Method Results

Carlos Restrepo   Jacobo Montes   Maria Clara Medina   Alejandro Rendón

November 2025

1. Incremental search method

Incremental search Method Pseudocode:

```
BEGIN

 READ x0, deltaX, f, maxIterations
 roots = []

 WHILE i < maxIterations DO:

  x1 = x0 + deltaX

  IF f(x0)*f(x1) < 0 THEN
   PRINT In [x0,x1] there is at least one root
   Add Interval to roots[n]  § n is the last position of roots[]
  ELSE THEN
   x0=x1
  END IF
        i = i + 1

 END WHILE

 PRINT Table
 PRINT Roots

END
```

Incremental search Method Results:

i	x_i	$f(x_i)$
0	-2.450000000	-6.7e-03
1	-1.950000000	3.7e-02

In the interval [-2.450000000,-1.950000000 there is at least one root

Table 1: Iterations of the Newton method for $f(x) = \ x^2 - x + 1 - e^x + \frac{99}{200}$

2. Bisection method

Bisection Pseudocode:

```
BEGIN

 READ a, b, f, tolerance, maxIterations
 cbefore = (a+b)/2

 WHILE i < maxIterations DO

  IF f(a) * f(cbefore) < 0 THEN
   b = cbefore
  ELSE THEN
   a = cbefore

  c = (a+b)/2
  abs_error = |c-cbefore|

  IF abs_error < tolerance THEN
   PRINT Root
   BREAK WHILE
  ELSE THEN
   cbefore = c
  END IF

  i = i + 1
  END WHILE
  PRINT Table

END
```

Bisection Method Results:

i	x_i	$f(x_i)$	**Error**
0	0.250000000	2.3e-02	-
1	0.250000000	2.3e-02	3.8e-01
2	0.625000000	-6.1e-01	1.9e-01
3	0.437500000	-3.0e-01	9.4e-02
4	0.343750000	-1.4e-01	4.7e-02
5	0.296875000	-5.9e-02	2.3e-02
6	0.273437500	-1.8e-02	1.2e-02
7	0.261718750	2.6e-03	5.9e-03
8	0.267578125	-7.8e-03	2.9e-03
9	0.264648438	-2.6e-03	1.5e-03
10	0.263183594	1.6e-05	7.3e-04
11	0.263916016	-1.3e-03	3.7e-04
12	0.263549805	-6.3e-04	1.8e-04
13	0.263366699	-3.1e-04	9.2e-05
14	0.263275146	-1.5e-04	4.6e-05
15	0.263229370	-6.5e-05	2.3e-05
16	0.263206482	-2.4e-05	1.1e-05
17	0.263195038	-3.9e-06	5.7e-06
18	0.263189316	6.3e-06	2.9e-06
19	0.263192177	1.2e-06	1.4e-06
20	0.263193607	-1.3e-06	7.2e-07

An approximation of the root was found at $x \approx 0.263193607$

Table 2: Iterations of the Newton method for $f(x) = x^2 - x + 1 - e^x + \frac{99}{200}$

3. False position method

False position Method Pseudocode:

```
BEGIN

 READ a, b, f, tolerance, maxIterations
 cbefore = b - ((f(b)*(b-a))/(f(b)-f(a))

 WHILE i < maxIterations DO
  c = cbefore
  IF f(a) * f(cbefore) < 0 THEN
   b = c
  ELSE THEN
   a = c
  END IF

  c = b - ((f(b)*(b-a))/(f(b)-f(a))
  abs_error = |c-cbefore|
  PRINT Table

  IF abs_error < tolerance THEN
   PRINT Root = c
   BREAK WHILE
  ELSE THEN
   cbefore = c
  END IF

 i = i + 1
 END WHILE

END
```

False position method Results:

i	x_i	$f(x_i)$	**Error**
0	1.333333333	-9.6e-01	-
1	1.333333333	-9.6e-01	3.1e-01
2	1.642857143	7.9e-01	1.4e-01
3	1.503250975	-1.1e-01	1.7e-02
4	1.519780718	-9.5e-03	1.5e-03
5	1.521239851	-8.3e-04	1.3e-04
6	1.521367483	-7.3e-05	1.1e-05
7	1.521378639	-6.3e-06	9.7e-07
8	1.521379613	-5.5e-07	8.5e-08

An approximation of the root was found at $x \approx 1.521379613$

Table 3: Iterations of the Newton method for $f(x) = x^3 - x - 2$

4. Multiple roots method

Multiple roots (Modified Newton) pseudocode:

```
BEGIN

    READ x0, tolerance, maxIterations, f(x), f'(x), f''(x)

    WHILE i < maxIterations DO:

        fx  = f(x0)
        fpx = f'(x0)
        fppx = f''(x0)
        num = fx * fpx
        den = (fpx)^2 - fx * fppx

        IF |num| < 1e-30 THEN
            PRINT Root found at x0
            BREAK WHILE
        END IF

        IF |den| < 1e-15 THEN
            PRINT Denominator near zero, cannot continue safely
            BREAK WHILE
        END IF
        x1 = x0 - num / den
        E = |x1 - x0|
        PRINT Table

        IF E < tolerance THEN
            PRINT Root found at x1
            BREAK WHILE
        END IF

        x0 = x1
        i = i + 1
    END WHILE
END
```

Multiple roots results:

We solve the system using:

$$h(x) = e^x - x - 1, \quad h'(x) = e^x - 1, \quad h''(x) = e^x, \quad x_0 = 1,\ \text{tol} = 10^{-7}.$$

iter	x_i	$f(x_i)$	Error
0	1.0000000000	7.2e-01	
1	-0.2342106136	2.5e-02	1.2e+00
2	-0.0084582799	3.6e-05	2.3e-01
3	-0.0000118902	7.1e-11	8.4e-03
4	-4.2264e-11	0.0e+00	1.2e-05
5	-4.2264e-11	0.0e+00	0.0e+00
An approximation of the root was found at $x \approx -0.000000000042264$			

Table 4: Iterations of the multiple roots method for $h(x) = e^x - x - 1$.

5. Newton Method

Newton Method Pseudocode:

```
BEGIN
    READ f, df, x0, tol, maxIterations
    x = x0

    WHILE i < maxIterations DO
        fx = f(x)

        IF |fx|< tol THEN
            PRINT Root found at x
            BREAK WHILE
        END IF
        x_prev = x
        x = x - fx / df(x)
        E = |x-x_prev|
        i = i + 1
    END WHILE
    PRINT Table

END
```

Newton Method Results:

i	x_i	$f(x_i)$	**Error**
0	0.500000000	-2.9e-01	-
1	0.928391990	-4.7e-03	4.3e-01
2	0.936366741	-2.2e-05	8.0e-03
3	0.936404580	-5.0e-10	3.8e-05

An approximation of the root was found at $x \approx 0.9364045800189902$

Table 5: Iterations of the Newton method for $f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$

6. Fixed Point Method Results

Fixed Point Pseudocode:

```
BEGIN
    READ g, x0, tol, maxIterations
    x = x0

    WHILE i < maxIterations DO
        x_new = g(x)

        IF |x_new - x| < tol THEN
            PRINT Root found at x_new
        END IF
        x = x_new

    END WHILE
    PRINT Table

END
```

