EAFIT University

School of Applied Sciences and Engineering

Numerical Analysis Report I

Teacher: Edwar Samir Posada Murillo

Names: Edy Julius López Rojas, Victor Daniel Arango Sohm, Samuel Madrid Ossa & Carlos David Sanchez Soto

September, 2025

Contents

Chapter 1

Numeric Methods

1.1 Solution of Nonlinear Equations

Nonlinear equations arise frequently in applied mathematics and engineering. In many cases, exact analytical solutions are not available, so numerical methods are employed to approximate the roots of functions. These techniques iteratively approach the solution with increasing accuracy, providing practical tools for real-world problems.

1.1.1 Incremental Search

Consecutive intervals of length Δx are evaluated until an interval $[a,b]$ is found where $f(a)\cdot f(b)<0$, indicating the existence of at least one root in that interval. in summary

$$f(a)\cdot f(b)<0 \quad \Rightarrow \quad \exists\, r\in(a,b): f(r)=0$$

Pseudocode

The user provides the input parameters under the assumptions that:

- f is a continuous function.
- f has at least one root in the given interval.

Algorithm 1 Incremental Search Method

procedure INCREMENTALSEARCH($f, x_0, \Delta x, N$)
    $a \leftarrow x_0$
    $fa \leftarrow f(a)$
    for $k \leftarrow 1$ to N **do**
        $b \leftarrow a + \Delta x$
        $fb \leftarrow f(b)$
        if $fa \cdot fb < 0$ **then**
            return Interval $[a, b]$ ▷ Root detected
        end if
        $a \leftarrow b$
        $fa \leftarrow fb$
    end for
    return "No root found within N steps"
end procedure

Testing

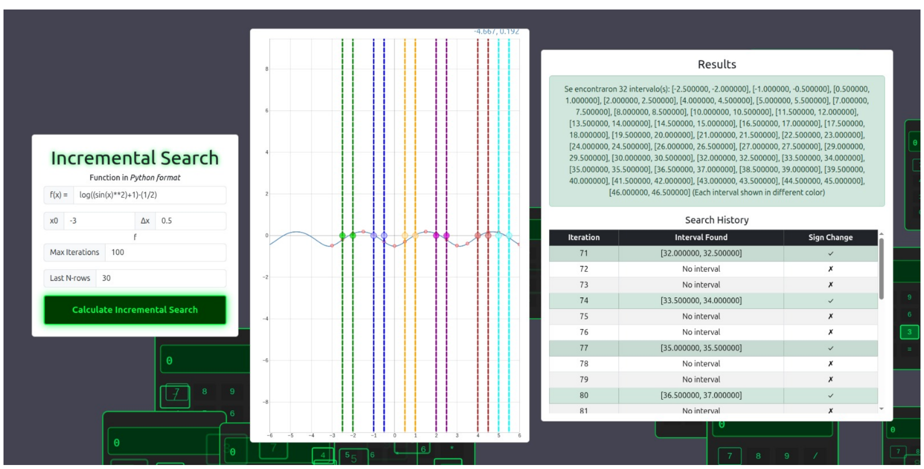


Figure 1.1: Testing on the page

1.1.2 Bisection

The bisection method is an iterative procedure to approximate real roots of a continuous function. Let $f : [a, b] \to \mathbb{R}$ be a continuous function on $[a, b]$, and suppose that

$$f(a) \cdot f(b) < 0,$$

then, by the *Intermediate Value Theorem*, there exists at least one root $r \in (a, b)$.

At each iteration, the midpoint is computed as

$$m = \frac{a+b}{2},$$

and the sign of $f(m)$ is evaluated. If $f(a) \cdot f(m) < 0$, then set $b \leftarrow m$; otherwise, set $a \leftarrow m$. Thus, the interval $[a,b]$ is halved at each step, ensuring it always contains a root.

The process continues until the absolute error $|x_k - x_{k-1}|$ is smaller than a given tolerance $\varepsilon > 0$, or until the maximum number of iterations $n_{\max}$ is reached.

If the initial interval is $[a,b]$, then after n iterations the absolute error satisfies

$$E_n \leq \frac{b-a}{2^n}.$$

This shows that the method converges linearly, with a convergence factor of $\frac{1}{2}$.

