EAFIT University - Department of Information and Computer Sciences

Mauricio Escudero Restrepo
César Esteban Peñuela Cubides
Diego Mesa Ospina

August 2024

1 Objective

Review the progress of the final project and the work methodologies of the workgroups

2 Course

Numerical Analisis

3 Responsible Faculty Member

Edwar Samir Posada Murillo

4 Current Report Delivery Date

October 1st 2024

5 Numerical Methods

The following numerical methods will be presented in this document in the following manner, first a pseudo-code version of the method algorithm will be presented, after the code for the implementation of the method in both languages selected will be presented, finally proof for the execution and results of the methods will be provided.

5.1 Incremental Search

Incremental Search Method is a numerical method used to find the roots of a function (i.e., where the function equals zero). This method works by iteratively narrowing down an interval where a root lies. The goal is to find an approximation of the root with increasing precision.

5.1.1 Pseudo-code

```
FUNCTION IncrementalSearch(f, x0, h, Nmax):

   INPUT:
   - f: continuous function for which a sign change is sought.
   - x0: initial point.
   - h: step size (increment in each iteration).
   - Nmax: maximum number of iterations.

   OUTPUT:
   - a: left endpoint of the interval where a sign change occurs.
   - b: right endpoint of the interval where a sign change occurs
       .
   - iter: number of iterations performed.
   - data_frame: table of results with details of each iteration.

   BEGIN:
   1. Initialize:
      - xprev = x0.
      - fprev = f(xprev).
      - xcurr = xprev + h.
      - fcurr = f(xcurr).
      - Create an empty list to store the results.

   2. FOR i = 1 to Nmax:

      a. If fprev * fcurr < 0, a sign change occurs in the
          interval [xprev, xcurr]:
         - Save the iteration result (i, xcurr, fcurr, error).
         - Break the loop.

      b. Save current iteration values:
         - Store iteration values: index i, current x, f(current
             x), and error (difference between current x and
             previous x).

      c. Update values for the next iteration:
         - xprev = xcurr.
```

```
        - fprev = fcurr.
        - xcurr = xprev + h.
        - fcurr = f(xcurr).

   3. Create a DataFrame with the stored results.

   4. Return:
      - xprev as the left endpoint of the interval.
      - xcurr as the right endpoint of the interval.
      - i as the number of iterations performed.
      - the DataFrame with the results.

END FUNCTION
```

5.1.2 Method Implementation

Python

Rust

5.1.3 Method Tests

The test parameters are as follows:

- f= math.log((math.sin(x)**2) + 1) - 1/2
- x0 = -3
- step = 0.5
- Tol = 1x10-7
- N = 100

Result: Interval: [-2.5, -2.0], Iterations: 2

i	x_i	f_xi	e
1	-2.5	-0.193863	0.5
2	-2.0	0.102578	0.5

5.2 Bisection

The bisection method is based on the Intermediate Value Theorem, which states that if a continuous function changes sign over an interval, there is at least one root in that interval. The method systematically reduces the interval in which the root lies by repeatedly bisecting it.

5.2.1 Pseudo-code

```
FUNCTION BisectionMethod(f, a, b, tolerance=1e-7, max_iterations
    =100):

    INPUT:
    - f: function to evaluate
    - a, b: interval [a, b] where the root is sought
    - tolerance: stopping criterion (optional)
    - max_iterations: maximum number of iterations (optional)

    OUTPUT:
    - c: approximate value of the root
    - iterations: number of iterations performed
    - converged: indicator of whether the algorithm converged
    - df_result: table of results from the iterations

    IF f(a) * f(b) >= 0 THEN:
        Raise an error ("f(a) and f(b) must have opposite signs")

    c = (a + b) / 2 // Initialize midpoint
    results_list = [] // Initialize list to store iteration
        results

    FOR i FROM 0 TO max_iterations DO:
        c = (a + b) / 2 // Calculate the midpoint of [a, b]

        IF (b - a) < tolerance OR abs(f(c)) < tolerance THEN:
            Create a DataFrame with results_list // Convert the
                list to a table
            PRINT the DataFrame // Display table with iteration
                results
            RETURN (c, i + 1, True, DataFrame) // Algorithm has
                converged

        // Determine in which half of the interval the root is
            located
        IF f(a) * f(c) < 0 THEN:
            b = c // The root is in the interval [a, c]
        ELSE:
            a = c // The root is in the interval [c, b]

        // Store the current iteration's results
        result = { 'iteration': i, 'x_i': c, 'f(x_i)': f(c), '
            error': abs(b - a) }
        results_list.ADD(result)
    // If the maximum number of iterations is reached without
        convergence
    RETURN (c, max_iterations, False, results_list)

END FUNCTION
```