Fixed Point Results:

i	x_i	$g(x_i)$	$f(x_i)$	**E**
0	-0.50000000	-0.29310873	2.07e-01	-
1	-0.29310873	-0.41982154	-1.27e-01	1.27e-01
2	-0.41982154	-0.34630452	7.35e-02	7.35e-02
3	-0.34630452	-0.39095846	-4.47e-02	4.47e-02
4	-0.39095846	-0.36440503	2.66e-02	2.66e-02
5	-0.36440503	-0.38042630	-1.60e-02	1.60e-02
6	-0.38042630	-0.37083680	9.59e-03	9.59e-03
7	-0.37083680	-0.37660565	-5.77e-03	5.77e-03
8	-0.37660565	-0.37314542	3.46e-03	3.46e-03
9	-0.37314542	-0.37522464	-2.08e-03	2.08e-03
10	-0.37522464	-0.37397659	1.25e-03	1.25e-03
11	-0.37397659	-0.37472622	-7.50e-04	7.50e-04
12	-0.37472622	-0.37427613	4.50e-04	4.50e-04
13	-0.37427613	-0.37454643	-2.70e-04	2.70e-04
14	-0.37454643	-0.37438413	1.62e-04	1.62e-04
15	-0.37438413	-0.37448159	-9.75e-05	9.75e-05
16	-0.37448159	-0.37442307	5.85e-05	5.85e-05
17	-0.37442307	-0.37445821	-3.51e-05	3.51e-05
18	-0.37445821	-0.37443711	2.11e-05	2.11e-05
19	-0.37443711	-0.37444978	-1.27e-05	1.27e-05
20	-0.37444978	-0.37444217	7.61e-06	7.61e-06
21	-0.37444217	-0.37444674	-4.57e-06	4.57e-06
22	-0.37444674	-0.37444399	2.74e-06	2.74e-06
23	-0.37444399	-0.37444564	-1.65e-06	1.65e-06
24	-0.37444564	-0.37444465	9.90e-07	9.90e-07
25	-0.37444465	-0.37444525	-5.94e-07	5.94e-07
26	-0.37444525	-0.37444489	3.57e-07	3.57e-07
27	-0.37444489	-0.37444510	-2.14e-07	2.14e-07
28	-0.37444510	-0.37444498	1.29e-07	1.29e-07
29	-0.37444498	-0.37444505	-7.73e-08	7.73e-08

An approximation of the root was found at $x \approx -0.3744449757$

Table 6: Iterations of the fixed-point method for $f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2} - x$

7. Secant Results

Secant Method pseudocode:

```
BEGIN
    READ f, x0, x1, tol, maxIterations

    WHILE i < maxIterations DO
        IF |f(x1)| < tol THEN
            PRINT root found at x1
        END IF
        x_new = x1 - f(x1)*(x1 - x0)/(f(x1) - f(x0))
        x0 = x1
        x1 = x_new

    END WHILE
    PRINT Table

END
```

Secant Method results:

Iteration	x_i	$f(x_i)$	**Error**
0	0.500000000	-2.9e-01	-
1	1.000000000	3.5e-02	-
2	0.946166222	5.6e-03	5.4e-02
3	0.935996581	-2.4e-04	1.0e-02
4	0.936407002	1.4e-06	4.1e-04
5	0.936404581	3.4e-10	2.4e-06
6	0.936404581	-5.0e-16	5.9e-10

An approximation of the root was found at $x \approx 0.9364045808795615$

Table 7: Iterations of the Secant method for $f(x) = \ln(\sin^2(x) + 1) - \frac{1}{2}$

8. Gaussian Elimination Results

Gaussian Elimination pseudocode:

```
BEGIN
    READ A[1..n,1..n+1]

    § Forward elimination
    FOR k = 1 TO n-1 DO
        FOR i = k+1 TO n DO
            factor = A[i,k] / A[k,k]
            FOR j = k TO n+1 DO
                A[i,j] = A[i,j] - factor * A[k,j]
            END FOR
        END FOR
    END FOR

    § Back substitution
    x[n] = A[n,n+1] / A[n,n]
    FOR i = n-1 DOWN TO 1 DO
        sum = 0
        FOR j = i+1 TO n DO
            sum = sum + A[i,j]*x[j]
        END FOR
        x[i] = (A[i,n+1] - sum)/A[i,i]
    END FOR
    RETURN x
    PRINT Steps

END
```

Gaussian Elimination results:
We solve the system using Gaussian elimination on the augmented matrix:

$$[A|b] = \begin{bmatrix} 2 & -1 & 0 & 3 & 1 \\ 1 & 0.5 & 3 & 8 & 1 \\ 0 & 13 & -2 & 11 & 1 \\ 14 & 5 & -2 & 3 & 1 \end{bmatrix}$$

Step 0: Initial augmented matrix

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

Step 1: Zeroing first column below pivot

$$\begin{bmatrix} 2 & -1 & 0 & 3 & 1 \\ 0 & 1 & 3 & 6.5 & 0.5 \\ 0 & 13 & -2 & 11 & 1 \\ 0 & 12 & -2 & -18 & -6 \end{bmatrix}$$

Step 2: Zeroing second column below pivot

$$\begin{bmatrix} 2 & -1 & 0 & 3 & 1 \\ 0 & 1 & 3 & 6.5 & 0.5 \\ 0 & 0 & -41 & -73.5 & -5.5 \\ 0 & 0 & -38 & -96 & -12 \end{bmatrix}$$

Step 3: Zeroing third column below pivot

$$\begin{bmatrix} 2 & -1 & 0 & 3 & 1 \\ 0 & 1 & 3 & 6.5 & 0.5 \\ 0 & 0 & -41 & -73.5 & -5.5 \\ 0 & 0 & 0 & -27.878 & -6.9024 \end{bmatrix}$$

Solutions (back substitution)

$$x_4 \approx 0.2476$$
$$x_3 \approx -0.3097$$
$$x_2 \approx -0.1802$$
$$x_1 \approx 0.0385$$

9. Partial pivoting method

Partial pivoting pseudocode:

```
BEGIN
    READ A[1..n,1..n+1]

    FOR k = 1 TO n-1 DO
        pivot_row = k
        FOR i = k TO n DO
            IF |A[i,k]| > |A[pivot_row,k]| THEN
                pivot_row = i
            END IF
        END FOR

        IF |A[pivot_row,k]| < tol THEN
            PRINT Pivot  0 in column k
            FOR BREAK
        END IF

        IF pivot_row  k THEN
            SWAP rows A[k], A[pivot_row]
        END IF

        piv = A[k,k]
        FOR i = k+1 TO n DO
            factor = A[i,k] / piv
            FOR j = k TO n+1 DO
                A[i,j] = A[i,j] - factor * A[k,j]
            END FOR
        END FOR
    END FOR

    § Back substitution
    x[n] = A[n,n+1] / A[n,n]
    FOR i = n-1 DOWN TO 1 DO
        sum = 0
        FOR j = i+1 TO n DO
            sum = sum + A[i,j]*x[j]
        END FOR
        x[i] = (A[i,n+1] - sum)/A[i,i]
    END FOR
    RETURN x
    PRINT Steps

END
```

Partial pivoting results:

We solve the system using partial pivoting on the augmented matrix:

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

Step 0: Initial augmented matrix

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

Step 1

$$\begin{bmatrix} 14 & 5 & -2 & 3 & 1 \\ 0 & 0.142857 & 3.142857 & 7.785714 & 0.928571 \\ 0 & 13 & -2 & 11 & 1 \\ 0 & -1.714286 & 0.285714 & 2.571429 & 0.857143 \end{bmatrix}$$

Step 2

$$\begin{bmatrix} 14 & 5 & -2 & 3 & 1 \\ 0 & 13 & -2 & 11 & 1 \\ 0 & 0 & 3.164835 & 7.664835 & 0.917582 \\ 0 & 0 & 0.021978 & 4.021978 & 0.989011 \end{bmatrix}$$

