TEC | Tecnológico de Costa Rica

Área académica de ingeniería en computadores

Curso: Análisis numérico para ingeniería
Profesor: Juan Pablo Soto

Integrantes del equipo:
Joseph Jiménez Zuñiga
Alvaro Vargas Molina
Manuel Bojorge Araya

Paquete computacional NumInt

Tabla de Contenidos

Descripción de NumInt

El paquete computacional NumInt proporciona un conjunto de funciones para realizar cálculos numéricos de integración mediante diferentes métodos. Estos métodos se utilizan para aproximar el valor de una integral definida, es decir, calcular la integral de una función en un intervalo dado.

Los métodos incluidos en este paquete son los siguientes:

1. Cuadratura Gaussiana Compuesta:
    - Esta función implementa el método de cuadratura Gaussiana compuesta el cual es un método de integración que utiliza una combinación de puntos de muestreo y pesos específicos para aproximar la integral.
2. Regla de Simpson:
    - El paquete también incluye una función que aplica el método de Simpson para la integración numérica. El método de Simpson utiliza interpolación polinómica para aproximar la función integrada y calcular la integral mediante la suma de áreas de segmentos de parábola.
3. Regla del Trapecio:
    - Otra función proporcionada por el paquete es la que implementa el método del trapecio para la integración numérica. El método del trapecio divide el intervalo de integración en segmentos de forma trapezoidal y aproxima la integral como la suma de áreas de estos trapecios.
4. Método de Romberg:
    - Además de los métodos anteriores, el paquete también incluye una función que implementa el método de Romberg. El método de Romberg es un enfoque iterativo que mejora la precisión de la aproximación de la integral mediante una combinación de extrapolación y refinamiento de los resultados obtenidos con otros métodos de integración.

Estas funciones te permiten realizar cálculos numéricos de integración de manera eficiente y precisa. Se utilizan para aproximar el valor de una integral definida en un intervalo dado, proporcionando la función y los límites de integración como entrada.

Instalación

A continuación, se detallan los pasos para instalar el paquete computacional NumInt:

1. Abrir Octave
2. Ir a la ubicación del archivo del paquete. Puede utilizar el comando “pwd” en la línea de comandos de Octave para verificar tu ubicación actual.
3. Utilize el siguiente comando “pkg install NumInt.zip” en la línea de comandos de Octave para instalar el paquete
4. Una vez que el paquete esté instalado correctamente, puedes cargarlo en tu sesión de Octave utilizando el comando “pkg load NumInt”
5. Verificar si el paquete se instaló correctamente utilizando el comando “pkg list” en la línea de comandos de Octave. Esto mostrará una lista de paquetes instalados en tu sistema.

Prerrequisitos:

1. Última versión de Octave
2. Tener instalado el paquete symbolic-3.1.1

Funciones implementadas y formulación matemática

1. Cuadratura Gaussiana

La implementación de este método numérico se hace de manera compuesta, es decir que el intervalo [a,b] se divide en subintervalos y a cada uno se le aplica la cuadratura.

```
function I = gaussiana_compuesta( f, a, b, N)
  % Aproximación numérica de la integral de una función en un intervalo mediante la fórmula de Gauss.
  % Sintaxys : gaussiana_compuesta(f,a,b,N)
  % Inputs:
  %    a, b = intervalo [a, b],
  %    N = numero de puntos en los que se divide el intervalo [a, b]
  % Outputs:
  %    I =  I = Aproximacion numerica de (1)
  g = @(x) ((b-a)/2) * f( (b-a) / 2*x + (b+a)/2 );
  [x,w]=pesos_X_gauss(N);
  y=0;
  for i = 1:N
    y = y + w(i) * g(x(i));
  end
  I = y;
end
```

Existe una función auxiliar para calcular los pesos

```
function [x,w]=pesos_X_gauss(n)
  % Devuelve los nodos y pesos para la integración numérica mediante la fórmula de Gauss.
  % Sintaxys : pesos_X_gauss(n)
  % Inputs:
  %   n: número de puntos de integración
  %
  % Outputs:
  %   x: vector de nodos de integración
  %   w: vector de pesos de integración
  switch (n)
    case 2
      x = [-0.5773502691896257, 0.5773502691896257];
      w = [1, 1];
    case 3
      x = [0, -0.7745966692414834, 0.7745966692414834];
      w = [0.8888888888888888, 0.5555555555555556, 0.5555555555555556];

    .
    .
    .

    case 20
      x = [-0.07652652113349733, 0.07652652113349733, -0.2277858511416451, 0.2277858511416451, -0.3737060887154196, 0.3737060887154196, -0.5150834541676965, 0.5150834541676965, -0.651
      w = [0.1527533871307259, 0.1527533871307259, 0.1491729864726037, 0.1491729864726037, 0.1420961093183820, 0.1420961093183820, 0.1316886384491766, 0.1316886384491766, 0.1181945315
  endswitch
end
```