5.2.2 Method Implementation

Python

Rust

5.2.3 Method Test

The test parameters are as follows:

- f= math.log((math.sin(x)**2) + 1) - 1/2
- a = 0
- b = 1
- Tol = 1x10-7
- N = 100

Result: - Root: 0.9364047050476074 - Iterations: 21 - Converged: True

i	x_i	f_x_i	e
0	0.5	-0.2931087267313766	0.5
1	0.75	-0.11839639385347844	0.25
2	0.875	-0.036817690757380506	0.125
3	0.9375	0.0006339161592386899	0.0625
4	0.90625	-0.017772289226861138	0.03125
5	0.921875	-0.008486582211768012	0.015625
6	0.9296875	-0.0039053586270640928	0.0078125
7	0.93359375	-0.0016304381170096915	0.00390625
8	0.935546875	-0.0004969353153196909	0.001953125
9	0.9365234375	6.882244496264622e-05	0.0009765625
10	0.93603515625	-0.00021397350516405567	0.00048828125
11	0.936279296875	-7.255478812057126e-05	0.000244140625
12	0.9364013671875	-1.860984900181606e-06	0.0001220703125
13	0.93646240234375	3.348202684883006e-05	6.103515625e-05
14	0.936431884765625	1.581084516011355e-05	3.0517578125e-05
15	0.9364166259765625	6.975011174192858e-06	1.52587890625e-05
16	0.9364089965820312	2.5570333977986692e-06	7.62939453125e-06
17	0.9364051818847656	3.4802931392352576e-07	3.814697265625e-06
18	0.9364032745361328	-7.564765268641693e-07	1.9073486328125e-06
19	0.9364042282104492	-2.042232898902263e-07	9.5367431640625e-07

5.3 False Rule

The false position method (also known as the regula falsi method) is a root-finding technique used in numerical analysis. It is similar to the bisection method in that it iteratively narrows down an interval where a root of a function exists. However, instead of using the midpoint of the interval as in the bisection method, the false position method uses a more refined estimate by linearly interpolating the function between the endpoints.

5.3.1 Pseudo-code

```
FUNCTION FalsePosition(f, a, b, tol, Nmax):

   INPUT:
   - f: continuous function whose root is sought.
   - a: left endpoint of the initial interval.
   - b: right endpoint of the initial interval.
   - tol: tolerance that defines the stopping criterion (when the
        error is sufficiently small).
   - Nmax: maximum number of iterations allowed.

   OUTPUT:
   - x: approximate solution.
   - iter: number of iterations performed.
   - err: estimated error.
   - result_array: list of results with values for each iteration
        (point, f(x) value, and error).

   BEGIN:
   1. Compute f(a) and f(b) // fa = f(a), fb = f(b)

   2. Compute the first midpoint using the false position formula
      :
     pm = (fb * a - fa * b) / (fb - fa) // pm is the first trial
          point

   3. Evaluate the function at the midpoint:
     fpm = f(pm)

   4. Initialize a large error value (E = 1000) to start the loop
      .

   5. Initialize an iteration counter (cont = 1).

   6. Create an empty list to store the results of each iteration
        (result_array).
```

```
    7. While the error (E) is greater than the tolerance (tol) AND
         the number of iterations is less than Nmax:
        a. If f(a) * f(pm) < 0:
            - The root is in the interval [a, pm], so update b =
                pm and fb = fpm.
        b. Else:
            - The root is in the interval [pm, b], so update a =
                pm and fa = fpm.

        c. Store the previous value of pm (p0 = pm).

        d. Compute a new midpoint using the false position formula
            :
           pm = (fb * a - fa * b) / (fb - fa)

        e. Evaluate the function at the new midpoint:
           fpm = f(pm)

        f. Compute the error as the difference between the new and
             previous midpoint:
           E = abs(pm - p0)

        g. Store the current iteration results (iteration number,
            point value, f(pm), and error) in result_array.

        h. Increment the iteration counter (cont += 1).

    8. When the loop ends (when the tolerance is met or the
        maximum number of iterations is reached):
        - The approximate solution is x = pm.
        - The number of iterations is iter = cont.
        - The estimated error is err = E.

    9. Return the approximate solution, the number of iterations,
        the error, and the set of iteration results.

END FUNCTION
```