Step 3

$$\begin{bmatrix} 14 & 5 & -2 & 3 & 1 \\ 0 & 13 & -2 & 11 & 1 \\ 0 & 0 & 3.164835 & 7.664835 & 0.917582 \\ 0 & 0 & 0 & 3.968750 & 0.982639 \end{bmatrix}$$

Back substitution

After performing backward substitution, the final solution vector is:

$$x = \begin{pmatrix} 0.038495 \\ -0.180227 \\ -0.309711 \\ 0.247594 \end{pmatrix}.$$

10. Total pivoting method

Total pivoting pseudocode:

```
BEGIN
    READ A[1..n,1..n+1]

    FOR k = 1 TO n-1 DO
        max_val = 0
        p = k
        q = k

        FOR i = k TO n DO
            FOR j = k TO n DO
                IF |A[i,j]| > max_val THEN
                    max_val = |A[i,j]|
                    p = i
                    q = j
                END IF
            END FOR
        END FOR

        IF max_val < 1e-16 THEN
            PRINT Submatrix k is singular or nearly singular
            BREAK FOR
        END IF

        IF p  k THEN
            SWAP rows A[k], A[p]
        END IF

        IF q  k THEN
            SWAP columns A[k], A[q]
            perm[k]  perm[q]
        END IF

        piv = A[k,k]
        FOR i = k+1 TO n DO
            factor = A[i,k] / piv
            FOR j = k TO n+1 DO
                A[i,j] = A[i,j] - factor * A[k,j]
            END FOR
        END FOR
    END FOR

    x[n] = A[n,n+1] / A[n,n]
    FOR i = n-1 DOWN TO 1 DO
        sum = 0
        FOR j = i+1 TO n DO
            sum = sum + A[i,j]*x[j]
        END FOR
        x[i] = (A[i,n+1] - sum)/A[i,i]
    END FOR

    FOR j = 1 TO n DO
        x_final[perm[j]] = x[j]
    END FOR

    RETURN x_final
    PRINT Steps

END
```

Total pivoting results:

We solve the system using total pivoting on the augmented matrix:

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

Step 0: Initial augmented matrix

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

Step 1

$$\begin{bmatrix} 14 & 5 & -2 & 3 & 1 \\ 0 & 0.142857 & 3.142857 & 7.785714 & 0.928571 \\ 0 & 13 & -2 & 11 & 1 \\ 0 & -1.714286 & 0.285714 & 2.571429 & 0.857143 \end{bmatrix}$$

Step 2

$$\begin{bmatrix} 14 & 5 & -2 & 3 & 1 \\ 0 & 13 & -2 & 11 & 1 \\ 0 & 0 & 3.164835 & 7.664835 & 0.917582 \\ 0 & 0 & 0.021978 & 4.021978 & 0.989011 \end{bmatrix}$$

Step 3

$$\begin{bmatrix} 14 & 5 & 3 & -2 & 1 \\ 0 & 13 & 11 & -2 & 1 \\ 0 & 0 & 7.664835 & 3.164835 & 0.917582 \\ 0 & 0 & 0 & -1.638710 & 0.507527 \end{bmatrix}$$

Back substitution

After performing backward substitution, we obtain the solution vector:

$$x = \begin{pmatrix} 0.038495 \\ -0.180227 \\ -0.309711 \\ 0.247594 \end{pmatrix}.$$

11. Gaussian elimination with LU decomposition

Gaussian elimination with LU decomposition pseudocode:

```
BEGIN
    READ A, b

    FOR k = 0 TO n - 2 DO
        piv = Awork[k, k]

        IF |piv| < 1e-12 THEN
            PRINT Zero pivot at row k; pivoting required
            RETURN
        END IF

        FOR i = k + 1 TO n - 1 DO
            m = Awork[i, k] / piv
            L[i, k] = m
            FOR j = k TO n - 1 DO
```

```
                Awork[i, j] = Awork[i, j] - m * Awork[k, j]
            END FOR
        END FOR

        FOR j = k TO n - 1 DO
            U[k, j] = Awork[k, j]
        END FOR

        IF k + 1 < n - 1 THEN
            FOR j = k + 1 TO n - 1 DO
                U[k + 1, j] = Awork[k + 1, j]
            END FOR
        ELSE IF k + 1 = n - 1 THEN
            U[n - 1, n - 1] = Awork[n - 1, n - 1]
        END IF

        PRINT Stage
    END FOR

    § Solve Ly = b using Forward Substitution
    FOR i = 0 TO n - 1 DO
        s = 0
        FOR j = 0 TO i - 1 DO
            s = s + L[i, j] * y[j]
        END FOR
        y[i] = (b[i] - s) / L[i, i]
    END FOR

    § Solve Ux = y using Back Substitution
    FOR i = n - 1 DOWNTO 0 DO
        s = 0
        FOR j = i + 1 TO n - 1 DO
            s = s + U[i, j] * x[j]
        END FOR
        x[i] = (y[i] - s) / U[i, i]
    END FOR

    PRINT Results
END
```

Gaussian elimination with LU decomposition results:

We solve the system using LU decomposition

Stage 0

$$\begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 1.000000 & 15.500000 & 3.000000 & 8.000000 \\ 0.000000 & -1.300000 & -4.000000 & 1.100000 \\ 14.000000 & 5.000000 & -2.000000 & 30.000000 \end{bmatrix}$$

Stage 1

$$\begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & -1.300000 & -4.000000 & 1.100000 \\ 0.000000 & 8.500000 & -2.000000 & 19.500000 \end{bmatrix}$$

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 3.500000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 \end{bmatrix}$$

Stage 2

$$\begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & -3.752381 & 1.698413 \\ 0.000000 & 0.000000 & -3.619048 & 15.587302 \end{bmatrix}$$

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & -0.082540 & 1.000000 & 0.000000 \\ 3.500000 & 0.539683 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & -3.752381 & 1.698413 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 \end{bmatrix}$$

Stage 3

$$\begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & -3.752381 & 1.698413 \\ 0.000000 & 0.000000 & 0.000000 & 13.949239 \end{bmatrix}$$

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & -0.082540 & 1.000000 & 0.000000 \\ 3.500000 & 0.539683 & 0.964467 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & -3.752381 & 1.698413 \\ 0.000000 & 0.000000 & 0.000000 & 13.949239 \end{bmatrix}$$

Forward and backward substitution

After the substitutions, we obtain:

$$x = \begin{pmatrix} 0.525109 \\ 0.255459 \\ -0.410480 \\ -0.281659 \end{pmatrix}$$

