FACULTAD *de* CIENCIAS
de la COMPUTACION

Diálogo Activo para el Cálculo de una Buena Aproximación Polinomial Mediante Métodos Probabilísticos

Que para obtener el título de
Ingeniero en Ciencias de la Computación

Presenta
Alfonso Marquez Nieva

Director de tesis
Prof. Dr., Dr. Sc. Miguel A. Jiménez Pozo
Facultad de Ciencias Físico Matemáticas

Coasesor
Dr. Gerardo Martínez Guzmán
Facultad de Ciencias de la Computación

Puebla, Puebla Abril, 2008

Mis logros
siempre serán tus logros.
Gracias Madre mía.

Índice general

Agradecimientos

Agradezco primeramente a mi Director de Tesis, el Dr. Miguel A. Jiménez Pozo por su paciencia y valiosas aportaciones hacia mi tesis. A la M.C. Ivonne L. Martínez Cortés de la F.C.F.M. por el tiempo dedicado en ayudarme en cada uno de los objetivos. Agradezco también al Dr. Gerardo Martínez Guzmán por sus observaciones y por su ayuda para redactar mi tesis. Igualmente a mi compañera de tesis, Mónica Romero Cruz por dejarme compartir este proyecto de tesis. Quiero agradecer a la SEP–VIEP y al CONACyT por la beca que me fue proporcionada a través de los proyectos que llevan por nombre 37/EXC/07, "*Aproximaciones asociadas a subespacios funcionales*" y "*Aproximación y Suavidad*" respectivamente.

Del mismo modo, quiero agradecer a mi estimado amigo Carlos A. Archundia Loaiza por su ayuda incondicional y por haberme iniciado en el mundo matricial de MATLAB.

Y me permito agradecer especialmente en estas líneas, a mis hermanas; que me brindaron su apoyo cuando más lo necesité. Porque todas dieron lo mejor de sí mismas, para que yo pudiera concluir una etapa en mi vida y empezar nuevamente con otra. Pero quiero hacer un reconocimiento especial a mi hermana *Alicia Nieva*, por sus incontables esfuerzos, junto a mi madre, a lo largo de todos estos años que duró mi paso por la universidad. Gracias *Leticia*, *Rosario* y *Silvia*.

Introducción

Esta introducción y primer capítulo son comunes para las dos tesis de ingeniería en Ciencias de la Computación que enmarcamos en el proyecto de investigación SEP–VIEP con título "Aproximaciones Asociadas a Subespacios Funcionales". Este tema de la aproximación de funciones ha ido ampliando sus horizontes desde los primeros proyectos de investigación en la Facultad de Ciencias Físico Matemáticas de la BUAP. Es dentro de este tema donde se desarrolla esta tesis, una de cuyas partes se enfoca hacia la generalización de un algoritmo conocido como algoritmo de Remez. Existe una abundante y rica teoría que demuestra teóricamente que este algoritmo permite calcular el polinomio de mejor aproximación a una función continua. El objetivo de una de estas tesis es la elaboración de un programa computacional para implementar el algoritmo. Por otra parte, para funciones difíciles de tratar computacionalmente en cuanto a operaciones matemáticas, queremos igualmente elaborar en la otra tesis un algoritmo inspirado en métodos probabilísticos.

Gracias a sus singulares y fascinantes capacidades; el lenguaje MATLAB ha cambiado el concepto de programación para el Análisis Matemático, por lo cual, será empleado en estas tesis. Este lenguaje tiene integrado cinco elementos fundamentales:

1. Programación en MATLAB.
2. Fundamentos matemáticos del Análisis Matemático.
3. Aplicación de métodos numéricos a problemas de ingeniería.
4. Ciencias y matemáticas.
5. Gráficos con MATLAB.

MATLAB puede considerarse un lenguaje de programación, como Fortran o C, aunque sería difícil describirlo en unas cuantas palabras. He aquí algunas de sus características notables:

1. La programación se logra de manera sencilla.
2. Se manejan fácilmente valores enteros, reales y complejos.
3. Hay una mayor exactitud numérica en los cálculos.
4. Cuenta con una biblioteca matemática amplia.
5. Posee abundantes herramientas gráficas, incluidas funciones de interfaz gráfica con el usuario.
6. Capacidad de vincularse con los lenguajes de programación tradicionales.
7. Transportabilidad de los programas MATLAB.

Una característica extraordinaria de los números en MATLAB es que no hay distinción entre reales, complejos, enteros, de precisión sencilla y de doble precisión. En MATLAB, todos estos números están conectados continuamente. Esto significa que en MATLAB cualquier variable puede contener números de cualquier tipo, sin una declaración especial durante la programación, con lo cual ésta última se hace más rápida y productiva. La biblioteca matemática de MATLAB facilita los análisis matemáticos. Además, el usuario puede crear rutinas matemáticas adicionales con mayor facilidad usualmente que en otros lenguajes de programación, gracias a la continuidad entre las variables reales y complejas. Entre las numerosas funciones matemáticas, los solucionadores de álgebra lineal desempeñan un papel crucial; de hecho, todo el sistema MATLAB se basa en estos solucionadores.

Con respecto a la importancia de las gráficas, el análisis visual de los problemas matemáticos ayuda a comprenderlos y a hacerlos más accesibles. Aunque esta ventaja es bien conocida, la presentación de resultados calculados con gráficos de computadora solía requerir un esfuerzo adicional considerable. Con MATLAB, en cambio, bastan unos cuantos comandos para producir presentaciones gráficas del material matemático. Es posible crear gráficos científicos e incluso artísticos en la pantalla mediante expresiones matemáticas. Se ha comprobado que las gráficas de MATLAB motivan e incluso excitan a los estudiantes para aprender métodos matemáticos

y numéricos que de otra forma podrían resultar tediosos. Las gráficas de MATLAB son sencillas y resultan prácticas para los lectores.

El proyecto de estas tesis está orientado a la realización de los objetivos siguientes:

1. Desarrollar el algoritmo de Remez con programas computacionales amigables.
2. Implementar una aplicación software para el cálculo de una buena aproximación polinomial con base probabilística.

La utilización simultánea de ambos programas, permitirá la comparación de resultados.

El trabajo de las tesis consta de una parte de elaboración colectiva y otras de elaboración individual. En la parte colectiva, una presentación escrita conjunta de los conceptos básicos comunes, donde se prepara un programa integrado conformado por los dos algoritmos, el clásico de Remez y el probabilístico.

Las investigaciones de estos trabajos de tesis están estructuradas en tres capítulos:

En el primero se estudia de manera resumida y conjunta los conceptos básicos tales como; métricas, normas sobre espacios vectoriales, aproximación, mejor aproximación uniforme y el teorema de alternancia de Chebyschev. Igualmente algunos ejemplos de mejor aproximación, primordiales para los temas posteriores. Aunque muchos de estos conceptos los podemos encontrar en libros de texto, aquí se presentan con un enfoque dirigido hacia nuestras investigaciones, dando al lector un material relativamente autocontenido.

En el segundo capítulo de una de las tesis se explica detalladamente la implementación del algoritmo de Remez. En la otra, se explican los fundamentos del método probabilístico.

En el tercer capítulo se expondrá el análisis y diseño de la implementación computacional, con detalles de la GUI[1] y del lenguaje MATLAB. Se

[1]Interfaz Gráfica de Usuario.

presentará también como un apéndice, una guía de usuario dando la explicación del manejo del diálogo activo.

En cada una de las tesis, se hará énfasis en el algoritmo correspondiente al trabajo individual del tesista.

Resumiendo, se presentan dos tesis con una base teórica común y como producto un programa integrado. Un tesista desarrolla el algoritmo de Remez y el otro uno probabilístico, después ambos se integran.

Capítulo 1

Conceptos Básicos

En estos trabajos de tesis se desarrollan programas de computación para obtener polinomios algebraicos que aproximen a una función continua definida en un intervalo. Para describir los métodos aproximativos necesitamos de algunos conceptos básicos del Análisis Matemático tales como espacio métrico, espacio normado y definiciones fundamentales de Teoría de Aproximación. Por tal motivo en este primer capítulo mencionaremos de manera resumida algunos conceptos que son básicos para el entendimiento de los algoritmos que en éstas tesis se estudian. Todo este material se puede ver en una basta bibliografía que existe y que se puede consultar, como por ejemplo en [11], [12], [5] y [7].

Aunque la notación que se utiliza en este texto es bastante común, se hará una revisión para evitar posibles confusiones futuras. Así, el conjunto de los números naturales y reales, se denotaran por $\mathbb{N}$ y $\mathbb{R}$ respectivamente.

El espacio donde desarrollamos la gran mayoría de estos trabajos de tesis es el espacio de todas las funciones reales continuas definidas en un intervalo cerrado [a,b] que denotamos C[a,b]. Los elementos de C[a,b] serán aproximados mediante polinomios algebraicos por lo que el subespacio de todos los polinomios algebraicos de orden menor o igual a n lo denotamos por P_n.

1.1. Espacios métricos

El sistema de números reales tiene dos tipos de propiedades. El primer tipo consiste del algebraico, tratando con la suma y la multiplicación. El otro tipo consiste en propiedades que tienen que ver con la noción de distancia entre dos números y con el concepto de un límite. Estas últimas propiedades se llaman métricas o topológicas.

Definición 1.1.1 *Un espacio métrico es un par (X, d), donde X es un conjunto y d es una métrica sobre X, esto es, una función definida sobre $X \times X$ tal que para todo $x, y, z \in X$ tenemos:*

1. *d es de valor real finita y no negativa.*
2. $d(x, y) = 0 \Leftrightarrow x = y$.
3. $d(x, y) = d(y, x)$.
4. $d(x, y) \leq d(x, z) + d(z, y)$.

