Computer Aided Numerical Methods – I

Report on Take Home Exam

Question 2:

Please see below the set of linear equations:

Find the following:

1. The condition number of this matrix
2. Use Grahm Schmidt method to estimate the Q matrix.
3. Use Householder transformation to estimate the Q matrix and then find the solution using QR method.

Question 2 – Solution:

Answer 2-1: Finding Condition Number of a Matrix

The condition number of a matrix tells us how much the solution vector ‘x’ to a system changes due to perturbations in ‘’. For an matrix ‘’, the condition number given by , where is the maximum among the sums of the absolute values of the elements in each row of , i.e. .

To find condition number , we first find the inverse of matrix , and then compute the product of norm of , and norm of , . So, let us explain the algorithm to find the inverse of matrix A.

Algorithm and Explanation to find Inverse

The inverse of matrix A is defined to be ,

(where is determinant of and is the adjoint of )

So, to find , we must first find the determinant of matrix .

Finding the Determinant of a Matrix:

Let matrix , then

Now, let us define what the minor matrix is as we will need it for calculating the determinant of A.

The minor matrix of element of matrix , , is defined as the submatrix we get when removing the row and column of , i.e.

With this, we can calculate determinant of by expanding along row :

With this formula, we can implement a recursive approach to find the determinant of any given matrix A. For the purposes of this exam, we have expanded along the top row of the matrix, i.e. along row .

Finding the Adjoint of a Matrix:

For a matrix , the adjoint is given by

where .

So, using our algorithm for determinant, we can calculate the elements of the adjoint of matrix , .

With and , we can calculate the inverse of matrix , using the previously given formula. With the matrix , we can now calculate the condition number .

We can then get the condition number by computing the norms of and and returning their product.

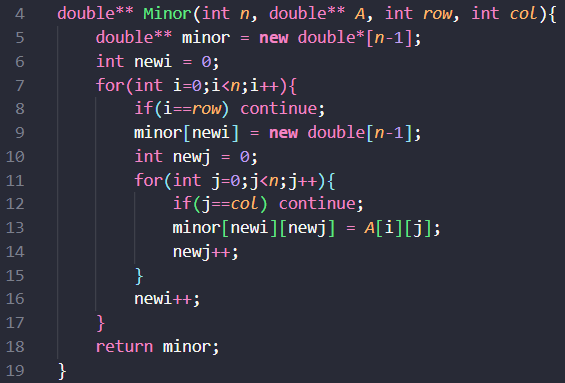
**Explanation of Code for Finding Condition Number of a Matrix:**

**0 –** Preprocessing

A group of colorful text

AI-generated content may be incorrect.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++. Here the library ‘cmath’ has also been included to allow the use of in-built math functions in C++. ‘Using namespace std;’ prevents having to prefix input and output functions with “std::’.

**I –** Minor Function

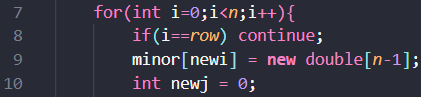


This function takes an integer ‘’, pointer to matrix ‘’ of size , index of column to be removed ‘’ and index of row to be removed ‘’ as inputs and returns a pointer to the submatrix formed by deleting the specified row and column from .

Functionality:



1. We first allocate memory for rows of submatrix minor and initialize variable . Variable tracks the current row index in the minor matrix.



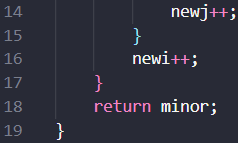
1. In the outer loop, iterate through each row of the original matrix, i.e., from and if the current row is the row to be removed, we skip it. We then allocate memory for the current row in minor matrix and initialize variable . Variable tracks the current column index in the submatrix.



1. Next in the inner loop, we iterate through each column of the original matrix and if the current column is the column to be removed, we skip it.

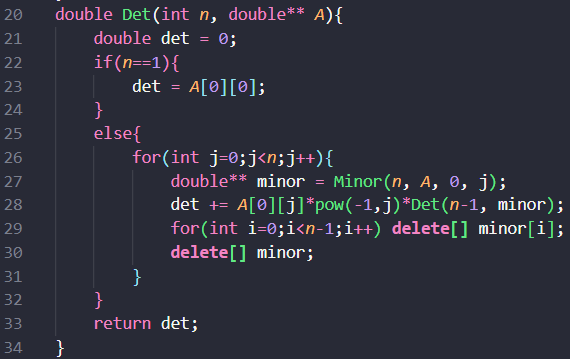


1. The element at position in original matrix, i.e. is copied over to the submatrix at position



1. We then increment in the inner loop and in the outer loop. After the termination of the outer loop, the resulting submatrix is returned.

**II –** Determinant Function



This function takes an integer ‘’ and a pointer to an matrix ‘’ as inputs and returns the computed determinant of , as output.

