University Assignment

Team members :

Alverth Antonio De La Barrera Moreno
Juan Pablo Restrepo Restrepo
Pablo Soto Calle

Software: Python, Go

Repo: https://github.com/Psotoc112/NumericalMethodsBack

Repo: https://github.com/Psotoc112/NM-API

Introduction :

This project aims to develop a web application that allows users to select various numerical methods for problem-solving. In this initial phase, our focus will be on implementing the underlying logic of these numerical methods using Python and Go. By creating a robust foundation, we will ensure that the subsequent web interface can effectively utilize these methods to provide accurate and efficient solutions.

Simultaneously this document outlines the design and implementation of a numeric analysis application developed as part of the course "Numeric Analysis." The application implements various root-finding methods, including False Position, Multiple Roots, Newton-Raphson, Incremental Search, Fixed Point, Secant, and Bisection. The system is divided into frontend, backend, and a Go service that handles core calculations.

1 Method Descriptions and Results

This section provides detailed descriptions of the root-finding methods implemented in the application. Each method includes the relevant mathematical functions, their derivatives, and the required input values used in the system.

1.1 Incremental Search

Description

Incremental search is a simple method used to find an interval where the function changes sign, indicating the presence of a root.

Pseudocode

```
Procedure Incremental_Search_Method
    INPUT: f (function), x0 (initial value),  x  (step size), TOL (tolerance),
       Max_Iterations (maximum iterations)

    SET iteration_count to 0
    SET x1 to x0 +  x

    WHILE iteration_count < Max_Iterations DO
        COMPUTE f_x0 as f(x0)
        COMPUTE f_x1 as f(x1)

        IF f_x0 * f_x1 < 0 THEN
            PRINT "Sign change detected between x0 and x1"
            RETURN (x0, x1)
        END IF

        SET x0 to x1
        SET x1 to x0 +  x
        INCREMENT iteration_count
    END WHILE

    PRINT "No root found"
END Procedure
```

Listing 1: Incremental Search Method

Function

$$f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$$

Input Values

- Initial guess: $x_0 = -3$
- Step size: $\Delta x = 0.5$
- Number of iterations: 100

Results

```
Iteration 0: x = -3, f(x) = -0.4802808500361744
Iteration 1: x = -2.5, f(x) = -0.1938625991661741
Iteration 2: x = -2, f(x) = 0.10257774140337728, There is a root between -2.5 and -2
Iteration 3: x = -1.5, f(x) = 0.19064216978879167
Iteration 4: x = -1, f(x) = 0.03536607938024017
Iteration 5: x = -0.5, f(x) = -0.2931087267313766, There is a root between -1 and -0.5
Iteration 6: x = 0, f(x) = -0.5
Iteration 7: x = 0.5, f(x) = -0.2931087267313766
Iteration 8: x = 1, f(x) = 0.03536607938024017, There is a root between 0.5 and 1
Iteration 9: x = 1.5, f(x) = 0.19064216978879167
Iteration 10: x = 2, f(x) = 0.10257774140337728
Iteration 11: x = 2.5, f(x) = -0.1938625991661741, There is a root between 2 and 2.5
Iteration 12: x = 3, f(x) = -0.4802808500361744
Iteration 13: x = 3.5, f(x) = -0.3839528053078892
Iteration 14: x = 4, f(x) = -0.04717430978375031
Iteration 15: x = 4.5, f(x) = 0.17067922120050372, There is a root between 4 and 4.5
Iteration 16: x = 5, f(x) = 0.15208336750093676
Iteration 17: x = 5.5, f(x) = -0.09601121378159777, There is a root between 5 and 5.5
Iteration 18: x = 6, f(x) = -0.4248247926701879
Iteration 19: x = 6.5, f(x) = -0.45476222450315373
Iteration 20: x = 7, f(x) = -0.1411853731889448
Iteration 21: x = 7.5, f(x) = 0.13118877149775998, There is a root between 7 and 7.5
```

1.2 Bisection Method

Description

The Bisection method is a bracketing method that repeatedly divides the interval in half to converge on a root.

Pseudocode

```
Procedure Bisection_Method
    INPUT: f (function), a (lower bound), b (upper bound), TOL (tolerance),
       Max_Iterations (maximum iterations)

    IF f(a) * f(b)     0 THEN
        PRINT "No root in this interval"
        EXIT
    END IF

    SET iteration_count to 0
    SET error to b - a

    WHILE iteration_count < Max_Iterations AND error > TOL DO
        SET c to (a + b) / 2
        COMPUTE f_c as f(c)

        IF ABS(f_c) < TOL THEN
            PRINT "Root found at c"
            RETURN c
        END IF

        IF f(a) * f_c < 0 THEN
            SET b to c
        ELSE
            SET a to c
        END IF

        SET error to b - a
        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 2: Bisection Method

Function

$$f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$$

Input Values

- Interval: $a = 0$, $b = 1$
- Tolerance: 10^{-7}

- Number of iterations: 100

Results