Definición 1.1.2 *Decimos que $(Y, \bar{d})$ es un subespacio de (X, d) si $Y \subset X$ y d es la restricción de d en $Y \times Y$. La métrica d es llamada la métrica inducida en Y por d.*

Ejemplos

1. **Recta real** $\mathbb{R}$. En el conjunto de los números reales $\mathbb{R}$ se define una distancia de la forma siguiente. Para cualesquiera números reales x, y definimos:
$$d(x, y) = |x - y|$$
2. **Plano Euclidiano** $\mathbb{R}^2$. El espacio $\mathbb{R}^2$, está formado por el conjunto de todos los pares ordenados de números reales. Si $x = (\xi_1, \xi_2)$ y $y = (\eta_1, \eta_2)$, son dos pares ordenados
$$d(x, y) = \sqrt{(\xi_1 - \eta_1)^2 + (\xi_2 - \eta_2)^2},$$
con ésta métrica $\mathbb{R}^2$ es un espacio métrico, llamado plano Euclidiano.

3. **Espacio Euclidiano $\mathbb{R}^n$**. Los ejemplos anteriores son casos particulares del espacio Euclidiano n-dimensional $\mathbb{R}^n$. Este espacio se obtiene si nosotros tomamos el conjunto de todas las n-uplas de números reales, escrito:

$$x = (\xi_1, \ldots, \xi_n) \qquad \text{y} \qquad y = (\eta_1, \ldots, \eta_n)$$

y la métrica Euclidiana se define por:

$$d(x, y) = \sqrt{(\xi_1 - \eta_1)^2 + \ldots + (\xi_n - \eta_2)^2}.$$

4. **Espacio de funciones C[a,b]**. Sea X el conjunto de todas las funciones de valores reales de una variable independiente t definidas y continuas sobre un intervalo cerrado $J = [a, b]$. Escogemos la métrica definida por

$$d(x, y) = \max_{t \in J} |x(t) - y(t)|,$$

donde máx denota el máximo sobre J. Así obtenemos un espacio métrico que es denotado por C[a,b].

1.2. Espacios vectoriales

Los espacios vectoriales juegan un rol importante en muchas ramas de las matemáticas y sus aplicaciones. En efecto, en varios problemas prácticos (y teóricos) tenemos un conjunto X cuyos elementos pudiesen ser vectores en el espacio de tres dimensiones o sucesiones de números o funciones, y esos elementos pueden ser sumados y multiplicados por constantes (números) de una forma natural, y los resultados nuevamente son elementos de X. Tal situación concreta sugiere el concepto de un espacio vectorial tal como lo definiremos. La definición envuelve un campo general K, pero nosotros nos interesamos en el campo de los números reales. Los elementos de K se llaman escalares, por lo cual en el Análisis Funcional a los números reales o complejos se les llama escalares.

Definición 1.2.1 *Un espacio vectorial V sobre $\mathbb{R}$ es un conjunto no vacío de elementos llamados vectores, con dos operaciones, una interna llamada suma $(+)$ de vectores y otra externa denominada producto $(\cdot)$ de vectores por elementos del campo llamados escalares, y tal que para cualesquiera $a, b, c \in V$ y $\alpha, \beta \in \mathbb{R}$*

1. $a + b \in V$
2. $a + b = b + a$
3. $(a + b) + c = a + (b + c)$
4. *Existe un vector* $\bar{0}$ *(llamado vector nulo) en* V *tal que* $a + \bar{0} = a$
5. *Para cada* $a \in V$*, existe un vector* a' *tal que* $a + a' = \bar{0}$
6. $\alpha \cdot a \in V$
7. $\alpha \cdot (\beta \cdot a) = (\alpha\beta) \cdot a$
8. $1 \cdot a = a$
9. $\alpha \cdot (a + b) = (\alpha \cdot a) + (\alpha \cdot b)$
10. $(\alpha + \beta) \cdot a = (\alpha \cdot a) + (\beta \cdot a)$

Como K es el conjunto de números reales, entonces el espacio vectorial se llama espacio vectorial real.

Ejemplos

1. **Espacio** $\mathbb{R}^n$. Es el espacio Euclidiano de las operaciones definidas como sigue: si $x = (\xi_1, \ldots, \xi_n)$, $y = (\eta_1, \ldots, \eta_n)$ y $\alpha \in \mathbb{R}$, entonces

$$x + y = (\xi_1 + \eta_1, \ldots, \xi_n + \eta_n) \qquad \text{y} \qquad \alpha x = (\alpha\xi_1, \ldots, \alpha\xi_n)$$

2. **Espacio C[a,b]**. Un punto de este espacio es una función continua que toma valores reales sobre el intervalo [a,b]. El conjunto de todas estas funciones forman un espacio vectorial real con las operaciones algebraicas definidas como:

$$(x + y)(t) = x(t) + y(t)$$
$$(\alpha x)(t) = \alpha x(t)$$

es decir; $(x + y)$ y (αx) son funciones continuas con valores reales definida sobre [a,b] si x y y son funciones y α es real.

1.3. Espacios normados

Definición 1.3.1 *Una norma sobre un espacio vectorial real X es una función de X en $\mathbb{R}$ que se denota por*

$$x \to \mathbb{R}$$

la cual tiene las propiedades siguientes:

1. $\|x\| \geq 0$
2. $\|x\| = 0 \Leftrightarrow x = 0$
3. $\|\alpha x\| = |\alpha|\|x\|$
4. $\|x + y\| \leq \|x\| + \|y\|$ *(desigualdad del triángulo)*

donde x y y son vectores arbitrarios en $X \in \mathbb{R}$.

Una norma sobre X define una métrica d en X y está dada por:

$$d(x, y) = \|x - y\|, \text{ donde } x, y \in X$$

y es llamada la métrica inducida por la norma. El espacio normado justamente definido es denotado por $(X, \|\cdot\|)$ o simplemente por X.

Ejemplos

1. **Espacio Euclidiano** $\mathbb{R}^n$. Los espacios vectoriales $\mathbb{R}^n$ son espacios normados con la norma definida por

$$\|x\| = \left(\sum_{j=1}^{n} |\xi_j|^2 \right)^{1/2} = \sqrt{|\xi_1|^2 + \ldots + |\xi_n|^2}.$$

   Esta norma produce la métrica

$$d(x, y) = \|x - y\| = \sqrt{|\xi_1 - \eta_1|^2 + \ldots + |\xi_n - \eta_n|^2}.$$

2. **Espacio C[a,b]**. Este espacio vectorial es un espacio normado con la norma dada por

$$\|x\| = \max_{t \in J} |x(t)|$$

   donde $J = [a, b]$.

1.4. Teoría de Aproximación

Sea $U \subset X$, un método de aproximación y que es un mapeo de X en U. En otras palabras, dado un elemento $f \in X$, el elemento u_f de U es en algún sentido una aproximación a f.

Definición 1.4.1 *Sea X un espacio métrico, $U \subset X$ dado, con $U \neq 0$. Una mejor aproximación a $f \in X$ por elementos de U, es un elemento $u^* \in U$ tal que*

$$d(f, u^*) = \text{ínf}\{d(f, u) : u \in U\} := \text{dist}(f, U).$$

Con respecto al elemento de mejor aproximación, existen cuatro cuestiones básicas como son: (i) existencia, (ii) caracterización (iii) unicidad y (iv) cálculo. En el caso de aproximación uniforme por polinomios algebraicos, siempre existe el polinomio de mejor aproximación, es único y se caracteriza por el llamado Teorema de alternancia de Chebyshev.

Otra cuestión interesante es encontrar un "buen" método que escoja u_f tal que $d(f, u_f)$ esté cerca a $\text{dist}(f, U)$.

Usualmente en Teoría de Aproximación, U es un subespacio lineal de un espacio lineal normado X y $d(f, g) = \|f - g\|$.

1.4.1. Aproximación lineal y proyección

Definición 1.4.1.1 *Se dice que la función M de X en U es un método de aproximación lineal si se cumplen las condiciones siguientes:*

1. *X es un espacio vectorial y U es un subespacio vectorial de X, y*
2. *la función M tiene la propiedad que para cualesquiera f y $g \in X$ y cualesquiera escalares α y β se tiene*

$$M(\alpha f + \beta g) = \alpha M(f) + \beta M(g).$$

Un método de aproximación lineal M se dice que es una proyección si para todo $u \in U$, se cumple

$$M(u) = u.$$

Ejemplo 1

Sea $X = C[a,b]$ y $U = P_n$, el espacio vectorial de funciones continuas con valores reales y definidas en el intervalo cerrado $[a,b]$ y el subespacio de polinomios de grado menor o igual a n, respectivamente. Veamos el método siguiente de aproximación. Consideremos los puntos x_i, $i = 0, 1, \ldots, n$, tales que $a \leq x_0 < x_1 < \ldots < x_n \leq b$, y dada cualquier $f \in C[a,b]$, entonces tomamos $p = M(f)$ como el polinomio de P_n que satisface las condiciones

$$p(x_i) = f(x_i), i = 0, 1, 2, \ldots, n.$$

Así M es un método de aproximación lineal que también es una proyección, la solución del problema de interpolación es única y puede ser expresada en la forma de polinomios de Lagrange, a saber,

$$p(x) = \sum_{i=0}^{n} \ell_i(x)\, f(x_i), \qquad x \in [a,b],$$

donde $\ell_i,\ i = 0, 1, \ldots, n$, es el polinomio

$$\ell_i(x) = \Pi_{j=0, j\neq i}^{n} \left\{ (x - x_j)/(x_i - x_j) \right\},$$

tal que $\ell_i(x_j) = \delta_{ij}$.

Definición 1.4.1.2 *Sea X un espacio lineal normado. Entonces, si M es una función lineal de X en U, su norma se define como el número*

$$\|M\| := \sup \{ \|M(f)\| : f \in X, \|f\| = 1 \}.$$

Entonces para cada $f \in X$ tenemos $\|M(f)\| \leq \|M\| \|f\|$.