Functionality:

1. We first initialize a variable .
2. When the size of matrix ‘’ is 1, determinant value is just the only element in matrix A, i.e., and hence, we set to equal and return that value.
3. Otherwise, in the main loop, we calculate by first calling the Minor function for row and column and assigning the returned Minor matrix, to a variable and then update the variable like so:

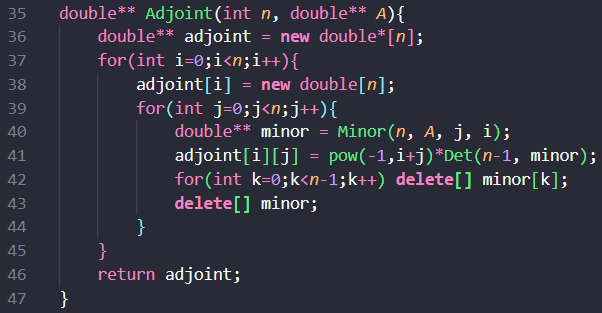
We calculate the determinant of by calling the function on submatrix . Clearly, we calculate the determinant of using recursion. After we update the value, we deallocate memory for .

1. We get the final value of det as:

which is determinant of matrix .

1. After the loop terminates, the resulting value of , which is the determinant of matrix is returned.

**III –** Adjoint Function



This function takes an integer ‘’ and a pointer to an matrix ‘’ as inputs and returns a pointer to the adjoint of matrix , .

Functionality:

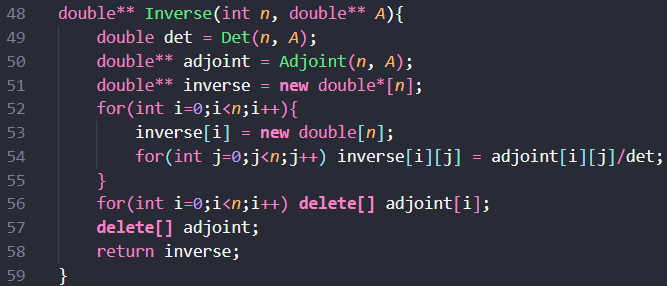
1. Initially, we allocate memory for the rows of adjoint matrix, named .
2. In each iteration of the outer loop from to , we allocate memory for each element of the row of .
3. In the iteration of the inner loop, we call the Minor function and assign the returned pointer of submatrix to a variable and then set the element at position of adjoint to be:

We calculate the determinant of by calling the function on submatrix . After we assign value of , we deallocate memory for .

1. Therefore, we find each element at position of to be:

, which is the value of any element of

1. After the loop terminates, the resulting pointer , which points to the matrix is returned.

**IV –** Inverse Function

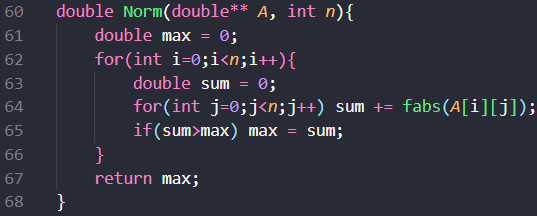
This function takes an integer ‘’ and a pointer to an matrix ‘’ as inputs and returns a pointer to the inverse of matrix , .

Functionality:

1. First, we compute the determinant of matrix by calling the function on and then store it in a variable .
2. Similarly, we compute the adjoint of matrix by calling the function on and store it in the pointer variable .
3. Next, we allocate memory for the rows of pointer variable . This will point to the inverse of matrix .
4. In each iteration of the outer loop from to , we allocate memory for each element of the row of .
5. In each iteration of the inner loop from to , we set the element at position of inverse to be:
6. Therefore, we find each element at position of to be:

which is the value of element at position of .

1. After the loop terminates, the resulting pointer , which points to the matrix is returned.

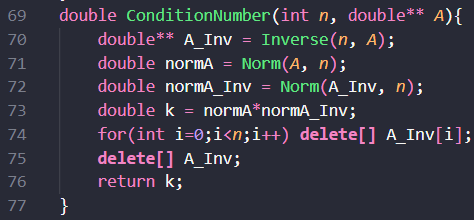
**V –** Norm Function

This function takes an integer ‘’ and a pointer to an matrix ‘’ as inputs and returns the norm of matrix ,.

Functionality:

1. We first initialize a ‘’ variable and set it to 0. This will be our norm value at the end of the function.
2. In the iteration of the outer loop, we define a ‘’ variable and set it to 0.
3. Using the inner loop, we update value of sum in each iteration of the main loop from loop to like so:
4. Our final sum value is .
5. If the ‘’ is greater than ‘’, then we set .
6. After termination of the outer loop, we get the maximum of the sums in the variable, i.e., we have, which is our required norm of matrix , The function then returns this value.