12. LU factorization with partial pivoting

LU factorization with partial pivoting pseudocode:

```
BEGIN
    READ A, b
    n = number of rows in A
    § Awork = copy of A
    § L = identity matrix of size n×n
    § U = zero matrix of size n×n
    § P = identity matrix of size n×n

    FOR k = 0 TO n - 2 DO

        pivot_row = k + index of maximum |Awork[i, k]| for i = k TO n - 1

        IF |Awork[pivot_row, k]| < 1e-12 THEN
            PRINT Zero or near-zero pivot at column, k
            RETURN
        END IF

        IF pivot_row  k THEN
            -- Swap rows in A
            SWAP rows k and pivot_row in Awork
            -- Swap rows in P
            SWAP rows k and pivot_row in P
            -- Swap partial columns in L (only 0..k-1)
            IF k > 0 THEN
                SWAP rows k and pivot_row in L for columns 0 TO k - 1
            END IF
        END IF

        piv = Awork[k, k]

        FOR i = k + 1 TO n - 1 DO
            m = Awork[i, k] / piv
            L[i, k] = m
            FOR j = k TO n - 1 DO
                Awork[i, j] = Awork[i, j] - m * Awork[k, j]
            END FOR
        END FOR

        -- Update U for this stage
        FOR j = k TO n - 1 DO
            U[k, j] = Awork[k, j]
        END FOR

        IF k + 1 < n - 1 THEN
            FOR j = k + 1 TO n - 1 DO
                U[k + 1, j] = Awork[k + 1, j]
            END FOR
        ELSE IF k + 1 = n - 1 THEN
            U[n - 1, n - 1] = Awork[n - 1, n - 1]
        END IF
        PRINT Stage

    END FOR
    bp = P × b

    § Forward Substitution (Ly = Pb)
    FOR i = 0 TO n - 1 DO
        s = 0
        FOR j = 0 TO i - 1 DO
```

```
            s = s + L[i, j] * y[j]
        END FOR
        y[i] = (bp[i] - s) / L[i, i]
    END FOR

    § Back Substitution (Ux = y)
    FOR i = n - 1 DOWNTO 0 DO
        s = 0
        FOR j = i + 1 TO n - 1 DO
            s = s + U[i, j] * x[j]
        END FOR
        x[i] = (y[i] - s) / U[i, i]
    END FOR
    PRINT x

END
```

LU factorization with partial pivoting results:

Stage 0

$$\begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 1.000000 & 15.500000 & 3.000000 & 8.000000 \\ 0.000000 & -1.300000 & -4.000000 & 1.100000 \\ 14.000000 & 5.000000 & -2.000000 & 30.000000 \end{bmatrix}$$

Stage 1

$$\begin{bmatrix} 14.000000 & 5.000000 & -2.000000 & 30.000000 \\ 0.000000 & 15.142857 & 3.142857 & 5.857143 \\ 0.000000 & -1.300000 & -4.000000 & 1.100000 \\ 0.000000 & -2.428571 & 0.571429 & -5.571429 \end{bmatrix}$$

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.071429 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 0.285714 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 14.000000 & 5.000000 & -2.000000 & 30.000000 \\ 0.000000 & 15.142857 & 3.142857 & 5.857143 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 \end{bmatrix}, \quad P = \begin{bmatrix} 0.00000 \\ 0.00000 \\ 0.00000 \\ 1.00000 \end{bmatrix}$$

Stage 2

$$\begin{bmatrix} 14.000000 & 5.000000 & -2.000000 & 30.000000 \\ 0.000000 & 15.142857 & 3.142857 & 5.857143 \\ 0.000000 & 0.000000 & -3.730189 & 1.602830 \\ 0.000000 & 0.000000 & 1.075472 & -4.632075 \end{bmatrix}$$

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.071429 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & -0.085849 & 1.000000 & 0.000000 \\ 0.285714 & -0.160377 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 14.000000 & 5.000000 & -2.000000 & 30.000000 \\ 0.000000 & 15.142857 & 3.142857 & 5.857143 \\ 0.000000 & 0.000000 & -3.730189 & 1.602830 \\ 0.000000 & 0.000000 & 0.000000 & 0.000000 \end{bmatrix}, \quad P = \begin{bmatrix} 0.000 \\ 0.000 \\ 0.000 \\ 1.000 \end{bmatrix}$$

Stage 3

$$\begin{bmatrix} 14.000000 & 5.000000 & -2.000000 & 30.000000 \\ 0.000000 & 15.142857 & 3.142857 & 5.857143 \\ 0.000000 & 0.000000 & -3.730189 & 1.602830 \\ 0.000000 & 0.000000 & 0.000000 & -4.169954 \end{bmatrix}$$

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.071429 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & -0.085849 & 1.000000 & 0.000000 \\ 0.285714 & -0.160377 & -0.288316 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 14.000000 & 5.000000 & -2.000000 & 30.000000 \\ 0.000000 & 15.142857 & 3.142857 & 5.857143 \\ 0.000000 & 0.000000 & -3.730189 & 1.602830 \\ 0.000000 & 0.000000 & 0.000000 & -4.169954 \end{bmatrix}, \quad P = \begin{bmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 1.0 \end{bmatrix}$$

Forward and Backward Substitution

After performing the substitutions, we obtain:

$$x = \begin{pmatrix} 0.525109 \\ 0.255459 \\ -0.410480 \\ -0.281659 \end{pmatrix}$$

13. Crout Decomposition method

Crout method pseudocode:

```
BEGIN
    READ A, b
    n = number of rows of A

    FOR k = 0 TO n - 1 DO
        FOR i = k TO n - 1 DO
            s = 0
            FOR p = 0 TO k - 1 DO
                s = s + L[i, p] * U[p, k]
            END FOR
            L[i, k] = A[i, k] - s
        END FOR

        FOR j = k + 1 TO n - 1 DO
            s = 0
            FOR p = 0 TO k - 1 DO
                s = s + L[k, p] * U[p, j]
            END FOR
            U[k, j] = (A[k, j] - s) / L[k, k]
        END FOR

        PRINT Stage k + 1
        PRINT L
        PRINT U
    END FOR

    § Solve Ly = b using Forward Substitution
    FOR i = 0 TO n - 1 DO
        s = 0
        FOR j = 0 TO i - 1 DO
```

```
            s = s + L[i, j] * y[j]
        END FOR
        y[i] = (b[i] - s) / L[i, i]
    END FOR

    § Solve Ux = y using Back Substitution
    FOR i = n - 1 DOWNTO 0 DO
        s = 0
        FOR j = i + 1 TO n - 1 DO
            s = s + U[i, j] * x[j]
        END FOR
        x[i] = (y[i] - s) / U[i, i]
    END FOR
    PRINT Results

END BEGIN
```

Crout method results:

Stage 0

$$\begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 1.000000 & 15.500000 & 3.000000 & 8.000000 \\ 0.000000 & -1.300000 & -4.000000 & 1.100000 \\ 14.000000 & 5.000000 & -2.000000 & 30.000000 \end{bmatrix}$$

Stage 1

$$L = \begin{bmatrix} 4.000000 & 0.000000 & 0.000000 & 0.000000 \\ 1.000000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 14.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 1.000000 & -0.250000 & 0.000000 & 0.750000 \\ 0.000000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Stage 2

$$L = \begin{bmatrix} 4.000000 & 0.000000 & 0.000000 & 0.000000 \\ 1.000000 & 15.750000 & 0.000000 & 0.000000 \\ 0.000000 & -1.300000 & 1.000000 & 0.000000 \\ 14.000000 & 8.500000 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 1.000000 & -0.250000 & 0.000000 & 0.750000 \\ 0.000000 & 1.000000 & 0.190476 & 0.460317 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Stage 3

$$L = \begin{bmatrix} 4.000000 & 0.000000 & 0.000000 & 0.000000 \\ 1.000000 & 15.750000 & 0.000000 & 0.000000 \\ 0.000000 & -1.300000 & -3.752381 & 0.000000 \\ 14.000000 & 8.500000 & -3.619048 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 1.000000 & -0.250000 & 0.000000 & 0.750000 \\ 0.000000 & 1.000000 & 0.190476 & 0.460317 \\ 0.000000 & 0.000000 & 1.000000 & -0.452623 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Stage 4

$$L = \begin{bmatrix} 4.000000 & 0.000000 & 0.000000 & 0.000000 \\ 1.000000 & 15.750000 & 0.000000 & 0.000000 \\ 0.000000 & -1.300000 & -3.752381 & 0.000000 \\ 14.000000 & 8.500000 & -3.619048 & 13.949239 \end{bmatrix}, \quad U = \begin{bmatrix} 1.000000 & -0.250000 & 0.000000 & 0.750000 \\ 0.000000 & 1.000000 & 0.190476 & 0.460317 \\ 0.000000 & 0.000000 & 1.000000 & -0.452623 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Forward and Backward Substitution

