Ingeniería en Seguridad Informática y Redes

Ingeniería en Sistemas Computacionales

Algoritmos de Solución Numérica

Función y Gráfica

SI727576 - Edgar Guzmán Claustro

IS727272 - Marco Ricardo Cordero Hernández

Jal., 12 de septiembre de 2023

Contexto

En el ámbito de los algoritmos de solución numérica, se puede encontrar un método primitivo: el método de bisección. Este es un algoritmo numérico utilizado para encontrar las raíces de una función matemática continua en un intervalo dado. Corta por mitad el intervalo, determinando si la raíz verdadera se encuentra a la derecha o a la izquierda de la aproximación. Repitiendo la operación hasta que la diferencia entre las dos últimas aproximaciones sea menor a la tolerancia preestablecida (Cortés Rosas et al., 2019).

En un contexto de aplicación real de este método, podemos considerar como ejemplos la optimización, a menudo es necesario encontrar el valor de una variable que minimice o maximice una función objetivo; cálculo de tasas de interés; En finanzas, se utiliza el método de bisección para calcular tasas de interés efectivas o tasas de rendimiento requeridas en inversiones financieras; modelado y simulación de problemas, donde se necesite determinar el valor de ciertas variables dependiendo de sus condiciones, entre otras.

Dentro de este ejercicio se prueba dicho método numérico utilizando múltiples funciones para demostrar su funcionamiento. Iniciando graficando la función y después haciendo llamada a la función “bisection”, tomando como parámetros y, intervalo inicial, intervalo final y la tolerancia de error. Subsecuentemente, dentro de la función invocada se procede a realizar ciertas validaciones para establecer que el intervalo sea válido y después inicia con el cálculo. Finaliza añadiendo los puntos calculados dentro de la gráfica inicial.

El método por el cual se ha realizado este ejercicio ha sido a través de la herramienta “*Matlab*” (The MathWorks Inc., 2023), cuyo funcionamiento programático ha sido revisado en ocasiones pasadas. El principal punto de interés en esta ocasión, como se ha podido intuir, es la creación, invocación y manipulación de funciones definidas por el usuario o programador. Los valores que la función desarrollada estaría regresando son meramente valores unidimensionales contenidos dentro de un vector, sin embargo, su interpretación gráfica también puede ser demostrada haciendo uso de las mismas utilerías que proveé el software para visualizar resultado.

Resultados

El siguiente código despliega una visualización del método de bisección, mostrando todos los puntos calculados hasta llegar al resultado más acercado al error absoluto definido

Función de bisección

```
function bis = bisection(f, li, ls, e)
%{
   f  = Función a evaluar
   li = Límite inferior (a)
   ls = Límite superior (b)
   e  = Error absoluto
%}

% Variables locales
a = li;                    % Límite por la izquierda
b = ls;                    % Límite por la derecha
le = 1;                    % Error absoluto actual
bis = [];                  % Puntos resultantes
helper = 0;                % Variable de ayuda
iters = 500;               % Iteraciones máximas antes de fallar

% Asegurar intervalo correcto
if (f(a)*f(b) > 0)         % Revisar convergencia inicial
   while (f(a) > 0)        % Modificar límite izquierdo
       a = a + 0.1;

       iters = iters - 1;
       if (iters == 0)
           fprintf('Límite inferior incorrecto.\n');
           return
       end
   end

   iters = 500;

   while (f(b) < 0)        % Modificar límite derecho
       b = b - 0.1;

       iters = iters - 1;
       if (iters == 0)
           fprintf('Límite superior incorrecto.\n');
           return
       end
   end

   if (a > b)              % Invertir límites si se requiere
       helper = a;
```

```
        a = b;
        b = helper;
    end
end

% Proceso de bisección
while (le > e)
    bis(end + 1) = (a + b) / 2;

    % Sustituir límites por nuevo valor
    if (f(bis(end)) < 0)
        a = bis(end);
    else
        b = bis(end);
    end

    % Evaluar error absoluto
    if (length(bis) > 1)
        le = abs(bis(end) - bis(end - 1));
    end
end
```