**VI –** Condition Number Function

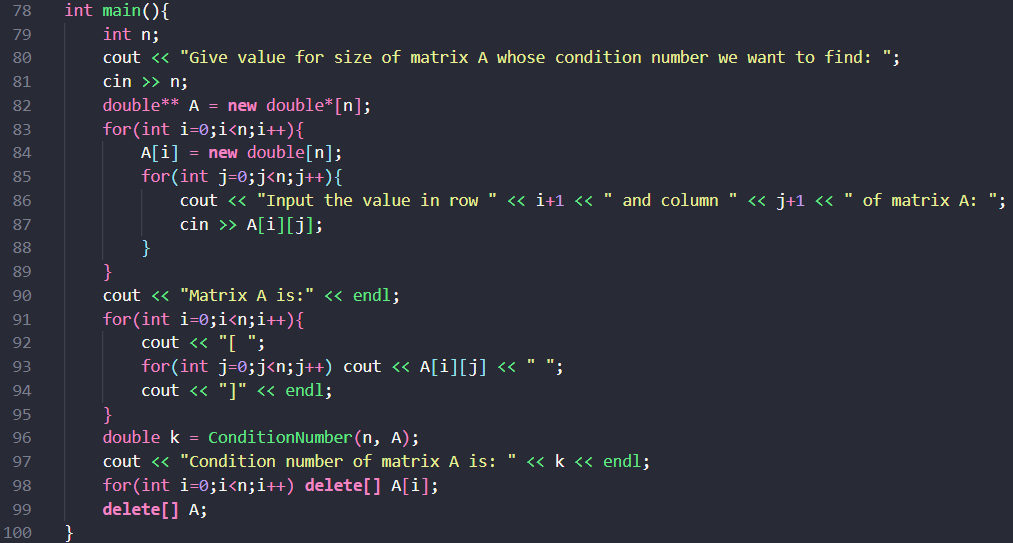


This function takes an integer ‘’ and a pointer to an matrix ‘’ as inputs and returns the condition number of matrix ,.

Functionality:

1. We get the inverse of by calling the function and storing it in pointer variable .
2. We compute the norms of and and store them in variables and respectively.
3. Finally, we compute the product of these norms and store the value in variable . Here, is our required condition number and this value is returned at the end of the function.
4. Finally, we deallocate memory for as it is not required anymore.

**VII –** Main Function

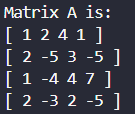


The main function is responsible for taking inputs, allocating necessary memory, calling functions, and computing and printing the necessary inputs/outputs.

Functionality:

1. We first take input for the size of matrix ‘’ and allocate the necessary memory for each element of the matrix ‘’.
2. The function then takes inputs for the elements of matrix ‘’ and prints the resulting matrix back to the user.
3. Then the function is called on the matrix ‘’ and the resulting value is stored in variable ‘’.
4. The program then prints the computed condition number ‘’, followed by freeing the memory of matrix ‘’ to prevent any memory leaks and ensure proper code termination.

**Condition Number of the Given Matrix:**

****

****This is matrix A for which we have to find the condition number.

As we can see here, the condition number of matrix is .

So, in a system , any small perturbation in vector ‘’, leads to a perturbation of the solution vector of about times the perturbation in . Due to this huge change in value of solution vector ‘’, we can say that matrix is ill-conditioned.

Answer 2-2: Using Grahm-Schmidt to find Orthogonal Matrix

The Grahm-Schmidt Method is a method used to generate an orthogonal matrix from the columns of a given matrix .

(A matrix is said to be orthogonal if the product of the matrix and its transpose results in the identity matrix, i.e., )

Explanation and Algorithm for Grahm-Schmidt Orthogonalization

Consider the matrix such that

Let vectors , then

Let , where are an orthogonal set.

(That is, )

Set .

We construct a vector such that it is orthogonal to the previous vector , i.e., .

Let , where is a scalar, then

As ,

Similarly, we now find a vector which is orthogonal to the previous vectors, i.e., and

Let where are scalars, then

As ,

As So we get

As ,

As So we get

Similarly, for a general , , we get that

where .

We normalize vector to say, .

We now construct a matrix with the orthonormal set as its columns to get , then is our required orthogonal matrix.

Note-1: Now, in C++, without the use of functions, it is impossible to access the column vector at a specified index. But we can use indexing to access the rows of a matrix in C++. So we modify our algorithm to use the transpose of matrix A, as the column vectors that were previously impossible to access by ordinary means now become row vectors, which we can easily make use of. So, transpose of is

Now, applying Grahm-Schmidt method to the row vectors of , we get the matrix (again, it is easier to set as row vectors of a matrix because row vectors are much more flexible and accessible than column vectors), which is just the transpose of our required orthogonal matrix . So, we can get our correct orthogonal matrix by transposing to get .

Note**-2:** In the Gram-Schmidt process, the matrix is used as an intermediate step to construct the orthogonal matrix . Since it is only an intermediate variable, whether its vectors are stored as rows or columns does not affect the outcome, so transposing it is not necessary.

