STATIK UND DYNAMIK DER FLÄCHENTRAGWERKE

UNIVERSITÄT DUISBURG ESSEN

Offen im Denken

Anupam Chakraborty
ES0303924200
M.Sc. Computational Mechanics

Computer Language for Engineers

Project Report

Universität Duisburg-Essen
Institut für Mechanik
Fakultät für Ingenieurwissenschaften
Abteilung Bauwissenschaften

Supervision

Dr. Rer. Nat. ERNST BAECK

August 2019

Contents

List of Figures

List of Tables

1 Introduction to programming languages

Programming languages provides a means to communicate with the computers through instructions often referred to as a code or set of codes fed in syntactical order in an apprehensive manner as dictated by the guidelines of a Programming Language. These set of instructions/codes devised for carrying out a task or a multitudes of tasks are called Programs. The guidelines of a Programming Language are the syntax which are predefined by the developers and that could be easily formulated by the computer to be comprehended in it's own language, termed as the Machine Language. This method of formulation is famously known as Compilation.

In view of this subject, following are the two programming languages explained concisely in which the assigned tasks have been written.

1.1 FORTRAN

Originally developed by IBM, Fortran (formerly, FORTRAN) stands as an abbreviated form for "Formula Translation" (For = Formula + Tran = Translation). It was developed predominantly for numeric computations and scientific computing in 1950's for scientific and engineering applications and was first published in 1957. It's structured programming led to easy translation of math formulae into code and hence is widely used.

The first ever compiler developed for Fortran rendered it a powerful scientific language since, it made all the other languages obsolete which required machine language coding which were highly esoteric, time consuming and involves painful debugging.

1.2 C++

For most of the history of computer programming, most programs were written procedurally. Procedural programs consist of a series of steps or procedures that take place one after the other. The programmer determines the exact conditions under which a procedure takes place, how often it takes place, and when the program stops. Over the years, as programmers have sought better ways to accommodate the way people work best on computers, procedural programming techniques have evolved into object-oriented techniques. Maintaining the traits of procedural programming like in C, C++ was developed with an added features of Object-oriented programming. Extending C for the scope of including these features, C++ was developed by Bjarne Stroustrup at Bell Labs since 1979.

1.3 Object-Oriented Programming (OOP)

Dissertation of a program into its elements, where each element characterises its own attributes and functionalities - is the prime idea upon which Object-Oriented Programming, (hereafter, shall be referred with its abbreviated form OOP) is based. Every element or an object constitute together a class of objects of similar kind. These classes are then constituted together to form a program.[1]

2 Project 1 – Implementation of Newton's Method to evaluate roots of functions in Fortran90

In engineering and similar scientific studies arise a multitudes of non-linear equation systems, e.g., elastic and plastic deformation processes in engineering. Finding definite solutions of such non-linear equations involves many trivialities. Hence, Iterative methods provide an escape to these trivialities. Newton's Method, (a concise form of Taylor's Expansion) is one such iterative Method for extracting roots of equations.

2.1 Problem definition

Following functions were given as a task for which the roots are supposed to be determined.

$$f_1(x) = \frac{2x - 8sin(2x)}{1 - x^2}.tan(x) \tag{1}$$

$$f_2(x) = \frac{x^6}{7} + \frac{x^5}{2} - 2x^4 + 2x^2 - 10x - 3 \tag{2}$$

$$f_3(x) = (1 - x)e^{\frac{-x}{2}}(1 - 2cos(\frac{x}{2})) - (1 + x)e^{\frac{x}{2}}(1 + 2cos(\frac{x}{2})) \tag{3}$$

The above functions are highly non-linear in character and hence, finding the roots through an Iterative Numerical Method is imperative in order to avoid the trivialities involved in the determination of the exact root values. Hence, for this project, as it dictates the implementation of Fortran90 program for Newton's Method, also known as Newton-Raphson Method, an iterative numerical approach for evaluating roots of a function, the method is explained in details in the following section.

Due to the non-linearities of the given functions, it can be estimated that there could be infinite roots if plotted on a Number line. Hence, it is convenient to consider a fixed interval within which the roots shall be evaluated.Newton's Method comprise of an arbitrary initial value x_0, hence, also at times referred to as the Initial Value Problem, which is further evaluated for finding the roots of the function $f(x)$.

The roots found are verified systematically to foresee if any error exists. Functions are re-plotted using MATLAB programs to analyse the values of the roots for both quantitative and qualitative purposes.

2.2 Newton's Method

This Iterative Method aims at finding an approximate value of the root of the function with an initially provided value. With every new iteration, we draw much closer to the exact value of a root as interpreted below mathematically.

Let the function f which maps itself in a space of Real numbers be a continuously differentiable function.

$$f : R^n \to R^n$$

As explained above, the approach of Newton's Method aims at finding the root of f, i.e., finding $\mathbf{x}^* \in R^n$ such that $f(\mathbf{x}^*) = \mathbf{0}$.

Considering the Taylor's Expansion,

$$f(x+h) = \frac{h^0}{0!}f(x) + \frac{h^1}{1!}f'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + ... + 0(h^n) \tag{4}$$

So Taylor Expansion of Order 1 about a point $x^{(k)}$ in an infinitesimally small neighbourhood $x^{(k)}$ can be expressed as,

$$f(x) \approx f(x^{(k)}) + f'(x^{(k)}) - (x - x^{(k)}) \tag{5}$$

Thus, Newton's method can be idealised as,

$$x^{(1)} = x^{(0)} + \Delta x \tag{6}$$

where, $\Delta x = -\frac{f(x)}{f'(x)}$

Therefore,

$$x^{(1)} = x^{(0)} - \frac{f(x^{(0)})}{f'(x^{(0)})} \tag{7}$$

For perpetually evaluating the roots iteratively until a value where $f(x) = 0$ or $f(x) <= tol$ where tol is the tolerance tending to zero ($tol \to 0$), when the method does not yield a value of the function equal to zero after certain numbers of iterations.

$$x^{(n+1)} = x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})} \tag{8}$$

The derivative of the function is approximated numerically by Central Differencing Scheme as below,

$$f'(x) \approx \frac{f(x + \frac{h}{2}) - f(x - \frac{h}{2})}{h} \tag{9}$$

where h is an infinitesimally small distance or perturbation which will enable us to draw a chord between the initial value and the successive value of the function, approximated as the tangent at the initial value, since the slope of both chord as well as the tangent is nearly the same.

With the following example, an illustration to the Newton's Method could be made.

Considered function $\to f(x) = x^2 - 1$;
Its derivative $\to f'(x) = 2x$;
inital value $\to x^{(0)} = 0.3$;

Following graphical plot ascertains how Newton method works

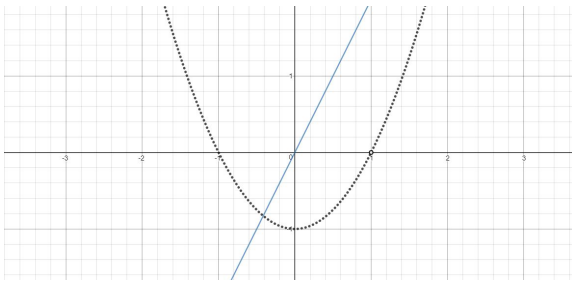


Figure 1: Newton's method plot for $f(x) = x^2 - 1$

It is quite picturesque that from an initial value of $x(0)$, a root of the function is evaluated in every iteration, however, not exact. With every iteration, the value is closer to the exact value of the roots than the previous one.

Following values of x over the iterations by Newton's Method were found.

Iteration No.	--- x* -------	f(x*) --
0	0.300000	-0.910000
1	1.816667	2.300278
2	1.183563	0.400821
3	1.014235	0.028672
4	1.000100	0.000200
5	1.000000	0.000000

Figure 2: Newton's method Tabular results

Care shall be taken where $f'(x^*) = 0$, since a vanishing slope appears and the tangents drawn from that point is parallel to the x-axis, and doesn't yield any root value.

2.3 Programmer's guide

The programs are coded, implemented and debugged on the **Code::Blocks** IDE, the application which provides ways to edit, compile, run and debug the edited programs. This section describes the understanding of the developed program to find the roots of the arbitrary function. The code is developed using subprograms or modules and each subprogram is discussed in this section.

2.3.1 Implementation of $function.f90$

In subprogram file **function.f90** the input function **myF** has been introduced to input the functions whose roots need to be evaluated. It substitutes x in the input function and returns the function value $f(x)$ at x. The flow diagram of the program $function.f90$ is represented as follows.

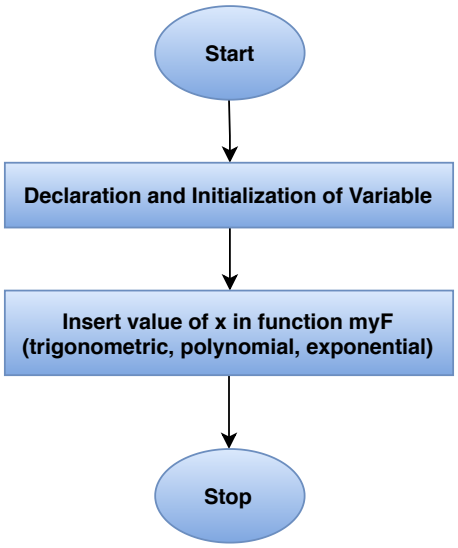


Figure 3: Flow diagram for function.f90

2.3.2 Implementation of $Anufunction.f90$

In subprogram file **Anufunction.f90** the derivative of function **fp** is implemented using central difference scheme. It returns $f'(x)$ at x. The flow diagram of the program $Anufunction.f90$ is represented as follows.

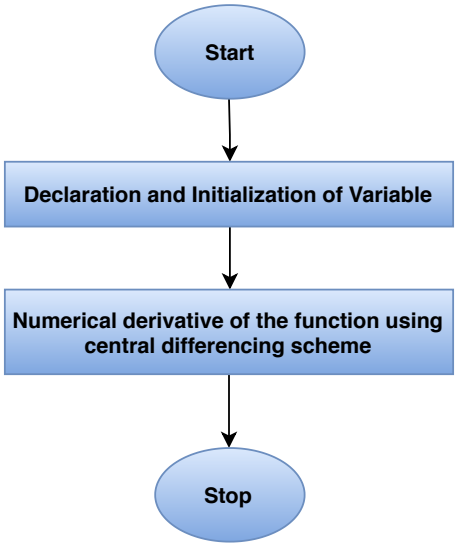


Figure 4: Flow diagram for Anufunction.f90

2.3.3 Implementation of $mainNewton.f90$

The **mainNewton.f90** is the main program to find the roots of the given function $f(x)$. In this program various function viz. myF, fp, $newton$, $readData$, $ScanForRoots$, $funcToWriteResult$ has been declared. There are various variables declared. After declaration of the variables, the subprogram **readData** has been called to read the input data and to store them. It will also check for an error in input file. If it not finds any error then dynamic array is allocated to store the roots. Then **ScanForRoots** subprogram is called to find the roots of the input function f(x). If it will find any error to open the log file then it shows the error message. If the number of roots are greater than zero then it prints the solution on the screen and to the log file. If there are no roots found then it shows an error message. The flowchart of the mainNewton program is represented as follows.

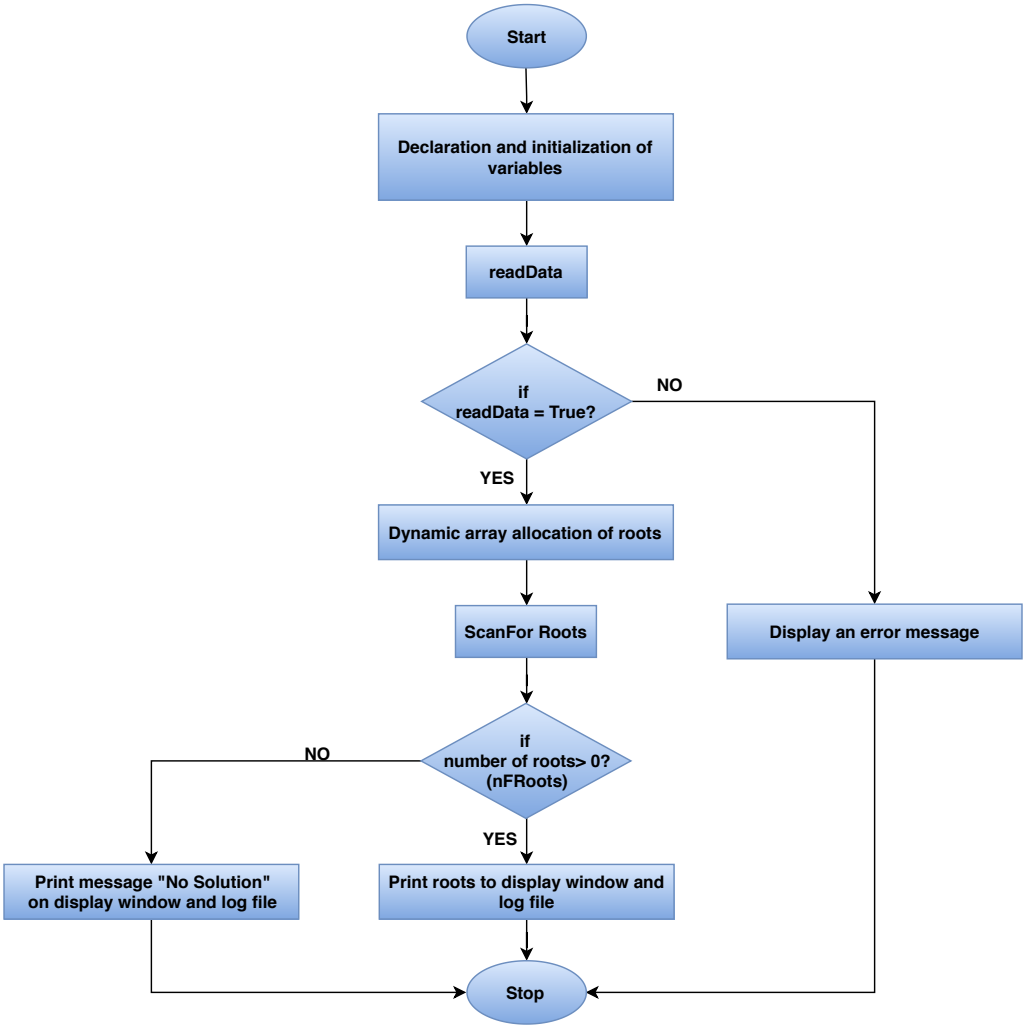


Figure 5: Flow diagram for mainNewtonf90

2.3.4 Implementation of *readData.f90*

The subprogram **readData.f90** is developed to read the input data from the input file and assign the values to the parameters passed. In case of any incorrigible errors found during the runtime, the program shows the error message and try to fix the error. If the fixing of the error is not possible then program will be stopped.

It is essential to understand the algorithm used for checking an errors while reading the input data from input file. The detail algorithm has been discussed here. If there is any error present while reading the file, it will show an error message and further execution of the program will be stopped. The first line consist the maximum number of roots need to be found. The algorithm of error 1 is represented in Figure-6. The number roots need to be found must be a positive number, so it can not be zero or negative number. If number of roots need to be found set zero then it will correct and set minimum positive number i.e 1. If any negative number is set for number of roots need to be found then program will multiply it with (-1) to get positive number.

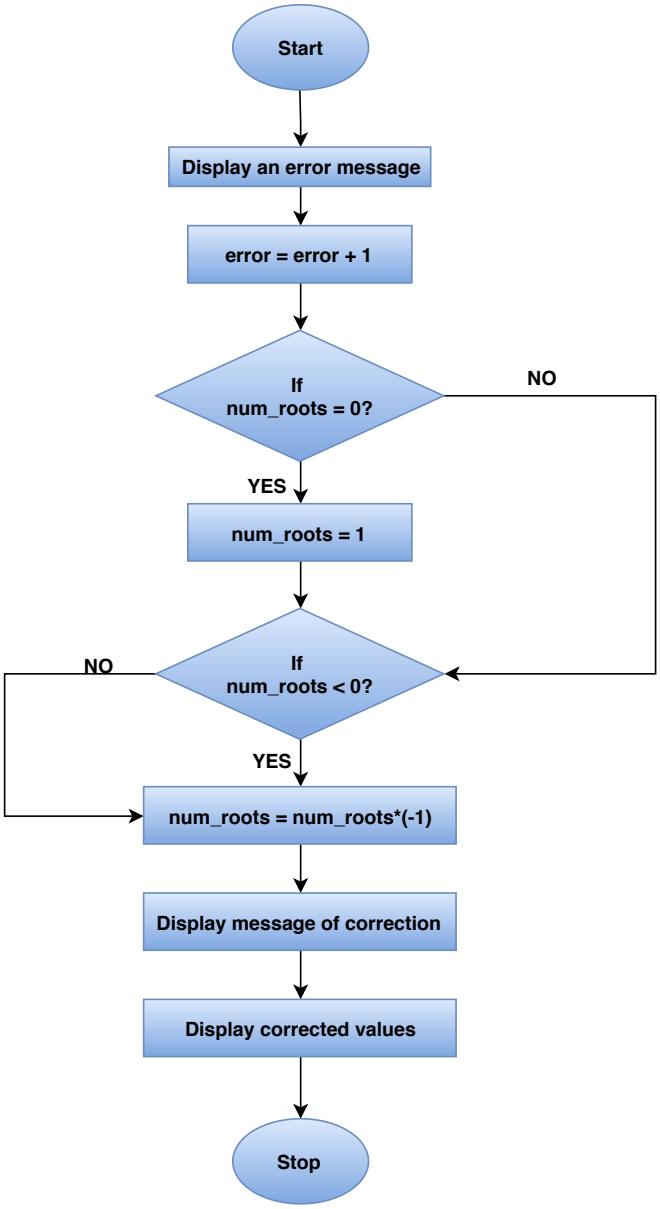


Figure 6: Flow diagram for error-1

The second line consist lower and upper bound of the search interval and step width for newton algorithm. The value of lowerBound must be lesser than upperBound. If by mistake one set the value of lowerBound greater than upperBound then program should exchange the values of upperBound and lowerBound. If one enters the same values for the lowerBound and upperBound, then execution of the program will be stopped. The flow chart of the same is represented in Figure-7.

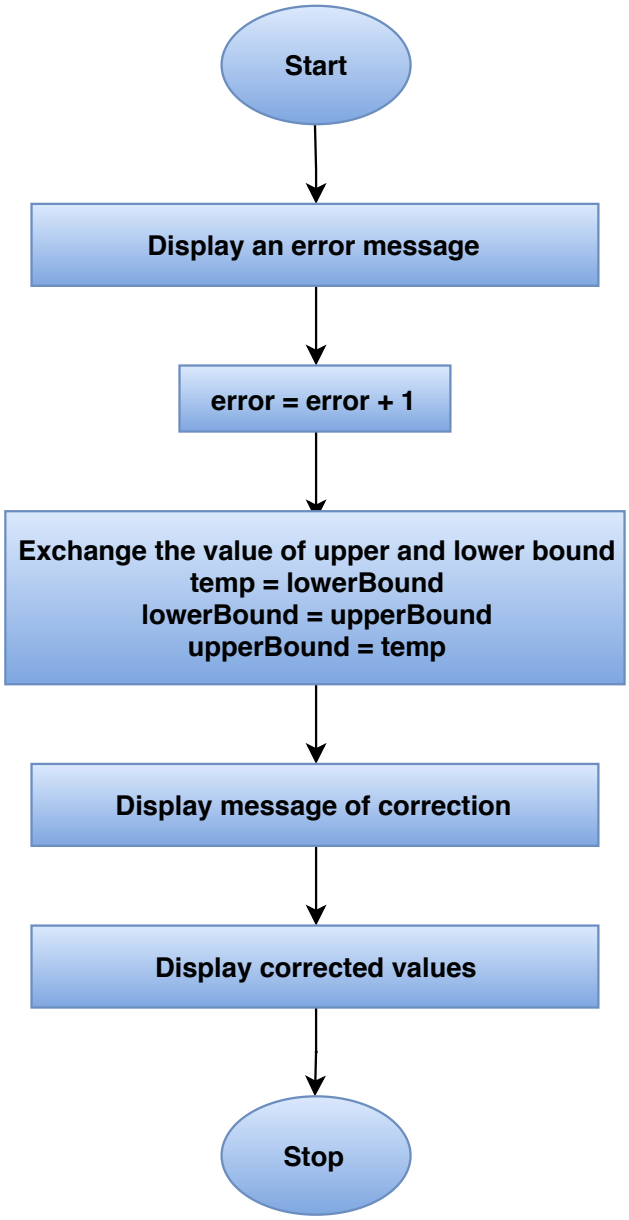


Figure 7: Flow diagram for error-2

The value of the step width must not be zero or greater than the width of the search interval. In such case the step width is set to 0.01 if initial guess x0 is equal to lowerBound of search interval. It will set to -0.01 if initial guess x0 is equal to upperBound of the search interval. The flow chart of the same is represented in Figure-8.

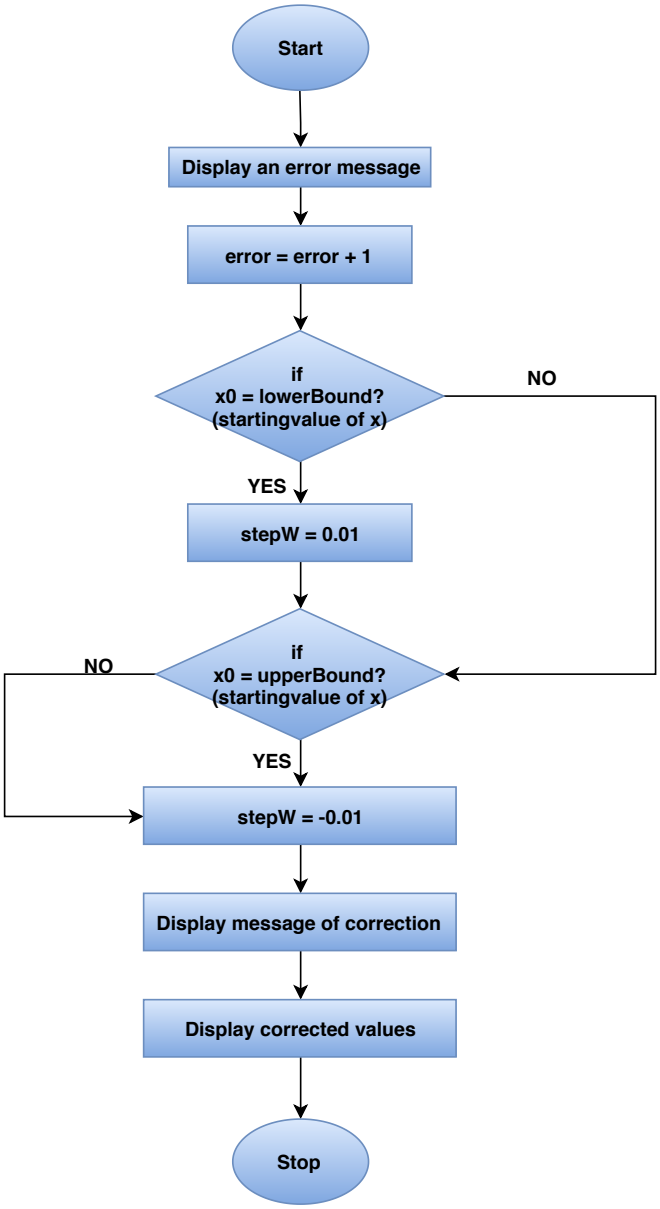


Figure 8: Flow diagram for error- 4 & 5

The third line in input file represents the precision for finding the roots, step width for derivative and maximum number of iteration for one root search. If the precision is less than 0 or greater than $1.e^{-7}$,then it will set to $1.e^{-7}$. The flow chart of the same is shown in Figure-9.

