Maths for Software Development

# Contents

[Contents 2](#_Toc438168364)

[Introduction 4](#_Toc438168365)

[1.1 Length of a Vector 4](#_Toc438168366)

[1.2 Code – Length of a Vector 4](#_Toc438168367)

[1.3 Test Schedule – Length of a Vector 5](#_Toc438168368)

[2.1 Normalising a Vector 5](#_Toc438168369)

[2.2 Code - Normalising a Vector 6](#_Toc438168370)

[2.3 Test Schedule - Normalising a Vector 6](#_Toc438168371)

[3.1 Distance between two Points 7](#_Toc438168372)

[3.2 Code - Distance between two Points 7](#_Toc438168373)

[3.3 Test Schedule - Distance between two Points 8](#_Toc438168374)

[4.1 Dot product of two Vectors 8](#_Toc438168375)

[4.2 Code - Dot product of two Vectors 9](#_Toc438168376)

[4.3 Test Schedule - Dot product of two Vectors 9](#_Toc438168377)

[5.1 Multiply a (4x4) and a (4x4) Matrix 10](#_Toc438168378)

[5.2 Code - Multiply a (4x4) and a (4x4) Matrix 11](#_Toc438168379)

[5.3 Test Schedule - Multiply a (4x4) and a (4x4) Matrix 12](#_Toc438168380)

[6.1 Multiply a (4x4) Matrix and a (4x1) Column Matrix 13](#_Toc438168381)

[6.2 Code - Multiply a (4x4) Matrix and a (4x1) Column Matrix 14](#_Toc438168382)

[6.3 Test Schedule - Multiply a (4x4) Matrix and a (4x1) Column Matrix 15](#_Toc438168383)

[7. Stats - Mean, Median, Mode and Standard Deviation 16](#_Toc438168384)

[7.11 Mean 16](#_Toc438168385)

[7.12 Code – Mean 16](#_Toc438168386)

[7.13 Test Schedule – Mean 17](#_Toc438168387)

[7.21 Median 17](#_Toc438168388)

[7.22 Code – Median 18](#_Toc438168389)

[7.23 Test Schedule – Median 19](#_Toc438168390)

[7.31 Mode 20](#_Toc438168391)

[7.32 Code – Mode 20](#_Toc438168392)

[7.33 Test Schedule – Mode 22](#_Toc438168393)

[7.41 Standard Deviation 22](#_Toc438168394)

[7.42 Code – Standard Deviation 23](#_Toc438168395)

[7.43 Test Schedule – Standard Deviation 23](#_Toc438168396)

[Conclusion 24](#_Toc438168397)

[References 24](#_Toc438168398)

# Introduction

This report contains part of a Maths Library which includes several mathematical procedures including: Working with Vectors, Matrices and Statistics. Each procedure will be explained in depth. The procedure code will be included in full and will be tested using a test schedule to ensure expected results are output.

# 1.1 Length of a Vector

The length of a vector (|a|) is needed to be able to perform calculations using multiple vectors, such as addition, subtraction and multiplying (scaling). Magnitude is also a term used to identify the length of vector.

To calculate the length of a vector from a programming in 3d space perspective you would start by obtaining the tail point and head point of the vector. You would then need to use Pythagoras’ Theorem (3-dimensional in this case) using the obtained points, next we would need to find the square root of the answer, which results in the length of the vector [1].

Example: (using vector << 3, 4, 0 >>)

# 1.2 Code – Length of a Vector

// Procedure to calculate the length of a vector

proc float vectorLength( float $lengthX, float $lengthY, float $lengthZ )

{

// Use Pythagoras' Theorem

float $lengthResult = $lengthX \* $lengthX + $lengthY \* $lengthY + $lengthZ \* $lengthZ;

// Calculate the square root of the result

// Return the answer

return sqrt($lengthResult);

}

// Enter values of a vector to find the vector length

vectorLength(5, 2, 1);

