6]) = [2,4] \times ([-5,1]) = [-20,4]$$

Mientras que la evaluación de F2 es:

$$F_2(X_1, X_2, X_3) = [2,4] \text{ x } [1,4] - [2,4] \times [3,6] = [2,16] - [6,24] = [-22,10]$$

Como se puede observar los resultados son distintos para F_1 y F_2, sin embargo sí se cumple que $F_1 = [-20,4] \subset F_2 = [-22,10]$.

La razón de este problema, llamado *problema de dependencia* (Hansen, 1992), es que cuando una variable se repite en una función, es tratada como una variable distinta en cada ocurrencia. Para minimizar esto, se debe examinar una forma de evaluar la función que busque disminuir el problema de dependencia. Para el caso de este proyecto, se considera trabajar con el esquema más sencillo, llamado extensión natural, correspondiente a la transformación de las variables reales a variables intervalares explicada anteriormente. Para obtener mayores detalles, se puede revisar el libro de Hansen (Hansen, 1992).

2.7 Error de Redondeo

Todo lo mencionado hasta este momento compone las herramientas que se necesitan para evitar el error de redondeo, problema presente en las operaciones matemáticas que realizan los computadores y que se resuelve con una adecuada implementación de la aritmética de intervalos.

Cuando se quiere almacenar números decimales en un computador, mejor dicho cuando se quiere representar números decimales en un computador, se destinan ciertos bits para la parte entera y otros para la parte fraccionaria. Esto se conoce como precisión fija. Ahora bien, cuando estos bits no son suficientes, el número se aproxima o redondea, o simplemente se trunca o corta. Resultando así una representación del número, pero en ningún caso el número que se quiso almacenar en primer lugar. Esto se conoce como el error de redondeo o error de punto flotante.

Un ejemplo de lo ineficiente que resulta el uso de precisión fija lo presenta S. M. Rump (Rump, 1988) con la siguiente función:

$$f(x,y) = 333.75y^6 + x^2 (11x^2y^2 – y^6 - 121y^4-2) + 5.5y^8 + x/(2y)$$

La cual fue evaluada por Leclerc (Leclerc, 1992), con x=77617 e y=33096, utilizando el lenguaje de programación FORTRAN en un computador SPARCStation SLC. Para luego realizar una comparación con tres tipos de precisión: simple, doble y extendida (single, double y extended), que corresponden a 6, 14 y 35 dígitos decimales respectivamente. En este caso, se consideraron sólo operaciones aritméticas básicas, por lo que el cálculo de las potencias de x e y fueron evaluadas mediante multiplicaciones. Las cuales entregaron los siguientes resultados:

***Caso precisión simple*:** f = 6.33825×10^{29}

***Caso precisión doble*:** f = 1.1726039400532

***Caso precisión extendida*:** f = 1.172603940053117863185883490452018 38

Claramente se puede concluir que el resultado con la precisión simple es erróneo. Sin embargo por el criterio de resultados similares, se puede llegar a una equivocada y apresurada conclusión, asumiendo que el resultado obtenido con la precisión doble es correcto. Lo extraño de este caso, y por eso su uso masivo para demostrar la necesidad de los intervalos, es que los tres resultados están errados, partiendo por el primer digito, mas aún, partiendo del signo. Pero si se utiliza la aritmética intervalar con precisión variable (Leclerc, 1992), usando 40 dígitos de precisión, se obtiene que el resultado exacto está contenido en el siguiente intervalo:

$$F = [a,b]$$

$$a = -0.827396059946821368141165095479816292005$$
$$b = -0.827396059946821368141165095479816291986$$

Con esto se puede afirmar que el error de redondeo presentado por la precisión fija puede comprometer la confiabilidad de los datos, aun cuando haya un número de dígitos similares entre dos tipos de precisión.

Capítulo 3: Optimización Global

Como se mencionó al principio de este trabajo la aritmética intervalar permite resolver el problema de optimización global, la cual se puede definir como la técnica que pretende encontrar el óptimo global de cualquier función, asumiendo cotas iniciales de los valores de las variables, dichas cotas pueden asumir cualquier valor incluso infinito.

Diversas son las áreas donde se aplica o se puede aplicar la técnica de optimización: economía, ingeniería y producción industrial por mencionar algunas.

3.1 El Problema de Optimización Global

El problema de optimización global se puede definir como la búsqueda del punto óptimo de una función. De forma más específica, se puede definir como la búsqueda de un mínimo (o máximo) global en una función, continua o discreta, con dominio en un subconjunto de los reales. La optimización local solo se preocupa de encontrar el (los) valor(es) mínimo o máximo parcial de una función. Este valor puede ser el óptimo global, pero no se sabe con certeza.

Se puede definir el punto máximo o mínimo global de una función como el valor máximo o mínimo respectivamente, que se puede encontrar al evaluar la función en todos sus puntos. De la misma manera, se logra definir el máximo y mínimo local como los valores máximos o mínimos parciales respectivamente, de una función cualquiera. En otras palabras, los puntos de máximo y mínimo local corresponden a los puntos donde la pendiente de la función se hace 0, o a los extremos de ésta. En el caso de no existir un valor mayor o menor que un punto de máximo o mínimo local respectivamente, entonces ese punto pasaría a convertirse en un máximo o mínimo global.

En la figura 3.1 a continuación se pueden apreciar claramente los máximos y mínimos locales y globales de una función cúbica.

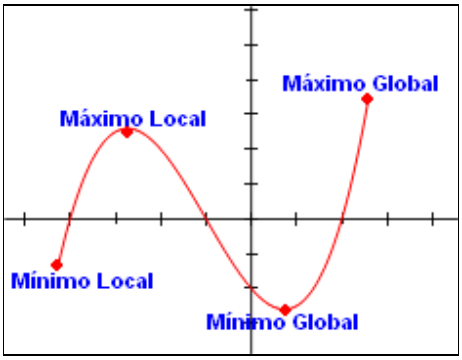

Figura 3.1: Función objetivo

Se puede dar el caso que la función objetivo, aquella que queremos analizar, contenga varios mínimos o máximos globales. En tal caso, si el objetivo es encontrar, por ejemplo, el mínimo global de la función, este sería el conjunto de mínimos globales hallados.

Muchas veces, cuando se habla del problema de optimización se dice que puede ser un problema de optimización sin restricciones, o un problema de optimización con restricciones. Dichas restricciones son de vital importancia, ya que definen a la región factible, es decir, aquella en la cual es posible que se encuentren las soluciones (Vivallos, 2007).

Por ejemplo, si se considera la figura anterior, pero se dice que ahora la función esta restringida a la región que se encuentra achurada en la figura 3.2, entonces el espacio de solución factible será menor y por lo tanto será mas acotado el conjunto en el cual se buscará una solución.

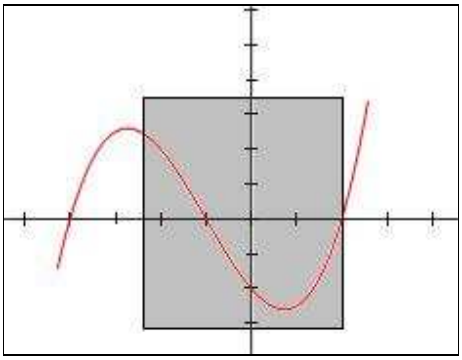

Figura 3.2: Función objetivo con restricciones

Es importante destacar que la región factible de un problema sin restricciones es, en teoría, todos los posibles valores que pueden llegar a tomar una función. En cambio la región factible de un problema de optimización con restricciones, corresponde sólo a aquellos valores que respetan las restricciones impuestas.

Formalmente el problema de optimización global se puede definir como:

$$f^* = \min_{x \in X} f(x)$$

Donde $f : \mathbb{R}^n \to \mathbb{R}^1$ es una función objetivo (que se quiere minimizar) continua y real, $X \subset \mathbb{R}^n$ siendo X el espacio donde la función esta definida, el dominio de la función; a este espacio se le denomina comúnmente región factible. Si se considera que minimizar $f(x)$ es equivalente a maximizar - $f(x)$, entonces la definición anterior sirve tanto para hallar el mínimo global como para hallar el máximo global.

El conjunto de todos los puntos, en los cuales la función objetivo posee un valor mínimo global se denomina: X^*, este conjunto contiene todos los puntos x^* tales que $f(x^*) = f^*$, a este conjunto se le conoce como el conjunto de minimizadores globales. Dado que en todo momento se habla de un conjunto, se debe entender que existe la posibilidad de que exista más de un mínimo o máximo global, ya que la función puede alcanzar su valor óptimo, ya sea máximo o mínimo, en más de un punto.

3.2 Optimización Global por Intervalos

Moore creó el primer algoritmo de optimización global por intervalos al utilizar la aritmética intervalar para calcular el rango de una función sobre un hiperrectángulo. Esto por la necesidad de contar con una cota inferior para la evaluación de una función sobre una región dada. (Vivallos, 2007)

Tiempo después, Skelboe (Skelboe, 1974) creó una mezcla entre la metodología por intervalos de determinación de rango de Moore, con el principio de "Branch and Bound". El cual consiste en 2 características:

- No se realiza en ningún caso una búsqueda uniforme en el espacio factible. Sino que, se realiza la búsqueda primero y con mayor profundidad en ciertas subregiones.

- Se requiere el cálculo de una cota inferior de la subregión.

Esta mezcla de metodologías, busca de primera forma particionar el espacio inicial de búsqueda, X, en subregiones más pequeñas S_i^x. La búsqueda del mínimo global, f^*, se realiza optando por aquellas subregiones S_i^x para las cuales la cota inferior del valor de la función, $F_L(S_i^x)$, es menor. Lo que se busca con lo anterior, es que cada caja tenga una mayor posibilidad de contener un minimizador global, x^*.

Un detalle de este algoritmo es que no contiene ninguna prueba para eliminar aquellas subregiones que definitivamente no contienen ningún mínimo (o máximo) global. Por lo tanto continúa desarrollándose mientras no se cumpla algún criterio de detención arbitrario. Como por ejemplo, podría continuar hasta que:

$$w(F(S_m^x)) < \varepsilon$$

Es decir, cuando el ancho del rango F sobre S_m^x sea menor que un valor ε.

Al basarse en esta condición, el ancho del rango F sobre S_m^x es una cota superior para el error absoluto cuando F^* es aproximada por cualquier punto $x \in S_m^x$.

3.3 Algoritmo de Optimización Global

Con lo presentado en las secciones anteriores se está en condiciones de describir un algoritmo para la optimización global utilizando intervalos. El algoritmo resuelve problemas de la forma:

$$\min_{x \in \beta} f(x)$$

Sujetos al conjunto de restricciones:

$$p_i \leq 0, i = 1,2,...,k$$

Permitiendo así definir el espacio o región factible, ósea, la región donde es válido buscar óptimos.

Ahora, se define B como cualquier hiperrectangulo inicial dado, o "caja", definido como el siguiente intervalo de n-dimensiones (vector de intervalos)

$$B = \{x : a_i \leq x_i \leq b_i \text{ para todo } i = 1,2,...,n\} = \begin{bmatrix} [a_1, b_1] \\ [a_2, b_2] \\ ... \\ [a_n, b_n] \end{bmatrix}$$

A continuación se busca una cota arbitrariamente ajustada, en la caja B, para el conjunto X^* de todos los minimizadores globales x^* que están en la región factible definida

por las restricciones $p_i \leq 0, i = 1,2,...,k$. Del mismo modo, se desea una cota arbitrariamente ajustada del valor del mínimo global, f^*, de la función objetivo dada, f. Esto es, se busca una cota para X^* y f^* de manera que:

$$f(x^*) = f^* \text{ y } f^* \leq f(x) \text{ para todo } x \in B \text{ que satisfaga } p_i \leq 0, i = 1,2,...,k$$

Se asume que las funciones $f, p_1, p_2, ..., p_k$ son programables, pero no necesariamente diferenciables. Una vez finalizado el algoritmo, se cuenta con una lista de pequeñas cajas cuya unión contiene el conjunto X^* de todos los minimizadores globales factibles x^*. El algoritmo termina cuando el ancho máximo de todas las cajas en la lista es menor que la tolerancia preescrita, ε_x. Este algoritmo entrega también las cotas superiores e inferiores para el valor mínimo $f_* = f(x_*)$.

La figura 3.3, capturada de la plataforma, muestra como se va particionando el espacio de búsqueda en cajas más pequeñas en una operación de optimización global por intervalos. Se observa claramente un área mas ennegrecida, que corresponde al resultado de la optimización.

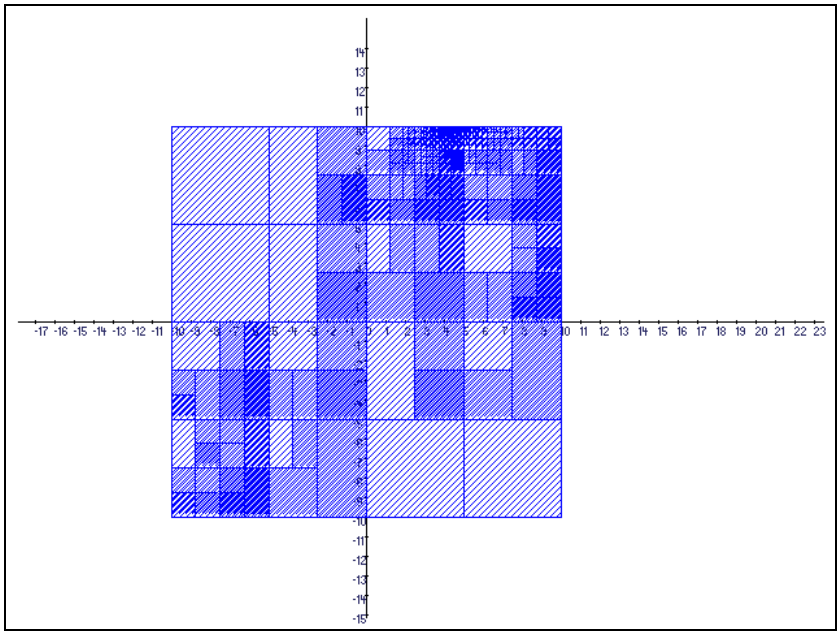

Figura 3.3: Optimización global por intervalos

Durante el procedimiento el algoritmo va rechazando las partes de la caja inicial B que no pueden contener un minimizador global, dejando una lista de sub-cajas (de B) cuya unión contiene el conjunto de todos los minimizadores globales de $f(x)$.

Entonces, si eliminamos todas las cajas que no contienen un minimizador global de la figura 3.3, el resultado de la optimización global por intervalos corresponde solo a la unión de las cajas que sí lo pueden contener. La figura 3.4 muestra la unión de esas cajas (con mucho zoom aplicado en esa zona).

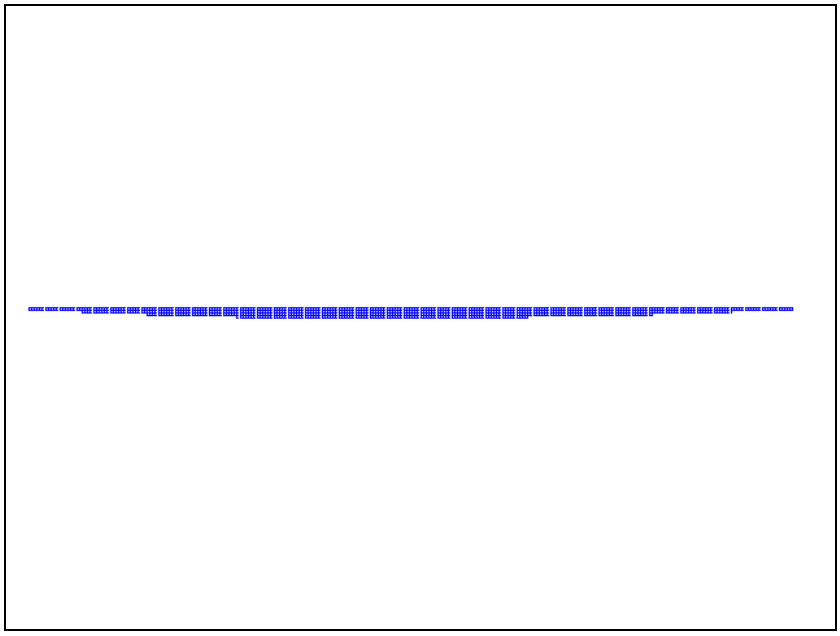

Figura 3.4: Resultado de optimización global por intervalos

En la plataforma se reservaron 4 botones en el menú para almacenar funciones de optimización global como la que se acaba de ilustrar en las imágenes 3.3 y 3.4.

Mayor información sobre la Optimización Global se puede encontrar en el Anexo F.

Capítulo 4: Graficación en Computadores

Los componentes de graficación, así como el resto de la plataforma de optimización global por intervalos, están desarrollados en C++. En el caso de los componentes de graficación también se utilizó la librería gráfica SDL ya que C++ no soporta de forma nativa la graficación por pantalla.

4.1 Historia de C/C++

C++ es un lenguaje de programación de propósito general basado en el lenguaje de programación C. Entre sus principales características destacan (Ceballos, 1993):

- Soporte a la programación Estructurada.
- Soporte a la programación orientada a Objetos.
- Economía en las expresiones.
- Abundancia en operadores y tipos de datos.
- Codificación en alto y bajo nivel simultáneamente.
- Facilidad de aprendizaje.

El lenguaje C nació en los laboratorios Bell de AT&T y está estrechamente asociado al sistema operativo UNIX, ya que su desarrollo se realizó en este sistema y debido a que tanto UNIX como el propio compilador C y la casi totalidad de los programas y herramientas de UNIX, fueron escritos en C (Ceballos, 1993).

Como se dijo anteriormente C++ evolucionó de C, el cual, a su vez evolucionó de dos lenguajes de programación anteriores, BCPL y B. En 1967, Martin Richards desarrolló BCPL como un lenguaje para escribir software para sistemas operativos y compiladores. Ken Thompson modeló muchas de las características de C en su lenguaje B, luego del

desarrollo de su contraparte BCPL y, en 1970, utilizó B para crear las primeras versiones de UNIX en los laboratorios Bell, sobre un computador DEC PDP-7. Tanto BCPL como B eran lenguajes “sin tipo” (cada dato ocupaba una “palabra” en memoria y, por ejemplo, el trabajo de procesar un elemento como un número entero o un numero real era responsabilidad del programador).