**Explanation of Code for Finding Orthogonal Matrix from given Matrix using Grahm-Schmidt Method:**

**0 –** Preprocessing

A group of colorful text

AI-generated content may be incorrect.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++. Here the library ‘cmath’ has also been included to allow the use of in-built math functions in C++. ‘Using namespace std;’ prevents having to prefix input and output functions with “std::’.

**I –** dot Helper Function

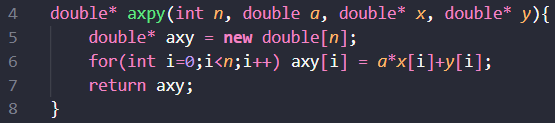
**A computer screen shot of a black background with white text

AI-generated content may be incorrect.**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 each iteration of the main loop from to like so:

Finally, we get the value of as

**II** **–** 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 each iteration of the main loop from to using the formula:

The resulting pointer is returned.

**III** **–** Transposition Function

A computer screen shot of a code

AI-generated content may be incorrect.This function takes size of matrix ‘’ and pointer to matrix ‘’ as inputs and returns a pointer to the transpose of matrix .

Functionality:

1. First, we allocate memory for the rows of pointer variable . This will point to the transpose of matrix .
2. In each iteration of the outer loop from to , we allocate memory for each element of the row of .
3. In each iteration of the inner loop from to , we set the element at position of to be:
4. Therefore, we find each element at position of to be:

, which is the value of element at position of .

1. After the loop terminates, the resulting pointer , which points to the matrix is returned.

**IV** **–** Grahm-Schmidt Function

A computer screen shot of a program code

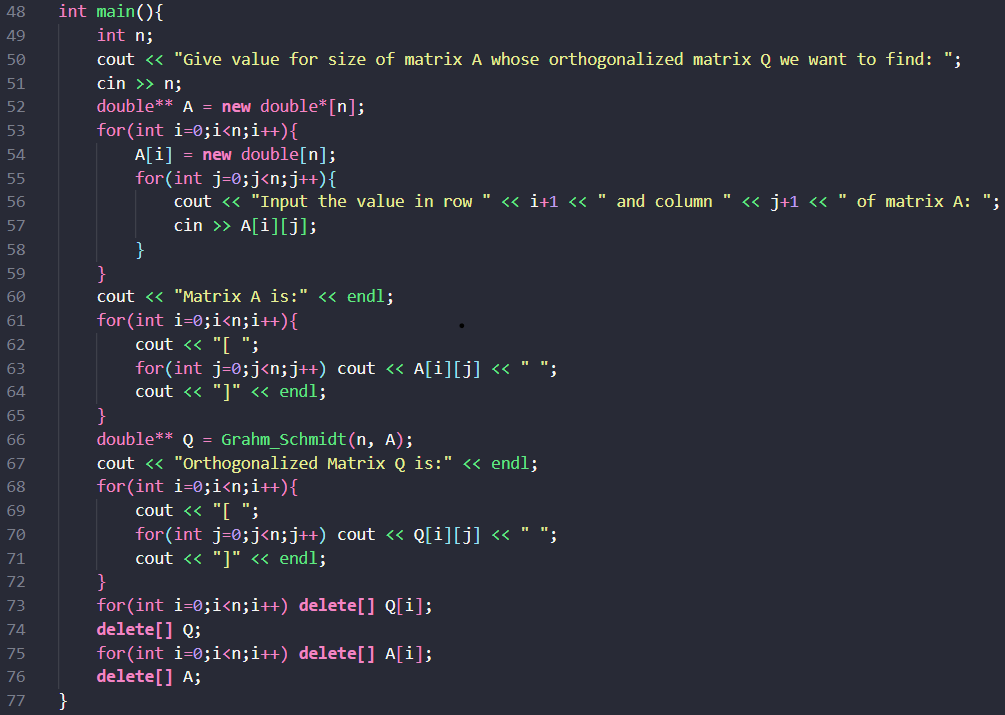
AI-generated content may be incorrect.

This function takes size of matrix ‘’ as input, a pointer to an matrix and returns the pointer to orthogonal matrix , obtained by applying the Gram-Schmidt method as explained previously to the columns of .

