C++ Programme which solves a system of linear equations using Gaussian Elimination and Back-substitution methods.

Software Development I Course Project 2015

ELEN 2004

Martinique Koster 732293

Jared Ping 704447

**C++ PROGRAMME WHICH SOLVES A SYSTEM OF LINEAR EQUATIONS USING GAUSSIAN ELIMINATION AND BACK-SUBSTITUTION**

**Martinique Koster & Jared Ping**

*732293, Bachelor of Engineering Science in Biomedical Engineering, YOS2 & 704447, Bachelor of Engineering in Electrical Engineering, YOS2, ELEN 2004, School of Electrical and Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa, 13 April 2015*

**Abstract:** The design of a C++ programme which can solve a system of linear equations using the mathematical manipulations of Gaussian elimination and back-substitution is presented. This project demonstrates the use of programming techniques such as the use of multi-dimensional arrays, file streaming, loops and multiple functions within one programme. The analysis of the problem, design process, pseudo-code, troubleshooting and results (output) of the C++ programme are discussed. Troubleshooting the programme brought attention to the issue of an output value for one of the given input matrices to have a negative zero as one of the solutions for the unknown variables, as a result of very small decimal place holders which were not carried through to the final output file. It also highlighted the error of a solution being “not-a-number”. It was established that in order for the C++ programme to have a high accuracy whilst maintaining efficiency that the output solutions remain defined as double data types as set in the code. Also, an if-loop was introduced to parameterize the output solutions so that if they were not a number the programme would indicate that the matrix has no unique solution. The solutions of the matrices given in the *testData.dat* file are streamed into an output text file, namely *output.txt*.

**Keywords:** C++ programme, Gaussian elimination, Back-substituting, multi-dimensional arrays, file streaming, loops, functions, pseudo-code, troubleshooting, analysis, output, not-a-number, if-loop.

1. **INTRODUCTION**

C++ is a powerful, universally used computer programming language, which allows engineers and software developers to create functional and useful programmes in order to solve arithmetic problems encountered in various fields of occupation. The content of programmes vary with the needs and objectives of the programmer and vary from simple codes containing only one function, the main driver function, to complex algorithms which may require multiple structured functions that are then called into the main function.

Mathematically, a system of linear equations can be solved simultaneously for the unknown variables through the methods of manipulating a matrix by Gaussian elimination and back-substitution. An approach to solve a system of linear equations through these above mentioned methods in a C++ program, is to assign the various input coefficient values to a multi-dimensional array and then applying elementary column operations to the augmented matrix to convert it into echelon form. Then back-substitution is applied and an output of all the unknown variable values is given once the programme has completed the process.

* 1. *Problem Definition*

The project put forward asks for the C++ programme to solve a series of input matrices given in the data file *testData.dat*. It is defined in the project brief that all input matrices will be of the form, where is representative of the number of equations and is representative of the number of variables, thus giving a square matrix. Preceding the values of the coefficients and solutions to the linear equations is the value of, indicative for the size of the square matrix to follow. Together, the square matrix and solution value matrix (the last column presented) form the augmented matrix entered in to the programme. For readability purposes the code functions have been defined as suggested in the following list:

* *functRead()*:determines the size of each matrix and stores it in a multidimensional array.
* *functGauss()*: processes augmented matrix into triangular echelon form.
* *backSubstitution()*: derives solutions of unknown variables using back substitution.
* *functResult()*: assesses and writes results to output file *(output.txt).*
* *main()* : main function calls user defined functions above.
  1. *Analysis*

The approach taken to solve the problem given was to set up multiple user defined functions, as defined above, to go through the mathematical methods of simplifying the system of linear equations using accurate, yet efficient programming techniques. The following list of header files were included into the programme in order to use the corresponding specified operations:

* *iostream*: allowing the compiler to identify stream extraction and insertion operatives.
* *cmath*: allowing compiler to identify mathematical and the *“nan”* operatives.
* *fstream*: allowing program to stream in the input data file and stream out the output matrix solutions into *output.txt.*

A two-dimensional array was incorporated into the program to create a matrix so that values could be evaluated in the form of columns and rows. This allows the programme to analyse the linear equations with simple elementary column operations. The loops in the programme were designed in order to continue their calculations until the end of input file data is reached. If the programme encounters solutions which are not unique, have conflicting information or inconsistencies, the output is simply a statement which states the matrix solution was not unique.

1. **DESIGN PROCESS**
   1. *Method of Solution*

To solve the problem defined above, the C++ programme was separated into five user defined functions named above. First, the void *functRead()* function reads the first value presented in the data file, using the *fstream* library operatives, and assigns this to a two-dimensional array, initializing the size of the array. Next the function collects an input of matrix values. The values are assembled inside the defined two-dimensional array, where i represents the row location and j represents the column location to which the values are assigned. The resulting array is then called into the next void function; *functGauss().*

In the second function, column operations are applied in order to manipulate the given matrix into echelon form. Beginning with the first column the programme goes through each row of the matrix and makes all values equal to zero except for the value located at i=1 and j=1.