Ejemplo 2

Una norma para una función del ejemplo 1 es la siguiente:

$$\|f\|_\infty := \sup_{x \in [a,b]} |f(x)|,$$

entonces la norma de la proyección M está dada por

$$\|M\| := \sup\left\{\left|\sum_{i=0}^{n} \ell_i(x)\ f(x_i)\right| : x \in [a,b], \|f\|_\infty = 1\right\}$$

Si para cualquier x y para cualquier f se satisface que $\|f\|_\infty = 1$ y $\ell_i(x)\ f(x_i) = |\ell_i(x)|,\ i = 0, 1, \ldots, n$. Entonces

$$\|M\| = \sup\left\{\sum_{i=0}^{n} |\ell_i(x)| : x \in [a,b]\right\}.$$

1.4.2. Grado de aproximación

Considere una sucesión creciente (U_n) de subconjuntos de X. El grado de aproximación considera el comportamiento de $E_n(f) := \text{dist}(f, U_n)$ como una función del parámetro n con $n \to \infty$. La primera cuestión es si

$$E_n(f) \to 0, (n \to \infty).$$

La segunda cuestión interesante es si la convergencia es rápida o lenta para un f en particular. Para esto $E_n(f)$ se compara con ciertas sucesiones estándares, como por ejemplo $(n^{1/n})$.

1.4.3. Existencia de la mejor aproximación polinomial

Teorema 1.4.3.1 *Sea U un subespacio de dimensión finita de un espacio lineal normado X. Entonces, para todo $f \in X$, existe un elemento $u^* \in U$ de mejor aproximación.*

La demostración para éste y otros teoremas posteriores se pueden consultar en [3] y [10].

Con este teorema se garantiza la existencia de la mejor aproximación cuando $X = C[a,b]$ y $U = P_n[a,b]$ es el espacio de los polinomios algebraicos de grado a lo más n.

1.4.4. Teorema de alternancia de Chebyshev

Teorema 1.4.4.1 *Un polinomio $p^* \in P_n$ es una mejor aproximación a $f \in C[a,b]$ si y solo si existen $(n+2)$ puntos $a \leq x_1 < \ldots < x_{n+1} \leq b$ tales que*

$$f(x_i) - p^*(x_i) = (-1)^i \gamma \qquad |\gamma| = \|f - p^*\|$$

esto es, si y solo si la diferencia $f(x) - p^*(x)$ toma consecutivamente sus valores máximo y mínimo con signos alternados en al menos $(n+2)$ veces.

Esto es un resultado de caracterización y se puede verificar en [3].

Para los algoritmos que se desarrollan en estas tesis, es importante destacar que este resultado vale si $[a,b]$ se sustituye por un conjunto compacto de $\mathbb{R}$ que tenga al menos $(n+2)$ puntos.

1.4.5. Convergencia

Teorema 1.4.5.1 *Sigue del teorema clásico de Weierstrass, que $P_n[a,b]$ el conjunto de todos los polinomios algebraicos $\cup_n P_n$ es denso en $C[a,b]$. En particular, la sucesión de los polinomios de mejor aproximación a $f \in C[a,b]$, convergen uniformemente a f.*

1.4.6. Unicidad

Teorema 1.4.6.1 *Como P_n es de dimensión finita, entonces para toda función continua existe un polinomio de grado n de la mejor aproximación a f. En el caso de la mejor aproximación polinomial de funciones continuas, se demuestra la unicidad del polinomio de mejor aproximación con el uso del teorema de alternancia.*

Capítulo 2

Cálculo del polinomio de la mejor aproximación

El algoritmo de Remez consiste en una serie de iteraciones para modificar una tabla inicialmente dada de puntos T, de manera que los polinomios P_T que mejor aproximan la función objeto de estudio sobre T, converjan al polinomio P^* de la mejor aproximación global. Para realizar estas iteraciones se necesita calcular extremos, lo cual puede ser engorroso cuando las funciones que aproximamos sean complicadas desde el punto de vista computacional.

Para paliar esta situación, lo que hacemos es generar aleatoriamente tablas de puntos T e ir seleccionando aquellos polinomios P_T cuyos errores de aproximación $|d_T|$ sobre T sean más grandes. En efecto, sobre una tabla de $(n+2)$ puntos contenidos sobre el intervalo $[a, b]$, el error d_T siempre será menor o igual al error global d que se alcanza justamente para una tabla de $(n+2)$ puntos, de la cual solo se conoce su existencia de manera teórica. Resumiendo, la generación aleatoria de tablas y la selección de los mejores errores bajo el criterio de selección establecido, nos conduce heurísticamente, al crecer el número de tablas generadas, a una tabla cercana a la tabla óptima. Aunque las posibilidades sean infinitas, la continuidad uniforme de las funciones continuas y los polinomios sobre los intervalos compactos y la aritmética finita de las computadoras, deben de garantizar nuestra expectativa.

2.1. Algoritmo probabilístico

Este algoritmo está inspirado en el algoritmo de Remez, pero utiliza métodos probabilísticos, que pueden resultar deseables cuando las funciones sean difíciles de tratar desde el punto de vista computacional.

El algoritmo inicia con la selección de una tabla de $(n+2)$ puntos $T = \{t_0, t_1, \ldots, t_{n+1}\}$, sobre el intervalo cerrado $[a, b]$; para inmediatamente resolver el sistema de ecuaciones lineales descrito en la ecuación 2.1, y así obtener un número real d_T y un polinomio P_T que mejor aproxima a la función sobre T.

$$\sum_{j=0}^{n} a_j x_i^j + (-1)^i d = f(x_i); \qquad i = 0, 1, \ldots, n+1 \tag{2.1}$$

Antes de continuar iterando, el algoritmo debe realizar la comparación siguiente: se toma en consideración el valor d_T, error absoluto sobre T de la diferencia entre el polinomio P_T de la iteración actual y la función f sobre T; y el valor del error d almacenado. Si

$$d_T \geq d$$

se guarda la información y se intercambia el d por d_T, que es el mejor encontrado hasta el momento. Por otro lado, si es menor, hacer caso omiso y dejar los valores que estaban guardados.

En cualquiera de los dos casos, se continúa con el plan de iteraciones. Para una mejor comprensión ver la figura 2.1.

Como se puede ver en el diagrama de flujo, el algoritmo es simple, puesto que sólo está formado por la rutina *Resolver el sistema de ecuaciones* y la condición que es la responsable de cumplir el objetivo del algoritmo.

2.1.1. Argumentos del algoritmo

En esta sección se explicarán los argumentos requeridos para la ejecución del algoritmo probabilístico; es importante mencionar que estos se deben entender completamente, ya que un mal ingreso de los argumentos podría influir en un resultado no esperado del algoritmo.

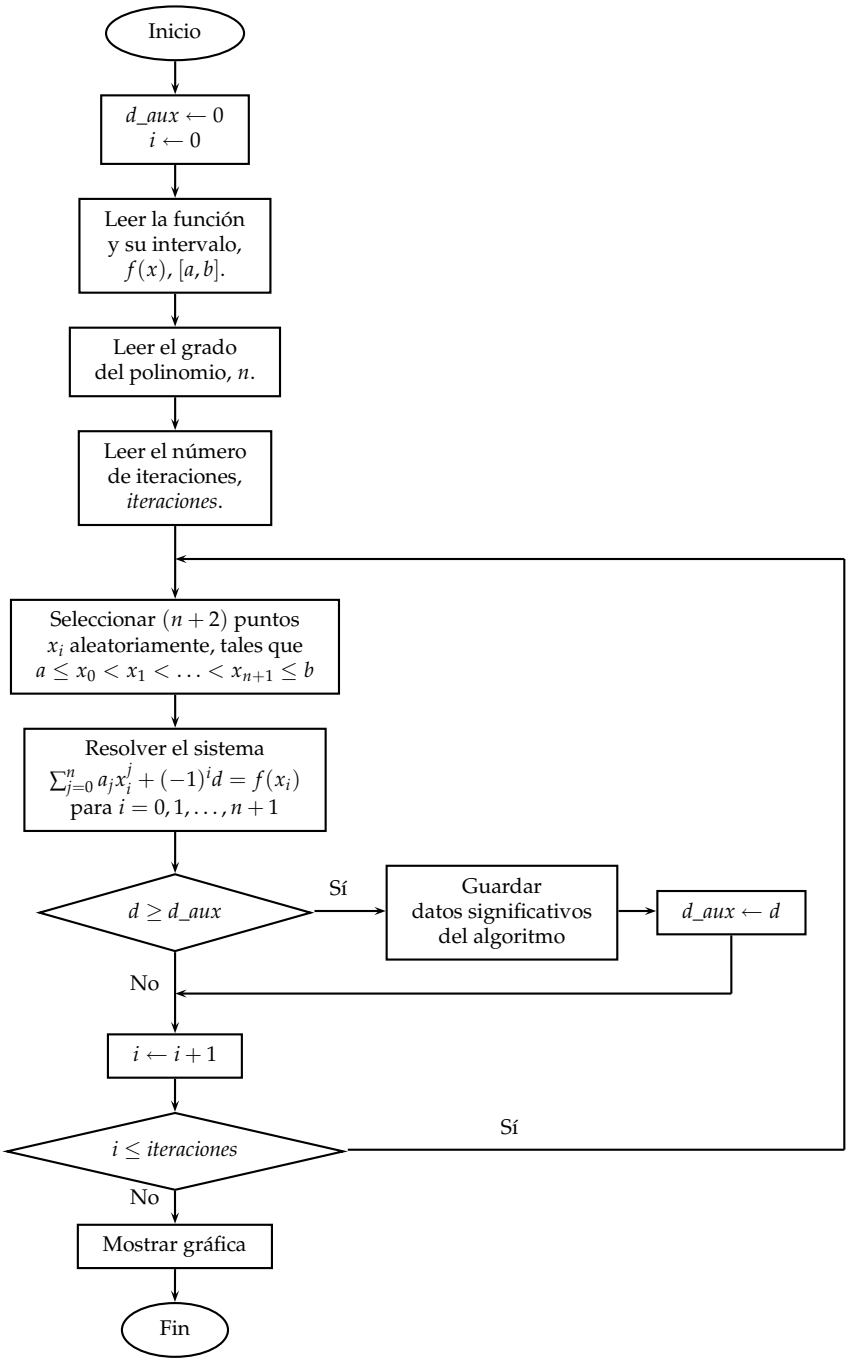


Figura 2.1: Diagrama de flujo del algoritmo probabilístico.