Functionality:

1. Pointer variables of size ‘’ and ‘’ are declared and necessary memory is allocated for both. ’ will point to the transpose of required orthogonal matrix ‘’ and ‘’ points to the intermediate set of vectors used for this process.
2. Variable ‘’ is initialized and set to the pointer of transpose of matrix by calling the Transposition function on A.
3. In the outer loop, we iterate over each row vector of . In this loop:
   * set to initially be in iteration.
   * Initiate the inner loop, in which we iterate each row vector of V before i.e., we iterate over all such that . In this inner loop:
     + We compute scalar and store its value in variable. Here, we are computing by calling the dot helper function on vectors and . We do the same in the denominator, but for vector twice.
     + Subtract from vector using the Helper function, where , and and store it in a pointer variable . Here, we are effectively calculating .
     + Then reassign the updated vector value in back to
   * Next, we compute the norm of , by taking the square root of result of dot product function on twice and store it in a variable ‘’, i.e. assign = .
   * We normalize vector by dividing with the computed norm. To do this, we call the Helper function again, where and and assign it to This effectively calculates , which is our normalized vector.
4. As we exit the loop, our matrix is complete. as it points to the transpose of our required orthogonal matrix, we call the Transposition function on and assign it to the variable ’. This ‘’ is our required matrix.
5. Finally, the memory taken up by the ‘’ and ‘’ is freed and the pointer to the orthogonal matrix ‘’ is returned.

**V –** Main Function



The main function is responsible for taking inputs, allocating necessary memory, calling functions, and computing and printing the necessary inputs/outputs.

Functionality:

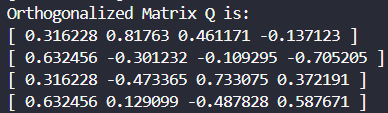
1. We first take input for the size of matrix ‘’ and allocate the necessary memory for each element of the matrix ‘’.
2. The function then takes inputs for the elements of matrix ‘’ and prints the resulting matrix back to the user.
3. Then the function is called on the matrix ‘’ and the resulting pointer to orthogonal matrix is stored in variable ‘’.
4. The program then prints the resulting orthogonal matrix ‘’, followed by freeing the memory of matrices ‘’ and ‘’ to prevent any memory leaks and ensure proper code termination.

**Orthogonal Matrix Q obtained from Gram-Schmidt:**

A screenshot of a computer

AI-generated content may be incorrect.

This is the given matrix A for which we perform the Grahm-Schmidt method.

Orthogonal matrix Q obtained using Grahm-Schmidt method is:

Answer 2-3: Householder Transformation to Solve the System

Explanation and Algorithm for Householder Transformations  
The Householder Transformation is a method that uses a sequence of orthogonal matrices known as Householder matrices to generate a right triangular matrix by multiplying given matrix by the series of Householder matrices. Then, we have:

We know that we can express matrix as a product of orthogonal matrix and upper-triangular , i.e., we can write . Comparing with the previous expression, we have

As the Householder matrices are orthogonal, their product is also orthogonal. Hence, we have

Now, we consider Householder matrix to be defined as:

,

where is the identity matrix and is a unit vector which will be determined below. Now, consider . Then we get

So, we get

Therefore, orthogonal matrix becomes . Initially, we set matrix .

We can also show that is orthogonal like so:

Now, let us first find .

Consider the matrix .

Take the first column vector

To convert A to an upper triangular matrix R, all elements below the diagonal have to be zero. So, in case of , we have to find such that multiplying it with zeroes out every element below , i.e., we have to find such that:

where is non-zero scalar and is the unit vector defined by

.

On pre-multiplication by , we get

So, we get scalar , i.e., is just the Euclidean norm of given by .

Again, consider our original expression and premultiply it by , then we have

Equating each element on both sides

for

Hence, we get and for .

Our resulting vector is

With this, we get vector and subsequently, matrix .

Now, premultiplying by , we get .

And finally, we postmultiply with and get .

Now, a second Householder transformation is required to zero out all the elements in the column below . For this consider the matrix obtained by removing the first row and column of the matrix . We use the vector given by:

.

Following a similar process to last time, we construct a Householder matrix where is the identity matrix and is vector given by:

Premultiplication of by will give

And finally, we postmultiply the updated value with and get .

Repeat the process for matrices obtained by removing two rows and two columns, three rows and three columns and so on till we are able to transform the coefficient matrix to upper triangle form and get the orthogonal matrix , where

and .

Now, we have and such that .

So let us consider the system *,* which we can write as .

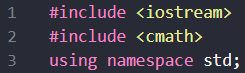
Premultiplying with , we have .

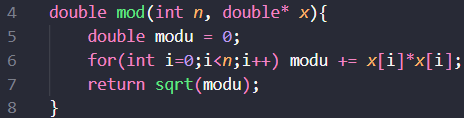
Let , then our system becomes .

As R is an upper triangular matrix, we can now solve this system using the Backwards Substitution Algorithm that we’ve previously discussed.

**Explanation of Code for Finding Solution to System using Householder Transformation:**

**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++. Here the library ‘cmath’ has also been included to allow the use of in-built math functions in C++. ‘Using namespace std;’ prevents having to prefix input and output functions with “std::’.