La evolución de B a C estuvo a cargo de Dennis Ritchie, también en los laboratorios Bell, quien en 1972 lo implementó en un computador DEC PDP-11.
A principios de la década de los ochenta Bjarne Stroustrup, informático danés, desarrolló una extensión de C en los laboratorios Bell: C++. Este lenguaje proporciona un conjunto de características que “pulen” al lenguaje C; pero lo más importante, proporciona la capacidad de una programación orientada a objetos. (Deitel y Paul, 1995). En la actualidad la mayoría de los sistemas operativos están escritos en C y/o C++, y C se encuentra disponible para la mayoría de los computadores y es independiente del hardware.

4.2 Historia de la graficación por computador

4.2.1 Hardware

Desde los albores de la computación y la informática se ha necesitado ver resultados visibles de las operaciones que realiza un computador. En los primeros días de la computación, se elaboraban gráficos rudimentarios en dispositivos de impresión como teletipos e impresoras de línea. El computador Whirlwind, desarrollado en el MIT en el año 1950, tenía un tubo de rayos catódicos dirigido por computador que se empleaba para presentar las salidas, tanto para el operador como para las cámaras que producían las versiones impresas.

El nacimiento de la graficación interactiva moderna se puede hallar en el importante trabajo doctoral de Ivan Sutherland acerca del sistema de dibujo Sketchpad (Sutherland, 1963). Él introdujo estructuras de datos para almacenar jerarquías de símbolos construidas

por medio de copias de componentes estándar. Sutherland desarrolló también técnicas de interacción que usaban el teclado y un lápiz de luz (un dispositivo apuntador manual que detecta la luz emitida por los objetos en la pantalla) para seleccionar opciones, apuntar y dibujar; además, formuló otras ideas y técnicas fundamentales que aún se usan en la actualidad. (Foley *et al*, 1994)

4.2.2 Software

A mediados de la década de 1970 se hizo evidente la necesidad de estándares para estos gráficos independientes de los dispositivos, lo que culminó con la especificación de un 3D Core Graphics System (Sistema Núcleo de gráficos Tridimensionales, conocido como Core), producidos por un comité de ACM SIGGRAPH en 1977 y refinado en 1979 (Foley *et al*, 1994).

La especificación Core cumplió con la función que se pretendía como especificación base. No solo tuvo muchas implantaciones, sino que se usó además en proyectos oficiales (gubernamentales) de estándares tanto en ANSI como en ISO. La primera especificación gráfica estandarizada oficialmente fue GKS, Graphical Kernel System (Sistema de Kernel Gráfico), una versión elaborada y depurada de Core que, a diferencia de éste, estaba limitado a dos dimensiones.

4.3 SDL

A lo largo del desarrollo de este proyecto se trabajó con una librería gráfica llamada SDL (Simple DirectMedia Layer), la cual es una librería multimedia multiplataforma, creada por Loki Games, diseñada para proveer acceso de bajo nivel de programación a componentes tales como hardware 3D por medio de OpenGL, video 2D, teclado, Mouse, sonido y Joysticks. Utilizada comúnmente para el desarrollo de juegos y aplicaciones 2D - 3D.

SDL soporta Linux, Windows, MacOS, MacOS X, FREEBSD, NetBSD, OpenBSD, BSD/OS, Solaris, IRIX y QNX. Está escrita en C, pero soporta C++ de forma nativa y tiene soporte adicional para muchos otros lenguajes, destacándose: D, Java, Pascal, Perl, PHP, Python y Ruby.

SDL se distribuye bajo GNU LGPL[1] versión 2. Esta licencia permite usar SDL de forma gratuita para cualquier uso que se le proponga, mientras esté enlazado con la librería dinámica.

SDL está compuesto por ocho subsistemas: audio, CDROM, gestor de eventos, Entrada/Salida de archivos, manejo de joystick, threading (ejecución de instrucciones simultáneas), temporizador y video. Antes de utilizar cualquiera de ellos, deben ser iniciados mediante la función *SDL_Init*, la cual por defecto inicializa los subsistemas gestor de eventos, Entrada/Salida de archivos y threading independientemente del subsistema que se inicializó. A continuación se ejemplifica como inicializar el subsistema de video.

```
SDL_Init ( SDL_INIT_VIDEO );
```

Si se desea inicializar más de un sistema, como por ejemplo video y audio, la función debe llamarse de la siguiente forma:

```
SDL_Init ( SDL_INIT_VIDEO | SDL_INIT_TIMER );
```

O sea, solo debe incluirse el carácter | para separar cada subsistema que se desea inicializar.

SDL_Init se complementa con la función *SDL_Quit*, la cual cuál cierra todos los subsistemas, incluyendo los por defecto. Esta función debe llamarse siempre antes de que una aplicación de SDL termine.

[1] Licencia Pública General Reducida de GNU (del inglés GNU Lesser General Public License).

Para mejor comprensión, se incluye en anexos (Anexo A) un código básico para inicializar y cerrar SDL y sus subsistemas.

La compilación y ejecución del código mencionado, se puede observar en la figura 4.1 a continuación.

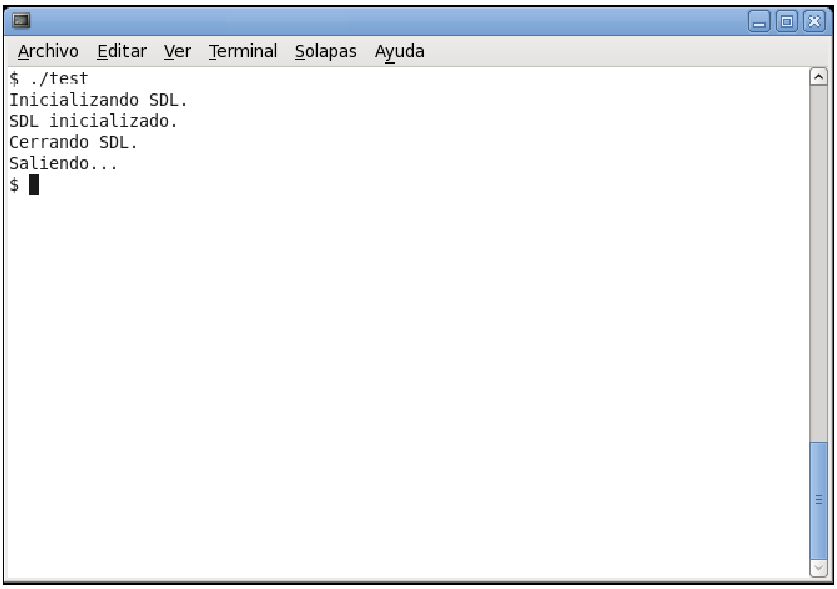

Figura 4.1: Iniciando SDL

Ahora bien, lo mas relevante para este proyecto es la utilización del subsistema de video (así como el gestor de eventos que se inicializa por defecto), como probablemente lo es para la mayoría de las aplicaciones que utilizan esta librería. Se puede encontrar en Anexos (Anexo B) un código básico que inicializa el subsistema de video y crea una ventana.

Si se compila y ejecuta el código mencionado, se obtiene lo que muestra la figura 4.2 a continuación.

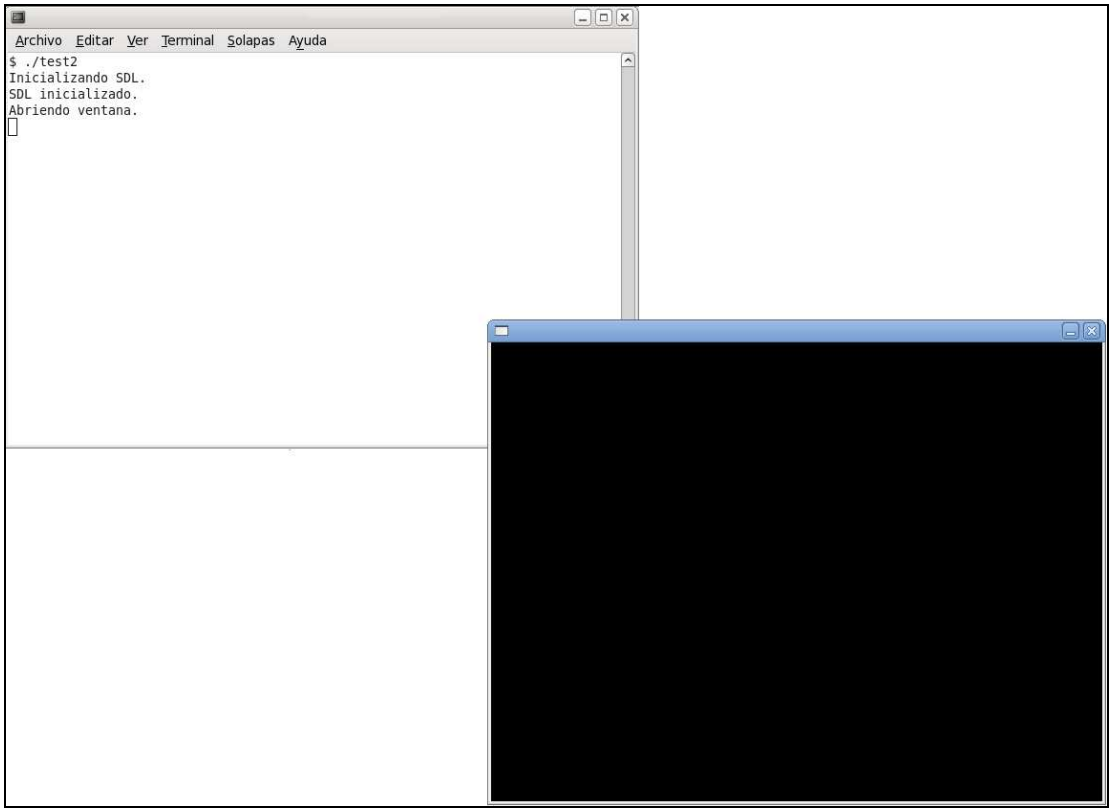

Figura 4.2: Ventana gráfica en SDL

Estas son algunas de las operaciones básicas que se pueden implementar en SDL. Si se revisa el código necesario para implementar el ejemplo visto en la figura 4.2 en los anexos (Anexo B), se puede observar la considerable cantidad de código para implementar algo tan simple, a diferencia de otros lenguajes de programación como Visual Basic, en donde la creación de ventanas, botones y menús es tan simple como arrastrar estos componentes desde la barra de herramientas. Pero es importante mencionar la ventaja que tiene la presente librería bajo C/C++, que corresponde a su gran flexibilidad que le permite al programador crear y modificar cualquier componente acorde a sus necesidades.

La razón por la cual se escogió esta librería es por el acceso de bajo nivel que se tiene a los distintos dispositivos o recursos del computador así como también su licencia que permite utilizarla sin costo alguno, al igual que la plataforma Linux bajo el cual se desarrolló el presente trabajo.

Al tratarse de un tipo de programación de bajo nivel, presenta ciertas ventajas como la flexibilidad con la que se pueden manejar los recursos del computador y los eventos del sistema. De la misma forma, agrega un cierto grado de dificultad el programar de esta forma, pues todo debe ser construido desde cero, y la codificación puede llegar a ser algo complicada.

Para trabajar con la presente librería, esta debe estar instalada en el sistema (junto con los componentes de desarrollo gráfico para el sistema X-org)[2] y al momento de compilar el código fuente, debe ser enlazada para que se incluyan sus rutinas. A continuación se incluye un ejemplo de cómo compilar un archivo fuente:

g++ codigo.c –o ejecutable.out -lSDL `sdl-config --cflags --libs`

En este caso se utiliza el compilador g++, parte de la distribución de GNU GCC[3], para la compilación de código orientado a objetos. Luego se incluye el código fuente *'codigo.c'* y su salida binaria *–o ejecutable.out*. Al final del comando se observa el enlace con la librería SDL *-lSDL `sdl-config --cflags --libs`*.

Una característica de esta librería es que se le pueden agregar otras librerías para el manejo de diversos elementos, tales como la inserción de imágenes, texto, sonido, etc. Si se pretende agregar la librería para la inserción de fuentes true type, el compilador debe llamarse de la siguiente forma:

g++ codigo.c -o ejecutable.out -lSDL `sdl-config --cflags --libs` ***-lSDL_ttf***

Donde *-lSDL_ttf* es la llamada a la librería que maneja la inserción de fuentes ttf (True Type Font). Se trabaja de la misma manera con las otras librerías multimedia disponibles para SDL.

[2] Implementación de código abierto del sistema X Window System que dota de una interfaz gráfica a sistemas Unix.
[3] Colección de compiladores creados por el proyecto GNU.

Con el propósito de visualizar de forma efectiva todo lo mencionado hasta este punto, se incluyen en las figuras 4.3, 4.4 y 4.5 algunas imágenes de la plataforma Vale decir que todas las imágenes corresponden a la misma aplicación (mismo código fuente), modificándose en tiempo de ejecución, gracias al gestor de eventos mencionado con anterioridad.

En la figura 4.3 se muestra la aplicación con cuatro intervalos dibujados sobre el eje cartesiano. Tanto el eje cartesiano como los intervalos, fueron dibujados mediante la inserción de pixeles en la ventana, a diferencia de los números sobre el eje, que corresponden a inserciones de texto TTF.

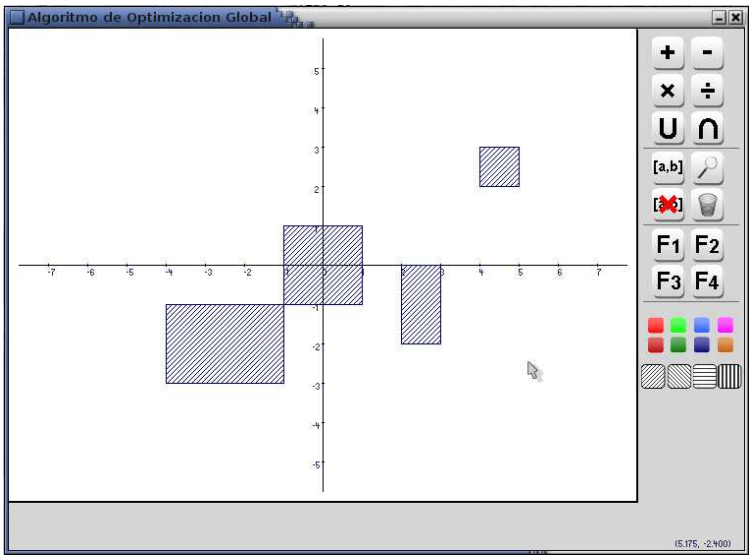

Figura 4.3: Captura 1

En la figura 4.4 se observa lo mismo, pero después de aplicar un zoom hacia dentro, ampliando el eje y los intervalos para obtener un mayor grado de detalle sobre una zona específica de la ventana.

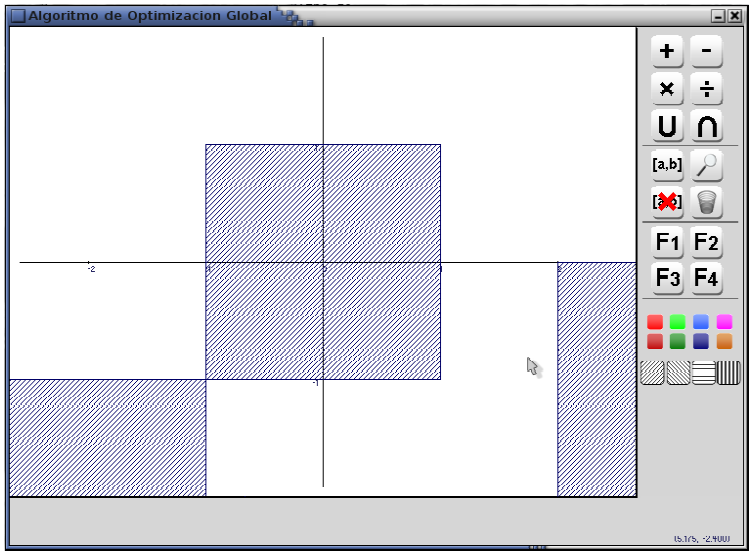

Figura 4.4: Captura 2

En la figura 4.5 se observa nuevamente la misma situación, pero ahora con un movimiento hacia la derecha del eje, luego de realizar un zoom hacia adentro.

El movimiento sobre el eje y el manejo del zoom de la aplicación le ceden al usuario un completo control sobre la aplicación en cuanto a visualización se refiere. Esto, mediante el uso del teclado o el Mouse.

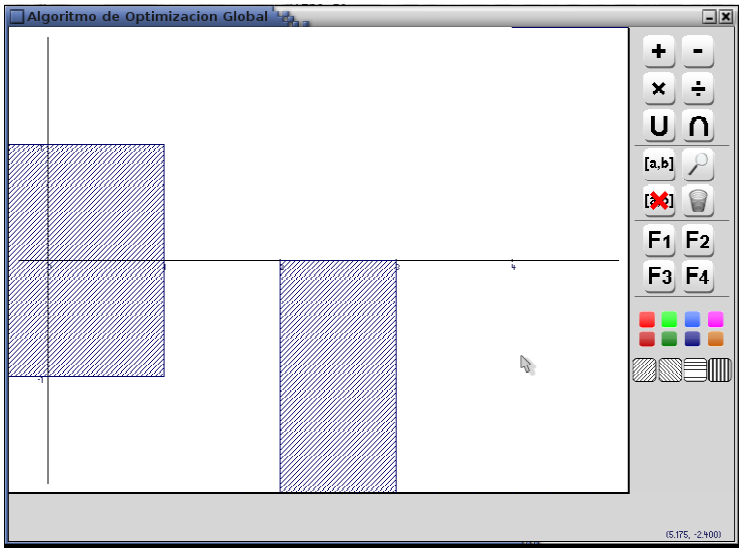

Figura 4.5: Captura 3

Enajenándose un poco al proyecto, se incluye en la figura 4.6 una captura de pantalla de una aplicación desarrollada con SDL, llamado Aleph One, que corresponde a un juego en primera persona y de código libre, que muestra otros alcances de la librería. En este caso, una aplicación 3D manejada mediante eventos del teclado o un Joystick, que incorpora los subsistemas de audio y video entre otros.

Figura 4.6: Aleph One

La ventaja que exhibe la mencionada librería ante otras como GLUT o Amiga (sus alternativas más similares) es su licencia, que le entrega total libertad al programador en cuanto a temas legales, así como su flexibilidad para el desarrollo, en cuanto a plataforma y diseño de componentes se refiere, tal como se ha venido comentando a lo largo del capítulo.

