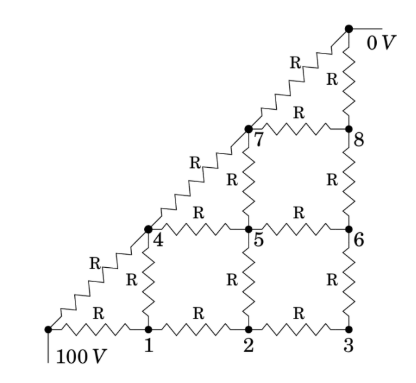
Computer Aided Numerical Methods – I

Report on Take Home Exam

Question 1:

Solve all the nodal voltages in the resistance network shown in figure below. Use Ohm’s law to write your set of linear equations.



Take R = 100 Ω. Dot the following:

1. Write the system of linear equations that makes this resistance network.
2. Solve the above equation using
   1. LU Decomposition
   2. Gauss Siedel
   3. Conjugate Gradient

methods to solve this linear equation. The resolution for the solution voltages should be accurate to the 3rd place after decimal.

1. Using your computer’s clock, please estimate the time taken to solve this problem for each of the three methods. Does SOR be used to accelerate the solution from Gauss Siedel? Please show through your computations.

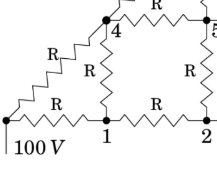
Question 1 – Solution:

Answer 1-1: Finding System of Equations

To find the system of linear equations, we will be using two laws:

1. Kirchhoff’s Current Law: The sum of all currents entering a node in an electrical circuit must equal the sum of all currents leaving that node.
2. Ohm’s Law: Ohm's law states that the electric circuit through a conductor between two points is directly proportional to the voltage across the two points. Formula relating voltage and current across a conductor is given by , where is the voltage, is the current and is the resistance of the conductor.

* Consider the node at point 1, :

By Kirchhoff’s Law, we can write:

Current going into = Current going out of

(is current from point ‘’ to ‘’)

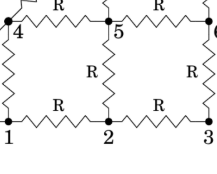
(here we consider point ‘ to be node)

By Ohm’s Law, we write .

( denotes voltage at node ‘’)

Doing the same for the other currents, our equation becomes

Simplifying the expression, we get our equation at as

* Next, we consider the node at point 2, :

Using Kirchhoff’s Law again, we write:

Again, using Ohm’s Law, the equation becomes

Simplifying the expression, we get our equation at as

* A diagram of a circuit

  AI-generated content may be incorrect.Next, we consider the node at point 3, :

Using Kirchhoff’s Law again, we write:

Again, using Ohm’s Law, the equation becomes

Simplifying the expression, we get our equation at as

A diagram of a voltage

AI-generated content may be incorrect.

* Next, we consider the node at point 4, :

Using Kirchhoff’s Law again, we write:

Again, using Ohm’s Law, the equation becomes

Simplifying the expression, we get our equation at as

* A diagram of a number of squares

  AI-generated content may be incorrect.Next, we consider the node at point 5, :

Using Kirchhoff’s Law again, we write:

Again, using Ohm’s Law, the equation becomes

Simplifying the expression, we get our equation at as

* A diagram of a circuit

  AI-generated content may be incorrect.Next, we consider the node at point 6, :

Using Kirchhoff’s Law again, we write:

Again, using Ohm’s Law, the equation becomes

Simplifying the expression, we get our equation at as

* A diagram of a diagram

  AI-generated content may be incorrect.Next, we consider the node at point 7, :

Using Kirchhoff’s Law again, we write:

(Here, we consider point ‘’ to be node)

Again, using Ohm’s Law, the equation becomes

Simplifying the expression, we get our equation at as

* Finally, we consider the node at point 8, :

A diagram of a staircase

AI-generated content may be incorrect.Using Kirchhoff’s Law again, we write:

Again, using Ohm’s Law, the equation becomes

Simplifying the expression, we get our equation at as

So, the system of equations that determine the Resistance Network are:

Answer 1-2: Solving the System

Rewriting the obtained system in terms of , we get:

In Matrix form, we can write the linear system as , where

and

Now, we can solve this linear system with the methods we have been tasked with using, i.e., a) LU Decomposition, b) Gauss Siedel and c) Conjugate Gradient methods.

**a) - LU Decomposition**

To explain the full algorithm of LU Decomposition and how it helps us solve the system , we first explain the algorithms for Forward and Backward Substitution to show how the factorization reduces the problem to solving two simpler triangular systems.

Algorithm and Explanation of Forward Substitution

We use this algorithm when we want to solve the system of the form , where is a lower triangular matrix (i.e., all entries above the main diagonal are zero), is a known vector and is the unknown vector we want to find. So, consider and , then we have

Multiplying left hand side, we get

Equating the components, we get that for any , .

When , , so we get .

When , , so we get , which we can get with the value of which we found earlier.

When , so we get , which we can get with the value of and which we found earlier.

So, for any general , we have

Each equation depends only on the current unknown ​ and the previously solved values .

So, it can be solved sequentially using loops like so:

Algorithm and Explanation of Backward Substitution

We use this algorithm when we want to solve the system of the form , where is an upper triangular matrix (i.e., all entries below the main diagonal are zero), is a known vector and is the unknown vector we want to find. So, consider and , then we have

Multiplying left hand side, we get

Equating the components, we get that for any , .

When , , so we get .

When , , so we get , which we can get with the value of which we found earlier.

When , so we get , which we can get with the value of and which we found earlier.

So, for any general , we have

Each equation depends only on the current unknown ​ and the previously solved values .

So, it can be solved sequentially using loops like so:

Now that we have explained the algorithms for Forward and Backward substitution, we can move on to explaining the algorithm for LU Decomposition and how it helps us obtain the solution to a system .

Algorithm and Explanation of LU Decomposition

We can decompose any matrix into matrices and , where is a lower triangular matrix and is an upper triangular matrix such that . Then, our equation becomes:

We first solve for in equation using Forward Substitution and then solve for in equation using Backwards Substitution, thus giving us our final solution vector .

So, let

So, we can write:

So, our system of equations is:

when

when

when

But the total number of unknowns we have is ( unknowns from and each). However, we only have equations (each element of A corresponds to an equation, hence elements give equations). As the number of unknowns is more than the number of equations, we set , for all . With this, we have equations and unknowns and so, we can solve the system. So, let us derive the general expression for and

When , when (as L is a lower triangular matrix) and when , and . Rewriting the summation in terms of , we get:

(as we set )

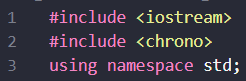
When , (as U is an upper triangular matrix) and . Rewriting the summation in terms of , we get:

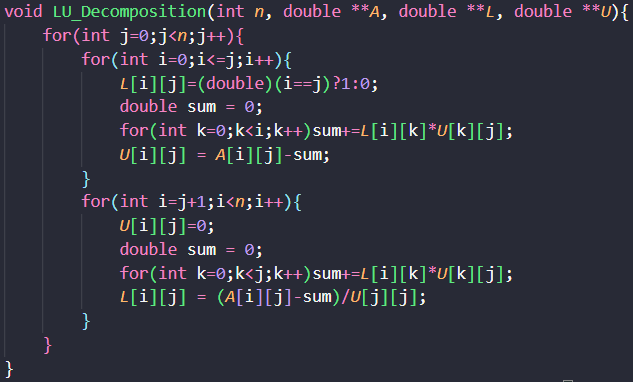
Each equation depends only on the current unknown or ​ and the previously solved elements of matrices and .