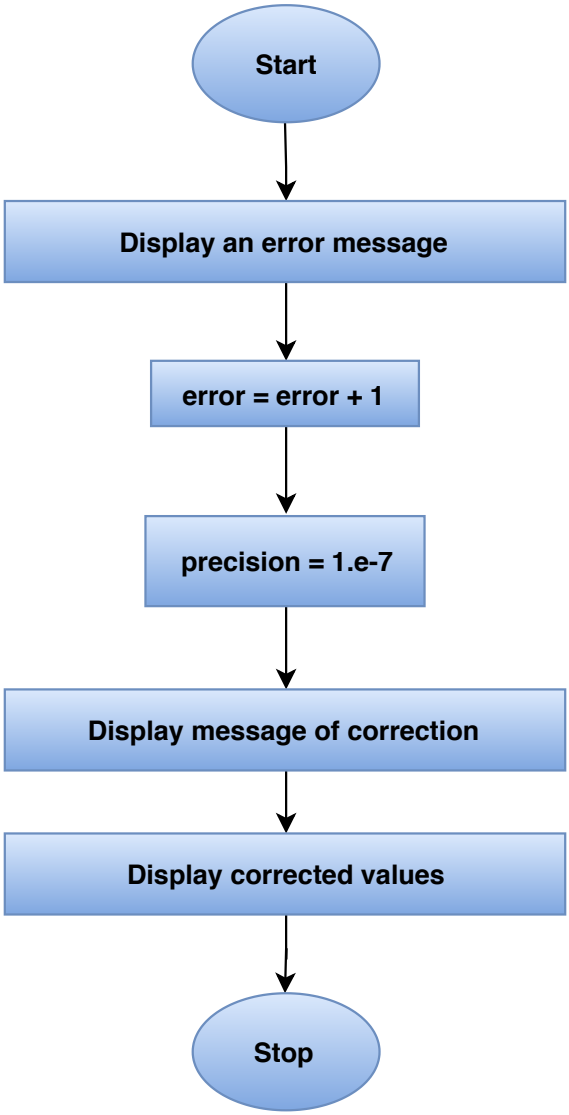


Figure 9: Flow diagram for error- 6

If the step width for calculating numerical derivative of the function is zero or if the step width greater or equal the width of the search interval then it gives an error. In such cases the step width for the derivative will be set to 0.01. If the number of iteration for one root search is less or equal to 0 then it must show an error message. The flow chart of the same is shown in Figure-10.

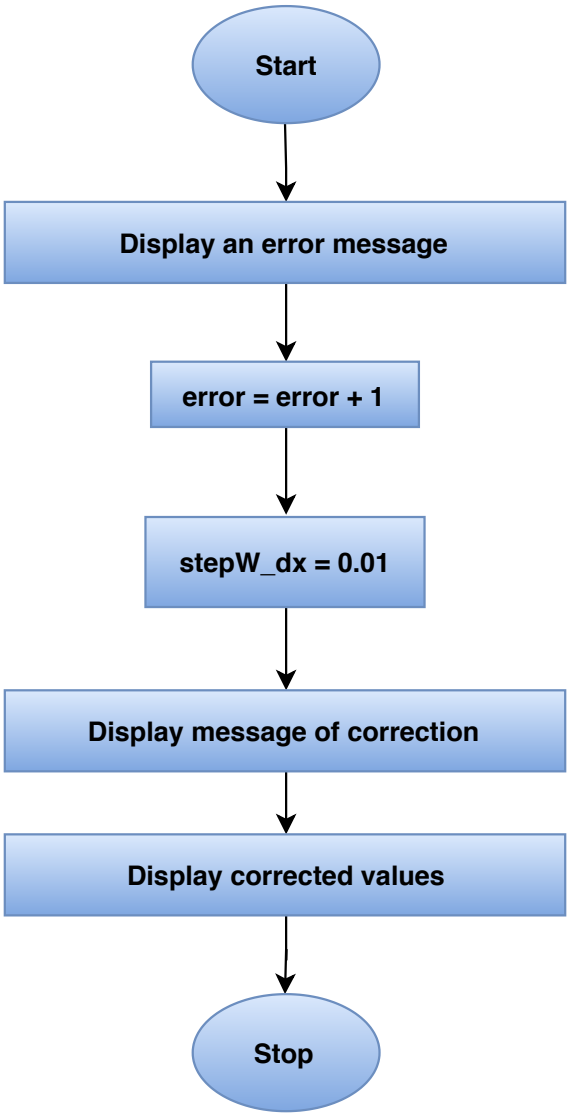


Figure 10: Flow diagram for error- 7 & 8

If the maximum number of iteration is negative value then it will be multiplied with (-1) to get positive number and if the maximum number of iteration is equal to 0 then it will be set to 100. The flow chart of the same is shown in Figure-11.

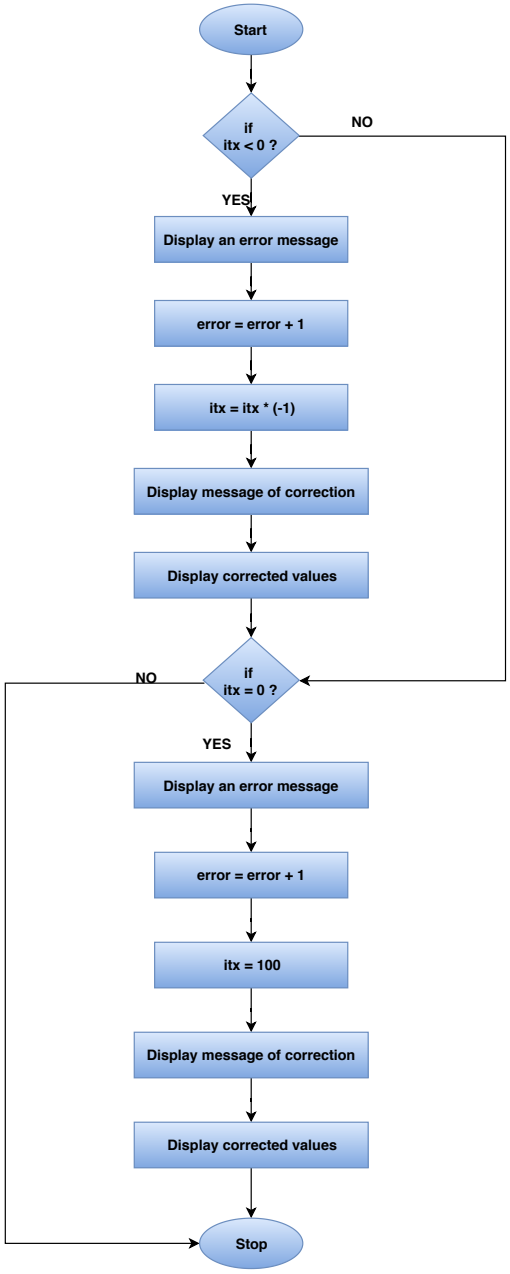


Figure 11: Flow diagram for error- 9 & 10

2.3.5 Implementation of $ScanForRoots.f90$

The program **ScanForRoots.f90** is used to find the roots of the function and returns the number of roots found. The subprograms viz f, $newton$ and $CheckDuplicateRoots$ are called. The initial value $x0$ is set to the lower bound of the search interval. If the root is found then it will be checked whether it lies within the required search interval. If the root is lying in the required interval then it will be checked for its uniqueness. This is done by calling module $CheckDuplicateRoots$. If the found root satisfies all condition then it will be saved. If the conditions are not satisfied then the root is discarded. In both the cases, the value of initial guess x is incremented for next root search. If the new value of x does not lie within the interval then control exits the function. This loop goes on running until the number of roots found are less then maximum number of roots to be found. Once all the roots in the given interval are found then the $ScanForRoots.f90$ returns the number of roots found. The flow diagram of $ScanForRoots.f90$ is represented in Figure- 12.

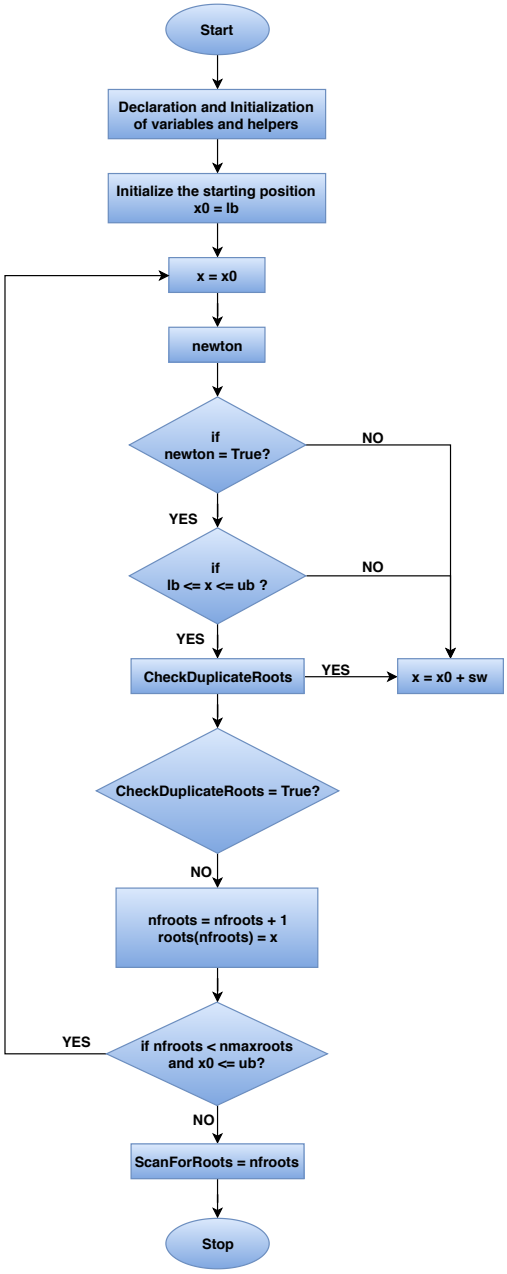


Figure 12: Flow diagram for ScanForRoots.f90

2.3.6 Implementation of *CheckDuplicateRoot.f90*

The program **CheckDuplicateRoot.f90** is implemented for checking of the uniqueness of the roots by neglecting the repeating roots. A root found is compared with all other found earlier and if the difference is less than a certain precision then the newly found root is discarded. The flowchart of the program *CheckDuplicateRoot.f90* is represented as follows.

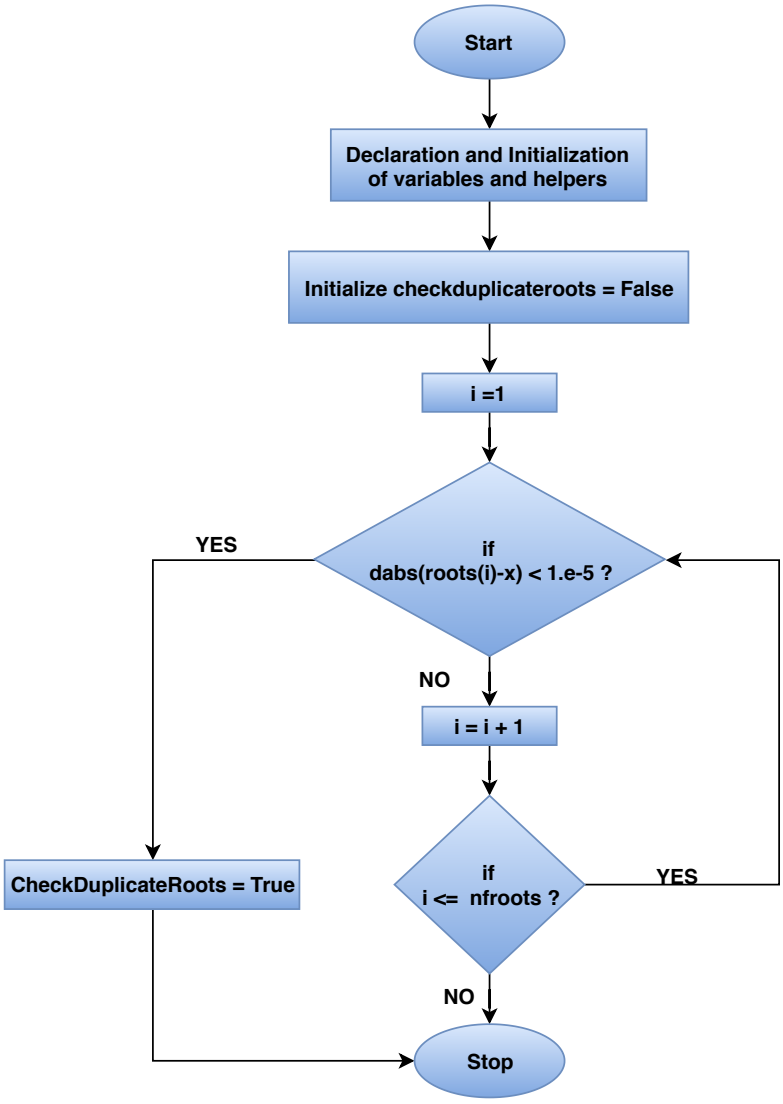


Figure 13: Flow diagram for CheckDuplicateRoot.f90

2.3.7 Implementation of *newton.f90*

The program **newton.f90** is implemented to find the roots of the function. It will perform the iteration to find the value of the function and its derivative. If the function's value $f(x)$ is close enough to zero then the corresponding x value is one of its roots and the control return to the program *ScanForRoots.f90*. If it is not close enough to zero then slope of the function is checked. If the slope is close to zero then the x value is incremented as $x = x + s$. The iteration continues until the maximum number of iteration reached. The flow diagram of *newton.f90* is represented in Figure-14.

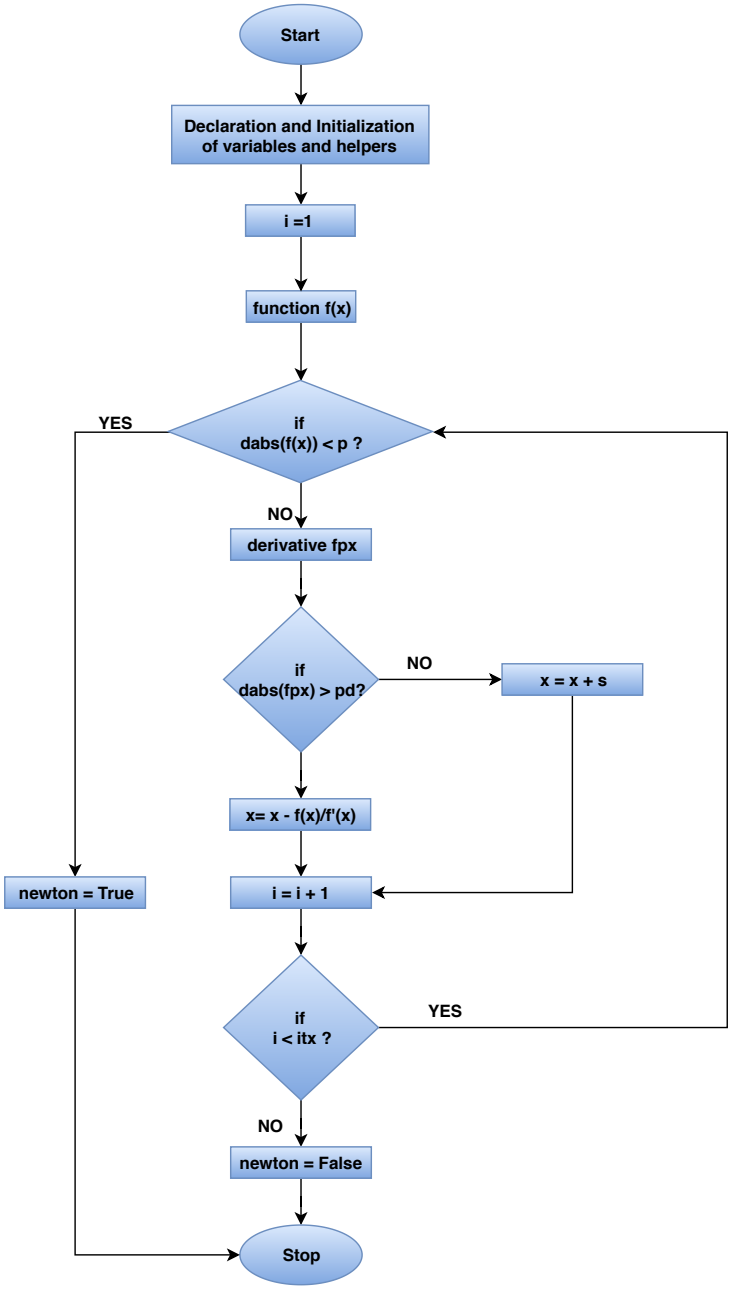


Figure 14: Flow diagram for newton.f90

2.4 User Manual

This section provides the instruction to use the developed **Fortran** code to compute the roots of the arbitrary function using newton's method.

- **Open the developed program** – Open the **Code::Blocks** from the task bar of the computer and open 'Project1.cbp' file from the project folder 'Project-1-Newtonmethod'. User can see the window as shown in Figure-15.

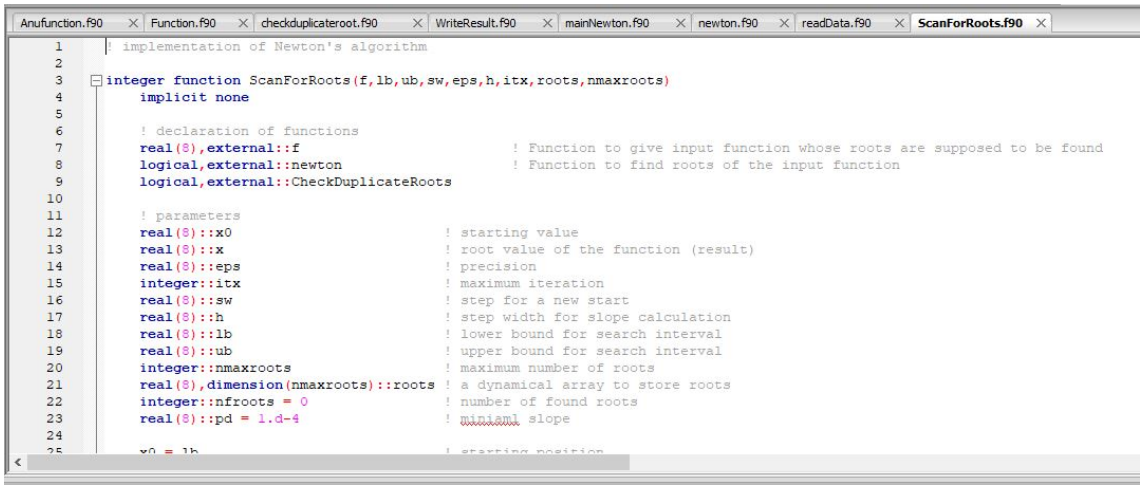


Figure 15: Display of Code::Blocks GUI after opening project file

- **Open function.f90 file, write the function and run the program** – Open the 'function.f90' file from the project folder and user can see the window as shown in Figure- 16. The 'function.f90' file consist different input functions. There are three functions written and the trigonometric function is un-commented in this example using ! exclamation mark to find the newton roots as shown in Figure-16. User can also add new function as per requirement and also remove the functions which are not needed. The user should specify the input parameters as discussed in section-2.5.

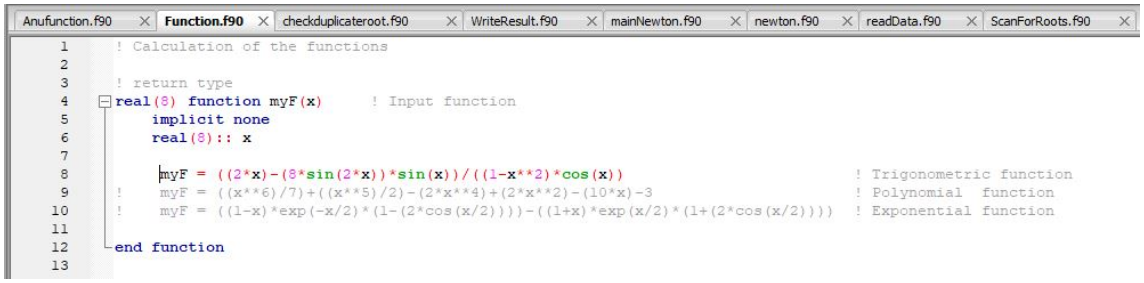


Figure 16: Display of Code::Blocks GUI after opening function.f90 file

- If two or more functions are uncommented then program will calculate the roots for the last uncommented function. So user need to be careful while commenting and uncommenting the function.

- Click on the 'Run' or 'Build and Run' button which can be found on the menubar which is located on the top of the window as shown in Figure-17.

Figure 17: Run the program to find roots of the function

- **Output visualization** – After successful execution of the program user can see the output of the calculated roots of the function. As in our example we have uncommented the trigonometric function so the roots of the trigonometric function can be seen in consol as shown in Figure-18.

```
**********************************************************************
    INPUT data / Corrected INPUT data
**********************************************************************
Maximum number roots to be found...................:   30
Lower bound of search interval.....................:   -4.0000
Upper bound of search interval.....................:    4.0000
Step width for the starting position x0............:    0.0500
Precision for the newton algorithm.................:    0.1000E-06
Step width for derivative..........................:    0.0100
Maximum number of iteration for one root search...:  100
**********************************************************************



**********************************************************************
     Solution found
**********************************************************************
*****************************
Number of roots found:  15
*****************************
 No.          Roots(x)
*****************************
  1          -3.9787
  2          -3.7489
  3          -3.1416
  4          -1.3930
  5          -0.0001
  6          -0.0001
  7          -0.0000
  8          -0.0000
  9           0.0000
 10           0.0001
 11           0.0001
 12           1.3930
 13           3.1416
 14           3.7489
 15           3.9787
*****************************

Process returned 0 (0x0)   execution time : 0.144 s
Press any key to continue.
```

Figure 18: Roots of the trigonometric function

- User can also access the generated 'newtonlogfile.txt' from project folder 'Project-1-Newtonmethod' which includes the used input parameter values in input file and calculated roots of the function.

```
     INPUT data / Corrected INPUT data
**********************************************************************
Maximum number roots to be found..................:   30
Lower bound of search interval....................:   -4.0000
Upper bound of search interval....................:    4.0000
Step width for the starting position x0...........:    0.0500
Precision for the newton algorithm................:    0.1000E-06
Step width for derivative.........................:    0.0100
Maximum number of iteration for one root search...:  100
**********************************************************************


**********************************************************************
     Solution found
**********************************************************************
******************************
Number of roots found:  15
******************************
 No.          Roots(x)
******************************
  1          -3.9787
  2          -3.7489
  3          -3.1416
  4          -1.3930
  5          -0.0001
  6          -0.0001
  7          -0.0000
  8          -0.0000
  9           0.0000
 10           0.0001
 11           0.0001
 12           1.3930
 13           3.1416
 14           3.7489
 15           3.9787
******************************
```

Figure 19: Output file of calculated Newton's Root for Trigonometric function

2.5 Description of Input file

The program should read the input parameter from the input file for successful execution of the program. The format of the input file used in 'ScanForRoots.inp' file is given as below.

- **max. number of roots to find**
- **[lower bound: lb] [upper bound: ub] [step width: sw]**
- **[precision eps] [stepwidth h] [max iteration: ix]**

The first line describes the maximum number of roots to find for the input function. The second line consist of upper and lower bound of the search interval in which the roots should be found and step width for the newton's algorithm. The third line describes the precision for the roots need to be found, step width for the derivative and maximum number of iteration for one root search. If there is any error present in input file then program will try to correct and if the correction is not possible then it will show an error message. If sequence is also changed then the program will give incorrect solution or errors. In this project the values of the input parameters which are used in input file is shown in Figure-20.