También se hace una versión iterativa

```
function I = gaussiana_compuesta_iterativa(f, a, b, M, tol, iterMax)
  % Aproximación numérica iterativa de la integral de una función en un intervalo mediante la fórmula de Gauss compuesta.
  % Sintaxis: I = gaussiana_compuesta_iterativa(f, a, b, M, tol, iterMax)
  % Inputs:
  %   f: función a integrar
  %   a, b: límites de integración [a, b]
  %   M: orden de la cuadratura (considerar un orden máximo de 10)
  %   tol: tolerancia para la convergencia del resultado
  %   iterMax: número máximo de iteraciones
  % Output:
  %   I: aproximación numérica de la integral de f en [a, b]

  % Inicializar variables
  I_prev = gaussiana_compuesta(f, a, b, M-1); % Valor de la aproximación anterior
  I = gaussiana_compuesta(f, a, b, M); % Valor de la aproximación actual
  iter = 1; % Contador de iteraciones

  while abs(I - I_prev) > tol && iter < iterMax
    I_prev = I; % Guardar el valor anterior
    I = gaussiana_compuesta(f, a, b, M); % Calcular nueva aproximación
    iter = iter + 1; % Incrementar contador de iteraciones
  end

  if iter == iterMax
    fprintf('No se alcanzó la convergencia después de %d iteraciones.', iter);
  end
end
```

En cuanto a su formulación matemática, una cuadratura gaussiana es un enfoque para aproximar el valor de una integral definida de una función. Específicamente, una cuadratura gaussiana **n** se construye para obtener resultados exactos al integrar polinomios de grado 2n-1 o inferior. Para lograr esto, se seleccionan cuidadosamente los puntos de evaluación x_i y los pesos w_i La regla de cuadratura gaussiana se expresa comúnmente para una integral en el intervalo [-1, 1] y se define mediante la siguiente expresión:

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

Los coeficientes w_i están dados por el método de Gauss-Legendre

$$w_i = \frac{2}{\left(1 - x_i^2\right)\left[P_n'(x_i)\right]^2}$$

donde P_n es el polinomio de Legendre de grado n en el intervalo [−1, 1], y los x_i son las raíces de dicho polinomio.

2. Regla de Simpson

Se implementa de manera que se utiliza la regla de Simpson simple recursivamente en cada subintervalo de la siguiente manera

```
function I=simpson_compuesto(f,a,b,n);
  % Aproximación numérica de la integral de una función en un intervalo mediante el metodo de Simpson compuesto, el cual divide en subitervalos y utiliza recursivamente para aplicar el m
  % Sintaxys : simpson_compuesto(f,a,b,n)
  % Inputs:
  %    f = funcion a evaluar,
  %    a, b = intervalo [a, b],
  %    n = numero de puntos en los que se divide el intervalo [a, b]
  % Outputs:
  %    I =  I = Aproximacion numerica de la integral

  h=(b-a)/(n-1);
  xSoporte=a:h:b;
  I=0;
  for i=1:n-1
    I=I+simpson(f,xSoporte(i),xSoporte(i+1));
  end
end

function I = simpson(f, a, b)
  % Aproximación numérica de la integral de una función en un intervalo mediante el método de Simpson.
  % Sintaxis: I = simpson(f, a, b)
  % Inputs:
  %   f: función a evaluar
  %   a, b: límites de integración [a, b]
  % Output:
  %   I: aproximación numérica de la integral
  I = (b - a) / 6 * (f(a) + 4 * f((a + b) / 2) + f(b));
end
```