Hence, we can find matrices and like so:

**Explanation of Code for Finding Solution with LU Decomposition:**

**0 –** Preprocessing

****This program has been coded in C++ language. Hence, the main library ‘iostream’ has been included to allow use of the basic functions, control-flow statements, etc. of C++. ‘Using namespace std;’ prevents having to prefix input and output functions with “std::’. For the purpose of tracking the time taken to find the solution to the system, we include the library ‘chrono’ to make use of its time-based functions.

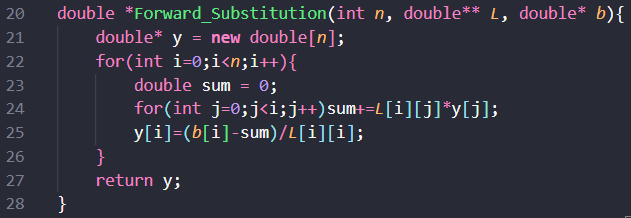
I – LU Decomposition Function

This function takes a pointer to the matrix ‘’, size of A ‘’ and two empty pointers ‘’ and ‘’ as input and in accordance with the LU Decomposition Algorithm described above, computes and fills the lower triangular matrix and upper triangular matrix such that .

Functionality:

1. The outer loop iterates through each column of matrix .
2. In the first inner loop, we iterate through each row such that:
   * The diagonal elements of are set to 1, whereas the upper diagonal elements of are set to 0.
   * For each row we set a variable to zero and increment it by in each iteration of another inner loop from to . Then, we set the value of by setting , which effectively sets , like we calculated in the algorithm explanation for LU Decomposition beforehand.
3. In the second inner loop, we iterate through each row such that:
   * The lower diagonal elements of are set to 0.
   * For each row we set a variable to zero and increment it by in each iteration of another inner loop from to . Then, we set the value of by setting , which effectively sets , like we calculated in the algorithm explanation for LU Decomposition beforehand.
4. At the end of the function, we get both matrices and .

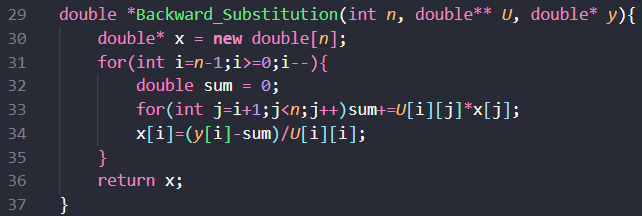
**II – Forward Substitution Function**

This function takes a pointer to Lower Triangular Matrix ‘’, size of ’, a pointer to vector ‘’ as input and using the Forward Substitution Algorithm explained above, returns a pointer to solution vector ‘’ such that .

Functionality:

1. For each in outer loop from to , we set a variable to zero and increment it by in each iteration of inner loop from to .
2. Then, we compute the value of by calculating , which effectively sets , like we calculated in the algorithm explanation for Forward Substitution beforehand.
3. The obtained solution vector satisfies .

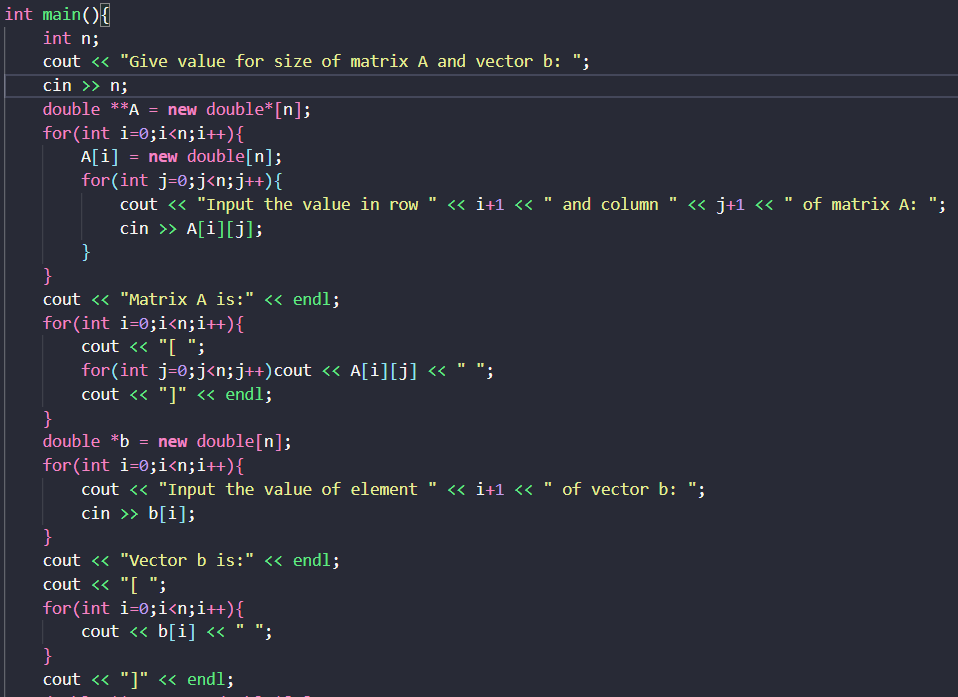
**III – Backward Substitution Function**

This function takes a pointer to Upper Triangular Matrix ‘’, size of ‘’, a pointer to vector ‘’ as input and using the Backwards Substitution Algorithm explained above, returns a pointer to solution vector ‘’ such that .

Functionality:

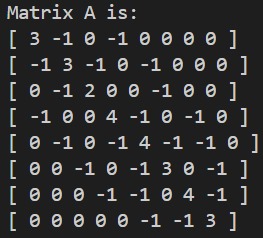
1. For each in outer loop from to , we set a variable to zero and increment it by in each iteration of inner loop from to .
2. Then, we compute the value of by calculating , which effectively sets , like we calculated in the algorithm explanation for Backward Substitution beforehand.
3. The obtained solution vector satisfies .

**IV – Main Function**

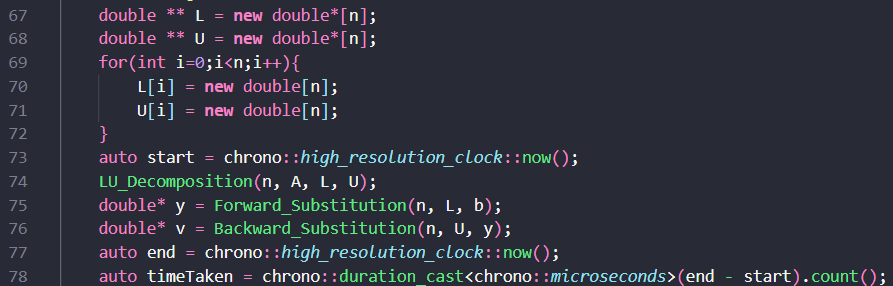
This function handles user input, LU decomposition, and solving the system.