After performing the substitutions, we obtain:

$$x = \begin{pmatrix} 0.525109 \\ 0.255459 \\ -0.410480 \\ -0.281659 \end{pmatrix}$$

14. Doolitle Decomposition method

Doolitle method pseudocode:

```
BEGIN
    READ A, b
    n = number of rows in A
    L = Identity n
    U = Identity n

    FOR k = 1 TO n DO
        FOR j = k TO n DO
            U[k,j] = A[k,j] - SUM(L[k,p] * U[p,j]) for p = 1 TO k-1
        END FOR

        FOR i = k+1 TO n DO
            L[i,k] = (A[i,k] - SUM(L[i,p] * U[p,k]) for p = 1 TO k-1) / U[k,k]
        END FOR
    END FOR

    PRINT Stage

    § Forward substitution: L * y = b
    y = zero vector of size n
    FOR i = 1 TO n DO
        s = SUM(L[i,j] * y[j]) for j = 1 TO i-1
        y[i] = (b[i] - s) / L[i,i]
    END FOR

    § Backward substitution: U * x = y
    x = zero vector of size n
    FOR i = n DOWNTO 1 DO
        s = SUM(U[i,j] * x[j]) for j = i+1 TO n
        x[i] = (y[i] - s) / U[i,i]
    END FOR
    PRINT x

END BEGIN
```

Doolitle method results:

Stage 0

$$\begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 1.000000 & 15.500000 & 3.000000 & 8.000000 \\ 0.000000 & -1.300000 & -4.000000 & 1.100000 \\ 14.000000 & 5.000000 & -2.000000 & 30.000000 \end{bmatrix}$$

Stage 1

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 3.500000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Stage 2

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & -0.082540 & 1.000000 & 0.000000 \\ 3.500000 & 0.539683 & 0.000000 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & 1.000000 & 0.000000 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Stage 3

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & -0.082540 & 1.000000 & 0.000000 \\ 3.500000 & 0.539683 & 0.964467 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & -3.752381 & 1.698413 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \end{bmatrix}$$

Stage 4

$$L = \begin{bmatrix} 1.000000 & 0.000000 & 0.000000 & 0.000000 \\ 0.250000 & 1.000000 & 0.000000 & 0.000000 \\ 0.000000 & -0.082540 & 1.000000 & 0.000000 \\ 3.500000 & 0.539683 & 0.964467 & 1.000000 \end{bmatrix}, \quad U = \begin{bmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 0.000000 & 15.750000 & 3.000000 & 7.250000 \\ 0.000000 & 0.000000 & -3.752381 & 1.698413 \\ 0.000000 & 0.000000 & 0.000000 & 13.949239 \end{bmatrix}$$

Forward and Backward Substitution

After performing the substitutions, we obtain:

$$x = \begin{pmatrix} 0.525109 \\ 0.255459 \\ -0.410480 \\ -0.281659 \end{pmatrix}$$

15. Cholesky Decomposition method

Cholesky method pseudocode:

```
BEGIN
    READ A, b
    n = number of rows in A
    L = Identity n
    U = Identity n

    FOR k = 1 TO n DO
        Lkk = A[k,k]
        L[k,k] = Lkk
        U[k,k] = Lkk

        FOR j = k+1 TO n DO
            U[k,j] = A[k,j] / Lkk
        END FOR
        FOR i = k+1 TO n DO
            L[i,k] = A[i,k] / Lkk
        END FOR

        FOR i = k+1 TO n DO
            FOR j = k+1 TO n DO
                A[i,j] = A[i,j] - L[i,k] * U[k,j]
            END FOR
        END FOR
    END FOR

    PRINT Stage

    § Forward substitution: L * y = b
    y = zero vector of size n
    FOR i = 1 TO n DO
        s = SUM(L[i,j] * y[j]) for j = 1 TO i-1
        y[i] = (b[i] - s) / L[i,i]
    END FOR

    § Backward substitution: U * x = y
    x = zero vector of size n
    FOR i = n DOWNTO 1 DO
        s = SUM(U[i,j] * x[j]) for j = i+1 TO n
        x[i] = (y[i] - s) / U[i,i]
    END FOR

    PRINT x
END
```

Cholesky method results:
Data used:

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

Results:

Stage 0:

$$\begin{pmatrix} 4.000000 & -1.000000 & 0.000000 & 3.000000 \\ 1.000000 & 15.500000 & 3.000000 & 8.000000 \\ 0.000000 & -1.300000 & -4.000000 & 1.100000 \\ 14.000000 & 5.000000 & -2.000000 & 30.000000 \end{pmatrix}$$

Stage 1:

$$\mathbf{L} = \begin{pmatrix} 2.000000 & 0 & 0 & 0 \\ 0.500000 & 1.000000 & 0 & 0 \\ 0 & 0 & 1.000000 & 0 \\ 7.000000 & 0 & 0 & 1.000000 \end{pmatrix} \qquad \mathbf{U} = \begin{pmatrix} 2.000000 & -0.500000 & 0.000000 & 1.500000 \\ 0 & 1.000000 & 0 & 0 \\ 0 & 0 & 1.000000 & 0 \\ 0 & 0 & 0 & 1.000000 \end{pmatrix}$$

Stage 2:

$$\mathbf{L} = \begin{pmatrix} 2.000000 & 0 & 0 & 0 \\ 0.500000 & 3.968627 & 0 & 0 \\ 0 & -0.327569 & 1.000000 & 0 \\ 7.000000 & 2.141799 & 0 & 1.000000 \end{pmatrix} \qquad \mathbf{U} = \begin{pmatrix} 2.000000 & -0.500000 & 0.000000 & 1.500000 \\ 0 & 3.968627 & 0.755929 & 1.826828 \\ 0 & 0 & 1.000000 & 0 \\ 0 & 0 & 0 & 1.000000 \end{pmatrix}$$

Stage 3:

$$\mathbf{L} = \begin{pmatrix} 2.000000 & 0 & 0 & 0 \\ 0.500000 & 3.968627 & 0 & 0 \\ 0 & -0.327569 & 1.937106\,i & 0 \\ 7.000000 & 2.141799 & 1.868275\,i & 1.000000 \end{pmatrix} \qquad \mathbf{U} = \begin{pmatrix} 2.000000 & -0.500000 & 0.000000 & 1.500000 \\ 0 & 3.968627 & 0.755929 & 1.826828 \\ 0 & 0 & 1.937106\,i & -0.876778\,i \\ 0 & 0 & 0 & 1.000000 \end{pmatrix}$$

Stage 4:

$$\mathbf{L} = \begin{pmatrix} 2.000000 & 0 & 0 & 0 \\ 0.500000 & 3.968627 & 0 & 0 \\ 0 & -0.327569 & 1.937106\,i & 0 \\ 7.000000 & 2.141799 & 1.868275\,i & 3.734868 \end{pmatrix} \qquad \mathbf{U} = \begin{pmatrix} 2.000000 & -0.500000 & 0.000000 & 1.500000 \\ 0 & 3.968627 & 0.755929 & 1.826828 \\ 0 & 0 & 1.937106\,i & -0.876778\,i \\ 0 & 0 & 0 & 3.734868 \end{pmatrix}$$

