Gauss seidel

## Reading Matrix and Vector Input from File

**Purpose:** This function reads a **square matrix A** of size n x n and a **vector b** of size n from a specified file. It ensures the file is properly formatted and handles errors during input parsing.

A screen shot of a computer program

AI-generated content may be incorrect.

**Step-by-Step Explanation**

**1. Open File**

* FILE \*file = fopen(filename, "r"); opens the file in read mode.
* If the file cannot be opened (e.g., it does not exist), the function prints an error message and exits with status 1.

**2. Read Matrix A**

* The nested loops iterate over each row (i) and column (j) of the matrix:
  + fscanf(file, "%lf", &A[i][j]) reads a floating-point number into A[i][j].
  + If the read operation fails (e.g., non-numeric data), the function prints an error, closes the file, and exits.

**3. Read Vector b**

* The loop iterates over each element of the vector:
  + fscanf(file, "%lf", &b[i]) reads a floating-point number into b[i].
  + If the read operation fails, the function prints an error, closes the file, and exits.

**4. Close File**

* fclose(file); closes the file after all data is read to release system resources.

## function: Matrix-Vector Multiplication

**Purpose:** This function multiplies a **square matrix** A of size n x n with a **vector** b of size n and returns the resulting vector. It computes the dot product of each row of A with b.

A computer screen shot of a program code

AI-generated content may be incorrect.

**Step-by-Step Explanation**

1. **Memory Allocation**:
   * Allocates memory for the result vector result of size n.
   * Exits with an error if memory allocation fails.
2. **Row Iteration**:
   * The outer loop iterates over each row i of the matrix A.
3. **Dot Product Calculation**:
   * Initializes result[i] to 0.
   * The inner loop iterates over each column j of the matrix.
   * Computes the dot product of the i-th row of A and the vector b.
4. **Return Result**:
   * Returns the computed vector result.

## function: Vector Subtraction

**Purpose**: This function computes the **element-wise subtraction** of two vectors **v1** and **v2** of size n and returns the result as a new vector

A screen shot of a computer code

AI-generated content may be incorrect.

**Step-by-Step Explanation**

1. **Memory Allocation**:
   * Allocates memory for the result vector result of size n.
   * Exits with an error if memory allocation fails.
2. **Element-wise Subtraction**:
   * Iterates over each index i from 0 to n-1.
   * Computes result[i] = v1[i] - v2[i] for each element.
3. **Return Result**:
   * Returns the vector result containing the difference of v1 and v2.

## function: Calculate the square of the Euclidean Norm of a Vector

**Purpose:** This function computes the square of the **Euclidean norm** (or magnitude) of a vector.

A computer screen shot of a program code

AI-generated content may be incorrect.

**Step-by-Step Explanation**

1. **Initialization**:
   * norm is initialized to 0.0 to accumulate the sum of squared elements.
2. **Element-wise Squaring and Summation**:
   * The loop iterates over each index i from 0 to n-1.
   * For each element vector[i], its square (vector[i] \* vector[i]) is added to norm.
3. **Return Result**:

The norm is returned:  
Norm=

## Function: **Gauss-Seidel**

**Purpose:** This function solves the linear system **Ax = b** using the **Successive Over-Relaxation (SOR)** method, an iterative technique that accelerates convergence by introducing a relaxation factor (weight). It is a generalization of the Gauss-Seidel method and is effective for **diagonally dominant** or **symmetric positive-definite** matrices.

A screen shot of a computer program

AI-generated content may be incorrect.

**Function Parameters**

* A: Coefficient matrix (n x n).
* b: Right-hand side vector (size n).
* n: Size of the system.
* weight: Relaxation factor (1.0 for standard Gauss-Seidel).
* epsilon: Convergence tolerance.