Functionality:

1. User Input: The program reads matrix size , dynamically allocates memory for matrix and vector , accepts user input for elements of matrix and vector and displays the input matrices like so:



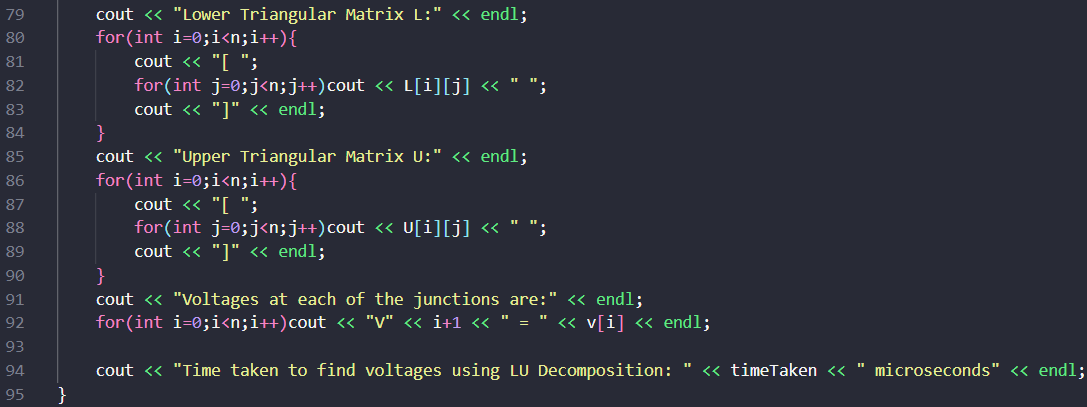




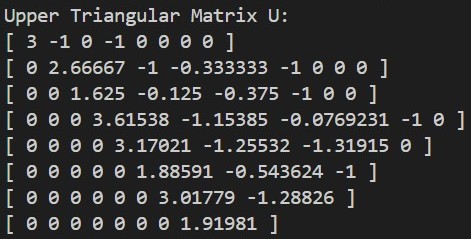
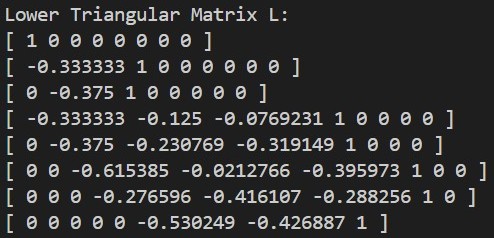
1. LU Decomposition: Here, the code dynamically allocates memory for two matrices and , calls the LU\_Decomposition function to decompose matrix into the lower triangular matrix as and upper triangular matrix as to use in the Substitution functions.
2. Solving the Linear System: The code first calls the Forward\_Substitution function to compute solution vector of the system .

Then, it calls the Backward\_Substitution function to compute solution vector to the system . Hence, is the solution to . Our solution vector consists of the nodal voltages of the resistance network, where .

1. Tracking the duration: The time taken for the process of finding the solution, i.e. time taken for LU Decomposition + Forward Substitution + Backward Substitution is tracked using the chrono functions which shall be explained later.



1. Printing Outputs: At last, the code prints the matrices and , the nodal voltages and the time taken for the whole process.

Printed Outputs and Solution to Linear System using LU Decomposition including time taken for the process:

A blue screen with white text

AI-generated content may be incorrect.

Hence, the nodal voltages are 74.324V at node , 58.108V at node , 50V at node , 64.865V at node , 50V at node , 41.892V at node , 35.135V at node and 26.676V at node .

The time taken for this process in this instance is 4 microseconds.

**b) – Gauss-Siedel Method**

**Explanation and Algorithm for Gauss-Siedel Method**

Let us consider the system of equations in matrix form , where

and

We can then write

We set

.

Then we have . Substituting in our equation of the system, we get

is given by .

We can find x iteratively by setting , where is the value of in the iteration. Expanding on both sides and equating components, we get

This is the Gauss-Jacobi method for finding the solution to the system .

Now, for is already known. Hence, we can modify our current formula to obtain the expression

This is the Gauss-Siedel method for finding the solution to the system .

If the rate of convergence is too fast, it can lead to oscillations around the solution. If the rate is too slow, it increases the number of iterations needed—both resulting in more computation. To address this, weights are introduced to control the rate of convergence, helping to strike a balance between speed and stability, like so:

,

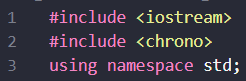
This generalized form of Gauss-Siedel is called Successive Over-Relaxation.

Residual: Residual at the iteration is defined as It determines how close the current approximation is to solving the linear system . When , is the solution.

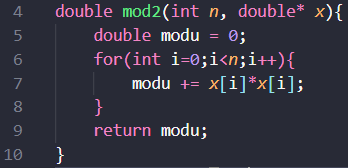
Here, we stop iterating when (the square of the norm of the residual) is smaller than a user-defined tolerance, say , indicating that is a satisfactory approximation. The full algorithm is detailed below:

**Explanation of Code for Finding Solution with Gauss-Siedel:**

**0 –** Preprocessing

****This program has been coded in C++ language. Hence, the main library ‘iostream’ has been included to allow use of the basic functions, control-flow statements, etc. of C++. ‘Using namespace std;’ prevents having to prefix input and output functions with “std::’. For the purpose of tracking the time taken to find the solution to the system, we include the library ‘chrono’ to make use of its time-based functions.

**I –** Modulus2 Helper Function

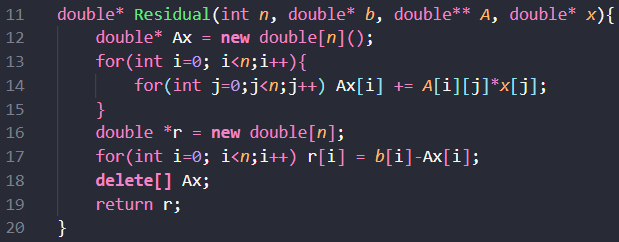
This function takes an integer ‘’ and pointer to a vector ‘’ of length as inputs and returns the square of the vector’s Euclidean norm, denoted by , i.e. the function computes:

.

The function accumulates the sum of the squares of each element in the vector by setting a variable to zero and updating the value of in each iteration of the main loop from to like so:

Finally, we get the value of as

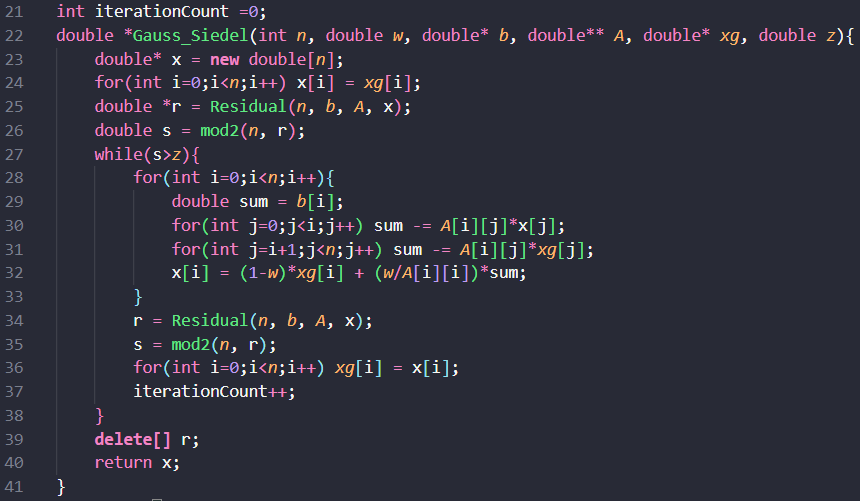
**II –** Residual Helper Function

This function takes integer ‘’, pointer to vectors of size , ‘’ and ‘’, and a pointer to matrix ‘’ of size as inputs, computes and returns the residual vector as output.