After forward and backward substitution:

$$\mathbf{x} = \begin{pmatrix} 0.525109 \\ 0.255459 \\ -0.410480 \\ -0.281659 \end{pmatrix}.$$

16. Jacobi iterative method

Jacobi pseudocode:

```
BEGIN

    READ A, b, x0, tolerance, max_iter
    iter = 0
    error = 1

    § Build D, L, U
    D = diagonal matrix A
    L = lower triangular A
    U = upper triangularA


    inv_D = diagonal matrix 1 / diag(A)

    T = inv_D * (L + U)
    C = inv_D * b
    rho = SPECTRAL RADIUS T
    PRINT rho






    WHILE error > tolerance AND iter < max_iter DO

        x1 = T * x0 + C
        error = norm_inf( x1 - x0 )
        x0 = x1
        iter = iter + 1

        PRINT row iter: error and x0
    END WHILE

    IF error < tolerance THEN
        PRINT "Jacobi: solution found"
        PRINT x0
    ELSE
        PRINT "Jacobi: failed in max iterations"
    END IF

    RETURN x0, rho

END
```

Jacobi method results:

We solve the following linear system:

$$\begin{cases} 10x_1 - x_2 + 2x_3 = 6, \\ -1x_1 + 11x_2 - x_3 + 3x_4 = 25, \\ 2x_1 - x_2 + 10x_3 - x_4 = -11, \\ 3x_2 - x_3 + 8x_4 = 15. \end{cases}$$

with initial approximation $x^{(0)} = (0,0,0,0)^T$ and tolerance $\varepsilon = 10^{-7}$.

iter	x_1	x_2	x_3	x_4	**Error**
0	0.000000	0.000000	0.000000	0.000000	-
1	0.600000	2.272727	-1.100000	1.875000	2.272727e+00
2	0.939394	2.680682	-1.381818	2.096591	4.079545e-01
3	0.995886	2.746690	-1.416477	2.137179	6.600776e-02
4	1.002879	2.754940	-1.420790	2.142190	8.250347e-03
5	1.003946	2.756191	-1.421331	2.142742	1.251088e-03
6	1.004093	2.756357	-1.421401	2.142811	1.662425e-04
7	1.004112	2.756379	-1.421410	2.142821	2.120097e-05
8	1.004114	2.756381	-1.421411	2.142822	2.802789e-06
9	1.004114	2.756381	-1.421411	2.142822	3.659548e-07
Convergence achieved: $\mathbf{x} = (1.0041,\ 2.7564,\ -1.4214,\ 2.1428)$					

Table 8: Iterations of the Jacobi method for the given linear system.

17. Gauss–Seidel Iterative Method

Gauss–Seidel Pseudocode:

```
BEGIN
    READ A, b, x0, tolerance, niter
    inv_DL = INVERSE(D - L)
    T = inv_DL * U
    C = inv_DL * b
    ro = SPECTRAL RADIUS T
    error = 1
    WHILE error > tolerance AND i < niter DO
        x1 = T * x0 + C
        error = NORM_INF(x1 - x0)
        x0 = x1
        i = i + 1

    END WHILE

    IF error < tolerance THEN
        PRINT Solution found at x0
    ELSE
        PRINT Failed
    END IF

    PRINT Table

END
```

Gauss–Seidel method results: Data used:

$$A = \begin{pmatrix} 4 & -1 & 0 & 3 \\ 1 & 15.5 & 3 & 8 \\ 0 & -1.3 & -4 & 1.1 \\ 14 & 5 & -2 & 30 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \text{Tol} = 10^{-7}, \; N_{\max} = 40.$$

Results:

$$T = \begin{pmatrix} 0.000000 & 0.250000 & 0.000000 & -0.750000 \\ 0.000000 & -0.016129 & -0.193548 & -0.467742 \\ 0.000000 & 0.005242 & 0.062903 & 0.427016 \\ 0.000000 & -0.113629 & 0.036452 & 0.456425 \end{pmatrix}$$

$$C = \begin{pmatrix} 0.250000 & 0.048387 & -0.265726 & -0.109113 \end{pmatrix}$$

Spectral radius: $\rho(T) = 0.599488$

Iteration Table:

iter	$E = \|x^{(k+1)} - x^{(k)}\|_\infty$	x_1	x_2	x_3	x_4
0	–	0.000000	0.000000	0.000000	0.000000
1	2.7e-01	0.250000	0.048387	-0.265726	-0.109113
2	1.0e-01	0.343931	0.150074	-0.328780	-0.174099
3	7.4e-02	0.418093	0.191035	-0.359964	-0.217613
4	4.3e-02	0.460969	0.216763	-0.380292	-0.243265
5	2.6e-02	0.486640	0.232281	-0.392389	-0.258638
6	1.6e-02	0.501249	0.241279	-0.399511	-0.267029
7	9.9e-03	0.509642	0.246422	-0.403620	-0.271980
8	6.3e-03	0.514622	0.249552	-0.406020	-0.275024
9	4.1e-03	0.517823	0.251495	-0.407468	-0.276948
10	2.7e-03	0.519964	0.252738	-0.408358	-0.278160
11	1.9e-03	0.521431	0.253561	-0.408910	-0.278939
12	1.3e-03	0.522468	0.254123	-0.409259	-0.279444
13	9.2e-04	0.523236	0.254524	-0.409486	-0.279792
14	6.5e-04	0.523840	0.254827	-0.409640	-0.280048
15	4.6e-04	0.524350	0.255069	-0.409752	-0.280246
16	3.3e-04	0.524812	0.255273	-0.409835	-0.280409
17	2.3e-04	0.525252	0.255450	-0.409896	-0.280548
18	1.7e-04	0.525109	0.255458	-0.410135	-0.280829
19	1.2e-04	0.525109	0.255458	-0.410298	-0.280998
20	8.8e-05	0.525109	0.255458	-0.410392	-0.281100
21	6.4e-05	0.525109	0.255458	-0.410448	-0.281159
22	4.7e-05	0.525109	0.255458	-0.410481	-0.281194
23	3.4e-05	0.525109	0.255458	-0.410498	-0.281212
24	2.5e-05	0.525109	0.255458	-0.410506	-0.281221
25	1.8e-05	0.525109	0.255458	-0.410511	-0.281226
26	1.3e-05	0.525109	0.255458	-0.410515	-0.281231
27	9.4e-06	0.525109	0.255458	-0.410518	-0.281234
28	6.9e-06	0.525109	0.255458	-0.410520	-0.281237
29	1.2e-07	0.525109	0.255458	-0.410480	-0.281659
30	7.2e-08	0.525109	0.255458	-0.410480	-0.281659

Table 9: Gauss–Seidel iterations with $\| \cdot \|_\infty$ stopping rule.

