TAREA 11 - METODOS NUMERICOS

Guillermo Segura Gómez
Centro de Investigación en Matemáticas
Métodos Numéricos
5 de noviembre de 2023

1 Introducción

Existen dos problemas fundamentales a resolver en las técnicas de aproximación. El primer problema es la aproximación de una función. Durante las tareas pasadas revisamos las técnicas de interpolación. La interpolación busca una aproximación a una función que pase por todos los puntos de la función. El segundo problema a resolver es el de ajustar funciones a un conjunto de datos establecidos, es decir, encontrar la mejor función para representar a los datos. Esto se conoce como **regresión** [1]. Una de las mejores técnicas es el método de mínimos cuadrados. Este método consiste en minimizar una cantidad conocida como *error cuadrático medio*. El problema general de mínimos cuadrados se puede resducir mediante una serie de cálculos a un problema de álgebra de matrices [2]. Durante este trabajo revisaremos la técnica de mínimos cuadrados regularizados, una técnica de regresión la cual introduce un factor externo a los datos para evitar el sobre ajuste del modelo generado.

Una aplicación de técnicas de interpolación es que se pueden utilizar para calcular integrales numéricamente, ya que las funciones interpoladas (generalmente polinomios) son funciones suaves y bien comportadas, las cuales funcionan bastante bien en los métodos de integración numérica. El método básico asociado con la aproximación de una integral se conoce como **cuadratura numérica**, el cual utiliza la sumas de coeficientes de polinomios para calcular las integrales. Durante este trabajo revisaremos las formulaciones de Newton Cortés abiertas y cerradas, las cuales son técnicas para integrar numéricamente que consisten en dividir un intervalo de integración en intervalos equidistantes. Además revisaremos la técnica de cuadratura gaussiana la cual no necesariamente divide el intervalo en sub intervalos equidistantes.

2 Problema 1

1. Crear un código en C que resuelva el problema de mínimos cuadrados en el sentido de interpolación numérica para: a) Un polinomio interpolador de $n+1$ puntos de la forma $P(x) = a_0 + a_1 x + \ldots + a_n x^n$ para una función $f(x)$. b) Una función interpoladora trigonométrica de $n+1$ puntos para $f(x)$ de la forma $F_{\text{trig}}(x) = a_0 \cos\left(0\frac{\pi x}{6}\right) + a_1 \cos\left(1\frac{\pi x}{6}\right) + \ldots + a_n \cos\left(n\frac{\pi x}{6}\right)$. c) Una función interpoladora de base radial de $n+1$ puntos para $f(x)$ de la forma $F_{RBF}(x) = a_0 e^{-r_0^2} + a_1 e^{-r_1^2} + \ldots + a_n e^{-r_n^2}$, dónde $r = (x - x_i)$ para $i = 0, \ldots, n$. NOTA: Recordemos que el problema de mínimos cuadrados "regularizado" resuelve el sistema:

$$\text{Conjunto de entrenamiento } \mathcal{T} = \left\{\left(\mathbf{x}^{(k)}, y^{(k)}\right)\right\}_{k=1}^{p} \qquad y = f(\mathbf{x}) = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) \qquad \left(\mathbf{\Phi}^T\mathbf{\Phi} + \mathbf{\Lambda}\right)\mathbf{w}^* = \mathbf{\Phi}^T\mathbf{y}$$

$$\mathbf{w}^* = \left(\mathbf{\Phi}^T\mathbf{\Phi} + \mathbf{\Lambda}\right)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

$$\Phi = (\varphi_1, \varphi_2, \ldots, \varphi_m)$$

$$\mathbf{w}^* = \begin{bmatrix} w_1^* \\ w_2^* \\ \vdots \\ w_m^* \end{bmatrix} \quad \boldsymbol{\varphi}_j = \begin{bmatrix} \phi_j\left(\mathbf{x}^{(1)}\right) \\ \phi_j\left(\mathbf{x}^{(2)}\right) \\ \vdots \\ \phi_j\left(\mathbf{x}^{(p)}\right) \end{bmatrix} \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(p)} \end{bmatrix} \quad \Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_m \end{bmatrix}$$

Encontrar los valores λ_i es todo un problema de optimización, para nuestro fines consideremos que los valores son constantes, es decir, $\lambda = \lambda_i, \forall i$.