5.3.2 Method Implementation

Python

Rust

5.3.3 Method Tests

Approximate solution:

- x = 0.9364045808798893
- Iterations = 5
- Error = 2.2097967899981086e-10

i	x_i	f_xi	e
1	0.9365060516656253	5.875600835791861e-05	0.0025656709474095596
2	0.9364047307426415	8.67825411532408e-08	0.00010132092298376083
3	0.936404581100869	1.2815393191090152e-10	1.4964177252885236e-07
4	0.9364045808798893	1.894040480010517e-13	2.2097967899981086e-10

5.4 Fixed Point

The fixed-point method is an iterative numerical technique used to solve equations of the form x=g(x)x = g(x)x=g(x). In this method, the goal is to find a value xxx such that g(x)=xg(x) = xg(x)=x, which is known as a fixed point of the function g(x)g(x)g(x).

5.4.1 Pseudo-code

```
FUNCTION FixedPointMethod(g, x0, tol=1e-7, max_iter=1000):

    INPUT:
    - g: function for fixed-point iteration, where we seek x = g(x
        ).
    - x0: initial guess or starting value for the iteration.
    - tol: tolerance to determine if the method has converged (
        optional, default is 1e-7).
    - max_iter: maximum number of iterations allowed (optional,
        default is 1000).

    OUTPUT:
    - x: approximate solution to the fixed point.
    - iterations: number of iterations performed.
    - converged: a boolean indicating whether the method converged
        .
    - result_array: list containing details of each iteration (
        values of x, g(x), and error).

    BEGIN:
```

```
    1. Create an empty list to store results of each iteration (
        result_array).

    2. Initialize x with the initial value x0.

    3. FOR i FROM 0 TO max_iter DO:
        a. Compute the new value x_new = g(x).

        b. Compute the error as error = abs(x_new - x).

        c. Create a dictionary with the current iteration results:
           - i: iteration number.
           - x: value of x_new.
           - g_x: value of g(x_new).
           - error: computed error.
           Add this dictionary to result_array.

        d. IF the error is less than the tolerance (error < tol):
            - The method has converged.
            - Return x_new, number of iterations (i + 1), True (
                converged), and result_array.

        e. Update the value of x with x_new.

    4. IF the maximum number of iterations is reached without
        convergence:
        - Return the last value of x, max_iter, False (not
            converged), and result_array.

END FUNCTION
```

5.4.2 Method Implementation

Python

Rust

5.4.3 Method Test

Approximate solution:

- Root found: -0.37444505296105535
- Iterations: 30
- Error: 7.726074024994034e-08