Pseudocode

Algorithm 2 Bisection Method

Require: Function $f(x)$, interval $[a,b]$, maximum iterations $n_{\max}$, tolerance ε
Ensure: Approximate root of $f(x) = 0$

$x_0 \leftarrow a$
for $i \leftarrow 1$ to $n_{\max}$ **do**
  $m \leftarrow \dfrac{a+b}{2}$ ▷ Midpoint
  $E \leftarrow |x_0 - m|$ ▷ Absolute error
  if $f(a) \cdot f(m) < 0$ **then**
    $b \leftarrow m$
  else
    $a \leftarrow m$
  end if
  Save i, a, b, m, E
  if $E < \varepsilon$ **then**
    return m, "Converged"
  end if
  $x_0 \leftarrow m$
end for
return m, "Max iterations reached"

Testing

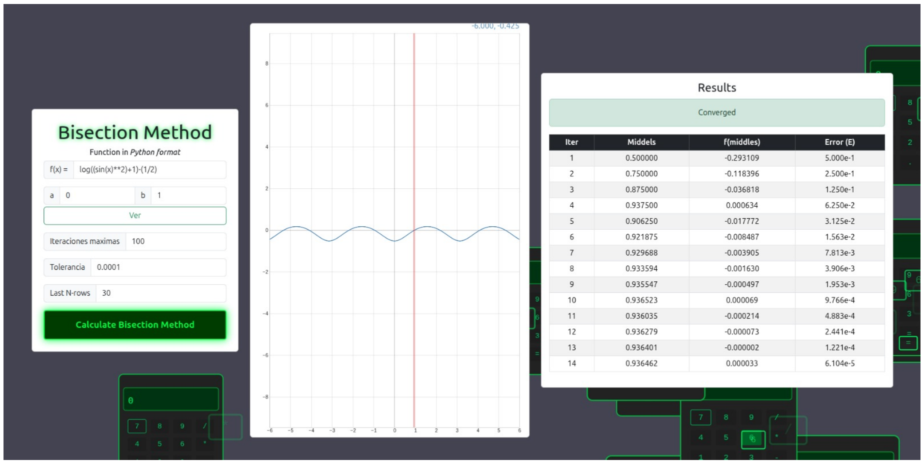


Figure 1.2: Testing on the page

1.1.3 False Position

Also known as the Regula Falsi method, it is a root-finding algorithm that starts with an interval $[a,b]$ such that $f(a) \cdot f(b) < 0$. Instead of taking the midpoint (as in the bisection method), it computes the point of intersection of the secant line between $(a, f(a))$ and $(b, f(b))$ with the x-axis:

$$x = \frac{af(b) - bf(a)}{f(b) - f(a)}.$$

The interval is then updated depending on the sign of $f(x)$, and the process is repeated until the root is approximated within a desired tolerance.

Algorithm 3 False Position Method

Require: Function $f(x)$, interval $[a,b]$, maximum iterations $n_{\max}$, tolerance ε
Ensure: Approximate root x^*, status message

```
if f(a) · f(b) > 0 then
    return {"final_root": None, "message": "Function does not change sign on [a,b]"}
end if
x_0 ← a
for i = 1 to n_max do
    Compute f(a) and f(b)
    x_r ← b − f(b) (b − a) / (f(b) − f(a))
    Compute f(x_r) and error E = |x_r − x_0|
    if E < ε or f(x_r) = 0 then
        return {"final_root": x_r, "message": "Converged"}
    end if
    if f(a) · f(x_r) < 0 then
        b ← x_r
    else
        a ← x_r
    end if
    x_0 ← x_r
end for
return {"final_root": x_r, "message": "Max iterations reached"}
```