También se implementa de manera iterativa como se muestra a continuación

```
function I = simpson_compuesto_iterativo(f, a, b, n, tol, iterMax)
  % Aproximación numérica de la integral de una función en un intervalo mediante el método de Simpson compuesto iterativo.
  % Sintaxis: I = simpson_compuesto_iterativo(f, a, b, n, tol, iterMax)
  % Inputs:
  %   f: función a evaluar
  %   a, b: límites de integración [a, b]
  %   n: número de puntos en los que se divide el intervalo [a, b]
  %   tol: tolerancia para la convergencia del resultado
  %   iterMax: número máximo de iteraciones
  % Output:
  %   I: aproximación numérica de la integral

  I_prev = 0; % Valor de la aproximación anterior
  I = simpson_compuesto(f, a, b, n); % Valor de la aproximación actual
  iter = 1; % Contador de iteraciones

  while abs(I - I_prev) > tol && iter < iterMax
    I_prev = I; % Guardar el valor anterior
    I = simpson_compuesto(f, a, b, n); % Calcular nueva aproximación con mayor número de puntos

    iter = iter + 1; % Incrementar contador de iteraciones
  end

  if iter == iterMax
    fprintf('No se alcanzó la convergencia después de %d iteraciones.', iter);
  end
end
```

En el campo del análisis numérico, el método de Simpson (llamado así en honor a Thomas Simpson) es una técnica utilizada para aproximar la integral de una función. Este método se basa en la idea de aproximar los subintervalos de la función mediante polinomios de segundo grado.

Cuando el intervalo [a, b] no es lo suficientemente pequeño, el cálculo de la integral puede presentar un error significativo. Por esta razón, se emplea la fórmula compuesta de Simpson. Se divide el intervalo [a, b] en n subintervalos iguales (donde n es un número par), y se definen los puntos

$$x_j = a + j * h$$

donde $h = \frac{(b-a)}{n}$ para $j = 0, 1, \ldots, n.$

Luego, se aplica la regla de Simpson a cada subintervalo, lo que implica utilizar polinomios de segundo grado para aproximar la función y calcular la integral en cada uno de ellos de la siguiente manera

$$\int_{x_{j-1}}^{x_{j+1}} f(x)\,dx \approx \frac{x_{j+1} - x_{j-1}}{6}\left[f(x_{j-1}) + 4f(x_j) + f(x_{j+1})\right]$$

El máximo error viene dado por la expresión

$$(b-a)\,\frac{h^4}{180}\max_{a\le\xi\le b}\left|f^{(4)}(\xi)\right|$$

3. Regla del Trapecio

Se implementa de manera recursiva en la que se utiliza la regla del trapecio simple a cada subdivisión del intervalo

```
function I=trapecio_compuesto(f,a,b,N);
  % Aproximación numérica de la integral de una función en un intervalo mediante
  % el metodo del trapecio compuesto, el cual divide en subitervalos y utiliza
  % recursivodad para aplicar el metodo del trapecio a cada intervalo.
  % Sintaxys : trapecio_compuesto(f,a,b,N)
  % Inputs:
  %    f = funcion a evaluar,
  %    a, b = intervalo [a, b],
  %    N = numero de puntos en los que se divide el intervalo [a, b]
  % Outputs:
  %    I =  I = Aproximacion numerica de la integral
  h=(b-a)/(N-1);
  xSoporte=a:h:b;
  I=0;
  for i=1:N-1
    I=I+trapecio(f,xSoporte(i),xSoporte(i+1));
  end
end

function I=trapecio(f,a,b)
  % Aproximación numérica de la integral de una función en un intervalo mediante
  % el método del trapecio.
  % Sintaxis: I = trapecio(f,a,b)
  % Inputs:
  %   f: función a evaluar
  %   a, b: límites de integración [a, b]
  % Output:
  %   I: aproximación numérica de la integral
  fnum=str2func(['@(x)' f]);
  I=(b-a)*(fnum(b)+fnum(a))/2;
end
```

Al igual que con las otras funciones, se hace una implementación iterativa

```
function I=trapecio_compuesto_iterativa(f,a,b,iterMax,tol);
  % Aproximación numérica de la integral de una función en un intervalo mediante
  % el metodo del trapecio compuesto iterativo, el cual divide en subitervalos y
  % utiliza iteraciones para aplicar el metodo del trapecio a cada intervalo con
  % diferentes cantidades de divisiones hasta hallar la que supere una
  % tolerancia.
  % Sintaxys : trapecio_compuesto(f,a,b)
  % Inputs:
  %    f = funcion a evaluar,
  %    a, b = intervalo [a, b]
  %    iterMax = cantidad de iteraciones maxima
  %    tol = tolerancia permitida del error
  % Outputs:
  %    I =  I = Aproximacion numerica de la integral
  fnum = str2func(['@(x)' f])
  Ir = quad(fnum, a, b);
  I = 0
  for k=1:iterMax
    I=trapecio_compuesto(f,a,b,k);
    error=abs(I-Ir);
    if error<tol
      break
    end
  end
end
```