# 1.3 Test Schedule – Length of a Vector

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Vector | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Insert Vector values into procedure “vectorLength”  2) Run procedure in Maya  3) Length of vector output | << 5, 2, 1 >> | 5.477 | 5.477 | Pass |
| 2 | 1) Insert Vector values into procedure “vectorLength”  2) Run procedure in Maya  3) Length of vector output | << 2, 3, 5 >> | 6.164 | 6.164 | Pass |
| 3 | 1) Insert Vector values into procedure “vectorLength”  2) Run procedure in Maya  3) Length of vector output | << 6, 2, 4 >> | 7.483 | 7.483 | Pass |
| 4 | 1) Insert Vector values into procedure “vectorLength”  2) Run procedure in Maya  3) Length of vector output | << 1, 2, 1 >> | 2.449 | 2.449 | Pass |

# 2.1 Normalising a Vector

A normalised vector (â) will always equal to 1 [2]. Normalised vectors can be used in multiple scenarios such as calculating the distance between two moving objects, where the vector length would change. Vectors can be easily scaled using a normalised version of the vector.

To find the normalised version of a vector you first need to find the length of the vector, once the length has been determined you would then divide each vector component (x, y and z in this case) and divide each value by the vectors length. This will result in the normalised vector value for each component.

Example: (using vector << 6, 3, 8 >>)

Length:

Normalise: â = ( 6 / 10.440307, 3 / 10.440307, 8 / 10.440307 )

= << 0.574696, 0.287348, 0.766261 >>

# 2.2 Code - Normalising a Vector

2.2 Code - Normalising a Vector  
// Procedure to find the normalization of a given vector

proc vector vectorNormalisation( vector $norm )

{

// Calculate the length of the vector using procedure 'vectorLength'

float $length = vectorLength($norm.x, $norm.y, $norm.z);

// Take each vector value x, y, z and divide each by the length

float $normX = $norm.x / $length;

float $normY = $norm.y / $length;

float $normZ = $norm.z / $length;

return << $normX, $normY, $normZ >>;

}

vectorNormalisation<< 6, 3, 8 >>;

# 2.3 Test Schedule - Normalising a Vector

On testing it was found that MEL rounds numbers to 6 decimal places, therefore it is possible that unity will not be reached in this programming language. However, the outcome is very close to unity (approximately around 0.9999~).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Vector | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Input vector values  into procedure “vectorNormalisation”  2) Run procedure in Maya  3) Length of vector output | <<1, 2, 3>> | <<  0.267261, 0.534522, 0.801784  >> | <<  0.267261, 0.534522, 0.801784  >> | Pass |
| 2 | 1) Input vector values  into procedure “vectorNormalisation”  2) Run procedure in Maya  3) Length of vector output | <<4, 2, 1>> | <<  0.872872, 0.436436, 0.218218  >> | <<  0.872872, 0.436436, 0.218218  >> | Pass |
| 3 | 1) Input vector values  into procedure “vectorNormalisation”  2) Run procedure in Maya  3) Length of vector output | <<1, 5, 3>> | <<  0.169031, 0.845154, 0.507093  >> | <<  0.169031, 0.845154, 0.507093  >> | Pass |
| 4 | 1) Input vector values  into procedure “vectorNormalisation”  2) Run procedure in Maya  3) Length of vector output | <<2, 1, 5>> | <<  0.365148, 0.182574, 0.912871  >> | <<  0.365148, 0.182574, 0.912871  >> | Pass |

# 3.1 Distance between two Points

The distance between two points is used to check such things as collision detection, i.e. is the objectOne close enough to be hitting the objectTwo.

To calculate the distance between two points you first need to subtract each value of the second point away from each value of the first point () [3]. Then use the resulting values to find the length of a vector, as stated in 1.1 of this report ( ).