18. Successive Over-Relaxation (SOR) Method

SOR pseudocode (2-norm, $w = 1.5$):

```
BEGIN
    READ A, b, x0, w, tolerance, nmax

    inv_term = INVERSE( D - w * L )

    T = inv_term * ( (1 - w) * D + w * U )
    C = w * inv_term * b

    ro = SPECTRAL RADIUS T
    xk = x0
    FOR k = 1 TO nmax DO

        x_next = T * xk + C
        error = MAX( | x_next[i] - xk[i] | )   FOR i = 1 TO n
        PRINT Table
        xk = x_next
        IF error < tolerance THEN
            PRINT Solution found at k
            PRINT xk

        END IF

    END FOR
END
```

Data to use:

$$A = \begin{pmatrix} 4 & -1 & 0 & 3 \\ 1 & 15.5 & 3 & 8 \\ 0 & -1.3 & -4 & 1.1 \\ 14 & 5 & -2 & 30 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad x^{(0)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad \text{Tol} = 10^{-7}, \quad N_{\max} = 100, \quad w = 1.5.$$

For reference, the permutation helper table is

Table =

	-1	0	3	4
15.5	3	8	1	

Affine form ($x^{(k+1)} = Tx^{(k)} + C$, $w = 1.5$)

$$T = \begin{pmatrix} -0.500000 & 0.375000 & 0.000000 & -1.125000 \\ 0.048387 & -0.536290 & -0.290323 & -0.665323 \\ -0.023589 & 0.261442 & -0.358468 & 0.736845 \\ 0.335544 & -0.102283 & 0.036734 & 0.527515 \end{pmatrix}, \qquad C = \begin{pmatrix} 0.375000 \\ 0.060484 \\ -0.404486 \\ -0.268070 \end{pmatrix}.$$

C as a row (to match console style):

$$C^\top = (\,0.375000 \;\; 0.060484 \;\; -0.404486 \;\; -0.268070\,).$$

$$\text{Spectral radius } \rho(T) = 0.631208.$$

iter	$E = \|x^{(k+1)} - x^{(k)}\|_2$	x_1	x_2	x_3	x_4
0	–	0.000000	0.000000	0.000000	0.000000
1	6.2e-01	0.375000	0.060484	-0.404486	-0.268070
2	3.2e-01	0.511760	0.341976	-0.450049	-0.304696
3	1.5e-01	0.590144	0.235228	-0.390336	-0.308594
4	1.1e-01	0.515306	0.281526	-0.444371	-0.271236
5	7.4e-02	0.528060	0.243909	-0.383605	-0.283362
6	4.3e-02	0.521218	0.255125	-0.424458	-0.279399
7	2.1e-02	0.524387	0.258003	-0.403799	-0.282252
8	1.1e-02	0.527091	0.252514	-0.412630	-0.282229
9	6.9e-03	0.523655	0.258137	-0.410946	-0.281073
10	5.5e-03	0.526181	0.253697	-0.409146	-0.282129
11	4.2e-03	0.524441	0.256380	-0.411790	-0.281318
12	2.8e-03	0.525405	0.255085	-0.409503	-0.281846
13	1.7e-03	0.525031	0.255513	-0.411073	-0.281584
14	8.8e-04	0.525084	0.255547	-0.410197	-0.281673
15	4.4e-04	0.525171	0.255337	-0.410569	-0.281674
16	2.7e-04	0.525049	0.255562	-0.410493	-0.281637
17	2.2e-04	0.525153	0.255389	-0.410431	-0.281679
18	1.7e-04	0.525083	0.255497	-0.410532	-0.281646
19	1.1e-04	0.525122	0.255443	-0.410441	-0.281667
20	6.8e-05	0.525106	0.255461	-0.410504	-0.281656
21	3.6e-05	0.525108	0.255462	-0.410469	-0.281660
22	1.8e-05	0.525111	0.255454	-0.410484	-0.281660
23	1.1e-05	0.525107	0.255463	-0.410481	-0.281659
24	8.4e-06	0.525111	0.255456	-0.410479	-0.281660
25	6.6e-06	0.525108	0.255460	-0.410482	-0.281659
26	4.5e-06	0.525110	0.255458	-0.410479	-0.281660
27	2.7e-06	0.525109	0.255459	-0.410481	-0.281659
28	1.5e-06	0.525109	0.255459	-0.410480	-0.281659
29	7.2e-07	0.525109	0.255458	-0.410481	-0.281659
30	4.2e-07	0.525109	0.255459	-0.410480	-0.281659
31	3.3e-07	0.525109	0.255458	-0.410480	-0.281659
32	2.6e-07	0.525109	0.255459	-0.410480	-0.281659
33	1.8e-07	0.525109	0.255458	-0.410480	-0.281659
34	1.1e-07	0.525109	0.255459	-0.410480	-0.281659
35	5.9e-08	0.525109	0.255459	-0.410480	-0.281659

Table 10: SOR iterations with $w = 1.5$ and the 2-norm stopping rule for the given system.

19. Vandermonde interpolation method

Vandermonde pseudocode:

```
BEGIN
    READ x, y
    n = LENGTH x
    V = ZERO MATRIX n X n

    § Build Vandermonde matrix with descending powers
    FOR r = 1 TO n DO
        FOR c = 1 TO n DO
```

```
            V[r, c] = x[r]^( n - c )
        END FOR
    END FOR

    § Solve the linear system V a = y
    a = SOLVE_LINEAR_SYSTEM(V, y)

    PRINT Vandermonde matrix
    PRINT Polynomial coefficients
    PRINT Interpolating polynomial
    RETURN a

END
```

Data to use:

Table =

x	-1	0	3	4
y	15.5	3	8	1

Number of points: $n = 4$

Results:

Vandermonde matrix:

$$V = \begin{pmatrix} -1.000000 & 1.000000 & -1.000000 & 1.000000 \\ 0.000000 & 0.000000 & 0.000000 & 1.000000 \\ 27.000000 & 9.000000 & 3.000000 & 1.000000 \\ 64.000000 & 16.000000 & 4.000000 & 1.000000 \end{pmatrix}$$

Polynomial coefficients:

$$a = \begin{pmatrix} -1.141667 \\ 5.825000 \\ -5.533333 \\ 3.000000 \end{pmatrix}$$

Interpolating polynomial:

$$P(x) = -1.141667x^3 + 5.825000x^2 - 5.533333x + 3.000000$$

Hence, the interpolating polynomial fitting the given data is:

$$\boxed{P(x) = -1.141667x^3 + 5.825000x^2 - 5.533333x + 3.000000.}$$

20. Newton Interpolation Method

Newton interpolation pseudocode:

```
BEGIN

    READ x, y
    n = LENGTH x

    § Build the divided-difference table
    tabla = ZERO_MATRIX(n, n + 1)

    FOR i = 1 TO n DO
```

```
        tabla[i, 1] = x[i]
        tabla[i, 2] = y[i]
    END FOR

    § Compute divided differences
    FOR j = 3 TO n + 1 DO
        FOR i = j - 1 TO n DO
            tabla[i, j] =
                ( tabla[i, j - 1] - tabla[i - 1, j - 1] ) /
                ( tabla[i, 1] - tabla[i - j + 2, 1] )
        END FOR
    END FOR

    § Extract Newton coefficients
    coef = VECTOR(size = n)
    FOR i = 1 TO n DO
        coef[i] = tabla[i, i + 1]
    END FOR

    PRINT Divided difference table
    PRINT Newton polynomial
    RETURN pol_str, coef

END
```

Data to use:

Table =

x	-1	0	3	4
y	15.5	3	8	1

Results:

Divided differences table:

$$\begin{pmatrix} 15.500000 & & & \\ 3.000000 & -12.500000 & & \\ 8.000000 & 1.666667 & 2.357143 & \\ 1.000000 & -7.000000 & -2.333333 & -0.011905 \end{pmatrix}$$

Polynomial coefficients:

$$a = \begin{pmatrix} 15.500000 \\ -12.500000 \\ 2.357143 \\ -0.011905 \end{pmatrix}$$