Pseudocode

Testing

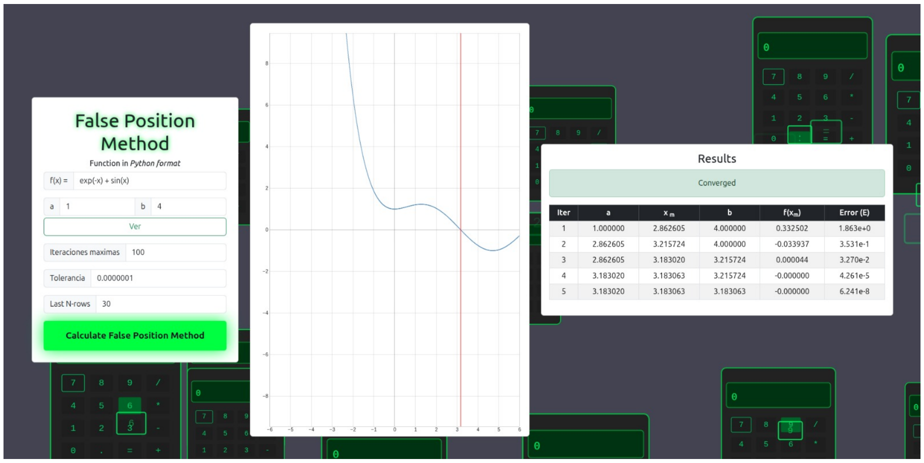


Figure 1.3: Enter Caption

1.1.4 Fixed Point

This root-finding technique rewrites the equation $f(x) = 0$ in the form $x = g(x)$. Starting from an initial guess x_0, the method generates a sequence defined by

$$x_{k+1} = g(x_k), \quad k = 0, 1, 2, \ldots$$

If $g(x)$ satisfies certain convergence conditions (e.g., $|g'(x)| < 1$ near the root), the sequence converges to the fixed point x^*, which is also a solution of $f(x) = 0$.

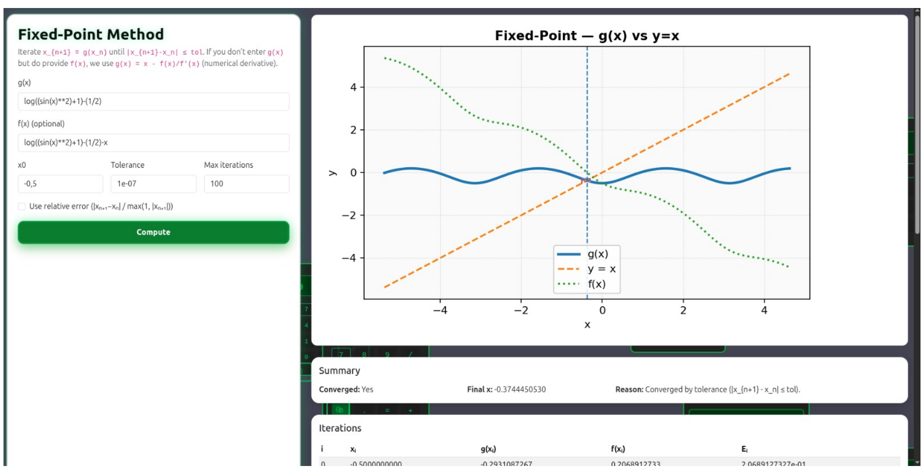
Pseudocode

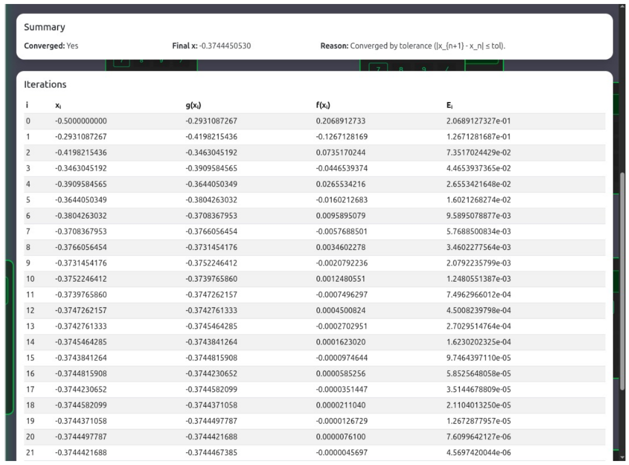
Algorithm 4 Fixed-Point Method

Require: Initial guess x_0, function $g(x)$, tolerance ε, maximum iterations $n_{\max}$
Ensure: Approximate root of $f(x) = 0$

for $i \leftarrow 0$ to $n_{\max} - 1$ **do**
    $x_{i+1} \leftarrow g(x_i)$
    if using relative error **then**
        $E \leftarrow \dfrac{|x_{i+1} - x_i|}{\max(1, |x_{i+1}|)}$
    else
        $E \leftarrow |x_{i+1} - x_i|$
    end if
    Save iteration data $(i, x_i, g(x_i), f(x_i), E)$
    if $E \le \varepsilon$ **then**
        return x_{i+1}, "Converged"
    end if
end for
return $x_{n_{\max}}$, "Max iterations reached"

Testing





Summary

Converged: Yes **Final x:** -0.3744450530 **Reason:** Converged by tolerance (|x_{n+1} - x_n| ≤ tol).