ScanForRoots.inp

```
30              ! Maximum number of roots to be found
-4.0 4.0 0.05   ! Lower bound / Upper Bound / Step width for newton algorithm
1.e-7 0.01 100 ! Precision / Step width for derivative / Maximum number of iteration for one root search
```

Figure 20: Input file containing input parameters

2.6 Result and Discussion

This section describes the results of calculated roots of the given functions. There are three different functions used in the developed program viz. trigonometric function, polynomial function and exponential function. The results of these functions are discussed and also validated by comparing with online graphing platform called **Desmos**.

Input parameters	Value
Maximum number of roots to be found	30
Lower bound of the search interval	-4
Upper bound of the search interval	4
Step width of newton algorithm	0.05
Precision	$1.e^{-7}$
Step width for derivative	0.01
Maximum number of iteration for one root search	100

Table 1: Input parameters used to find the roots of the function

2.6.1 Result of trigonometric function

The roots of the following trigonometric function is found using the input parameters discussed in Table-1

$$f_1(x) = \frac{2x - 8sin(2x)}{1 - x^2}.tan(x) \tag{10}$$

The output of the program can be seen in Figure-21

```
*****************************************************************
    INPUT data / Corrected INPUT data
*****************************************************************
Maximum number roots to be found...................:   30
Lower bound of search interval.....................:   -4.0000
Upper bound of search interval.....................:    4.0000
Step width for the starting position x0............:    0.0500
Precision for the newton algorithm.................:    0.1000E-06
Step width for derivative..........................:    0.0100
Maximum number of iteration for one root search...:  100
*****************************************************************



*****************************************************************
     Solution found
*****************************************************************
******************************
Number of roots found:  15
******************************
 No.          Roots(x)
******************************
  1          -3.9787
  2          -3.7489
  3          -3.1416
  4          -1.3930
  5          -0.0001
  6          -0.0001
  7          -0.0000
  8          -0.0000
  9           0.0000
 10           0.0001
 11           0.0001
 12           1.3930
 13           3.1416
 14           3.7489
 15           3.9787
******************************

Process returned 0 (0x0)   execution time : 0.144 s
Press any key to continue.
```

Figure 21: Roots of the trigonometric function in Consol

The function is plotted using the online graphing platform called **Desmos** as shown in Figure-22.

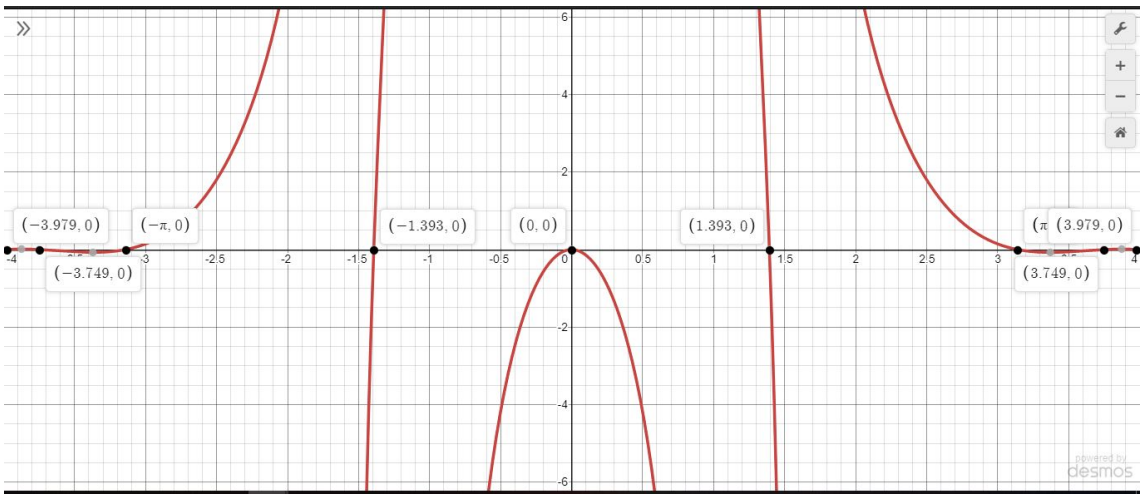


Figure 22: Roots of the trigonometric function using Desmos

The roots found are as follows –

No.	Roots(x)
1	-3.979
2	-3.749
3	-3.141
4	-1.393
5	0
6	1.393
7	3.141
8	3.749
9	3.979

Table 2: Roots of trigonometric function using Desmos

It can be seen that the roots found using developed code matches with the graphically displayed roots. However, the developed code shows 7 roots around '0', whereas, the graph shows 1 root only. This is because of the way in which the curve approaches '0' and the graph does not have the exact precision as the code to capture the roots. Hence, the code displays 15 roots (7 roots around '0'), whereas the graph shows 9 roots(only one root at '0').

2.6.2 Result of polynomial function

The roots of the following polynomial function is found using the input parameters discussed in Table-1

$$f_2(x) = \frac{x^6}{7} + \frac{x^5}{2} - 2x^4 + 2x^2 - 10x - 3 \tag{11}$$

The output of the program can be seen in Figure-23

```
*****************************************************************
    INPUT data / Corrected INPUT data
*****************************************************************
Maximum number roots to be found.................:   30
Lower bound of search interval...................:   -4.0000
Upper bound of search interval...................:    4.0000
Step width for the starting position x0..........:    0.0500
Precision for the newton algorithm...............:    0.1000E-06
Step width for derivative........................:    0.0100
Maximum number of iteration for one root search...:  100
*****************************************************************



*****************************************************************
     Solution found
*****************************************************************
*****************************
Number of roots found:   3
*****************************
 No.          Roots(x)
*****************************
  1          -1.6811
  2          -0.2851
  3           2.6415
*****************************

Process returned 0 (0x0)   execution time : 0.230 s
Press any key to continue.
```

Figure 23: Roots of the polynomial function in consol

The function is plotted using the online graphing platform called **Desmos** as shown in Figure-24.

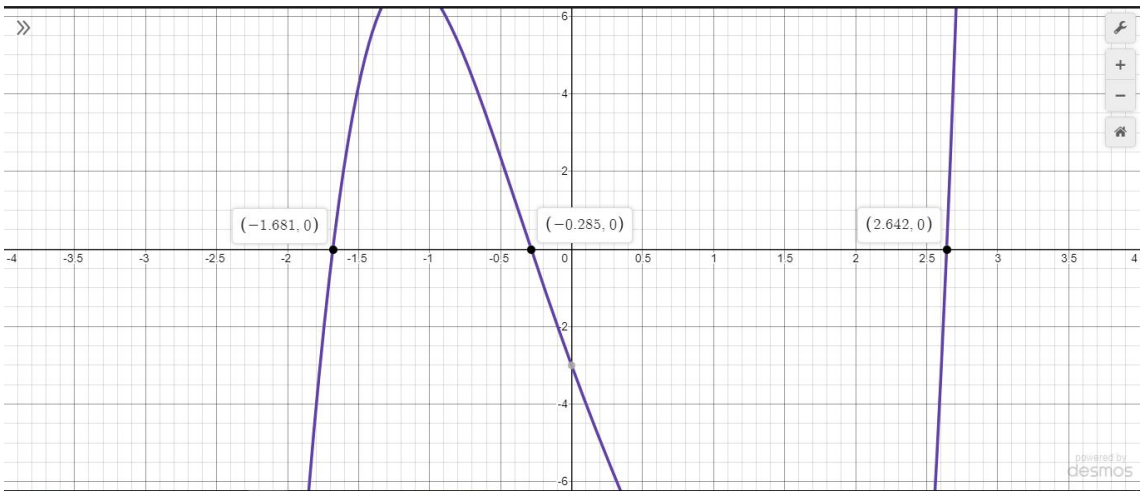


Figure 24: Roots of the polynomial function using Desmos

The roots found are as follows –

No.	Roots(x)
1	-1.681
2	-0.285
3	-2.642

Table 3: Roots of polynomial function using Desmos

It can be seen that the roots found using developed code matches with the graphically displayed roots.

2.6.3 Result of exponential function

The roots of the following exponential function is found using the input parameters discussed in Table-1

$$f_3(x) = (1-x)e^{\frac{-x}{2}}(1-2cos(\frac{x}{2})) - (1+x)e^{\frac{x}{2}}(1+2cos(\frac{x}{2})) \tag{12}$$

The output of the program can be seen in Figure-25

```
**********************************************************************
    INPUT data / Corrected INPUT data
**********************************************************************
Maximum number roots to be found.................:   30
Lower bound of search interval...................:   -4.0000
Upper bound of search interval...................:    4.0000
Step width for the starting position x0..........:    0.0500
Precision for the newton algorithm...............:    0.1000E-06
Step width for derivative........................:    0.0100
Maximum number of iteration for one root search...:  100
**********************************************************************



**********************************************************************
     Solution found
**********************************************************************
******************************
Number of roots found:   1
******************************
 No.          Roots(x)
******************************
  1          -1.9828
******************************

Process returned 0 (0x0)   execution time : 0.187 s
Press any key to continue.
```

Figure 25: Roots of the exponential function in consol

The function is plotted using the online graphing platform called **Desmos** as shown in Figure-26.

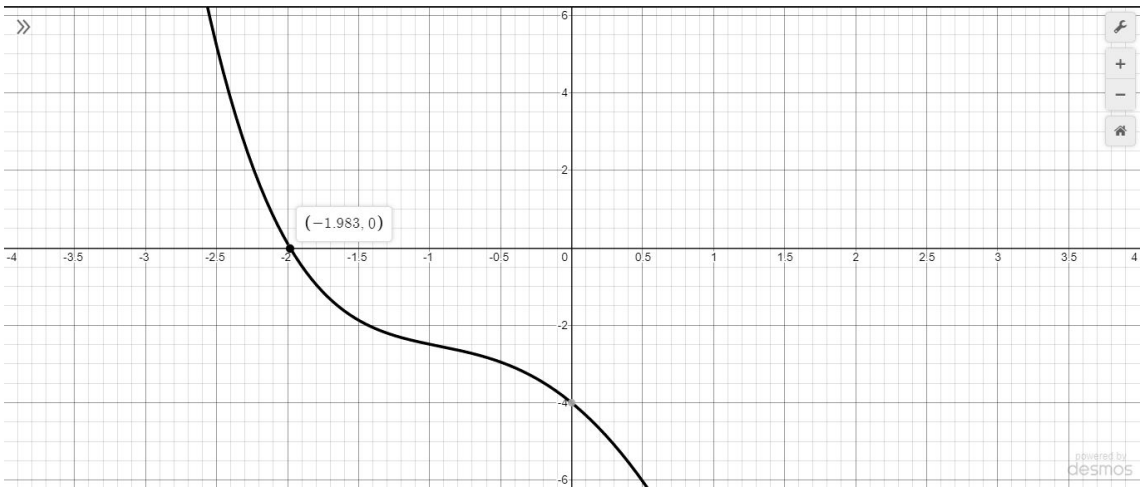


Figure 26: Roots of the exponential function using Desmos

The roots found are as follows –

No.	Roots(x)
1	-1.983

Table 4: Roots of exponential function using Desmos

From the above results it can be seen that the developed program for finding the roots of an arbitrary function using **fortran.f90** provides reliable results as it is validated by comparing with online graph-making tool. Hence, this program can also be used to find the roots of the other function and the input parameter can also be changed as per requirement in input file which is explained in section-2.5.

3 Project 2 – Implementation of Matrix Multiplication in Fortran90

Many Engineering applications, viz. FEA, FEM, etc., involves solving numerical problems. These problems can be generally expressed as linear equation systems of the form,

$$\mathbf{Ax} = \mathbf{b} \tag{13}$$

$\mathbf{A}\in R^{n\times n}$ being invertible.

Like in case of solving numerous equations with 2 or more variables in space and/or time could be solved easily using Matrices. Formulating and performing mathematical operations between matrices hence, become essential and crucial. The most exacting of all Matrices operations are Multiplying and finding an Inverse of a Matrix.

In this Project, a Fortran90 program is written that would perform multiplication of two or more matrices or arbitrary size.

3.1 Problem definition

Nevertheless, the prime aspect of the project is to code a Program that should be able to handle an arbitrary number of dynamically allocatable Matrices of various sizes, the task entrusted is merely to multiply 5 matrices of predefined sizes as follows.

$$\mathbf{M}^1_{[4\times 8]} \times \mathbf{M}^2_{[8\times 7]} \times \mathbf{M}^3_{[7\times 2]} \times \mathbf{M}^4_{[2\times 5]} \times \mathbf{M}^5_{[5\times 4]} = \mathbf{M}^R_{[4\times 4]} \tag{14}$$

3.2 Matrix multiplication

Matrix Multiplication follow the rule that the number of columns of the right matrix shall be equal to the number of rows of the left matrix operated under multiplication. Mathematically it can be expressed as follows.

Let $\mathbf{A}\in R^{i\times j}$ and $\mathbf{B}\in R^{j\times k}$ such that

$$\mathbf{A}_{[i\times j]} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1j} \\ A_{21} & A_{22} & \cdots & A_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ A_{i1} & A_{i2} & \cdots & A_{ij} \end{bmatrix} \tag{15}$$

and

$$\mathbf{B}_{[j\times k]} = \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1k} \\ B_{21} & B_{22} & \cdots & B_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ B_{j1} & B_{j2} & \cdots & B_{jk} \end{bmatrix} \tag{16}$$

The Product Matrix that the multiplication of A and B will yield can be expressed as;

$$P_{ik} = \mathbf{P} = \mathbf{A} \times \mathbf{B} = \sum_{s=1}^{j} A_{is} B_{sk} \tag{17}$$

for $r\in[1,i], t\in[1,k]$ and s being the summation index

Hence, it can be put up for the given example

$$\begin{aligned}
\mathbf{M}^1_{[4\times8]} \times \mathbf{M}^2_{[8\times7]} &= \mathbf{P}^1_{[4\times7]} \\
\mathbf{P}^1_{[4\times7]} \times \mathbf{M}^3_{[7\times2]} &= \mathbf{P}^2_{[4\times2]} \\
\mathbf{P}^2_{[4\times2]} \times \mathbf{M}^4_{[2\times5]} &= \mathbf{P}^3_{[4\times5]} \\
\mathbf{P}^3_{[4\times5]} \times \mathbf{M}^5_{[5\times4]} &= \mathbf{P}_{[4\times4]}
\end{aligned}$$

3.3 Programmer's guide

This section provides the information of the developed **Fortran** program code for matrix multiplication. It consist of main program **MatMult90.f90** and a module **matmultlib.f90**.

3.3.1 Implementation of matmultlib.f90

The file **matmultlinb.f90** consist the functions namely **iReadMatDim**, **iReadNumMat**, **iReadMat**, **iMatMult** and one subroutine namely **listMat**.

- **iReadNumMat()**: The function $iReadNumMat$(channel, number of matrix) reads the line containing single numerical values from the input file . This single number indicates the number of matrix need to be multiplied. If number of the matrix is less than 2 then it returns an error because the multiplication of the single matrix is not possible. If it does not encounter any error then $iReadNumMat$ is assigned 0. The flow diagram of function $iReadNumMat$ is shown in Figure-27

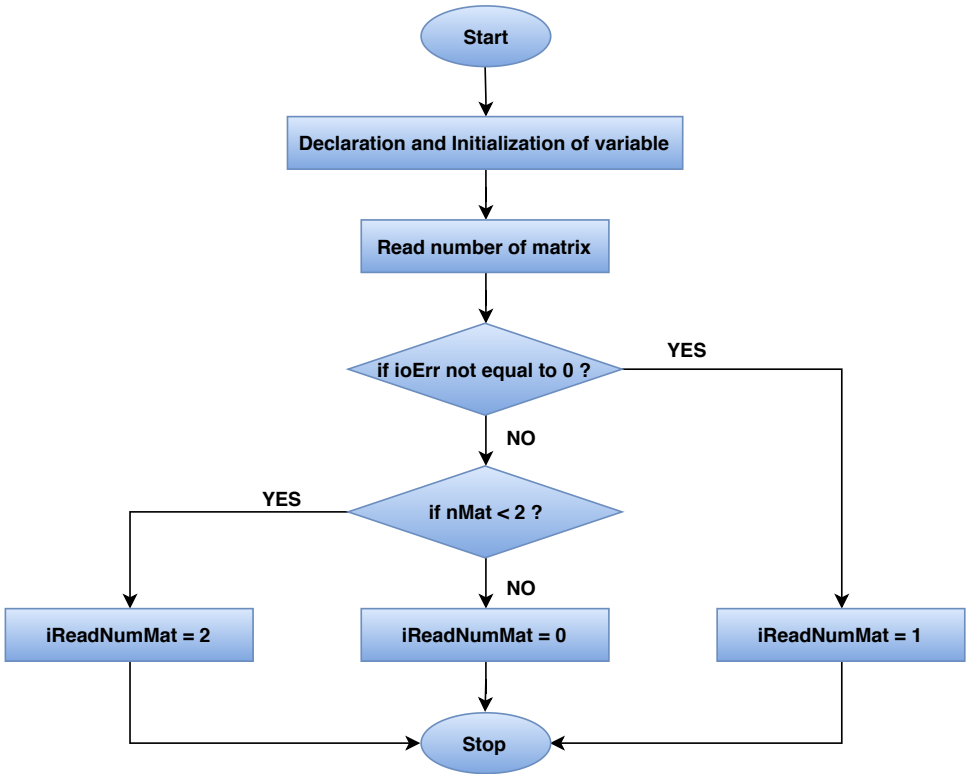


Figure 27: Flow diagram of iReadNumMat

- **iReadMatDim()**: The function $iReadMatDim$(channel,dimensions) reads the line containing two numerical values from the input file . These two numbers are the dimensions of the matrix means number of rows and number of columns. If any dimension of the matrix is less than 1 then it returns an error. If it does not encounter any error then $iReadMatDim$ is assigned 0. The flow diagram of function $iReadMatDim$ is shown in Figure-28

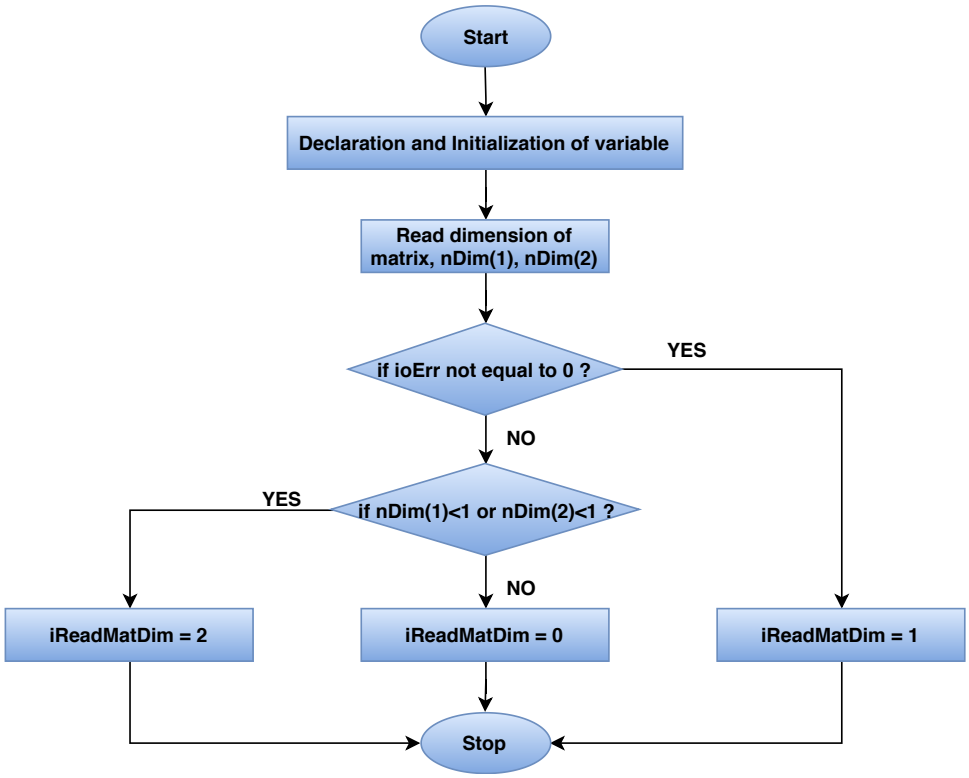


Figure 28: Flow diagram of iReadMatDim

- **iReadMat()**: The function $iReadMat$(channel,matrix dimension,matrix) reads the dimension of the matrix and entries of matrix row wise from the input file 'anupaminput.in'. The flow chart of function $iReadMat$ is shown in Figure-29

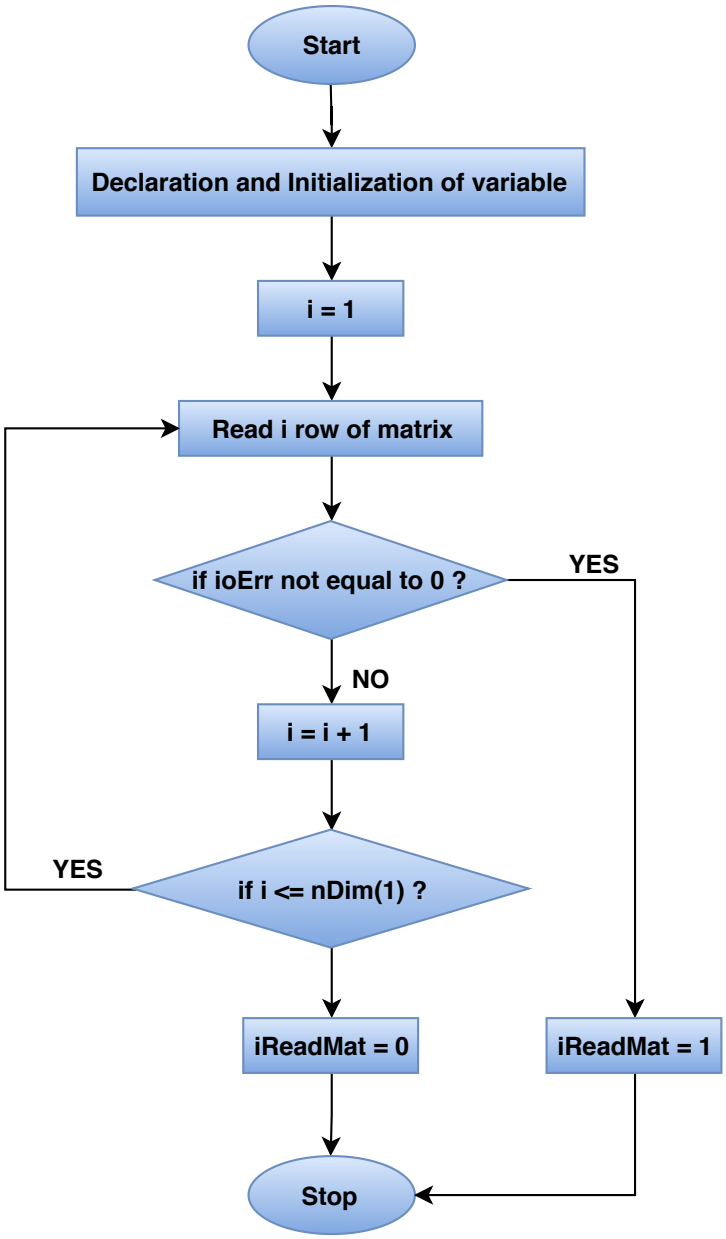


Figure 29: Flow diagram of iReadMat

- **lisMat()**: The subroutine *listMat*(channel, title, dimension of matrix, matrix data, number of matrix) list the calculated matrix entries in to the output file 'anupamoutput.out'. The flow chart of function *listMat* is shown in Figure-30