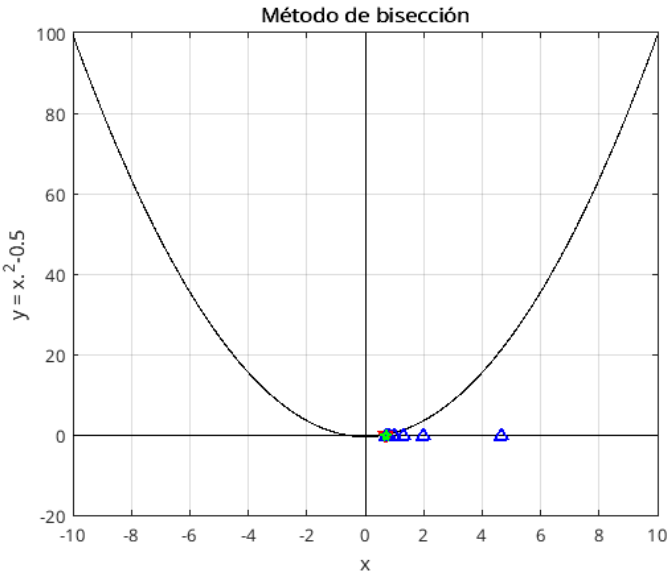
Gráfica de bisección

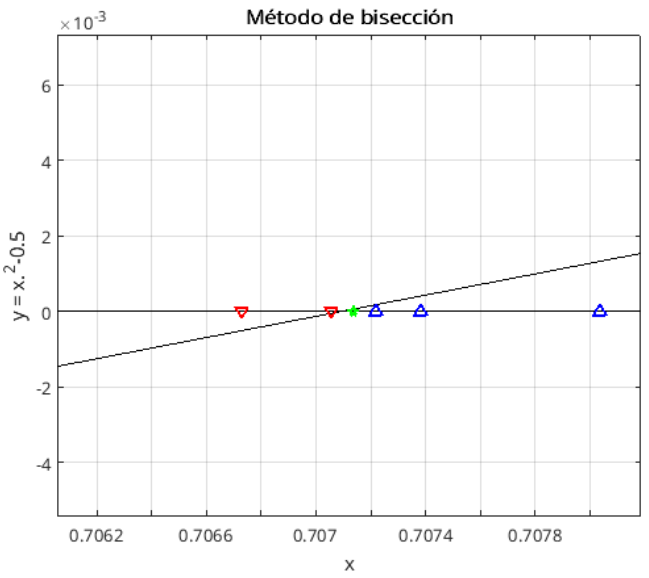
```
% Restablecer entorno
clear, clc, close all

% Definición de parámetros
x = -10 : 0.01 : 10;                        % Dominio predefinido
y = @(x) x.^2 - 0.5;                        % Función con raíces
reales
e = 0.0001;                                 % Tolerancia de
bisección

% Graficar resultados
figure('name','Método de  bisección', ...
    'NumberTitle','off');                   % Título de ventana
plot(x, y(x), 'k')                          % Gráfica inicial
title('Método de bisección')                % Título de la gráfica
xlabel('x')                                 % Etiqueta del eje x
ylabel(strrep(char(y),'@(x)','y = '))       % Etiqueta del eje y
grid on                                     % Cuadrícula de la
gráfica
hold on                                     % Conservar gráfica
plot([0 0], ylim, 'k-')                     % Eje de abscisas
plot(xlim, [0 0], 'k-')                     % Eje de ordenadas

% Visualización de la bisección
points = bisection(y, x(1), x(end), e); % Invocar función
```

```
for p = points                                  % Iterar sobre
resultados
    if (p > points(end))                        % Punto a la izquierda
        plot(p, 0, '^b', 'Linewidth', 2);
    elseif (p < points(end))                    % Punto a la derecha
        plot(p, 0, 'vr', 'Linewidth', 2);
    else                                        % Resultado final
        plot(p, 0, 'pentagramg', 'Linewidth', 2);
    end
end
```