Newton interpolating polynomial:

$$P(x) = 15.500000 - 12.500000(x+1) + 2.357143(x+1)(x) - 0.011905(x+1)(x)(x-3)$$

Thus, the interpolating polynomial obtained with Newton's divided differences is:

$$\boxed{P(x) = 15.5 - 12.5(x+1) + 2.3571(x+1)x - 0.0119(x+1)x(x-3).}$$

21. Lagrange Interpolation Method

Lagrange interpolation pseudocode:

```
BEGIN
    READ x, y
    n = length(x)
    L_table = matrix(n, n)

    FOR i = 0 TO n-1 DO

        Li = [1]
        denom = 1

        FOR j = 0 TO n-1 DO
            IF j  i THEN

                Li = multiply Li * (x - x[j])
                denom = denom * (x[i] - x[j])

            ENDIF
        END FOR

        Li_coeffs = Li / denom
        offset = n - Length(Li_coeffs)

        FOR k = 0 TO Length(Li_coeffs)-1 DO
            L_table[i][offset + k] = Li_coeffs[k]
        END FOR

    END FOR
    PRINT Interpolating polynomial
    RETURN L_table

END
```

Data to use:

Table =

x	-1	0	3	4
y	15.5	3	8	1

Results:

Lagrange basis polynomials:

$$\begin{aligned}
L_0(x) &= -0.055556x^3 + 0.066667x^2 + 0.277778x + 1.000000,\\
L_1(x) &= 0.083333x^3 - 0.416667x^2 + 0.333333x + 0.000000,\\
L_2(x) &= -0.027778x^3 + 0.083333x^2 - 0.388889x + 0.333333,\\
L_3(x) &= -0.000000x^3 + 0.000000x^2 - 0.222222x + 0.333333.
\end{aligned}$$

Interpolating polynomial:

$$P(x) = 15.5L_0(x) + 3L_1(x) + 8L_2(x) + 1L_3(x)$$

Simplified polynomial:

$$P(x) = -1.141667x^3 + 5.825000x^2 - 5.533333x + 3.000000$$

Thus, the Lagrange interpolating polynomial that fits the given data is:

$$\boxed{P(x) = -1.141667x^3 + 5.825000x^2 - 5.533333x + 3.000000.}$$

22. Spline Interpolation Method

Spline interpolation pseudocode:

```
BEGIN
    READ x, y, d
    n = length(x)

    A = matrix( (d+1)*(n-1), (d+1)*(n-1) )
    b = vector( (d+1)*(n-1) )

    cua = x^2
    cub = x^3

    c = 0
    h = 0


    IF d = 1 THEN

        FOR i = 0 TO n-2 DO
            A[h][c]     = x[i]
            A[h][c + 1] = 1
            b[h]        = y[i]
            c = c + 2
            h = h + 1
        END FOR

        c = 0

        FOR i = 1 TO n-1 DO
            A[h][c]     = x[i]
            A[h][c + 1] = 1
            b[h]        = y[i]
            c = c + 2
            h = h + 1
        END FOR


    ELSE IF d = 2 THEN

        FOR i = 0 TO n-2 DO
            A[h][c]     = cua[i]
            A[h][c + 1] = x[i]
            A[h][c + 2] = 1
            b[h]        = y[i]
            c = c + 3
            h = h + 1
        END FOR

        c = 0

        FOR i = 1 TO n-1 DO
            A[h][c]     = cua[i]
            A[h][c + 1] = x[i]
            A[h][c + 2] = 1
            b[h]        = y[i]
            c = c + 3
            h = h + 1
        END FOR

        c = 0

        FOR i = 1 TO n-2 DO
            A[h][c]     = 2*x[i]
            A[h][c + 1] = 1
            A[h][c + 3] = -2*x[i]
            A[h][c + 4] = -1
            b[h]        = 0
            c = c + 3
            h = h + 1
        END FOR

        A[h][0] = 2
        b[h]    = 0


    ELSE IF d = 3 THEN

        FOR i = 0 TO n-2 DO
            A[h][c]     = cub[i]
            A[h][c + 1] = cua[i]
            A[h][c + 2] = x[i]
            A[h][c + 3] = 1
            b[h]        = y[i]
            c = c + 4
            h = h + 1
        END FOR

        c = 0

        FOR i = 1 TO n-1 DO
            A[h][c]     = cub[i]
            A[h][c + 1] = cua[i]
            A[h][c + 2] = x[i]
            A[h][c + 3] = 1
            b[h]        = y[i]
            c = c + 4
            h = h + 1
        END FOR

        c = 0

        FOR i = 1 TO n-2 DO
            A[h][c]     = 3*cua[i]
            A[h][c + 1] = 2*x[i]
            A[h][c + 2] = 1
            A[h][c + 4] = -3*cua[i]
            A[h][c + 5] = -2*x[i]
```

```
            A[h][c + 6] = -1
            b[h]        = 0
            c = c + 4
            h = h + 1
        END FOR

        c = 0

        FOR i = 1 TO n-2 DO
            A[h][c]     = 6*x[i]
            A[h][c + 1] = 2
            A[h][c + 4] = -6*x[i]
            A[h][c + 5] = -2
            b[h]        = 0
            c = c + 4
            h = h + 1
        END FOR

        A[h][0] = 6*x[0]
        A[h][1] = 2
        b[h]    = 0
        h = h + 1

        A[h][c]     = 6*x[n-1]
        A[h][c + 1] = 2
        b[h]        = 0

    END IF
    val = Solve A, b
    RETURN table

END
```

Data to use:

Table =

x	-1	0	3	4
y	15.5	3	8	1

Results:

Linear splines ($d = 1$)

Spline coefficients:

$$\begin{pmatrix} -12.500000 & 3.000000 \\ 1.666667 & 3.000000 \\ -7.000000 & 29.000000 \end{pmatrix}$$

Linear spline equations:

$$\begin{aligned} S_1(x) &= -12.500000x + 3.000000, & x &\in [-1, 0] \\ S_2(x) &= 1.666667x + 3.000000, & x &\in [0, 3] \\ S_3(x) &= -7.000000x + 29.000000, & x &\in [3, 4] \end{aligned}$$

Quadratic splines ($d = 2$)

Spline coefficients:

$$\begin{pmatrix} 6.250000 & -12.500000 & 3.000000 \\ -0.277778 & 1.666667 & 3.000000 \\ -1.722222 & -7.000000 & 29.000000 \end{pmatrix}$$

Quadratic spline equations:

$$\begin{aligned} S_1(x) &= 6.250000x^2 - 12.500000x + 3.000000, && x \in [-1, 0] \\ S_2(x) &= -0.277778x^2 + 1.666667x + 3.000000, && x \in [0, 3] \\ S_3(x) &= -1.722222x^2 - 7.000000x + 29.000000, && x \in [3, 4] \end{aligned}$$

Cubic splines ($d = 3$)

Spline coefficients:

$$\begin{pmatrix} -3.613095 & 7.226190 & -9.071429 & 3.000000 \\ 0.654762 & -3.928571 & 7.476190 & 3.000000 \\ -0.041667 & -2.625000 & 4.500000 & 8.000000 \end{pmatrix}$$

Cubic spline equations:

$$\begin{aligned} S_1(x) &= -3.613095x^3 + 7.226190x^2 - 9.071429x + 3.000000, && x \in [-1, 0] \\ S_2(x) &= 0.654762x^3 - 3.928571x^2 + 7.476190x + 3.000000, && x \in [0, 3] \\ S_3(x) &= -0.041667x^3 - 2.625000x^2 + 4.500000x + 8.000000, && x \in [3, 4] \end{aligned}$$

Hence, the cubic spline provides the smoothest interpolation, ensuring continuity of the first and second derivatives across all subintervals.