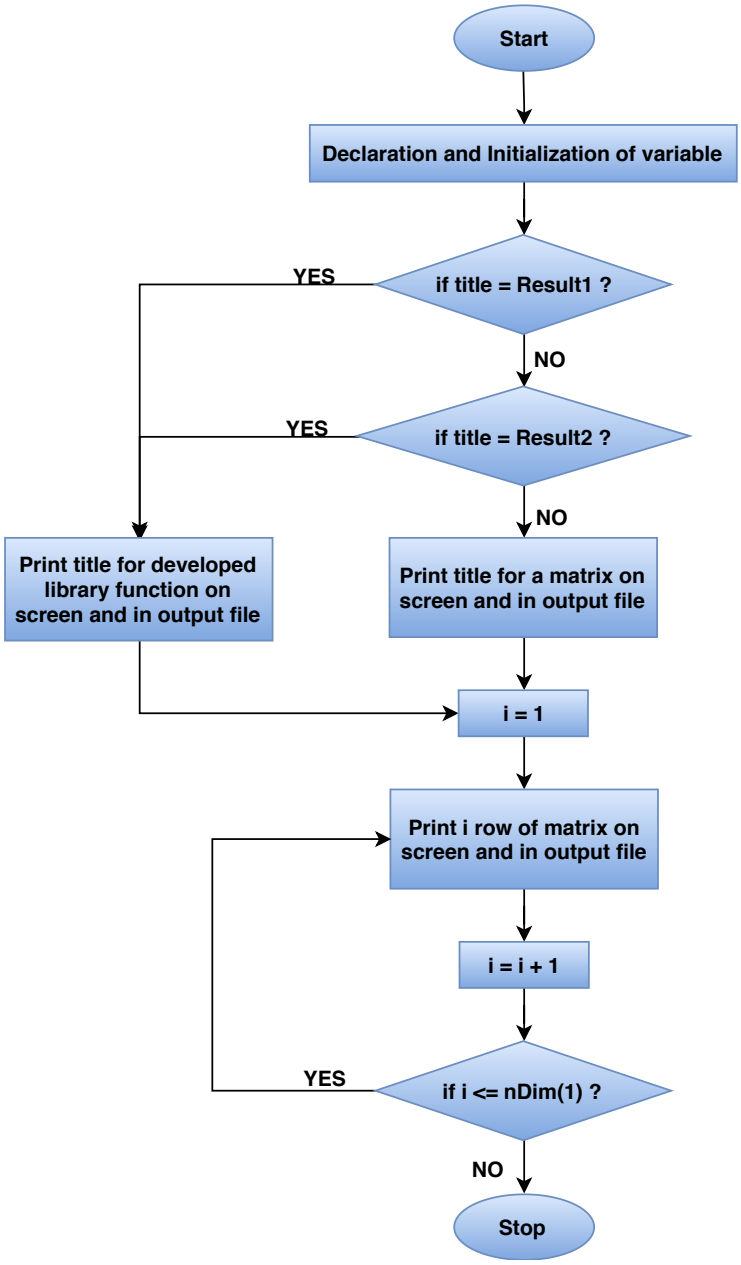


Figure 30: Flow diagram of listMat

- **iMatMult()**: The function $iMatMult$(matrix1, matrix2, resultant matrix, dimension of matrix1, dimension of matrix2) multiplies the matrices. Initially it will check the dimension agreement of two matrices. If the dimension does not match then it assigns greater than 0 number to $iMatMult$ else it assigns 0. The flow chart of function $iMatMult$ is shown in Figure-31

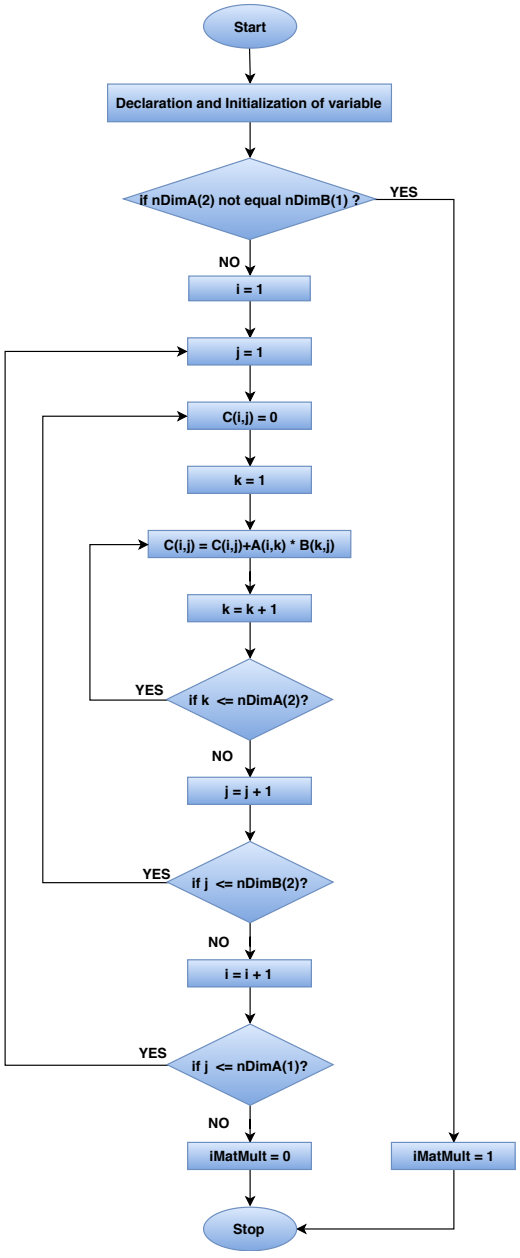


Figure 31: Flow diagram of iMatMult

3.3.2 Implementation of MatMult.f90

The file **MatMult90.f90** consists of initialization and declaration of variables. It also includes the name of the input and output file. The functions **iReadMatDim**, **iReadNumMat**, **iReadMat**, **iMatMult** are called which are discussed in **matmultlib** module.

The program will check the existence of input file and output file and error in these files. If file do not exist then it returns 'file not found' and if there is any error found in these files then it returns message of error found and program will be stopped. The program will also write the name of the input and output files on the screen.

If there are no errors in these files then program reads the number of matrix need to be multiplied using the function **iReadNumMat**. If there is an error in reading the number of matrix then it will returns message of error and program will be terminated. If there is no error in reading number of matrix then program will read the dimensions of the matrix using function **iReadMatDim**.

If there is no error then program will write the dimension of the matrix on the screen and also in output file. The array of the matrix are also allocated else returns error message. Further program will read the entries of matrix and write the matrix in output file using subroutine **listmat**. Once again program will check the inner dimension of matrix for the multiplication and if there is no error then multiplication of the matrix is calculated using the function **iMatMult**. The array for the resultant matrix is also allocated. After first successful multiplication, the next matrix is multiplied with the resultant matrix of the first multiplication. This procedure is continued until the last matrix and the resultant product of the all matrices are written on the screen and also in output file.

3.4 User Manual

This section provides the instructions to use the developed program for multiplication of arbitrary number of matrices. The user should follow the steps as described here.

- Open IDE **CodeBlocks** from start menu of your computer. User should open the project file **Matrixmultiplication.cbp** from project folder 'Project-2-Matrixmultiplication'. The user can see the window as shown in Figure-32

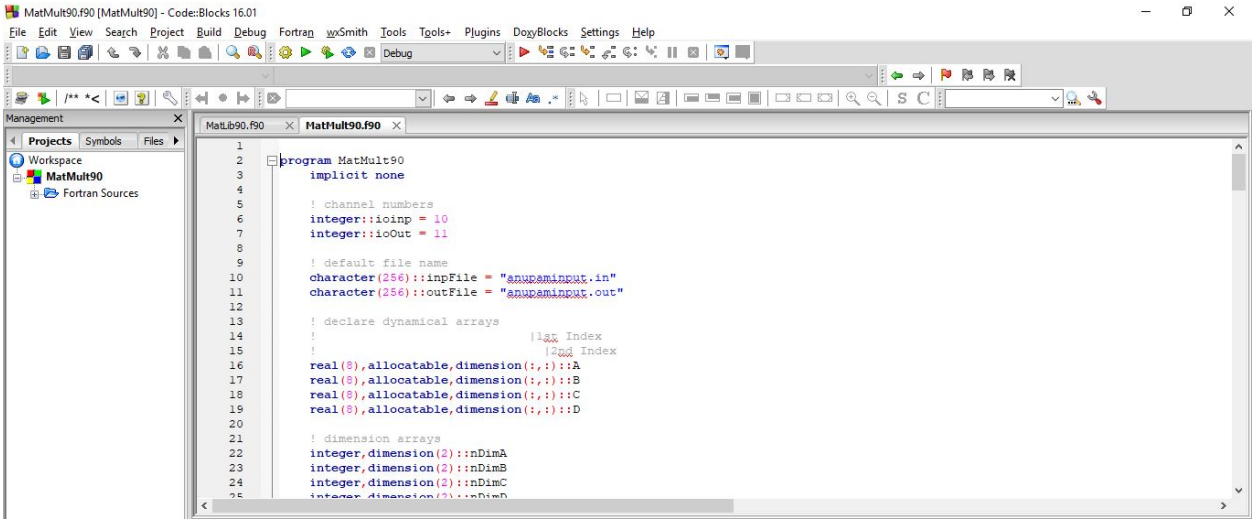


Figure 32: MatMult90 window in CodeBlocks IDE

- Open input file **anupaminput.in** from project folder 'Project-2-Matrixmultiplication. User should write the data of matrices viz. matrix dimensions and values of the matrices in input file as follows:

anupaminput.in

```
5                                   ! Number of matrices
4 8                                 ! Dimension of 1st matrix
1 2 5 6 7 3 8 1                     !
5 6 7 4 5 8 2 4                     ! 1st matrix elements
5 5 6 6 7 7 8 8                     !
8 9 6 4 2 1 7 3                     !

8 7                                 ! Dimension of 2nd matrix
0.1 2.1 1.1 2.3 1.3 0.1 0.7         !
1.1 0.3 1.4 1.8 2.1 0.8 0.4         !
2.7 0.9 1.4 2.4 3.1 4.2 0.4         !
1.2 0.8 1.7 2.6 2.8 1.2 1.9         ! 2nd matrix elements
3.1 4.1 0.3 0.7 1.1 2.8 1.7         !
1.3 2.1 4.1 0.9 0.4 0.3 0.7         !
4.5 2.1 0.4 0.7 0.2 0.1 2.2         !
2.3 4.7 3.8 2.6 1.8 0.6 0.3         !

7 2                                 ! Dimension of 3rd matrix
1 5                                 !
5 9                                 !
6 8                                 !
1 3                                 ! 3rd matrix elements
3 9                                 !
1 2                                 !
2 7                                 !
```

Figure 33: Input file of Matrix Multiplication

- To Run the program and to get the output of the matrix multiplication, user should click on the 'Run' or 'Build and Run' button from the toolbar as shown in Figure-34.

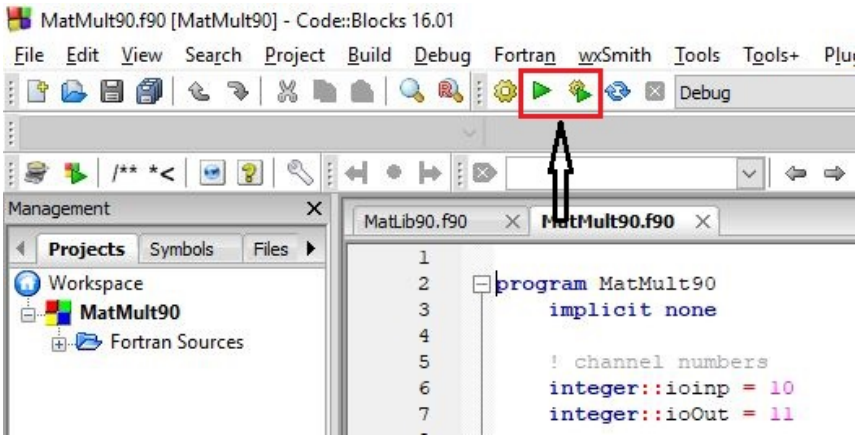


Figure 34: 'Run' and 'Build and Run' button on tool bar

- After successful execution of the program user can see the the result of matrix multiplication in command prompt as shown in Figure-35 and 36.

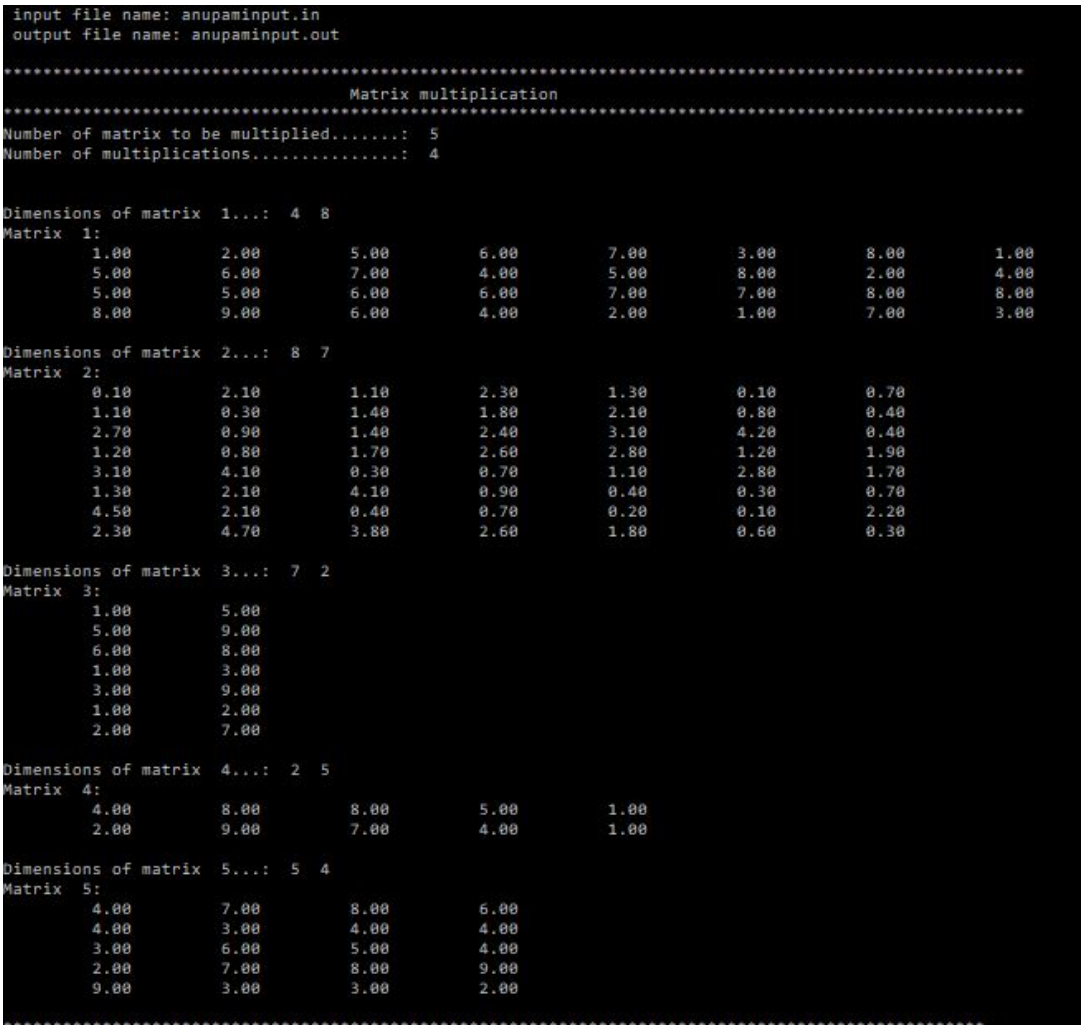


Figure 35: Matrix Multiplication output window Part-1

```
**********************************************************************************************************
                                    Resultant Product of Matrices
**********************************************************************************************************

Dimensions of resultant matrix...:  4  4

Result of Matrix multiplication by developed library function:
    292197.40    418051.20    446683.80    414963.00
    368040.60    527491.80    563566.20    523359.00
    483605.80    692879.40    740278.60    687513.00
    321854.10    460690.80    492231.70    457234.50

Result of Matrix multiplication by intrinsic function (matmul):
    292197.40    418051.20    446683.80    414963.00
    368040.60    527491.80    563566.20    523359.00
    483605.80    692879.40    740278.60    687513.00
    321854.10    460690.80    492231.70    457234.50

Process returned 0 (0x0)   execution time : 0.094 s
Press any key to continue.
```

Figure 36: Matrix Multiplication output window Part-2

- User can also access the generated 'anupamoutput.out' file from project folder which include the entered data of the matrices with its dimensions and also the resultant matrix product and resultant dimension of the matrix as shown in output file as follows:

```
1
2   **********************************************************************************************************
3                                       Matrix multiplication
4   **********************************************************************************************************
5   Number of matrix to be multiplied.......:  5
6   Number of multiplications...............:  4
7
8
9   Dimensions of matrix  1...:  4  8
10  Matrix  1:
11              1.00          2.00          5.00          6.00          7.00          3.00          8.00          1.00
12              5.00          6.00          7.00          4.00          5.00          8.00          2.00          4.00
13              5.00          5.00          6.00          6.00          7.00          7.00          8.00          8.00
14              8.00          9.00          6.00          4.00          2.00          1.00          7.00          3.00
15
16  Dimensions of matrix  2...:  8  7
17  Matrix  2:
18              0.10          2.10          1.10          2.30          1.30          0.10          0.70
19              1.10          0.30          1.40          1.80          2.10          0.80          0.40
20              2.70          0.90          1.40          2.40          3.10          4.20          0.40
21              1.20          0.80          1.70          2.60          2.80          1.20          1.90
22              3.10          4.10          0.30          0.70          1.10          2.80          1.70
23              1.30          2.10          4.10          0.90          0.40          0.30          0.70
24              4.50          2.10          0.40          0.70          0.20          0.10          2.20
25              2.30          4.70          3.80          2.60          1.80          0.60          0.30
26
27  Dimensions of matrix  3...:  7  2
28  Matrix  3:
29              1.00          5.00
30              5.00          9.00
31              6.00          8.00
32              1.00          3.00
33              3.00          9.00
34              1.00          2.00
35              2.00          7.00
36
37  Dimensions of matrix  4...:  2  5
38  Matrix  4:
```

Figure 37: Matrix Multiplication output file Part-1

```
Matrix  4:
        4.00         8.00         8.00         5.00         1.00
        2.00         9.00         7.00         4.00         1.00

Dimensions of matrix  5...:  5  4
Matrix  5:
        4.00         7.00         8.00         6.00
        4.00         3.00         4.00         4.00
        3.00         6.00         5.00         4.00
        2.00         7.00         8.00         9.00
        9.00         3.00         3.00         2.00

*****************************************************************************************************
                                  Resultant Product of Matrices
*****************************************************************************************************

Dimensions of resultant matrix...:  4  4

Result of Matrix multiplication by developed library function:
    292197.40    418051.20    446683.80    414963.00
    368040.60    527491.80    563566.20    523359.00
    483605.80    692879.40    740278.60    687513.00
    321854.10    460690.80    492231.70    457234.50

Result of Matrix multiplication by intrinsic function (matmul):
    292197.40    418051.20    446683.80    414963.00
    368040.60    527491.80    563566.20    523359.00
    483605.80    692879.40    740278.60    687513.00
    321854.10    460690.80    492231.70    457234.50
```

Figure 38: Matrix Multiplication output file Part-2

3.5 Result and Discussion

It is essential to verify the reliability of the program. The commercial software Matlab has been used to validate the developed Fortran program. The same input data of matrix are taken which is discussed in previous section.

The result of matrix multiplication using Fortran can be seen in section-3.5.1. It can be observed that the input data of the 'anupaminput.in' file is successfully read by the developed program. The input file for matrix multiplication using Matlab can be seen in section-3.5.2. The result of matrix multiplication using Matlab can be seen in section-3.5.3.

The resultant product of matrices obtained by **Fortran** and **Matlab** are same. Hence, it can be concluded that the developed program for the multiplication of arbitrary number of matrices is reliable.

3.5.1 Result of Matrix multiplication using Fortran

```
 input file name: anupaminput.in
 output file name: anupaminput.out

*****************************************************************************************************
                                Matrix multiplication
*****************************************************************************************************
Number of matrix to be multiplied.......:  5
Number of multiplications...............:  4


Dimensions of matrix  1...:  4  8
Matrix  1:
        1.00         2.00         5.00         6.00         7.00         3.00         8.00         1.00
        5.00         6.00         7.00         4.00         5.00         8.00         2.00         4.00
        5.00         5.00         6.00         6.00         7.00         7.00         8.00         8.00
        8.00         9.00         6.00         4.00         2.00         1.00         7.00         3.00

Dimensions of matrix  2...:  8  7
Matrix  2:
        0.10         2.10         1.10         2.30         1.30         0.10         0.70
        1.10         0.30         1.40         1.80         2.10         0.80         0.40
        2.70         0.90         1.40         2.40         3.10         4.20         0.40
        1.20         0.80         1.70         2.60         2.80         1.20         1.90
        3.10         4.10         0.30         0.70         1.10         2.80         1.70
        1.30         2.10         4.10         0.90         0.40         0.30         0.70
        4.50         2.10         0.40         0.70         0.20         0.10         2.20
        2.30         4.70         3.80         2.60         1.80         0.60         0.30

Dimensions of matrix  3...:  7  2
Matrix  3:
        1.00         5.00
        5.00         9.00
        6.00         8.00
        1.00         3.00
        3.00         9.00
        1.00         2.00
        2.00         7.00

Dimensions of matrix  4...:  2  5
Matrix  4:
        4.00         8.00         8.00         5.00         1.00
        2.00         9.00         7.00         4.00         1.00

Dimensions of matrix  5...:  5  4
Matrix  5:
        4.00         7.00         8.00         6.00
        4.00         3.00         4.00         4.00
        3.00         6.00         5.00         4.00
        2.00         7.00         8.00         9.00
        9.00         3.00         3.00         2.00
```