Capítulo 5: Ingeniería de Componentes

5.1 Componentes

Para empezar a hablar acerca de la ingeniería de componentes, es necesario primero tener clara la definición de componente, dentro del contexto de la ingeniería del software.

Se puede definir un componente como un bloque de construcción modular para el software de cómputo. En otras palabras corresponde a una unidad estructural, que en su conjunto forma un sistema, en este caso, un software. La especificación unificada de lenguaje de modelado de OMG[4] define un componente como "una parte modular, desplegable y reemplazable de un sistema que encapsula implementación y expone un conjunto de interfaces".

OMG corresponde al Grupo de Gestión de Objetos (de sus siglas en inglés) que publicó una arquitectura común de distribución de objetos (CORBA; por sus siglas en inglés). Un distribuidor de objetos (ORB; por sus siglas en inglés) proporciona una diversidad de servicios que permiten que los componentes reutilizables (objetos) se comuniquen con otros componentes, sin importar su ubicación dentro de un sistema (Pressman, 2005).

El significado de componente, en todo caso, varía según el uso que se le tenga consignado. De aquí se distinguen tres conceptos claves de lo que es un componente

[4] Object Management Group: Consorcio internacional sin fines de lucro dedicado al cuidado y establecimiento de diversos estándares de tecnologías orientadas a objetos.

5.1.1 Concepto Orientado a Objetos

Aquí un componente corresponde a un conjunto de clases que colaboran entre sí. Cada clase de un componente incluye al mismo tiempo, todos los atributos y métodos que necesita. Es necesario también definir todas las interfaces que permiten la comunicación entre las clases. Este concepto se enfoca entonces a la elaboración de clases de diseño que provienen tanto del problema como del dominio de la infraestructura.

5.1.2 Concepto Convencional

En este contexto, el concepto de componente se refiere a un elemento funcional de un programa que incorpora la lógica del procesamiento, las estructuras internas de los datos necesarios para implementar dicha lógica, y una interfaz que permita la invocación del componente y el paso de los datos.

5.1.3 Concepto Relacionado al Proceso

Este enfoque nace de la necesidad de construir sistemas que usen los componentes de software existentes. Estos componentes son creados con la noción de reutilización de antemano. A medida que se avanza en el diseño de un sistema queda a disposición del ingeniero de software un catálogo de componentes probados a nivel de diseño o de código, con su respectiva documentación y descripción.

5.2 Ingeniería de Software basada en Componentes

Mencionadas las distintas definiciones que caracterizan a un componente, se puede proceder a hablar acerca del desarrollo de software basado en ellos. Desarrollo que nace de la búsqueda de abaratar el costo de desarrollo de software. Entre otras características también destacan la búsqueda de aplicaciones más confiables y robustas, mejorando la calidad mediante la integración de componentes ya creados, probados y depurados con anterioridad.

Antes de continuar, es necesario llegar a un consenso sobre algunos términos que pueden ser confusos (Pressman, 2005):

Componente.

Parte importante, casi independiente y reemplazable de un sistema que satisface una función clara en el contexto de una arquitectura bien definida.

Componente del Software en ejecución.

Paquete dinámico de unión de uno o más programas gestionados como unidad y a los cuales se tiene acceso por medio de interfases documentadas que se pueden descubrir en la ejecución.

Componente de software.

Unidad de composición que solo tiene dependencias de contexto explícitas y especificadas en forma contractual.

Componente de negocio.

La implementación de software de un concepto o proceso de negocio “autónomo”.

5.2.1 Ventajas del desarrollo de Software basado en Componentes

Reutilización.

Si se diseña un componente con el fin de ser reutilizado a futuro, mejora los tiempos de desarrollo, calidad y costos de futuras aplicaciones o sistemas que utilicen aquellos componentes.

Reducción de costos y tiempo.

Ventaja principal, si se piensa en que para la manufactura de un sistema solo es necesario ensamblar los distintos componentes que son necesarios (en el mejor de los casos, cuando se tiene un gran catálogo de componentes) y realizar algunas modificaciones para ajustarlos a las necesidades del usuario.

Simplifica las pruebas.

Como cada componente es independiente, éste puede ser probado de forma individual antes de probar el sistema en conjunto.

Simplifica el mantenimiento del sistema.

Debido al bajo acoplamiento que presentan, cualquier cambio en ellos no afecta al sistema completo.

Mayor calidad.

Cada vez que se incorpora un componente a un sistema, este es probado y mejorado en cada ocasión, lo que aumenta su calidad en cada oportunidad.

5.3 Ingeniería del Dominio

La ingeniería del dominio tiene como finalidad la identificación, construcción, catalogación y diseminación de un conjunto de componentes de software que pueden ser aplicables para el software presente y futuro en un dominio de aplicación determinado.

5.3.1 Proceso de análisis del dominio

Los pasos para el proceso de análisis del dominio lo define Pressman (Pressman, 2005) como:

1.- Definir el dominio que se investigará.

2.- Categorizar los elementos extraídos del dominio.

3.- Recopilar una muestra representativa de las aplicaciones en el dominio.

4.- Analizar cada aplicación en la muestra y definir las clases de análisis.

5.- Desarrollar un modelo de análisis para las clases.

Destáquese que el análisis del dominio se puede aplicar independiente del paradigma de ingeniería que se utilice.

5.4 Desarrollo Basado en Componentes

El desarrollo basado en componentes es una actividad de la ingeniería de software basado en componentes que ocurre de forma paralela con la ingeniería del dominio. En este punto se refina un estilo arquitectónico apropiado para el modelo de análisis creado para la aplicación que se construirá. Entonces, una vez establecida la arquitectura del sistema, se debe proceder a añadirle los componentes disponibles en las bibliotecas de estos mismos si existen. En caso contrario se debe proceder a diseñar los componentes para luego integrarlos a la arquitectura del sistema y a la biblioteca de componentes.

5.5 Orientación a Objetos

De los conceptos que se han venido mencionando a lo largo del capítulo, es que se llega finalmente a optar por un estilo de programación que incluya los conceptos asociados a la ingeniería de componentes, como lo es la programación orientada a objetos, para el desarrollo del presente proyecto.

La programación orientada a objetos tiene su origen en la década de los ochenta y corresponde a unos de los paradigmas de la programación, así como también lo es la programación estructurada. Las propiedades que caracterizan a este estilo de programación son la herencia, polimorfismo, encapsulación, abstracción y ocultación de la información. Características que a su vez le permite al programador codificar dentro de un objeto todos los datos y métodos que trabajan sobre estos datos, que son propios del mismo objeto.

Existe una interfaz para cada objeto que se ocupa de comunicarlo con otros objetos, y a la vez se preocupa de proteger su contenido especificando como deben actuar con él.

Pressman (Pressman, 2005) define algunas tareas que ayudan en el análisis orientado a objetos:

1.- Deben comunicarse los requisitos básicos del usuario entre el cliente y el ingeniero de software.

2.- Deben identificarse las clases (es decir, se definen los atributos y métodos).

3.- Se define una jerarquía de clases.

4.- Deben representarse las relaciones de objeto a objeto (conexiones entre objetos).

5.- Debe modelarse el comportamiento del objeto.

6.- Las tareas 1 a 5 se vuelven a aplicar de manera iterativa hasta que el modelo esté completo.

Capítulo 6: Entorno de Desarrollo

6.1 Plataforma inicial

A continuación se presenta el estado de la plataforma de optimización global por intervalos antes de dar comienzo a este proyecto. Estado del mismo que se pretende extender mediante la inclusión de componentes gráficos.

6.1.1 Diagrama de Clases

A continuación se presenta en la figura 6.1, el diagrama de clases correspondiente a la plataforma de optimización global por intervalos implementada, que contempla una librería de aritmética por intervalos (Campos, 2004).

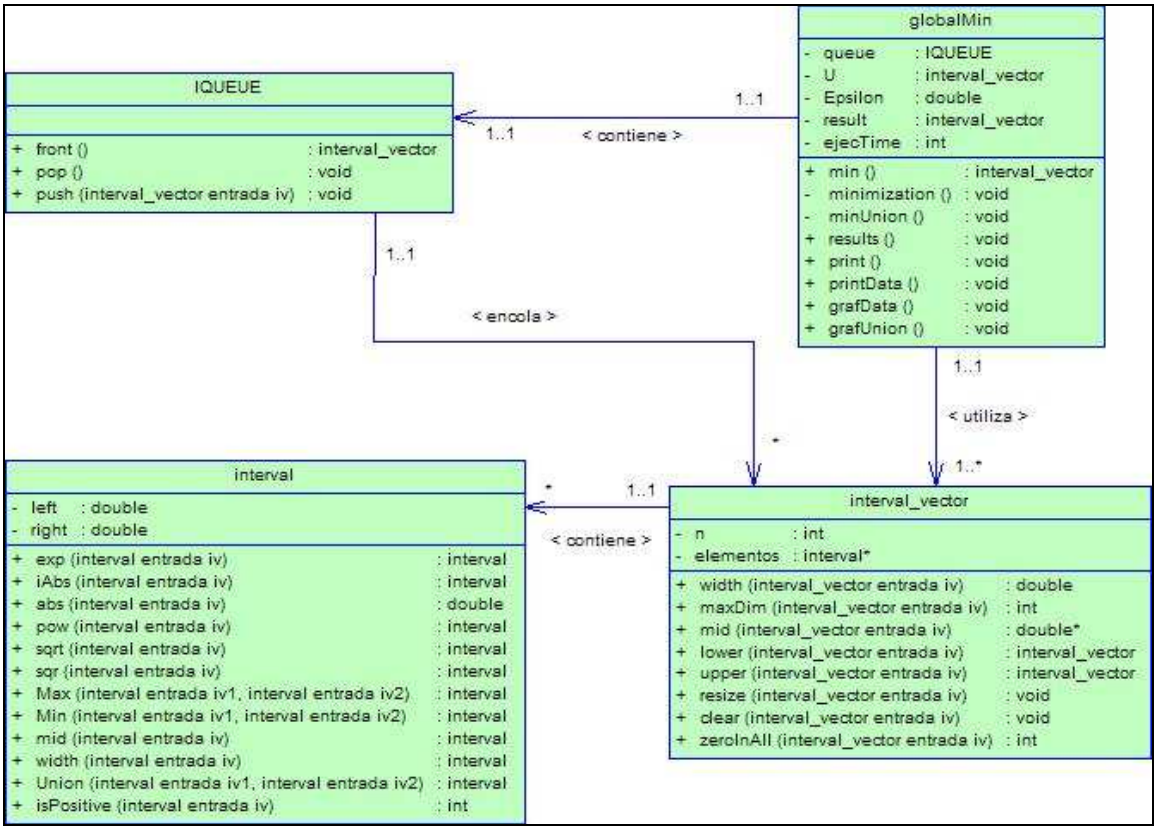

Figura 6.1: Diagrama de Clases

En la figura 6.1 se pueden observar las 4 clases que conforman la plataforma, sus atributos y los métodos que implementan.

Los métodos de la clase *interval* corresponden a funciones matemáticas en su mayoría, además de algunas funciones que permiten caracterizar a un intervalo, tales como *width() y mid()*.

Los métodos de la clase *interval_vector* permiten manipular el vector, como por ejemplo *clear()* o *resize()*, además de caracterizar al vector de intervalos, con funciones como *width()* y *mid()*, que en este caso entregan las propiedades de cada intervalo en el vector.

En el caso de la clase *globalMin*, posee el método que implementa la optimización global, así como métodos que permiten obtener el resultado de dicho proceso en distintos formatos.

6.1.2 Manejo del error de redondeo

Como se puede dar el caso de que un intervalo, resultado de alguna operación, tenga como extremo un número que el computador no sea capaz de representar, se toma la precaución de redondear hacia fuera los extremos. Tómese como ejemplo el intervalo [3.14159265, 3,14159266] que contiene en su interior a π. Si el computador solo maneja una precisión de 5 números decimales, solo podría almacenar el intervalo [3.14159, 3.14159] perdiendo el resultado original. Pero redondeando hacia fuera el resultado, buscando para el extremo inferior el número más grande que sea menor o igual al él y buscando para el extremo superior el número mas pequeño que sea mayor o igual a él, se obtiene el intervalo [3.14159, 3.1416] que contiene al intervalo resultado inicial [3.14159265, 3,14159266] (Campos, 2004).

En la plataforma se toma la precaución entonces, de redondear hacia fuera antes de asignarle valores a los extremos de un nuevo intervalo. Esto se hace mediante las funciones *SetRoundUp* y *SetRoundDown* que redondean un número hacia arriba o hacia abajo.

Considere ahora como un ejemplo práctico, un extracto del código de la clase *interval* que muestra como se implementó la suma de intervalos:

```
interval operator + (const interval &x, const interval &y)
{
  interval res;                    //Resultado
  SetRoundDown();                  //Redondeo hacia abajo
  res.left = x.left + y.left;      //Asignación extremo izquierdo
  SetRoundUp();                    //Redondeo hacia arriba
  res.right = x.right + y.right; //Asignación extremo derecho
```

```
    SetRoundDef();          //Redondeo al mas cercano
    return res;             //Retorno del resultado
}
```

Se puede observar claramente que antes de asignarle el extremo izquierdo al resultado *'res'* se setea la máquina para redondear hacia abajo, y luego, antes de asignarle el extremo derecho al resultado se configura para redondear hacia arriba. Finalmente se establece que la máquina redondee hacia el número más cercano (redondeo por defecto) y se retorna el resultado. Esta es la forma en que la plataforma controla el error de redondeo, y lo hace con todas las operaciones que generan un nuevo intervalo.

6.1.3 Como se controla el error de redondeo en un Computador

Para trata de minimizar o controlar el error de redondeo, se hace necesario modificar la forma en la que opera el redondeo de punto flotante de un computador. La IEEE[5] ha definido un estándar para la aritmética de punto flotante binaria; Dicho estándar es utilizado hoy en día por la mayoría de los procesadores de computadores personales y estaciones de trabajo, contribuyendo así a la normalización de la caótica situación previa (Jaulin *et al*, 2001).

La IEEE 754 además de especificar la representación binaria de los números de punto flotante, especifica cuatro modos de redondeo. El modo de redondeo, que se considera por defecto, es hacia el número más cercano. Adicionalmente se define el redondeo hacia 0, el redondeo hacia $+\infty$ y el redondeo hacia $-\infty$. Estos dos ultimos modos son particularmente utiles para la implementacion de la aritmetica de intervalos, ya que permite redondear hacia el numero de maquina mayor mas pequeño(redondeo hacia $+\infty$) y el numero de maquina mas grande (redondeo hacia $-\infty$) (Campos, 2004).

[5] **IEEE** corresponde a las siglas en ingles de ***The Institute of Electrical and Electronics Engineers***, o sea el Instituto de Ingenieros Eléctricos y Electrónicos en español, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas.

En la figura 6.2 muestra los cuatro modos de redondeo antes mencionados. Los segmentos verticales gruesos corresponden a los números que pueden ser representados en el sistema de punto flotante que se esta considerando. *X* es un número real que no puede ser representado de forma exacta por el equipo en cuestión y los cuatro modos de redondeo llevan a dos posibles resultados (Campos, 2004).

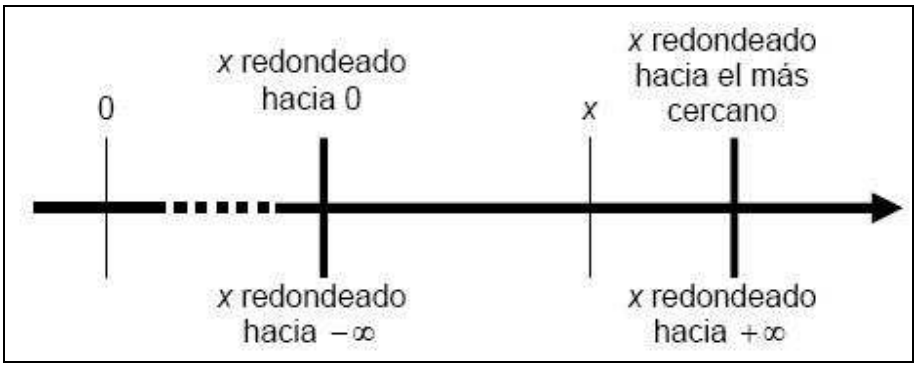

Figura 6.2: Los cuatro modos de redondeo especificados por la IEEE 754.

La implementación de la aritmética de intervalos, finalmente consiste en manejar un tipo de dato abstracto que permita mantener los valores de los extremos de los intervalos y definir las operaciones tanto para un extremo como para otro, utilizando para esto el redondeo hacia $-\infty$ en el caso del extremo izquierdo y el redondeo hacia $+\infty$ en el caso del extremo derecho. (Campos, 2004).

6.1.4 Intervalos en C++

En la clase *interval* de la plataforma, se sobrecargaron los operadores +, -, / y * entre otros y se definieron operadores como *pow* (potencia) y *sqrt* (raiz) para cualquier cálculo que los necesite. Para todas estas operaciones se manipuló el redondeo de punto flotante, utilizando la función *_contro87()* de Microsoft ® Visual C++ 6.0, que permite definir el modo de redondeo para operaciones de punto flotante, de acuerdo al estándar IEEE 754 (Campos, 2004).

En el código de la plataforma de pueden encontrar las definiciones de constantes *SetRoundUp* y *SetRoundDown*, que el compilador de C++ reemplaza por la llamada a la función *_control87()* correspondiente.

También están definidas las constantes *Inf (Infinito)* y *Nan (Not a Number)* que representan un número infinito y un valor no numérico correspondientemente. Estas constantes están definidas en la librería *limits* de Microsoft ® Visual C++ 6.0. Estos valores se utilizaron para representar algunos resultados de divisiones por 0.

6.1.5 Modificaciones y Correcciones a la Librería de Aritmética Intervalar

Entre las operaciones que mejor se aprecian, al realizarlas con intervalos, están la Unión y la Intersección. Estas operaciones no se encontraban implementadas en la librería con la cual se comenzó este proyecto, sin embargo, ya que el objetivo práctico principal de éste es mostrar las distintas instancias y operaciones que se pueden realizar con intervalos de una y dos dimensiones y siendo, la unión e intersección, tan importantes se procedió a su implementación.

