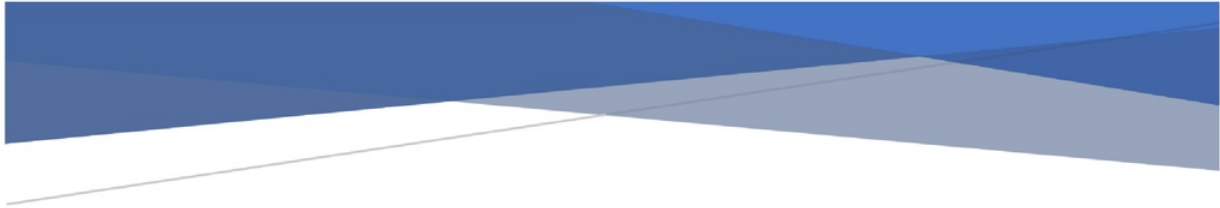


Math Library Report

by Jonathan Davies

FILE	4079_JONATHAN_DAVIES_MATH_LIBRARY_REPORT_112721_49456898 4.DOCX		
TIME SUBMITTED	25-APR-2018 11:04 PM (UTC+0100)	WORD COUNT	1474
SUBMISSION ID	87484777	CHARACTER COUNT	9805



Maths for Software Development

Math Library in C++

Name: **Jonathan S. Davies**

Student Number: **1707239**

Module: Maths for Software Engineering

Lecturer: Gordon Dickers

Contents

Introduction	2
1. Vectors	2
1.1. Length	2
1.2. Normalisation.....	2
1.3. Distance between two points	3
1.4. Dot Product	4
1.5. Cross Product	4
2. Matrices	4
3. Statistics	5
3.1. Mean	5
3.2. Median	6
3.3. Mode	7
3.4. Standard Deviation	7
4. Testing.....	9
4.1. Vector – Length.....	9
4.2. Vector – Normalising.....	9
4.3. Vector – Distance between two points.....	9
4.4. Vector – Dot Product	10
4.5. Vector – Cross Product.....	10
4.6. Matrix.....	11
4.7. Statistics – Mean	12
4.8. Statistics – Median	12
4.9. Statistics – Mode.....	12
4.10. Statistics – Standard Deviation	13

Introduction

The purpose of this report is to provide evidence of understanding of the creation of an accompanying mathematical library that has been written in the C++ programming language. Sections will be split into the various exercises found in the assignment sheet and screenshots shall be provided to further explain how the code works. The various sections of the report are brief in wordcount as the screenshots provided hold comments that are believed to be sufficient in explaining how the code works.

1. Vectors

1.1. Length

In order to find the length of the vector, otherwise known as the magnitude, a vector is obviously required. In the screenshot below, the InputVector function allows the user to enter an X, Y and Z point which is then passed into local variables via the Get functions, present in figure 1.

```
InputVector01();    //Sets the x, y, z
double x = GetX1(); //Assigns the attribute _x to this variable x
double y = GetY1(); //Assigns the attribute _y to this variable y
double z = GetZ1(); //Assigns the attribute _z to this variable z
```

Figure 1

The vector magnitude formula is then applied to the new local variables and the result is saved into the lengthOfVector. Once the calculation has been completed, the result is displayed to the screen as shown below, shown in figure 2.

```
double lengthOfVector = sqrt((x*x) + (y*y) + (z*z)); //Square roots the three points
printf_s("|a| = %.5lf\r\n\r\n", lengthOfVector);    //Display answer
```

Figure 2

Finally, the result is saved into an attribute of the class and also returned so that it may be used in later math operations, as can be seen in figure 3.

```
SetVectorLength(lengthOfVector); //Stores answer in lengthOfVector attribute
return lengthOfVector;
```