```
Iteration 1: a = 0.50000000000, b = 1.00000000000, c = 0.50000000000, f(c) = -0.29310872673, Error = 1.00000010000
Iteration 2: a = 0.75000000000, b = 1.00000000000, c = 0.75000000000, f(c) = -0.11839639385, Error = 0.25000000000
Iteration 3: a = 0.87500000000, b = 1.00000000000, c = 0.87500000000, f(c) = -0.03681769076, Error = 0.12500000000
Iteration 4: a = 0.87500000000, b = 0.93750000000, c = 0.93750000000, f(c) = 0.00063391616, Error = 0.06250000000
Iteration 5: a = 0.90625000000, b = 0.93750000000, c = 0.90625000000, f(c) = -0.01777228923, Error = 0.03125000000
Iteration 6: a = 0.92187500000, b = 0.93750000000, c = 0.92187500000, f(c) = -0.00848658221, Error = 0.01562500000
Iteration 7: a = 0.92968750000, b = 0.93750000000, c = 0.92968750000, f(c) = -0.00390535863, Error = 0.00781250000
Iteration 8: a = 0.93359375000, b = 0.93750000000, c = 0.93359375000, f(c) = -0.00163043812, Error = 0.00390625000
Iteration 9: a = 0.93554687500, b = 0.93750000000, c = 0.93554687500, f(c) = -0.00049693532, Error = 0.00195312500
Iteration 10: a = 0.93554687500, b = 0.93652343750, c = 0.93652343750, f(c) = 0.00006882244, Error = 0.00097656250
Iteration 11: a = 0.93603515625, b = 0.93652343750, c = 0.93603515625, f(c) = -0.00021397351, Error = 0.00048828125
Iteration 12: a = 0.93627929688, b = 0.93652343750, c = 0.93627929688, f(c) = -0.00007255479, Error = 0.00024414062
Iteration 13: a = 0.93640136719, b = 0.93652343750, c = 0.93640136719, f(c) = -0.00000186098, Error = 0.00012207031
Iteration 14: a = 0.93640136719, b = 0.93646240234, c = 0.93646240234, f(c) = 0.00003348203, Error = 0.00006103516
Iteration 15: a = 0.93640136719, b = 0.93643188477, c = 0.93643188477, f(c) = 0.00001581085, Error = 0.00003051758
Iteration 16: a = 0.93640136719, b = 0.93641662598, c = 0.93641662598, f(c) = 0.00000697501, Error = 0.00001525879
Iteration 17: a = 0.93640136719, b = 0.93640899658, c = 0.93640899658, f(c) = 0.00000255703, Error = 0.00000762939
Iteration 18: a = 0.93640136719, b = 0.93640518188, c = 0.93640518188, f(c) = 0.00000034803, Error = 0.00000381470
Iteration 19: a = 0.93640327454, b = 0.93640518188, c = 0.93640327454, f(c) = -0.00000075648, Error = 0.00000190735
Iteration 20: a = 0.93640422821, b = 0.93640518188, c = 0.93640422821, f(c) = -0.00000020422, Error = 0.00000095367
Converged after 21 iterations
Result: 0.9364047050476074
```

1.3 False Position Method

Description

False position is another bracketing method that uses linear interpolation to approximate the root.

Pseudocode

```
Procedure False_Position_Method
    INPUT: f (function), a (lower bound), b (upper bound), TOL (tolerance),
       Max_Iterations (maximum iterations)

    IF f(a) * f(b)     0 THEN
        PRINT "No root in this interval"
        EXIT
    END IF

    SET iteration_count to 0
    SET error to ABS(b - a)

    WHILE iteration_count < Max_Iterations AND error > TOL DO
        COMPUTE c as b - f(b) * (b - a) / (f(b) - f(a))
        COMPUTE f_c as f(c)

        IF ABS(f_c) < TOL THEN
            PRINT "Root found at c"
            RETURN c
        END IF

        IF f(a) * f_c < 0 THEN
            SET b to c
        ELSE
            SET a to c
        END IF

        SET error to ABS(b - a)
        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 3: False Position Method

Function

$$f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$$

Input Values

- Interval: $a = 0$, $b = 1$
- Tolerance: 10^{-7}

- Number of iterations: 100

Results

```
Iteration 1: a = 0.00000000000, b = 1.00000000000, c = 0.93394038072, f(c) = -0.00142907670, Error = 1.00000010000
Iteration 2: a = 0.93394038072, b = 1.00000000000, c = 0.93650605167, f(c) = 0.00005875601, Error = 0.00256567095
Iteration 3: a = 0.93394038072, b = 0.93650605167, c = 0.93640473074, f(c) = 0.00000008678, Error = 0.00010132092
Iteration 4: a = 0.93394038072, b = 0.93640473074, c = 0.93640458110, f(c) = 0.00000000013, Error = 0.00000014964
Iteration 5: a = 0.93394038072, b = 0.93640458110, c = 0.93640458088, f(c) = 0.00000000000, Error = 0.00000000022
Converged after 5 iterations
Result: 0.9364045808798892
```

1.4 Newton-Raphson Method

Description

Newton-Raphson is an iterative method that uses the function's derivative to find roots more rapidly.

Pseudocode

```
Procedure Newton_Method
    INPUT: f (function), f' (derivative of f), x0 (initial guess), TOL (
        tolerance), Max_Iterations (maximum iterations)

    SET iteration_count to 0
    SET error to TOL + 1

    WHILE iteration_count < Max_Iterations AND error > TOL DO
        COMPUTE f_x0 as f(x0)
        COMPUTE f'_x0 as f'(x0)

        IF f'_x0 = 0 THEN
            PRINT "Division by zero error"
            EXIT
        END IF

        SET x1 to x0 - f_x0 / f'_x0
        SET error to ABS(x1 - x0)

        IF error < TOL THEN
            PRINT "Root found at x1"
            RETURN x1
        END IF

        SET x0 to x1
        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 4: Newton-Raphson Method

Function

$$f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$$

Derivative

$$f'(x) = 2(\sin^2(x) + 1)^{-1} \sin(x) \cos(x)$$

Input Values

- Initial guess: $x_0 = 0.5$

- Tolerance: 10^{-7}
- Number of iterations: 100

Results