La regla del trapecio es un método utilizado para aproximar el valor de una integral definida. Este método se basa en la idea de aproximar la integral de una función f(x) mediante el área de un trapecio formado por los puntos $(a,\ f(a))\ y\ (b,\ f(b))$, donde a y b son los límites de integración.

La regla del trapecio compuesta hace la subdivisión del intervalo de integración en n trapecios. Este método se basa en la suposición de que la función f es continua y positiva en el intervalo [a, b], donde a y b son los límites de integración.

La integral definida $\int_a^b f(x)\ dx$ representa el área encerrada por la gráfica de f y el eje **X** desde $x\ =\ a$ hasta $x\ =\ b$. Para aplicar la regla del trapecio compuesta, primero se divide el intervalo [a, b] en n subintervalos de igual ancho $\Delta x\ = \frac{(b-a)}{n}$.

Luego, se aplica la fórmula correspondiente, que se obtiene a través de un proceso matemático, para calcular la aproximación de la integral utilizando la suma de las áreas de los trapecios formados por los puntos de la función en cada subintervalo.

La fórmula resultante será utilizada para realizar los cálculos necesarios y obtener la aproximación de la integral definida mediante la regla del trapecio compuesta.

$$\int_a^b f(x)dx \sim \frac{b-a}{n}\left[\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a+k\frac{b-a}{n}\right)\right]$$

El error en esta aproximación se corresponde con:

$$-\frac{(b-a)^3}{12n^2} f''(\xi)$$

4. Método de Romberg

El Método de Romberg se utiliza para generar una matriz triangular que contiene estimaciones numéricas de una integral definida.
Este Método utiliza la extrapolación de Richardson de forma iterativa, basándose en la regla del trapecio. El proceso consiste en evaluar la función f en puntos equiespaciados dentro del intervalo de integración.

```
function result = romberg(f, a, b, n)
    % Romberg method for numerical integration
    % Inputs:
    %   - f: Function to be integrated
    %   - a: Lower limit of integration
    %   - b: Upper limit of integration
    %   - n: Number of rows in the Romberg table
    % Output:
    %   - result: Approximation of the integral

    % Initialize the Romberg table
    R = zeros(n, n);

    % Compute the step size
    h = (b - a) ./ (2.^(0:n-1));

    % Compute the first column of the Romberg table
    R(1, 1) = (h(1) / 2) * (feval(f, a) + feval(f, b));

    % Use the trapezoidal rule for the remaining columns
    for j = 2:n
        % Compute the composite trapezoidal rule approximation
        sum_f = 0;
        for i = 1:2^(j-2)
            sum_f = sum_f + feval(f, a + (2*i-1)*h(j));
        end
        R(j, 1) = 0.5 * R(j-1, 1) + h(j) * sum_f;

        % Use Richardson extrapolation for the remaining elements
        for k = 2:j
            R(j, k) = R(j, k-1) + (R(j, k-1) - R(j-1, k-1)) / ((4^(k-1)) - 1);
        end
    end

    % Return the final approximation
    result = R(n, n);
end
```

Para que el Método sea efectivo, es necesario que la función f sea suficientemente derivable en el intervalo de integración. Sin embargo, este método puede proporcionar resultados bastante precisos incluso para funciones que no son altamente derivables.

La idea principal es construir una matriz triangular que contenga las estimaciones numéricas de la integral definida utilizando diferentes refinamientos y extrapolaciones sucesivas. Al realizar este proceso iterativo, se obtiene una aproximación más precisa del valor de la integral.

El método se define de forma recursiva de la siguiente manera:

$$R(n,m) = \frac{1}{4^m - 1}(4^m R(n,m-1) - R(n-1,m-1))$$

donde:

$n \geq 1$

$m \geq 1$

$h_n = \frac{b-a}{2^n}.$