Iterations

i	x_i	$g(x_i)$	$f(x_i)$	E_i
0	-0.5000000000	-0.2931087267	0.2068912733	2.0689127327e-01
1	-0.2931087267	-0.4198215436	-0.1267128169	1.2671281687e-01
2	-0.4198215436	-0.3463045192	0.0735170244	7.3517024429e-02
3	-0.3463045192	-0.3909584565	-0.0446539374	4.4653937365e-02
4	-0.3909584565	-0.3644050349	0.0265534216	2.6553421648e-02
5	-0.3644050349	-0.3804263032	-0.0160212683	1.6021268274e-02
6	-0.3804263032	-0.3708367953	0.0095895079	9.5895078877e-03
7	-0.3708367953	-0.3766056454	-0.0057688501	5.7688500834e-03
8	-0.3766056454	-0.3731454176	0.0034602278	3.4602277564e-03
9	-0.3731454176	-0.3752246412	-0.0020792236	2.0792235799e-03
10	-0.3752246412	-0.3739765860	0.0012480551	1.2480551387e-03
11	-0.3739765860	-0.3747262157	-0.0007496297	7.4962966012e-04
12	-0.3747262157	-0.3742761333	0.0004500824	4.5008239798e-04
13	-0.3742761333	-0.3745464285	-0.0002702951	2.7029514764e-04
14	-0.3745464285	-0.3743841264	0.0001623020	1.6230202325e-04
15	-0.3743841264	-0.3744815908	-0.0000974644	9.7464397110e-05
16	-0.3744815908	-0.3744230652	0.0000585256	5.8525648058e-05
17	-0.3744230652	-0.3744582099	-0.0000351447	3.5144678809e-05
18	-0.3744582099	-0.3744371058	0.0000211040	2.1104013250e-05
19	-0.3744371058	-0.3744497787	-0.0000126729	1.2672877957e-05
20	-0.3744497787	-0.3744421688	0.0000076100	7.6099642127e-06
21	-0.3744421688	-0.3744467385	-0.0000045697	4.5697420044e-06

Figure 1.4: Testing on a page

1.1.5 Newton Method

Also called the Newton–Raphson method, it is an iterative root-finding algorithm. Starting from an initial approximation x_0, the method uses the tangent line at the point $(x_k, f(x_k))$ to compute a better approximation:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots$$

Under suitable conditions (when f is differentiable and $f'(x^*) \neq 0$), the sequence $\{x_k\}$ converges quadratically to the root x^*.v

Pseudocode

Algorithm 5 Newton Method

Require: Function $f(x)$, initial guess x_0, tolerance ε, maximum iterations $N_{\max}$
Ensure: Approximate root of $f(x) = 0$

```
for n ← 0 to N_max − 1 do
    if f′(x_n) = 0 then
        return "Error: Division by zero"
    end if
    x_{n+1} ← x_n − f(x_n)/f′(x_n)
    E ← |x_{n+1} − x_n|                                ▷ Absolute error
    Save (n, x_n, E)
    if E < ε then
        return x_{n+1}, "Tolerance satisfied"
    end if
    x_n ← x_{n+1}
end for
return x_{N_max}, "Maximum iterations exceeded"
```

Testing

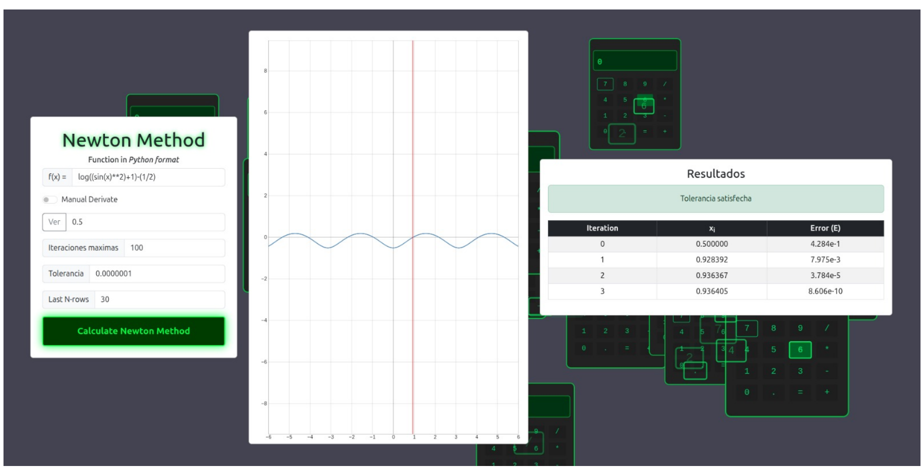


Figure 1.5: Testing on a page

1.1.6 Secant Method

This root-finding method is similar to Newton's method but does not require the derivative of $f(x)$. Instead, it approximates the derivative by using two initial guesses x_0 and