```
Iteration 1: x = 0.50000000000, f(x) = -0.29310872673, Error = 1.00000010000
Iteration 2: x = 0.92839198991, f(x) = -0.00466215710, Error = 0.42839198991
Iteration 3: x = 0.93636674127, f(x) = -0.00002191262, Error = 0.00797475135
Iteration 4: x = 0.93640458002, f(x) = -0.00000000050, Error = 0.00003783875
Iteration 5: x = 0.93640458088, f(x) = 0.00000000000, Error = 0.00000000086
Converged after 5 iterations
Result: 0.9364045808795624
```

1.5 Fixed Point Method

Description

The fixed point method rearranges the equation to express it as $x = g(x)$, then iteratively solves for x.

Pseudocode

```
Procedure Fixed_Point_Method
    INPUT: g (rearranged function), x0 (initial guess), TOL (tolerance),
       Max_Iterations (maximum iterations)

    SET iteration_count to 0
    SET error to TOL + 1

    WHILE iteration_count < Max_Iterations AND error > TOL DO
        SET x1 to g(x0)
        SET error to ABS(x1 - x0)

        IF error < TOL THEN
            PRINT "Root found at x1"
            RETURN x1
        END IF

        SET x0 to x1
        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 5: Fixed Point Method

Functions

$$f_1(x) = \ln(\sin^2(x) + 1) - \frac{1}{2} - x$$

$$g(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$$

Input Values

- Initial guess: $x_0 = -0.5$
- Tolerance: 10^{-7}
- Number of iterations: 100

Results

```
Iteration 3: x = -0.41982154361, f(x) = 0.07351702443, g(x) = -0.34630451918, Error = 0.05319579245
Iteration 4: x = -0.34630451918, f(x) = -0.04465393736, g(x) = -0.39095845654, Error = 0.02886308706
Iteration 5: x = -0.39095845654, f(x) = 0.02655342165, g(x) = -0.36440503489, Error = 0.01810051572
Iteration 6: x = -0.36440503489, f(x) = -0.01602126827, g(x) = -0.38042630317, Error = 0.01053215337
Iteration 7: x = -0.38042630317, f(x) = 0.00958950789, g(x) = -0.37083679528, Error = 0.00643176039
Iteration 8: x = -0.37083679528, f(x) = -0.00576885008, g(x) = -0.37660564536, Error = 0.00382065780
Iteration 9: x = -0.37660564536, f(x) = 0.00346022776, g(x) = -0.37314541761, Error = 0.00230862233
Iteration 10: x = -0.37314541761, f(x) = -0.00207922358, g(x) = -0.37522464119, Error = 0.00138100418
Iteration 11: x = -0.37522464119, f(x) = 0.00124805514, g(x) = -0.37397658605, Error = 0.00083116844
Iteration 12: x = -0.37397658605, f(x) = -0.00074962966, g(x) = -0.37472621571, Error = 0.00049842548
Iteration 13: x = -0.37472621571, f(x) = 0.00045008240, g(x) = -0.37427613331, Error = 0.00029954726
Iteration 14: x = -0.37427613331, f(x) = -0.00027029515, g(x) = -0.37454642846, Error = 0.00017978725
Iteration 15: x = -0.37454642846, f(x) = 0.00016230202, g(x) = -0.37438412643, Error = 0.00010799312
Iteration 16: x = -0.37438412643, f(x) = -0.00009746440, g(x) = -0.37448159083, Error = 0.00006483763
Iteration 17: x = -0.37448159083, f(x) = 0.00005852565, g(x) = -0.37442306518, Error = 0.00003893875
Iteration 18: x = -0.37442306518, f(x) = -0.00003514468, g(x) = -0.37445820986, Error = 0.00002338097
Iteration 19: x = -0.37445820986, f(x) = 0.00002110401, g(x) = -0.37443710585, Error = 0.00001404067
Iteration 20: x = -0.37443710585, f(x) = -0.00001267288, g(x) = -0.37444977873, Error = 0.00000843114
Iteration 21: x = -0.37444977873, f(x) = 0.00000760996, g(x) = -0.37444216876, Error = 0.00000506291
Iteration 22: x = -0.37444216876, f(x) = -0.00000456974, g(x) = -0.37444673851, Error = 0.00000304022
Iteration 23: x = -0.37444673851, f(x) = 0.00000274410, g(x) = -0.37444399441, Error = 0.00000182564
Iteration 24: x = -0.37444399441, f(x) = -0.00000164781, g(x) = -0.37444564222, Error = 0.00000109628
Iteration 25: x = -0.37444564222, f(x) = 0.00000098950, g(x) = -0.37444465272, Error = 0.00000065831
Iteration 26: x = -0.37444465272, f(x) = -0.00000059419, g(x) = -0.37444524691, Error = 0.00000039531
Iteration 27: x = -0.37444524691, f(x) = 0.00000035681, g(x) = -0.37444489010, Error = 0.00000023738
Iteration 28: x = -0.37444489010, f(x) = -0.00000021426, g(x) = -0.37444510436, Error = 0.00000014255
Iteration 29: x = -0.37444510436, f(x) = 0.00000012866, g(x) = -0.37444497570, Error = 0.00000008560
Converged after 29 iterations with g(x) = ln(sin(x)^2 + 1) - 1/2
Result: -0.3744449757003151
```

1.6 Secant Method

Description

The secant method approximates the derivative by using two points and uses them to iteratively find the root.