Función. Función o trozos de la función; es importante mencionar que el ingreso de este parámetro debe ser escrito como lo requiera el lenguaje de programación en que está desarrollada la aplicación del algoritmo probabilístico.

Intervalo. Este parámetro puede ser individual o general, según sea el caso del tipo de función a ingresar (elemental o definida por partes). Este(os) intervalo(s) siempre será(n) tomado(s) como cerrado(s), aún cuando la función sea definida por partes.

Grado del polinomio. El algoritmo de Remez requiere $(n+2)$ puntos para ejecutarse, y como este algoritmo probabilístico está fundamentado en el algoritmo de Remez, igualmente se necesitan $(n+2)$ puntos. Este argumento actúa como la variable n, es decir, si el grado es 3, entonces 5 puntos requiere el algoritmo para su ejecución. Como observación se puede mencionar que un grado 1 generará una línea recta.

Iteraciones. Es el número total de repeticiones solicitadas por el usuario para encontrar el mejor polinomio de la mejor aproximación.

2.1.2. Observaciones

En el desarrollo, y posteriormente en las pruebas realizadas del algoritmo probabilístico, se observaron algunos patrones en el comportamiento de los resultados y por ser muy importantes, no se pueden omitir en esta redacción.

1. Un número enorme de iteraciones no garantiza que se obtendrá una buena exactitud para el mejor polinomio de la mejor aproximación.
2. El error general del algoritmo disminuye cuando el número de iteraciones aumenta.
3. El mejor polinomio encontrado podría no estar muy cerca de la función, pero es la mejor aproximación en los $(n+2)$ puntos de T.

Capítulo 3

Diseño e implementación del dialogo activo

El objetivo de esta tesis es el desarrollo de un dialogo activo para la implementación de un algoritmo probabilístico para funciones difíciles de tratar computacionalmente. En cuanto a operaciones matemáticas se refiere, ésta idea principal del algoritmo está fundamentada en métodos probabilísticos.

En este capítulo, se mencionarán las etapas para la gestión del dialogo activo. Analizaremos los requerimientos y las funcionalidades que se desean implementar en el diálogo. Diseñaremos la mejor configuración para cubrir totalmente las necesidades del *usuario final*[1] y nos responderemos la pregunta de ¿cómo desarrollar el dialogo activo? Igualmente analizaremos algunos lenguajes de programación para la implementación computacional del algoritmo probabilístico y seleccionaremos el lenguaje apropiado para cubrir todos los requerimientos expresados por el usuario final.

3.1. Definición de aplicación de computadora

Una aplicación es un programa de computadora diseñado para facilitar al usuario la realización de un determinado tipo de trabajo. Posee ciertas ca-

[1]Persona o personas que van a manipular de manera directa un producto software.

racterísticas que lo diferencian de un *sistema operativo*[2], de una utilidad informática (que realiza tareas de mantenimiento o de uso general) y de un lenguaje de programación (con el cual se crean los programas para computadora). Suele resultar una solución computacional para la automatización de ciertas tareas complicadas como pueden ser la contabilidad o la gestión de un almacén [8].

El motivo por el cual se ha mencionado la definición de aplicación de computadora, es que de aquí en adelante se puede tomar de igual manera dialogo activo y aplicación. Los dos casos nos llevarán a la construcción del dialogo activo.

3.2. Definición de una metodología

La ISO[3], en su norma 12207 define al ciclo de vida de un software como un marco de referencia que contiene las actividades y las tareas involucradas en el desarrollo, la explotación y el mantenimiento de un producto software, abarcando desde la definición hasta la finalización de su uso. Las etapas que se han establecido en un modelo de ciclo de vida son las siguientes: expresión de necesidades, especificaciones, análisis, diseño, implementación, debugging, validación y evolución [4].

En esta tesis, el producto software será el dialogo activo y como tal, no requiere de todas las etapas que envuelven al ciclo de vida de un software. Esta conclusión es justificable debido a que se implementará un algoritmo, mas no se desarrollará un sistema de información en el que se deben cubrir todas las características y necesidades de la Ingeniería de Software.

En la etapa de diseño nos apoyaremos de algunas herramientas de UML[4] para detallar el funcionamiento general de la aplicación, tal es el caso de los diagramas de estados que proveen de manera gráfica los distintos eventos y estados del transcurso del sistema.

A futuro, si este trabajo de tesis tiene continuidad, se podría implementar en versiones posteriores de esta aplicación, un análisis y diseño más pro-

[2]Conjunto de programas de computadora, destinado a permitir una administración eficaz de los recursos de la computadora.

[3]International Organization for Standardization.

[4]Lenguaje Unificado de Modelado.

fundo y profesional. Por el momento, el trabajo de Ingeniería de Software presentado en esta tesis es suficiente. En esta referencia se puede justificar lo antes mencionado [13].

3.3. Catálogo de requerimientos

De acuerdo a la evaluación de todas las necesidades que se requieren para el usuario final, se generó un catálogo de requerimientos, con el fin de obtener la mejor interacción posible entre el usuario y la aplicación. Y de esta forma, diseñar una aplicación amigable al usuario final. Este catálogo se detalla a continuación.

- El argumento *función* está definido en dos tipos: *elemental* y *por trozos*. Cada definición requiere de controles propios para almacenar la función o trozos de función. Igualmente, se desea previsualizar la función ingresada (con trozos, si es necesario) antes de la ejecución del algoritmo.
- El ingreso de cada función o trozo de función se desea hacerlo manualmente o a través de un archivo con sintaxis propia.
- El cálculo del mejor polinomio de la mejor aproximación será una representación gráfica que incluya a la función ingresada, el polinomio calculado y la función error. Además de representar cada función con un color distinto, de esta forma se tendrá una mejor comprensión para el usuario.
- Considerando el grupo de usuarios al que está dirigido esta aplicación, se requiere incluir al inicio de la misma, una sugerencia de ayuda, en la cual se explicaría brevemente el funcionamiento de la aplicación; incluyendo además, la posibilidad de deshabilitar esta característica. Asimismo, se requiere de construir un instalador para facilitar la ejecución de la aplicación, de tal modo que se pueda compartir la aplicación en diferentes sitios de Internet para su descarga e instalación. Posteriormente hablaremos de los requisitos de software necesarios para difundir este dialogo activo.

3.3.1. Requisitos en contexto a dos columnas

Para analizar con mayor detalle los requerimientos del usuario final, se usará la técnica llamada *Escritura de requisitos en contexto a dos columnas* para definir estos requisitos. Ya que destaca el hecho de que se establece una interacción entre el actor y el sistema. Además de que el número de procedimientos son mínimos en esta aplicación.

Antes de continuar, haremos mención de algunas definiciones relacionadas con los casos de uso.

Actor. Es algo con comportamiento, como una persona (identificada por un rol), sistema de información u organización; por ejemplo, un cajero.

Escenario. Es una secuencia específica de acciones e interacciones entre los actores y el sistema objeto de estudio.

Caso de uso. Es una colección de escenarios con éxito y fallo relacionados, que describe a los actores utilizando un sistema para satisfacer un objetivo.

El funcionamiento general de la aplicación precisa de validar los argumentos del algoritmo antes de ser enviados a este mismo para su ejecución. Estas validaciones consisten en verificar que un número sea mayor a cero; además de verificar campos vacíos omitidos por un descuido del usuario.

En la figura 3.1 se muestra el caso de uso para el funcionamiento general del dialogo activo.

3.4. Diseño del dialogo activo

En el análisis para la implementación del algoritmo, se consideraron algunos lenguajes de programación que se fomentaron en el transcurso de la preparación universitaria, como por ejemplo Java; pero nos encontramos con la problemática principal de la ilustración de gráficas altamente detalladas para la percepción del usuario, y Java no incluía en sus bibliotecas funciones nativas para construir fácilmente este tipo de gráficas.

Actor principal: Usuario.
Personal involucrado e intereses:
Usuario: quiere calcular el mejor polinomio de la mejor aproximación.
Precondiciones: Ninguna.
Escenario principal de éxito:

Acción del actor (o intención)	Responsabilidades del sistema
1. El usuario ingresa una función $f(x)$.	
2. El usuario ingresa el intervalo $[a, b]$ de la función $f(x)$.	
3. El usuario quiere previsualizar la función ingresada sobre el intervalo	
	4. Verifica que los campos no estén vacíos.
	5. El sistema previsualiza la gráfica para $f(x)$ sobre $[a, b]$.
6. El usuario ingresa el grado del polinomio.	
7. El usuario ingresa la tolerancia del error.	
8. El usuario ingresa el número de iteraciones.	
9. El usuario ejecuta el algoritmo.	
	10. Verifica que los campos no estén vacíos.
	11. Verifica que el grado y el número de iteraciones sean mayores a cero.
	12. El sistema muestra la gráfica resultante.
13. Cierra la aplicación.	

Figura 3.1: Caso de uso para calcular el mejor polinomio de la mejor aproximación.

En la búsqueda de la solución a esta problemática, encontramos paquetes gratuitos ya desarrollados que solucionaban nuestro problema, como por ejemplo jFreeChart[5], pero la documentación es comercial y se tiene que pagar por ella. En otros casos, la documentación era escasa y no había una comprensión absoluta de estos paquetes. Después, en distintos sitios de Internet se mencionaba mucho el lenguaje MATLAB, ya que éste está diseñado para la resolución de problemas matemáticos y la elaboración de gráficas accesibles a los usuarios. Esto es debido a que MATLAB permite fácilmente la manipulación de matrices [6]; de ahí el origen de su nombre: "Matrix

[5]http://www.jfree.org/jfreechart/

Laboratory".

Además, MATLAB es un ambiente para el Cálculo Numérico y lenguaje de programación al mismo tiempo. Tiene una basta biblioteca de funciones matemáticas y es extraordinario para manejar números y no distingue entre reales, complejos, enteros y de alta presición. Porque todos estos tipos de números los pueden guardar cualquier variable sin una declaración especial durante la programación.