This residual vector ‘’ is used to check for convergence.

Functionality:

1. The function first allocates memory for the matrix-vector product and initializes its elements to 0.
2. Then, it computes each element by incrementing it by in each iteration of the inner loop from to , giving the expression .
3. Then memory is allocated for the residual vector r and each element is computed in the each iteration of another loop from to by subtracting the corresponding value in from : . The resulting value pointer to vector is returned.

**III –** Gauss-Siedel Iterative Function

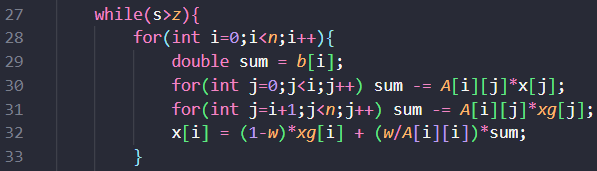
Before the function definition, a global variable is defined and initialized to 0. This variable will be used to count the number of iterations taken to reach convergence. Now, on to the function definition:

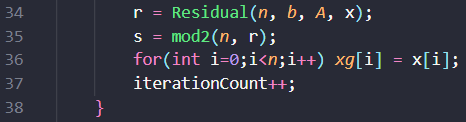
This function takes size ‘’, weight ‘’, pointers to vector ‘’, initial guess vector ‘’ and matrix ‘’, and tolerance ‘’ as input and using the Gauss-Siedel iterative method with Successive Over-Relaxation (SOR), the function computes and returns the solution vector ‘’ to the linear system .

Functionality:

1. The function begins by allocating memory for the solution vector and initializing it with values from the initial guess vector . This ensures that updates made to during the current iteration do not overwrite before it is fully used.

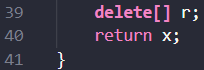


1. The residual vector is computed with the helper function, and its squared Euclidean norm is calculated using the helper function and stored in variable to be used as a convergence check.
2. The iterative loop runs as long as the residual norm is greater than the given convergence threshold . Within each iteration of the main loop from to , we calculate the value of each element vector . For each element of the solution vector:
   * The value is computed and stored into a variable. The summation consists of two parts:
     + For , updated values from the current iteration are used.
     + For , old values from the previous iteration are used.
   * The new value of is calculated with the SOR update formula:
   * This blending of old and new values allows the user to control the acceleration of convergence by allowing them to set value of .



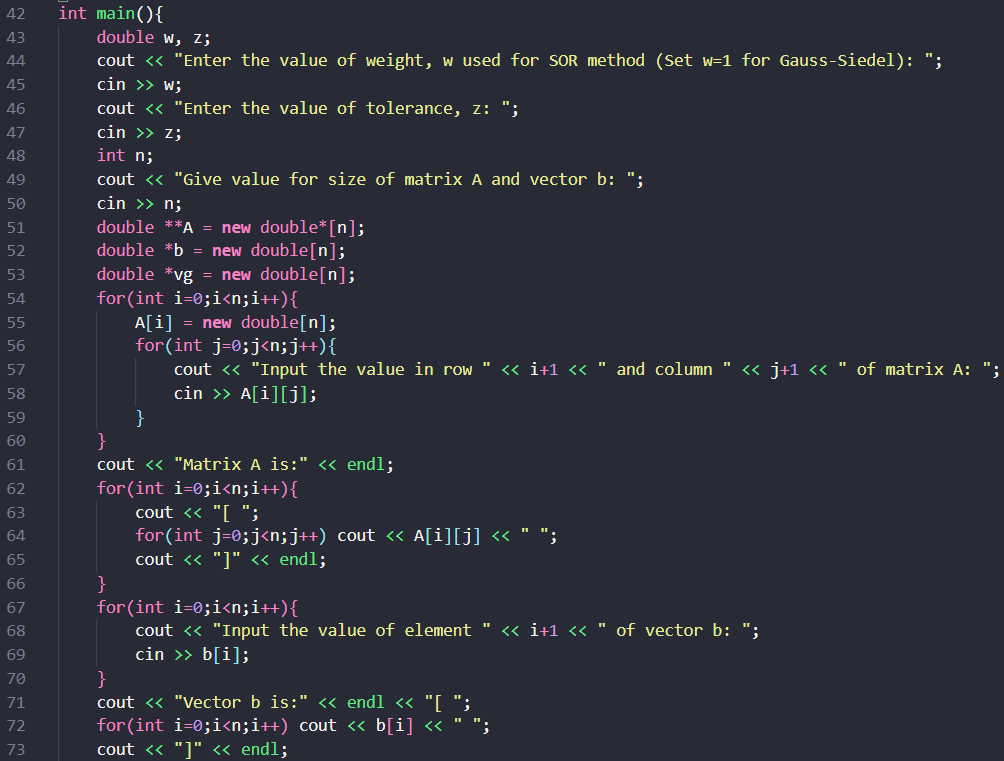
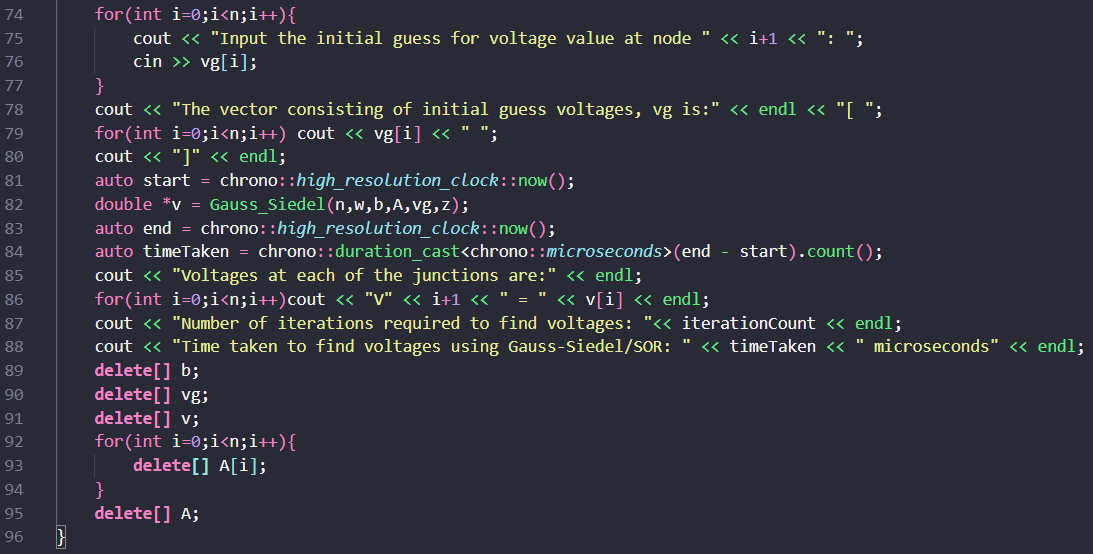
1. After each iteration:

* The residual and its norm squared are recomputed using the updated vector .
* The contents of are copied into for use in the next iteration.
* The global iteration counter is incremented by 1.



1. Once convergence is achieved (i.e., ), memory for the final residual vector is deallocated, and the function returns the computed solution vector .