Example: (Using p0 = <<3, 4, 0>> and p1 = <<2, 3, 1>>)

p3 =

=

=

= 1.732051

# 3.2 Code - Distance between two Points

// Procedure to find the distance between two Points

proc float vectorDistance( float $p0X, float $p0Y, float $p0Z, float $p1X, float $p1Y, float $p1Z )

{

// Subtract point 2 from point 1 for each component x, y, z

$p3X = $p1X - $p0X;

$p3Y = $p1Y - $p0Y;

$p3Z = $p1Z - $p0Z;

// Calculate the distance between both vectors using procedure 'vectorLength'

return vectorLength($p3X, $p3Y, $p3Z);

}

// Enter values of two vectors to find the distance between two points

vectorDistance(9, 3, 5, 4, 2, 8)

# 3.3 Test Schedule - Distance between two Points

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Vector | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Insert Vector values into procedure “vectorDistance”  2) Run procedure in Maya  3) Distance between two points output | << 9, 3, 5 >>  << 4, 2, 8 >> | 5.91608 | 5.91608 | Pass |
| 2 | 1) Insert Vector values into procedure “vectorDistance”  2) Run procedure in Maya  3) Distance between two points output | << 2, 1, 3 >>  << 4, 3, 1 >> | 3.464102 | 3.464102 | Pass |
| 3 | 1) Insert Vector values into procedure “vectorDistance”  2) Run procedure in Maya  3) Distance between two points output | << 5, 1, 1 >>  << 0, 3, 4 >> | 6.164414 | 6.164414 | Pass |
| 4 | 1) Insert Vector values into procedure “vectorDistance”  2) Run procedure in Maya  3) Distance between two points output | << 1, 2, 4 >>  << 1, 3, 3 >> | 1.414214 | 1.414214 | Pass |

# 4.1 Dot product of two Vectors

Dot product is useful in games programming to find such things like if a player character is in the line of site of an enemy by finding how much the vectors are pointing in the same direction [4].

To find the dot product, we need to multiply each component of each vector and then add them together [5].

Example: (Using Vectors <<1, 2, 5>> and <<2, 1, 3>>)

If the result is a positive scalar, the polygon / enemy is facing the screen / player.

# 4.2 Code - Dot product of two Vectors

// Procedure to find the dot product of two vectors

proc float vectorDot( float $dotP0X, float $dotP0Y, float $dotP0Z, float $dotP1X, float $dotP1Y, float $dotP1Z )

{

// Multiply the first vector by the second vector

// Add each above answer together

return ( $dotP0X \* $dotP1X + $dotP0Y \* $dotP1Y + $dotP0Z \* $dotP1Z );

}

vectorDot(1, 2, 5, 2, 1, 3)

# 4.3 Test Schedule - Dot product of two Vectors

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Vector | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Insert Vector values into procedure “vectorDot”  2) Run procedure in Maya  3) Dot product of two vectors output | << 2, 3, 1 >>  << 1, 5, 3 >> | 20 | 20 | Pass |
| 2 | 1) Insert Vector values into procedure “vectorDot”  2) Run procedure in Maya  3) Dot product of two vectors output | << 9, 3, 3 >>  << 10, 3, 1 >> | 102 | 102 | Pass |
| 3 | 1) Insert Vector values into procedure “vectorDot”  2) Run procedure in Maya  3) Dot product of two vectors output | << 1, 4, 1 >>  << 4, 3, 2 >> | 18 | 18 | Pass |
| 4 | 1) Insert Vector values into procedure “vectorDot”  2) Run procedure in Maya  3) Dot product of two vectors output | << 0, 2, 7 >>  << 13, 1, 0 >> | 2 | 2 | Pass |

# 5.1 Multiply a (4x4) and a (4x4) Matrix

We use matrix multiplication since graphics cards are designed to calculate a matrix multiplication in one clock cycle, rather than the multiple clock cycles it would take to use simultaneous equations.

“Two matrices can be multiplied together only when the number of columns in the first is equal to the number of rows in the second.” [6]

To multiply a (4x4) matrix by another (4x4) matrix, you will need to multiply each value in the first row of matrix1 with each value in the first column of matrix2, then the first row of matrix1 by the second, third and fourth columns of matrix2 to find the dot product of the values.