Iteration	x_i	f(x_i)	g(x_i)	Error
1	-0.2931087267313766	-0.12671281687488073	-0.41982154360625734	0.2068912732686234
2	-0.41982154360625734	0.07351702442859243	-0.3463045191776649	0.12671281687488073
3	-0.3463045191776649	-0.044653937364644736	-0.39095845654230965	0.07351702442859243
4	-0.39095845654230965	0.026553421648170428	-0.3644050348941392	0.044653937364644736
5	-0.3644050348941392	-0.016021268273817058	-0.3804263031679563	0.026553421648170428
6	-0.3804263031679563	0.009589507887747428	-0.37083679528020885	0.016021268273817058
7	-0.37083679528020885	-0.005768850083372357	-0.3766056453635812	0.009589507887747428
8	-0.3766056453635812	0.003460227756392209	-0.373145417607189	0.005768850083372357
9	-0.373145417607189	-0.002079223579867173	-0.3752246411870562	0.003460227756392209
10	-0.3752246411870562	0.00124805513874654	-0.37397658604830963	0.002079223579867173
11	-0.37397658604830963	-0.0007496296601224861	-0.3747262157084321	0.00124805513874654
12	-0.3747262157084321	0.00045008239797816874	-0.37427613331045395	0.0007496296601224861
13	-0.37427613331045395	-0.00027029514763832196	-0.3745464284580923	0.00045008239797816874
14	-0.3745464284580923	0.0001623020232475736	-0.3743841264348447	0.00027029514763832196
15	-0.3743841264348447	-9.746439711039168e-05	-0.3744815908319551	0.0001623020232475736
16	-0.3744815908319551	5.8525648058027624e-05	-0.37442306518389706	9.746439711039168e-05
17	-0.37442306518389706	-3.514467880877392e-05	-0.37445820986270584	5.8525648058027624e-05
18	-0.37445820986270584	2.110401325022826e-05	-0.3744371058494556	3.514467880877392e-05
19	-0.3744371058494556	-1.2672877957420337e-05	-0.37444977872741303	2.110401325022826e-05
20	-0.37444977872741303	7.609964212673681e-06	-0.37444216876320036	1.2672877957420337e-05
21	-0.37444216876320036	-4.5697420043566694e-06	-0.3744467385052047	7.609964212673681e-06
22	-0.3744467385052047	2.744098679452467e-06	-0.37444399440652526	4.5697420043566694e-06
23	-0.37444399440652526	-1.647814738270359e-06	-0.37444564222126353	2.744098679452467e-06
24	-0.37444564222126353	9.895019896788426e-07	-0.37444465271927385	1.647814738270359e-06
25	-0.37444465271927385	-5.941897863737111e-07	-0.3744452469090602	9.895019896788426e-07
26	-0.3744452469090602	3.568071592630062e-07	-0.37444489010190096	5.941897863737111e-07
27	-0.37444489010190096	-2.1426045238026603e-07	-0.37444510436235334	3.568071592630062e-07
28	-0.37444510436235334	1.28662038245686e-07	-0.3744449757003151	2.1426045238026603e-07
29	-0.3744449757003151	-7.726074024994034e-08	-0.37444505296105535	1.28662038245686e-07
30	-0.37444505296105535	4.63945839523916e-08	-0.3744450065664714	7.726074024994034e-08

5.5 Newton

5.5.1 Pseudo-code

```
FUNCTION NewtonRaphson(f, df, x0, tol=1e-7, max_iter=30):

    INPUT:
    - f: function whose zero (root) is to be found.
    - df: derivative of the function f.
    - x0: initial guess for the root.
    - tol: tolerance for convergence (default value is 1e-7).
    - max_iter: maximum number of iterations allowed (default
        value is 30).

    OUTPUT:
    - x: approximate root of the function f.
    - n: number of iterations performed.
    - result_array: table with details of each iteration.

    BEGIN:
    1. Initialize xn = x0 (initial estimate).
    2. Create an empty list result_array to store details of the
        iterations.

    3. For n from 0 to max_iter - 1 DO:

        a. Compute fxn = f(xn) and dfxn = df(xn) (the function and
             its derivative evaluated at xn).

        b. IF |xn - (xn - fxn / dfxn)| < tol:
           - Store in result_array the details of the current
               iteration (n, xi, f(xi), error).
           - Print the DataFrame with iteration results.
           - Return xn as the approximate root.
           - End the function.

        c. IF dfxn == 0 (the derivative is zero):
           - Print "Derivative is zero. No solution found."
           - Return None (indicating no solution found).

        d. Store in result_array the details of the current
            iteration (n, xi, f(xi), error).

        e. Update xn using the Newton-Raphson formula:
           - xn = xn - fxn / dfxn.

    4. IF the maximum number of iterations is exceeded:
       - Print "Maximum number of iterations exceeded. No solution
            found."
       - Return None.

END FUNCTION
```

5.5.2 Method Implementation

Python

Rust

5.5.3 Method Test

- The root is: 0.9364045808526077
- Found solution after 4 iterations.