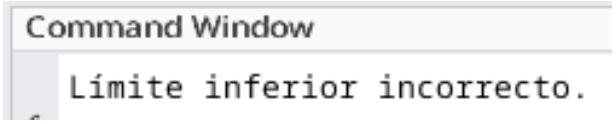
Resultados

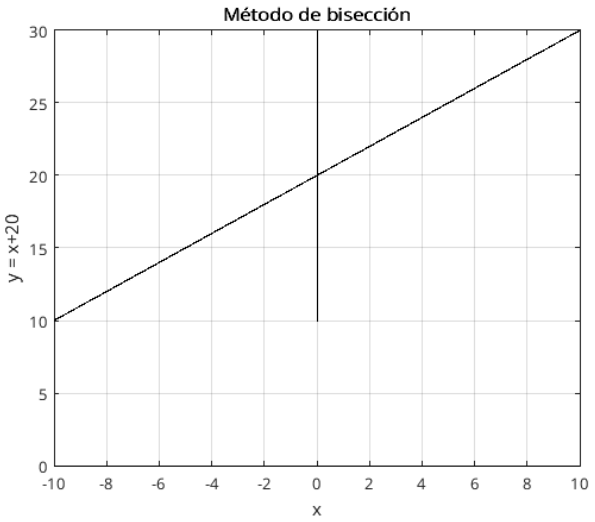
$y = x^2 - 0.5$; e = 0.01%



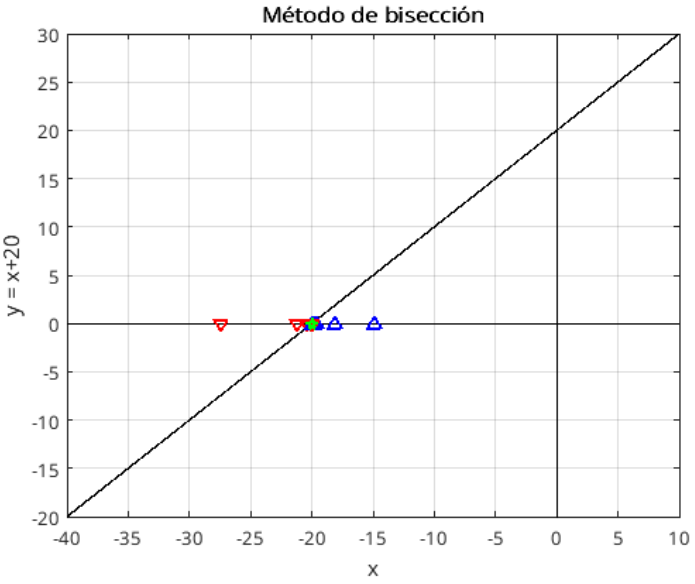


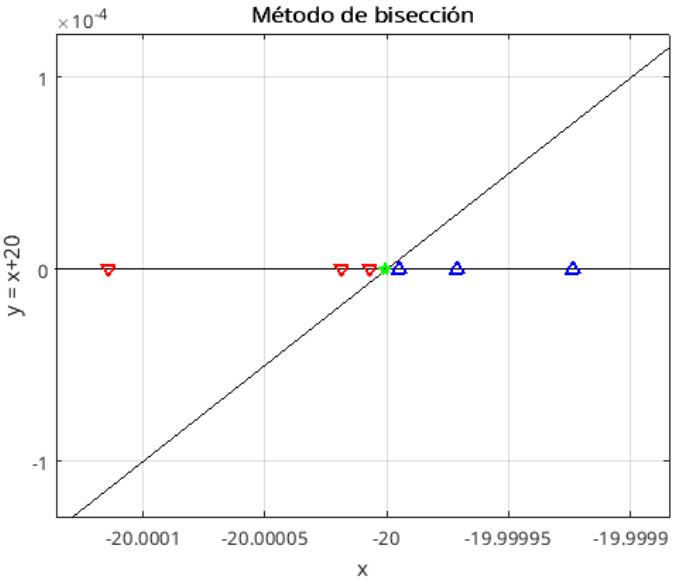
y = x + 20; rango = [-10, 10]





y = x + 20; rango = [-30, 10]; e = 0.001%





Conclusiones

Guzmán Claustro, Edgar

La utilización de Matlab para crear algoritmos de solución numérica es interesante debido a que muestra la gran capacidad que tiene el lenguaje. Dando una ventaja significativa para el usuario al tener incorporadas las gŕaficas, sin la necesidad de tener que programarlas desde cero. Además, quiero destacar que he visto sus usos hasta en la simulación, transmisión y modulación de señales, un tema más apegado a mi área de estudio. Este trabajo se hizo sobre la bisección, me sirvió mucho para recordar de lo que trataba y cómo se hacía.

Cordero Hernández, Marco R.

El ejercicio realizado ha servido para dar cuenta de múltiples factores en relación a métodos numéricos y a la utilización de Matlab. Atendiendo a este último, definir variables y la manipulación y paso de parámetros hacen posible una apertura a un paradigma distinto en cuanto a sintaxis se refiere, conservando la esencia de los lenguajes tradicionales como C, pero permitiendo una (peligrosa) flexibilidad de lenguajes como Javascript o Python. Vale la pena realizar estas comparaciones para una fácil adaptación al nuevo lenguaje revisado, ya que contar con un trasfondo en herramientas como la manejada permite el aprendizaje temprano para una mejor habilidad de desarrollo en el futuro.

Ahora, hablando del método construído, resulta interesante la traslación de conceptos prevalecientes en ámbitos similares como lo podría ser la búsqueda binaria en ciencias computacionales, para ver cómo su utilización también resulta relevante en el campo del análisis numérico de funciones. Finalmente, para unir ambos tópicos, hacer uso del lenguaje de programación revisado permite ir más allá del papel hacía una visualización en tiempo real, lo cual da un valor agregado en cuanto al aspecto didáctico y científico refiere.

Bibliografía

Cortés Rosas, J. J., González Cárdenas, M. E., Pinilla Morán, V. D., et al. (2019). *Métodos cerrados para la solución numérica de ecuaciones algebraicas y trascendentes*. Recuperado de https://www.ingenieria.unam.mx/pinilla/PE105117/pdfs/tema2/2-1_metodos_cerrados.pdf.

The MathWorks Inc. (2022). MATLAB version: 9.14.0 (R2023a), Natick, Massachusetts: The MathWorks Inc. https://www.mathworks.com.