Example:

Multiply first row value from matrix1 by first column value from matrix2 and continue with the first row of matrix1 for all columns of matrix2:

matrix1 matrix2

\* =

\* =

Add together to get the first row of the resulting matrix as 12, 7, 20 and 0. Repeat above steps using the second row of matrix1, and continue onwards with the third and fourth of matrix1 to get the result below:

\* =

# 5.2 Code - Multiply a (4x4) and a (4x4) Matrix

//procedure to Multiply a (4x4) and a (4x4) Matrix

proc string multiply4By4(float $matrix1[], float $matrix2[])

{

// Multiplied Matrix result

float $matrix3[16];

// Row 1 of resulting Matrix

$matrix3[0] = ($matrix1[0] \* $matrix2[0]) + ($matrix1[1] \* $matrix2[4]) + ($matrix1[2] \* $matrix2[8]) + ($matrix1[3] \* $matrix2[12]);

$matrix3[1] = ($matrix1[0] \* $matrix2[1]) + ($matrix1[1] \* $matrix2[5]) + ($matrix1[2] \* $matrix2[9]) + ($matrix1[3] \* $matrix2[13]);

$matrix3[2] = ($matrix1[0] \* $matrix2[2]) + ($matrix1[1] \* $matrix2[6]) + ($matrix1[2] \* $matrix2[10]) + ($matrix1[3] \* $matrix2[14]);

$matrix3[3] = ($matrix1[0] \* $matrix2[3]) + ($matrix1[1] \* $matrix2[7]) + ($matrix1[2] \* $matrix2[11]) + ($matrix1[3] \* $matrix2[15]);

// Row 2 of resulting Matrix

$matrix3[4] = ($matrix1[4] \* $matrix2[0]) + ($matrix1[5] \* $matrix2[4]) + ($matrix1[6] \* $matrix2[8]) + ($matrix1[7] \* $matrix2[12]);

$matrix3[5] = ($matrix1[4] \* $matrix2[1]) + ($matrix1[5] \* $matrix2[5]) + ($matrix1[6] \* $matrix2[9]) + ($matrix1[7] \* $matrix2[13]);

$matrix3[6] = ($matrix1[4] \* $matrix2[2]) + ($matrix1[5] \* $matrix2[6]) + ($matrix1[6] \* $matrix2[10]) + ($matrix1[7] \* $matrix2[14]);

$matrix3[7] = ($matrix1[4] \* $matrix2[3]) + ($matrix1[5] \* $matrix2[7]) + ($matrix1[6] \* $matrix2[11]) + ($matrix1[7] \* $matrix2[15]);

// Row 3 of resulting Matrix

$matrix3[8] = ($matrix1[8] \* $matrix2[0]) + ($matrix1[8] \* $matrix2[4]) + ($matrix1[8] \* $matrix2[8]) + ($matrix1[8] \* $matrix2[12]);

$matrix3[9] = ($matrix1[8] \* $matrix2[1]) + ($matrix1[9] \* $matrix2[5]) + ($matrix1[10] \* $matrix2[9]) + ($matrix1[11] \* $matrix2[13]);

$matrix3[10] = ($matrix1[8] \* $matrix2[2]) + ($matrix1[9] \* $matrix2[6]) + ($matrix1[10] \* $matrix2[10]) + ($matrix1[11] \* $matrix2[14]);

$matrix3[11] = ($matrix1[8] \* $matrix2[3]) + ($matrix1[9] \* $matrix2[7]) + ($matrix1[10] \* $matrix2[11]) + ($matrix1[11] \* $matrix2[15]);

// Row 4 of resulting Matrix

$matrix3[12] = ($matrix1[12] \* $matrix2[0]) + ($matrix1[13] \* $matrix2[4]) + ($matrix1[14] \* $matrix2[8]) + ($matrix1[15] \* $matrix2[12]);

$matrix3[13] = ($matrix1[12] \* $matrix2[1]) + ($matrix1[13] \* $matrix2[5]) + ($matrix1[14] \* $matrix2[9]) + ($matrix1[15] \* $matrix2[13]);

$matrix3[14] = ($matrix1[12] \* $matrix2[2]) + ($matrix1[13] \* $matrix2[6]) + ($matrix1[14] \* $matrix2[10]) + ($matrix1[15] \* $matrix2[14]);

