First Report - Numerical Methods and SolverPro

Universidad EAFIT
Computer and Systems Department

Delivery Date of This Report: 04/08/2024

Project Name: SolverPro

Website URL (GitHub repository)

https://github.com/MauricioCa07/SolverPro.git

Team Members

- Mauricio Carrillo Carvajal
- Sebastian Cano
- Manuel Quintero
- Juan Diego Llorente

Objective

To demonstrate the implementation and testing of various numerical methods for root finding and Gaussian elimination.

General Objective

Develop SolverPro, an interactive web platform that allows students, engineers, and professionals to solve complex mathematical problems by implementing various numerical methods, providing accurate real-time results, and promoting the learning of numerical analysis.

Specific Objectives

1. Implement Multiple Numerical Methods: Integrate various numerical methods into SolverPro, such as:
    - Incremental Search

- Bisection
- False Position
- Newton
- Fixed Point
- Multiple Roots
- Secant Method
- Gaussian Elimination
- Partial Pivoting
- Total Pivoting

2. Design an Intuitive User Interface: Create a user-friendly graphical interface that allows users to select and apply numerical methods efficiently.

3. Develop Real-Time Functionality: Ensure SolverPro provides immediate and accurate results, enhancing the user experience.

4. Promote Learning of Numerical Analysis: Provide educational resources and detailed explanations about each numerical method, helping users better understand the techniques and their applications.

5. Implement an English Section: Ensure that the platform is available in English, expanding its accessibility globally.

6. Optimize System Performance and Scalability: Ensure the platform can handle multiple users simultaneously without compromising the speed or accuracy of calculations.

Test Cases for Numerical Methods

All methods will use a tolerance of $\text{Tol} = 10^{-7}$ and a maximum of $N = 100$ iterations. The absolute error will be calculated as $E_n = |x_n - x_{n-1}|$.

Functions to Use

- $f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$
- $f'(x) = 2(\sin^2(x) + 1)^{-1} \sin(x) \cos(x)$
- $f_1(x) = \ln(\sin^2(x) + 1) - \frac{1}{2} - x$
- $g(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$
- $h(x) = e^x - x - 1$
- $h'(x) = e^x - 1$
- $h''(x) = e^x$

Methods and Inputs

- **Incremental Search:**
  - Input: f, $x_0 = -3$, $\Delta x = 0.5$, N
- **Bisection:**
  - Input: f, $a = 0$, $b = 1$, Tol, N
- **False Position:**
  - Input: f, $a = 0$, $b = 1$, Tol, N
- **Newton:**
  - Input: f, f', $x_0 = 0.5$, Tol, N
- **Fixed Point:**
  - Input: f_1, g, $x_0 = -0.5$, Tol, N
- **Secant Method:**
  - Input: f, $x_0 = 0.5$, $x_1 = 1$, Tol, N
- **Multiple Roots:**
  - Input: h, h', h'', $x_0 = 1$, Tol, N
- **Simple Gaussian Elimination:**
  - Input: A, b
- **Gaussian Elimination with Partial Pivoting:**
  - Input: A, b
- **Gaussian Elimination with Total Pivoting:**
  - Input: A, b

Matrixes for Gaussian Elimination

$$A = \begin{pmatrix} 2 & -1 & 0 & 3 \\ 1 & 0.5 & 3 & 8 \\ 0 & 13 & -2 & 11 \\ 14 & 5 & -2 & 3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Pseudocode of Numerical Methods

1. Bisection

Algorithm Bisection Method with Error Handling and Iteration Limit

Input: f, a, b, tol, N_max
Initialize: $fa \leftarrow f(a)$, $pm \leftarrow \frac{a+b}{2}$, $fpm \leftarrow f(pm)$
$E \leftarrow 1000$, $cont \leftarrow 1$
while $E >$ tol **and** $cont <$ N_max **do**
    if $fa \times fpm < 0$ **then**
        $b \leftarrow pm$
    else
        $a \leftarrow pm$
        $fa \leftarrow fpm$ ▷ Update fa when new asigned a
    end if
    $p_0 \leftarrow pm$
    $pm \leftarrow \frac{a+b}{2}$
    $fpm \leftarrow f(pm)$
    $E \leftarrow |pm - p_0|$
    $cont \leftarrow cont + 1$
end while
Print: "Root found at", pm, "in", $cont$, "iterations with an error of:", E
Return: pm

2. False Position

Algorithm False Position Method with Error Handling and Iteration Limit

Input: f, a, b, tol, N_max
Initialize: $fa \leftarrow f(a)$, $fb \leftarrow f(b)$
$pm \leftarrow \frac{fb \cdot a - fa \cdot b}{fb - fa}$
$fpm \leftarrow f(pm)$, $E \leftarrow 1000$, $cont \leftarrow 1$
while $E >$ tol **and** $cont <$ N_max **do**
    if $fa \times fpm < 0$ **then**
        $b \leftarrow pm$
        $fb \leftarrow fpm$
    else
        $a \leftarrow pm$
        $fa \leftarrow fpm$
    end if
    $p_0 \leftarrow pm$
    $pm \leftarrow \frac{fb \cdot a - fa \cdot b}{fb - fa}$
    $fpm \leftarrow f(pm)$
    $E \leftarrow |pm - p_0|$
    $cont \leftarrow cont + 1$
end while
Print: "Root found at", pm, "in", $cont$, "iterations with an error of", E
Return: pm