Figure 3

1.2. Normalisation

To normalise a vector, the length must be known before the formula can be applied. In this case, when the user chooses the normalising option on the menu, they are presented with another option that asks whether or not they already have a vector length value, as shown in figure 4.

```
Vector normalization
Have you found the length of a vector in the 'Vector Length' section of this program?
1. Yes
2. No
Enter answer:
```

Figure 4

If the user were to choose the 'yes' option, they would be presented with the normalised vector of the vector length that was discovered in the previous VectorLength function, as seen in figure 5.

```
double x = GetX1(); //Gets X1
double y = GetY1(); //Gets Y2
double z = GetZ1(); //Gets Z3
double vL = GetVectorLength(); //Gets the previous length of the vector
double normalizedVectorX = (x / vL); //Calculates X normal
double normalizedVectorY = (y / vL); //Calculates Y normal
double normalizedVectorZ = (z / vL); //Calculates Z normal
```

Figure 5

If however the user were to choose 'no', they would be asked to insert new X, Y and Z points which would then be automatically calculated and presented to the screen, as seen in figure 6.

```
InputVector01(); //Input new points
double x = GetX1(); //Get X and store in 'x'
double y = GetY1(); //Get Y and store in 'y'
double z = GetZ1(); //Get Z and store in 'z'
double vL = sqrt((x*x) + (y*y) + (z*z)); //Sets vL to the square root of x, y and z
double normalizedVectorX = (x / vL); //x divided by the vector length
double normalizedVectorY = (y / vL); //y divided by the vector length
double normalizedVectorZ = (z / vL); //z divided by the vector length
```

Figure 6

1.3. Distance between two points

To calculate the distance between two points, two vectors are needed. The user is prompted to enter the X, Y and Z points of two different vectors and these points are stored and provided to the calculation section of the function, this has been outlined below in figure 7.

```
InputVector01(); //Input X, Y and Z of vector 1
int x1 = GetX1();
int y1 = GetY1();
int z1 = GetZ1();
InputVector02(); //Input X, Y and Z of vector 2
int x2 = GetX2();
int y2 = GetY2();
int z2 = GetZ2();
//Square root of respective points 1 and 2
```

Figure 7

Once the points have been passed into the local variables, the arithmetic operation is ready to start and be displayed to the user. This is detailed in figure 8 below.

```
//Square root of respective points 1 and 2
double vectorDistance = sqrt((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2) + (z1 - z2)*(z1 - z2));
printf("Distance between vectors = %.5lf\r\n", vectorDistance);
SetVectorDistance(vectorDistance); //Stored for potential later use
```

Figure 8

1.4. Dot Product

The dot product works in similar way to other operations, the user is first asked to provide an X, Y and Z point for two separate vectors. Once that is completed, the X, Y and Z points of both vectors are multiplied together and each are added together. The code has been provided in figure 9.

```
InputVector01();    //New vector points
double x1 = GetX1();
double y1 = GetY1();
double z1 = GetZ1();
InputVector02();    //New vector points
double x2 = GetX2();
double y2 = GetY2();
double z2 = GetZ2();
//Dot product formula
double dotProduct = ((x1*x2) + (y1*y2) + (z1*z2));
SetDotProduct(dotProduct);
printf("A.B = %.2lf\r\n", dotProduct);
```

Figure 9

The dot product returns a single value and is stored in an attribute of the class so that it can be used in other parts of the program.

1.5. Cross Product

Following the same procedure as before, the user will be asked to enter X, Y and Z points for two different vectors and the results are declared inside new local variables. Once this is complete, three variables are created named I, J and K. Each of these will yield the result of the calculation that finds the cross product. It is a very similar method to the way in which one would work out the determinant when working with matrices. The result is then displayed in vector format to the screen. The code has been shown in figure 10.

```
int i = (y1*z2) - (z1*y2); //
int j = (x1*z2) - (z1*x2); //Same as working out the determinant for a matrix
int k = (x1*y2) - (y1*x2); //
//J must be negated in order for the formula to be correct
printf("|AxB| = (%d, %d, %d)\r\n", i, -j, k); //Displayed as vector to the user
SetCrossProduct(i, -j, k); //Stored for later use
```