2.1 Pseudocódigo

El problema de mínimos cuadrados se puede mejorar mediante la introducción de información adicional para solucionar un problema mal definido o para evitar sobreajuste. Se introduce un término λ que regulariza la importancia de ciertos datos y así tener soluciones mas generales y menos sobreajustadas. El sistema a resolver para mínimos cuadrados regularizados queda de la forma

$$(\Phi^T\Phi + \alpha)c = \Phi^T y \tag{1}$$

en donde Φ es la matriz de las funciones base sobre las cuales se esta realizando el ajuste. Para la implementación el funcionamiento del programa en sus pasos mas fundamentales es el siguiente:

1. Se construye la matriz Φ
2. Se construye el sistema $Ac = b$. En donde $A = \Phi^T\Phi + \alpha$ y $b = \Phi^T y$
3. Se resuelve el sistema $Ac = b$ mediante un método numérico.

La lógica en la construcción de la matriz Φ es prácticamente la misma en los tres casos. Se presenta el pseudocódigo de la función polinomial.

```
void MatrixBuild(double* Phi, int rows, int cols, double* x, int n) {
  // Para cada observación
  for (int i = 0; i < rows; i++) {
    // Término constante, x^0
    Phi[i * cols] = 1.0;
    // Para cada término del polinomio
    for (int j = 1; j < cols; j++) {
      // x^j para la observación i
      Phi[i * cols + j] = pow(x[i], j);
    }
  }
}
```

Listing 1: Construcción de la matriz Phi para ajuste polinomial.

A continuación se presenta el pseudocódigo para la función que construye el sistema $Ac = b$.

```
Función buildSystem(m: Entero, n: Entero, x: Arreglo de Real, y: Arreglo de
    Real, A: Arreglo de Real, b: Arreglo de Real, lambda: Real, option: Entero
    )

    // Reservar memoria para la matriz Phi de size n*m
    Phi := ReservarMemoria(n * m)
    Si Phi es NULL entonces
        MostrarError("Error al asignar memoria.")
        TerminarEjecución()
    FinSi
```

```
// Construir la matriz Phi basada en la opción seleccionada
Según option hacer
    caso 1:
        MatrixBuild(Phi, n, m, x, n)
    caso 2:
        MatrixBuildTrig(Phi, n, m, x, n)
    caso 3:
        MatrixBuildRBF(Phi, n, m, x)
    caso contrario:
        MostrarError("Ajuste no reconocido")
        TerminarEjecución()
FinSegún

// Reservar memoria para la matriz transpuesta PhiT de size m*n
PhiT := ReservarMemoria(n * m)
Si PhiT es NULL entonces
    MostrarError("Error al asignar memoria para PhiT.")
    LiberarMemoria(Phi)
    TerminarEjecución()
FinSi

// Transponer la matriz Phi y almacenar en PhiT
MatrixT(n, m, Phi, PhiT)

// Reservar memoria para la matriz Alpha de size m*m e inicializar en
cero
Alpha := ReservarCeroMemoria(m * m)
Si Alpha es NULL entonces
    MostrarError("Error al asignar memoria para Alpha.")
    LiberarMemoria(Phi)
    LiberarMemoria(PhiT)
    TerminarEjecución()
FinSi

// Asignar lambda en la diagonal de la matriz Alpha
Para i := 0 hasta m-1 hacer
    Alpha[i * m + i] := lambda
FinPara

// Reservar memoria para PhiProd que es el producto de PhiT * Phi
PhiProd := ReservarMemoria(m * m)

// Calcular el producto de matrices PhiT y Phi y almacenar en PhiProd
MatrixProduct(PhiT, Phi, PhiProd, m, n, m)
MatrixSum(PhiProd, Alpha, A, m, m)

// Construir b como el producto de PhiT y y
Si b es NULL entonces
    MostrarError("Error al asignar memoria para b.")
    LiberarMemoria(Phi)
    LiberarMemoria(PhiT)
    LiberarMemoria(Alpha)
    LiberarMemoria(A)
    LiberarMemoria(PhiProd)
    TerminarEjecución()
FinSi
MatrixProduct(PhiT, y, b, m, n, 1)
```

```
    // Liberar memoria utilizada por la función
    LiberarMemoria(Phi)
    LiberarMemoria(PhiT)
    LiberarMemoria(Alpha)
    LiberarMemoria(PhiProd)

FinFunción
```

Listing 2: Pseudocódigo de la función buildSystem.