La primera que se implementó, dada su facilidad de comprensión tanto teórica como práctica, fue la Unión. Como se mostró en el capítulo 2, en la unión de dos intervalos se considera como el extremo izquierdo del resultado a el menor valor de los extremos izquierdos de ambos intervalos, así de la misma manera se obtiene el extremo derecho, el cual resulta ser el mayor valor que poseen los extremos derechos de los intervalos que se están uniendo.

En cuanto a la Intersección, esta operación es más compleja pues depende del orden en que se seleccionen los intervalos, ya que se debe tener muy claro cual es el *sector* que ambos intervalos tienen en común, en caso de no poseer elementos en común, la librería es capaz de retornar dicha condición. Si se quiere considerar un aspecto más gráfico o visual

de la intersección entre intervalos, consiste en determinar cuál intervalo posee un extremo izquierdo mayor pues será éste el extremo izquierdo del intervalo de solución, así también se debe buscar el intervalo que posea el extremo derecho menor pues ese será el extremo derecho del intervalo solución.

La multiplicación presentaba un pequeño error en una de sus implementaciones, tal como se detalla en el Anexo D. Si se define la multiplicación tal como en el punto 2.4.3 entonces, dicho error se produce cuando el multiplicando contiene al cero y además el multiplicador es menor o igual a cero; en dicho caso el resultado reflejaba todo lo que debía contener. En la figura 6.3 se muestra en rojo el resultado de la multiplicación de los intervalos $\begin{bmatrix}[2,4]\\[-1,3]\end{bmatrix}$ y $\begin{bmatrix}[-1,2]\\[-2,3]\end{bmatrix}$ antes de corregir el error.

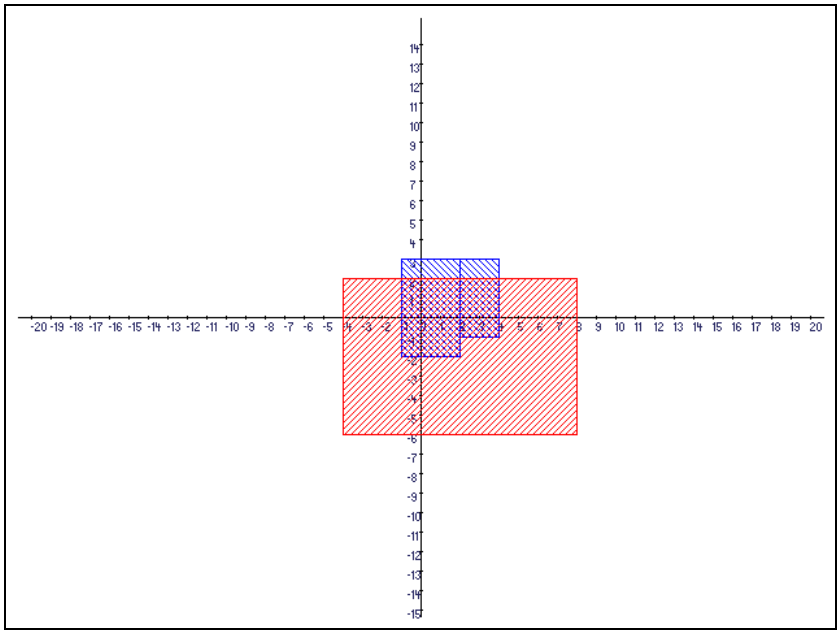

Figura 6.3: Multiplicación errada

Una vez corregido el error, el resultado de la misma multiplicación estaba bien representado por el intervalo resultado, rojo en la imagen 6.4 a continuación.

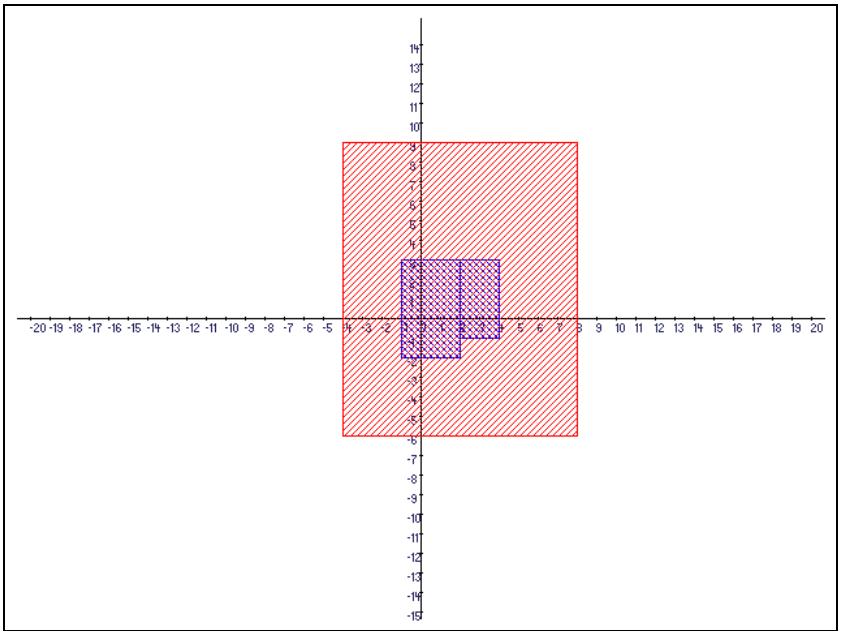

Figura 6.4: Multiplicación corregida

Otra modificación que se realizó a la librería fue la implementación de la división por cero, que se refiere cuando en el intervalo divisor se encuentra contenido el cero. Para la implementación se utilizó el trabajo realizado por Timothy J. Hickey y Maarten H. van Emden, llamado: *Interval Arithmetic from Principles to Implementation* (Hickey y van Emden, 2001) en dicho trabajo se exponen, entre otros, los diferentes casos o situaciones que pueden surgir de la división entre dos intervalos. Esto a su vez esta basado en el trabajo de Dietmar Ratz (Ratz, 1996) el cual explica que dados los intervalos [a,b] y [c,d] entonces [a,b] / [c,d] esta dado por:

$$[a,b] / [c,d] = \begin{cases} [a,b] * [1/d, 1/c] & \text{Si } 0 \notin [c, d] \\ [-\infty, \infty] & \text{Si } 0 \in [a, b] \wedge 0 \in [c, d] \\ [b/c, \infty] & \text{Si } b < 0 \wedge c < d = 0 \\ [-\infty, b/d] \cup [b/c, \infty] & \text{Si } b < 0 \wedge c < 0 < d \quad (1) \\ [-\infty, b/d] & \text{Si } b < 0 \wedge 0 = c < d \\ [-\infty, a/c] & \text{Si } 0 < a \wedge c < d = 0 \\ [-\infty, a/c] \cup [a/d, \infty] & \text{Si } 0 < a \wedge c < 0 < d \quad (2) \\ [a/d, \infty] & \text{Si } 0 < a \wedge 0 = c < d \\ \emptyset & \text{Si } 0 \notin [a, b] \wedge c = d = 0 \end{cases}$$

Estos casos se implementaron, con la salvedad que en aquellos en los que el resultado esta compuesto por dos intervalos, casos (1) y (2), estos se considerarán como dos intervalos independientes luego de ser mostrados en pantalla.

Además se modificó el código para que permitiese realizar operaciones entre intervalos de una dimensión con intervalos de dos dimensiones, para lograr esto se consideró como [0,0] la dimensión restante del intervalo unidimensional. Para ejemplificar estos casos, considérese el siguiente ejemplo. Sean los intervalos X= $\begin{bmatrix}[-6,2]\\[-1,6]\end{bmatrix}$ e Y= $[1,8]$, cuya suma se interpreta como $\begin{bmatrix}[-6,2]+[1,8]\\[-1,6]+[0,0]\end{bmatrix}$ y su resultado es el intervalo $\begin{bmatrix}[-5,10]\\[-1,6]\end{bmatrix}$. Lo anterior se puede apreciar en la figura 6.5 a continuación.

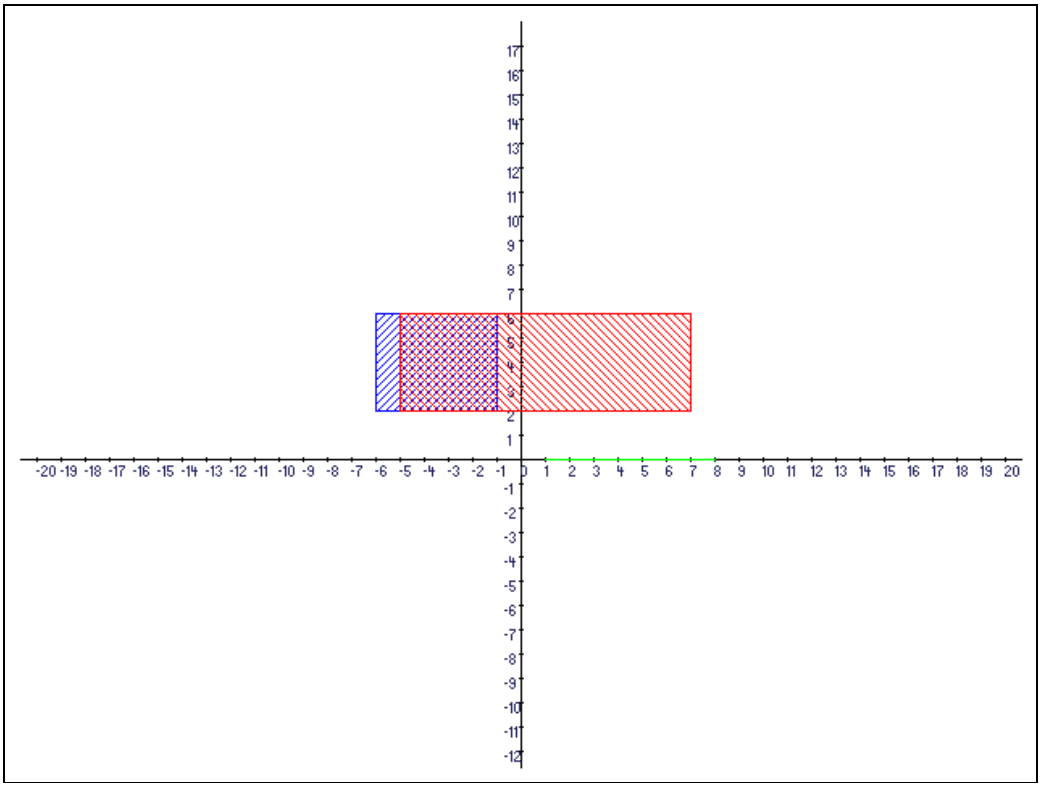

Figura 6.5: Suma de intervalos de distinta dimensión.

6.1.6 Funcionamiento de la Plataforma Inicial

El funcionamiento de la plataforma se basa en la colaboración de las clases implementadas. Las clases *interval* e *interval_vector* en conjunto forman la librería de aritmética por intervalos. La clase *globalMin*, junto a *IQUEUE* proporcionan la funcionalidad del proceso de optimización global, utilizando para realizar los cálculos la librería antes mencionada. Para comprender de mejor forma el comportamiento general de la plataforma, se presenta el siguiente diagrama de bloques.

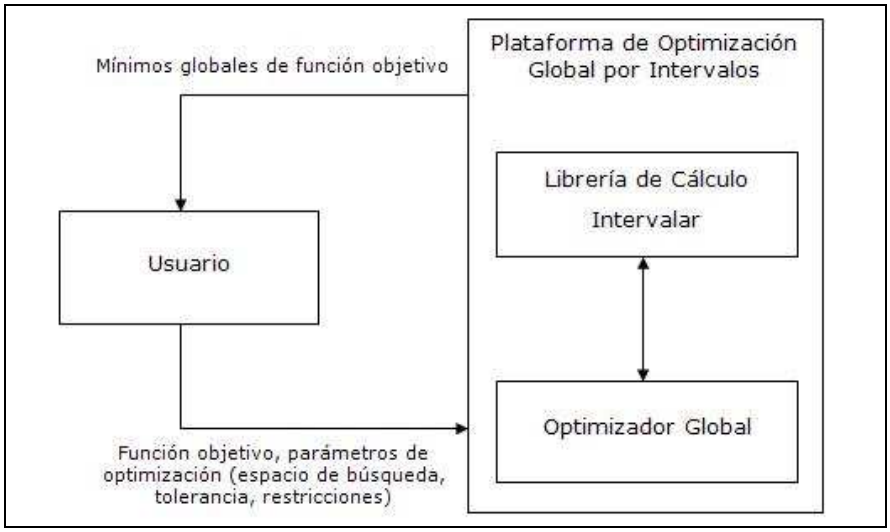

Figura 6.6: Diagrama de Bloque de funcionamiento de la Plataforma

La figura 6.6 muestra como interactuaban el usuario con la plataforma en un principio, antes de agregarle el entorno gráfico. Como se puede observar el usuario entrega la función objetivo que se desea optimizar, así como los parámetros de optimización, y la plataforma retorna los mínimos globales de la función (en realidad, cotas superiores e inferiores de ellos).

Para mayor información se incluye en el *Anexo C* un instructivo de cómo usar esta plataforma.

6.2 Metodología de Desarrollo

Para el desarrollo de este trabajo se utilizó un ciclo de vida basado en el llamado prototipo evolutivo, ya que entre sus fortalezas se encuentra su enfoque hacia las interfaces gráficas, además como se sabe este tipo de desarrollo esta basado en entregas o incrementos que se entregan al usuario y la retroalimentación que en esta instancia se genera, para así desarrollar mejoras de acuerdo con esa retroalimentación.

Otra de sus fortalezas es el concepto de progreso tangible, ya que en muy corto plazo se pueden ver avances o versiones de proyecto, generando así una sensación de avance.

Sin embargo sus debilidades van por el lado del desperdicio de recursos, tanto de tiempo de desarrollo de un prototipo que tal vez no se vea en el resultado final, así como de gasto en planificación de versiones intermedias.

Capítulo 7: Librería Gráfica de Aritmética Intervalar

7.1 Nueva plataforma.

Aquí se presentan los componentes que se adicionaron a la plataforma inicial y el resultado de su combinación.

7.1.1 Diagrama de Clases

A continuación se presenta en la figura 7.1, el diagrama de clases correspondiente a los componentes de graficación. Vale decir que estas clases interactuarán con las clases ilustradas en la figura 6.1.

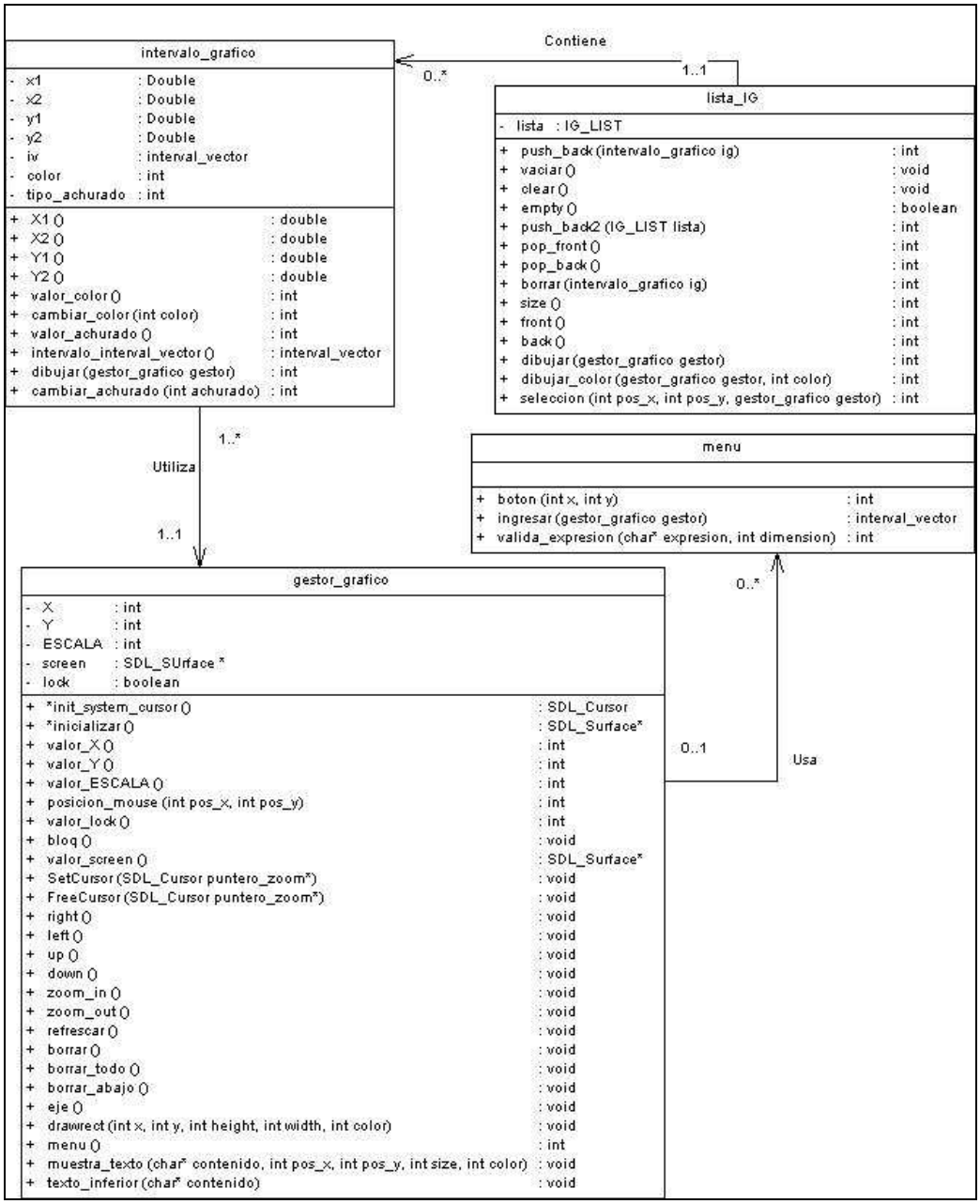

Figura 7.1: Diagrama de Clases de los componentes de graficación

En la figura 7.1 se pueden observar las 4 clases que conforman la plataforma, sus atributos y los métodos que implementan.