3. Secant Method

Algorithm Secant Method with Error Handling

Input: $func$, $tolerance$, $iterations$, $X_0^{initial}$, $X_1^{initial}$
Initialize: $X_0 \leftarrow X_0^{initial}$, $X_1 \leftarrow X_1^{initial}$
for $i = 0$ **to** $iterations - 1$ **do**
  $fx0 \leftarrow func(X0)$
  $fx1 \leftarrow func(X1)$
  if $fx1 = 0$ **then**
    Return: "Root found at:", $X1$
  end if
  if $fx1 - fx0 = 0$ **then**
    Return: "Error: Division by zero"
  end if
  $Xn \leftarrow X1 - \dfrac{fx1 \cdot (X1 - X0)}{fx1 - fx0}$
  $error \leftarrow |Xn - X1|$
  if $error < tolerance$ **then**
    Return: "Root found at:", Xn
  end if
  $X0 \leftarrow X1$
  $X1 \leftarrow Xn$
  if $|X1| < 1 \times 10^{-10}$ **then**
    Return: "Error: Xn values are becoming too small"
  end if
  if $|fx1| < 1 \times 10^{-10}$ **then**
    Return: "Error: Function values are approaching zero"
  end if
end for
Return: "No root found after", $iterations$, "iterations."

4. Gaussian Elimination

Algorithm Gaussian Elimination with Back Substitution

```
Input: matrixA, matrixB, n
Initialize: M ← new Array(n)
for i = 0 to n − 1 do
    M[i] ← new Array(n + 1)
    for j = 0 to n − 1 do
        M[i][j] ← A[i][j]
    end for
    M[i][n] ← B[i]
end for
for k = 0 to n − 2 do
    for i = k + 1 to n − 1 do
        ratio ← M[i][k] / M[k][k]
        for j = k to n do
            M[i][j] ← M[i][j] − ratio · M[k][j]
        end for
    end for
end for
X ← new Array(n)
for i = n − 1 to 0 do
    sum ← 0
    for j = i + 1 to n − 1 do
        sum ← sum + M[i][j] · X[j]
    end for
    X[i] ← (M[i][n] − sum) / M[i][i]
end for
Return: X
```

5. Partial Pivoting

Algorithm Gaussian Elimination with Partial Pivoting

Input: A, b, n ▷ A is the coefficient matrix, b is the constant vector, n is the number of variables
Combine A and b into an augmented matrix M
for $k = 1$ **to** $n - 1$ **do**
  Initialize: $\max \leftarrow |M[k,k]|$, maxRow $\leftarrow k$
  for $i = k + 1$ **to** n **do**
    if $|M[i,k]| > \max$ **then**
      $\max \leftarrow |M[i,k]|$
      maxRow $\leftarrow i$
    end if
  end for
  if $\max \approx 0$ **then**
    Throw error: "Singular matrix"
  end if
  Swap: $M[k,*]$ **with** $M[\text{maxRow},*]$ ▷ Swap rows for partial pivoting
  for $i = k + 1$ **to** n **do**
    factor $\leftarrow M[i,k]/M[k,k]$
    for $j = k$ **to** $n + 1$ **do**
      $M[i,j] \leftarrow M[i,j] - \text{factor} \cdot M[k,j]$ ▷ Elimination step
    end for
  end for
end for
Initialize: $x \leftarrow$ new vector of size n
for $i = n$ **to** 1 **(descending) do**
  sum $\leftarrow 0$
  for $j = i + 1$ **to** n **do**
    sum $\leftarrow \text{sum} + M[i,j] \cdot x[j]$
  end for
  $x[i] \leftarrow (M[i, n+1] - \text{sum})/M[i,i]$ ▷ Back-substitution
end for
Return: x

6. Total Pivoting

Algorithm Gaussian Elimination with Total Pivoting

```
Input: A, b, n    ▷ A es la matriz de coeficientes, b es el vector de constantes, n es el número de variables
Combine A y b en una matriz aumentada M
for k = 1 to n − 1 do
    Initialize: max ← 0
    for i = k to n do
        for j = k to n do
            if |M[i, j]| > max then
                max ← |M[i, j]|
                maxRow ← i
                maxCol ← j
            end if
        end for
    end for
    if max ≈ 0 then
        Throw error: "Singular matrix"
    end if
    Swap: M[k, ∗] with M[maxRow, ∗]    ▷ Intercambiar filas
    Swap: M[∗, k] with M[∗, maxCol]    ▷ Intercambiar columnas
    Record the column swap for reordering the final solution
    for i = k + 1 to n do
        factor ← M[i, k]/M[k, k]
        for j = k to n + 1 do
            M[i, j] ← M[i, j] − factor · M[k, j]    ▷ Paso de eliminación
        end for
    end for
end for
Initialize: x ← new vector of size n
for i = n to 1 (descending) do
    sum ← 0
    for j = i + 1 to n do
        sum ← sum + M[i, j] · x[j]
    end for
    x[i] ← (M[i, n + 1] − sum)/M[i, i]    ▷ Sustitución hacia atrás
end for
Reorder x according to the recorded column swaps    ▷ Reordenar según los intercambios de columnas
Return: x
```