How this is done is by first checking if the current row position is greater than the current column position. If this is true the programme proceeds to do the following. It divides the number at the location it is presently at, by the previous row position. This number is then multiplied by the value assigned to the previous row position. The result of this operation is subtracted from the value assigned to the current location. This gives a zero at that position and the upper triangular echelon forms once all rows and columns have been operated on.

To apply the back-substitution method to the echelon matrix called from the previous function, the unknown variable we wish to solve for is assigned the answer of the value found in the last row and last column divided by the coefficient which was left from the Gaussian elimination manipulations (this coefficient is the value assigned to the location of which is representative of the desired unknown variable). We output this as the solution for the last unknown variable (). To find all following unknown variable solutions, (that is moving backwards through the matrix from) the desired unknown variable for a specific column is assigned the answer of the solution found in that row (value in the very last column) minus the previously defined unknown variables multiplied by their corresponding coefficients in that row currently in operation, all of which is divided by the coefficient of the desired unknown variable.

Finally, the last void function *functResult()*  prints all solutions to the unknown variables into an output file.

However, for this C++ programme to work, all the above mentioned functions must be called into the driver function; that is the main function. The compiler begins reading the main function and from there calls all other user defined functions and performs the various operations, thus ending the programme once the input data file has reached its end and returns the resulting output presented.

* 1. *Pseudo-code*

Below are the algorithms, or otherwise referred to as pseudo-codes, for each user defined function within the C++ programme presented.

***functRead():***

Input: Matrices from *testData.dat* file.

Output: Writes data from file into a matrix that can be operated on

i ← 1

j ← 1

n ← first value from file

matrix array ← matA[n][n]

for i ≤ n do

for j ≤ n+1 do

z ← next value from file

matA[i][j] ← z

j++

end for

i++

end for

***functGauss():***

Input: Matrix array created by read function

Output: Matrix operated into triangular upper echelon form

j ← 1

i ← 1

k ← 1

x[]

For j ≤ n do

For i ≤ n do

If i ≥ j then

M =

For k ≤ n+1 do

matA[i][j] ← matA[i][k] – M\*matA[j][k]

k++

end for

end if

j++

end for

i++

end for

x[n] ←

***backSubstitute():***

Input: Matrix in triangular upper echelon form

Output: Substitutes back into normal matrix to solve for values of x

i ← n-1

j ← I+1

For i ≥ 1 do

Sum ← 0

For j ≤ n do

Sum ← Sum + matA[i][j]\*x[j]

j++

end for

x[i] ←

i--

end for

***functResult():***

Input: Solved matrix array

Output: Results of the solved variables x

i ← 1

For i ≤ n do

If isnan(x[i]) then

output file ← No solution

break

Else do

output file ← x[i]

end if

i++

end for

* 1. *Final Code*

The final C++ coded programme is written in the attached *SoftwareProject15.cpp* file.

1. **RESULTS**

The results of the programme were streamed into an output text file (*output.txt*), with an initial statement declaring what type of square matrix the followed solutions are for, followed by the declared unknown variableand its assigned solution. If the programme encountered solutions which were not unique, had conflicting information or inconsistencies, or was found to be *not-a-number*, the initial statement is followed by *“There is no unique solution”.*

1. **DISCUSSION**

Troubleshooting brought attention to issues within the results given in the output file. An output value for one of the given input matrices has a negative zero as a solution for. This is due to very small decimal place holders which were not carried through to the final output file as the variables were defined as a double data type. The remaining decimal places are important for following mathematical operations carried through to the back-substitution function, therefore it was decided not to truncate the solution in order for the output to have the modulus of that decimal.

An error of a solution being *not-a-number,* was encountered during troubleshooting of the programme*.* An if-loop was introduced into the user defined function *functResult()* with a parameter that if the solution were *not-a-number* then the programme would output that the matrix has no unique solution

1. **CONCLUSSION**

In conclusion, the C++ programme presented is capable of streaming in an input file containing multiple systems of linear equations, applying Gaussian elimination and back-substitution operations, deciphering which solutions are unique and streaming the output to a new text file. It has proven to be sufficiently efficient and gives very reasonably accurate solutions to the matrices given to solve in the *testData.dat* file. Keeping in mind that the ELEN2004 software course is an introduction for beginners to C++ programming, it is evident that improvements to simplification and efficiency of this program can be made, if the knowledge of the C++ language is improved.

1. **TIME-ACTIVITY LOG**

Refer to *Figure 1*, the Gantt-Chart of the Time-Activity Log, which tracks the time spent on different aspects of this project.

**REFFERENCES**

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

[]

*Figure 1: Gantt-Chart of Time Activity Log*