Como es de esperarse, todo lenguaje de programación tiene limitaciones en su uso y MATLAB, a pesar de sus fascinantes capacidades presenta algunos inconvenientes para la implementación de este algoritmo probabilístico.

MATLAB está diseñado para resolver problemas matemáticos y manejar números de cualquier tipo, pero ésta ventaja nos hace ver una debilidad del lenguaje MATLAB: el diseño y programación de interfaces gráficas de usuario.

El diseño de las interfaces gráficas en MATLAB es muy complicado y poco personalizable [9], debido a que MATLAB todo lo maneja por medio de matrices y vectores. En comparación con otros lenguajes, MATLAB ofrece muy pocos métodos para el diseño de interfaces de usuario que limitan la personalización y el buen funcionamiento de la aplicación que se quiere desarrollar. Por ejemplo, Java ofrece un sin fin de métodos para cada elemento de una GUI, además ofrece la posibilidad de desarrollar métodos propios para cada elemento o mejor aún, la creación de clases con las cuales se puedan generar instancias de ellas para una personalización completa a cada elemento de una GUI de la aplicación a desarrollar.

Finalmente, la evaluación de todos los factores que envolvían a MATLAB y otros lenguajes de programación, nos hizo seleccionar a MATLAB como lenguaje de programación para la implementación del algoritmo probabilístico. Las tantas características que MATLAB ofrecía con respecto al uso de las gráficas fue una razón absoluta que aminoraban las limitaciones de este lenguaje de programación.

3.4.1. ¿Cómo desarrollaremos el catálogo de requerimientos?

Hasta este punto, se ha recopilado un catálogo de requerimientos para el usuario final, los cuales deben considerarse para el diseño de la aplicación; y también se ha seleccionado un lenguaje de programación para la implementación del algoritmo probabilístico.

El catálogo de requerimientos define ¿qué necesita el usuario? Y nuestro siguiente paso es definir ¿cómo se desarrollarán estas necesidades?

Primeramente, se debe construir la aplicación en base a funciones independientes, de tal forma que puedan ser utilizadas por otras aplicaciones distintas a este algoritmo. Después, tenemos que considerar a cada argumento del algoritmo como un tipo de variable distinto, para poder manejarlo de una forma diferente. Por ejemplo, la variable función requiere definirse y manejarse de tal forma que el texto ingresado se pueda manipular como una función matemática y no como un número.

El diseño de la aplicación debe basarse a las reglas sintácticas y semánticas de MATLAB.

Función

Este argumento debe manipularse de tal forma, que el usuario al ingresar una función de una variable, MATLAB la interprete como una función matemática. MATLAB en sus múltiples herramientas, provee un *toolbox*[6] llamado *Symbolic Math Toolbox*, el cual tiene la capacidad de manejar variables como si fueran funciones [2]. A continuación se muestra un ejemplo de este tipo de variables.

```
x = sym('x');
f = x^2;
```

De esta forma, MATLAB evaluará la variable f como una función $f(x)$, y devolverá el cálculo de la evaluación de la función con el valor actual de x.

```
x = 2;
```

[6] Ventana o panel de iconos de herramientas.

```
eval(f)

ans = 4
```

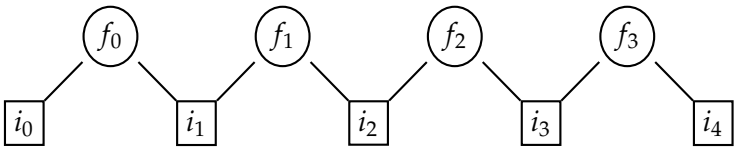
Intervalo

Cada función tiene asociado un intervalo, $f(x);\ x \in [a,b]$; en esta aplicación una función puede estar definida por trozos en un intervalo, por ejemplo, la función siguiente está definida en el intervalo $[0,4]$ como sigue

$$f(x) = \begin{cases} 1-x & \text{si } 0 \leq x < 1 \\ x-1 & \text{si } 1 \leq x < 2 \\ 3-x & \text{si } 2 \leq x < 3 \\ x-3 & \text{si } 3 \leq x \leq 4 \end{cases}$$

Para solucionar la definición por partes, nos apoyamos en vectores para guardar los intervalos de los trozos.

De esta manera, para una función definida en cuatro trozos, el vector de funciones queda definido de esta forma, $F = \{f_0, f_1, f_2, f_3\}$; y el vector de los puntos extremos de los intervalos queda definido $I = \{i_0, i_1, \ldots, i_4\}$. Y como consecuencia, f_0 comparte con f_1 el punto i_1. La figura siguiente ilustra con mayor detalle cómo comparten los intervalos los trozos de función.



Funcionamiento general del dialogo activo

Después de haber explicado algunos detalles en el diseño de la programación, pasamos a mostrar gráficamente el funcionamiento general de la aplicación. Nos apoyaremos de los diagramas de estados de UML.

Este tipo de diagramas tienen la finalidad de representar el ciclo de vida de un objeto; qué experimenta, sus transiciones y los estados en los que se encuentra entre estos eventos.

Las figuras 3.2 y 3.3 muestran los diagramas de estados para el ingreso de trozos y funcionamiento general de la aplicación.

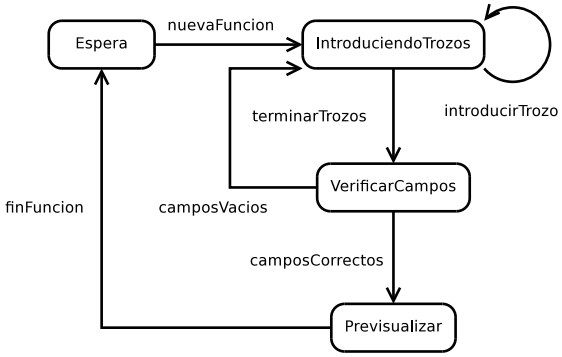


Figura 3.2: Diagrama de estados para el ingreso de los trozos de una función.

El ingreso de los trozos de función inicia en estado de *Espera* hasta que el usuario requiera comenzar a *introducir los trozos* de la función. Cuando termine con este ingreso, la aplicación verificará si existen *campos vacíos*; si existen, la aplicación le advertirá al usuario de este suceso para revisar los campos. En caso de que no existen campos vacíos, la aplicación *Previsualizará* la función.

El cálculo del mejor polinomio de la mejor aproximación inicia en estado de *Espera* hasta que el usuario requiera un *nuevo cálculo*. Después, el usuario *ingresa los datos* del algoritmo y *los trozos de la función*. Posteriormente, la aplicación *valida estos datos* (que fueron ingresados correctamente); si los *datos son incorrectos* (que no fueron ingresados correctamente) la aplicación le notificará al usuario de este suceso para revisarlos. Y si los *datos son correctos* (que todos los datos fueron ingresados correctamente) la aplicación *calculará* el mejor polinomio de la mejor aproximación y *mostrará la gráfica*.

3.5. Implementación computacional

En esta etapa del ciclo de vida de software, se ha recopilado toda la información necesaria para la implementación del algoritmo probabilístico;

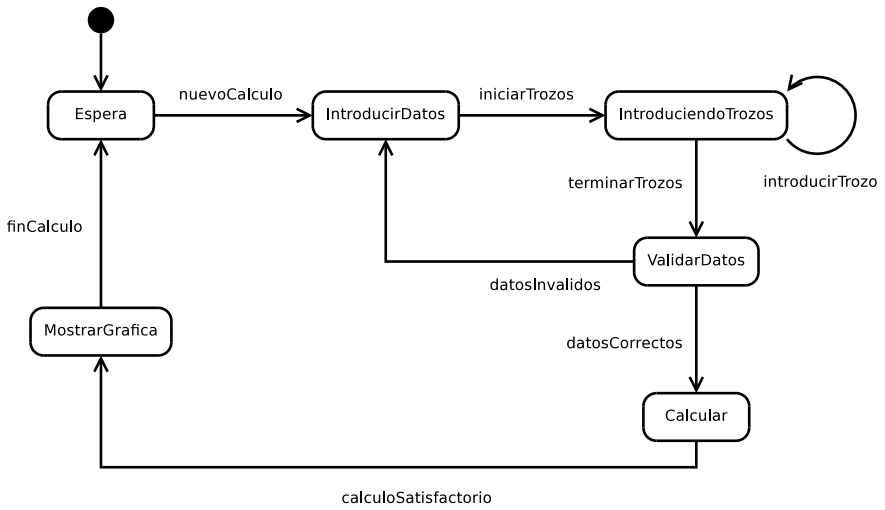


Figura 3.3: Diagrama de estados para calcular el mejor polinomio de la mejor aproximación.

desde el catálogo de requerimientos, hasta el diseño de estos mismos; considerando aspectos importantes como la validación de los argumentos.

Para consolidar toda esta información, se requiere programar todas las necesidades del usuario final.

A continuación se muestra la función principal que es la encargada de calcular el mejor polinomio de la mejor aproximación.

```
A = zeros(n+2, n+2);
B = zeros(n+2, 1);
X = zeros(n+2);
T = zeros(1, n+2);

a = Is(1);
b = Is(length(Is));
```

En esta primera parte de la función se inicializan las variables que se utilizarán en todo el proceso del algoritmo.

Variable A. Matriz de $(n+2)$ columnas $\times$ $(n+2)$ renglones, que almacenará los valores t_i correspondientes a cada sistema de ecuaciones.

Variable X. Vector de tamaño $(n+2)$ que almacenará la solución encontrada para el sistema de ecuaciones de la iteración correspondiente. Esta variable es importante puesto que además de contener el mejor polinomio de la mejor aproximación, contendrá el valor de la variable d que será utilizado al final de cada iteración.

Variable B. Vector de tamaño $(n+2)$ que almacenará todas las evaluaciones $f(t_i)$.

Variable T. Vector de tamaño $(n+2)$ que almacenará todos los t_i generados aleatoriamente. Estos t_i serán distintos en cada iteración.