La clase *intervalo_grafico* es la representación de los intervalos en su forma visual, con un color y un estilo de achurado propio para su fácil diferenciación. La clase *lista_IG* es un contenedor de intervalos gráficos, que le permite a la aplicación mantenerlos en memoria. La clase *gestor_grafico* es la que tiene la tarea de hacer visible los intervalos y otros componentes visuales. Esto significa llevar el control de la ventana y todos los elementos que ahí se muestran. Por último la clase *menu* maneja el menú lateral y reconoce los botones que se presionan.

7.1.2 Funcionamiento de la Plataforma

Luego de la incorporación de los componentes de graficación, la plataforma se basa en la colaboración de todas las clases incluidas en las figuras 6.1 y 6.4. Y el diagrama de bloques que muestra la figura 6.2 evoluciona al diagrama que muestra la figura 7.2 a continuación.

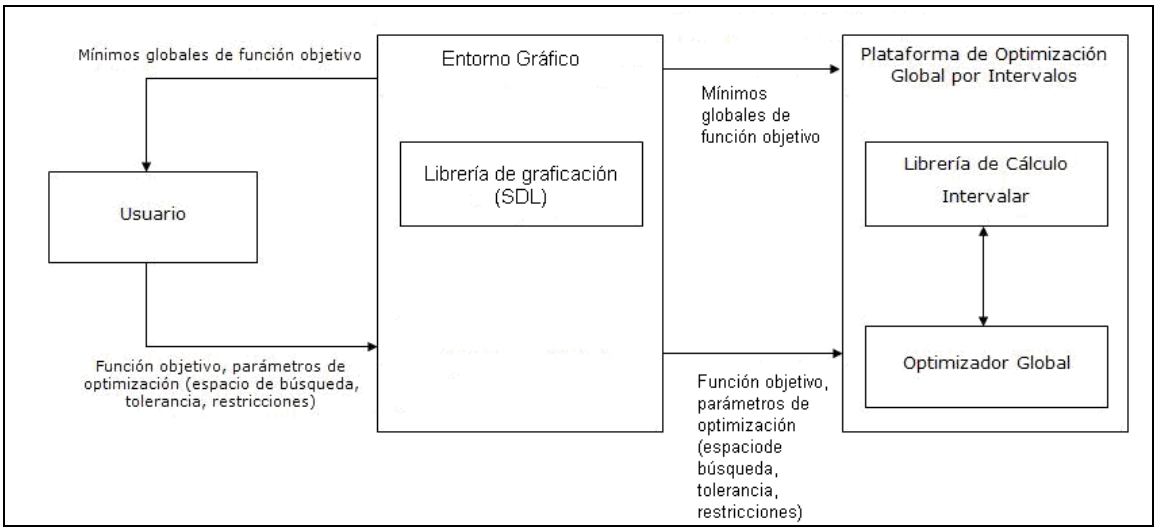

Figura 7.2: Diagrama de Bloque de funcionamiento de la Plataforma

En la figura 7.2 se puede apreciar que ahora el usuario no interactúa directamente con la plataforma de optimización global por intervalos, sino que lo hace con el entorno gráfico que se desarrolló, haciendo mas transparente el funcionamiento de la plataforma.

Los métodos y clases más importantes de este trabajo se pueden ver en el Anexo E.

7.2 Notas de Desarrollo.

7.2.1 Iniciando SDL

Para trabajar con la librería gráfica SDL, al momento de inicializar el subsistema de video se debe verificar que la máquina que correrá la aplicación soporta las configuraciones de video definidas en el código. Si no hay problemas, entonces se muestra una ventana de fondo negro por pantalla. De aquí en adelante es tarea de los programadores el representar lo que necesiten.

En el caso de la aplicación, una vez inicializado el subsistema de video, se limpia la pantalla (dejarla en blanco), se dibuja el eje cartesiano mediante la inserción de puntos por pantalla (encapsulados en una función que dibuja una recta en pantalla) y texto numérico y se carga el menú de selección a la derecha de la ventana, la cual consta básicamente de una imagen en formato *bmp*. Es importante resaltar que la aplicación maneja 3 áreas dentro de la ventana: área de trabajo, área de menú y área de mensajes. Estas áreas y sus dimensiones en píxeles se pueden apreciar mejor en la figura 7.3 a continuación.

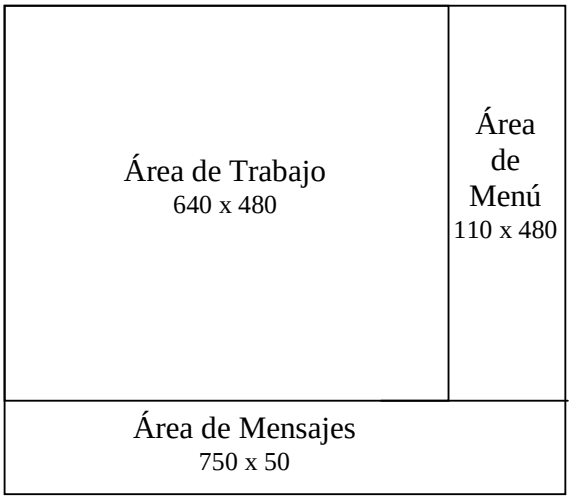

Figura 7.3: Distribución de la ventana

7.2.2 Evolución de la plataforma

A través del transcurso del proyecto se han diseñado distintas interfaces gráficas, donde destacan las que muestran las figuras 7.4, 7.5 y 7.6 a continuación.

El primer prototipo era en fondo negro y tenía graficado el eje cartesiano, en el cual se podían dibujar solo puntos. El prototipo permitía moverse por el eje y ajustar el nivel de zoom.

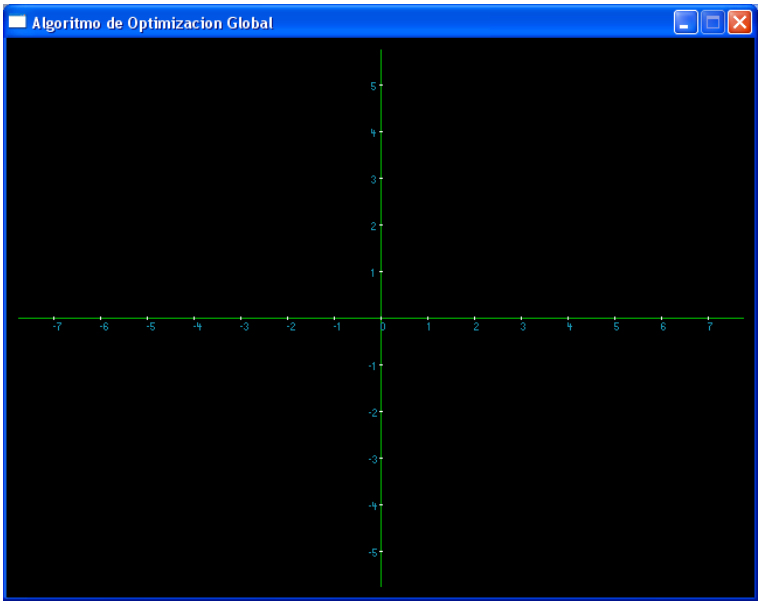

Figura 7.4: Primer prototipo

Para el segundo prototipo, por un tema de contraste se optó por un fondo blanco, y se le incluyó un pequeño menú donde se podían elegir las distintas operaciones que el usuario podía realizar. Aquí ya era posible dibujar los intervalos que estaban en el código fuente de la aplicación. También tenía, entre otras funcionalidades, salida de mensajes por pantalla en vez de consola.

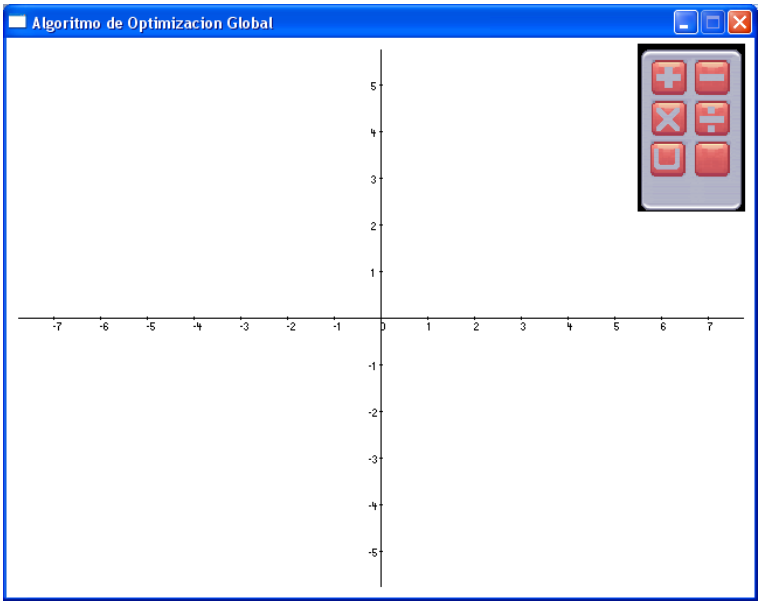

Figura 7.5: Segundo Prototipo

El estado de la aplicación en este momento es que cuenta con las áreas de trabajo mencionadas anteriormente, un menú más completo, una función que informa al usuario de la posición del puntero dentro del eje y por último se logró una mayor interacción con el usuario, pues le permite a este ingresar y eliminar información en tiempo de ejecución.

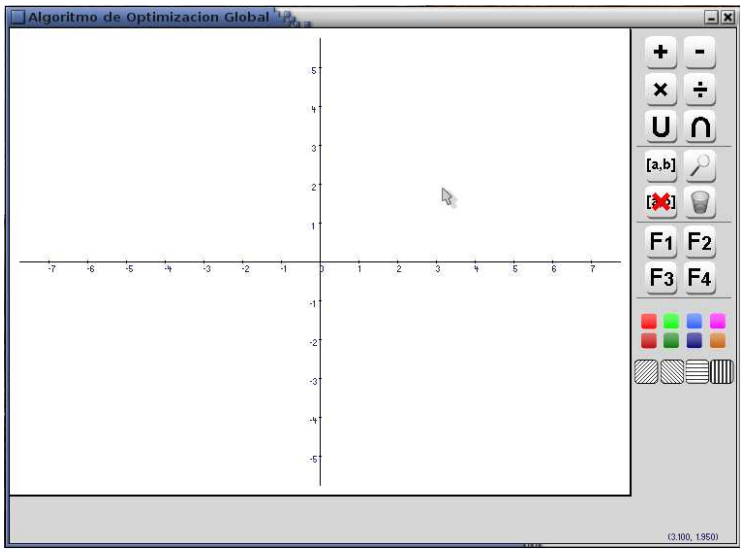

Figura 7.6: Estado actual de la plataforma

7.2.3 Manejo de eventos

SDL maneja la captura de eventos de forma nativa, por lo cual no es necesario instalar librerías de terceros ni inicializar difíciles rutinas. Lo que hace SDL es capturar todos los eventos de mouse y teclado y los guarda en una cola, la cual se recorre para que el programa "reaccione" de determinada forma ante un evento conocido, como cuando se presiona la tecla *escape* y la aplicación sabe que debe cerrarse. La forma de implementar lo anterior se soluciona con un ciclo infinito *While(1)* y dentro de él hacer que SDL capture eventos para luego ver si alguno de ellos es importante para la aplicación mediante un *switch.* Es aquí donde se realizan las distintas operaciones de la aplicación, tales como la inserción de un intervalo gráfico, suma de intervalos, intersección de intervalos, zoom sobre el eje cartesiano, desplazamiento del eje, etc.

7.2.4 Graficación

Ahora bien, para dibujar un intervalo por pantalla se debe, al igual que para dibujar cualquier punto en el plano, hacer una conversión de los puntos de pantalla o píxeles a puntos cartesianos, pues el origen de la ventana (punto 0,0) es el extremo superior izquierdo. Considerando que el ancho de la ventana es 640 y el alto 480 y que píxel_x y píxel_y corresponden a las coordenadas x e y de la ventana, la forma de hacer la conversión es la siguiente:

$$\text{Punto cartesiano= píxel\_x} - \frac{640}{2}\text{, -(píxel\_y} - \frac{480}{2}\text{)}$$

Para el caso en que el usuario se ha desplazado por el eje, es necesario saber a que distancia se encuentra del origen. Esta tarea cumplen los atributos X e Y de la clase *gestor_grafico*, donde X indica cuanto se ha desplazado el usuario por el eje X y de forma equivalente para Y. Con esto podemos ampliar la fórmula de conversión a:

$$\text{Punto cartesiano= píxel\_x} - \frac{640}{2}\text{ - X, -(píxel\_y} - \frac{480}{2}\text{ - Y)}$$

Por último, si el usuario ha optado por modificar el nivel de zoom es necesario conocer el nuevo valor (almacenado en ESCALA), y la fórmula de conversión que considera el zoom queda:

$$\text{Punto cartesiano} = \frac{\text{píxel\_x} - \frac{640}{2} - X}{ESCALA}, \frac{\text{píxel\_y} - \frac{480}{2} + Y}{-ESCALA}$$

Para optimizar la graficación por pantalla de un intervalo en tiempo de ejecución, se limita a dibujar a este dentro de los límites de la ventana. Es decir, si se necesita graficar el intervalo [0,16] puede ser que en ese instante el eje de coordenadas muestre solo hasta el punto 8, entonces solo se intenta graficar hasta ese punto y no se gasta mas tiempo de procesamiento intentando graficar aquellos puntos que ni siquiera serían visibles, pues se escapan al límite del eje en ese momento.

Cada vez que se realice un movimiento sobre el eje o se aumente o disminuya el zoom, se limpia la pantalla y se vuelven a dibujar todos los elementos (intervalos) que estaban presentes con los nuevos parámetros. Se puede dar el caso entonces de que se estén manejando muchos intervalos simultáneamente, del orden de los cientos, y el tiempo que toma el computador para dibujarlos a todos, tome un tiempo ya perceptible para el usuario. Si se aumentan aún más los intervalos en memoria, entonces el tiempo para dibujarlos se puede volver un inconveniente. Para eso se implementó una forma de ‘bloquear’ la pantalla. En otras palabras, cuando se activa el bloqueo de pantalla (con la tecla “Bloq Despl” del teclado) no se dibujan los intervalos en memoria cada vez que el usuario se mueve por el eje o haga un zoom, sino que se dibujan cuando se desbloquea la pantalla (nuevamente con la tecla “Bloq Despl”). Esto le permite al usuario moverse y hacer zoom con rapidez, en situaciones en que por manejar muchos intervalos simultáneamente hace de estas operaciones un proceso algo lento.

7.2.5 Ingreso de Intervalos

Para que el usuario pueda ingresar un intervalo (en tiempo de ejecución) se implementó una función que permite ingresar por teclado una expresión que lo represente. Esta expresión se valida mediante el uso de expresiones regulares (cabecera estándar para C++ regex.h) almacenadas en el código fuente. Por ejemplo si se ingresa la expresión que representa un intervalo unidimensional: "[-2,5]", se compara con la expresión regular: " ^[\[](-)?[0-9]+[.[0-9]*]?,(-)?[0-9]+[.[0-9]*]?]$ ". Que en palabras simples significa que la expresión que ingresó el usuario debe empezar con un '[' seguido quizás por un signo '-' antes de una series de dígitos entre 0 y 9. Luego le puede seguir (como no) un '.' y otra serie de dígitos entre 0 y 9. A continuación debe ir una ',' para luego repetir la posibilidad de que le siga un signo '-' antes de la serie de dígitos entre 0 y 9 que quizás le siga otro '.' y otra serie de dígitos entre 0 y 9. Finalmente la expresión debe terminar con un ']'. Si la expresión que ingresó el usuario no cumple con estas reglas, entonces el intervalo mal ingresado se rechaza y se informa al usuario su error. Para intervalos de dos dimensiones se usa la expresión regular: "^[\[][\[](-)?[0-9]+[.[0-9]*]?,(-)?[0-9]+[.[0-9]*]?][\[](-)?[0-9]+[.[0-9]*]?,(-)?[0-9]+[.[0-9]*]?]]$" que evalúa la expresión ingresada de la misma forma, con la salvedad de que busca en su interior el segundo componente del intervalo.

Una vez validado el intervalo, se crea un objeto *interval_vector* que contiene 'n' intervalos (uno para cada dimensión). Por ejemplo si se ingresa el intervalo "[[0,4][2,6]]", entonces se crea un objeto *interval_vector* que contiene un objeto *interval* de valores [0,4] y otro de valores [2,6]. Luego se crea un objeto *intervalo_grafico* que contiene al objeto *interval_vector*, y le asigna un color y un estilo de achuramiento predefinido. Finalmente este *intervalo_grafico* se muestra por pantalla gracias a su método *"dibujar"* y se agrega a la lista de intervalos gráficos *"lista_IG"* que almacena en memoria todos los intervalos gráficos que se están manejando en ese momento.

Una vez creado el intervalo gráfico, este se puede manipular cambiando su color por cualquiera de los colores que muestra el menú. Para esto, solo es necesario seleccionar el intervalo deseado (pinchando con el mouse dentro de cualquier punto del área que ocupa

ese intervalo) y luego pinchar el color deseado. De la misma forma también se puede cambiar su estilo de achuramiento por alguno de los cuatro estilos que muestra el menú. Nótese que al seleccionar un intervalo, este se volverá amarillo (un color reservado para el sistema). Esto es solo para efectos de resaltar el intervalo seleccionado, el color del intervalo seleccionado no ha variado con esta acción. En la figura 7.7 se puede ver una captura de una selección de un intervalo.

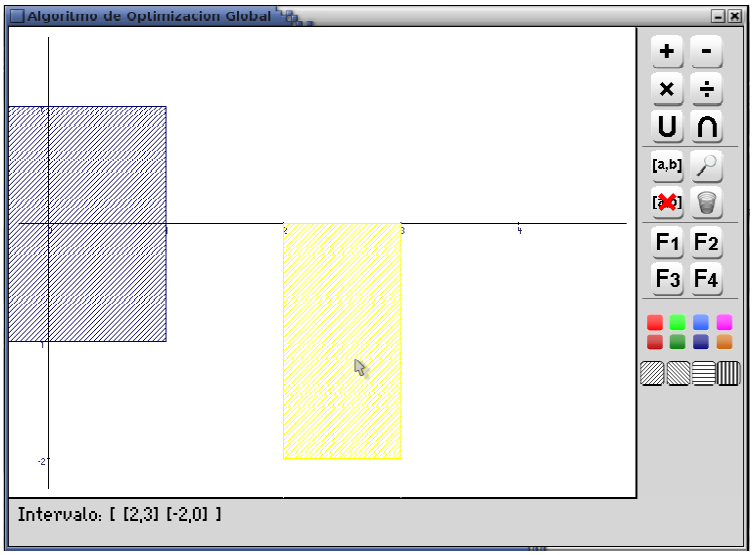

Figura 7.7: Selección de un intervalo

7.2.6 Eliminación de Intervalos

Existen dos formas de eliminar los intervalos almacenados en memoria por la plataforma. Uno de ellos consta de seleccionar un intervalo y luego pinchar el botón 'eliminar intervalo' representado por el símbolo [a,b] con una X roja sobre él. Esta operación busca el intervalo seleccionado en la lista de intervalos gráficos (*lista_IG lista*) y lo elimina de ahí.