Figure 10

2. Matrices

This section takes two matrices of different dimensions and multiplies them together. The two dimensions used are 4x4 and 4x1. To programme this, two dimensional arrays were created that hold the values that user inputs. Once the values are entered, two 'for' loops were assembled to systematically check every column per row of the 4x4 matrix and multiply it to each iteration of the second matrix. This can be seen in the screenshot labelled figure 11 below.

```

int rows = 4;
int columns = 4;
int totalValue = 0;
printf("Multiplied matrix:\r\n");
for (int i = 0; i < rows; ++i) { //Check each row
    for (int j = 0; j < columns; ++j) { //Check each column
        //Multiply each element in the first matrix by the
        //upcoming element in the second matrix
        totalValue = totalValue + (_matrix4x4[i][j] * _matrix4x1[0][j]);
    }
    printf("\t%d", totalValue); // Store answer in new array
    totalValue = 0; //Reset total value
    printf("\n");
}
printf("\r\n");

```

Figure 11

3. Statistics

3.1. Mean

This is a relatively simple method as all that is required is the user to enter as many numbers as they require, these numbers are then taken, and the sum is found. Once the sum has been found, the total is divided by the amount of numbers that the user originally entered. A variable named 'x' keeps a running record of the total sum so that it is not lost at each iteration of the loop. Code has been provided in figure 12.

```

int userChoice = 0;
double x = 0;
double result;
printf("How many numbers do you require: ");
scanf_s("%d", &userChoice); //UserChoice is used to set the amount of iterations
for (int i = 0; i < userChoice; i++) {
    printf("Enter number: ");
    scanf_s("%lf", &result);
    x = x + result;
}
printf("Mean: %.2lf\r\n", x/userChoice); //The mean is x divided by the original number the user chose
return x/userChoice; //Number is returned for later use

```

Figure 12

3.2. Median

This method is unfinished as there was a problem displaying the correct output from the array. Firstly, the user was asked how many numbers they wished to have in their range. The program would then take that number and create an array with the number of elements specified by the user, this is shown in figure 13.

```
int arraySize;
printf("\r\nHow many numbers do you require: ");
scanf_s("%d", &arraySize); //User decides what the range of numbers is
int* medianNumbers = new int[arraySize]; //Array of that size is created and called 'medianNumbers'
for (int i = 0; i < arraySize; i++) {
    printf("Enter number: ");
    scanf_s("%d", &medianNumbers[i]); //User enters desired numbers
}
```

Figure 13

Once the user entered their desired numbers, a bubble sort arranged their numbers in numerical order and displayed them to the screen. The problem was encountered because each time the sort checked an element in the array, it also checked the next element in the array to. Once the last element was reached, the sort would try and check the next element along but produced a random number due to the fact that there was nothing stored in that address. The code can be seen in figure 14.

```
int placeholder = 0;
int check = arraySize;
for (int counter = arraySize; counter > 0; counter--) {
    for (int i = 0; i < check; i++) { //This is bubble sort that pushes larger numbers to the end of the line
        if (medianNumbers[i] > medianNumbers[i + 1]) {
            placeholder = medianNumbers[i];
            medianNumbers[i] = medianNumbers[i + 1];
            medianNumbers[i + 1] = placeholder;
        }
    }
}
for (int i = 0; i < arraySize; i++) {
    printf("%d ", medianNumbers[i]);
}
```

Figure 14

This problem could not be overcome and the result of it can be seen in the console window in the screenshot labelled figure 15.

```
How many numbers do you require: 6
Enter number: 52
Enter number: 2
Enter number: 526
Enter number: 68
Enter number: 9
Enter number: 11
-33686019 2 9 11 52 68
```