2.2 Ejecución

El código con nombre **Minimus2coeff.c** regresa el valor de los coeficientes utilizando los tres enfoques, polinomial, trigonométrico y radial. Para datos del libro burden tenemos el siguiente resultado [1].

```
guillermo_sego@192 Tarea11 % make
gcc -g -o build/Debug/Minimus2coeff.o -c Minimus2coeff.c
gcc -g -o build/Minimus2coeff build/Debug/matrix.o build/Debug/Minimus2coeff.
    o
guillermo_sego@192 Tarea11 % ./build/Minimus2coeff
Los coeficientes del ajuste m = 2 para opción polinómica son los siguientes:
-0.360000
1.538182
Los coeficientes del ajuste m = 2 para opción trigonométrica son los
    siguientes:
7.785868
-1.683429
Los coeficientes del ajuste m = 2 para opción radial son los siguientes:
-0.876910
4.930649
```

Se comparan con el ejemplo 8.1 y los coeficientes del ajuste polinomial son los mismos.

Probamos los códigos para datos del libro burden y tenemos lo siguiente.

3 Problema 2

Para evaluar los códigos del ejercicio 1 . Consideremos la función de Sutherland que define la viscosidad de un gas según la temperatura T en Kelvin (entre 1000), definida por $f(x) = m_0 \left(\frac{1000T}{T_0}\right)^{\frac{3}{2}} \left(\frac{T_0+S_u}{1000T+S_u}\right)$. Calculemos la viscosidad del oxígeno O_2, para ello usemos $m_0 = 1.919e^{-2}, T_0 = 273$, y $S_u = 139$. Realiza lo siguiente:

a) Evalúa los puntos $Ti = [0.273, 0.303, 0.323, 0.353, 0.423, 0.573, 1.473]$ en la función de Sutherland. Los puntos Ti corresponden a los $n + 1$ puntos xi, y las evaluaciones son los valores yi (estamos simulando que se ha realizado un experimento de laboratorio físico, es decir, sólo tenemos mediciones experimentales).

b) Usando el valor de $\lambda = 0$, 1e-5, 1e-7, para $0 \leq$ i $\leq$ n, aplica las 3 funciones de interpolación programadas obtenidas del método de mínimos cuadrados en el inciso 1), con ellos calcula la viscosidad para T = 1.2, compara con el valor real obtenido directamente con la función de Sutherland.

c) Gráfica las soluciones obtenidas en el intervalo [0.273, 1.5], usa incrementos de =0.05.

d) Repite los incisos a), b) y c), pero con los nodos T i = [0.273, 0.473, 0.673, 0.873, 0.1073, 0.1273, 1.473].

¿Qué puedes concluir de los nodos y el comportamiento de las funciones de interpolación según λ?

Las funciones son las mismas utilizadas en el problema anterior.

3.1 Ejecución

Para la ejecución y evaluación de un punto T en los polinomios calculados se utilizó el código **Minimus2c**. Para la evaluación se generaron funciones para cada caso. Todo vive en la biblioteca **matrix.h**.

Se escoje la opción de ajuste que se va realizar. Esta fue la razón por la cual se construyeron de esta manera las funciones. Se construyeron lo mas general posible para poder tener este tipo de espacio de trabajo sobre los cuales evaluar y comparar los resultados sin hacer cambios al programa utilizando los tres tipos de ajuste.