La otra forma es pinchar el botón con el dibujo de una papelera. Lo que hace esta operación es vaciar la lista de intervalos gráficos, eliminando todos los intervalos en memoria.

7.2.7 Operaciones con Intervalos

Para realizar cualquiera de las operaciones aritméticas implementadas en la plataforma, primero se deben seleccionar dos intervalos. Si se seleccionan más de dos intervalos, la operación se realizará sobre los dos últimos intervalos seleccionados, que se resaltarán en amarillo. Luego se crea un objeto *interval_vector* que almacenará el resultado de la operación y un objeto *intervalo_grafico* que guarde el objeto anterior. A su vez, la clase *interval_vector* crea dos intervalos para guardar el resultado de la operación seleccionada por cada componente del *interval_vector*, y es ésta la clase que realiza finalmente la operación aritmética seleccionada. Una vez terminada la operación se grafica en rojo el intervalo resultado y se guarda el *intervalo_grafico* en la lista de intervalos gráficos que maneja la plataforma.

7.2.8 Zoom

Para manipular el zoom de una forma más cómoda que presionar la tecla ‘+’ y moverse por el eje hasta la zona deseada, se implementó una forma de aumentar el zoom sobre un área seleccionada por el mouse. Para esto, el usuario debe pinchar el botón con una lupa en el menú (notará que el puntero cambiará de forma) y luego pinchar en dos puntos que formen la selección deseada como se muestra en la figura 7.8.

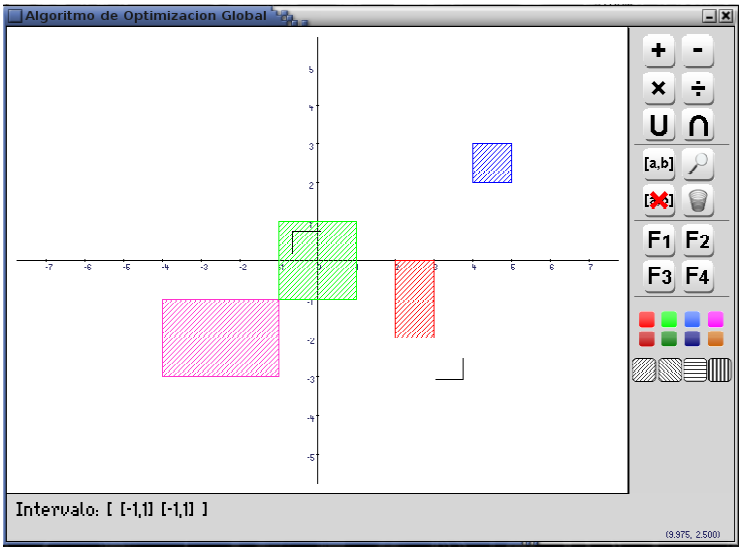

Figura 7.8: Selección de la región a aumentar

Después de haber seleccionado el segundo punto, el zoom aumenta la región deseada, centrándola en la ventana como se muestra en la figura 7.9 a continuación.

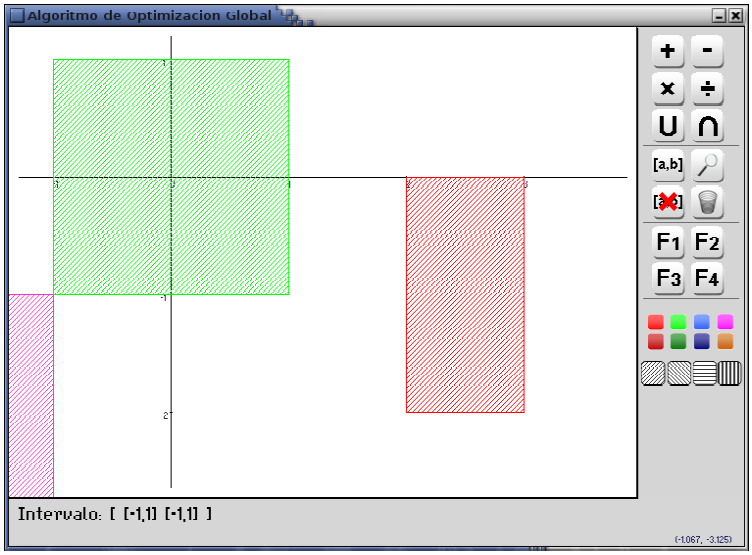

Figura 7.9: Región ampliada.

7.2.9 Casos Especiales de la División de Intervalos

A continuación se expondrán los diferentes tipos o casos especiales que se pueden presentar con la división de intervalos, específicamente por la denominada división por cero, cuyos casos fueron mencionados en el capitulo 6.1.2 (Modificaciones y Correcciones a la Librería de Aritmética Intervalar).

Se define de forma general: X= [a, b] e Y= [c, d], entonces:

Caso 1: Si $0 < a \wedge c < 0 < d \Rightarrow X/Y = [-\infty, a/c] \cup [a/d, \infty]$

Caso 2: Si $0 \in [a, b] \wedge 0 \in [c, d] \Rightarrow X/Y = [-\infty, \infty]$

A continuación, en la figura 7.10, ambos casos de forma gráfica:

Sea $X=\begin{bmatrix}[2,3]\\[0,-2]\end{bmatrix}$ e $Y=\begin{bmatrix}[-1,1]\\[-1,1]\end{bmatrix}$, entonces $X/Y = \begin{bmatrix}[-\infty,-2]\\[-\infty,+\infty]\end{bmatrix} \cup \begin{bmatrix}[2,+\infty]\\[-\infty,+\infty]\end{bmatrix}$

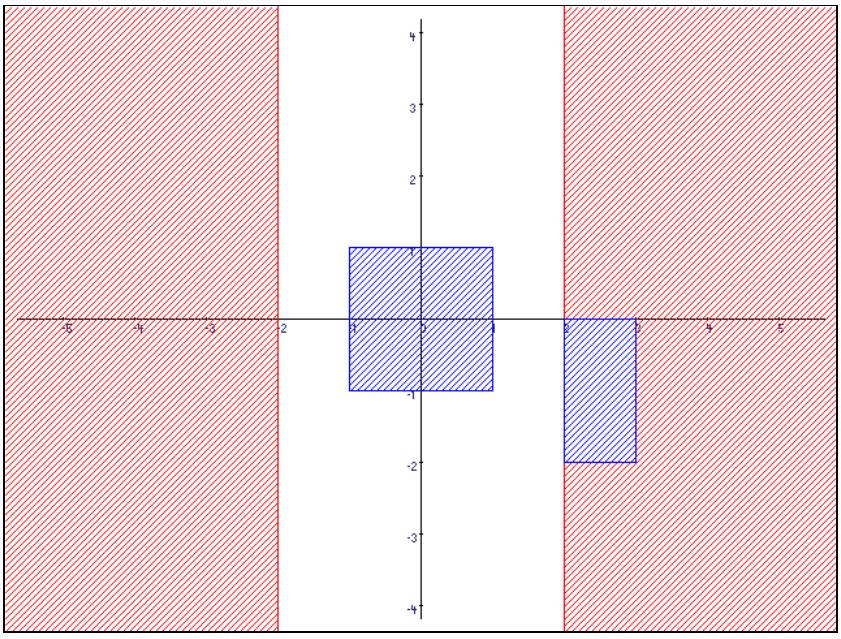

Figura 7.10: Caso 1(Eje X) – Caso 2(Eje Y).

El caso 1 se refleja en el eje X mientras que el caso 2 se muestra en el eje Y.

Caso 3: Si $0 \notin [c, d] \Rightarrow X/Y = [a,b] * [1/d, 1/c]$

En la figura 7.11, se ven representados los intervalos X e Y con valores asignados:

Sea $X=\begin{bmatrix}[-1,1]\\[-1,1]\end{bmatrix}$ e $Y=\begin{bmatrix}[2,3]\\[-2,0]\end{bmatrix}$, entonces $X/Y = \begin{bmatrix}[-1/2,1/2]\\[-\infty,+\infty]\end{bmatrix}$

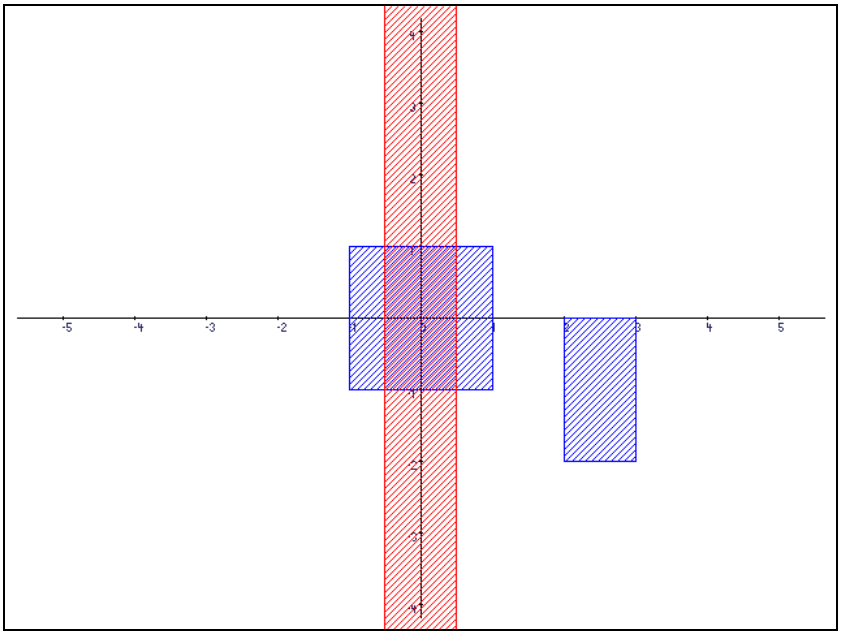

Figura 7.11: Caso 3(Eje X) – Caso 2(Eje Y).

La situación en la cual en el eje X se encuentra representado el caso 2 y en el eje Y el caso 3 se muestra en la figura 7.14.

Caso 4: Si $b < 0 \wedge c < 0 < d \Rightarrow X/Y = [-\infty, b/d] \cup [b/c, \infty]$

A continuación el caso de forma gráfica (figura 7.12):

Sea $X=\begin{bmatrix}[-4,-1]\\[-3,-1]\end{bmatrix}$ e $Y=\begin{bmatrix}[-1,1]\\[-1,1]\end{bmatrix}$, entonces $X/Y = \begin{bmatrix}[-\infty,-1]\\[-\infty,-1]\end{bmatrix} \cup \begin{bmatrix}[1,+\infty]\\[1,+\infty]\end{bmatrix}$

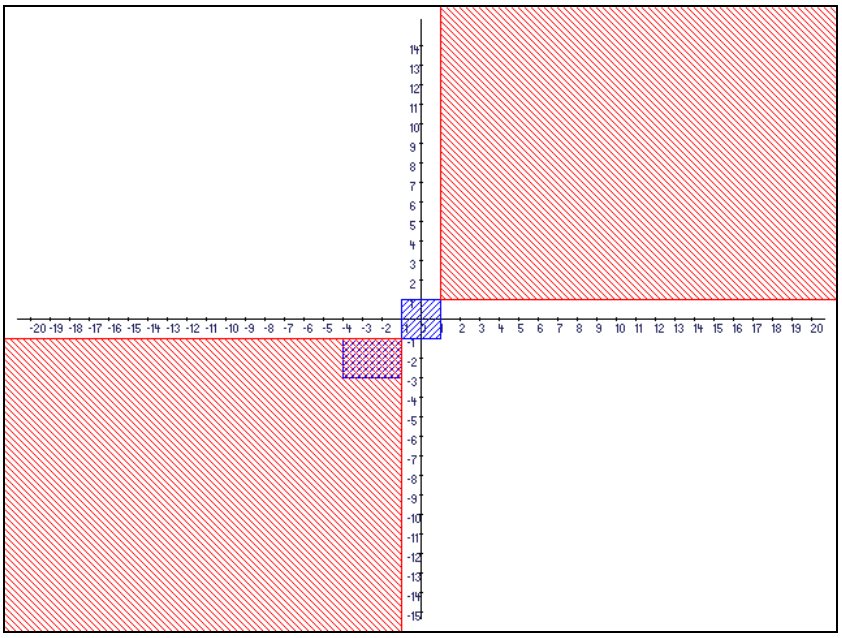

Figura 7.12: Caso 4(Eje X) – Caso 4(Eje Y).

En la figura 7.13 se muestra el caso de $\begin{bmatrix}[-\infty,+\infty]\\[-\infty,+\infty]\end{bmatrix}$ o sea dos veces el caso 2.

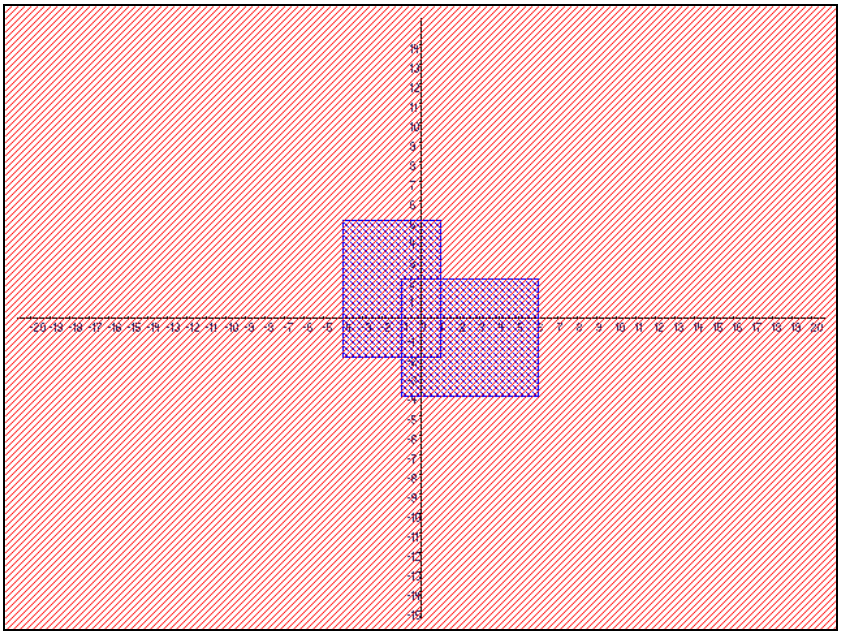

Figura 7.13: Caso 2(Eje X) – Caso 2(Eje Y).

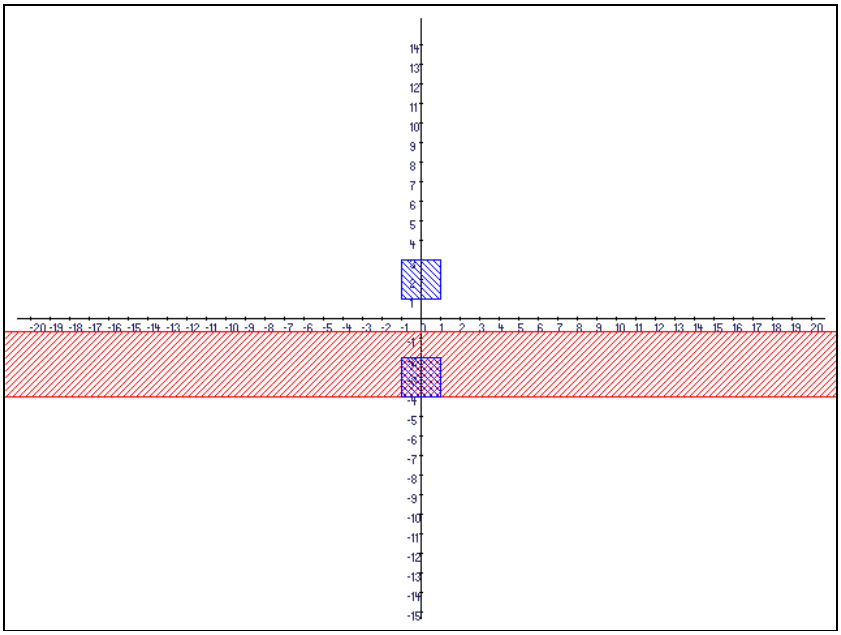

Figura 7.14: Caso 2(Eje X) – Caso 3(Eje Y).

Caso 5: Si $b < 0 \wedge c < d = 0 \Rightarrow X/Y = [b/c, \infty]$

Ejemplo:

Sea $X=\begin{bmatrix}[-5,-1]\\[-6,-3]\end{bmatrix}$ e $Y=\begin{bmatrix}[-1,0]\\[1,2]\end{bmatrix}$, entonces $X/Y = \begin{bmatrix}[1,\infty]\\[-6,-1/2]\end{bmatrix}$ (figura 7.15)

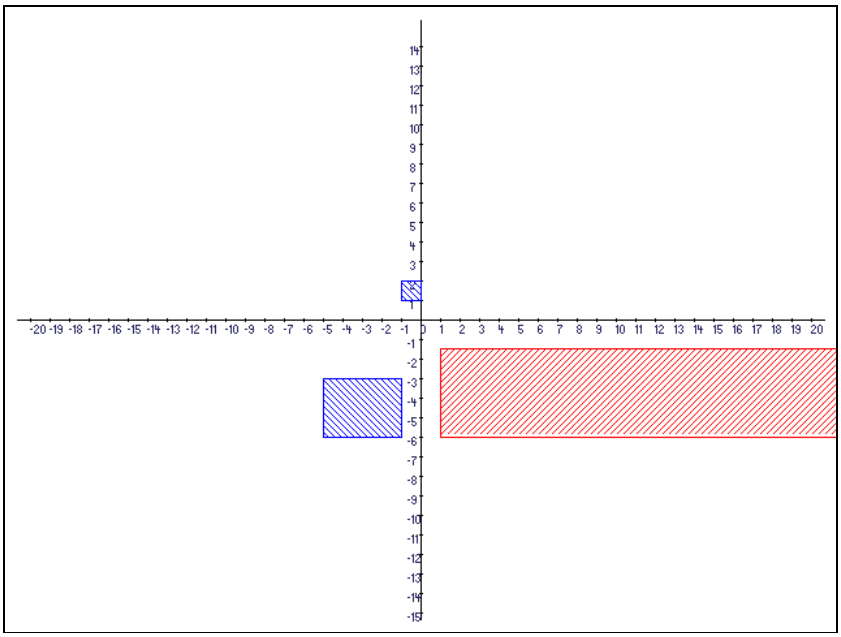

Figura 7.15: Caso 5(Eje X) – Caso 3(Eje Y).