x_1. The iterative formula is:

$$x_{k+1} = x_k - f(x_k)\,\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}, \quad k = 1, 2, \ldots$$

The method generally converges faster than the bisection method, though its convergence is superlinear (slower than Newton's quadratic convergence).

Pseudocode

Algorithm 6 Secant Method

Require: Function $f(x)$, initial guesses x_0, x_1, tolerance ε, maximum iterations $N_{\max}$
Ensure: Approximate root of $f(x) = 0$

for $n \leftarrow 0$ to $N_{\max} - 1$ **do**
  if $f(x_1) - f(x_0) = 0$ **then**
    return "Error: Division by zero"
  end if
  $x_2 \leftarrow x_1 - f(x_1)\dfrac{x_1 - x_0}{f(x_1) - f(x_0)}$
  $E \leftarrow |x_2 - x_1|$ ▷ Absolute error
  Save $(n, x_1, f(x_1), E)$
  if $E < \varepsilon$ **then**
    return x_2, "Tolerance satisfied"
  end if
  $x_0 \leftarrow x_1$, $x_1 \leftarrow x_2$
end for
return $x_{N_{\max}}$, "Maximum iterations exceeded"

Testing

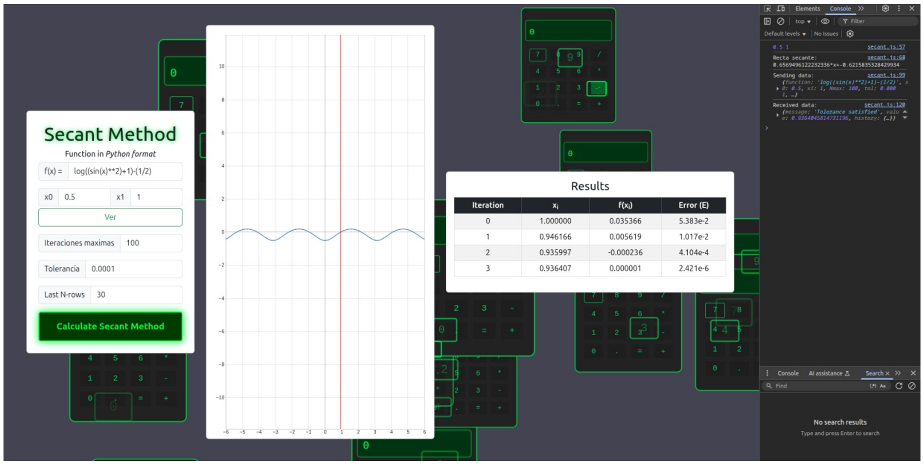


Figure 1.6: Testing on a page

1.1.7 Multiple Roots

This method is a modification of the classical Newton's method designed to find roots of a function $f(x)$ that have multiplicity greater than one. Standard Newton's method converges slowly for multiple roots, so this approach improves convergence. The iterative formula is:

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n)}{[f'(x_n)]^2 - f(x_n)f''(x_n)},$$

where $f'(x)$ and $f''(x)$ are the first and second derivatives of f. Starting from an initial guess x_0, the method iterates until the absolute difference $|x_{n+1} - x_n|$ is below a given tolerance or the maximum number of iterations is reached. This method achieves faster convergence for multiple roots compared to standard Newton's method.