Pseudocode

```
Procedure Secant_Method
    INPUT: f (function), x0 (initial guess 1), x1 (initial guess 2), TOL (
        tolerance), Max_Iterations (maximum iterations)

    SET iteration_count to 0
    SET error to ABS(x1 - x0)

    WHILE iteration_count < Max_Iterations AND error > TOL DO
        COMPUTE f_x0 as f(x0)
        COMPUTE f_x1 as f(x1)

        IF f_x0 = f_x1 THEN
            PRINT "Division by zero error"
            EXIT
        END IF

        SET x2 to x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)
        SET error to ABS(x2 - x1)

        IF error < TOL THEN
            PRINT "Root found at x2"
            RETURN x2
        END IF

        SET x0 to x1
        SET x1 to x2
        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 6: Secant Method

Function

$$f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$$

Input Values

- Initial guesses: $x_0 = 0.5$, $x_1 = 1$
- Tolerance: 10^{-7}
- Number of iterations: 100

Results

```
Iteration 0: x = 0.5, f(x) = -0.2931087267313766
Iteration 1: x = 1, f(x) = 0.03536607938024017
Iteration 2: x = 0.946166222306525, f(x) = 0.005619392737863826, error = 0.05383377769347497
Iteration 3: x = 0.9359965807911726, f(x) = -0.00023632217470059835, error = 0.010169641515352379
Iteration 4: x = 0.9364070023767039, f(x) = 1.4022358909571153e-06, error = 0.00041042158553128427
Iteration 5: x = 0.9364045814731197, f(x) = 3.4371649970665885e-10, error = 2.420903584265943e-06
Converged after 5 iterations
Result: 0.9364045808795616
```

1.7 Multiple Roots Method

Description

This method is used when a function has multiple roots close to each other or repeated roots.

Pseudocode

```
Procedure Multiple_Roots_Method
    INPUT: f (function), f' (first derivative), f'' (second derivative), x0 (
       initial guess), TOL (tolerance), Max_Iterations (maximum iterations)

    SET iteration_count to 0
    SET error to TOL + 1

    WHILE iteration_count < Max_Iterations AND error > TOL DO
        COMPUTE f_x0 as f(x0)
        COMPUTE f'_x0 as f'(x0)
        COMPUTE f''_x0 as f''(x0)

        IF (f'_x0)^2 - f_x0 * f''_x0 = 0 THEN
            PRINT "Division by zero error"
            EXIT
        END IF

        SET x1 to x0 - (f_x0 * f'_x0) / ((f'_x0)^2 - f_x0 * f''_x0)
        SET error to ABS(x1 - x0)

        IF error < TOL THEN
            PRINT "Root found at x1"
            RETURN x1
        END IF

        SET x0 to x1
        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 7: Multiple Roots Method

Function

$$h(x) = e^x - x - 1$$

Derivatives

$$h'(x) = e^x - 1$$
$$h''(x) = e^x$$

Input Values

- Initial guess: $x_0 = 1$
- Tolerance: 10^{-7}
- Number of iterations: 100

Results

```
Iteration 0: x = 1, f(x) = 0.7182818284590451, f'(x) = 1.718281828459045, f''(x) = 2.718281828459045, error = 1
Iteration 1: x = -0.23421061355351425, f(x) = 0.025405775475345838, f'(x) = -0.20880483807816852, f''(x) = 0.7911951619218315, error = 0
Iteration 2: x = -0.00845827991076109, f(x) = 3.567060801401567e-05, f'(x) = -0.008422609302746964, f''(x) = 0.991577390697253, error = 0
Iteration 3: x = -1.1890183808588653e-05, f(x) = 7.068789997788372e-11, f'(x) = -1.1890113120638368e-05, f''(x) = 0.9999881098868794, error = 0
Iteration 4: x = -4.218590698935789e-11, f(x) = 0, f'(x) = -4.218592142279931e-11, f''(x) = 0.9999999999578141, error = 0
Converged after 5 iterations
Result: -0.0000000000421859
```

1.8 Simple Gaussian Elimination

Description

Simple Gaussian Elimination is a method for solving a system of linear equations by reducing the system to an upper triangular matrix without pivoting. The system is then solved using backward substitution.

Pseudocode

```
Procedure Simple_Gaussian_Elimination
    INPUT: A (coefficient matrix), b (right-hand side vector)

    FOR k = 1 to n - 1 DO
        FOR i = k + 1 to n DO
            COMPUTE factor as A[i,k] / A[k,k]

            FOR j = k to n DO
                A[i,j] = A[i,j] - factor * A[k,j]
            END FOR

            b[i] = b[i] - factor * b[k]
        END FOR
    END FOR

    Perform Backward_Substitution(A, b)
END Procedure
```

Listing 8: Simple Gaussian Elimination

Input Values

Input Values

- Coefficient matrix A

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

- Right-hand side vector b

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Results

```
Matriz Aumentada Inicial:
 [[ 2.   -1.    0.    3.    1. ]
 [ 1.    0.5  3.    8.    1. ]
 [ 0.   13.   -2.   11.    1. ]
 [14.    5.   -2.    3.    1. ]]

Matriz intermedia después de la eliminación en la columna 1:
 [[  2.    -1.     0.     3.     1. ]
 [  0.     1.     3.     6.5    0.5]
 [  0.    13.    -2.    11.     1. ]
 [  0.    12.    -2.   -18.    -6. ]]

Matriz intermedia después de la eliminación en la columna 2:
 [[  2.    -1.     0.     3.     1. ]
 [  0.     1.     3.     6.5    0.5]
 [  0.     0.   -41.   -73.5   -5.5]
 [  0.     0.   -38.   -96.   -12. ]]

Matriz intermedia después de la eliminación en la columna 3:
 [[  2.          -1.           0.           3.           1.        ]
 [  0.           1.           3.           6.5          0.5       ]
 [  0.           0.         -41.         -73.5         -5.5       ]
 [  0.           0.           0.         -27.87804878  -6.90243902]]

Matriz intermedia después de la eliminación en la columna 4:
 [[  2.          -1.           0.           3.           1.        ]
 [  0.           1.           3.           6.5          0.5       ]
 [  0.           0.         -41.         -73.5         -5.5       ]
 [  0.           0.           0.         -27.87804878  -6.90243902]]

Matriz Triangular Superior:
 [[  2.          -1.           0.           3.           1.        ]
 [  0.           1.           3.           6.5          0.5       ]
 [  0.           0.         -41.         -73.5         -5.5       ]
 [  0.           0.           0.         -27.87804878  -6.90243902]]

Soluciones finales del sistema:
x1 = 0.0385
x2 = -0.1802
x3 = -0.3097
x4 = 0.2476
```