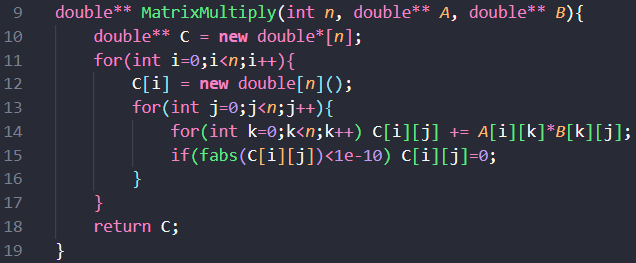
**I –** Modulus Helper Function

This function takes an integer ‘’ and pointer to a vector ‘’ of length as inputs and returns 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 and then the function returns the square root of the , , which is our required norm.

**II –** Matrix Multiplication Helper Function

This function takes size of matrix ‘’ and two pointers to matrices ‘’ and ‘’ as inputs and returns a pointer to the product of the two matrices .

Functionality:

1. First, the code allocates memory for the rows of pointer variable . This will point to the product of matrices and , .
2. In each iteration of the outer loop from to , we allocate memory for each element of the row of and then initialize all element values to zero.
3. In each iteration of the inner loop from to , we set the element at position of by incrementing its value using another inner loop from to like so:
4. Therefore, we find each element at position of to be:

, which is the value of element at position of .

1. After the loop terminates, the resulting pointer , which points to the matrix is returned.

A computer screen shot of a code

AI-generated content may be incorrect.**III** **–** Transposition Helper Function

This function takes size of matrix ‘’ and pointer to matrix ‘’ as inputs and returns a pointer to the transpose of matrix .

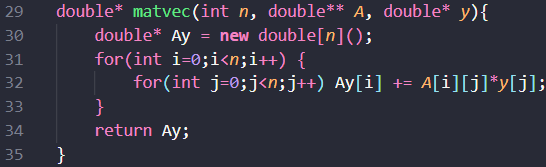
Functionality:

1. First, the code allocates memory for the rows of pointer variable . This will point to the transpose of matrix .
2. In each iteration of the outer loop from to , we allocate memory for each element of the row of .
3. In each iteration of the inner loop from to , we set the element at position of to be:
4. Therefore, we find each element at position of to be:

, which is the formula for element at position of .

After the loop terminates, the resulting pointer , which points to the matrix is returned.

**IV –** 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 .

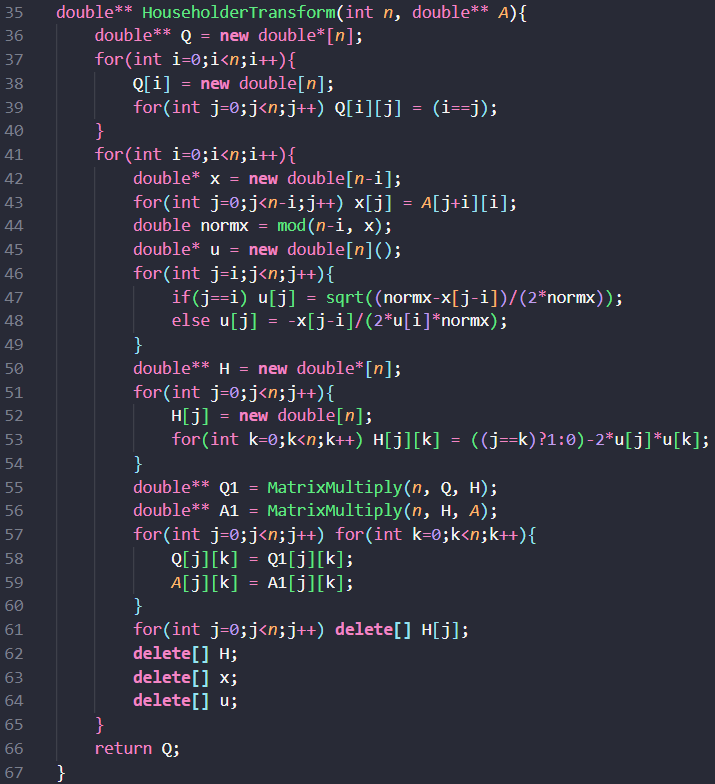
Functionality:

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

.

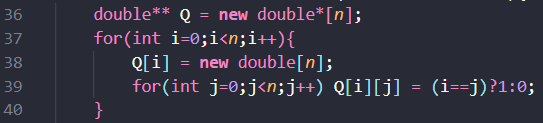
The final value of is computed as . At the end of the outer loop, the final pointer to vector is returned.

**V –** Householder Transformation Function

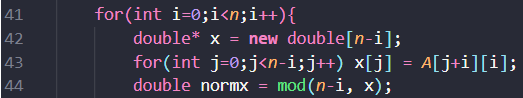


This function takes an integer ‘n’ and a matrix ‘A’ of size n x n as inputs, and after the series of Householder Transformations, returns the Orthogonal Matrix Q. In the process, matrix A is also converted to an upper triangular matrix.