**Returns**: Solution vector x.

**Step-by-Step Explanation**

**1. Initialization**

* **Memory Allocation**:
  + x: Stores the updated solution in each iteration.
  + xg: Stores the solution from the previous iteration. Initialized to [1.0, 1.0, ..., 1.0].
* **Residual Calculation**:

c

residual = b - A \* xg;

tolerance = ||residual|| (Euclidean norm);

**2. Iterative Loop**

The loop continues until tolerance ≤ epsilon:

**while** (tolerance > epsilon) {

*// Update each element x[i]*

**for** (**int** i = 0; i < n; i++) {

sigma = Σ\_{j≠i} A[i][j] \* x[j];

x[i] = (1 - weight) \* xg[i] + weight \* (b[i] - sigma) / A[i][i];

}

*// Update xg with the new x*

xg = x;

*// Recompute residual and tolerance*

residual = b - A \* xg;

tolerance = ||residual||;

iteration++;

}

**Key Steps in Each Iteration**:

1. **Element-wise Update**:
   * For each element x[i]:
     + Compute sigma, the sum of products of off-diagonal elements of row i and **latest** x[j] values.
     + Update x[i] using the relaxation formula:

where ω (weight) controls convergence:

* + - * ω = 1: Standard Gauss-Seidel.
      * ω > 1: Over-relaxation (faster convergence).
      * ω < 1: Under-relaxation (stabilize convergence).

1. **Update Previous Solution**:
   * Copy x into xg to prepare for the next iteration.
2. **Residual Recalculation**:
   * Compute the new residual b - A \* xg and its norm to check convergence.

**3. Termination and Output**

* **Convergence**: The loop exits when the residual norm tolerance falls below epsilon.
* **Result**: Prints the number of iterations and returns the solution vector x.

## **Main Program for Gauss-Seidel with SOR Method**

**Purpose:** This program solves the linear system **Ax = b** using the **Gauss-Seidel method with Successive Over-Relaxation (SOR)**. It reads the matrix **A** and vector **b** from a file, computes the solution **x**, and prints the results along with performance metrics.

A screen shot of a computer screen

AI-generated content may be incorrect.

**Step-by-Step Explanation**

1. **User Input**:
   * Prompts the user for:
     + n: Number of variables (size of the matrix/vector).
     + epsilon: Convergence tolerance (e.g., 1e-6).
     + w: Relaxation factor (1.0 for standard Gauss-Seidel, >1.0 for over-relaxation).
2. **Memory Allocation**:
   * **A**: Allocates an n x n matrix (dynamic 2D array).
   * **b**: Allocates a vector of size n.
   * Includes error handling for memory allocation failures.
3. **Read Input File**:
   * Calls readInputsFromFile("inputs.txt", A, b, n) to load **A** and **b** from the file.
   * **File Format**: The file must contain n x n matrix elements followed by n vector elements.
4. **Solve with Gauss-Seidel SOR**:
   * Gauss\_seidel(A, b, n, w, epsilon) computes the solution vector **x**.
     + **Input**: Matrix **A**, vector **b**, size n, relaxation factor w, tolerance epsilon.
     + **Output**: Solution x and number of iterations to converge (printed internally).
5. **Performance Timing**:
   * Uses clock() to measure CPU time taken by the solver.
   * Converts time to nanoseconds for readability.
6. **Print Results**:
   * Displays the solution vector **x** with 6 decimal places.
   * Prints the execution time in nanoseconds.
7. **Memory Cleanup**:
   * Frees dynamically allocated memory for **A**, **b**, and **x** to prevent leaks.
8. Keynote
   * **Timing Function Behavior in Different Environments**
     + The program uses the clock() function to measure execution time in CPU cycles and converts it into nanoseconds using CLOCKS\_PER\_SEC.

* In environments like **Windows Subsystem for Linux (WSL)** running in VS Code, this function works reliably and provides accurate timing results.
* However, in a terminal-based VS Code setup or native Windows environments, the precision and behavior of clock() may vary due to differences in how CPU time is calculated across platforms.