Los casos que restantes son gráficamente parecidos por lo que no será necesario entrar en mayor detalle.

Capítulo 8: Conclusiones

Durante el desarrollo del presente proyecto, se realizó un estudio de la aritmética intervalar, un concepto totalmente nuevo para nosotros, que se utiliza entre otras cosas para resolver los problemas de optimización global. El uso de esta aritmética surge de la necesidad de buscar una solución al problema de redondeo de los cálculos por computador. Aprendimos además con este estudio, la importancia de contar con algoritmos de cálculo que entreguen información que no se ve comprometida por los límites de la arquitectura de las máquinas que los realizan, ya que pequeñas variaciones en los resultados pueden significar grandes problemas.

También podemos concluir que la mayoría de las aplicaciones se explican y comprenden de mejor manera utilizando algún tipo de interfaz gráfica, pues esta facilita su aprendizaje y asimilación.

Nos enorgullece la oportunidad de aportar en la librería de aritmética intervalar del profesor Pedro Campos, ya que la unión e intersección, las cuales ayudamos a implementar, son las operaciones que mejor muestran el funcionamiento de la aritmética intervalar dado la naturalidad de su resultado. No podemos dejar de mencionar a la división por cero que si bien su implementación en la aritmética no presentó mayor problema si lo fue su implementación en la parte gráfica, dado el concepto de infinito que esta lleva asociada.

Un concepto, en el cual pudimos profundizar es la programación orientada a objetos, que nos sorprendió gratamente, tanto por sus características propias como la de polimorfismo, herencia y abstracción así como la encapsulación y el ocultamiento de información, estas dos últimas bastantes útiles en el desarrollo de este trabajo ya que permiten incluir dentro de un objeto los datos y los métodos que emplean dichos datos.

A modo de conclusión más personal, creemos que si bien la plataforma que se desarrolló no esta enfocada a un grupo muy amplio de gente, si se puede destacar por la funcionalidad que ofrecerá a ellos, la cual permitirá continuar desarrollando algoritmos, modelos y técnicas que mejoren la optimización global utilizando la aritmética intervalar, ya que ahora podrán ver representados en la pantalla de sus computadores las distintas operaciones y características de los números intervalos.

En ningún caso creemos que este trabajo y la aplicación que del se generó, son perfectos, pero concluimos y creemos firmemente que hemos hecho un aporte para aquellos que se adentren a este, muchas veces, desconocido mundo de los números intervalos.

Fuentes Bibliográficas

(Andersen *et al*, 1972) Andersen, L. S., Jennings, R. S. y Ryan, D. M. Global Optimization. En R. S. Andersen, editor, Optimización. University of Queensland Press, 1972.

(Campos, 2004) Campos, Pedro, Desarrollo de una Plataforma de Optimización Global utilizando Aritmética por Intervalos, Memoria de Título, Departamento de Computación e Informática, Universidad de Tarapacá, Chile (2004).

(Campos, 2005) Campos, Pedro, "State estimation under parameter uncertainty: Approach by IMHSE Method", Tesis de Magíster, Departamento de Computación e Informática, Universidad de Tarapacá, Chile (2005).

(Campos y Valdés-González, 2006) Campos, Pedro y Valdés-González, H. M. Implementación de un algoritmo basado en aritmética de intervalos para optimización global (2006).

(Ceballos, 1993) Ceballos Sierra, Francisco Javier. Curso de programación en C++, Programación orientada a objetos, 1993.

(Deitel y Paul, 1995) Deitel, Harvey M. y Paul, J. Como programar en C/C++, México Pearson Educación 1995.

(Foley *et al*, 1994) Foley, J. D. – Van Dam, A. – Feiner, S.K. – Hughes, J.F. – Phillips, R.L., "Computer Graphics: Principles and Practice", Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, E.U.A., 1994.

(Hansen, 1965) Hansen, E. R., Interval arithmetic in matrix computations – part I. SIAM Journal on Numerical Analysis: Series B2 (2), (1965), pp. 308-320.

(Hansen, 1992) Hansen, E. R., Global Optimization Using Interval Analysis. Marcel Dekker, Inc., New York, 1992.

(Hickey y **van Emden, 2001)** Hickey, Timothy J. y van Emden, Maarten H. Interval Arithmetic from Principles to Implementation, Journal of the ACM, volume 48, issue 5, Septiembre 2001, pp. 1038-1068.

(Jaulin *et al*, 2001) Jaulin, L - Kieffer, N - Didrit, O - Walter, E. Applied Interval Analysis. Springer-Verlag, London, 2001.

(Leclerc, 1991) Leclerc, A. P. Technical report OSU-CISRC-01/91-TR-04. The Ohio State University Computer and Information Science Research Center, Enero, 1991.

(Leclerc, 1992) Leclerc, A. P. Efficient and reliable global optimization. PhD thesis, The Ohio State University, 1992.

(Moore, 1959) Moore, R. E. Automatic error análisis in digital computation. Technical Report LMSD-48421, Loockheed Missiles and Space Co, Palo Alto, CA, Estados Unidos (1959).

(Moore, 1966) Moore, R. E. Interval Analysis, 1º edición, Prentice-Hall, Englewood Cliffs, NJ, Estados Unidos (1966).

(Muñoz y Peña, 2007) Muñoz, I. y Peña, C. Optimización de Biblioteca de Cálculo Científico utilizando Matemática Intervalar (2007).

(Nickel, 1966) Nickel, K. Über die Notwendigkeit einer Fehlerschranken-Arithmetik für Rechenauto-maten. Numerische Mathematik. Vol 9, (1966), pp. 69-79. Tomado de Jaulin et al., Applied Interval Analysis. Springer-Verlag, London (2001).

(Pressman, 2005) Pressman, Roger E. Ingeniería del Software: Un enfoque práctico. Mcgraw Hill, Interamericana, México D.F (2005).

(Ratz, 1996) Dietmar Ratz, On extended interval arithmetic and inclusion isotonicity. Institut fur Angewandte Mathematik, Universität Karlsruhe (1996).

(Sahinidis, 2004) Sahinidis, N. V., Optimization Under uncertainty: state-of-the-art and opportunities. Computers and Chemical Engineering. Vol. 28, (2004), pp. 971-983.

(Skelboe, 1974) Skelboe, S., "Computation of rational interval functions", BIT 14, 87-95 (1974).

(Sutherland, 1963) Sutherland, I.E., "Sketchpad: A Man-Machine Graphical Communication System", en SJCC, Spartan Books, Baltimore, Maryland, 1963.

(Trafalis y Kasap, 2002) T. Trafalis, y S. Kasap. A novel metaheuristics approach for continuous global optimization. Journal of Global Optimization 23 pp. 171–190, 2002.

(Van Iwaarden, 1996) R. J. VAN IWAARDEN. An improved unconstrained global optimization algorithm. PhD thesis, University of Colorado at Denver, 1996.

(Vivallos, 2007) Vivallos, J., Implementación de Algoritmos de Cálculo Intervalar en C++. Informe de Habilitación Profesional, Ingeniería Civil Informática, Departamento de Sistemas de Información, Universidad del Bío-Bío, 2007.

ANEXOS

Anexo A

Código fuente que inicializa y cierra la librería SDL.

```
#include <SDL.h>   /* Librería SDL */
#include <stdio.h>

int main() {

   printf("Inicializando SDL.\n");

   /* Inicialización de los subsistemas por defecto, Video y Audio */
   if((SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO)==-1)) {
      printf("No se pudo inicializar SDL: %s.\n", SDL_GetError());
      exit(-1);
   }

   printf("SDL inicializado.\n");

   printf("Cerrando SDL.\n");

   /* Se cierran todos los subsistemas */
   SDL_Quit();

   printf("Saliendo...\n");
```

```
   exit(0);
}
```

Anexo B

Código fuente que inicializa SDL y crea una ventana gráfica.

```
#include <SDL.h>   /* Librería SDL */
#include <stdio.h>

int main() {

printf("Inicializando SDL.\n");

SDL_Surface *screen;

   /* Inicio de la librería SDL */
   if( SDL_Init(SDL_INIT_VIDEO) < 0 ) {
      fprintf(stderr,
            "No se pudo inicializar SDL: %s\n", SDL_GetError());
      exit(1);
   }

printf("SDL inicializado.\n");

   /* Cierre de los subsistemas al cierre de la aplicación */
   atexit(SDL_Quit);

   /*
    * Inicio del área gráfica a una resolución de 640x480 a 8 bits,
```

```
    * solicitando una superficie gráfica
    */
printf("Abriendo ventana.\n");

   screen = SDL_SetVideoMode(640, 480, 8, SDL_SWSURFACE);
   if ( screen == NULL ) {
      fprintf(stderr, "No se pudo setear el modo de video a 640x480x8: %s\n",
                  SDL_GetError());
      exit(1);
   }

getchar();
exit(0);
}
```

Anexo C

Uso de la plataforma actual.

La plataforma esta constituida por los siguientes archivos fuentes, desarrollados en C++ y compilados con *gcc*, que contienen las clases diseñadas:

- interval.h
- interval.cpp
- interval_math.h
- interval_math.cpp
- interval_vector.h
- interval_vector.cpp

La clase interval representa como su nombre lo dice a los intervalos, para utilizarla solo se le deben asignar valores y usar los operadores aritméticos deseados. Con el

operador << se imprime el valor de una instancia. Para obtener el valor del extremo izquierdo de una instancia se utiliza la función *left(x)* y *right(x)* para el extremo derecho, donde *x* es la instancia en cuestión.

De la misma forma se utilizan las funciones *pow(x, n)* y *sqrt(x)*, donde *n* corresponde a la potencia a la que se desea elevar la instancia *x.*

Para utilizar la optimización global, se debe crear una instancia de la clase *globalMin*. En caso que no se utilice una función de restricción, se requiere implementar la función que se desea optimizar como extensión por intervalos y como función real (para evaluación del punto medio). Para implementar la extensión por intervalos, basta con definir un procedimiento que reciba como argumento un vector de intervalos que corresponde al vector de variables de la función, y que simplemente realice la operatoria que corresponda a la función, retornando dicho valor como un intervalo. Para implementar la evaluación del punto medio, se debe definir un procedimiento similar al anterior, pero en este caso se deben calcular los puntos medios de cada variable (utilizando el método *mid()*), y se realiza la operatoria con estos valores, retornando un valor de tipo *double*. La función de restricción se implementa de acuerdo a las restricciones que se desee que cumpla la función. Esta última función se debe implementar como extensión por intervalos, de manera que su resultado sea un intervalo. Para implementar las pruebas de rechazo mediante gradiente, se ha incluido la librería *interval_autodiff.h* junto con su respectiva implementación en un fichero .cpp, además de indicar en la librería *interval_directives.h* que se debe hacer uso de gradiente para el descarte de regiones que no contienen óptimos (Muñoz y Peña, 2007).

Anexo D

Modificación a la operación multiplicación de un intervalo por un intervalo.

```
interval operator * (const interval &x, const interval &y)
{
        interval res;

        SetRoundDown();
        if (sign (x.left) == 0) {                       /* x >= 0 */
           if (sign (y.left) == 0) {                        /* y >= 0 */
                        res.left = x.left * y.left;
                        SetRoundUp();
                res.right = x.right * y.right;
           }
                else if (sign (y.right)) {                      /* y <= 0 */
                        res.left = x.right * y.left;
                        SetRoundUp();
                   res.right = x.left * y.right;
                }
           else {                                       /* 0 en y */
                        res.left = x.right * y.left;
                        SetRoundUp();
                        res.right= x.right * y.right;
                }
        }
        else if (sign (x.right)) {                      /* x <= 0 */
                if (sign (y.left) == 0) {                       /* y >= 0 */
                res.left = x.left* y.right;
                SetRoundUp();
```

```
                res.right = x.right * y.left;
                }
                else if (sign (y.right)) {                      /* y <= 0 */
                        res.left= x.right* y.right;
                        SetRoundUp();
                        res.right= x.left * y.left;
                }
                else {                                  /* 0 en y */
                        res.left= x.left* y.right;
                        SetRoundUp();
                        res.right= x.left * y.left;
                }
        }
        else {                                  /* 0 en x */
                if (sign (y.left) == 0) {                       /* y >= 0 */
                        res.left= x.left* y.right;
                        SetRoundUp();
                        res.right = x.right* y.right;
                }
/*CORREGIDO EL ELSE IF... originalmente decia 'sign (y.left)'*/
                else if (sign (y.right)) {                      /* y <= 0 */
                        res.left= x.right* y.left;
                        SetRoundUp();
                        res.right = x.left * y.left;
                }
                else {                                  /* 0 en y */
                        double r1 = x.left * y.right;
                        double r2 = x.right * y.left;

                        res.left= r1 < r2 ? r1 : r2;
```

```
                    SetRoundUp();

                    r1 = x.left * y.left;
                    r2 = x.right * y.right;

                    res.right = r1 > r2 ? r1 : r2;
                }
            }
            SetRoundDef();
            return res;
}
```

Anexo E

Definición de algunas clases del proyecto.

Clase: gestor_grafico

Nombre Método: menu

Parámetros de entrada: No posee.

Descripción: Método que carga la imagen que servirá de menú para la aplicación.

Retorno:

Nombre Método: posicion_mouse

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
x	entero	Posición del puntero en el eje de abcisas o eje X o el eje horizontal.
y	entero	Posición del puntero en el eje de ordenadas o eje Y o el eje vertical.

Descripción: Método que muestra en que posición del plano cartesiano se encuentra el puntero del mouse.

Retorno: No posee.

Nombre Método: drawrect

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
x	entero	Coordenada del eje X del extremo superior izquierdo del rectángulo.
y	entero	Coordenada del eje X del extremo superior izquierdo del rectángulo.
width	entero	Ancho del rectangulo.

height	entero	Altura del rectangulo.
c	entero	Color del rectangulo.

Descripción: Método que dibuja un rectángulo, en una posición dada con un color determinado.

Retorno: No tiene.

Nombre Método: borrar

Parámetros de entrada: No posee.

Descripción: Método que limpia el área de trabajo y dibuja el eje cartesiano.

Retorno: No posee.

Nombre Método: borrar_todo

Parámetros de entrada: No posee.

Descripción: Método que limpia el área de trabajo, el área de mensaje y el área de menú.

Retorno: No posee.

Nombre Método: borrar_abajo

Parámetros de entrada: No posee.

Descripción: Método que limpia el área de mensajes.

Retorno: No posee.

Nombre Método: inicializar

Parámetros de entrada: No posee.

Descripción: Método que inicializa la librería grafica.

Retorno: No posee.

Nombre Método: left

Parámetros de entrada: No posee.

Descripción: Este método permite al usuario desplazarse hacia la izquierda por el eje de coordenadas.

Retorno: No posee.

Nombre Método: right

Parámetros de entrada: No posee.

Descripción: Este método permite al usuario desplazarse hacia la derecha por el eje de coordenadas.

Retorno: No posee.

Nombre Método: up

Parámetros de entrada: No posee.

Descripción: Método que permite al usuario desplazarse hacia arriba por el eje de coordenadas.

Retorno: No posee.

Nombre Método: down

Parámetros de entrada: No posee.

Descripción: Método que permite al usuario desplazarse hacia abajo por el eje de coordenadas.

Retorno: No posee.

Nombre Método: zoom_in

Parámetros de entrada: No posee.

Descripción: Este método aumenta el valor del zoom de la zona de trabajo, de acuerdo al factor de zoom definido.

Retorno: No posee.

Nombre Método: zoom_out

Parámetros de entrada: No posee.

Descripción: Este método disminuye el valor del zoom de la zona de trabajo, de acuerdo al factor de zoom definido.

Retorno: No posee.

Nombre Método: eje

Parámetros de entrada: No posee.

Descripción: Método que dibuja el eje cartesiano en la zona de trabajo.

Retorno: No posee.

Nombre Método: texto_inferior

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
contenido	Cadena de caracteres	Texto que se mostrara en la zona de mensajes.

Descripción: Método que muestra un texto en la zona de mensajes.

Retorno: No posee.

Nombre Método: muestra_texto

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
contenido	Cadena de caracteres	Texto que se mostrara en pantalla.
posicion_x	entero	Posición del eje X en el cual se mostrara el mensaje.
posicion_y	entero	Posición del eje Y en el cual se mostrara el mensaje.
size	entero	Tamaño de la fuente del mensaje.
color_texto	entero	Color de la fuente.

Descripción: Método que dibuja un rectángulo, en una posición dada con un color determinado.

Retorno: No tiene.

Clase: g_interval_vector

Nombre Método: g_interval_vector

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
I	Cola de vectores de intervalos	Estructura del tipo cola que almacena vectores de intervalos.
color	entero	Color con el que se representara un intervalo.
achurado	entero	Tipo de achurado con el que se dibujara un intervalo.

Descripción: Método que crea un vector de intervalos gráficos

Retorno: No posee.

Nombre Método: selección

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
conjunto	vector de intervalos gráficos	Intervalos que se encuentran en la posición de selección del mouse.
x	entero	Posición en el eje x del puntero del mouse
y	entero	Posición en el eje y del puntero del mouse
X,	entero	Factor de posición del eje x.
Y,	entero	Factor de posición del eje y.
ESCALA	entero	relación entre el tamaño de la pantalla y el tamaño del eje.

Descripción: Método que retorna el vector de intervalos gráficos seleccionados**.**

Retorno: selección, vector de intervalos gráficos.

Nombre Método: push

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
I	cola de intervalos gráficos	Estructura del tipo cola que almacena vectores de intervalos.
color,	entero	Color con el que se representara un intervalo.
achurado	entero	Tipo de achurado con el que se dibujara un intervalo.