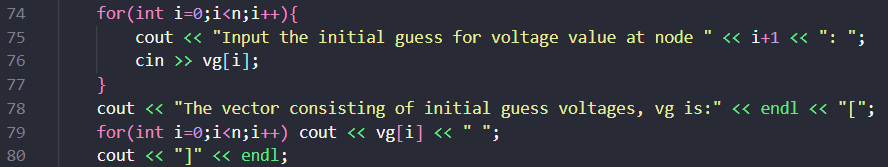
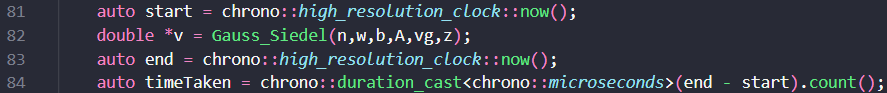
**IV – Main Function**

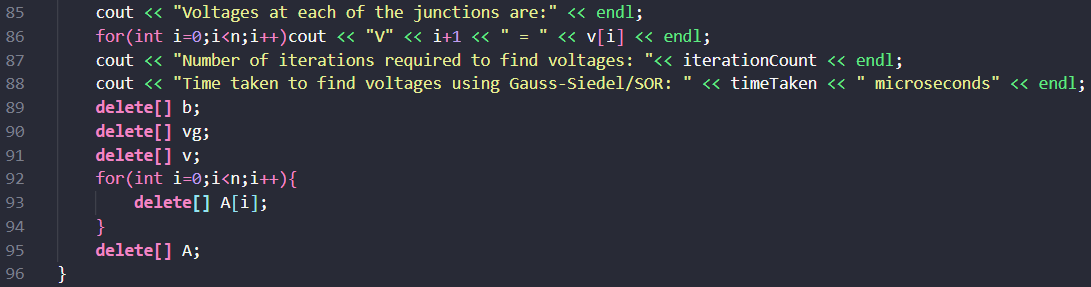


The main function is responsible for taking inputs, allocating memory, managing execution, and displaying output. It allows the user to enter the required components of the linear system , including matrix , vector , and initial guess vector , and the required components of Gauss-Siedel method. It then calls the Gauss-Seidel/Successive Over-Relaxation method to solve the system and compute the required nodal voltages. Finally, it displays the results and cleans up the allocated memory.

A computer screen shot of a program code

AI-generated content may be incorrect.Functionality:

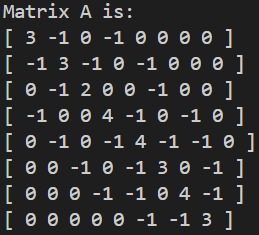
1. Taking User Input, Memory Allocation and Input Printing:
   * Firstly, the function prompts the user to input the values of weight factor ‘’ (used in SOR), tolerance ‘’ and then the size of matrix and vectors and ‘’.
   * It then dynamically allocates memory for matrices , which together form the linear system , and then the vector containing initial guess values .
   * Finally, the main function takes input for the element of matrix ‘’, vector ‘’ and vector ‘’ and then prints these inputs back to the user.
2. Solving the System and measuring the Time Taken:
   * The program first records the starting time (time before function call) and stores it in the variable.
   * Then it calls the function to iteratively determine the solution vector ‘’ of the system with initial guess vector ‘’, weight factor ‘’ and tolerance ‘’.
   * The code then records the ending time (time after function call) and stores it in the variable. With times and , we calculate the duration of the method for the given inputs. Here, we calculate the duration in microseconds and store the value in variable ‘’.



1. Printing the Outputs and Freeing up Allocated Memory:
   * Now, the program prints the nodal voltages we get from the solution vector (Nodal voltage at node , ]), the time taken for the process of solving the system, ‘’ and the number of iterations of till convergence, ‘’.
   * Finally, the code frees up the excess memory taken up by vectors ‘’, ‘’, ‘’ and matrix ‘’ to prevent memory leaks and ensure proper code termination.

Printed Inputs/Outputs and Solution to the Given System:

As we are required to find the solution using Gauss Siedel iterative method, we set weight factor ( is Over-Relaxation and is Under-Relaxation). As for tolerance, we set to the value of 0.000000001 or 1e-10 in scientific notation.





As deduced in the equation formation, these two are the Matrix A and vector b that together form the linear system to solve for the nodal voltages of the given resistance network.

A black screen with white text

AI-generated content may be incorrect.

Vector vg is the initial guess vector inputted into the program, and voltages at the junctions/nodes have been determined up to 3 point precision using Gauss Siedel to be:

The time taken for the process is this instance is 37 microseconds.

The number of iterations the program took to converge is 57.

**c) – Conjugate Gradient Method**

This method is used to solve a system when the matrix is a) sparse, b) symmetric and c) diagonally dominant.

Algorithm and Explanation of Conjugate Gradient Method

Consider the quadratic function of vector ,

Now,

As A is symmetric, .

So,

When we set

1. The system is satisfied.
2. is at an optimum, i.e., is optimized and hence optimization algorithms can be used.

Now, residual at the iteration is given by:

.

(where is the value of after iteration of Conjugate Gradient method)

In Conjugate Gradient method, we define search direction vector such that it is A-conjugate to the search directions in all other iterations, i.e. . Initially, we set

We then define the vector in the iteration to be:

, where is the step size of the iteration

We then choose such that

Now

Substituting value of in the previous expression, we get

The new search direction vector in the iteration is defined as:

, where is the direction coefficient of the iteration.

Now, we have defined search direction vector in such a way that it is A-conjugate to all other search directions, so we have .

Substituting value of , we get

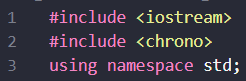
So, with the calculated value of , we can compute:

Using value of , we can calculate and subsequently compute:

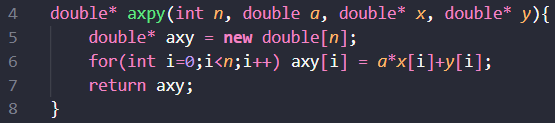
Similar to Gauss-Siedel, we stop iterating (say at the iteration) when the Euclidean norm, is less than a user defined tolerance, say z. Then, vector is our required solution vector. The full algorithm is detailed below:

**Explanation of Code for Finding Solution with Gauss-Siedel:**

**0 –** Preprocessing

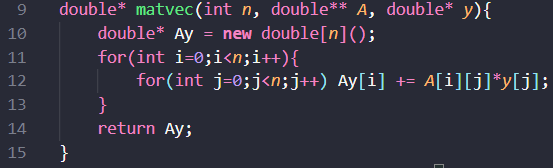
****This program has been coded in C++ language. Hence, the main library ‘iostream’ has been included to allow use of the basic functions, control-flow statements, etc. of C++. ‘Using namespace std;’ prevents having to prefix input and output functions with “std::’. For the purpose of tracking the time taken to find the solution to the system, we include the library ‘chrono’ to make use of its time-based functions.

**I –** axpy Helper Function

****This function takes size of vectors ‘’, two pointers to vectors ‘’ and ‘’, and a scalar ‘’ as inputs and returns a pointer to vector denoted by pointer variable in the code by computing each element in the each iteration of the main loop from to using the formula:

The resulting pointer is returned.

**II –** matvec Helper Function

This function takes size of matrix and vector ‘’, pointers to matrix ‘’ and vector ‘’ as inputs and returns a pointer to product of matrix and vector .

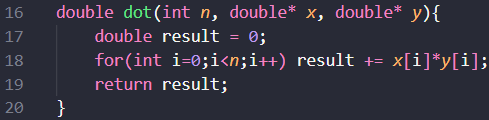
Functionality:

1. First, the code allocates memory for the pointer Ay and initializes all its elements to 0. This pointer will point to our Matrix-Vector profuct of A and y.
2. In each iteration of the the main loop from to , element is computed by incrementing its value by in each iteration of the inner loop from to like so:

.