Pseudocode

Algorithm 7 Multiple Roots

Require: Function $f(x)$, initial guess x_0, tolerance ε, maximum iterations $N_{\max}$
Ensure: Approximate root x^* or failure message

Define $f'(x)$ and $f''(x)$ (use given derivatives or compute symbolically)
for $n = 0$ to $N_{\max} - 1$ **do**
    Compute denominator $denom \leftarrow f'(x_0)^2 - f(x_0) \cdot f''(x_0)$
    if $denom = 0$ **then**
        return "Denominator zero, method fails"
    end if
    Update $x_1 \leftarrow x_0 - \frac{f(x_0) \cdot f'(x_0)}{denom}$
    Compute absolute error $E \leftarrow |x_1 - x_0|$
    if $E < \varepsilon$ **then**
        return x_1 as approximate root
    end if
    $x_0 \leftarrow x_1$
end for
return x_1 with message "Maximum iterations reached"

Testing

1.2 Solution of linear system equations

1.2.1 Gaussian Elimination

It is a direct method to solve a system of linear equations $A\mathbf{x} = \mathbf{b}$. The algorithm transforms the coefficient matrix A into an upper triangular form by applying elementary row operations (without pivoting). Once in triangular form, the solution is obtained through back-substitution.

Pseudocode

Algorithm 8 Gaussian Elimination (Simple)

Require: Matrix $A \in \mathbb{R}^{n\times n}$, vector $b \in \mathbb{R}^n$
Ensure: Solution vector x or failure message

```
Compute det(A)
if det(A) = 0 then
    return {"solution": None, "message": "Matrix is not invertible"}
end if
for k = 0 to n − 2 do
    if A[k,k] = 0 then
        return {"solution": None, "message": "Zero pivot encountered"}
    end if
    for i = k + 1 to n − 1 do
        if A[i,k] ≠ 0 then
            m ← A[i,k]/A[k,k]
            A[i,k : n] ← A[i,k : n] − m · A[k,k : n]
            b[i] ← b[i] − m · b[k]
        end if
    end for
end for
Initialize x ← 0 ∈ R^n
for i = n − 1 downto 0 do
    if A[i,i] = 0 then
        return {"solution": None, "message": "Zero pivot in back substitution"}
    end if
    x[i] ← (b[i] − Σ_{j=i+1}^{n−1} A[i,j] · x[j]) / A[i,i]
end for
return {"solution": x, "message": "Gaussian elimination completed successfully"}
```

```
--- Elimination step for column 2 ---
Multiplier for row 3 = 13.000000
Updated row 3 of A: [  0.    0.  -41.  -73.5]
Updated b[3] = -5.500000
Multiplier for row 4 = 12.000000
Updated row 4 of A: [  0.   0. -38. -96.]
Updated b[4] = -12.000000
Matrix A after elimination step:
[[  2.   -1.    0.    3. ]
 [  0.    1.    3.    6.5]
 [  0.    0.  -41.  -73.5]
 [  0.    0.  -38.  -96. ]]
Vector b after elimination step:
[  1.    0.5  -5.5 -12. ]
----------------------------------------
--- Elimination step for column 3 ---
Multiplier for row 4 = 0.926829
Updated row 4 of A: [  0.          0.          0.        -27.878049]
Updated b[4] = -6.902439
Matrix A after elimination step:
[[  2.         -1.          0.          3.       ]
 [  0.          1.          3.          6.5      ]
 [  0.          0.        -41.        -73.5      ]
 [  0.          0.          0.        -27.878049]]
Vector b after elimination step:
[ 1.         0.5       -5.5       -6.902439]
----------------------------------------
--- Back substitution ---
x[4] = 0.247594
x[3] = -0.309711
x[2] = -0.180227
x[1] = 0.038495
```

Figure 1.7: Testing on console

Testing

```
Initial system. Determinant = 2286.000000
Matrix A:
[[ 2.  -1.   0.   3. ]
 [ 1.   0.5  3.   8. ]
 [ 0.  13.  -2.  11. ]
 [14.   5.  -2.   3. ]]
Vector b:
[1. 1. 1. 1.]
----------------------------------------
--- Elimination step for column 1 ---
Multiplier for row 2 = 0.500000
Updated row 2 of A: [0.  1.  3.  6.5]
Updated b[2] = 0.500000
No operation needed for row 3 (element is zero).
Multiplier for row 4 = 7.000000
Updated row 4 of A: [  0.  12.  -2. -18.]
Updated b[4] = -6.000000
Matrix A after elimination step:
[[  2.   -1.    0.    3. ]
 [  0.    1.    3.    6.5]
 [  0.   13.   -2.   11. ]
 [  0.   12.   -2.  -18. ]]
Vector b after elimination step:
[ 1.   0.5  1.  -6. ]
```

1.2.2 Gaussian Elimination with Partial Pivoting

This method improves numerical stability in solving a system $A\mathbf{x} = \mathbf{b}$. At each elimination step, the algorithm selects the row with the largest absolute pivot element in the current column and swaps it with the current row. Then, elementary row operations are applied to form an upper triangular system, which is solved by back-substitution.

