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

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 Requirements

This section defines both functional and non-functional requirements for the system, including those related to the frontend, backend, and Go service components. The system is designed to solve a variety of numerical root-finding problems and provides users with an interface to interact with these methods.

1.1 General System Requirements

- **REQ001**: The user must be able to access each specified numerical method, which includes:
  - Incremental Searches
  - False Position
  - Secant
  - Newton-Raphson
  - Multiple Roots
  - Fixed Point
- **REQ002**: The user must be able to input data to execute the root-finding methods. The inputs must include:
  - **Base elements**:
    - Number of iterations <int>
    - Tolerance <float>
    - Error type <int>: 1 for absolute error, 2 for relative error
  - **Method-specific elements**:
    - Initial guesses (e.g., for the Secant method, initial points x_0 and x_1)
    - Function expression $f(x)$ in standard mathematical notation.
- **REQ003**: Upon submission, the user must receive a table of iterations containing the following columns:

i : Iteration number

x : Current approximation of the root

f(x) : Value of the function at the current approximation

E or e : Error (absolute or relative)

- **REQ004**: Upon submission, the user must also receive a graph showing the error vs. iteration count. This will help users quickly determine if the function is converging.
- **REQ005**: The user must have access to a mathematical expression system for evaluating functions. This system must support standard mathematical operations, including:
  - Basic arithmetic operations: addition, subtraction, multiplication, and division
  - Power operations (using a custom parser for the caret symbol ^)
  - Common functions: logarithms, trigonometric functions, etc.

1.2 Frontend Requirements

The frontend of the system must be intuitive and user-friendly, allowing users to interact with the implemented numerical methods seamlessly.

- **FR001**: The frontend must provide a user interface (UI) allowing users to access each numerical method. The methods must include:
  - Incremental Search
  - False Position
  - Secant
  - Newton-Raphson
  - Multiple Roots
  - Fixed Point
- **FR002**: The frontend must allow users to input the necessary data to run the root-finding algorithms. The inputs must include:

  - Number of iterations <int>
  - Tolerance <float>
  - Error type <int>: 1 for absolute error, 2 for relative error
  - Method-specific parameters, such as initial guesses and intervals (for example, the initial points x_0 and x_1 in the Secant method)
  - The function expression $f(x)$ in standard mathematical notation

- **FR003**: The frontend must validate user inputs before sending them to the backend. This includes ensuring:
  - Numerical fields contain valid numbers
  - Required fields are not left empty
  - User-friendly error messages are displayed for invalid inputs
- **FR004**: Upon valid submission, the frontend must send a request to the backend API with the input data in the required format.
- **FR005**: The frontend must display the iteration table received from the backend, including:
  - Iteration number [i]
  - Current approximation [x]
  - Function value [f(x)]
  - Error [E or e] (absolute or relative)
- **FR006**: The frontend must generate and display a graph of error vs. iterations to help users visualize convergence. Optional features include zooming or hovering over data points.
- **FR007**: The frontend must provide access to the expression system used to evaluate the function. This includes a text input for function expressions with support for standard mathematical syntax.
- **FR008**: The frontend must handle and display error messages received from the backend in a user-friendly manner. The system should suggest fixes or highlight input fields needing correction.

- **FR009**: The frontend must be responsive and compatible across different devices and screen sizes. Accessibility guidelines should be followed, such as proper labels for screen readers.

1.3 Backend Requirements

The backend is responsible for processing input data, communicating with the Go service for calculations, and returning the results to the frontend.

- **BR001**: The backend must provide API endpoints for each numerical method, ensuring clear routes for accessing each method (e.g., `/api/methods/secant`).
- **BR002**: The backend must receive and parse input data from the frontend. This includes:
  - Base elements: iterations, tolerance, error type
  - Method-specific elements: initial guesses, function expressions
- **BR003**: The backend must validate the received data to check for missing or invalid parameters and ensure function expressions are correctly formatted.
- **BR004**: The backend must communicate with the Go service to perform calculations, using an appropriate communication protocol (e.g., gRPC, REST API).
- **BR005**: The backend must process the results from the Go service, format them for the frontend, and return them as JSON or another appropriate format. This includes:
  - Iteration table (i, x, f(x), E or e)
  - Data for error graphs
- **BR006**: The backend must handle exceptions and errors, catching issues from the Go service or data processing, and returning meaningful error messages with appropriate HTTP status codes.