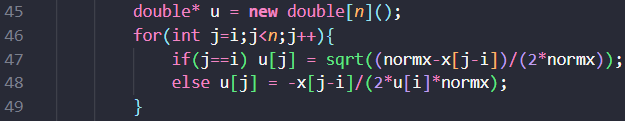
Functionality:



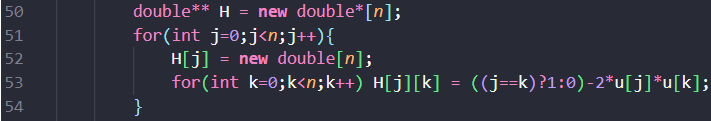
1. Here, the function allocates memory for the elements of matrix ‘’ and sets the element at position of to if and if , i.e., we initially set Q to the identity matrix .



1. Here, the function allocates memory and sets the value for a new vector of length such that it corresponds to the sub-column of the column of matrix , starting from row to the end. This is the portion of the column we use to obtain our unit vector ‘’. To do this, each element of , ] is set to in a loop from to . The Euclidean norm of vector is then computed by calling the mod function on vector and is assigned to variable .



1. Vector of length is initialized to zero. Elements of from index onwards are updated using the formulas we previously derived for elements of in our algorithm. Hence, the components of are set according to the formulae:



1. Memory for the elements of an matrix is allocated and filled using the formula for the Householder matrix: . With this formula, we can obtain the value of element at position of .

We know that

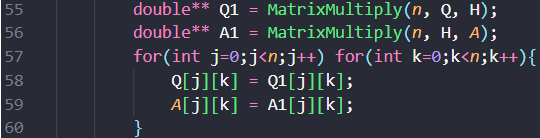
Expanding of both sides, we have

Comparing the components, we get that the element of matrix at position is given by:

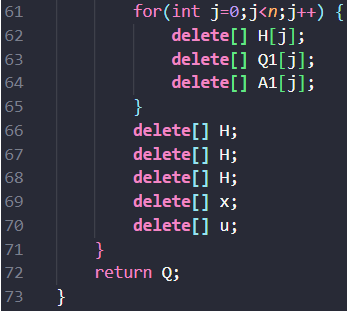
when and when

Using an outer loop from to and an inner loop from to , we assign the values of each element of , in accordance with the above formula.

This process creates a symmetric, orthogonal matrix that will reflect the column of such that all entries below the diagonal become zero.

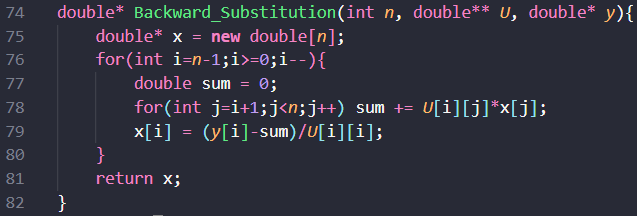


1. The Householder transformation is applied to and orthogonal matrix is generated via matrix multiplication:
   * accumulates the orthogonal transformation.
   * transforms the original matrix A by applying the Householder reflection.
   * After the multiplication, the contents of and are copied back into and respectively



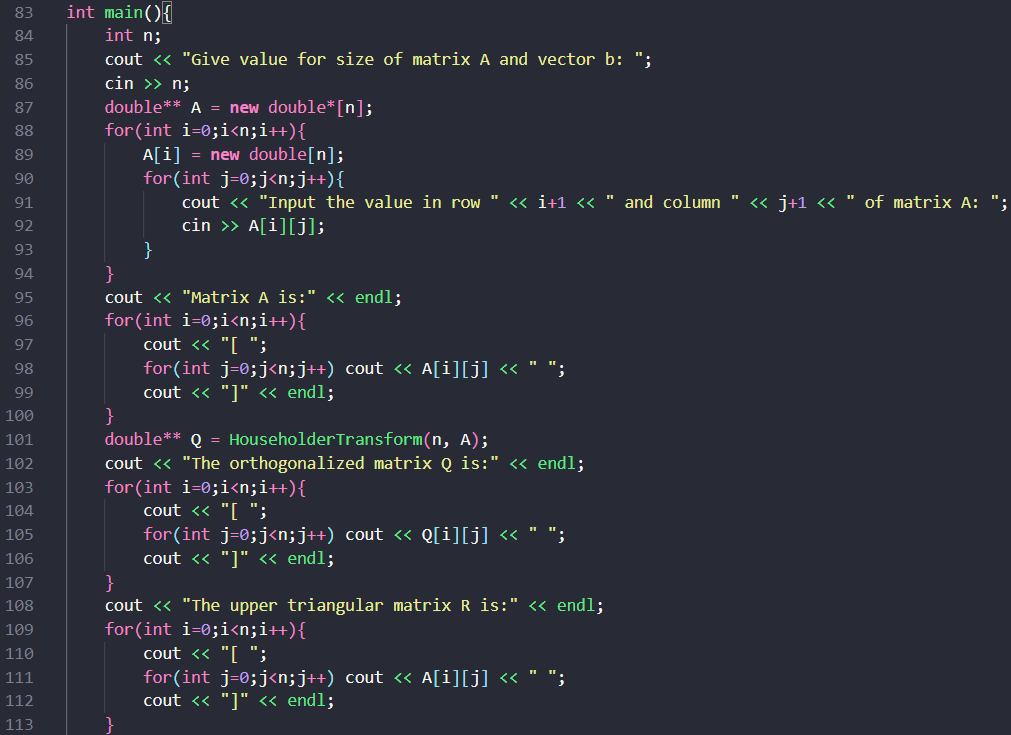
1. After the transformation at iteration of the main loop, the allocated memory for matrices and vectors is freed to prevent memory leaks. This ensures only necessary memory is kept at each iteration.