Pseudocode

Algorithm 9 Gaussian Elimination with Partial Pivoting

Require: Matrix $A \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$
Ensure: Solution vector x or failure message

Compute $\det(A)$
if $\det(A) = 0$ **then**
    return {"solution": None, "message": "Matrix is not invertible"}
end if
for $k = 0$ to $n - 2$ **do**
    $max\_row \leftarrow$ index of row with maximum $|A[i,k]|$ for $i = k..n-1$
    if $A[max\_row, k] = 0$ **then**
        return {"solution": None, "message": "All pivots in column $k+1$ are zero"}
    end if
    if $max\_row \neq k$ **then**
        Swap row k with row max_row in A and b
    end if
    for $i = k+1$ to $n-1$ **do**
        if $A[i,k] \neq 0$ **then**
            $m \leftarrow A[i,k]/A[k,k]$
            $A[i, k:n] \leftarrow A[i, k:n] - m \cdot A[k, k:n]$
            $b[i] \leftarrow b[i] - m \cdot b[k]$
        end if
    end for
end for
Initialize $x \leftarrow \mathbf{0} \in \mathbb{R}^n$
for $i = n-1$ downto 0 **do**
    if $A[i,i] = 0$ **then**
        return {"solution": None, "message": "Zero pivot in back substitution"}
    end if
    $x[i] \leftarrow \dfrac{b[i] - \sum_{j=i+1}^{n-1} A[i,j] \cdot x[j]}{A[i,i]}$
end for
return {"solution": x, "message": "Gaussian elimination with partial pivoting completed successfully"}

Testing

```
Enter the number of equations: 4

Enter the coefficients of the matrix A row by row (space separated):
Row 1: 2 -1 0 3
Row 2: 1 0.5 3 8
Row 3: 0 13 -2 11
Row 4: 14 5 -2 3

Enter the vector b (space separated):
1 1 1 1

Enter number of decimals to round: 6

Initial system:
A =
 [[ 2.  -1.   0.   3. ]
 [ 1.   0.5  3.   8. ]
 [ 0.  13.  -2.  11. ]
 [14.   5.  -2.   3. ]]
b =
 [1. 1. 1. 1.]

Determinant = 2286.0000
Starting Gaussian elimination with partial pivoting...
```

```
Iteration 1: Selecting pivot in column 1...
Swapped row 1 with row 4 for pivoting.
Pivot = 14.0
Matrix after pivoting (if any):
 [[14.   5.  -2.   3. ]
 [ 1.   0.5  3.   8. ]
 [ 0.  13.  -2.  11. ]
 [ 2.  -1.   0.   3. ]]
Vector b:
 [1. 1. 1. 1.]
Eliminating element A[2,1] using multiplier m = 0.071429
Updated row 2:
A = [0.       0.142857 3.142857 7.785714]
b = 0.928571
Skipping row 3 because element is already zero.
Eliminating element A[4,1] using multiplier m = 0.142857
Updated row 4:
A = [ 0.       -1.714286  0.285714  2.571429]
b = 0.857143
After elimination of column 1:
A =
 [[14.        5.       -2.        3.      ]
 [ 0.        0.142857  3.142857  7.785714]
 [ 0.       13.       -2.       11.      ]
 [ 0.       -1.714286  0.285714  2.571429]]
b =
 [1.       0.928571 1.       0.857143]
```

```
Iteration 2: Selecting pivot in column 2...
Swapped row 2 with row 3 for pivoting.
Pivot = 13.0
Matrix after pivoting (if any):
 [[14.        5.       -2.        3.      ]
 [ 0.       13.       -2.       11.      ]
 [ 0.        0.142857  3.142857  7.785714]
 [ 0.       -1.714286  0.285714  2.571429]]
Vector b:
 [1.       1.       0.928571 0.857143]
Eliminating element A[3,2] using multiplier m = 0.010989
Updated row 3:
A = [0.       0.       3.164835 7.664835]
b = 0.917582
Eliminating element A[4,2] using multiplier m = -0.131868
Updated row 4:
A = [0.       0.       0.021978 4.021978]
b = 0.989011
After elimination of column 2:
A =
 [[14.        5.       -2.        3.      ]
 [ 0.       13.       -2.       11.      ]
 [ 0.        0.        3.164835  7.664835]
 [ 0.        0.        0.021978  4.021978]]
b =
 [1.       1.       0.917582 0.989011]
```

```
Iteration 3: Selecting pivot in column 3...
Pivot = 3.1648351648351647
Matrix after pivoting (if any):
 [[14.        5.       -2.        3.      ]
 [ 0.       13.       -2.       11.      ]
 [ 0.        0.        3.164835  7.664835]
 [ 0.        0.        0.021978  4.021978]]
Vector b:
 [1.       1.       0.917582 0.989011]
Eliminating element A[4,3] using multiplier m = 0.006944
Updated row 4:
A = [0.      0.      0.      3.96875]
b = 0.982639
After elimination of column 3:
A =
 [[14.        5.       -2.        3.      ]
 [ 0.       13.       -2.       11.      ]
 [ 0.        0.        3.164835  7.664835]
 [ 0.        0.        0.        3.96875 ]]
b =
 [1.       1.       0.917582 0.982639]
```

```
Starting back substitution...

x[4] = (b[4] - sum(A[4,i+1:] * x[i+1:])) / A[4,4]
x[4] = (0.982639 - 0.000000) / 3.968750 = 0.247594
x[3] = (b[3] - sum(A[3,i+1:] * x[i+1:])) / A[3,3]
x[3] = (0.917582 - 1.897768) / 3.164835 = -0.309711
x[2] = (b[2] - sum(A[2,i+1:] * x[i+1:])) / A[2,2]
x[2] = (1.000000 - 3.342957) / 13.000000 = -0.180227
x[1] = (b[1] - sum(A[1,i+1:] * x[i+1:])) / A[1,1]
x[1] = (1.000000 - 0.461067) / 14.000000 = 0.038495

Back substitution complete.
Solution vector x =
 [ 0.038495 -0.180227 -0.309711  0.247594]
```