- **BR007**: The backend must ensure security and data integrity by sanitizing inputs and implementing necessary security measures (e.g., protection against injection attacks).

1.4 Go Service Requirements

The Go service is the core of the system and is responsible for performing all numerical computations.

- **GR001**: The Go service must implement the following numerical methods:
  - Incremental Searches
  - False Position
  - Secant
  - Newton-Raphson
  - Multiple Roots
  - Fixed Point
- **GR002**: The Go service must provide a CLI for testing purposes and include a `test_results` module with default values for initial testing without external requests.
- **GR003**: The Go service must accept function expressions in standard mathematical notation, allowing users to input expressions using common syntax (e.g., `^` for exponentiation).
- **GR004**: Implement a `pow_parser` to convert caret symbols `^` into Go's native `math.Pow(x, y)` function format. This parser should handle complex expressions recursively.
- **GR005**: Ensure safe evaluation of function expressions by using a library like `go-evaluate`. All potentially dangerous code should be blocked.
- **GR006**: The Go service must communicate with the FastAPI backend using the agreed protocol (gRPC or REST).

- **GR007**: The service must return detailed results, including the iteration table and error graph data.
- **GR008**: Handle errors and edge cases, ensuring informative error messages for invalid inputs or calculation errors.

2 System Architecture

The system architecture consists of three main components:

1. **Frontend**: A React-based user interface that allows users to select methods, input data, and visualize results.
2. **Backend**: A FastAPI server that receives input data from the frontend and communicates with the Go service.
3. **Go Service**: Implements core numerical methods and returns results (iteration tables and graphs) to the backend.

3 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.

3.1 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.

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
```

3.2 Bisection

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

3.3 False Position

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

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
```

3.4 Newton-Raphson

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

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
```

3.5 Fixed Point

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

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
```

3.6 Secant

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

3.7 Multiple Roots

Description: This method is used when a function has multiple roots close to each other or repeated roots.
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) = 0.9915773906972 53, 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
```

4 Pow_Parser

One of the main challenges was to convert power expressions from the user input format (using ^ 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: $\sin(x)^2 Output : math.Pow(sin(x), 2)$

5 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.

5.1 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 (^) into Go's native `math.Pow` format.

5.2 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 $ln(sin(x))^{-1}$.

5.3 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.

5.4 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.

6 Python implementation

Newton method

```
Introduce la ecuación f(x): ln(sin(x)^2 + 1) - 1/2
Introduce la derivada de f(x): 2 * (sin(x)^2 + 1)^(-1) * sin(x) * cos(x)
Introduce el valor inicial (xo): 0.5
Introduce la tolerancia (tol): 0.0000007
Se encontró una raíz en: 0.936404580879562
Con: 4 iteraciones
```

Fixed-point method

```
Introduce la ecuación f(x): ln(sin(x)^2 + 1) - 1/2 - x
Introduce la función g(x): ln(sin(x)**2 + 1) - 1/2
Introduce el valor inicial: -0.5
Introduce la tolerancia: 0.0000007
('la raiz es: -0.374445246909060', 'Error: ', 5.94189786373711e-7)
```

Bisection method

```
Introduce la ecuación f(x): ln((sin(x)^2)+1)-(1/2)
Introduce el valor inicial: 0
Introduce el valor final: 1
Introduce la tolerancia (tol): 0.0000007
xm es raíz con tolerancia: 7e-07 con xm: 0.9364047050476074
Iteraciones realizadas: 20
```

False position method

```
Introduce la ecuación f(x): ln((sin(x)^2)+1)-(1/2)
Introduce el valor inicial: 0
Introduce el valor final: 1
Introduce la tolerancia (tol): 0.0000007
xm es raíz: 0.936404580879562
 Iteraciones realizadas: 7
```

Gaussian Elimination

```
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
```

Partial Pivoting

```
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
```

Total Pivoting

```
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
```

6.1 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