Las últimas dos líneas almacenan en las variables a y b el intervalo de toda la función, que incluso puede ser definida por trozos; por esta razón el intervalo o los intervalos se guardan en un vector que contiene el intervalo de todos los trozos. Analizando los intervalos desde el lado de la programación, se decidió resolver este problema convirtiendo todas las funciones elementales (no definida por trozos) en funciones definidas en un trozo.

En la siguiente parte de la función se realizan todas las iteraciones solicitadas por el usuario; además de que en cada iteración se resuelve el sistema de ecuaciones y se compara el mejor d encontrado hasta el momento con el d actual de la iteración correspondiente.

```
for ctr=1 : iteraciones
    % Generar los n+2 puntos y ordenarlos
    T_AUX = sort(a + (b-a) * rand(1, (n+2)));

    % Evaluacion de la sumatoria
    for i=0 : n+1
        for j=0 : n
            A_AUX((i+1), (j+1)) = T_AUX(1, i+1)^j;   % a_i^j
        end
        A_AUX((i+1), (n+2)) = (-1)^i;                % (-1)^i
        % Leer el punto t
        x = T_AUX(1, i+1);
        % Evaluar x en la funcion correspondiente a las condiciones de cada
        % funcion
        B_AUX((i+1), 1) = evalF(x, Fs, Is);          % F(t_i)
    end

    % Solucion al sistema de ecuaciones generado en cada iteracion
    X_AUX = A_AUX \ B_AUX;

    % Si el d actual es mayor al d anterior, entonces este es el mejor para
    % encontrar el polinomio de mejor aproximacion
    if abs(X_AUX(n+2)) > abs(X(n+2))
```

```
        % Guardar las matrices y vectores del d mas grande, los valores
        % de T en esa iteracion y por ultimo el numero de iteracion
        A = A_AUX;
        B = B_AUX;
        X = X_AUX;
        T = T_AUX;
        I = ctr;
    end
end
```

Siguiendo el flujo de la función, se empieza con un ciclo `for` desde 1 hasta el número total de iteraciones solicitadas por el usuario. Posteriormente, dentro del primer ciclo `for` se generan ordenadamente $(n+2)$ puntos aleatorios distintos a la iteración anterior. Ya obtenidos estos puntos se procede a resolver el sistema de $(n+2)$ ecuaciones, que se almacenará en la variable X. Por último, como se ha explicado anteriormente, se compara el d actual por el mejor d encontrado hasta el momento. En caso afirmativo, guardar todas las variables importantes para uso propio de la aplicación.

Y así, la función concluye y se ha calculado aproximadamente el polinomio de la mejor aproximación con un d lo más cercano al error general d^* que las iteraciones arrojaron.

Conclusiones

En este trabajo de tesis, se desarrolló una aplicación para el cálculo de un polinomio algebraico que aproxima a una función continua en un intervalo $[a, b]$.

Se diseñó la interfaz gráfica de usuario considerando cada una de las necesidades del grupo de usuarios al que está dirigida esta aplicación. Asimismo, el diseño de esta aplicación cumple con las características de ser amigable e intuitiva al usuario.

Juntando todas estas particularidades se seleccionó a MATLAB como lenguaje de programación por sus capacidades para manejar fácilmente matrices y graficar funciones con un alto grado de detalle.

A futuro, este trabajo de tesis podría implementar el cálculo de un polinomio trigonométrico que aproxima a una función continua en el intervalo $[a, b]$. Además de afinar el algoritmo para funciones pares e impares. De esta forma, se tendría una mayor exactitud en el cálculo de la mejor aproximación para este tipo de funciones. Y más ambicioso sería; pero muy útil, implementar el cálculo de funciones en varias variables.

También, se podría desarrollar esta aplicación en lenguajes de programación multiplataforma (diferentes sistemas operativos), como Python y Java; para así instalar esta aplicación en uno o varios sistemas sin requerimientos forzosos de software comercial y reducir costos en la adquisición de software adicional.

Apéndice A

Aproximación minimax y el Algoritmo de Remez

La aproximación minimax busca el polinomio de grado n que aproxima a una función $f(x)$ en un intervalo $[a, b]$, tal que el error absoluto máximo es minimizado. El error está definido aquí como la diferencia entre la función y el polinomio.

Chebyshev prueba que existe un polinomio y que este es único. Incluso dio el criterio para que este polinomio sea minimax [1]. El criterio de Chebyshev establece que si $P_n(X)$ es el polinomio minimax de grado n, entonces debe haber al menos $(n+2)$ puntos en este intervalo en los cuales la función error alcanza el valor absoluto máximo, alternando en signo como se muestra en la figura A.1 para un $n = 3$ y por las ecuaciones siguientes.

$$a \leq x_0 < x_1 < \cdots < x_{n+1} \leq b$$

$$F(x_i) - P_n(x_i) = (-1)^i E \tag{A.1}$$

$$i = 0, 1, \ldots, n+1$$

$$E = \pm \max_{a \leq x \leq b} |F(x) - P_n(x)| \tag{A.2}$$

El polinomio minimax puede ser calculado analíticamente para cuando $n = 1$. Para los casos de un orden mayor se puede emplear un método numérico, debido a Remez [14].

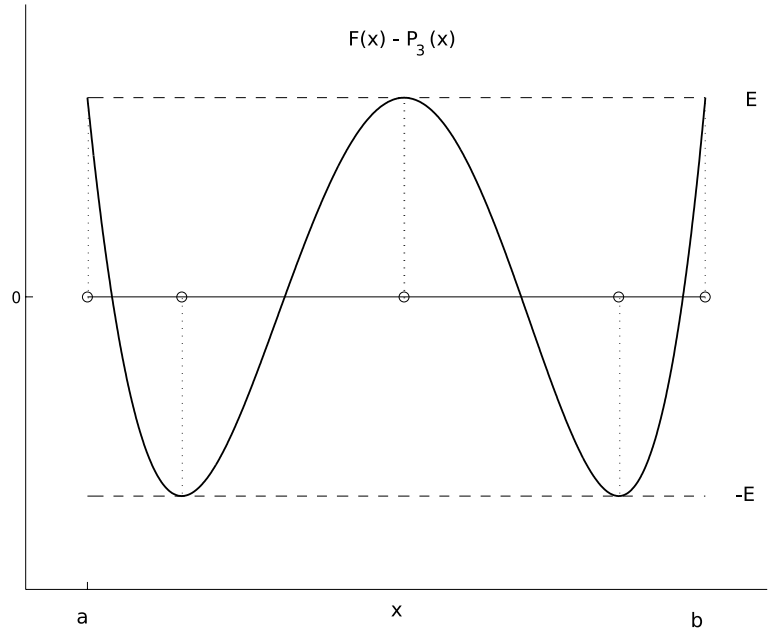


Figura A.1: Ejemplo de un polinomio minimax de tercer grado que se ajusta al Criterio de Chebyshev

El algoritmo de Remez es un algoritmo iterativo. Se inicia la primera iteración con un conjunto arbitrario de $(n+2)$ puntos $\{x_0, x_1, \ldots, x_{n+1}\}$ en el intervalo $[a, b]$. Cada iteración está compuesta de dos pasos.

En el primer paso se calculan los $(n+2)$ coeficientes, tales que la función error toma magnitudes iguales alternando en signo en los $(n+2)$ puntos dados.

$$F(x_i) - P_n(x_i) = (-1)^i E \tag{A.3}$$

$$F(x_i) - \left[c_0 + c_1(x_i - a) + c_2(x_i - a)^2\right] + \ldots + c_n(x_i - a)^n\big] = (-1)^i E \tag{A.4}$$

$$c_0 + c_1 h_i + \ldots + c_n h_i^n + (-1)^i E = F(x_i) \tag{A.5}$$
$$i = 0, 1, \ldots, n+1$$

La ecuación A.5 es un sistema de $(n+2)$ ecuaciones lineales con $(n+2)$ incógnitas $\{c_0, c_1, \ldots, c_n, E\}$. Estas ecuaciones están demostradas como in-

dependientes [14], por lo tanto, se puede resolver el sistema de ecuaciones usando cualquier método de álgebra lineal para obtener los valores de las incógnitas, así como el error en los $(n+2)$ puntos dados.

Dentro del primer paso, se calculan las incógnitas $\{c_0, c_1, \ldots, c_n, E\}$, tales que la función error en los $(n+2)$ puntos es igual en magnitud y alternando en signo. De cualquier modo, la magnitud de este error no es la máxima magnitud absoluta en el intervalo $[a, b]$. Por lo tanto, la condición minimax aún no se cumple. Se necesita un nuevo conjunto de puntos para que se cumpla la condición minimax.

El segundo paso del algoritmo de Remez busca un nuevo conjunto de $(n+2)$ puntos que cumpla con la condición minimax. Este paso también es conocido como *paso de intercambio*. Existen dos técnicas para realizar este intercambio.

En la primer técnica se intercambia un solo punto en el conjunto actual de $(n+2)$ puntos para obtener un nuevo conjunto de puntos; mientras que en la segunda, se intercambian todos los puntos del conjunto actual de $(n+2)$ puntos para obtener un nuevo conjunto de puntos.

El segundo paso se inicia observando que la función error alterna en signo en los $(n+2)$ puntos del primer paso, por lo tanto, la función error tiene $(n+1)$ raíces. Una raíz en cada uno de los intervalos: $[x_0, x_1]$, $[x_1, x_2]$, $\ldots$, $[x_n, x_{n+1}]$. Se calculan estas raíces usando cualquier método numérico, como por ejemplo el método de bisección. Posteriormente, se denotan estas raíces por $z_0, z_1, \ldots, z_n$. A continuación, se divide el intervalo $[a, b]$ en $(n+2)$ intervalos $[a, z_0]$, $[z_0, z_1]$, $[z_1, z_2]$, $\ldots$, $[z_{n-1}, z_n]$, $[z_n, b]$. En cada uno de estos intervalos calculamos el punto en el cual el error alcanza su valor máximo o mínimo y denotamos estos puntos por $x_0^*, x_1^*, \ldots, x_{n+1}^*$.