1. The final value of is computed as . At the end of the outer loop, final pointer to vector is returned.

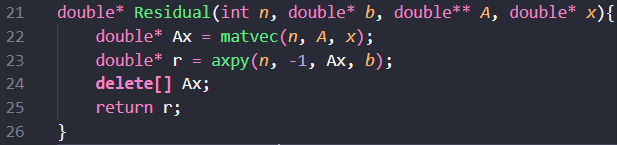
**III –** dot Helper Function

This function takes size of vectors ‘’, and two vectors ‘’ and ‘’ as input, calculates and returns their scalar dot product , as output. The dot product of two vectors and is computed here as .

The function accumulates the sum of the product of each element of the vectors by setting a variable to zero and updating the value of in each iteration of the main loop from to like so:

Finally, we get the value of as

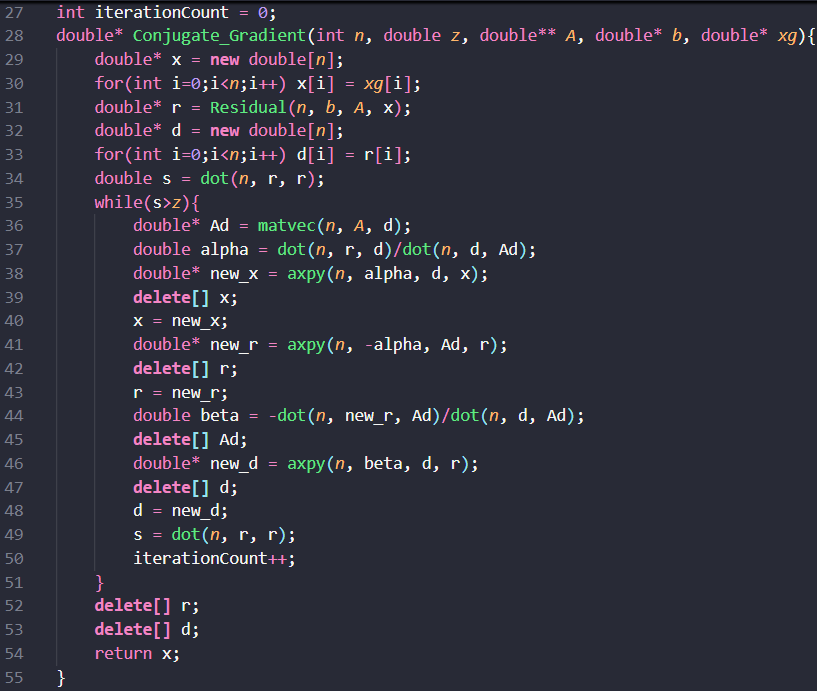
**IV –** Residual Helper Function

This function takes integer ‘’, pointer to vectors of size , ‘’ and ‘’, and a pointer to matrix ‘’ of size as inputs, computes and returns the residual vector as output.

Functionality:

1. The function first initializes the pointer variable Ax to store the product of matrix and vector , and computes this product by calling the function.
2. Next, it initializes the residual vector r and computes its value by calling the function with scalar , vector , and vector . This effectively calculates .

**V –** Conjugate Gradient Function

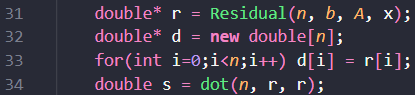
Before the function definition, a global variable is defined and initialized to 0. This variable will be used to count the number of iterations taken to reach convergence. Now, on to the function definition:

This function takes size ‘’, pointers to vector ‘’, initial guess vector ‘’ and matrix ‘’, and tolerance ‘’ as input and using the Conjugate Gradient iterative method, the function computes and returns the solution vector ‘’ to the linear system .

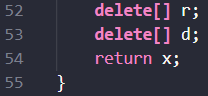
Functionality:



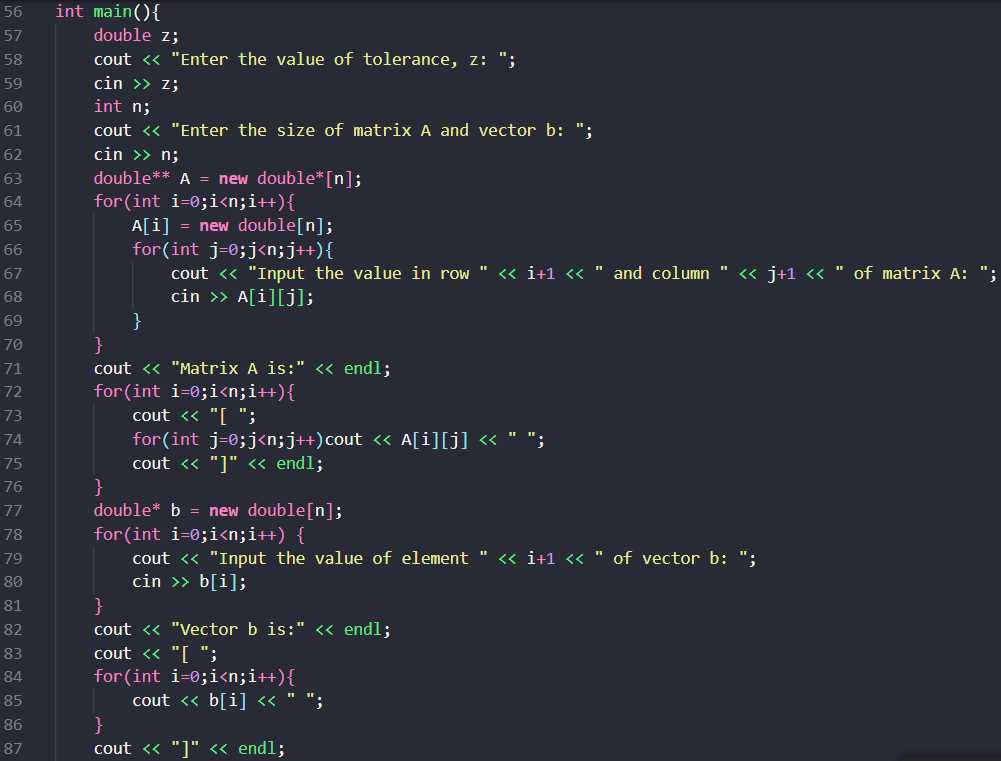
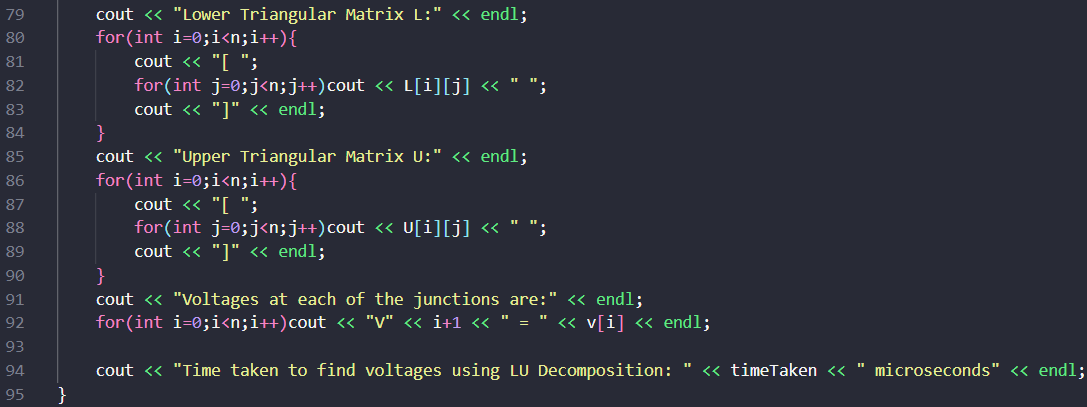
1. The function begins by allocating memory for the solution vector and initializing it with values from the initial guess vector . This ensures that updates made to during the current iteration do not overwrite .