Figure 39: Matrix Multiplication output consol window Part-1 using Fortran

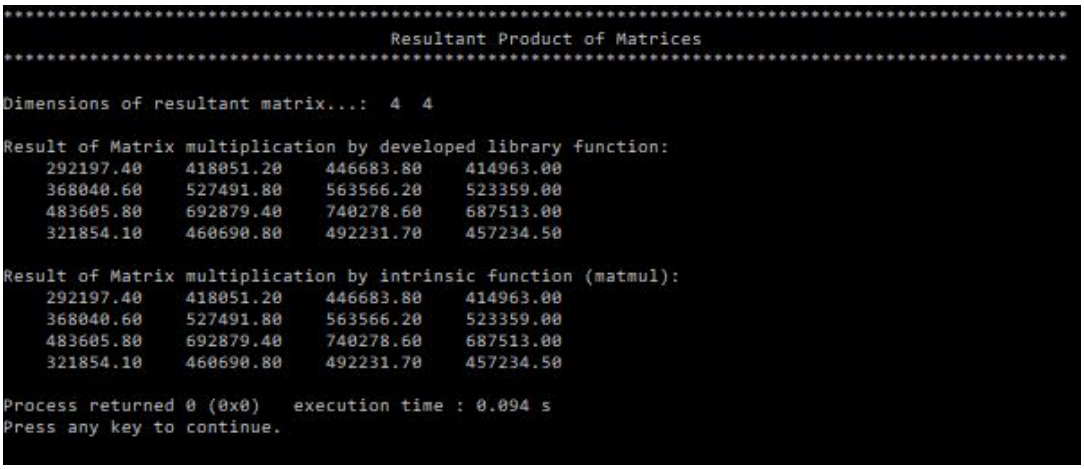


Figure 40: Matrix Multiplication output consol window Part-2 using Fortran

3.5.2 Input Matlab file of Matrix Multiplication

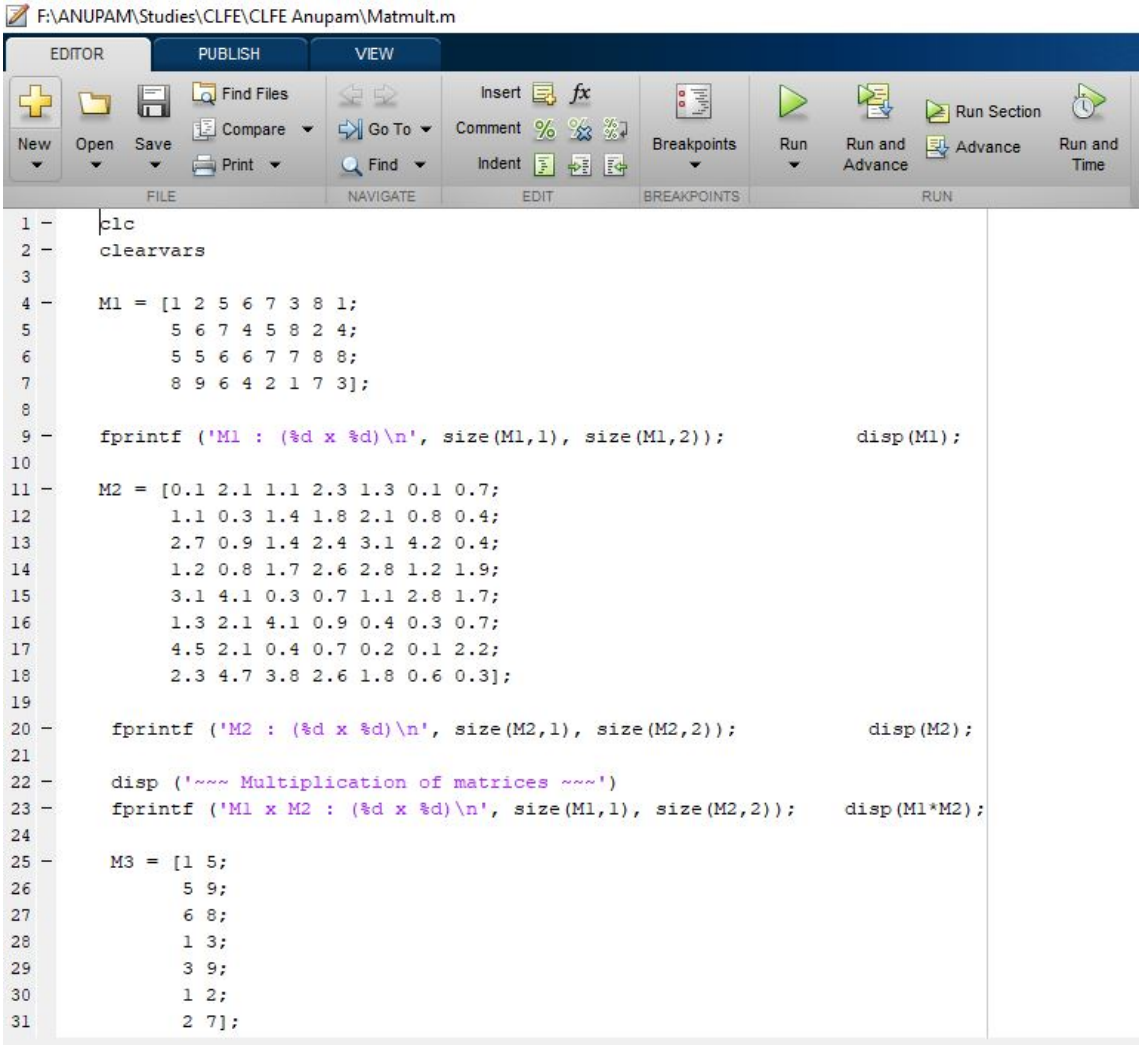


Figure 41: Input Matlab file of Matrix Multiplication - 1

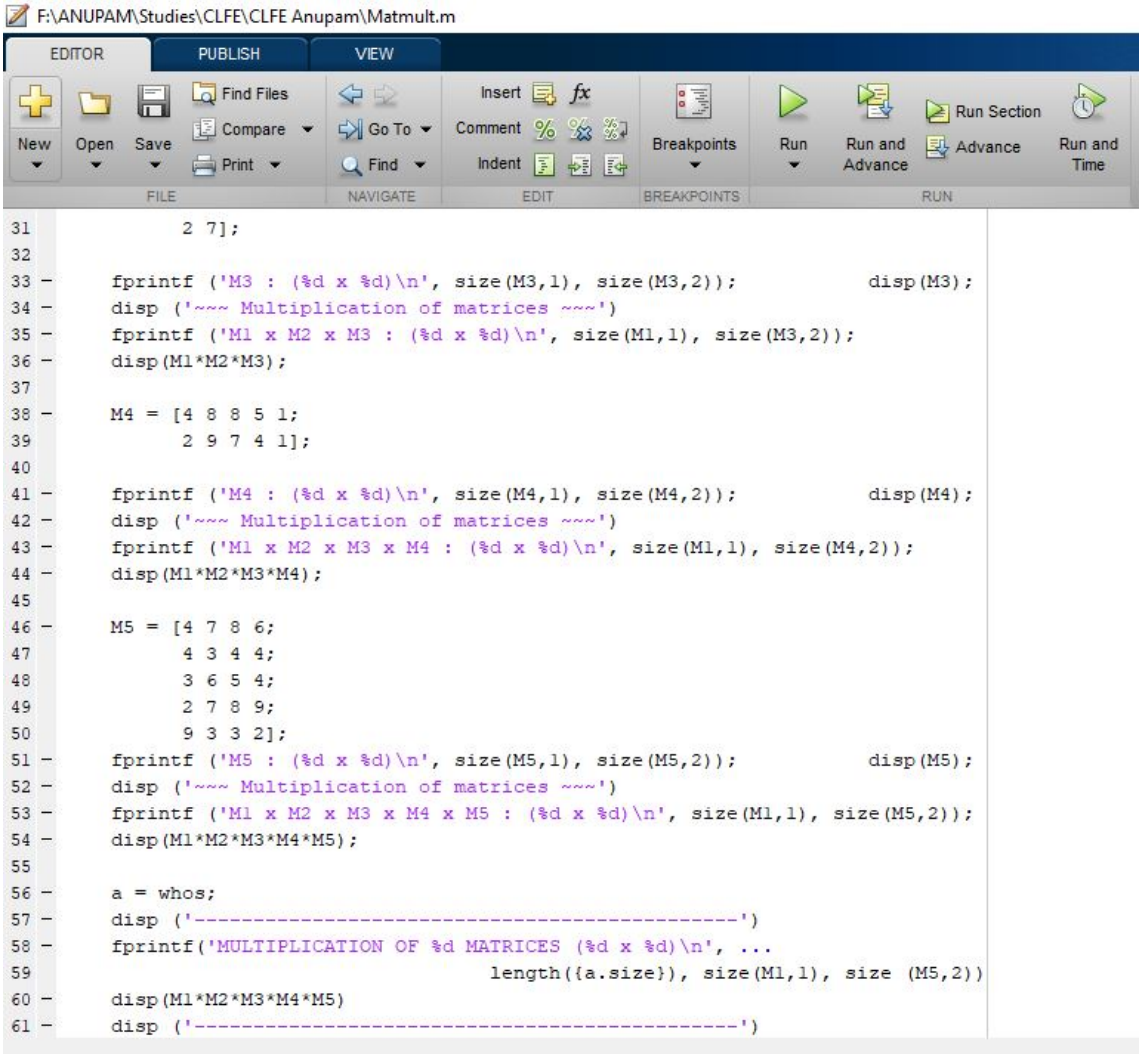


Figure 42: Input Matlab file of Matrix Multiplication - 2

3.5.3 Output Matlab file of Matrix Multiplication

Command Window

```
M1 : (4 x 8)
     1     2     5     6     7     3     8     1
     5     6     7     4     5     8     2     4
     5     5     6     6     7     7     8     8
     8     9     6     4     2     1     7     3

M2 : (8 x 7)
    0.1000    2.1000    1.1000    2.3000    1.3000    0.1000    0.7000
    1.1000    0.3000    1.4000    1.8000    2.1000    0.8000    0.4000
    2.7000    0.9000    1.4000    2.4000    3.1000    4.2000    0.4000
    1.2000    0.8000    1.7000    2.6000    2.8000    1.2000    1.9000
    3.1000    4.1000    0.3000    0.7000    1.1000    2.8000    1.7000
    1.3000    2.1000    4.1000    0.9000    0.4000    0.3000    0.7000
    4.5000    2.1000    0.4000    0.7000    0.2000    0.1000    2.2000
    2.3000    4.7000    3.8000    2.6000    1.8000    0.6000    0.3000

~~~ Multiplication of matrices ~~~
M1 x M2 : (4 x 7)
   86.9000   68.5000   42.5000   49.3000   50.1000   51.8000   46.8000
   74.9000   82.1000   80.8000   72.0000   68.3000   58.5000   36.0000
  114.6000  120.0000   95.5000   88.1000   78.9000   64.2000   56.1000
   77.6000   67.2000   55.5000   74.4000   68.5000   46.4000   39.6000

M3 : (7 x 2)
     1     5
     5     9
     6     8
     1     3
     3     9
     1     2
     2     7

~~~ Multiplication of matrices ~~~
M1 x M2 x M3 : (4 x 2)
   1.0e+03 *
```

Figure 43: Output Matlab file of Matrix Multiplication - 1

Command Window

```
~~~ Multiplication of matrices ~~~
M1 x M2 x M3 : (4 x 2)
   1.0e+03 *

    1.0294    2.4210
    1.3776    2.9595
    1.7888    3.9125
    1.1521    2.6465

M4 : (2 x 5)
     4     8     8     5     1
     2     9     7     4     1

~~~ Multiplication of matrices ~~~
M1 x M2 x M3 x M4 : (4 x 5)
   1.0e+04 *

    0.8960    3.0024    2.5182    1.4831    0.3450
    1.1429    3.7656    3.1737    1.8726    0.4337
    1.4980    4.9523    4.1698    2.4594    0.5701
    0.9901    3.3035    2.7742    1.6346    0.3799

M5 : (5 x 4)
     4     7     8     6
     4     3     4     4
     3     6     5     4
     2     7     8     9
     9     3     3     2

~~~ Multiplication of matrices ~~~
M1 x M2 x M3 x M4 x M5 : (4 x 4)
   1.0e+05 *

    2.9220    4.1805    4.4668    4.1496
    3.6804    5.2749    5.6357    5.2336
    4.8361    6.9288    7.4028    6.8751
    3.2185    4.6069    4.9223    4.5723
```

Figure 44: Output Matlab file of Matrix Multiplication - 2

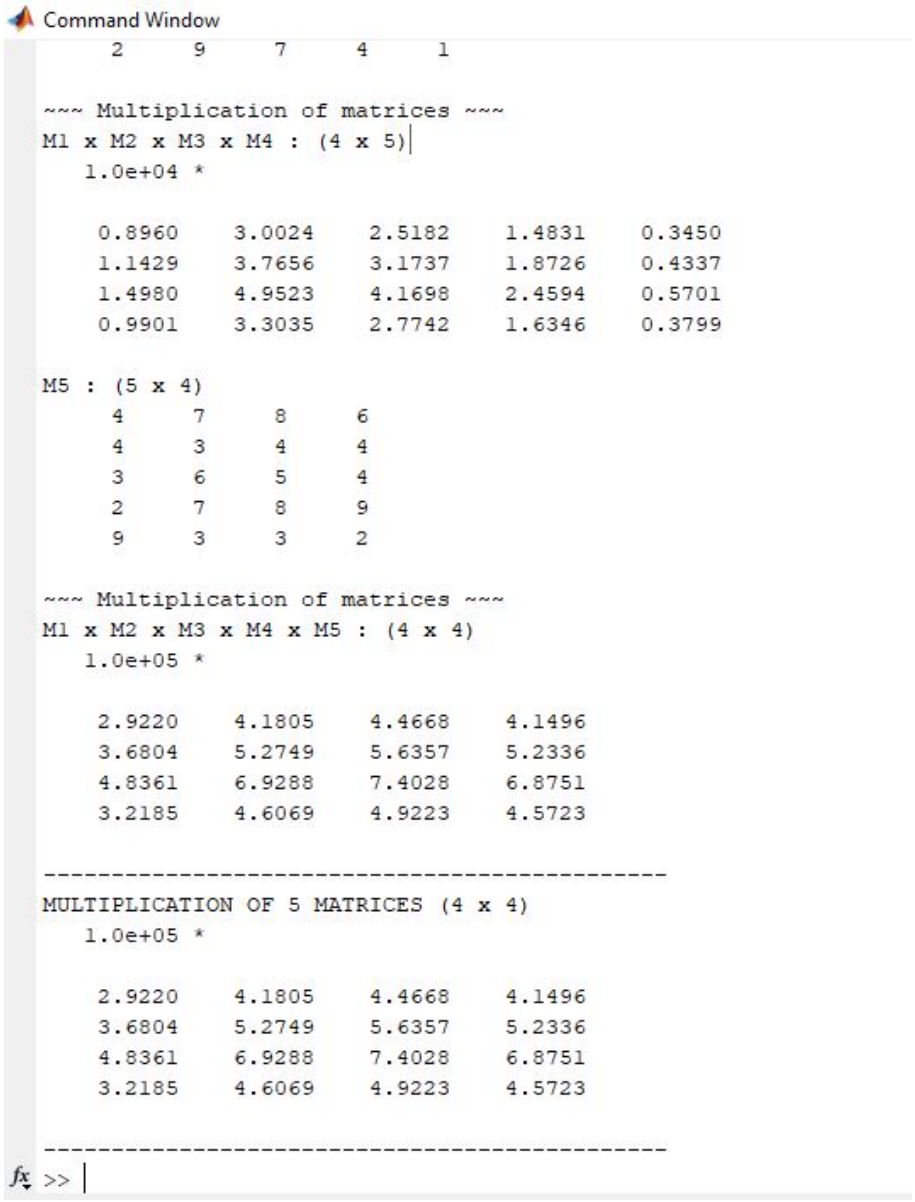


Figure 45: Output Matlab file of Matrix Multiplication - 3

4 Project 3 – Implementation of C++ program to calculate the sectional properties of a combined profile

A C++ program code is written to perform numerical calculations in order to determine – area, center of mass, static moments and moment of inertia of a given combined Profile. The combinations consists of two L-profiles and one O-profile and is symmetric about z-axis.

To achieve this task, various classes are built upon collecting attributes and methods with which objects (or) instances are generated.The combined profile is generated by creating classes like – base, node, element, profile and list. For simplification in programming, thin-walled approach is implemented for the profile.

The aim of the task is –

- To validate the results obtained from the program with analytical results.
- To discuss the errors in the evaluations.

4.1 Problem definition

With an overview of Object-Oriented Programming, it is required to develop a C++ program that calculates the section properties of a combination of several profiles placed juxtaposing each other in the below figure.

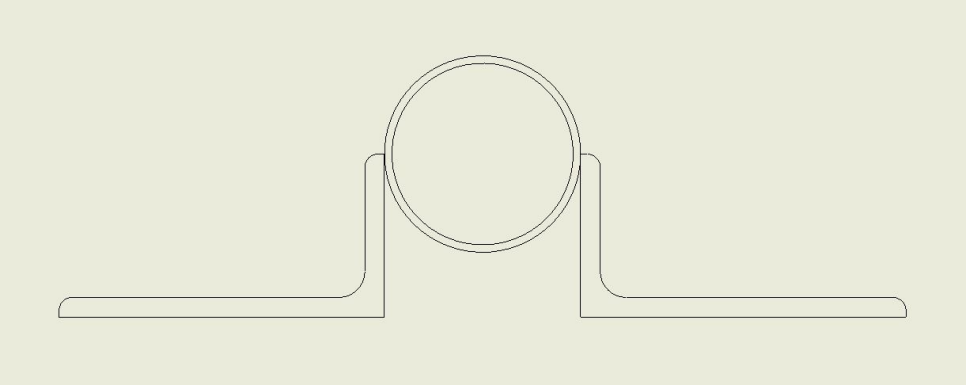


Figure 46: Actual view of the combined profile

The profile comprises of two L-Profiles of varying cross sections which is combined with one O-profile. The following sectional properties are needed to be evaluated with the help of this program –

- Area of combined profile.
- Static moments.
- Centre of mass.
- Moment of inertia.

The dimensional properties of the profiles as per the DIN standards are mentioned in the tables below. 5 and 6 –

Profile	k[mm]	bb[mm]	z[mm]
L 100x50x6	100	50	6
L 100x75x9	100	75	9
L 120x80x8	120	80	8

Table 5: L-profile dimensions

Profile	d[mm]	Tt[mm]
O 60.3x2.3	60.3	2.3

Table 6: O-profile dimensions

4.2 Thin-walled approximation

Thin-walled structures comprise an important and growing proportion of engineering construction with areas of application becoming increasingly diverse ranging from aircraft, bridges, ships and oil rigs to storage vessels, industrial buildings and warehouses.To analyze and calculate the section properties of combined profile, we have used the Thin Walled Approach.The Profile has been approximated as 2-Dimensional line elements having constant thickness throughout the section and fillets across the edges can be neglected. The simplified geometry of combined profile using Thin Walled Approach is shown in the below figure.

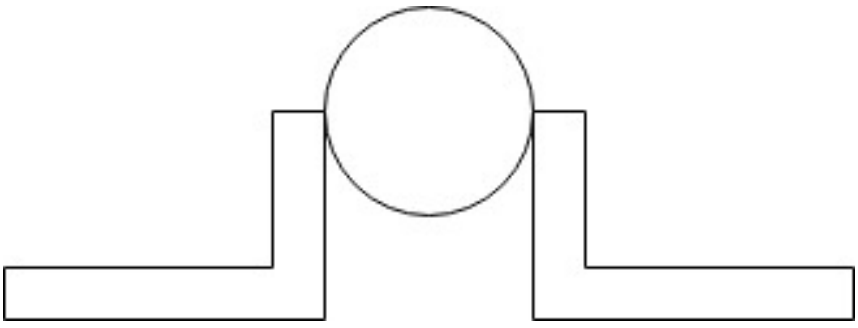


Figure 47: Simplified geometry of the combined profile

The sectional properties required to be approximated as listed above are,

- **Area:** Analytically the area of the profile is an integral function of all the infinitesimally small areas of the elements of the profile discretised for Finite element analysis. The corresponding approximation can be derived using Quadrature Rule,however, for a simplified geometry as depicted in the figure- 47, it is convenient to produce it by the expressions below.

$$A = \int_A e_\mu . dA \approx \sum_{i=1}^{nEle} A_i = \sum_{i=1}^{nEle} e_{\mu,i} . L_i . t_i \tag{18}$$

Where,

- L_i – Length of line i
- t_i – Thickness of line i
- $e_{\mu,i}$ – Relative elasticity of line i (1 for only one material)

- **First moment of an area:** Static or the first moments of an area is given below with its corresponding approximation for sum over individual elemental properties.

$$S_y = \int_A e_\mu.z.dA \approx \sum_{i=1}^{nEle} e_{\mu,i}.\overline{z}_i.A_i \tag{19}$$

$$S_z = \int_A e_\mu.y.dA \approx \sum_{i=1}^{nEle} e_{\mu,i}.\overline{y}_i.A_i \tag{20}$$

Where,

- A_i – Area of line i
- $\overline{y}_i$ – The y coordinate of the center of line i
- $\overline{z}_i$ – The z coordinate of the center of line i

- **Centre of mass:** The Centre of mass for a structure having uniformly distributed mass can be analytically integrated over the area and correspondingly approximated with the summation expression as:

$$y_c = \frac{\int_A y.dA}{\int_A dA} \approx \frac{\sum_{i=1}^{nEle} y_i A_i}{\sum_{i=1}^{nEle} A_i} \tag{21}$$

$$z_c = \frac{\int_A z.dA}{\int_A dA} \approx \frac{\sum_{i=1}^{nEle} z_i A_i}{\sum_{i=1}^{nEle} A_i} \tag{22}$$

- **Second moments of an area or Moment of inertia:** Moment of Inertia of the body is the tendency by virtue of which the body resists rotational motion or continue to maintain its rotational motion. The Area Moment of Inertia or the Second moment of Inertia about the Centre of Mass of a cylindrical body is expressed as:

$$I_y = \int_A e_\mu.z^2.dA \approx \sum_{i=1}^{nEle} e_{\mu,i}.\left(\frac{(z_{b,i} - z_{a,i})^2}{12} + \overline{z}_i^2\right).A_i \tag{23}$$

$$I_z = \int_A e_\mu.y^2.dA \approx \sum_{i=1}^{nEle} e_{\mu,i}.\left(\frac{(y_{b,i} - y_{a,i})^2}{12} + \overline{y}_i^2\right).A_i \tag{24}$$

where,

- A_i – The area of line i
- $\overline{y}_i$ – The y coordinate of the center of line i
- $\overline{z}_i$ – The z coordinate of the center of line i
- $y_{a,i}$ – The y coordinate of the first point of line i
- $z_{a,i}$ – The z coordinate of the first point of line i
- $y_{b,i}$ – The y coordinate of the second point of line i
- $z_{b,i}$ – The z coordinate of the second point of line i

On application of the Parallel axis theorem (Steiner theorem), the Moment of inertia of the body about any arbitrary point at a distance z_i from Y-axis and y_i from Z-Axis is given as,

$$I_y = I_{y,c} + \sum_{i=1}^{nEle} z_i^2.A_i \tag{25}$$

$$I_z = I_{z,c} + \sum_{i=1}^{nEle} y_i^2.A_i \tag{26}$$

4.3 Problem simplification

The actual profile contains complex geometry, like fillets at the edges and corners. The program has been coded to calculate the sectional properties of the rectangular elements and hence, it is required to be simplified. This simplification is undoubtedly sure to render the results imprecise however, accurate, which is further discussed in the section.

The simplified geometry discretised to nodes and elements is depicted in the figure below –

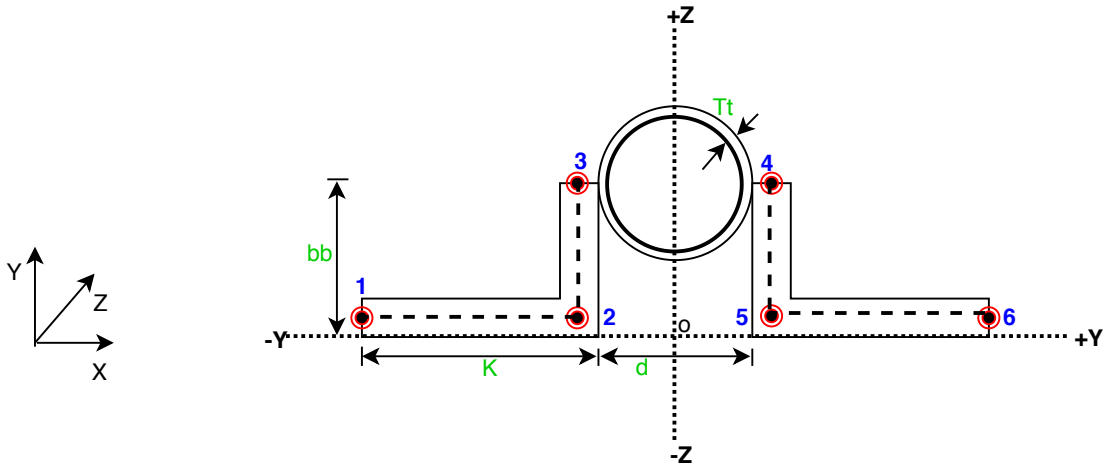


Figure 48: Discretised geometry of the combined profile

Node	Coordinates	
	Y	Z
1	-r - k	z/2
2	-r	z/2
3	-r	bb
4	r	bb
5	r	z/2
6	r + k	z/2

Table 7: Node chart with coordinates

Elements	Node-1	Node-2	Thickness
1	1	2	z
2	2	3	z
3	4	5	z
4	5	6	z

Table 8: Element chart

4.4 Programmer's Guide

Code::Blocks IDE's inherent feature for coding C/C++ Programs as its intrinsic console application is used to implement the program.

In the Thin-walled approximation approach, the classes used in the code are Base class, Node class, Element class, Line 2 class, Profile class, Combined profile class and the main file named as maintwa. With the help of these classes the user will be able to approximate the geometry as lines and points along with the computation of the approximate Area and Moment of Inertia of the combined profile. The results are validated and verified with the help of analytical values and compared with the thin walled approximation algorithm values.

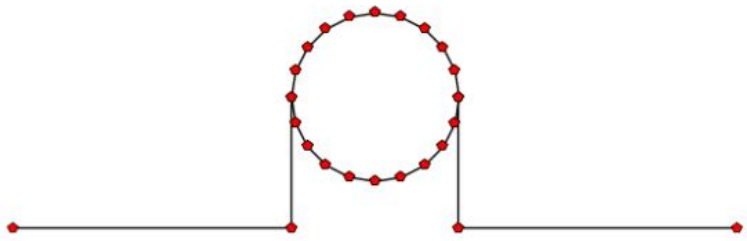


Figure 49: Combined profile using lines as elements and nodes as points

The program is divided into many subroutines to divide the computation process. Each subroutine

does its job and calls other subroutine in next step. The important file is 'maintwa.cpp' which reads and calls out necessary data for the computation as per the defined command in the code for the combined profile as shown in the Figure below. The user also has the option to select the profile type which has to be computed as per different dimensions. It is also necessary to understand the importance of how the classes and the profiles are constructed and this is discussed in the following section.