Descripción: Método que inserta un vector de intervalos gráficos a una cola de intervalos gráficos.

Retorno: No posee.

Nombre Método: push

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
I	cola de intervalos gráficos	Estructura del tipo cola que almacena vectores de intervalos.

Descripción: Método que inserta un intervalo a una cola de intervalos gráficos.

Retorno: No posee.

Nombre Método: dibujar

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
gestor	gestor grafico	Objeto que maneja la ventana SDL que se creo para ejecutar la aplicación.

Descripción: Método que dibuja un vector de intervalos graficos.

Retorno: No posee.

Clase: intervalo_grafico

Nombre Método: dibujar

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
gestor	gestor grafico	Objeto que maneja la ventana SDL que se creo para ejecutar la aplicación.

Descripción: Método que dibuja un intervalo grafico.

Retorno: No posee.

Clase: menu

Nombre Método: boton

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
x	entero	Posición en el eje x del puntero del mouse
y	entero	Posición en el eje y del puntero del mouse

Descripción: Método que determina la opción del menú que se selecciono.

Retorno: posición, entero.

Nombre Método: ingresar

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
gestor	gestor grafico	Objeto que maneja la ventana SDL que se creo para ejecutar la aplicación.

Descripción: Método que lee un intervalo por teclado.

Retorno: No posee.

Nombre Método: valida_expresion

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
expresion	Cadena de caracteres	Cadena de caracteres que contiene texto ingresado por teclado
dimension	entero	Dimensión del intervalo que supuestamente se ingreso

Descripción: Método que determina si el intervalo que se acaba de ingresar esta validamente escrito.

Retorno: res, entero.

Nombre Método: extrae_intervalos

Parámetros de entrada:

Nombre	Tipo de dato	Descripción
expresion	Cadena de caracteres	Cadena de caracteres que contiene texto ingresado por teclado.

Descripción: Método que extra los componentes del texto ingresado por teclado y los retorna transformados en un vector de intervalos.

Retorno: U, vector de intervalos.

Anexo F

Métodos de Optimización Global

Los algoritmos de optimización global se clasifican en probabilísticos o determinísticos y a su vez en confiables y no confiables.

Métodos Probabilísticos

Los métodos de optimización probabilísticas, también llamados métodos estocásticos, son procesos infinitos en los cuales la probabilidad de encontrar un óptimo tiende a 1 cuando el numero de paso que se requiere para tal tarea, tiende a infinito. Con esto se infiere que no se puede garantizar, en un número finito de pasos, encontrar un óptimo global. De lo anterior podemos clasificar estos algoritmos como no confiables, ya que solo se puede ejecutar una cantidad finita de pasos. No obstante se debe señalar que la fortaleza de estos métodos recae en su eficiencia, ya que en la mayoría de los casos permite encontrar rápidamente algún óptimo local. Por lo anterior los problemas de gran escala pueden ser resueltos de mejor manera utilizando un método estocástico de optimización. (Leclerc, 1991, 1992; Van Iwaarden, 1996)

El algoritmo probabilístico más básico, y por lo tanto más fácil de explicar, se llama Random Search (Búsqueda al Azar en español), el cual está basado en una función aleatoria para explorar el valor óptimo, partiendo desde un punto inicial aleatorio y buscando puntos, también al azar, en los que disminuye el valor de la función objetivo. Este método se encuentra ampliamente estudiado, y se ha llegado a demostrar que si los puntos en los que se evalúa la función tienen una distribución uniforme sobre la región de búsqueda, los valores mínimos encontrados se acercan aún más al mínimo global mientras mayor sea la cantidad de iteraciones de búsqueda, ósea convergen al mínimo. (Andersen *et al*, 1972)

Otro método probabilístico, el cual ha sido usado con buenos resultados es llamado Simulated Annealing (Temple Simulado en español), el cual se basa en la idea del calentamiento de los metales y su enfriamiento paulatino para encontrar su nivel más bajo de energía. Otro método estocástico que se puede mencionar es el llamado Clustering Method o Método de Clústeres. Un Clustering Method comienza con una muestra uniforme de puntos de la región de búsqueda y entonces se crean grupos o clústeres de puntos cercanos que corresponden a una región común de atracción; cada región de atracción contiene exactamente un mínimo local y cualquier algoritmo que comienza en un punto cualquiera de la región converge al mínimo local de dicha región. (Muñoz y Peña, 2007)

Métodos Determinísticos

Los métodos determinísticos presentan grandes diferencias con los métodos estocásticos, que se pueden generalizar en dos puntos. Primero no se basan en la utilización de un numero infinito de pasos para encontrar un optimo, pues su tiempo de ejecución es finito, pero esto no quita que este tiempo de ejecución sea bastante extenso, lo que si los diferencian es que entregan un conjunto de zonas en las que si se garantiza que están todos los óptimos globales. La segunda diferencia radica en que mientras los métodos estocásticos garantizan encontrar, aunque sea en un tiempo infinito, un único óptimo global, cuando existe la posibilidad que existan múltiples óptimos globales; los métodos determinísticos, en cambio, garantizan que todos y cada uno de los óptimos globales están contenidos dentro de las regiones que entregan

Dentro de los algoritmos determinísticos destacan los que pertenecen a la llamada familia Branch and Bound, Ramificación y Acotación en español, ya que son unos de los pocos métodos pueden ser aplicados a una gran variedad de problemas con muy poco o nada de conocimiento del estado de la función que se quiere minimizar, además pueden ser aplicados muy fácilmente a métodos con dominios cóncavos o convexos, funciones objetivo cóncavas o convexas, y funciones para las cuales se desconoce si son continuas o no. Aunque por lo general se posee alguna información sobre la función a optimizar, algunos métodos Branch and Bound no la requieren en lo absoluto. Otros métodos, de la

misma familia, utilizan la información que van descubriendo activamente, calculando las derivadas necesarias para explorar la estructura de la función. Branch and Bound, al igual que Simmulated Annealing, es una técnica que ha sido exitosamente aplicada a problemas de optimización. Una de sus versiones mas básicas es el Branch and Bound Entero, el cual fue adaptado para resolver el problema de optimización global no lineal en el Branch and Bound Continuo.(Muñoz y Peña, 2007)

Los métodos de optimización determinísticos mas utilizados son los llamados Métodos por Cotas ya que si se considera el error de redondeo puede llegar a entregar soluciones bastante rigurosas de optimización global. (Vivallos, 2007)

Método de Cotas

Estos métodos tienen como objetivo producir al menos una cota inferior para el rango de valores de f sobre un cierto conjunto compacto. Dentro de las muchas técnicas que existen para construir cotas inferiores de funciones, la mayoría de ellas operan sobre conjuntos compactos convexos, los cuales son *hiperrectángulos*, o rectángulos de n-dimensiones, y realizan algunas suposiciones acerca de la función objetivo(Vivallos, 2007).

Una de estas técnicas se conoce como la *Aproximación Lipschitziana*, la cual supone que existe una constante de Lipschitz, L, tal que:

$$\left| f(x_1) - f(x_2) \right| \leq L \left\| x_1 - x_2 \right\|$$

Entonces si el valor de f fuese conocido en algún punto, como x, se puede determinar una cota inferior del valor de la función para todo x entre x_1 y x_2 por:

$$f(x_1) \leq L \left\| x_1 - x_2 \right\|$$

Otra técnica que existe es la llamada *Aproximación de Cota Inferior Lineal*. Esta técnica genera cotas inferiores lineales para una función f, descomponiendo a f en funciones mas simples, tales como funciones monótonas convexas. A continuación se procede a encontrar cotas inferiores lineales de cada función simple, las cuales combinadas forman una cota inferior lineal sobre la función original. Sin embargo el método descrito anteriormente asume que f puede ser descompuesta en funciones más simples, lo cual ha sido probado hasta el momento solo para problemas de optimización global de una dimensión.

Otra técnica más eficaz, que combina conceptos fundamentales para este estudio, es la llamada *Aproximación de Intervalos*, que consiste en calcular cotas inferiores y superiores usando los principios de la aritmética de intervalos; así se puede escribir fácilmente una extensión de intervalos natural, F, para cualquier función programable, f. Dicha función de intervalos opera sobre hiperrectángulos y entrega resultados en intervalos que automáticamente acotan el rango de la función sobre el conjunto de entrada. La principal ventaja que presenta aquí la aritmética intervalar es que considera el error de redondeo.

Optimización Global por Cotas

Si se tiene una función F_1 que calcula la cota inferior del rango de f sobre un conjunto X, entonces se puede implementar un algoritmo exhaustivo de optimización global. El método más utilizado parte dividiendo el conjunto inicial X en varios subconjuntos S_i^x. Luego se debe obtener la cota inferior del valor de la función sobre cada subconjunto $F_L(S_i^x)$. También se debe obtener la cota superior del valor mínimo global calculado hasta este paso, como U_f. En este momento se puede proceder a rechazar todos los subconjuntos S_i^x cuya cota inferior sea mayor a U_f. Este proceso se repite una y otra vez hasta cumplir el criterio de detención.

La real efectividad de estos algoritmos está restringida por la función que permite calcular las cotas inferiores y la habilidad de ésta para entregar cotas inferiores *ajustadas* para el mínimo global sobre un conjunto dado. Si la cota inferior fuera demasiado *cruda*, ósea que posee un valor muy diferente al valor mínimo, la posibilidad de rechazar conjuntos disminuye considerablemente. En cambio, si la función de acotamiento nos entrega cotas mas *ajustadas* para el mínimo global, entonces se podría encontrar una solución con un rango de error ε para el problema de optimización global que se está tratando, sin embargo la mayoría de las funciones de acotamiento inferior no son tan precisas. No obstante, para poder acercarse arbitrariamente al mínimo global, la función de acotamiento inferior debe ser *asintóticamente* precisa, en el sentido que la cota inferior debe acercarse al mínimo global de f sobre un conjunto S a medida que el *volumen* de S se hace mas pequeño. (Vivallos, 2007).

Algoritmo de Optimización Global por Cotas

El algoritmo general que implementan todos los métodos por cotas es el siguiente (Leclerc, 1992):

1) Particionar el espacio inicial de búsqueda en regiones más pequeñas como se muestra en la siguiente figura:

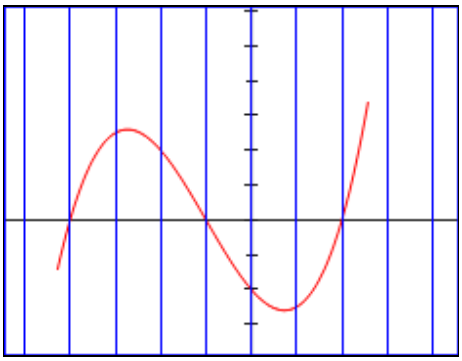

Figura A: Partición del espacio de búsqueda

Para realizar la partición del espacio inicial se debe utilizar una cierta geometría que permita cubrir todo el espacio y que no se intersecten unas a otras. Lo más recomendable es

utilizar subregiones rectangulares para particionar el espacio total. Ya que si se decide, por ejemplo, utilizar subregiones circulares, quedarán espacios entre cada figura que se escaparán al análisis del algoritmo. Y no es para menos, porque puede ocurrir que justo en esos espacios se encuentre el óptimo global.

2) Acotar la función (y posiblemente sus derivadas) sobre las subregiones, como se muestra a continuación en la figura:

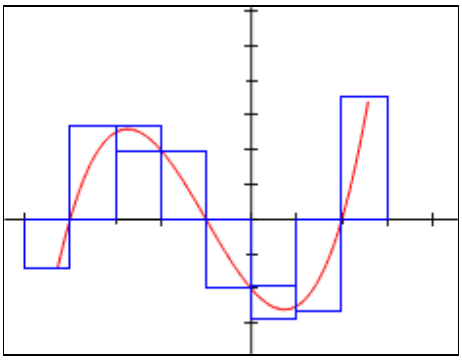

Figura B: Acotamiento de las subregiones

Para esto es necesario una función que se encargue de obtener las cotas de la función objetivo.

3) Rechazar (usando las cotas calculadas en el paso 2) aquellas subregiones que definitivamente no contienen un minimizador global.

Del resultado de este algoritmo se obtiene una o más regiones que contienen a todos los minimizadores globales. Si se busca el máximo global en la función que se está analizando, se llegaría a un resultado, una subregión, como la que se muestra a continuación en la figura:

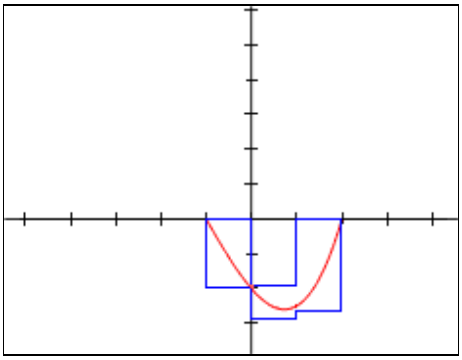

Figura C: Subregión que contiene al mínimo global

En cuanto a las subregiones que se rechazaron, estas lo fueron por una de las siguientes razones:

a) Rechazo de las subregiones cuyas cotas inferiores calculadas en el paso 2 del algoritmo sean mayores que la cota superior del mínimo global conocido hasta ese momento. En otras palabras, se rechazan las regiones que no contienen ningún elemento que sea menor que el mayor de los elementos que contiene la región que guarda al óptimo global.

b) Rechazo de las subregiones que no estén en el espacio factible (es decir, que no cumplen ciertas restricciones impuestas).

c) Si la función es diferenciable (derivable en todos sus puntos):

c.1) Rechazo de las subregiones en las que $0 \notin \nabla f$, donde ∇f es el gradiente de la función objetivo.

c.2) Rechazo de las subregiones donde la función no es convexa en toda la subregión.

Para encontrar más fácilmente las cotas superiores e inferiores de $f(x)$ para puntos factibles, existen los métodos o pruebas de rechazo. A continuación se describen las más utilizadas.

Prueba de Factibilidad

Sea X una subcaja de B. Si se considera que un punto en B es representado como una caja degenerada, entonces se puede evaluar, utilizando intervalos, las funciones de restricción $p_1(X), p_2(X),..., p_k(X)$ y realizar la siguiente prueba. Se dice que X es *certeramente factible* si $p_i(X) \leq 0 \forall i = 1,2,...,k$. Ahora bien, si X es certeramente factible, entonces cada punto $x \in X$ es factible. (Vivallos, 2007).

X es *certeramente infactible,* ósea no contiene puntos factibles, si para algún $i = 1,2,...,k$, $p_i(X) > 0$, X puede ser rechazada, no tomándosele en cuenta en lo sucesivo del proceso.

Prueba de Punto Medio

Sea mX el punto medio (o cualquier otro punto) de una subcaja X de B. Si mX es certeramente factible, entonces la función objetivo es evaluada en mX para obtener un intervalo $f(mX) = \lfloor L_{F_{mX}}, U_{F_{mX}} \rfloor$. Ciertamente, $U_{F_{mX}} \geq f^*$, esto es, $U_{F_{mX}}$ es una cota superior del valor mínimo de $f(x)$ sobre la región factible. Si f es evaluada sobre otra subcaja Y de B, teniendo como resultado $f(Y) = \lfloor L_{F_Y}, U_{F_Y} \rfloor$, entonces se puede realizar la siguiente prueba. Sea U_{F^*}, el menor $U_{F_{mX}}$ encontrado hasta el momento.

Si $L_{F_Y} > U_{F^*}$ entonces Y no puede contener un minimizador global factible en B, por lo tanto Y puede ser rechazada, no tomándosele en cuenta en lo sucesivo.

Usando estas pruebas, se puede llegar a formular un algoritmo muy simple de optimización global con restricciones en forma de desigualdades. El algoritmo es válido tanto si las funciones involucradas son diferenciables como si no lo son. Cuando $f(x)$ y las funciones de restricción son diferenciables, se pueden aplicar pruebas de rechazo más eficientes, basadas en la información de las derivadas de las funciones. Particularmente, existen dos pruebas basadas en derivadas que requieren que $f(x)$ sea diferenciable. Éstas son la prueba de monotonicidad y la prueba de no convexidad. (Hansen, 1992)

La lista de cajas usadas en este algoritmo es técnicamente una cola de espera. Los elementos son añadidos al final de la cola y son removidos del inicio de la misma. Cada vez que se remueve una caja, es biseccionada a lo largo de la dirección (eje) de ancho máximo. Luego se aplican las pruebas a cada mitad. Si con estas pruebas no se puede rechazar una caja, entonces es añadida al final de la cola.

Cabe destacar, la propiedad más importante de este algoritmo, la cual es que en cualquier momento del proceso las cajas remanentes en la lista contienen todos los puntos mínimos globales factibles. (Muñoz y Peña, 2007).

Prueba de Monotonicidad

Si $f(x)$ se asume como continuamente diferenciable, entonces el gradiente $\nabla f(x)$ de $f(x)$ es cero en el mínimo global y obviamente en los mínimos y máximos locales. Si $\nabla F(x)$ es una extensión en intervalos de $\nabla f(x)$, entonces se puede aplicar la siguiente prueba:

Si $0 \in \nabla F_i([S_X])$ para algún $i = 1,2,...,n$ entonces el gradiente no es cero en $[S_X]$. Por lo cual $[S_X]$ no contiene un mínimo global y puede ser eliminado sin volver a ser considerado (Vivallos, 2007).

Prueba de No-Convexo

Si $f(x)$ tiene un mínimo en x^*, entonces $f(x)$ debe ser convexo en algún sector vecino de x^*. Entonces, el Hessiano $\nabla^2 f(x)$ de $f(x)$ deben ser semidefinido positivo en x^*. De acuerdo a esto, los elementos diagonales $\nabla^2 f_{ii}(x)$, $i = 1,2,...,n$ de $\nabla^2 f(x)$ deben ser no negativos. Los elementos diagonales del Hessiano se dan por:

$$\nabla^2 f_{ii}(x) = \frac{\partial^2 f(x)}{\partial x^2},\ i = 1,2,...,n$$

Si $\nabla^2 F(x)$ es una extensión con intervalos de $\nabla^2 f(x)$, entonces se puede aplicar la siguiente prueba:

Si $\nabla^2 F_{ii}([S_X]) \not\subset [0,\infty]$ para algún $i = 1,2,...,n$, entonces $f(x)$ no es convexo en ningún punto de $[S_X]$, y por lo tanto, no contiene un mínimo global y puede ser eliminado sin considerarse nuevamente.