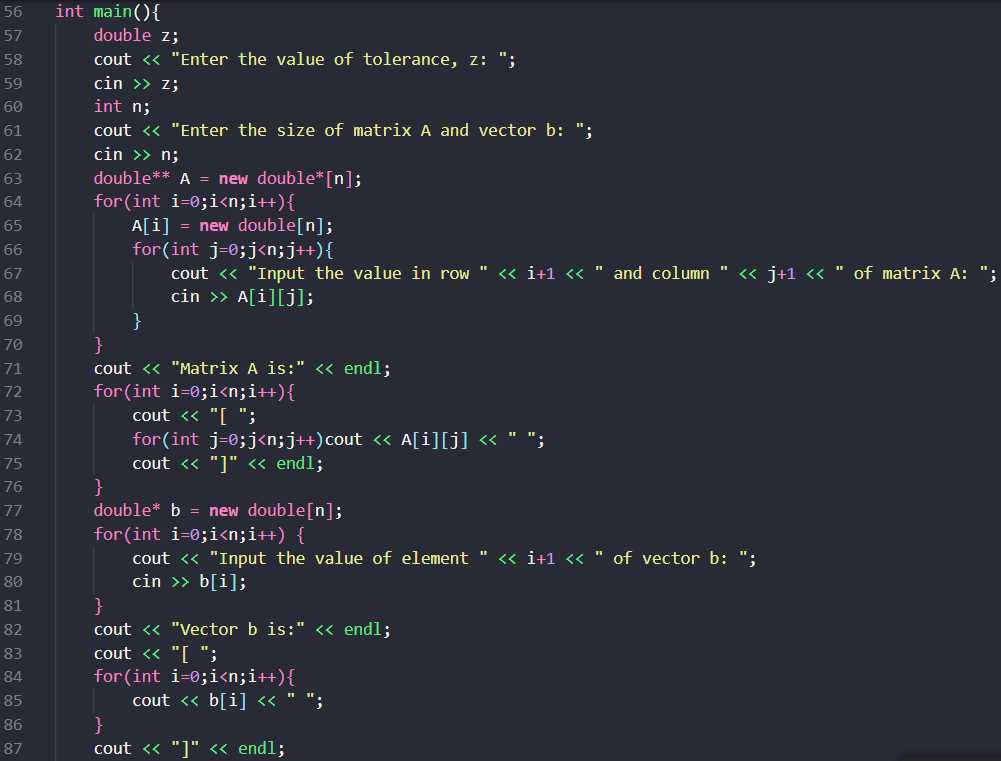
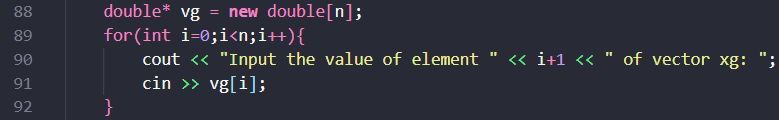
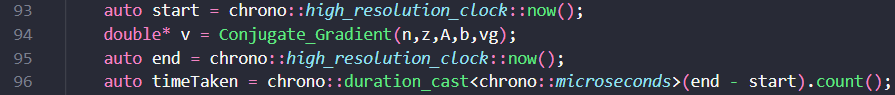
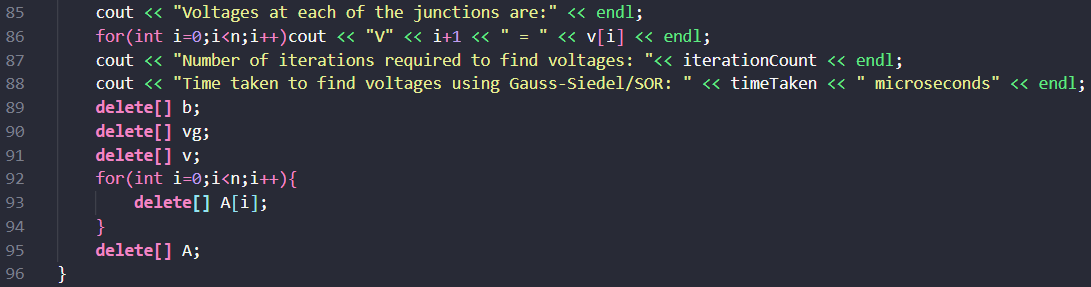
1. *A screen shot of a computer program

   AI-generated content may be incorrect.*The residual vector is computed with the helper function, and initial search direction vector is set to value of vector r. And the squared Euclidean norm of Residual r is calculated using the helper function and stored in variable to be used as a convergence check.
2. The iterative loop runs as long as the square of residual norm is greater than the given convergence threshold . Within each iteration, we calculate the updated values of vector , residual vector and search direction vector to be used in the next iteration.
   * As we are using the product of matrix and vector many times in this loop, we compute this product using Helper Function and store it in the variable.
   * Next, we find the step-size using the derived formula and store it in variable . We compute and using the dot helper function on vectors and , and dot helper function on and previously computed vector respectively.
   * We calculate the new x using the formula and store it in dummy variable and then reassign to variable . We compute by using the helper function, where , and .
   * We calculate the new r using the formula and store it in dummy variable and then reassign to variable . We compute by using the helper function, where , and .
   * We find direction coefficient using the derived formula and store it in variable . Again, we calculate by applying the helper function on vectors and and similarly, calculate by applying the same function on vectors and .
   * We calculate the new d using the formula and store it in dummy variable and then reassign to variable . We compute by using the helper function, where , and .
   * Finally, we calculate square of the norm of the updated residual and store it in to be compared with tolerance again and increment the .
3. Once convergence is achieved (i.e., ), memory for the final residual vector and search direction vector is deallocated, and the function returns the computed solution vector .

**IV – Main Function**

**** The main function is responsible for taking inputs, allocating memory, managing execution, and displaying output. It prompts the user to enter the required components of the linear system , including matrix , vector , and initial guess vector , and the required components of Conjugate Gradient method. It then calls the Conjugate Gradient function to solve the system and compute the required nodal voltages. Finally, it displays the results and cleans up the allocated memory.

Functionality:

1. ****Taking User Input, Memory Allocation and Input Printing:
   * Firstly, the function prompts the user to input the values of tolerance ‘’ and then the size of matrix and vectors and ‘’.
   * It then dynamically allocates memory for matrices , which together form the linear system , and then the vector containing initial guess values .
   * Finally, the main function takes input for the element of matrix ‘’, vector ‘’ and vector ‘’ and then prints these inputs back to the user.
2. Solving the System and measuring the Time Taken:
   * The program first records the starting time (time before function call) and stores it in the variable.
   * Then it calls the function to iteratively determine the solution vector ‘’ of the system with initial guess vector ‘’, and tolerance ‘’.
   * The code then records the ending time (time after function call) and stores it in the variable. With times and , we calculate the duration of the method for the given inputs. Here, we calculate the duration in microseconds () and store the value in variable ‘’.
3. Printing the Outputs and Freeing up Allocated Memory:
   * Now, the program prints the nodal voltages we get from the solution vector (Nodal voltage at node , ]), the time taken for the process of solving the system, ‘’ and the number of iterations of till convergence, ‘’.
   * Finally, the code frees up the excess memory taken up by vectors ‘’, ‘’, ‘’ and matrix ‘’ to prevent memory leaks and ensure proper code termination.