7. Newton's Method

Algorithm Newton's Method with Error Handling

```
Entrada: f, f', x_0, tol, max_iter
Inicializar: x ← x_0, iter ← 0, error ← 1
Verificar: Si f(x_0) o f'(x_0) no están definidos en el dominio, lanzar error
Verificar: Si f'(x_0) = 0, lanzar error (no se puede dividir por 0)
while error ¿ tol  y  iter ¡ max_iter do
    f(x) ← evaluar f en x
    f'(x) ← evaluar f' en x
    x_next ← x − f(x)/f'(x)
    error ← |x_next − x|
    if error < tol then
        Imprimir: "Solución encontrada en la iteración", iter
        Retornar: x_next
    else if f(x_next) = 0 then
        Imprimir: "Raíz exacta encontrada en la iteración", iter
        Retornar: x_next
    else if iter = max_iter then
        Imprimir: "No se encontró solución en el número máximo de iteraciones"
        Retornar: None
    end if
    x ← x_next
    iter ← iter + 1
end while
Imprimir: "It was impossible to find the root within", max_iter, "iterations"
Retornar: None
```

8. Fixed Point

Algorithm Fixed Point Method with Error Handling and Iteration Limit

Entrada: f, g, x_0, tol, max_iter
Inicializar: $x \leftarrow x_0$, iter $\leftarrow 0$, error $\leftarrow 1$
Verificar: Si $f(x_0)$ o $g(x_0)$ no están definidos en el dominio, lanzar error
Verificar: Si $f \equiv g$, lanzar error (no pueden ser iguales)
while error ¿ tol **y** iter ¡ max_iter **do**
$x_{\text{next}} \leftarrow g(x)$
$error \leftarrow |x_{\text{next}} - x|$
if $error < tol$ **then**
Imprimir: "Solución encontrada en la iteración", iter
Retornar: x_{next}
else if $f(x_{\text{next}}) = 0$ **then**
Imprimir: "Raíz exacta encontrada en la iteración", iter
Retornar: x_{next}
else if iter = max_iter **then**
Imprimir: "No se encontró solución en el número máximo de iteraciones"
Retornar: None
end if
$x \leftarrow x_{\text{next}}$
iter $\leftarrow$ iter + 1
end while
Imprimir: "It was impossible to find the root within", max_iter, "iterations"
Retornar: None

9. Incremental searches

Algorithm Incremental Search Method with Iteration Limit

Input: f, x_0, h, N_max
Initialize: $x_{\text{inf}} \leftarrow x_0$, $x_{\text{sup}} \leftarrow x_{\text{inf}} + h$
$y_{\text{inf}} \leftarrow f(x_{\text{inf}})$, $y_{\text{sup}} \leftarrow f(x_{\text{sup}})$
for $i = 1$ **until** N_max **do**
if $y_{\text{inf}} \times y_{\text{sup}} \leq 0$ **then**
Print: "Root at the interval between", x_{inf}, "y", x_{sup}, "in iteration", i
Return: x_{inf}, x_{sup}
end if
$x_{\text{inf}} \leftarrow x_{\text{sup}}$
$y_{\text{inf}} \leftarrow y_{\text{sup}}$
$x_{\text{sup}} \leftarrow x_{\text{inf}} + h$
$y_{\text{sup}} \leftarrow f(x_{\text{sup}})$
end for
if $i > N\_max$ **then**
Print: "Could not find sign switch on maximum iterations"
Return: None
end if

article amsmath

10. Multiple Roots Method (Euler-Chebyshev Method)

Algorithm Multiple Roots Method

Input: f, f', f'', x_0, ϵ, N_max
Initialize: $n \leftarrow 0$
if $|f(x_0)| < \epsilon$ **then**
    Return: x_0 ▷ x_0 is already a root
end if
while $n <$ N_max **do**
    $f_x \leftarrow f(x_0)$
    $f'_x \leftarrow f'(x_0)$
    $f''_x \leftarrow f''(x_0)$
    $x_1 \leftarrow x_0 - \frac{f_x \cdot f'_x}{(f'_x)^2 - f_x \cdot f''_x}$ ▷ Update x using the Euler-Chebyshev formula
    if $|x_1 - x_0| < \epsilon$ **or** $|f(x_1)| < \epsilon$ **then**
        Return: x_1 ▷ Convergence achieved
    end if
    $x_0 \leftarrow x_1$
    $n \leftarrow n + 1$
end while
Print: "Maximum iterations reached without convergence"
Return: None