```
int main()
{


    #ifdef CHECK_COMBINEDPROFILE
    try
    {
        // Insert the value from 1 to 3 for selecting the particular Combined Profile
        int a = 1;

    if (a ==1)
    {                                                          // |k    |bb  |z  |d    |Tr  |nodes
        CombinedProfile* p = new CombinedProfile("CombinedProfile1",100., 50., 6., 60.3, 2.3, 20);
        p->setData();
        p->listData();

    }
    else if (a==2)
    {                                                          // |k    |bb  |z  |d    |Tr  |nodes
        CombinedProfile* p = new CombinedProfile("CombinedProfile2",100., 75., 10., 60.3, 2.3, 20);
        p->setData();
        p->listData();
    }
    else
    {                                                          // |k    |bb  |z  |d    |Tr  |nodes
        CombinedProfile* p = new CombinedProfile("CombinedProfile3",120., 80., 8., 60.3, 2.3, 20);
        p->setData();
        p->listData();
    }
```

Figure 50: Input for the combined profile

4.4.1 UML Diagram:

UML(Unified Modeling Language) diagram is used to visualize design of a system. UML structure diagrams emphasize the things that must be present in the system being modelled. Since structure diagrams represent the structure, they are used extensively in documenting the architecture of software systems. In our description of the examples we want to implement, we use the class diagram which describes the structure of a system by showing the system's classes, their attributes, and the relationship among the classes. The UML diagram consists of a rectangular box, which is divided into three sections.

- **First Section** – Contains name of class.
- **Second Section** – Contains name of attribute.
- **Third Section** – Contains name of method.

Some symbols are used to develop the UML diagram. Every symbols are used for specific purpose.

- The white filled arrowhead points from inheriting class to the inherited class.
- Black filled rhombus represents the composition. The parts which exists in profile are described as a composition.
- White filled rhombus represents the aggregation. The parts which do not exists in profile are described as an aggregation.
- + : It represents that the items are accessible from outside which means it represent public attributes.

- – : It represents that the items are not accessible from outside which means it represents private attributes.

The class hierarchy of the combined profile is shown in the diagram below –

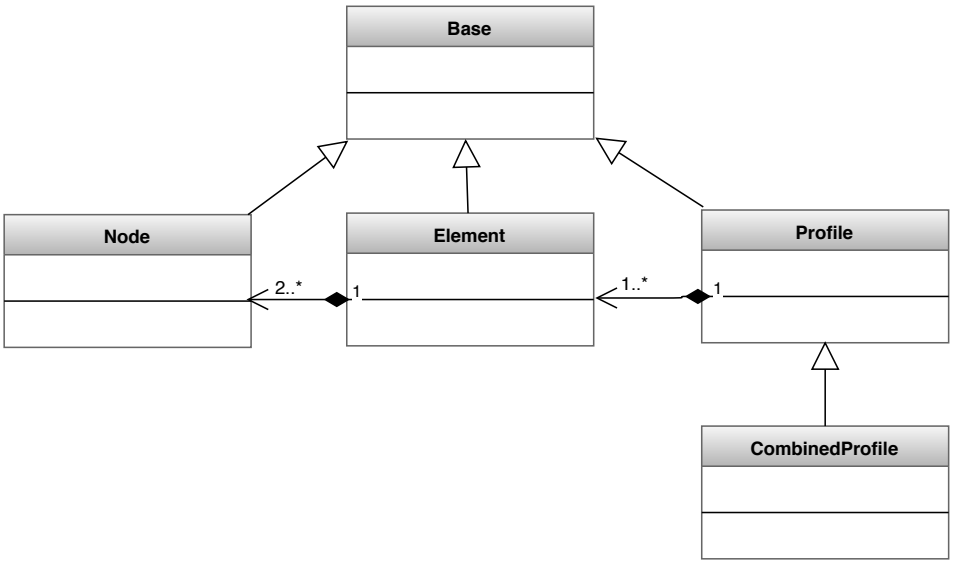


Figure 51: Class hierarchy of developed program

4.4.2 General base class:

The UML class diagram in figure- 52 shows the concept of the common class Base. The class contains three object attributes, the name of the common log file, a print buffer and the instance counter. These attributes are declared with a static property. Besides the constructor and the destructor, the class has a method which will write the content of the message buffer to the screen and/or into a log file. The object method ResetLog deletes an already existing log file. The class Base provides two different log methods appendLog using the same name. Only the parameter list differs. The method should be able to print a string form a character array and a string given as a constant string. This is called name mangling.

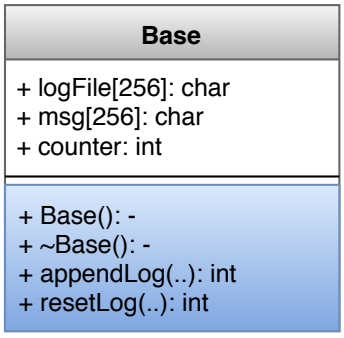


Figure 52: UML diagram of Base class

4.4.3 Node class:

The UML class diagram in figure-53 shows the concept of the class Node. The class contains the node's number and a double array to hold the instance's coordinate data. Besides the constructor and the destructor, the class has a method List which will write the instances data to the log using the inherited method of the class Base::appendLog. To keep it simple all attributes and methods of the class are declared as public.

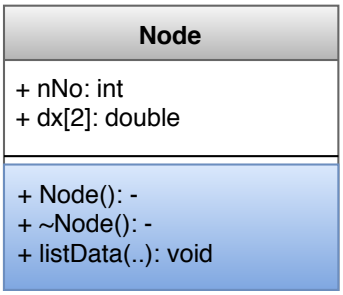


Figure 53: UML diagram of Node class

4.4.4 Element class:

The class `Element` will be able to connect two nodes. The Element class gets the pointer of it's Node instances. If these pointers (the addresses of the Node instances) are known, the Element will be able to calculate all it's sections values. So we can encapsulate these features into the Element. At the end to get the profiles total section values we simply have to add up all it's Element's values. To get a general implementation, we introduce a simple container for the Node instance pointers, which is a simple C array. In the general case we can have elements with an arbitrary number of nodes, so we introduce the number of Node pointers, which in our case will be 2. The Node instance pointers will be stored in the array `m_pN`, which we have to allocate dynamically. Besides the `Node` instance pointers, our element need to know it's thickness. The thickness is stored in the attribute `_t`.

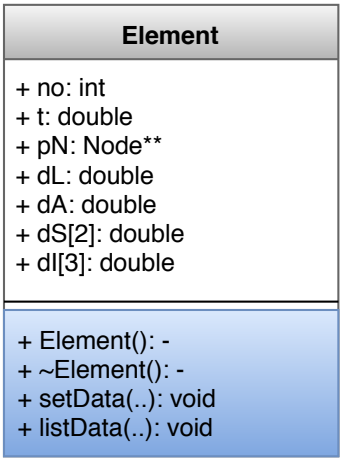


Figure 54: UML diagram of Element class

The following attributes will hold the element's section values –

- `dL:` – Length of the element. This value is used to calculate the section values.
- `dA:` – Calculates the area of the element.
- `dS:` – Calculates the Static moment of the element.
- `dI:` – Calculates the moment of inertia of the element.

Besides the constructor and the destructor, the class provides a method which will write the instances' data to the log using the inherited method of the `class Base::AppendLog`. The method `SetData` will calculate the elements section values. The `Element` class provides only one constructor, i.e only one interface. The constructor comes with four parameters which will describe a linear element.

- `int no:` – The element's number.
- `Node*pN1:` – The instance pointer to the element's starting node.
- `Node*pN2:` – The instance pointer to the element's terminating node.
- `double t:` – The element's thickness.

4.4.5 Line2 Class

The element dimensions like the thickness and the lengths are contained within this class. Area, Static moment and Moment of Inertia are computed with the help of the attributes of the Element class. The Element class is the parent of class Line2.

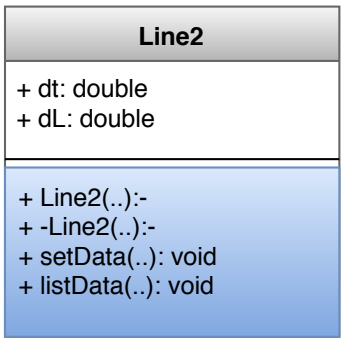


Figure 55: UML diagram of Line2 class

4.4.6 General profile class

The class `Profile` contains the profile's name and a container for the profile's nodes and profile's elements. The containers are built by a simple dynamical array and an integer, which holds the length of the array. The methods of the class creates the containers with a specified length. A `Node` instance and an `Element` instance can be added by a specific `Add` function. To check the node and element information `Check` methods are implemented, which checks the existence of the referenced node and element number. Besides the constructor and the destructor, the class has a method which will write the instances data to the log using the inherited method of the class `Base::appendLog`. To keep it simple all attributes and methods are set to public. We have the following class attributes.

The methods used in this class are as follows –

- `pName [256]`– The profile's name.
- `dA`– Total Area.
- `dS [2]`– Total Static moments.
- `de [2]`– Center of mass coordinates.
- `dIu [3]`– Moment of inertia in user coordinates.
- `dIc [3]`– Moment of inertia in center of mass coordinates.
- `dIm [3]`– Moment of inertia in main coordinates.
- `dAngle`– Main axis angle.
- `pNC`– Pointer to Node instance array.
- `nNC`– Length of node container.
- `pEC`– Pointer to Element instance array.
- `nEC`– Length of element container.

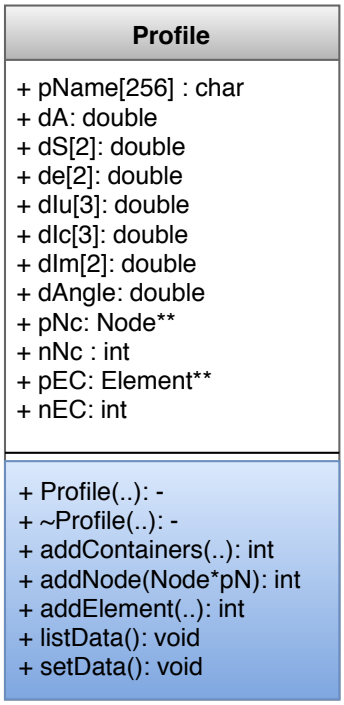


Figure 56: UML diagram of Profile class

- `Profile`– Constructor.
- `Profile`– Destructor.
- `addContainer`– Create node and element container.
- `addNode`– Add a Node pointer into it's array slot.

- `addElement`– Add an Element pointer into it's array slot.
- `listData`– Print all the data of the profile.
- `setData`– Initialize all section values.

4.4.7 CombinedProfile class

The class `CombinedProfile` is user defined Class. Various dimensional parameters are defined according to the geometry of the combined profile. The user coordinates are selected as shown in Table- 7. The nodes and elements are created according to chosen dimensional parameter.

User must enter valid dimensional parameters as per requirement. It is essential to check the entered dimensions of the profile, hence the function for checking the same is implemented in `CombinedProfile` class as shown in figure-57. If user enter invalid data of the profile, then user will get the message in output window referring to invalid dimension and the execution of the program will be stopped.

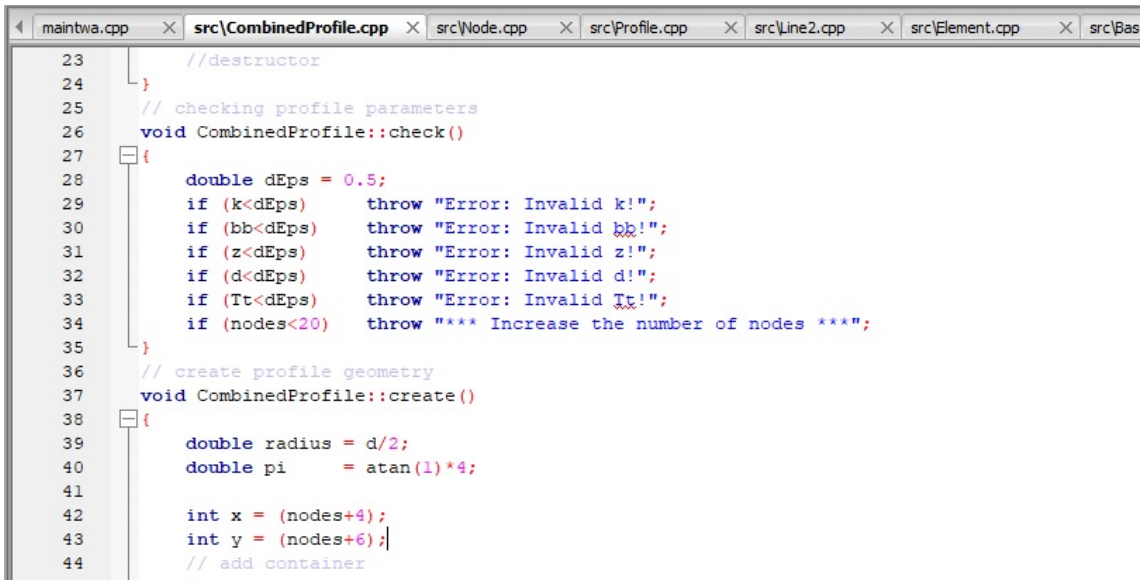


Figure 57: Checking of CombinedProfile Parameter

The only attributes the class `CombinedProfile` gets, are the parameters to describe the profile's geometry.

- `pName [256]`– The CombinedProfile's name.
- `k`– Length of long plate of L-Profile.
- `bb`– Length of short plate of L-Profile.
- `z`– Thickness of L-Profile.
- `d`– Outer diameter of O-Profile.
- `Tt`– Thickness of O-Profile.
- `nodes`– Total nodes on both Circles.

The following methods are implemented in CombinedProfile class.

- `check`– Check the parameter passed by the constructor call.
- `create`– Create the geometry of the profile in terms of nodes and elements.
- `listData`– List the profile's data calling the List method of the base class too.

The UML diagram of the CombinedProfile is shown in figure-58.

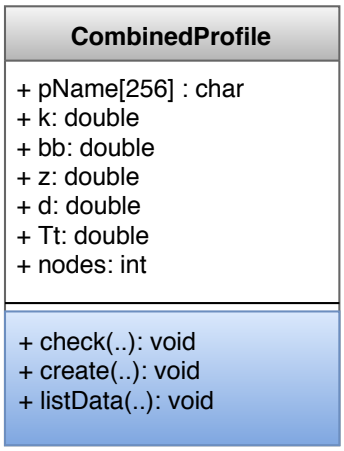


Figure 58: UML diagram of CombinedProfile class

4.5 User Manual

This section provides the instruction to use the developed C++ program to compute the section properties of given combined profile.

An example has been introduced step by step to use the developed program.

- **Open the developed program** – Open the **Code::Blocks** from the task bar of the computer and open 'twa.cbp' file from the project folder **Project-3-CombinedProfile**. User can see the window as shown in Figure-59.

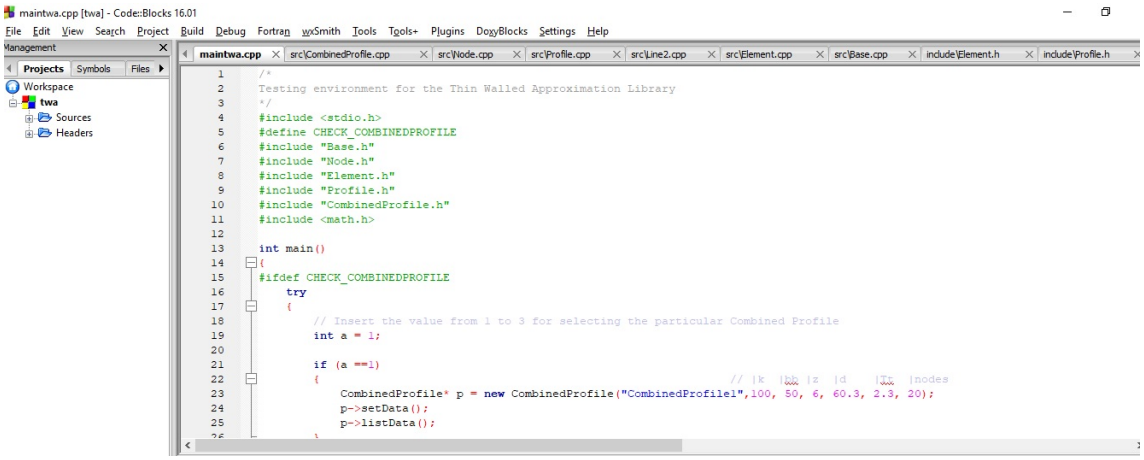


Figure 59: Display of Code::Blocks GUI after opening project file of thin walled approximation

- **Input data to the program and execute the program** – The developed program is capable to calculate section properties of three different combinations as shown in Table-9. There are three instances for three combinations are created.

  The section properties of any of these combined profiles can be calculated by inserting the value from 1 to 3 for selecting the particular Combined profile instance in file 'mainTwa.cpp'. The instances are coded with **If-else** loop in the program. In this example the first combined profile is selected by giving the value **int a = 1**. User can also create or add new instances as per requirement by modifying the loop.

Name	$k[mm]$	$bb[mm]$	$z[mm]$	$d[mm]$	$Tt[mm]$
Combination-1	100	50	6	60.3	2.3
Combination-2	100	75	10	60.3	2.3
Combination-3	120	80	8	60.3	2.3

Table 9: Combined profile dimensions

- Click on the 'Run' or 'Build and Run' button which can be found on the tool bar which is located on the top of the window as shown in Figure-60.

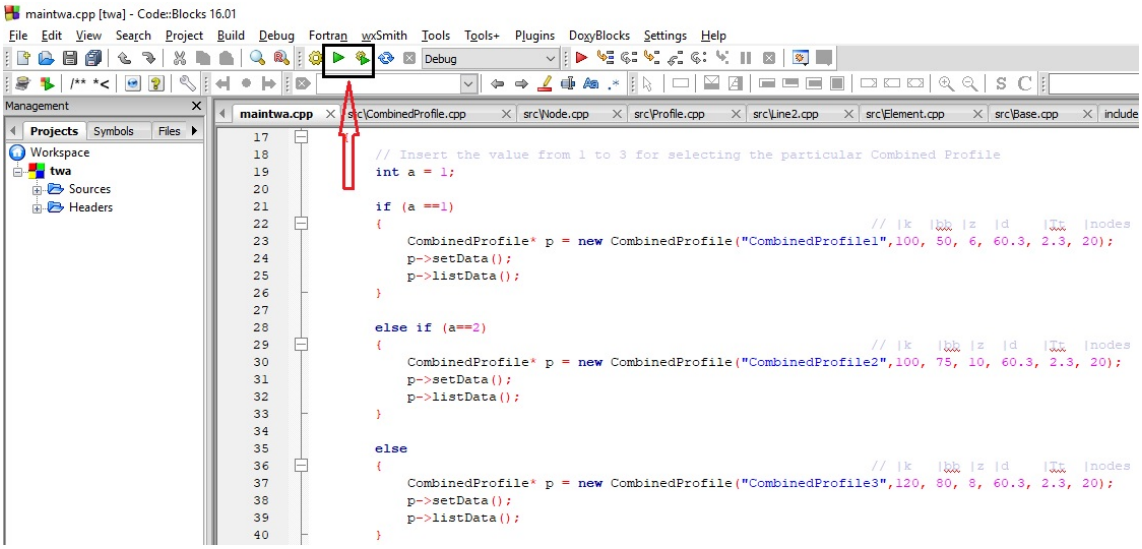


Figure 60: Run the Program

- **Visualization of the Output** After execution of the program, user can see the output window of the calculated section properties of the combined profile as shown in Figure-61.

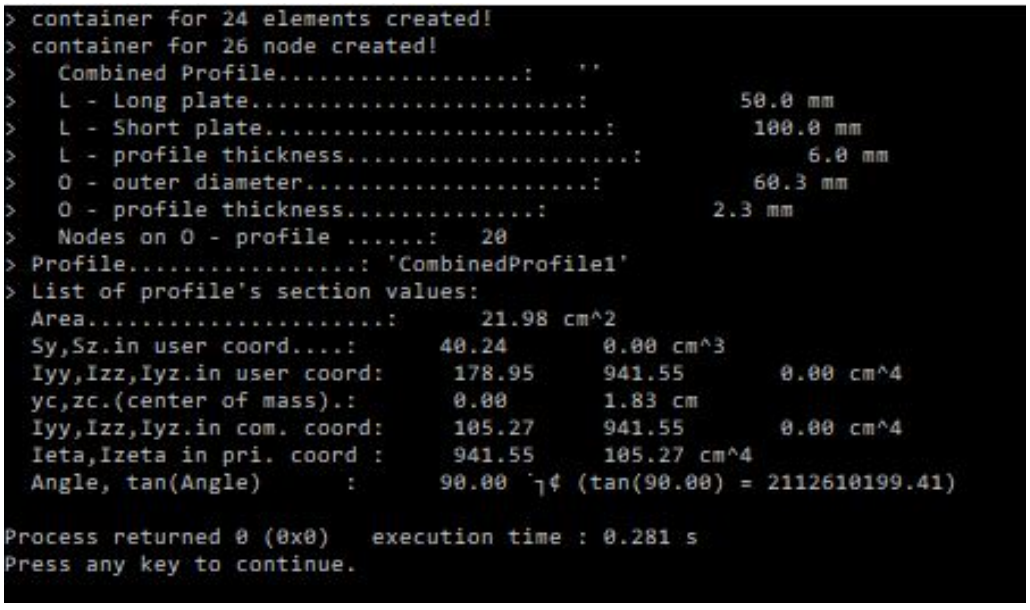


Figure 61: Output Window

Similarly user can also run the program for other combinations and see the calculated section properties] in consol. User can also access the generated log file `twa.log` containing calculated section properties of combined profile.

twa.log - Notepad

File Edit Format View Help