Figure 15

3.3. Mode

The mode method fell to the same problem as the median method. The array could not be displayed to the screen correctly as one of array elements was corrupted. A screenshot of the attempt has been provided in figure 16.

```
int arraySize;
printf("\r\nHow many numbers do you require: ");
scanf_s("%d", &arraySize);
int* modeNumbers = new int[arraySize];
for (int i = 0; i < arraySize + 1; i++) {
    modeNumbers[i] = 0;
}
for (int i = 0; i < arraySize; i++) {
    printf("Enter number: ");
    scanf_s("%d", &modeNumbers[i]);
}
int placeholder = 0;
int check = arraySize + 1;
for (int counter = arraySize; counter > 0; counter--) {
    for (int i = 0; i < check; i++) {
        if (modeNumbers[i] == modeNumbers[i + 1]) {
            placeholder = modeNumbers[i];
        }
    }
}
```

Figure 16

3.4. Standard Deviation

At the very start of the method for the standard deviation calculation, a list of variables needed to be created that would contain the relevant information required. These variables are shown in figure 17.

```
int arraySize;
double mean = 0;
double sum = 0;
double result = 0;
double difference = 0;
double variance = 0;
```

Figure 17

The next stage is to ask the user how many numbers they wish to work. Once this number has been entered, an array is created to the size of their choice. A loop is then entered that sets every value within the new array to zero. Next, the user must enter the values that they wish to work with and they are then added up and saved to a variable called 'sum'. The last stage of this part of the code is to calculate the mean by dividing the sum by the array size. This part of the program has been showcased in figure 18.

```

printf("\r\nHow many numbers do you require: ");
scanf_s("%d", &arraySize);
int* standardDeviation = new int[arraySize];
for (int i = 0; i < arraySize + 1; i++) {
    standardDeviation[i] = 0;
}
for (int i = 0; i < arraySize; i++) {
    printf("Enter number: ");
    scanf_s("%d", &standardDeviation[i]);
}
for (int i = 0; i < arraySize; i++) {
    sum = sum + standardDeviation[i];
}
mean = sum / arraySize;

```

Figure 18

The last stage is to work out the difference between the mean and the users inputted numbers and square that difference. The variance is then totalled and used to display the answer by dividing it with the original array size. Figure 19 shows this code below.

```

for (int i = 0; i < arraySize; i++) {
    if (standardDeviation[i] > result) {
        difference = (standardDeviation[i] - mean);
        difference = (difference*difference);
    }
    else {
        difference = (standardDeviation[i] - mean);
        difference = (difference*difference);
    }
    variance = variance + difference;
}
printf("\r\nStandard deviation = %.5lf\r\n", sqrt(variance/arraySize));

```

Figure 19

4. Testing

4.1. Vector – Length

```
Vector length
If the length of a two dimensional vector is needed, enter '0' for position Z.
Points of first vector:
Enter the X position: 16
Enter the Y position: -4
Enter the Z position: 12
|a| = 20.39608
```

Testing vector length of the points: 16, -4, 12

Test: Successful

4.2. Vector – Normalising

```
Vector normalization
Have you found the length of a vector in the 'Vector Length' section of this program?
1. Yes
2. No
Enter answer: 2
Points of first vector:
Enter the X position: 19
Enter the Y position: 0
Enter the Z position: -10
Normalized vector = (0.88, 0.00, -0.47)
```

Testing vector normalisation of the points: 19, 0, -10

Test: Successful

4.3. Vector – Distance between two points

```
Distance between two points
Points of first vector:
Enter the X position: 15
Enter the Y position: -19
Enter the Z position: 2
Points of second vector:
Enter the X position: -51
Enter the Y position: -12
Enter the Z position: 4
Distance between vectors = 66.40030
Would you like to use the calculator or exit?
```