1.9 Gaussian Elimination with Partial Pivoting

Description

Gaussian elimination with partial pivoting improves numerical stability by swapping rows to ensure the largest pivot element in the current column is used for elimination.

Pseudocode

```
Procedure Gaussian_Elimination_Partial_Pivoting
    INPUT: A (coefficient matrix), b (right-hand side vector)

    FOR k = 1 to n - 1 DO
        Find the row with the largest absolute value in column k, starting from
             row k
        Swap rows if necessary

        FOR i = k + 1 to n DO
            COMPUTE factor as A[i,k] / A[k,k]

            FOR j = k to n DO
                A[i,j] = A[i,j] - factor * A[k,j]
            END FOR

            b[i] = b[i] - factor * b[k]
        END FOR
    END FOR

    Perform Backward_Substitution(A, b)
END Procedure
```

Listing 9: Gaussian Elimination with Partial Pivoting

Input Values

- Coefficient matrix A

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

- Right-hand side vector b

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Results

```
Ingrese el número de filas y columnas (matriz cuadrada): 4
Ingrese los elementos de la fila 1 separados por espacio: 2 -1 0 3
Ingrese los elementos de la fila 2 separados por espacio: 1 0.5 3 8
Ingrese los elementos de la fila 3 separados por espacio: 0 13 -2 11
Ingrese los elementos de la fila 4 separados por espacio: 14 5 -2 3

Ingrese el vector de términos independientes (b) separados por espacio:
1 1 1 1
Matriz Aumentada:
 [[ 2.  -1.   0.   3.   1. ]
 [ 1.   0.5  3.   8.   1. ]
 [ 0.  13.  -2.  11.   1. ]
 [14.   5.  -2.   3.   1. ]]

Intercambio de fila 1 con fila 4 con pivoteo parcial.

Matriz intermedia después de la eliminación en la columna 1:
 [[14.          5.          -2.          3.          1.        ]
 [ 0.          0.14285714  3.14285714  7.78571429  0.92857143]
 [ 0.         13.         -2.         11.          1.        ]
 [ 0.         -1.71428571  0.28571429  2.57142857  0.85714286]]
```

```
Intercambio de fila 2 con fila 3 con pivoteo parcial.

Matriz intermedia después de la eliminación en la columna 2:
 [[ 1.40000000e+01  5.00000000e+00 -2.00000000e+00  3.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01 -2.00000000e+00  1.10000000e+01
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.16483516e+00  7.66483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  2.19780220e-02  4.02197802e+00
   9.89010989e-01]]

Matriz intermedia después de la eliminación en la columna 3:
 [[ 1.40000000e+01  5.00000000e+00 -2.00000000e+00  3.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01 -2.00000000e+00  1.10000000e+01
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.16483516e+00  7.66483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  0.00000000e+00  3.96875000e+00
   9.82638889e-01]]
```

```
Matriz intermedia después de la eliminación en la columna 4:
 [[ 1.40000000e+01  5.00000000e+00 -2.00000000e+00  3.00000000e+00
    1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01 -2.00000000e+00  1.10000000e+01
    1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.16483516e+00  7.66483516e+00
    9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  0.00000000e+00  3.96875000e+00
    9.82638889e-01]]

Matriz Triangular Superior:
 [[ 1.40000000e+01  5.00000000e+00 -2.00000000e+00  3.00000000e+00
    1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01 -2.00000000e+00  1.10000000e+01
    1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  3.16483516e+00  7.66483516e+00
    9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  0.00000000e+00  3.96875000e+00
    9.82638889e-01]]

Soluciones finales del sistema:
x1 = 0.0385
x2 = -0.1802
x3 = -0.3097
x4 = 0.2476
```

1.10 Gaussian Elimination with Full Pivoting

Description

Full pivoting further improves numerical accuracy by swapping both rows and columns based on the largest absolute value in the submatrix.

Pseudocode

```
Procedure Gaussian_Elimination_Full_Pivoting
    INPUT: A (coefficient matrix), b (right-hand side vector)

    FOR k = 1 to n - 1 DO
        Find the largest absolute value in the submatrix starting at A[k,k]
        Swap rows and columns accordingly

        FOR i = k + 1 to n DO
            COMPUTE factor as A[i,k] / A[k,k]

            FOR j = k to n DO
                A[i,j] = A[i,j] - factor * A[k,j]
            END FOR

            b[i] = b[i] - factor * b[k]
        END FOR
    END FOR

    Perform Backward_Substitution(A, b)
    Adjust solution based on column swaps
END Procedure
```