Para el polinomial

```
guillermo_sego@192 Tarea11 % make
gcc -g -o build/Debug/Minimus2n.o -c Minimus2n.c
gcc -g -o build/Minimus2n build/Debug/matrix.o build/Debug/Minimus2n.o
guillermo_sego@192 Tarea11 % ./build/Minimus2n
Introduce el tipo de ajuste:
1. Polinomial
2. Trigonométrico
3. Radial
1
Tabla de Evaluaciones para T = 1.2
m \ lambda      0.000000e+00    1.000000e-05    1.000000e-07
2               0.052802        0.052802        0.052802
3               0.055654        0.055650        0.055654
4               0.057924        0.057921        0.057924
5               0.061330        0.061328        0.061330

El valor real de la función de Sutherland para T = 1.2 es: 0.054415
```

Para el trigonométrico

```
guillermo_sego@192 Tarea11 % ./build/Minimus2n
Introduce el tipo de ajuste:
1. Polinomial
2. Trigonométrico
3. Radial
2
Tabla de Evaluaciones para T = 1.2
m \ lambda      0.000000e+00    1.000000e-05    1.000000e-07
2               -0.007621       -0.007608       -0.007621
3               0.003425        0.003425        0.003425
4               0.001637        0.001653        0.001637
5               -0.011584       -0.011575       -0.011584

El valor real de la función de Sutherland para T = 1.2 es: 0.054415
```

Para el radial

```
guillermo_sego@192 Tarea11 % ./build/Minimus2n
```

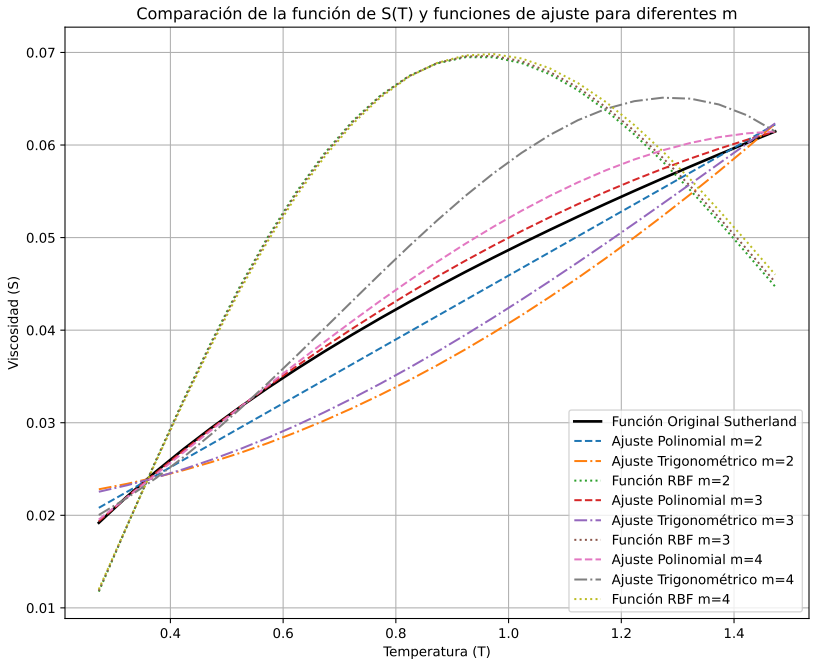

Figure 1: Comparación curvas de ajuste para diferentes valores de m

```
Introduce el tipo de ajuste:
1. Polinomial
2. Trigonométrico
3. Radial
3
Tabla de Evaluaciones para T = 1.2
m \ lambda      0.000000e+00    1.000000e-05    1.000000e-07
2               0.488342        0.469090        0.488141
3               0.624499        0.615662        0.624410
4               0.660465        0.657121        0.660432
5               0.509973        0.509386        0.509967

El valor real de la función de Sutherland para T = 1.2 es: 0.054415
```

Es evidente que el ajuste que mejor se aproxima a la función es el polinómico. Se graficaron los resultados para diferentes valores de m y también diferentes valores de lambda. En la figura 1 tenemos diferentes valores de m. Luego vemos los diferentes valores de lambda en la figura 2.

Repitiendo el proceso para el caso del inciso d.