Testing vector distance of the points: 15, -19, 2 and -51, -12, 4

Test Successful

4.4. Vector – Dot Product

```
Dot product
Points of first vector:
Enter the X position: 10
Enter the Y position: -2
Enter the Z position: 25
Points of second vector:
Enter the X position: -31
Enter the Y position: 9
Enter the Z position: 0
A.B = -328.00
```

Testing vector dot product of points: 10, -2, 25 and -31, 9, 0

Test: Successful

4.5. Vector – Cross Product

```
Cross product
Points of first vector:
Enter the X position: 10
Enter the Y position: -10
Enter the Z position: 0
Points of second vector:
Enter the X position: 15
Enter the Y position: 14
Enter the Z position: -1
|AxB| = (10, 10, 290)
```

Testing vector cross product of points: 10, -10, 0 and 15, 14, -1

Test: Successful

4.6. Matrix

```
Matrix menu
Enter values for 4x4 matrix:
Enter value: 15
Enter value: 3
Enter value: -4
Enter value: 19

Enter value: 0
Enter value: 15
Enter value: -3
Enter value: 1

Enter value: 22
Enter value: -4
Enter value: 43
Enter value: 12

Enter value: -14
Enter value: 91
Enter value: 53
Enter value: 3

4x4:
  15   3   -4   19
   0  15   -3    1
  22  -4   43   12
 -14  91   53    3
```

```
Enter values for 4x1 matrix:
Enter value: 5
Enter value: -3
Enter value: 12
Enter value: 5

4x1:
  5
 -3
 12
 5

Multiplied matrix:
 113
 -76
 698
 308
```

Testing the multiplication of two matrices

Test: Successful

4.7. Statistics – Mean

```
How many numbers do you require: 6
Enter number: 15
Enter number: 12
Enter number: -2
Enter number: 19
Enter number: 28
Enter number: -43
Mean: 4.83
```

Testing the values for the mean: 15, 12, -2, 19, 28, -43

Test: Successful

4.8. Statistics – Median

```
How many numbers do you require: 6
Enter number: 15
Enter number: 68
Enter number: 43
Enter number: -2
Enter number: 1
Enter number: 4
-33686019 -2 1 4 15 43
```

Testing the numerical order of values: 15, 68, 43, -2, 1, 4

Test: Failed

4.9. Statistics – Mode

```
How many numbers do you require: 6
Enter number: 15
Enter number: 53
Enter number: 2
Enter number: 35
Enter number: 2
Enter number: 4
15 53 2 35 2 4
```

Testing the frequency of the values: 15, 53, 2, 35, 2, 4

Test: Failed

4.10. Statistics – Standard Deviation

```
How many numbers do you require: 6
Enter number: 99
Enter number: 23
Enter number: 3
Enter number: 2
Enter number: -2
Enter number: 44

Standard deviation = 35.40912
```

Testing the result of the standard deviation of values: 99, 23, 3, 2, -2, 44

Test: Successful

Math Library Report

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

64/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4



Comment 1
no comment

PAGE 5

PAGE 6

PAGE 7



Comment 2
use one of c / c++ 's sort functions instead.

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

RESEARCH-INFORMED LITERATURE

 You've clearly investigated the relevant maths/SE/programming principles required - some operations not complete unfortunately. Ref your work. Otherwise a very good attempt.

3

KNOWLEDGE AND UNDERSTANDING

 Evidence of good understanding of mathematical concepts and SE/ programming concepts - a few tasks incomplete.

9

ANALYSIS

 Good analysis of problem and synthesis of solutions. Some issues with some stats tests, Set operations missing.

9

PRACTICAL APPLICATION AND DEP.

 A good attempt- the majority of tasks have been completed and tested. Some issues with median and mode coding. Where are set operations? Union and Intersection incomplete. Evidence of good SE practice and coding concepts.

40

SKILLS FOR PROFESSIONAL PRAC.



Well structured report with evidence of good test documentation. Well done. Try to include refs to maths and coding principles here.

3