Printed Inputs/Outputs and Solution to the Given System:

For tolerance, we set to the value of 0.000000001 or 1e-10 in scientific notation.

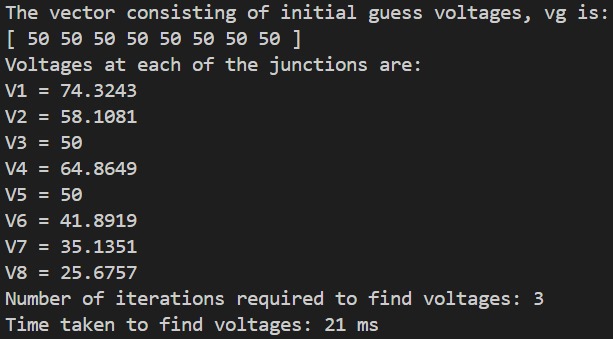
A screen shot of a computer code

AI-generated content may be incorrect.

A close up of a number

AI-generated content may be incorrect.

As deduced in the equation formation, these two are the Matrix A and vector b that together form the linear system to solve for the nodal voltages of the given resistance network.

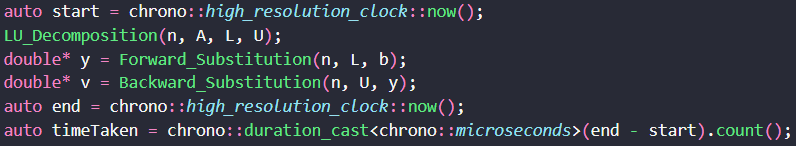
Vector is the initial guess vector inputted into the program, and voltages at the junctions/nodes have been determined up to 3 point precision using Conjugate Gradient to be:

The time taken for the process is this instance is 21 microseconds.

The number of iterations the program took to reach convergence is 3.

Answer 1-3: Calculating The Time Taken and Using SOR

To estimate the time taken using our computer’s clock, we make use of the chrono library in C++, which enables us to capture high-resolution timestamps before and after a function’s execution, and compute the duration by taking the difference between these timestamps.

For example, let us consider the code snippet of LU Decomposition responsible for calculating the time duration of the process:

Here the statement “” captures the current point of time using the highest possible resolution supported by the computer system’s clock. Using this, we can determine the starting and ending times of a process by finding the current time before and after the process respectively and the duration of this process is just the difference between these two times, as we have done here. For the purposes of this report, we have calculated the duration of this process in microseconds.

A blue screen with white text

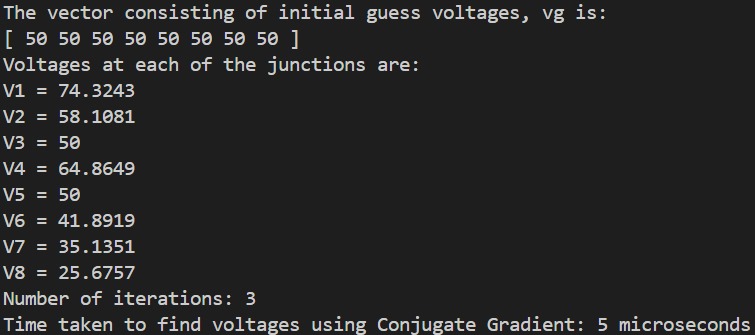
AI-generated content may be incorrect.Time Duration for Each Process:

For LU Decomposition method of finding solution to the given system, the time duration of the process in this instance is 2 microseconds.

A black screen with white text

AI-generated content may be incorrect.

For Gauss-Siedel method of finding solution to the given system, the time duration of the process in this instance is 37 microseconds.



For Conjugate Gradient method of finding solution to the given system, the time duration of the process in this instance is 5 microseconds.

Now, Successive Over-Relaxation (SOR) is when we modify the original Gauss-Siedel formula to control the acceleration of convergence as per our requirements by introducing weight factors like so:

A black screen with white text

AI-generated content may be incorrect.So we now vary the weight factor to check whether SOR accelerates the solution from Gauss-Siedel.

When we set the weight factor, (original Gauss-Siedel process), convergence is attained in 56 iterations.

The overall process takes 51 microseconds.

A black screen with white text

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.If we reduce the weight factor to say, , the number of iterations taken for the process is nearly triple compared to Gauss-Siedel, requiring 159 iterations. The time taken for this process is more than double that of the Gauss-Siedel, amounting to 125 microseconds in this instance.

If we increase the weight factor to say, , convergence is attained in 24 iterations, which is much lesser when compared to Gauss-Siedel and also takes 24 microseconds to converge, also much lesser than Gauss-Siedel’s time taken.

A black screen with white text

AI-generated content may be incorrect.But when we increase weight factor further, to say , the whole process takes 101 iterations for convergence, nearly double that of Gauss-Siedel and the time taken is 68 microseconds, which is more that the initial Gauss-Siedel process.

From these computations, we can say that SOR can be used to accelerate the solution from Gauss-Siedel, mainly when weight factor approaches 1.5.

Footnote 1: For recording durations of the different processes, I have used the Windows Subshell for Linux (WSL) framework as it records time more consistently and accurately than the Windows PowerShell/Command Prompt.

Footnote 2: The execution times recorded using the C++ chrono library may exhibit slight variations across multiple runs. These differences are primarily due to background system processes and resource management, and they do not significantly impact the overall performance analysis.

Footnote 3: Due to the inconvenient manner in which I took inputs for my code, I directly just pasted the following inputs on program runtime (works for output checking Windows PowerShell/Command Prompt mainly) to avoid wasting time entering them.

1. LU Decomposition: 8 3 -1 0 -1 0 0 0 0 -1 3 -1 0 -1 0 0 0 0 -1 2 0 0 -1 0 0 -1 0 0 4 -1 0 -1 0 0 -1 0 -1 4 -1 -1 0 0 0 -1 0 -1 3 0 -1 0 0 0 -1 -1 0 4 -1 0 0 0 0 0 -1 -1 3 100 0 0 100 0 0 0 0
2. Gauss-Siedel: 1 1e-10 8 3 -1 0 -1 0 0 0 0 -1 3 -1 0 -1 0 0 0 0 -1 2 0 0 -1 0 0 -1 0 0 4 -1 0 -1 0 0 -1 0 -1 4 -1 -1 0 0 0 -1 0 -1 3 0 -1 0 0 0 -1 -1 0 4 -1 0 0 0 0 0 -1 -1 3 100 0 0 100 0 0 0 0 50 50 50 50 50 50 50 50 (to change weight, replace the 1st input with the desired weight value)
3. Conjugate Gradient: 1e-10 8 3 -1 0 -1 0 0 0 0 -1 3 -1 0 -1 0 0 0 0 -1 2 0 0 -1 0 0 -1 0 0 4 -1 0 -1 0 0 -1 0 -1 4 -1 -1 0 0 0 -1 0 -1 3 0 -1 0 0 0 -1 -1 0 4 -1 0 0 0 0 0 -1 -1 3 100 0 0 100 0 0 0 0 50 50 50 50 50 50 50 50