```
> container for 24 elements created!
> container for 26 node created!
>   Combined Profile.....................: ''
>   L - Long plate.......................:        50.0 mm
>   L - Short plate......................:       100.0 mm
>   L - profile thickness................:         6.0 mm
>   O - outer diameter...................:        60.3 mm
>   O - profile thickness................:         2.3 mm
>   Nodes on O - profile ................: 20
> Profile.................: 'CombinedProfile1'
> List of profile's section values:
  Area.....................:      21.98 cm^2
  Sy,Sz.in user coord....:      40.24       0.00 cm^3
  Iyy,Izz,Iyz.in user coord:     178.95     941.55       0.00 cm^4
  yc,zc.(center of mass).:       0.00       1.83 cm
  Iyy,Izz,Iyz.in com. coord:     105.27     941.55       0.00 cm^4
  Ieta,Izeta in pri. coord :     941.55     105.27 cm^4
  Angle, tan(Angle)      :      90.00 ◆ (tan(90.00) = 2112610199.41)
```

Figure 62: Generated log file of calculated section properties for combination-1

4.6 Result Analysis

To check the reliability of the developed C++ program, it is essential to compare the results with analytical results.

4.6.1 Analytical section properties

Analytically calculated physical section properties of the profiles, viz., area, Moment of inertia and Centre of mass are represented in DIN Standards. DIN EN 10056-1 for L-Profile and DIN 2448 for O-profile can be referred for values pertaining to the required parameter. The table below is an excerpt of these values for reference from the standards.

Point to be noted, the newest version of DIN Standard (DIN EN 10056-1) does not contain the details for L 100x75x9 (as given in the problem statement). Hence, L 100x75x10 is considered.

Profile (L `kxbbxz`)	$A_L[cm^2]$	$I_y[cm^4]$	$I_z[cm^4]$	$e_y[cm]$	$e_z[cm]$
L 100x50x6	8.71	89.9	15.4	3.51	1.05
L 100x75x10	16.6	162	77.6	3.19	1.95
L 120x80x8	15.5	226	80.8	3.83	1.87

Table 10: Sectional properties of L-profile

Profile(`dxTt`)	A$[cm^2]$	I$[cm^4]$
O 60.3x2.3	4.19	17.7

Table 11: Sectional properties of O-profile

Comparison of results drawn from the program using approximation method and that calculated using the analytical values from above table 10 and table 11 is discussed in the section.

4.6.2 Analytical calculation for combination-1 (L100x50x6 + O60.3x2.3)

Some abbreviations are used as follows –
L100x50x6 = L_{100}
L100x75x10 = L_{75}
L120x80x8 = L_{120}
O60.3x2.3 = $O_{60.3}$

- **Total Area:**

$$\begin{aligned} A &= (2 \times A_{L_{100}}) + (A_{O_{60.3}}) \\ &= (2 \times 8.71) + (4.19) \\ &= 21.61 cm^2 \end{aligned}$$

- **Center of mass:** (y_c, z_c)

$$\begin{aligned} z_c &= \frac{\int_A z.dA}{\int_A dA} = \frac{S_y}{A} \\ &= \frac{(2 \times 8.71 \times 1.05) + (4.19 \times 5)}{21.61} \\ &= 1.81cm \end{aligned}$$

$$\begin{aligned} y_c &= \frac{\int_A y.dA}{\int_A dA} = \frac{S_z}{A} \\ &= \frac{(8.71 \times -6.5215) + (8.71 \times 6.5215) + (0 \times 4.19)}{21.61} \\ &= 0cm \end{aligned}$$

- **Static Moment:**

$$\begin{aligned} S_y &= z_c \times A \\ &= 1.81 \times 21.61 \\ &= 39.241cm^3 \end{aligned}$$

$$\begin{aligned} S_z &= y_c \times A \\ &= 0 \times 21.61 \\ &= 0cm^3 \end{aligned}$$

- **Moment of inertia w.r.t User coordinates:**

$$\begin{aligned} I_{yy} &= I_y + e_z^2 \times A \\ &= [2 \times (15.4 + (8.71 \times 1.05^2))] + [17.7 + (4.19 \times 5^2)] \\ &= 172.455cm^4 \end{aligned}$$

$$\begin{aligned} I_{zz} &= I_z + e_y^2 \times A \\ &= [2 \times (89.9 + (8.71 \times +6.525^2))] + [17.7 + (4.19 \times 0)] \\ &= 939.167cm^4 \end{aligned}$$

$$\begin{aligned} I_{yzu} &= I_{yz} + (e_y \times e_z) \times A \\ &= 0cm^4 \end{aligned}$$

- **Moment of inertia w.r.t Center of mass:**

$$\begin{aligned} I_{yc} &= I_y + (A \times d_z^2) \\ &= [2 \times (15.4 + (8.71 \times 0.76^2))] + [17.7 + (4.19 \times 3.19^2)] \\ &= 101.1996cm^4 \end{aligned}$$

$$\begin{aligned} I_{zc} &= I_z + (A \times d_y^2) \\ &= [2 \times (89.9 + (8.71 \times +6.525^2))] + [17.7 + (4.19 \times 0)] \\ &= 939.167cm^4 \end{aligned}$$

$$\begin{aligned} I_{yzc} &= I_{yz} + (d_y \times d_z) \times A \\ &= 0cm^4 \end{aligned}$$

The program results are generated as follows –

```
> container for 24 elements created!
> container for 26 node created!
>   Combined Profile.................:   ''
>   L - Long plate...................:          50.0 mm
>   L - Short plate..................:          100.0 mm
>   L - profile thickness............:             6.0 mm
>   O - outer diameter...............:          60.3 mm
>   O - profile thickness............:         2.3 mm
>   Nodes on O - profile ......:   20
> Profile.................: 'CombinedProfile1'
> List of profile's section values:
  Area.....................:      21.98 cm^2
  Sy,Sz.in user coord.....:     40.24       0.00 cm^3
  Iyy,Izz,Iyz.in user coord:    178.95      941.55       0.00 cm^4
  yc,zc.(center of mass).:      0.00        1.83 cm
  Iyy,Izz,Iyz.in com. coord:    105.27      941.55       0.00 cm^4
  Ieta,Izeta in pri. coord :    941.55      105.27 cm^4
  Angle, tan(Angle)      :      90.00 ¯┐$ (tan(90.00) = 2112610199.41)

Process returned 0 (0x0)   execution time : 0.281 s
Press any key to continue.
```

Figure 63: Result of the Combined Profile-1

- **Percentage Error:**

$$e_A = \frac{Area_{analytical} - Area_{python}}{Area_{analytical}} \times 100 \qquad = \frac{21.61 - 21.98}{21.61} \times 100 \approx -1.71\%$$

$$e_{S_y} = \frac{S_{y_{analytical}} - S_{y_{python}}}{S_{y_{analytical}}} \times 100 \qquad = \frac{39.24 - 40.24}{39.24} \times 100 \approx -2.54\%$$

$$e_{I_{yy}} = \frac{I_{yy_{analytical}} - I_{yy_{python}}}{I_{yy_{analytical}}} \times 100 \qquad = \frac{172.45 - 178.95}{172.45} \times 100 \approx -3.76\%$$

$$e_{I_{zz}} = \frac{I_{zz_{analytical}} - I_{zz_{python}}}{I_{zz_{analytical}}} \times 100 \qquad = \frac{939.167 - 941.55}{939.167} \times 100 \approx -0.25\%$$

$$e_{I_{yc}} = \frac{I_{yc_{analytical}} - I_{yc_{python}}}{I_{yc_{analytical}}} \times 100 \qquad = \frac{101.20 - 105.27}{101.20} \times 100 \approx -4.02\%$$

$$e_{I_{zc}} = \frac{I_{zc_{analytical}} - I_{zc_{python}}}{I_{zc_{analytical}}} \times 100 \qquad = \frac{939.167 - 941.55}{939.167} \times 100 \approx -0.25\%$$

4.6.3 Analytical calculation for combination-2 (L100x75x10 + O60.3x2.3)

- **Total Area:**

$$\begin{aligned} A &= (2 \times A_{L_{75}}) + (A_{O_{60.3}}) \\ &= (2 \times 16.6) + (4.19) \\ &= 37.39cm^2 \end{aligned}$$

- **Center of mass:** (y_c, z_c)

$$\begin{aligned} z_c &= \frac{\int_A z.dA}{\int_A dA} = \frac{S_y}{A} \\ &= \frac{(2 \times 16.6 \times 1.95) + (4.19 \times 7.5)}{37.39} \\ &= 2.57cm \end{aligned}$$

$$\begin{aligned} y_c &= \frac{\int_A y.dA}{\int_A dA} = \frac{S_z}{A} \\ &= \frac{(16.6 \times -6.205) + (16.6 \times 6.205) + (0 \times 4.19)}{37.39} \\ &= 0cm \end{aligned}$$

- **Static Moment:**

$$\begin{aligned} S_y &= z_c \times A \\ &= 2.29 \times 37.39 \\ &= 96.09cm^3 \end{aligned}$$

$$\begin{aligned} S_z &= y_c \times A \\ &= 0 \times 37.39 \\ &= 0cm^3 \end{aligned}$$

- **Moment of inertia w.r.t User coordinates:**

$$\begin{aligned} I_{yy} &= I_y + e_z^2 \times A \\ &= [2 \times (77.6 + (16.6 \times 1.95^2))] + [17.7 + (4.19 \times 7.5^2)] \\ &= 534.83cm^4 \end{aligned}$$

$$\begin{aligned} I_{zz} &= I_z + e_y^2 \times A \\ &= [2 \times (162 + (16.6 \times 6.205^2))] + [17.7 + (4.19 \times 0)] \\ &= 1619.967cm^4 \end{aligned}$$

$$\begin{aligned} I_{yzu} &= I_{yz} + (e_y \times e_z) \times A \\ &= 0cm^4 \end{aligned}$$

- **Moment of inertia w.r.t Center of mass:**

$$\begin{aligned} I_{yc} &= I_y + (A \times d_z^2) \\ &= [2 \times (77.6 + (16.6 \times 0.62^2))] + [17.7 + (4.19 \times 4.93^2)] \\ &= 287.49cm^4 \end{aligned}$$

$$
\begin{aligned}
I_{zc} &= I_z + e_y^2 \times A \\
&= [2 \times (162 + (16.6 \times 6.205^2))] + [17.7 + (4.19 \times 0)] \\
&= 1619.967 cm^4
\end{aligned}
$$

$$
\begin{aligned}
I_{yzc} &= I_{yz} + (d_y \times d_z) \times A \\
&= 0 cm^4
\end{aligned}
$$

The program results are generated as follows –

```
> container for 24 elements created!
> container for 26 node created!
>   Combined Profile.................:   ''
>   L - Long plate...................:          75.0 mm
>   L - Short plate..................:          100.0 mm
>   L - profile thickness............:            10.0 mm
>   O - outer diameter...............:          60.3 mm
>   O - profile thickness............:         2.3 mm
>   Nodes on O - profile ......:   20
> Profile.................: 'CombinedProfile2'
> List of profile's section values:
  Area.....................:      38.34 cm^2
  Sy,Sz.in user coord....:     98.54       0.00 cm^3
  Iyy,Izz,Iyz.in user coord:    549.65    1598.13       0.00 cm^4
  yc,zc.(center of mass).:      0.00       2.57 cm
  Iyy,Izz,Iyz.in com. coord:    296.36    1598.13       0.00 cm^4
  Ieta,Izeta in pri. coord :   1598.13     296.36 cm^4
  Angle, tan(Angle)      :     90.00 ¯┐¢ (tan(90.00) = 3190608931.77)