Para el polinomial

```
guillermo_sego@192 Tarea11 % ./build/Minimus2n
Introduce el tipo de ajuste:
1. Polinomial
2. Trigonométrico
3. Radial
1
```

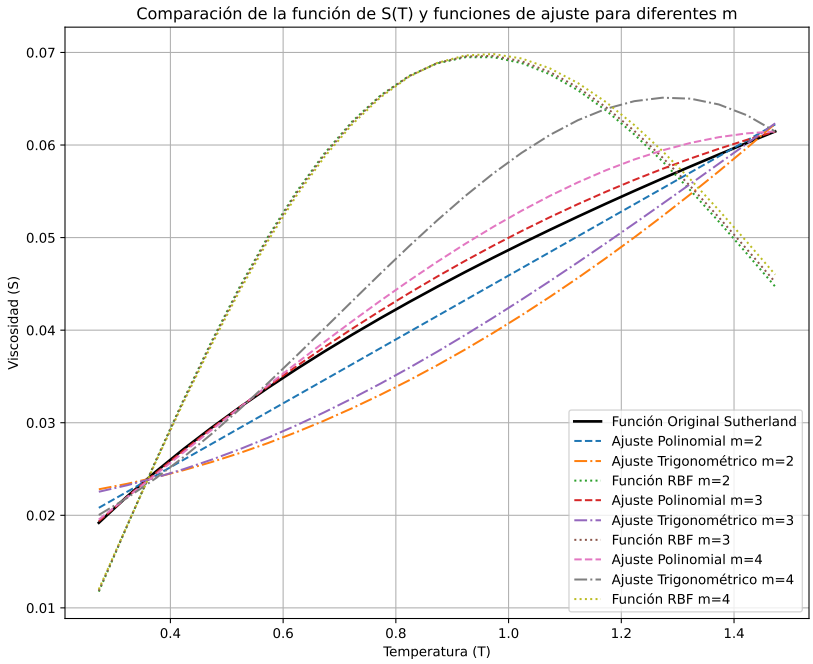

Figure 2: Comparación curvas de ajuste para diferentes valores de lambda

```
Tabla de Evaluaciones para T = 1.2
m \ lambda      0.000000e+00    1.000000e-05    1.000000e-07
2               0.054778        0.054778        0.054778
3               0.055588        0.055587        0.055588
4               0.053717        0.053736        0.053717
5               0.052658        0.052668        0.052658

El valor real de la función de Sutherland para T = 1.2 es: 0.054415
```

Para el trigonométrico

```
guillermo_sego@192 Tarea11 % ./build/Minimus2n
Introduce el tipo de ajuste:
1. Polinomial
2. Trigonométrico
3. Radial
2
Tabla de Evaluaciones para T = 1.2
m \ lambda      0.000000e+00    1.000000e-05    1.000000e-07
2               -0.017440       -0.017425       -0.017440
3               0.102610        0.088996        0.102454
4               -0.004737       -0.004739       -0.004737
5               -0.017345       -0.017344       -0.017345

El valor real de la función de Sutherland para T = 1.2 es: 0.054415
```