i:	xi:	f_xi:	E
0	1.2	0.12524132043913228	0.3464852988384717
1	0.8535147011615283	-0.05025900826059804	0.07953835548275334
2	0.9330530566442816	-0.0019446986420873502	0.003344827656164062
3	0.9363978843004457	-3.877863362977685e-06	6.696552162011038e-06
4	0.9364045808526077	-1.5608903058961232e-11	2.6954660725664326e-11

5.6 Secant

5.6.1 Pseudo-code

```
FUNCTION SecantMethod(f, x0, x1, tol=1e-7, max_iter=100)

  INPUT:
    - f: function for which we want to find a root.
    - x0: first initial value.
    - x1: second initial value.
    - tol: tolerance to determine convergence (optional, default
        value 1e-7).
    - max_iter: maximum number of iterations allowed (optional,
        default value 100).

  OUTPUT:
    - The approximate root of the function f or an error message
        if it does not converge.

  Initialize an empty list called results.

  FOR i FROM 0 TO max_iter - 1 DO:
    1. Compute f(x0) and f(x1).
    2. IF f(x1) is equal to f(x0), throw an error: "Division by
        zero".
    3. Use the secant method formula to compute x2:
       x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0)).
    4. IF |x2 - x1| < tol, DO the following:
        - Add to the results list the index i, the value x1, the
            value f(x2), and the error |x2 - x1|.
       - Print the results list.
       - Print: "Converged after i + 1 iterations."
       - Return x2 (the root).
    5. IF not converged, add the current result to the list with
        the index i, the value x1, the value f(x2), and the error
        |x2 - x1|.
    6. Update x0 = x1 and x1 = x2 for the next iteration.

  IF the maximum number of iterations is reached, throw an error:
       "The secant method did not converge within the maximum
      number of iterations."

END FUNCTION
```

5.6.2 Method Implementation

Python

Rust

5.6.3 Method Test

- Converged after 5 iterations.
- The root is: 0.9364045808795616

i	x_i	f_xi	e
0	1.0	0.946166222306525	0.05383377769347497
1	0.946166222306525	0.9359965807911726	0.010169641515352379
2	0.9359965807911726	0.9364070023767038	0.00041042158553117325
3	0.9364070023767038	0.9364045814731196	2.420903584265943e-06
4	0.9364045814731196	0.9364045808795616	5.935579805438351e-10

5.7 Multiple Roots

5.7.1 Pseudo-code

```
FUNCTION MultipleRootMethod(f, df, ddf, x0, tol=1e-7, max_iter
    =100):

    INPUT:
    - f: function for which the root is sought.
    - df: derivative of the function f.
    - ddf: second derivative of the function f.
    - x0: initial approximation of the root.
    - tol: tolerance for convergence (optional, default value 1e
        -7).
    - max_iter: maximum number of iterations allowed (optional,
        default value 100).

    OUTPUT:
    - x: approximate root of the function f.
    - iterations: number of iterations performed.
    - data_frame: table with details of each iteration (optional).

    BEGIN:

    1. Create an empty list to store results of each iteration (
        result_array).

    2. FOR i from 0 to max_iter DO:

        a. Evaluate f(x0), f'(x0) and f''(x0):
           - f_x0 = f(x0)
           - df_x0 = df(x0)
           - ddf_x0 = ddf(x0)

        b. Check if the denominator is zero:
           - IF [df_x0^2 - f_x0 * ddf_x0] equals 0, throw an error
               ("Division by zero in multiple root method").

        c. Calculate the new value of x using the multiple root
            method formula:
           - x1 = x0 - (f_x0 * df_x0) / (df_x0^2 - f_x0 * ddf_x0)

        d. Calculate the error as the absolute difference:
           - error = abs(x1 - x0)

        e. Store the results of the current iteration in the list:
           - Save (i, x0, f_x0, df_x0, ddf_x0, x1, error) in
              result_array.

        f. Check if the error is less than the tolerance (tol):
           - IF error < tol:
             - Create a DataFrame with the results from
                 result_array.
             - Print the DataFrame.
             - Return x1, number of iterations (i + 1), and the
                 DataFrame.
        g. Update x0 for the next iteration:
           - x0 = x1

    3. IF the maximum number of iterations is reached without
        converging, throw an error ("The multiple root method did
        not converge within the maximum number of iterations").

END FUNCTION
```