Listing 10: Gaussian Elimination with Full Pivoting

Input Values

- Coefficient matrix A

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

- Right-hand side vector b

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Results

```
Matriz Triangular Superior:
 [[ 1.40000000e+01  5.00000000e+00  3.00000000e+00 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01  1.10000000e+01 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  7.66483516e+00  3.16483516e+00
 [[ 1.40000000e+01  5.00000000e+00  3.00000000e+00 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01  1.10000000e+01 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  7.66483516e+00  3.16483516e+00
   1.00000000e+00]
 [ 0.00000000e+00  1.30000000e+01  1.10000000e+01 -2.00000000e+00
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  7.66483516e+00  3.16483516e+00
   1.00000000e+00]
 [ 0.00000000e+00  0.00000000e+00  7.66483516e+00  3.16483516e+00
 [ 0.00000000e+00  0.00000000e+00  7.66483516e+00  3.16483516e+00
   9.17582418e-01]
 [ 0.00000000e+00  2.22044605e-16  0.00000000e+00 -1.63870968e+00
   5.07526882e-01]]

Soluciones finales del sistema (ordenadas):
x1 = 0.0385
x2 = -0.1802
x3 = -0.3097
x4 = 0.2476
```

1.11 LU (simple)

Input Values

image.png

```
funcon LU_simple(A):
    n = tama o de la matriz A
    Crear matriz L con ceros de tama o n x n
    Crear matriz U con ceros de tama o n x n

    para i desde 0 hasta n-1:

        L[i][i] = 1
        para j desde i hasta n-1:
            U[i][j] = A[i][j] - suma(L[i][k] * U[k][j] para k desde 0 hasta i)
```

```

        para j desde i+1 hasta n-1:
            L[j][i] = (A[j][i] - suma(L[j][k] * U[k][i] para k desde 0 hasta i)) / U[i][i]

    retornar L, U

funci n sustitucion_progresiva(L, b):
    n = tama o de b
    Crear arreglo y con ceros de tama o n

    para i desde 0 hasta n-1:
        y[i] = b[i] - suma(L[i][j] * y[j] para j desde 0 hasta i)

    retornar y

funci n sustitucion_regresiva(U, y):
    n = tama o de y
    Crear arreglo x con ceros de tama o n

    para i desde n-1 hasta 0:
        x[i] = (y[i] - suma(U[i][j] * x[j] para j desde i+1 hasta n-1)) / U[i][i]
    retornar x
```

Results

results/LUsimple.png

1.12 LU (partial pivot)

Input Values

image.png

```
funcion LU_pivoteo(A):
    n = tama o de la matriz A
    Crear matriz P con unos en la diagonal (matriz identidad de tama o n x n)
    Crear matrices L y U con ceros de tama o n x n

    para i desde 0 hasta n-1:
        # Encontrar el  ndice  del pivote m ximo en la columna i
        max_index =  ndice  de la fila con el valor absoluto m ximo en la
            columna i (de U[i:n, i])

        si max_index != i:
            intercambiar filas de U entre las filas i y max_index
```

```
            intercambiar filas de P entre las filas i y max_index
            intercambiar elementos de L entre las filas i y max_index hasta la columna i


        L[i][i] = 1

        # Calcular los elementos de la matriz L y U
        para j desde i+1 hasta n-1:
            L[j][i] = U[j][i] / U[i][i]  # Calcular los elementos de L
            U[j] -= L[j][i] * U[i]       # Actualizar la fila de U

    retornar P, L, U
```

Results

results/LUpivot.png

1.13 Crout

Input Values

image.png

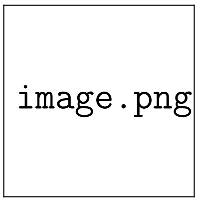
```
funcion crout(A, b):
    n = tama o de A
    Crear matriz L con ceros de tama o n x n
    Crear matriz U con unos en la diagonal y ceros en el resto de los elementos

    para j desde 0 hasta n-1:
        para i desde j hasta n-1:

            L[i][j] = A[i][j] - suma(L[i][k] * U[k][j] para k desde 0 hasta j)

        para i desde j+1 hasta n-1:

            U[j][i] = (A[j][i] - suma(L[j][k] * U[k][i] para k desde 0 hasta j)) / L[j][j]

    imprimir "Matriz L:"
    imprimir L
    imprimir "Matriz U:"
    imprimir U

    retornar L, U
```

Results

results/Crout.png

1.14 Dolittle

Input Values




```
funcion doolittle(A, b):
    n = tama o de A
    Crear matriz L con unos en la diagonal (matriz identidad de tama o n x n)
    Crear matriz U con ceros de tama o n x n

    para i desde 0 hasta n-1:
        para j desde i hasta n-1:
            U[i][j] = A[i][j] - suma(L[i][k] * U[k][j] para k desde 0 hasta i)

        para j desde i+1 hasta n-1:

            L[j][i] = (A[j][i] - suma(L[j][k] * U[k][i] para k desde 0 hasta i)
               ) / U[i][i]

    imprimir "Matriz L (Doolittle):"
    imprimir L
    imprimir "Matriz U (Doolittle):"
    imprimir U

    retornar L, U
```

Results

results/Dolittel.png

1.15 Vandermonde

Input Values

image1.png

```
funci n obtener_polinomio(x, y):
    # Crear la matriz de Vandermonde
    vandermonde = generar_matriz_vandermonde(x)

    # Resolver el sistema V * a = y para obtener los coeficientes
    coeficientes = resolver_sistema(vandermonde, y)

    # Generar el polinomio a partir de los coeficientes
    polinomio = ""
    para cada coeficiente en coeficientes:
        si el grado (exponente) es mayor que 0:
            polinomio += "{coef}x^{grado} + "
        sino:
            polinomio += "{coef}"

    retornar coeficientes
```