Para el radial

```
guillermo_sego@192 Tarea11 % ./build/Minimus2n
```

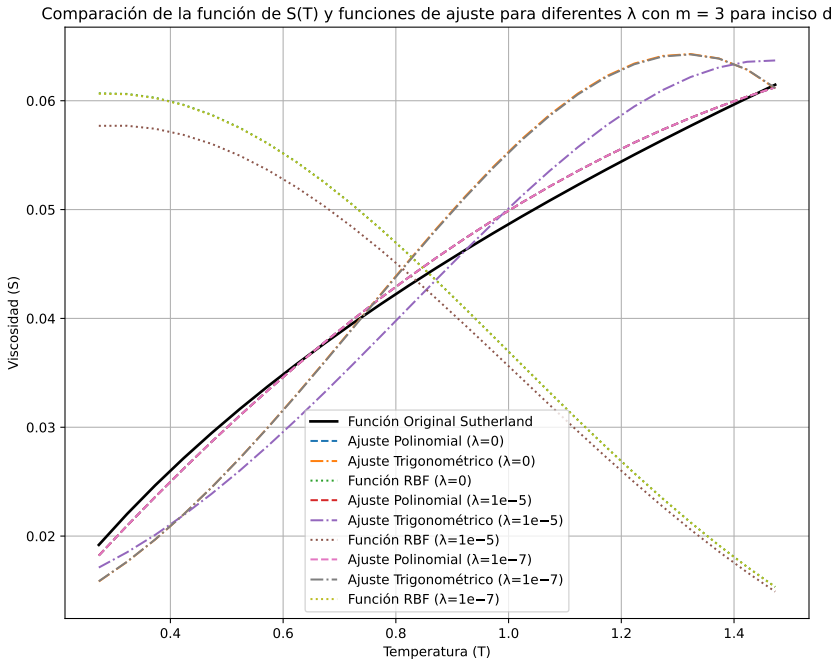

Figure 3: Comparación curvas de ajuste para diferentes valores de lambda para el inciso d

```
Introduce el tipo de ajuste:
1. Polinomial
2. Trigonométrico
3. Radial
3
Tabla de Evaluaciones para T = 1.2
m \ lambda      0.000000e+00    1.000000e-05    1.000000e-07
2               0.070989        0.070978        0.070989
3               0.125816        0.124826        0.125806
4               0.133624        0.133498        0.133623
5               0.245382        0.245092        0.245379

El valor real de la función de Sutherland para T = 1.2 es: 0.054415
```

Graficamos para diferentes valores de lambda en este caso. Vemos la gráfica en la figura 3.

Es evidente que hay algo incorrecto en las funcione generadoras para los caos trigonométrico y radial. La conclusión que podemos determinar en base a la correcta aproximación polinomial para lambda es que en función de este valor podemos introducir correcciones para evitar el sobre ajuste, es decir, cuando tenemos un modelo que esta sobre ajustado, un nuevo conjunto de datos del mismo experimento no se puede determinar. Los valores de lambda permiten generalizar el modelo y asi tener predicciones mas precisas para los nuevos conjuntos de datos.

4 Problema 3

Crea una librería en C que calcule la aproximación a una integral en el intervalo [a,b] arbitrario con los métodos de Newton-Cotes abierto (n = 0, 1, 2, 3), Newton-Cotes cerrado (n = 1, 2, 3, 4), y cuadratura gaussiana (n = 1, 2, 3, 4, 5). Con las librerías aproxima las siguientes integrales:

$$\int_0^{\pi/4} \sin(x)dx, \quad \int_1^{1,5} x^2 \ln x dx, \quad \int_0^1 x^2 e^{-x} dx$$

NOTA: Crea una tabla comparativa de los resultados. OJO: La primer integral es la vista en clase.

Se construyó la librería en el archivo **integration.c**.

4.1 Pseudocódigo

Los códigos tienen estructura similar debido a que se utilizan fórmulas. Sólo se presenta el pseudocódigo de la primer función.

```
function NewtonCotesAbiertoOrdenCero(a: real; b: real; f: function): real;
var
    h: real;
    puntoMedio: real;
    resultado: real;
begin
    h := b - a;
    puntoMedio := a + h / 2.0;
    resultado := h * f(puntoMedio);
    return resultado;
end;
```

Listing 3: Newton-Cotes Abierto de orden 0 (Punto Medio)

4.2 Ejecución

Vamos a evaluar las integrales y compararlas en un solo programa. A continuación se presenta la ejecución del código.

```
    guillermo_sego@192 Tarea11 % make
gcc -g -o build/Debug/IntegralExamples.o -c IntegralExamples.c
gcc -g -o build/IntegralExamples build/Debug/integration.o build/Debug/
    IntegralExamples.o
guillermo_sego@192 Tarea11 % ./build/IntegralExamples
Method                  | Integral of sin(x)    | Integral of x^2 * ln(x) |
    Integral of x^2 * e^(-x)
------------------|----------------|------------------|--------------
NC Open Order 0         | 0.3005588649            | 0.1743308995
      | 0.1516326649
NC Open Order 1         | 0.2979875422            | 0.1803127496
      | 0.1538999882
NC Open Order 2         | 0.2928586592            | 0.1922716135
      | 0.1590432683
NC Open Order 3         | 0.2928692281            | 0.1922678867
      | 0.1595142015
```

```
NC Closed Order 1       | 0.2776801836            | 0.2280741233
      | 0.1839397206
NC Closed Order 2       | 0.2929326378            | 0.1922453074
      | 0.1624016835
NC Closed Order 3       | 0.2929107025            | 0.1922530931
      | 0.1614099213
NC Closed Order 4       | 0.2928931826            | 0.1922593373
      | 0.1606105287
Gauss Quadrature Order 1| 0.3005588649            | 0.1743308995
      | 0.1516326649
Gauss Quadrature Order 2| 0.2928669073            | 0.1922687064
      | 0.1594104310
Gauss Quadrature Order 3| 0.2928932536            | 0.1922593773
      | 0.1605953868
Gauss Quadrature Order 4| 0.2928932188            | 0.1922593578
      | 0.1606027775
Gauss Quadrature Order 5| 0.3403824597            | 0.2353897363
      | 0.1978182805
```