5.7.2 Method Implementation

Python

Rust

5.7.3 Method Test

- Converged after 5 iterations.
- Root found: -4.218590698935789e-11

i	x_i	f(x_i)	Error
0	1.0	0.7182818284590451	1.2342106135535142
1	-0.23421061355351425	0.025405775475345838	0.22575233364275316
2	-0.00845827991076109	3.567060801401567e-05	0.008446389726952502
3	-1.1890183808588653e-05	7.068789997788372e-11	1.1890141622681664e-05
4	-4.218590698935789e-11	0.0	0.0

5.8 Gaussian Elimination No Pivot

5.8.1 Pseudo-code

```
FUNCTION GaussianEliminationWithoutPivoting(A, b):

    INPUT:
    - A: coefficient matrix (nxn).
    - b: right-hand side vector (nx1).

    OUTPUT:
    - x: solution vector to the system of equations Ax = b.

    BEGIN:
    1. Convert A and b to float matrices, if necessary.

    2. Create the augmented matrix by combining A with b:
       - AugmentedMatrix = [A | b]
    3. FOR i from 0 to n-1 DO: (n is the length of b)

        a. Check if the diagonal pivot AugmentedMatrix[i, i] is 0.
             If so, throw an error ("Zero pivot found in row i").

        b. **Forward elimination**: For each row j from i+1 to n
            -1:
            - Calculate the elimination factor: factor =
                AugmentedMatrix[j, i] / AugmentedMatrix[i, i].
            - Subtract factor * row i from row j to eliminate the
                elements below the diagonal in column i.

    4. **Back substitution**:

        a. Initialize the solution vector x of length n with zeros
            .

        b. FOR i from n-1 to 0 DO: (traverse from bottom to top)
           - Calculate x[i] using the formula:
             x[i] = (AugmentedMatrix[i, -1] - sum of the products
                 of the already solved elements) / AugmentedMatrix[
                 i, i]

    5. Return the solution vector x.

END FUNCTION
```

5.8.2 Method Implementation

Python

Rust

5.8.3 Method Test

```
           Initial augmented matrix:
[[ 2. -1. 0. 3. 1. ]
 [ 1. 0.5 3. 8. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [14. 5. -2. 3. 1. ]]
==================================================
Augmented matrix after eliminating row 0:
[[ 2. -1. 0. 3. 1. ]
 [ 0. 1. 3. 6.5 0.5]
 [ 0. 13. -2. 11. 1. ]
 [ 0. 12. -2. -18. -6. ]]
==================================================
Augmented matrix after eliminating row 1:
[[ 2. -1. 0. 3. 1. ]
 [ 0. 1. 3. 6.5 0.5]
 [ 0. 0. -41. -73.5 -5.5]
 [ 0. 0. -38. -96. -12. ]]
==================================================
Augmented matrix after eliminating row 2:
[[ 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]]
==================================================
Augmented matrix after eliminating row 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]]
==================================================
Solution vector x:
[ 0.03849519 -0.18022747 -0.30971129 0.24759405]
==================================================
```

5.9 Gaussian Elimination Partial Pivot

5.9.1 Pseudo-code

```
FUNCTION GaussianEliminationWithPartialPivoting(A, b):

    INPUT:
    - A: coefficient matrix (nxn).
    - b: right-hand side vector (nx1).

    OUTPUT:
    - x: solution vector to the system of equations Ax = b.

    BEGIN:
    1. Convert A and b to float matrices, if necessary.

    2. Create the augmented matrix by combining A with b:
       - AugmentedMatrix = [A | b]

    3. FOR i from 0 to n-1 DO: (n is the length of b)
```

```
        a. **Partial Pivoting**: Find the row with the largest
            absolute value in column i from row i to the last.
           - max_row = index of the row with the maximum value in
               column i.
           - If the value in AugmentedMatrix[max_row, i] is 0,
               throw an error ("The matrix is singular or nearly
               singular").

        b. If the row max_row is not equal to i:
           - Swap row i with row max_row.

        c. **Forward elimination**: For each row j from i+1 to n
            -1:
            - Calculate the elimination factor: factor =
                AugmentedMatrix[j, i] / AugmentedMatrix[i, i].
            - Subtract factor * row i from row j to eliminate the
                elements below the diagonal in column i.

    4. **Back substitution**:

        a. Initialize the solution vector x of length n with zeros
            .

        b. FOR i from n-1 to 0 DO: (traverse from bottom to top)
           - Calculate x[i] using the formula:
             x[i] = (AugmentedMatrix[i, -1] - sum of the products
                 of the already solved elements) / AugmentedMatrix[
                 i, i]

    5. Return the solution vector x.

END FUNCTION
```