El último paso puede realizarse numéricamente, calculando la raíz de la derivada de la función error si existe cada raíz; en otro caso, se calcula la función error en los extremos del intervalo y se toma el que tiene el valor absoluto más grande.

Para realizar el intercambio de punto o puntos se define k tal que

$$k = \max_i |F(x_i^*) - P_n(x_i^*)| \tag{A.6}$$

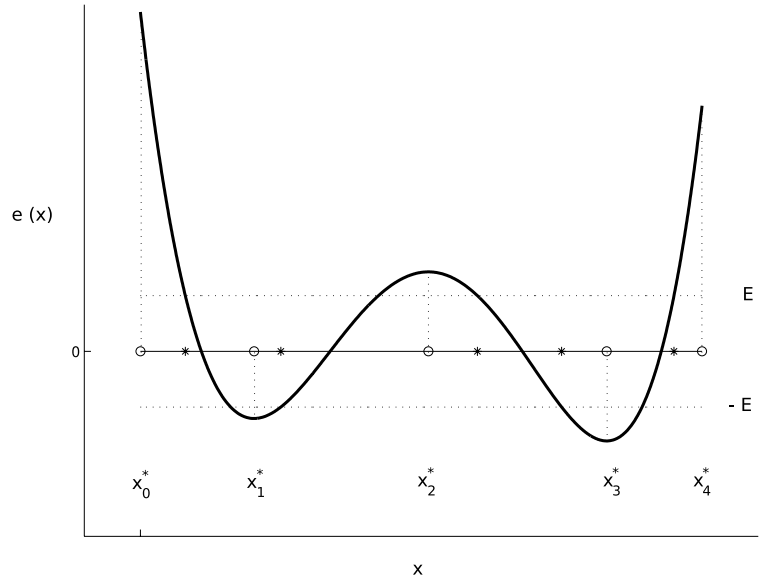


Figura A.2: Ilustración del segundo paso del algoritmo de Remez.

En la técnica de intercambio de un solo punto, se intercambia x_k por x_k^*, mientras que en la técnica de intercambio múltiple, se intercambian todos los $(n+2)$ puntos $\{x_i\}$ por $\{x_i^*\}$.

Usamos este nuevo conjunto de $(n+2)$ puntos para el primer paso de la iteración siguiente. Repetimos los dos pasos, hasta que la diferencia entre el conjunto anterior de $(n+2)$ puntos y el conjunto actual de $(n+2)$ puntos se encuentre en una tolerancia máxima permitida.

La figura A.2 ilustra gráficamente el segundo paso para un polinomio de tercer grado.

Al final del algoritmo de Remez, se acerca a la condición minimax, por lo tanto, la magnitud de la función error en el conjunto final de los $(n+2)$ puntos (E) representa el valor absoluto máximo del error aproximado.

Apéndice B

Guía de usuario

Esta guía de usuario explica paso a paso, el manejo y funcionamiento de la aplicación *Cálculo del polinomio de la mejor aproximación*. Ésta aplicación combina dos formas para calcular el mejor polinomio de la mejor aproximación y mostrarlo gráficamente en una ventana independiente, en donde se encuentran la función, el mejor polinomio y el error cometido del cálculo.

La aplicación cuenta con las características siguientes:

- Selección del método a utilizar para el cálculo del polinomio (Remez o probabilístico).
- Ingreso de una función elemental o una definida por partes o ingreso de la función desde un archivo con sintaxis propia.
- Previsualizar la función (elemental o definida por trozos) antes de ejecutar el algoritmo.
- Verificación de datos (necesitados por el método a utilizar).
- Limpiar todo el formulario[1] en un solo clic.
- Registro de los sucesos importantes generados por la aplicación.

[1]Conjunto de campos solicitados por un determinado programa, los cuales se almacenarán para su uso posterior o manipulación.

- Posibilidad de un nuevo cálculo sin ejecutar nuevamente la aplicación.

Todas estas características, hacen a esta aplicación amigable al usuario. Además, cada una de ellas está diseñada para cubrir las necesidades del usuario final.

Las dos formas con las que está conformada la aplicación para calcular el mejor polinomio de la mejor aproximación son:

1. Algoritmo de Remez; esta implementación fue hecha por Mónica Romero Cruz.
2. Métodos probabilísticos; implementación realizada por Alfonso Marquez Nieva.

B.1. Requisitos mínimos de software

- Microsoft Windows XP.
- MATLAB 7.0.
- Symbolic Math Toolbox de MATLAB.

B.2. Instalación de la aplicación

Esta aplicación fue desarrollada en el lenguaje de programación de MATLAB y para poder ejecutar la aplicación, requerimos primero de instalarla en el directorio de trabajo de MATLAB. Para realizar ésta instalación se deben seguir los pasos siguientes:

1. Ejecutar el archivo `aproximacionPolinomial.exe`. Después, se mostrará una ventana como aparece en la figura B.1. En caso de que se requiera instalar la aplicación en otra carpeta, dar clic en el botón `Examinar` (ver figura B.2). Esta guía de usuario instalará la aplicación en el directorio de trabajo de MATLAB (`C:\MATLAB7\work`); además de que es recomendable que se instale en esta carpeta.

2. Dar clic en el botón `Instalar`.

3. Cerrar la ventana del instalador (ver figura B.3).

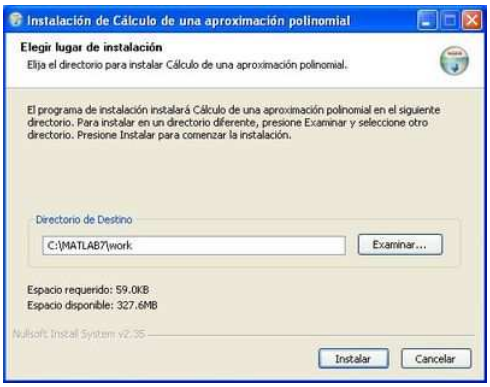


Figura B.1: Ejecutando el instalador de la aplicación.

Figura B.2: Seleccionando la carpeta de instalación.

Hasta aquí, la instalación de la aplicación ha sido exitosa. El siguiente paso es ejecutarla y para realizarlo, se necesita iniciar MATLAB.

1. Dar clic en el botón Inicio, Todos los programas, `MATLAB 7.0/MATLAB 7.0`.

Después de esto, se mostrará una ventana como la que se muestra en la figura B.4. Esta ventana, indica que MATLAB se está ejecutando y se pueden introducir comandos que MATLAB pueda interpretar y procesarlos como cálculos matemáticos.

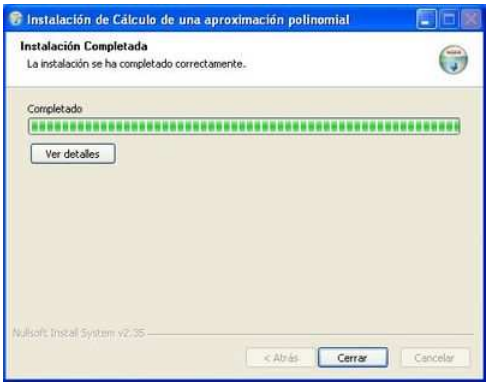


Figura B.3: Cerrando el instalador.

B.3. Ejecutando la aplicación

Realizaremos algunos ejemplos para mostrar el funcionamiento de la aplicación. Estos ejemplos están compuestos por funciones elementales y definidas por trozos.

Antes de mostrar los ejemplos, se necesita ejecutar la aplicación.

1. Escribir en la Command Window, `remezGUI`. Inmediatamente saldrá una ventana como la que se muestra en la figura B.5.

B.3.1. Función elemental

El siguiente ejemplo visualiza el manejo y comportamiento del ingreso de las funciones elementales directamente desde el formulario de la aplicación. La información que se requiere es la siguiente:

- Método a utilizar,
- número de iteraciones,
- grado del polinomio,
- tipo de función,

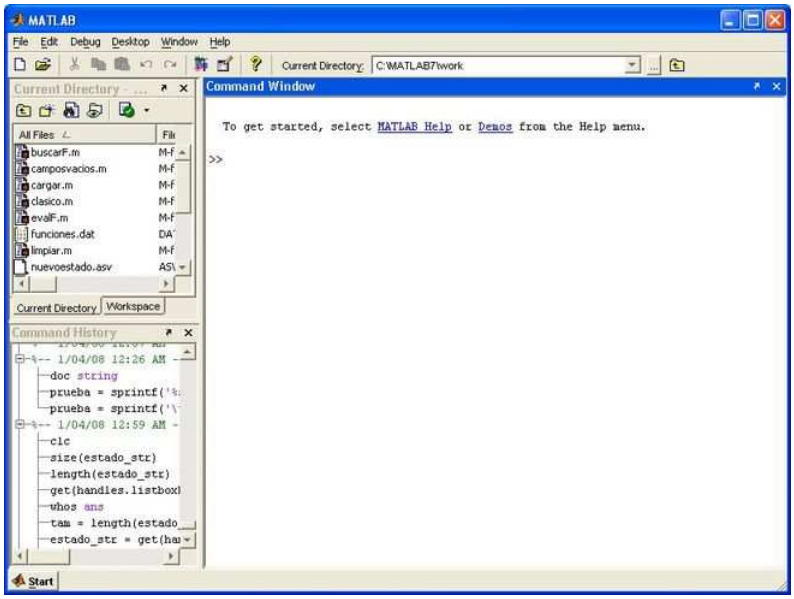


Figura B.4: MATLAB ejecutado.

- función,
- intervalo $[a, b]$.

Considerando la información que se necesita; seguiremos los siguientes pasos, ingresando al mismo tiempo los datos que se deseen; en este caso particular, ingresaremos los datos que se indican en cada paso (ver figura B.6).

1. Seleccionar el método *probabilístico*.
2. Ingresar en el campo de iteraciones, el número 10000.
3. Ingresar en el campo de grado del polinomio, el número 5.
4. Seleccionar en tipo de función, *elemental*. Esto activará los controles necesarios para el tipo de función elemental.
5. Ingresar en el campo de función sin x y como intervalo, $[-3{,}1416, 3{,}1416]$.