Results

results/Vandermonde.png

1.16 Linear Spline

Input Values

image1.png

```
funcion splineLineal(x, y):
    crear lista vac a splines
    n = longitud de x

    para i desde 0 hasta n-2:
        # Calcular la pendiente (m) entre los dos puntos consecutivos
        m = (y[i + 1] - y[i]) / (x[i + 1] - x[i])

        # Calcular la intersecci n (b) de la recta
        b = y[i] - m * x[i]

        # Crear el trazador en formato de ecuaci n "m * x + b"
        agregar "m * x + b" a la lista de "trazadores"

    retornar trazadores
```

Results

results/spline1.png

1.17 Lagrange

Input Values

image1.png

```
funcion splineLineal(x, y):
    crear lista vac a splines
    n = longitud de x

    para i desde 0 hasta n-2:
        # Calcular la pendiente (m) entre los dos puntos consecutivos
        m = (y[i + 1] - y[i]) / (x[i + 1] - x[i])

        # Calcular la intersecci n (b) de la recta
        b = y[i] - m * x[i]

        # Crear el trazador en formato de ecuaci n "m * x + b"
```

```
        agregar "m * x + b" a la lista de "trazadores"

    retornar trazadores
```

Results

Polinomio de Lagrange.png

1.18 Newton

Input Values

image1.png

Results

Polinomio de Newton.png

1.19 Gauss-Seidel Method

Description

Gauss-Seidel is an iterative method used to solve systems of linear equations, improving on Jacobi by using updated values within the iteration itself.

Pseudocode

```
Procedure Gauss_Seidel_Method
    INPUT: A (coefficient matrix), b (right-hand side vector), x0 (initial
       guess), tol (tolerance), niter (maximum iterations)

    SET iteration_count to 0
    SET x to x0

    WHILE iteration_count < niter DO
        SET x_old to x
        FOR i = 1 to n DO
            COMPUTE sum1 as sum(A[i,j] * x[j] for j = 1 to i-1)
            COMPUTE sum2 as sum(A[i,j] * x_old[j] for j = i+1 to n)
            SET x[i] to (b[i] - sum1 - sum2) / A[i,i]
        END FOR

        COMPUTE error as norm(x - x_old)
        IF error < tol THEN
            PRINT "Solution found"
            RETURN x
        END IF

        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 11: Gauss-Seidel Method

Input Values

- Coefficient matrix A

$$A = \begin{pmatrix} 4 & -1 & 0 & 3 \\ 1 & 15.5 & 3 & 8 \\ 0 & -1.3 & -4 & 1.1 \\ 14 & 5 & -2 & 30 \end{pmatrix}$$

- Right-hand side vector b

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

- Initial guess x_0

$$x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Tolerance $extTol = 1e - 7$
- Maximum number of iterations $N_{extmax} = 100$

Results

results/gauss seidel 1.png

results/gauss seidel 2.png

1.20 Successive Over-Relaxation (SOR) Method

Description

SOR is an iterative method similar to Gauss-Seidel, but with a relaxation factor w to potentially improve convergence speed.

Pseudocode

```
Procedure SOR_Method
    INPUT: A (coefficient matrix), b (right-hand side vector), x0 (initial
        guess), w (relaxation factor), tol (tolerance), niter (maximum
        iterations)

    SET iteration_count to 0
    SET x to x0

    WHILE iteration_count < niter DO
        SET x_old to x
        FOR i = 1 to n DO
            COMPUTE sum1 as sum(A[i,j] * x[j] for j = 1 to i-1)
            COMPUTE sum2 as sum(A[i,j] * x_old[j] for j = i+1 to n)
            SET x[i] to (1 - w) * x_old[i] + w * (b[i] - sum1 - sum2) / A[i,i]
        END FOR

        COMPUTE error as norm(x - x_old)
        IF error < tol THEN
            PRINT "Solution found"
            RETURN x
        END IF

        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 12: Successive Over-Relaxation (SOR) Method

Input Values

- Coefficient matrix A

$$A = \begin{pmatrix} 4 & -1 & 0 & 3 \\ 1 & 15.5 & 3 & 8 \\ 0 & -1.3 & -4 & 1.1 \\ 14 & 5 & -2 & 30 \end{pmatrix}$$

- Right-hand side vector b

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

- Initial guess x_0

$$x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Relaxation factor $w = 1.5$
- Tolerance $extTol = 1e - 7$
- Maximum number of iterations $N_{extmax} = 100$

Results

results/sor 1.png

results/sor 2.png

1.21 Jacobi Method

Description

The Jacobi method is an iterative algorithm for solving a system of linear equations by decomposing it into independent updates, using the old values from the previous iteration.

Pseudocode

```
Procedure Jacobi_Method
    INPUT: A (coefficient matrix), b (right-hand side vector), x0 (initial
       guess), tol (tolerance), niter (maximum iterations)

    SET iteration_count to 0
    SET x to x0

    WHILE iteration_count < niter DO
        SET x_new to x
        FOR i = 1 to n DO
            COMPUTE sum1 as sum(A[i,j] * x[j] for j = 1 to i-1)
            COMPUTE sum2 as sum(A[i,j] * x[j] for j = i+1 to n)
            SET x_new[i] to (b[i] - sum1 - sum2) / A[i,i]
        END FOR

        COMPUTE error as norm(x_new - x)
        IF error < tol THEN
            PRINT "Solution found"
            RETURN x_new
        END IF

        SET x to x_new
        INCREMENT iteration_count
    END WHILE

    PRINT "Method did not converge"
END Procedure
```

Listing 13: Jacobi Method

Input Values

- Coefficient matrix A

$$A = \begin{pmatrix} 4 & -1 & 0 & 3 \\ 1 & 15.5 & 3 & 8 \\ 0 & -1.3 & -4 & 1.1 \\ 14 & 5 & -2 & 30 \end{pmatrix}$$

- Right-hand side vector b

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

- Initial guess x_0

$$x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

- Tolerance $extTol = 1e - 7$
- Maximum number of iterations $N_{extmax} = 100$

Results

results/jacobi 1.png

results/jacobi 2.png

results/jacobi 3.png

1.22 Cholesky Decomposition Method

Description

Cholesky decomposition is a method for decomposing a symmetric positive-definite matrix into a product of a lower triangular matrix and its transpose.