5 Problema 4

Investiga el método de extrapolación Richardson, y el método de Romberg, presenta un ejemplo.

5.1 Métodos de Extrapolación de Richardson y Método de Romberg

La búsqueda de precisión en el cálculo numérico de integrales ha llevado al desarrollo de métodos sofisticados que mejoran las estimaciones aproximadas. Entre estas técnicas, la extrapolación de Richardson y el método de Romberg se destacan por su capacidad para refinar sucesivamente las aproximaciones. [1]

5.2 Extrapolación de Richardson

El método de extrapolación de Richardson aprovecha la relación entre el tamaño del paso de una aproximación numérica y su error asociado. Este procedimiento se basa en el cálculo de dos aproximaciones sucesivas, con tamaños de paso diferentes, para extrapolar una aproximación que sería obtenida con un paso hipotéticamente igual a cero, donde el error es mínimo.

El fundamento de este método reside en la suposición de que el error se comporta de manera polinomial respecto al tamaño del paso. Con dos estimaciones, $f(h)$ y $f(h/2)$, se puede eliminar el término principal del error y obtener una aproximación mejorada. Esta técnica es particularmente eficiente cuando se combina con métodos de alta precisión para la integración numérica. [1]

5.3 Método de Romberg

El método de Romberg es una aplicación específica de la extrapolación de Richardson en la integración numérica. Se basa en la Regla del Trapecio y utiliza una secuencia de aproximaciones de la integral con tamaños de paso decrecientes, que son halved sucesivamente.

Aplicando la extrapolación de Richardson a estas aproximaciones, Romberg construye una tabla de extrapolación, donde cada nivel sucesivo aumenta el orden de precisión de la aproximación. La convergencia es generalmente rápida, y el resultado tiende a ser muy preciso. [1]

5.4 Ejemplo de Aplicación

Consideremos la integral definida $\int_0^1 e^{-x^2} dx$), cuyo valor exacto no es sencillo calcular simbólicamente. Utilizando la Regla del Trapecio con un solo intervalo, obtenemos una primera aproximación. Repitiendo con dos intervalos, y luego aplicando la extrapolación de Richardson, se mejora la estimación. Continuando este proceso, y aplicando el método de Romberg, se construye una tabla de valores que converge rápidamente al valor exacto de la integral.

6 Conclusión

Durante este trabajo realizamos un estudio acerca de las técnicas de aproximación, el problema de aproximación a un conjunto de datos (mínimos cuadrados) y una aplicación de la aproximación de funciones por polinomios en la integración numérica. El problema general de mínimos cuadrados resulta de la solución de un sistema matricial. El término que añadimos para evitar sobre ajuste nos permite tener un mejor modelo, mas general y mejor adaptado a los diferentes conjuntos de datos. Además las aproximaciones a funciones, nos permite integrar numéricamente funciones cuyas integrales pueden ser muy complicadas o directamente no existen. En general las aproximaciones difieren muy poco cuando cambiamos el orden. Para términos prácticos en los que no necesitamos tanta precisión es indistinguible el método de integración. Para cuestiones en las que si se necesite algo muy específico necesitamos evaluar la precisión de las aproximaciones.

References

[1] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical analysis*. Cengage learning, 2015.

[2] H. Abdi, "Partial least square regression (pls regression)," *Encyclopedia for research methods for the social sciences*, vol. 6, no. 4, pp. 792–795, 2003.