Process returned 0 (0x0)   execution time : 0.266 s
Press any key to continue.
```

Figure 64: Result of the Combined Profile-2

- **Percentage Error:**

$$
e_A = \frac{Area_{analytical} - Area_{python}}{Area_{analytical}} \times 100 \qquad = \frac{37.39 - 38.34}{37.39} \times 100 \approx -2.54\%
$$

$$
e_{S_y} = \frac{S_{y_{analytical}} - S_{y_{python}}}{S_{y_{analytical}}} \times 100 \qquad = \frac{96.09 - 98.54}{96.09} \times 100 \approx -2.54\%
$$

$$
e_{I_{yy}} = \frac{I_{yy_{analytical}} - I_{yy_{python}}}{I_{yy_{analytical}}} \times 100 \qquad = \frac{534.83 - 549.65}{534.83} \times 100 \approx -2.77\%
$$

$$
e_{I_{zz}} = \frac{I_{zz_{analytical}} - I_{zz_{python}}}{I_{zz_{analytical}}} \times 100 \qquad = \frac{1619.96 - 1598.13}{1619.96} \times 100 \approx 1.34\%
$$

$$
e_{I_{yc}} = \frac{I_{yc_{analytical}} - I_{yc_{python}}}{I_{yc_{analytical}}} \times 100 \qquad = \frac{287.49 - 296.36}{287.49} \times 100 \approx -3.08\%
$$

$$
e_{I_{zc}} = \frac{I_{zc_{analytical}} - I_{zc_{python}}}{I_{zc_{analytical}}} \times 100 \qquad = \frac{1619.96 - 1598.13}{1619.96} \times 100 \approx 1.34\%
$$

4.6.4 Analytical calculation for combination-3 (L120x80x8 + O60.3x2.3)

- **Total Area:**

$$\begin{aligned} A &= (2 \times A_{L_{120}}) + (A_{O_{60.3}}) \\ &= (2 \times 15.5) + (4.19) \\ &= 35.19cm^2 \end{aligned}$$

- **Center of mass:** (y_c, z_c)

$$\begin{aligned} z_c &= \frac{\int_A z.dA}{\int_A dA} = \frac{S_y}{A} \\ &= \frac{(2 \times 16.6 \times 1.87) + (4.19 \times 8)}{35.19} \\ &= 2.71cm \end{aligned}$$

$$\begin{aligned} y_c &= \frac{\int_A y.dA}{\int_A dA} = \frac{S_z}{A} \\ &= \frac{(16.6 \times -6.845) + (16.6 \times 6.845) + (0 \times 4.19)}{35.19} \\ &= 0cm \end{aligned}$$

- **Static Moment:**

$$\begin{aligned} S_y &= z_c \times A \\ &= 2.71 \times 35.19 \\ &= 95.36cm^3 \end{aligned}$$

$$\begin{aligned} S_z &= y_c \times A \\ &= 0 \times 35.19 \\ &= 0cm^3 \end{aligned}$$

- **Moment of inertia w.r.t User coordinates:**

$$\begin{aligned} I_{yy} &= I_y + e_z^2 \times A \\ &= [2 \times (80.8 + (15.5 \times 1.87^2))] + [17.7 + (4.19 \times 8^2)] \\ &= 555.86cm^4 \end{aligned}$$

$$\begin{aligned} I_{zz} &= I_z + e_y^2 \times A \\ &= [2 \times (226 + (15.5 \times 6.845^2))] + [17.7 + (4.19 \times 0)] \\ &= 1922.17cm^4 \end{aligned}$$

$$\begin{aligned} I_{yzu} &= I_{yz} + (e_y \times e_z) \times A \\ &= 0cm^4 \end{aligned}$$

- **Moment of inertia w.r.t Center of mass:**

$$\begin{aligned} I_{yc} &= I_y + (A \times d_z^2) \\ &= [2 \times (77.6 + (16.6 \times 1.12^2))] + [17.7 + (4.19 \times 5.29^2)] \\ &= 331.79cm^4 \end{aligned}$$

$$
\begin{aligned}
I_{zc} &= I_z + e_y^2 \times A \\
&= [2 \times (226 + (15.5 \times 6.845^2))] + [17.7 + (4.19 \times 0)] \\
&= 1922.17 cm^4
\end{aligned}
$$

$$
\begin{aligned}
I_{yzc} &= I_{yz} + (d_y \times d_z) \times A \\
&= 0 cm^4
\end{aligned}
$$

The program results are generated as follows –

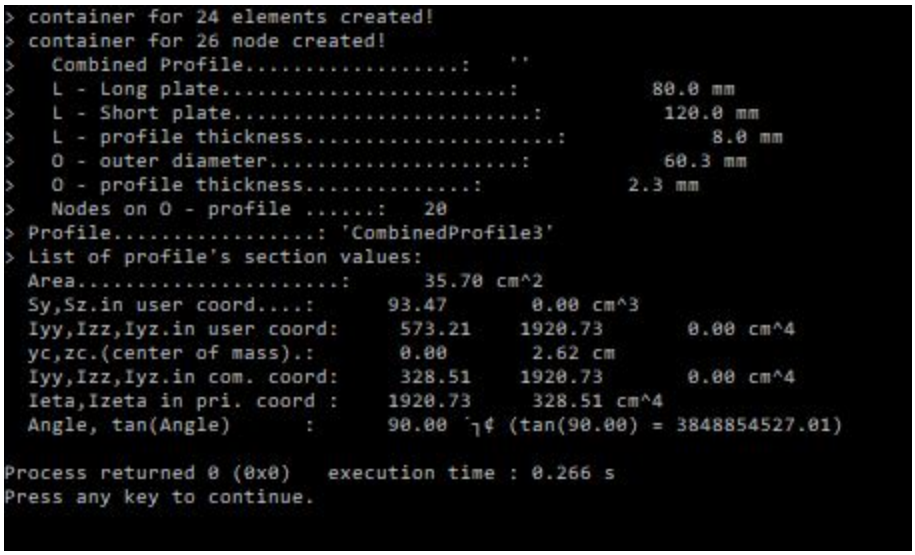


Figure 65: Result of the Combined Profile-3

- **Percentage Error:**

$$e_A = \frac{Area_{analytical} - Area_{python}}{Area_{analytical}} \times 100 \qquad = \frac{35.19 - 35.70}{35.19} \times 100 \approx -1.44\%$$

$$e_{S_y} = \frac{S_{y_{analytical}} - S_{y_{python}}}{S_{y_{analytical}}} \times 100 \qquad = \frac{95.36 - 93.47}{95.36} \times 100 \approx 1.98\%$$

$$e_{I_{yy}} = \frac{I_{yy_{analytical}} - I_{yy_{python}}}{I_{yy_{analytical}}} \times 100 \qquad = \frac{555.86 - 573.21}{555.86} \times 100 \approx -3.12\%$$

$$e_{I_{zz}} = \frac{I_{zz_{analytical}} - I_{zz_{python}}}{I_{zz_{analytical}}} \times 100 \qquad = \frac{1922.17 - 1920.73}{1922.17} \times 100 \approx 0.074\%$$

$$e_{I_{yc}} = \frac{I_{yc_{analytical}} - I_{yc_{python}}}{I_{yc_{analytical}}} \times 100 \qquad = \frac{331.79 - 328.51}{331.79} \times 100 \approx 0.98\%$$

$$e_{I_{zc}} = \frac{I_{zc_{analytical}} - I_{zc_{python}}}{I_{zc_{analytical}}} \times 100 \qquad = \frac{1922.17 - 1920.73}{1922.17} \times 100 \approx 0.074\%$$

4.7 Result and discussion

The tabular representation of the results for comparison is shown as follows -

Section Properties	Combined profile			Remarks
	L_{100}	L_{75}	L_{120}	
Area(A)[cm^2]	21.98	38.34	35.70	Computed
	21.61	37.39	35.19	Analytical
	-1.71%	-2.54%	-1.44%	Error
Static Moment(S_y)[cm^3]	40.24	98.54	93.47	Computed
	39.24	96.09	95.36	Analytical
	-2.54%	-2.54%	1.98%	Error
Moment of inertia w.r.t uc(I_{yy})[cm^4]	178.95	549.65	573.21	Computed
	172.45	534.83	555.86	Analytical
	-3.76%	-2.77%	-3.12%	Error
Moment of inertia w.r.t uc(I_{zz})[cm^4]	941.55	1598.13	1920.73	Computed
	939.16	1619.96	1922.17	Analytical
	-0.25%	1.34%	0.074%	Error
Moment of inertia w.r.t COM(I_{yc})[cm^4]	105.27	296.36	328.51	Computed
	101.20	287.49	331.79	Analytical
	-4.02%	-3.08%	0.98%	Error
Moment of inertia w.r.t COM(I_{zc})[cm^4]	941.55	1598.13	1920.73	Computed
	939.16	1619.96	1922.17	Analytical
	-0.25%	1.34%	0.074%	Error

Table 12: Result Analysis Chart

From the Table-12 it can be seen that we are getting different values of moment of inertia in user coordinates than center coordinates because we have considered asymmetric origin for user coordinate and due to eccentricity we are getting the different values. Due to symmetry with respect to z-axis we are getting 0 static moment S_z in all combinations.

It can be observed that, percentage of variation in moment of inertia w.r.t center coordinates I_{yc} are on negative side, whereas I_{zc} are on positive side for two out of three combinations. Such a variation is due to the approximation of the profile.

The negative sign indicates that the calculated values are higher than the actual values and positive sign indicates the calculated values are smaller than actual values. However, the variation lies below 5% for all combinations. This variation can be considered as a factor of safety.

The reason for such variation is due to the negligence of fillets and assuming overlap of profiles in Thin Walled Approach leads to variation in the area and center of mass resulting the variation in moment of inertia. This leads to change in exural rigidity (EI) of the combined profile in which young's modulus remains same for all combinations. The higher moment of inertia makes more

stiffer profile and hence, it is more safer to use.

Since % of variation in area and moment of inertia lies around or below 2.50%, it can be examined that the negligible variation in exural rigidity will not affect the maximum allowable load carrying capacity of the profile considering linear static analysis, therefore the developed program is on safer side.

It can be concluded that the developed python program for computation of the section properties of the combined profile is validated by comparing with analytical values. Hence, the developed code is reliable and on safer side.

Bibliography

[1] Dr. E. Baeck. Computer languages for engineers. 2016.

1 Annex-I

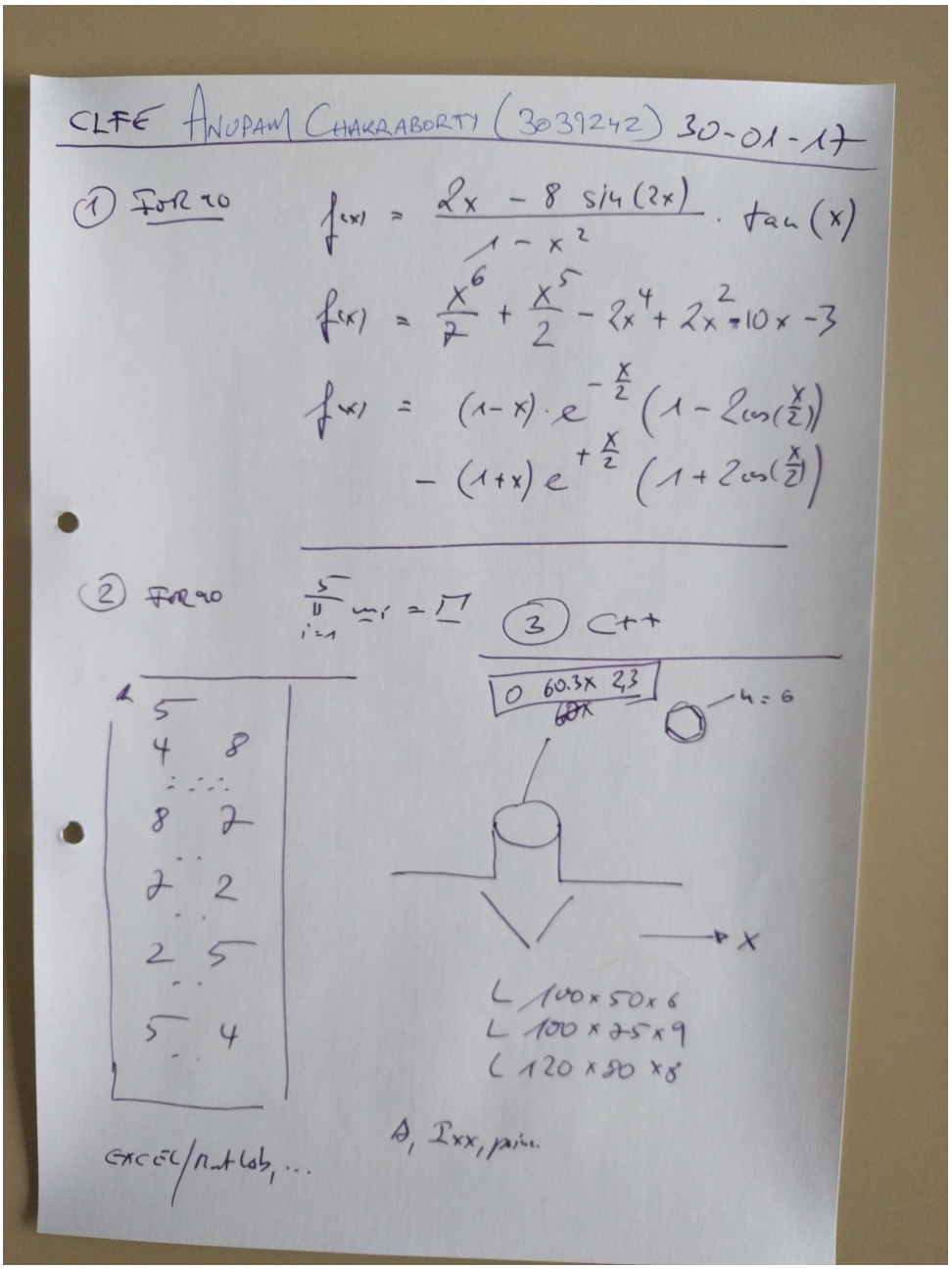


Figure 66: Task Sheet

2 Annex-II

DIN EN 10056-1:2017-06
EN 10056-1:2017 (D)

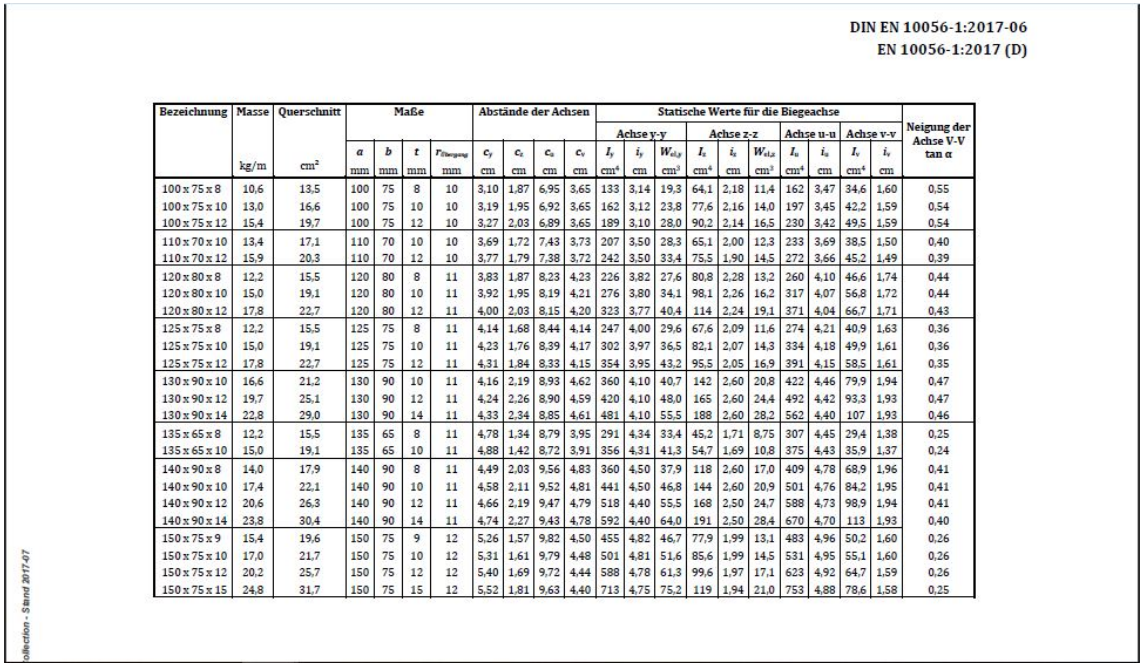
Tabelle 2 — Maße und statische Werte von warmgewalzten ungleichschenkligen Winkeln

Bezeichnung	Masse	Querschnitt	Maße				Abstände der Achsen				Statische Werte für die Biegeachse										Neigung der Achse V-V tan α
											Achse y-y			Achse z-z			Achse u-u		Achse v-v		
	kg/m	cm²	a mm	b mm	t mm	$r_{Übergang}$ mm	c_y cm	c_z cm	c_u cm	c_v cm	I_y cm⁴	i_y cm	$W_{el,y}$ cm³	I_z cm⁴	i_z cm	$W_{el,z}$ cm³	I_u cm⁴	i_u cm	I_v cm⁴	i_v cm	
30 x 20 x 3	1,12	1,43	30	20	3	4	0,99	0,50	2,05	1,04	1,25	0,94	0,62	0,44	0,55	0,29	1,43	1,00	0,26	0,42	0,43
30 x 20 x 4	1,46	1,86	30	20	4	4	1,03	0,54	2,02	1,04	1,59	0,93	0,81	0,53	0,55	0,38	1,81	0,99	0,33	0,42	0,42
40 x 20 x 4	1,77	2,26	40	20	4	4	1,47	0,48	2,58	1,17	3,59	1,26	1,42	0,60	0,51	0,39	3,80	1,30	0,39	0,42	0,25
40 x 25 x 4	1,93	2,46	40	25	4	4	1,36	0,62	2,69	1,35	3,89	1,26	1,47	1,16	0,69	0,62	4,35	1,33	0,70	0,53	0,38
45 x 30 x 4	2,25	2,87	45	30	4	4,5	1,48	0,74	3,07	1,58	5,78	1,42	1,91	2,05	0,85	0,91	6,65	1,52	1,18	0,64	0,44
50 x 30 x 5	2,96	3,78	50	30	5	5	1,73	0,74	3,33	1,65	9,36	1,57	2,86	2,51	0,82	1,11	10,3	1,65	1,54	0,64	0,35
60 x 30 x 5	3,36	4,28	60	30	5	5	2,17	0,68	3,88	1,77	15,6	1,91	4,07	2,63	0,78	1,14	16,5	1,97	1,71	0,63	0,26
60 x 40 x 5	3,76	4,79	60	40	5	6	1,96	0,97	4,10	2,11	17,2	1,89	4,25	6,11	1,13	2,02	19,7	2,03	3,54	0,86	0,43
60 x 40 x 6	4,46	5,68	60	40	6	6	2,00	1,01	4,08	2,10	20,1	1,88	5,03	7,12	1,12	2,38	23,1	2,02	4,16	0,86	0,43
65 x 50 x 5	4,35	5,54	65	50	5	6	1,99	1,25	4,53	2,39	23,2	2,05	5,14	11,9	1,47	3,19	28,8	2,28	6,32	1,07	0,58
70 x 50 x 6	5,41	6,89	70	50	6	7	2,23	1,25	4,83	2,52	33,4	2,20	7,01	14,2	1,43	3,78	39,7	2,40	7,92	1,07	0,50
75 x 50 x 6	5,65	7,19	75	50	6	7	2,44	1,21	5,12	2,64	40,5	2,37	8,01	14,4	1,42	3,81	46,6	2,55	8,36	1,08	0,44
75 x 50 x 8	7,39	9,41	75	50	8	7	2,52	1,29	5,08	2,62	52,0	2,35	10,4	18,4	1,40	4,95	59,6	2,52	10,8	1,07	0,43
80 x 40 x 6	5,41	6,89	80	40	6	7	2,85	0,88	5,20	2,38	44,9	2,55	8,73	7,59	1,05	2,44	47,6	2,63	4,93	0,85	0,26
80 x 40 x 8	7,07	9,01	80	40	8	7	2,94	0,96	5,14	2,34	57,6	2,53	11,4	9,61	1,03	3,16	60,9	2,60	6,34	0,84	0,25
80 x 60 x 7	7,36	9,38	80	60	7	8	2,51	1,52	5,55	2,92	59,0	2,51	10,7	28,4	1,74	6,34	72,0	2,77	15,4	1,28	0,55
100 x 50 x 6	6,84	8,71	100	50	6	8	3,51	1,05	6,55	3,00	89,9	3,21	13,8	15,4	1,33	3,89	95,4	3,31	9,92	1,07	0,26
100 x 50 x 8	8,97	11,40	100	50	8	8	3,60	1,13	6,48	2,96	116	3,19	18,2	19,7	1,31	5,08	123	3,28	12,8	1,06	0,26
100 x 65 x 7	8,77	11,2	100	65	7	10	3,23	1,51	6,83	3,49	113	3,17	16,6	37,6	1,83	7,53	128	3,39	22,0	1,40	0,42
100 x 65 x 8	9,94	12,7	100	65	8	10	3,27	1,55	6,81	3,47	127	3,16	18,9	42,2	1,83	8,54	144	3,37	24,8	1,40	0,41
100 x 65 x 9	11,1	14,1	100	65	9	10	3,32	1,59	6,78	3,42	141	3,20	21,1	46,7	1,82	9,52	160	3,36	27,4	1,39	0,41
100 x 65 x 10	12,3	15,6	100	65	10	10	3,36	1,63	6,76	3,45	154	3,14	23,2	51,0	1,81	10,5	175	3,35	30,1	1,39	0,41
100 x 65 x 11	13,4	17,1	100	65	11	10	3,40	1,67	6,74	3,41	167	3,10	25,3	55,1	1,80	11,4	189	3,33	32,6	1,38	0,41
100 x 65 x 12	14,5	18,5	100	65	12	10	3,44	1,71	6,72	3,40	180	3,10	27,4	59,1	1,80	12,3	203	3,32	35,2	1,38	0,41

ection - Stand 2017-07

Figure 67: Dimensional Details and Sectional Properties of L-Profile - I

3 Annex-III

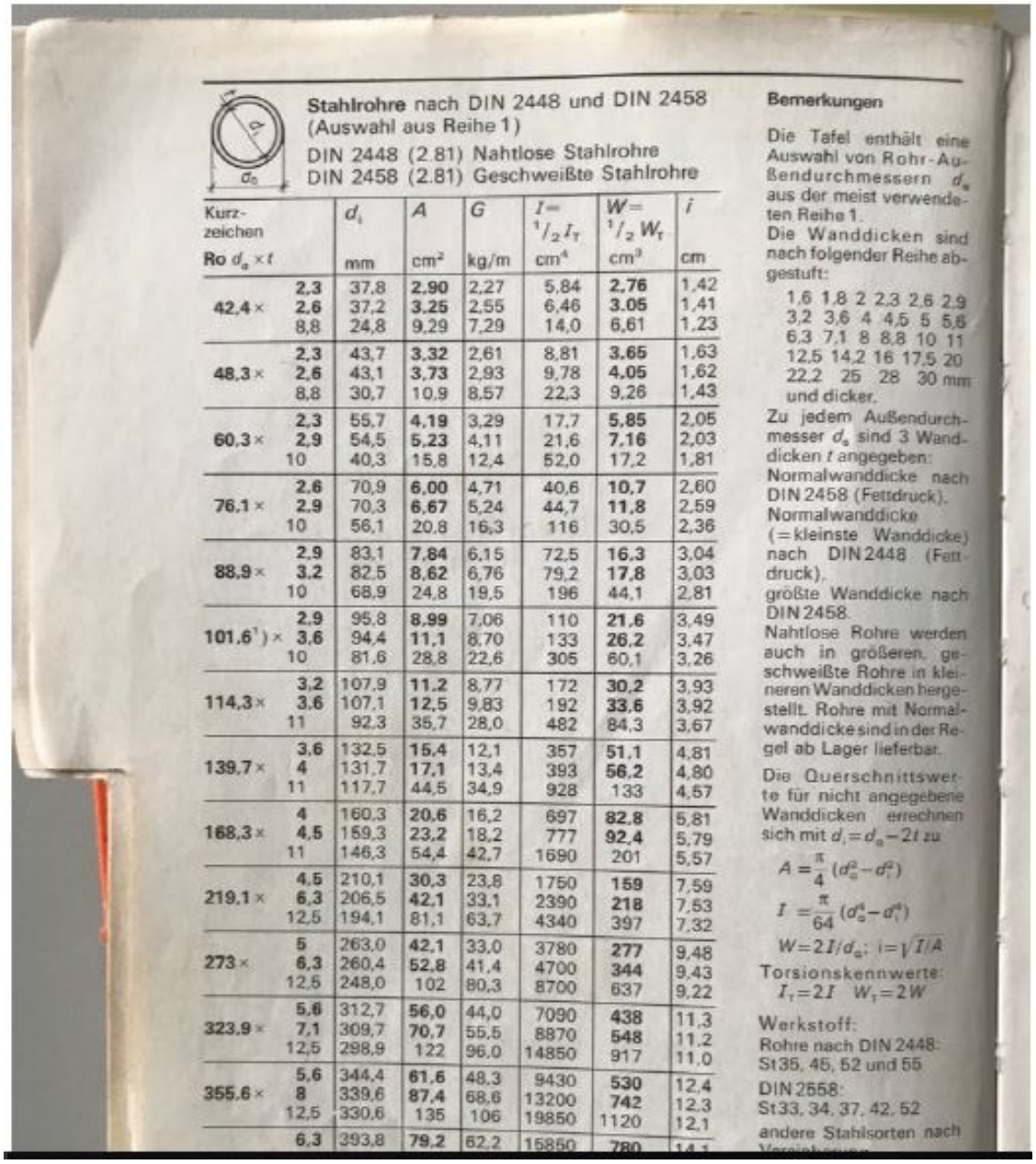


DIN EN 10056-1:2017-06
EN 10056-1:2017 (D)

Bezeichnung	Masse	Querschnitt	Maße				Abstände der Achsen				Statische Werte für die Biegeachse										Neigung der Achse V-V tan α
											Achse y-y			Achse z-z			Achse u-u		Achse v-v		
	kg/m	cm²	a mm	b mm	t mm	$r_{Biegung}$ mm	c_y cm	c_z cm	c_u cm	c_v cm	I_y cm⁴	i_y cm	$W_{el,y}$ cm³	I_z cm⁴	i_z cm	$W_{el,z}$ cm³	I_u cm⁴	i_u cm	I_v cm⁴	i_v cm	
100 x 75 x 8	10,6	13,5	100	75	8	10	3,10	1,87	6,95	3,65	133	3,14	19,3	64,1	2,18	11,4	162	3,47	34,6	1,60	0,55
100 x 75 x 10	13,0	16,6	100	75	10	10	3,19	1,95	6,92	3,65	162	3,12	23,8	77,6	2,16	14,0	197	3,45	42,2	1,59	0,54
100 x 75 x 12	15,4	19,7	100	75	12	10	3,27	2,03	6,89	3,65	189	3,10	28,0	90,2	2,14	16,5	230	3,42	49,5	1,59	0,54
110 x 70 x 10	13,4	17,1	110	70	10	10	3,69	1,72	7,43	3,73	207	3,50	28,3	65,1	2,00	12,3	233	3,69	38,5	1,50	0,40
110 x 70 x 12	15,9	20,3	110	70	12	10	3,77	1,79	7,38	3,72	242	3,50	33,4	75,5	1,90	14,5	272	3,66	45,2	1,49	0,39
120 x 80 x 8	12,2	15,5	120	80	8	11	3,83	1,87	8,23	4,23	226	3,82	27,6	80,8	2,28	13,2	260	4,10	46,6	1,74	0,44
120 x 80 x 10	15,0	19,1	120	80	10	11	3,92	1,95	8,19	4,21	276	3,80	34,1	98,1	2,26	16,2	317	4,07	56,8	1,72	0,44
120 x 80 x 12	17,8	22,7	120	80	12	11	4,00	2,03	8,15	4,20	323	3,77	40,4	114	2,24	19,1	371	4,04	66,7	1,71	0,43
125 x 75 x 8	12,2	15,5	125	75	8	11	4,14	1,68	8,44	4,14	247	4,00	29,6	67,6	2,09	11,6	274	4,21	40,9	1,63	0,36
125 x 75 x 10	15,0	19,1	125	75	10	11	4,23	1,76	8,39	4,17	302	3,97	36,5	82,1	2,07	14,3	334	4,18	49,9	1,61	0,36
125 x 75 x 12	17,8	22,7	125	75	12	11	4,31	1,84	8,33	4,15	354	3,95	43,2	95,5	2,05	16,9	391	4,15	58,5	1,61	0,35
130 x 90 x 10	16,6	21,2	130	90	10	11	4,16	2,19	8,93	4,62	360	4,10	40,7	142	2,60	20,8	422	4,46	79,9	1,94	0,47
130 x 90 x 12	19,7	25,1	130	90	12	11	4,24	2,26	8,90	4,59	420	4,10	48,0	165	2,60	24,4	492	4,42	93,3	1,93	0,47
130 x 90 x 14	22,8	29,0	130	90	14	11	4,33	2,34	8,85	4,61	481	4,10	55,5	188	2,60	28,2	562	4,40	107	1,93	0,46
135 x 65 x 8	12,2	15,5	135	65	8	11	4,78	1,34	8,79	3,95	291	4,34	33,4	45,2	1,71	8,75	307	4,45	29,4	1,38	0,25
135 x 65 x 10	15,0	19,1	135	65	10	11	4,88	1,42	8,72	3,91	356	4,31	41,3	54,7	1,69	10,8	375	4,43	35,9	1,37	0,24
140 x 90 x 8	14,0	17,9	140	90	8	11	4,49	2,03	9,56	4,83	360	4,50	37,9	118	2,60	17,0	409	4,78	68,9	1,96	0,41
140 x 90 x 10	17,4	22,1	140	90	10	11	4,58	2,11	9,52	4,81	441	4,50	46,8	144	2,60	20,9	501	4,76	84,2	1,95	0,41
140 x 90 x 12	20,6	26,3	140	90	12	11	4,66	2,19	9,47	4,79	518	4,40	55,5	168	2,50	24,7	588	4,73	98,9	1,94	0,41
140 x 90 x 14	23,8	30,4	140	90	14	11	4,74	2,27	9,43	4,78	592	4,40	64,0	191	2,50	28,4	670	4,70	113	1,93	0,40
150 x 75 x 9	15,4	19,6	150	75	9	12	5,26	1,57	9,82	4,50	455	4,82	46,7	77,9	1,99	13,1	483	4,96	50,2	1,60	0,26
150 x 75 x 10	17,0	21,7	150	75	10	12	5,31	1,61	9,79	4,48	501	4,81	51,6	85,6	1,99	14,5	531	4,95	55,1	1,60	0,26
150 x 75 x 12	20,2	25,7	150	75	12	12	5,40	1,69	9,72	4,44	588	4,78	61,3	99,6	1,97	17,1	623	4,92	64,7	1,59	0,26
150 x 75 x 15	24,8	31,7	150	75	15	12	5,52	1,81	9,63	4,40	713	4,75	75,2	119	1,94	21,0	753	4,88	78,6	1,58	0,25

Figure 68: Dimensional Details and Sectional Properties of L-Profile - II

4 Annex-IV



Stahlrohre nach DIN 2448 und DIN 2458
(Auswahl aus Reihe 1)
DIN 2448 (2.81) Nahtlose Stahlrohre
DIN 2458 (2.81) Geschweißte Stahlrohre

Kurzzeichen Ro $d_a \times t$		d_i mm	A cm²	G kg/m	$I = {}^1/_2 I_T$ cm⁴	$W = {}^1/_2 W_t$ cm³	i cm
42,4 ×	2,3	37,8	2,90	2,27	5,84	2,76	1,42
	2,6	37,2	3,25	2,55	6,46	3,05	1,41
	8,8	24,8	9,29	7,29	14,0	6,61	1,23
48,3 ×	2,3	43,7	3,32	2,61	8,81	3,65	1,63
	2,6	43,1	3,73	2,93	9,78	4,05	1,62
	8,8	30,7	10,9	8,57	22,3	9,26	1,43
60,3 ×	2,3	55,7	4,19	3,29	17,7	5,85	2,05
	2,9	54,5	5,23	4,11	21,6	7,16	2,03
	10	40,3	15,8	12,4	52,0	17,2	1,81
76,1 ×	2,6	70,9	6,00	4,71	40,6	10,7	2,60
	2,9	70,3	6,67	5,24	44,7	11,8	2,59
	10	56,1	20,8	16,3	116	30,5	2,36
88,9 ×	2,9	83,1	7,84	6,15	72,5	16,3	3,04
	3,2	82,5	8,62	6,76	79,2	17,8	3,03
	10	68,9	24,8	19,5	196	44,1	2,81
101,6¹) ×	2,9	95,8	8,99	7,06	110	21,6	3,49
	3,6	94,4	11,1	8,70	133	26,2	3,47
	10	81,6	28,8	22,6	305	60,1	3,26
114,3 ×	3,2	107,9	11,2	8,77	172	30,2	3,93
	3,6	107,1	12,5	9,83	192	33,6	3,92
	11	92,3	35,7	28,0	482	84,3	3,67
139,7 ×	3,6	132,5	15,4	12,1	357	51,1	4,81
	4	131,7	17,1	13,4	393	56,2	4,80
	11	117,7	44,5	34,9	928	133	4,57
168,3 ×	4	160,3	20,6	16,2	697	82,8	5,81
	4,5	159,3	23,2	18,2	777	92,4	5,79
	11	146,3	54,4	42,7	1690	201	5,57
219,1 ×	4,5	210,1	30,3	23,8	1750	159	7,59
	6,3	206,5	42,1	33,1	2390	218	7,53
	12,5	194,1	81,1	63,7	4340	397	7,32
273 ×	5	263,0	42,1	33,0	3780	277	9,48
	6,3	260,4	52,8	41,4	4700	344	9,43
	12,5	248,0	102	80,3	8700	637	9,22
323,9 ×	5,6	312,7	56,0	44,0	7090	438	11,3
	7,1	309,7	70,7	55,5	8870	548	11,2
	12,5	298,9	122	96,0	14850	917	11,0
355,6 ×	5,6	344,4	61,6	48,3	9430	530	12,4
	8	339,6	87,4	68,6	13200	742	12,3
	12,5	330,6	135	106	19850	1120	12,1
	6,3	393,8	79,2	62,2	15850	780	14,1

Bemerkungen

Die Tafel enthält eine Auswahl von Rohr-Außendurchmessern d_a aus der meist verwendeten Reihe 1.
Die Wanddicken sind nach folgender Reihe abgestuft:

1,6 1,8 2 2,3 2,6 2,9 3,2 3,6 4 4,5 5 5,6 6,3 7,1 8 8,8 10 11 12,5 14,2 16 17,5 20 22,2 25 28 30 mm und dicker.

Zu jedem Außendurchmesser d_a sind 3 Wanddicken t angegeben:
Normalwanddicke nach DIN 2458 (Fettdruck).
Normalwanddicke (= kleinste Wanddicke) nach DIN 2448 (Fettdruck),
größte Wanddicke nach DIN 2458.
Nahtlose Rohre werden auch in größeren, geschweißte Rohre in kleineren Wanddicken hergestellt. Rohre mit Normalwanddicke sind in der Regel ab Lager lieferbar.

Die Querschnittswerte für nicht angegebene Wanddicken errechnen sich mit $d_i = d_a - 2t$ zu

$$A = \frac{\pi}{4}(d_a^2 - d_i^2)$$

$$I = \frac{\pi}{64}(d_a^4 - d_i^4)$$

$W = 2I/d_a$; $i = \sqrt{I/A}$

Torsionskennwerte:
$I_T = 2I$ $W_T = 2W$

Werkstoff:
Rohre nach DIN 2448:
St35, 45, 52 und 55
DIN 2558:
St33, 34, 37, 42, 52
andere Stahlsorten nach

Figure 69: Dimensional Details and Sectional Properties of O-Profile