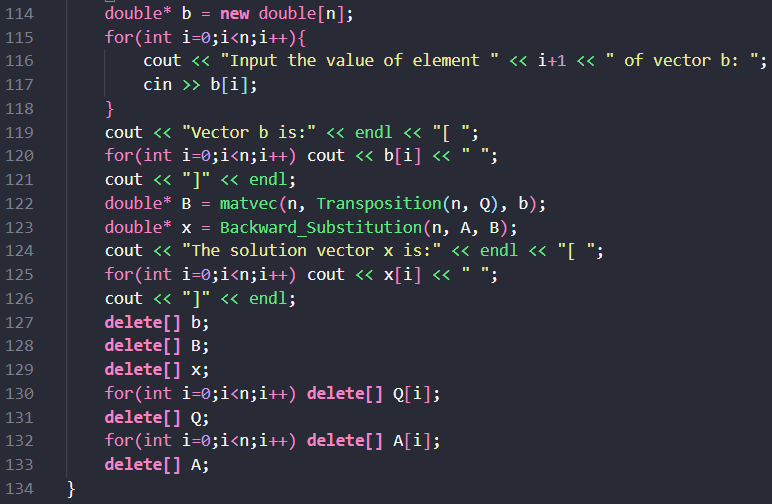
**VI –** Backward Substitution Function

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 .

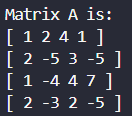
**VII –** Main Function

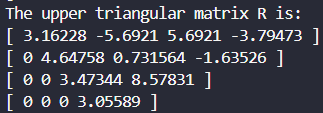
The main function is responsible for taking inputs, allocating necessary memory, calling functions, and computing and printing the necessary inputs/outputs.

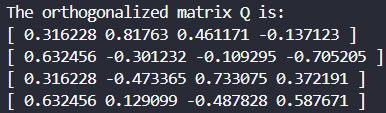
Functionality:

1. The function first takes input for the size of matrix ‘’ and allocate the necessary memory for each element of the matrix ‘’.
2. The function then takes inputs for the elements of matrix ‘’ and prints the resulting matrix back to the user.
3. Then the function is called on the matrix ‘’ and the resulting pointer to orthogonal matrix is stored in variable ‘’. During the function runtime, the matrix ‘’ is transformed into an upper triangular matrix.
4. The program then prints the resulting orthogonal matrix ‘’ and transformed upper triangular matrix ‘’.
5. Then, the function allocates memory for element of vector which along with matrix forms the system . This is followed by taking inputs for the elements of and printing the resulting vector back to the user.
6. The program then computes the vector by calling the function on matrix (which we get from calling the function on ) and vector . This new vector along with , constitutes our transformed system .
7. We then solve this transformed system by calling the backwards substitution function on upper triangular matrix and vector and assigning the solution to a pointer ‘’.
8. Finally, we print the solution vector ‘’, followed by freeing the memory of matrices ‘’,‘’ and vectors ‘’,’’,‘’ to prevent any memory leaks and ensure proper code termination.

**Matrices Q and R obtained from Householder Transformation and Solution to the given System:**



These are the matrix and vector that constitute our system .

After applying the Householder transformations, we get the orthogonal matrix and upper triangular matrix (just the transformed matrix ).

The solution vector we get after solving the system is:

Footnote 1: In this question, I have used the Window Powershell/Command Prompt to print my outputs.

Footnote 2: Due to the inconvenient manner in which I took inputs for my code, I directly just pasted the following inputs on program runtime to avoid wasting time entering them.

1. Condition Number: 4 1 2 4 1 2 -5 3 -5 1-4 4 7 2 -3 2 -5
2. Grahm-Schmidt: 4 1 2 4 1 2 -5 3 -5 1-4 4 7 2 -3 2 -5
3. Householder: 4 1 2 4 1 2 -5 3 -5 1-4 4 7 2 -3 2 -5 2 4 4 9