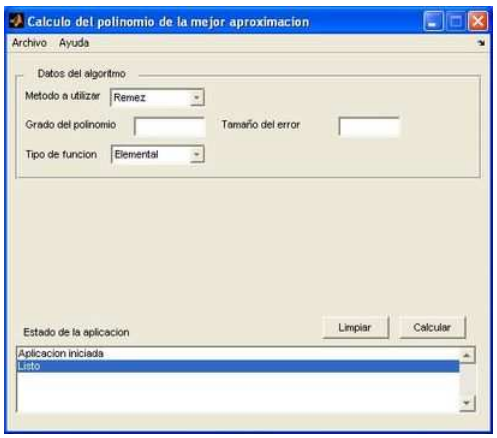


Figura B.5: La aplicación inicializada.

6. Dar clic en el botón *Calcular*. El resultado de este ejemplo se muestra en la figura B.7; esta gráfica contiene a la función, el polinomio calculado y el error cometido.

Figura B.6: Ejemplo de una función definida elemental.

B.3.2. Función definida por trozos

En este ejemplo se mostrará el funcionamiento de los controles para la función definida por trozos. El ingreso de los datos se realizará directamente

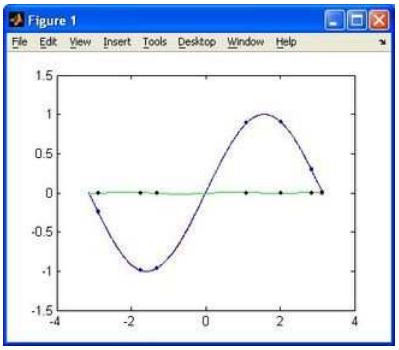


Figura B.7: Gráfica resultante para el ejemplo de la función elemental.

en el formulario de la aplicación.

La información que se requiere para este ejemplo es la siguiente:

- Método a utilizar.
- Número de iteraciones.
- Grado del polinomio.
- Función definida por partes.

Como en el ejemplo de una función elemental, ingresaremos los datos que se indican en cada paso (ver figura B.8).

1. Seleccionar *Probabilístico* como método a utilizar.
2. Ingresar 10000 como número de iteraciones.
3. Ingresar 5 como grado del polinomio.
4. Seleccionar *Por trozos* en tipo de función. Al realizar esto se mostrará el botón *Iniciar* al lado derecho del tipo de función.
5. Clic en el botón *Iniciar* para comenzar con el ingreso de los trozos de la función.

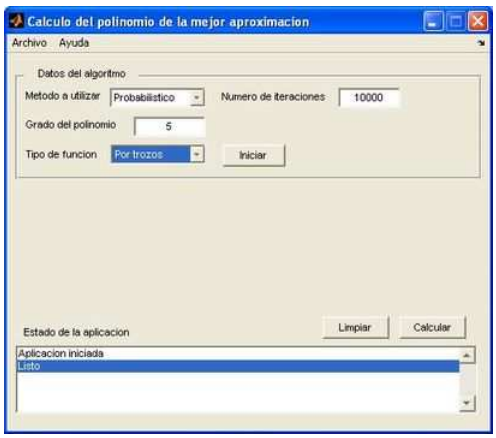


Figura B.8: Ejemplo de una función definida por trozos.

6. Ingresar el primer trozo: función, $1 - x$; intervalo, $[0, 1]$. Después, clic en el botón *Siguiente*. Se borrarán los tres campos y se mostrará en el registro de sucesos de la aplicación el trozo ingresado junto con su intervalo. Hacer lo mismo para los trozos siguientes:
   - $x - 1$, intervalo $[1, 2]$.
   - $3 - x$, intervalo $[2, 3]$.
   - $x - 3$, intervalo $[3, 4]$.
7. Para mostrar el funcionamiento del botón *Previsualizar* daremos clic en éste botón que nos mostrará la función definida en trozos (ver figura B.9).
8. Al finalizar el ingreso de los trozos, clic en el botón *Terminar*, que nos mostrará un cuadro de diálogo (ver figura B.10) con todos los trozos ingresados con sus respectivos intervalos.
9. Como último paso, dar clic en el botón *Calcular* para ejecutar el algoritmo. El resultado será una gráfica como la que se muestra en la figura B.11.

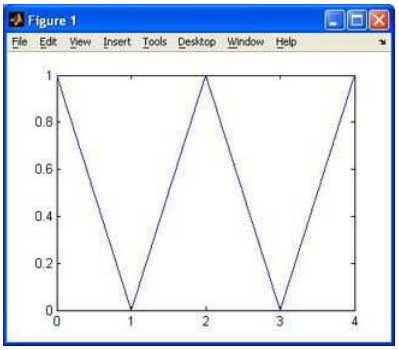


Figura B.9: Previsualización de la función definida por trozos.

Figura B.10: Cuadro de diálogo para mostrar los trozos ingresados con sus respectivos intervalos.

B.4. Cargar funciones desde archivo

El usuario puede crear archivos con extensión `.dat` para cargar funciones desde archivo. Cualquier editor de textos puede ser usado para crear este tipo de archivo. por ejemplo el Bloc de notas de Microsoft Windows.

La sintaxis para este tipo de archivos es la siguiente:

```
funcion1␣intervalo_izquierdo␣intervalo_derecho
funcion2␣intervalo_izquierdo␣intervalo_derecho
funcion3␣intervalo_izquierdo␣intervalo_derecho
```

El término `función` debe ser una cadena de caracteres (sin espacios) que defina una función. El término `intervalo_izquierdo` es un número que indica el intervalo izquierdo en que está definida la función, igualmente, el término `intervalo_derecho` es un número que indica el extremo derecho en que está definida la función.

A continuación se realizará un ejemplo de una función definida por par-

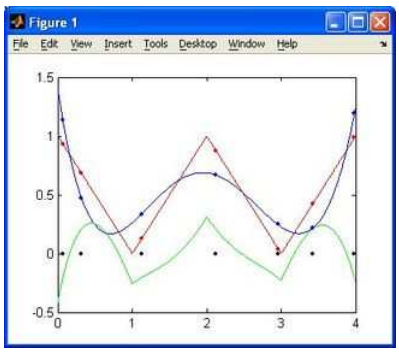


Figura B.11: Gráfica resultante para el ejemplo de la función definida por trozos.

tes. El nombre del archivo es `funciones.dat`; su contenido es el siguiente

```
1-x␣0␣1
x-1␣1␣2
3-x␣2␣3
x-3␣3␣4
```

Para cargar este archivo dentro de la aplicación, se realizan los pasos siguientes

1. En la aplicación dar clic en el menú *Archivo*, *Cargar funciones*; se mostrará un diálogo para abrir un archivo, como se muestra en la figura B.12. Buscar el archivo y dar clic en el botón *Abrir*. Después, el registro de sucesos de la aplicación mostrará que el archivo fue cargado satisfactoriamente.

2. Cuando el archivo fue cargado exitosamente la aplicación deshabilitará el ingreso de funciones directamente en el formulario. Esto es con el fin de evitar confusiones al usuario y además como una medida de precaución para la aplicación (ver figura B.13).

3. Ingresar 10000 como número de iteraciones, 5 como grado del polinomio y seleccionar *Probabilístico* en método a utilizar.

4. Para calcular el polinomio dar clic en el botón *Calcular*.

Figura B.12: Diálogo para cargar un archivo.

Figura B.13: Controles deshabilitados después de cargar satisfactoriamente un archivo.

B.5. Nuevo cálculo

Esta opción que se muestra en el menú Archivo, ofrece la posibilidad de realizar un nuevo cálculo dentro de la misma ventana, sin ejecutar nuevamente la aplicación.

Es importante mencionar que esta característica, elimina todos los valores almacenados por la aplicación de cálculos anteriores; es decir, si anteriormente se ingresaron trozos de función o se cargó un archivo de trozos de función, esta información se eliminará y no se podrá volver a utilizar. Esta información es manejada internamente por la aplicación y es distinta a la que se muestra en los campos del formulario.

Pero además, *Nuevo cálculo* nos permite limpiar el formulario (ver figura B.14); en caso de que el grado, número de iteraciones o la tolerancia del error sean nuevos valores.

Figura B.14: Diálogo para confirmar la limpieza del formulario.

B.6. Salir de la aplicación

Esta aplicación necesita que se cierre correctamente, para no generar errores posteriormente.

1. Dar clic en el menú *Archivo*, *Salir*.

Bibliografía

[1] NL Carothers, *A Short Course on Approximation Theory*, Department of Mathematics and Statistics, Bowling Green State University (1998).

[2] D. Chen and C. Moler, *Symbolic math toolbox: for use with Matlab*, (1994).

[3] E.W. Cheney, *Introduction to Approximation Theory*, American Mathematical Society, 1999.

[4] M. Cotterell and B. Hughes, *Software project management*, International Thomson Computer Press Boston, MA, USA, 1995.

[5] R.A. DeVore and G.G. Lorentz, *Constructive Approximation*, Springer, 1993.

[6] D.C. Hanselman and B. Littlefield, *Mastering MATLAB 7*, Pearson/Prentice Hall Upper Saddle River, NJ, 2005.

[7] E. Kreyszig, *Introductory functional analysis with applications*, John Wiley & Sons, 1989.

[8] Wikipedia La enciclopedia libre, *Aplicación informática*, http://es.wikipedia.org/, 2008.

[9] P. Marchand and O.T. Holland, *Graphics and Guis With Matlab*, Chapman & Hall/CRC, 2003.

[10] T.J. Rivlin, *An Introduction to the Approximation of Functions*, Courier Dover Publications, 2003.

[11] S. Rolewicz, *Metric Linear Spaces*, Springer, 1985.

[12] H.L. Royden, *Real analysis*, Macmillan New York, 1968.

[13] H. van Vliet, *Software engineering: principles and practice*, John Wiley & Sons, Inc. New York, NY, USA, 2000.

[14] L. Veidinger, *On the numerical determination of the best approximations in the Chebyshev sense*, Numerische Mathematik **2** (1960), no. 1, 99–105.