Figure 1.8: Testing on console

1.2.3 Gaussian Elimination with Total Pivoting

This variant of Gaussian elimination further increases numerical stability. At each step, the algorithm searches for the largest absolute element in the submatrix (rows and columns not yet eliminated), then swaps both rows and columns so that this element becomes the pivot. Afterward, elementary row operations are applied to form an upper triangular system, which is solved using back-substitution. Column swaps must also be tracked to correctly reorder the solution vector.

Pseudocode

Algorithm 10 Gaussian Elimination with Total Pivoting

Require: Matrix $A \in \mathbb{R}^{n \times n}$, vector $b \in \mathbb{R}^n$

Ensure: Solution vector x or failure message

Initialize $col\_order \leftarrow [0, 1, \ldots, n-1]$
Compute $\det(A)$
if $\det(A) = 0$ **then**
    return {"solution": None, "message": "Matrix is not invertible"}
end if
for $k = 0$ to $n-2$ **do**
    Find (max_row, max_col) = indices of largest $|A[i, j]|$ in submatrix $A[k : n-1, k : n-1]$
    if $A[max\_row, max\_col] = 0$ **then**
        return {"solution": None, "message": "All pivots in submatrix are zero"}
    end if
    if $max\_row \neq k$ **then**
        Swap row k with row max_row in A and b
    end if
    if $max\_col \neq k$ **then**
        Swap column k with column max_col in A
        Swap $col\_order[k]$ with $col\_order[max\_col]$
    end if
    for $i = k+1$ to $n-1$ **do**
        if $A[i,k] \neq 0$ **then**
            $m \leftarrow A[i,k]/A[k,k]$
            $A[i, k : n] \leftarrow A[i, k : n] - m \cdot A[k, k : n]$
            $b[i] \leftarrow b[i] - m \cdot b[k]$
        end if
    end for
end for
Initialize $x \leftarrow \mathbf{0} \in \mathbb{R}^n$
for $i = n-1$ downto 0 **do**
    if $A[i,i] = 0$ **then**
        return {"solution": None, "message": "Zero pivot in back substitution"}
    end if
    $x[i] \leftarrow \dfrac{b[i] - \sum_{j=i+1}^{n-1} A[i, j] \cdot x[j]}{A[i,i]}$
end for
Reorder x according to col_order to get x_final
return {"solution": x_final, "message": "Gaussian elimination with total pivoting completed successfully"}

Testing