5.9.2 Method Implementation

Python

Rust

5.9.3 Method Test

```
           [[ 2. -1. 0. 3. 1. ]
 [ 1. 0.5 3. 8. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [14. 5. -2. 3. 1. ]]
==================================================
Augmented matrix after swapping row 0 with row 3:
[[14. 5. -2. 3. 1. ]
 [ 1. 0.5 3. 8. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [ 2. -1. 0. 3. 1. ]]
==================================================
Augmented matrix after eliminating row 3:
[[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]]
==================================================
Augmented matrix after swapping row 1 with row 2:
[[14. 5. -2. 3. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [ 0. 0.14285714 3.14285714 7.78571429 0.92857143]
 [ 0. -1.71428571 0.28571429 2.57142857 0.85714286]]
==================================================
Augmented matrix after eliminating row 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 2.19780220e-02 4.02197802e+00
   9.89010989e-01]]
==================================================
Augmented matrix after swapping row 2 with row 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]]
==================================================
Augmented matrix after eliminating row 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]]
=====================================================
Augmented matrix after swapping row 3 with row 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]]
=====================================================
Augmented matrix after eliminating row 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]]
=====================================================
Solution vector x:
[ 0.03849519 -0.18022747 -0.30971129 0.24759405]
=====================================================
Solution: [ 0.03849519 -0.18022747 -0.30971129 0.24759405]
```

5.10 Gaussian Elimination

5.10.1 Pseudo-code

5.10.2 Method Implementation

Python

Rust

5.10.3 Method Test

```
           Initial augmented matrix:
[[ 2. -1. 0. 3. 1. ]
 [ 1. 0.5 3. 8. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [14. 5. -2. 3. 1. ]]
```

```
====================================================
Augmented matrix after swapping row 0 with row 3:
[[14. 5. -2. 3. 1. ]
 [ 1. 0.5 3. 8. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [ 2. -1. 0. 3. 1. ]]
====================================================
Augmented matrix after eliminating row 1:
[[14. 5. -2. 3. 1. ]
 [ 0. 0.14285714 3.14285714 7.78571429 0.92857143]
 [ 0. 13. -2. 11. 1. ]
 [ 2. -1. 0. 3. 1. ]]
====================================================
Augmented matrix after eliminating row 2:
[[14. 5. -2. 3. 1. ]
 [ 0. 0.14285714 3.14285714 7.78571429 0.92857143]
 [ 0. 13. -2. 11. 1. ]
 [ 2. -1. 0. 3. 1. ]]
====================================================
Augmented matrix after eliminating row 3:
[[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]]
====================================================
Augmented matrix after swapping row 1 with row 2:
[[14. 5. -2. 3. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [ 0. 0.14285714 3.14285714 7.78571429 0.92857143]
 [ 0. -1.71428571 0.28571429 2.57142857 0.85714286]]
====================================================
Augmented matrix after eliminating row 2:
[[14. 5. -2. 3. 1. ]
 [ 0. 13. -2. 11. 1. ]
 [ 0. 0. 3.16483516 7.66483516 0.91758242]
 [ 0. -1.71428571 0.28571429 2.57142857 0.85714286]]
====================================================
Augmented matrix after eliminating row 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 2.19780220e-02 4.02197802e+00
   9.89010989e-01]]
```

```
==================================================
Augmented matrix after swapping row 2 with row 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]]
==================================================
Augmented matrix after eliminating row 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]]
==================================================
Augmented matrix after swapping row 3 with row 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]]
==================================================
Solution vector x:
[ 0.03849519 -0.18022747 -0.30971129 0.24759405]
==================================================
Solution: [ 0.03849519 -0.18022747 -0.30971129 0.24759405]
```