Pseudocode

```
Procedure Cholesky_Decomposition_Method
    INPUT: A (symmetric positive-definite matrix), b (right-hand side vector)

    SET n to number of rows of A
    INITIALIZE L as an n x n matrix of zeros
    INITIALIZE U as an n x n matrix of zeros

    FOR k = 1 to n DO
        COMPUTE sum1 as sum(L[k,p] * U[p,k] for p = 1 to k-1)
        SET L[k,k] to sqrt(A[k,k] - sum1)
        SET U[k,k] to L[k,k]

        FOR i = k + 1 to n DO
            COMPUTE sum2 as sum(L[i,r] * U[r,k] for r = 1 to k-1)
            SET L[i,k] to (A[i,k] - sum2) / U[k,k]
        END FOR

        FOR j = k + 1 to n DO
            COMPUTE sum3 as sum(L[k,s] * U[s,j] for s = 1 to k-1)
            SET U[k,j] to (A[k,j] - sum3) / L[k,k]
        END FOR
    END FOR

    PRINT matrices L and U

    # Forward substitution to solve Ly = b
    INITIALIZE y as an n x 1 vector of zeros
    FOR i = 1 to n DO
        COMPUTE sum as sum(L[i,j] * y[j] for j = 1 to i-1)
        SET y[i] to (b[i] - sum) / L[i,i]
    END FOR

    # Backward substitution to solve Ux = y
    INITIALIZE x as an n x 1 vector of zeros
    FOR i = n DOWNTO 1 DO
        COMPUTE sum as sum(U[i,j] * x[j] for j = i+1 to n)
        SET x[i] to (y[i] - sum) / U[i,i]
    END FOR

    PRINT solution vector x
END Procedure
```

Listing 14: Cholesky Decomposition Method

Input Values

- Coefficient matrix A

$$A = \begin{pmatrix} 4 & -1 & 0 & 3 \\ 1 & 15.5 & 3 & 8 \\ 0 & -1.3 & -4 & 1.1 \\ 14 & 5 & -2 & 30 \end{pmatrix}$$

- Right-hand side vector b

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Results

results/cholesky 1.png

results/cholesky 2.png

2 Pow_Parser

One of the main challenges was to convert power expressions from the user input format (using ˆ for exponentiation) to Go's native 'math.Pow(x, y)' function. The algorithm starts from the right end of the expression and identifies the base and exponent recursively.

Test Example:

```
Input: sin(x)^2
Output: math.Pow(sin(x), 2)
```

3 Project Experience and Challenges

The development of this Numeric Analysis Application has been a rewarding and insightful journey for our team. Throughout the project, we encountered several technical and conceptual challenges, which ultimately allowed us to grow as developers and deepen our understanding of numerical methods.

3.1 Initial Challenges

When we first embarked on this project, one of the most significant challenges was deciding how to structure the application in a modular, scalable manner. Given the complexity of implementing several numerical methods, we chose to separate the system into three core components: a frontend (React), a backend (FastAPI), and a service (Go). This decision allowed us to clearly define each component's role and make the system more maintainable.

Another early challenge involved integrating a robust mathematical expression system. Since the Go language does not have a built-in operator for exponentiation, we had to develop our solution to handle power expressions in the form (x^y). Initially, we explored several regular expressions and existing algorithms, but they were not sufficient for our needs. This led us to implement our custom `pow_parser`, which efficiently converts expressions using the caret symbol (ˆ) into Go's native `math.Pow` format.

3.2 The Go Service and Expression Evaluation

Working with Go presented both opportunities and challenges. While Go's performance and simplicity made it an excellent choice for the computational heavy-lifting of the project, we found ourselves spending more time than anticipated developing a custom expression evaluation system. The `go-evaluate` library was instrumental in enabling us to parse and execute string-based mathematical expressions. However, ensuring that users would not need to adapt to Go's internal syntax required extra development effort, particularly in creating a user-friendly interface for mathematical operations.

Developing the `pow_parser` turned out to be a key accomplishment for the team. The algorithm that converts power expressions from user inputs allowed us to isolate the complexities of Go's mathematical operations from the user. This parser was implemented recursively, handling even complex nested expressions such as $sin(x)^2$ or $ln(sin(x))^{-1}$.

3.3 Backend Communication and gRPC Integration

While we initially considered multiple communication protocols between the FastAPI backend and the Go service, we settled on gRPC for its efficiency in handling real-time data exchange. Setting up the communication required us to dive deep into understanding gRPC and how it can be integrated with FastAPI. This also allowed us to explore connection pooling and concurrency to ensure the system could handle multiple user requests efficiently.

3.4 Time Management and Coordination

Another key learning from the project was the importance of time management and team coordination. Balancing the frontend and backend development efforts, while ensuring seamless communication between the two, required frequent team meetings and clear documentation. Dividing the project into smaller tasks and using tools like GitHub and Trello allowed us to track progress and avoid bottlenecks.

3.5 Conclusion

Through the course of this project, we not only expanded our knowledge of numeric methods but also gained practical experience in building a full-stack application from the ground up. By addressing each technical challenge—whether it was managing the communication between services or implementing custom parsers—we were able to create a robust system that simplifies complex numerical methods for the end-user. The process also taught us valuable lessons in collaboration, adaptability, and problem-solving that will benefit us in future endeavors