$matrix3[15] = ($matrix1[12] \* $matrix2[3]) + ($matrix1[13] \* $matrix2[7]) + ($matrix1[14] \* $matrix2[11]) + ($matrix1[15] \* $matrix2[15]);

// Print resulting matrix of matrix 1 \* matrix 2

print("\nResult of: Multiply a (4x4) and a (4x4) Matrix\n");

string $row1 = print($matrix3[0] + "\t" + $matrix3[1] + "\t" + $matrix3[2] + "\t" + $matrix3[3] + "\n");

string $row2 = print($matrix3[4] + "\t" + $matrix3[5] + "\t" + $matrix3[6] + "\t" + $matrix3[7] + "\n");

string $row3 = print($matrix3[8] + "\t" + $matrix3[9] + "\t" + $matrix3[10] + "\t" + $matrix3[11] + "\n");

string $row4 = print($matrix3[12] + "\t" + $matrix3[13] + "\t" + $matrix3[14] + "\t" + $matrix3[15] + "\n");

// Store result Matrix

string $printer = print($row1 + $row2 + $row3 + $row4)

return $printer;

}

float $matrix1[16] = {2, 2, 3, 1, 3, 2, 4, -2, 1, 0, 1, 2, 3, 2, 0, 1};

float $matrix2[16] = {1, 2, 1, -4, 3, 1, 0, 0, 2, 1, 1, 1, 3, 4, -1, 1};

multiply4By4($matrix1, $matrix2)

# 5.3 Test Schedule - Multiply a (4x4) and a (4x4) Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Matrices | Expected  Result | Actual  Result | Pass / Fail |
| 1 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By4”  2) Run procedure in Maya  3) (4x4)\*(4x4) matrix multiplication output | matrix1  matrix2 |  |  | Pass |
| 2 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By4”  2) Run procedure in Maya  3) (4x4)\*(4x4) matrix multiplication output | matrix1  matrix2 |  |  | Pass |
| 3 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By4”  2) Run procedure in Maya  3) (4x4)\*(4x4) matrix multiplication output | matrix1  matrix2 |  |  | Pass |
| 4 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By4”  2) Run procedure in Maya  3) (4x4)\*(4x4) matrix multiplication output | matrix1  matrix2 |  |  | Pass |

# 6.1 Multiply a (4x4) Matrix and a (4x1) Column Matrix

Multiplying a (4x4) matrix by a (4x1) column matrix is approached the same way as section `5.1 Multiply a (4x4) and a (4x4) Matrix`. However the resulting matrix will be a (4x1) column matrix.

Example:

\* =

\* =

# 6.2 Code - Multiply a (4x4) Matrix and a (4x1) Column Matrix

// procedure to Multiply a (4x4) and a (4x1) Matrix

proc string multiply4By1(float $matrix1[], float $matrix2[])

{

// Multiplied Matrix result

float $matrix3[4];

// Multiply Matrices together

$matrix3[0] = ($matrix1[0] \* $matrix2[0]) + ($matrix1[1] \* $matrix2[1]) + ($matrix1[2] \* $matrix2[2]) + ($matrix1[3] \* $matrix2[3]);

$matrix3[1] = ($matrix1[4] \* $matrix2[0]) + ($matrix1[5] \* $matrix2[1]) + ($matrix1[6] \* $matrix2[2]) + ($matrix1[7] \* $matrix2[3]);

$matrix3[2] = ($matrix1[8] \* $matrix2[0]) + ($matrix1[9] \* $matrix2[1]) + ($matrix1[10] \* $matrix2[2]) + ($matrix1[11] \* $matrix2[3]);

$matrix3[3] = ($matrix1[12] \* $matrix2[0]) + ($matrix1[13] \* $matrix2[1]) + ($matrix1[14] \* $matrix2[2]) + ($matrix1[15] \* $matrix2[3]);

// Print resulting matrix of matrix1 \* matrix2

string $printer = print($matrix3[0] + "\n" + $matrix3[1] + "\n" + $matrix3[2] + "\n" + $matrix3[3] + "\n");

return $printer;

}

float $matrix1[16] = {3, 1, 1, 0, 3, 0, 2, 0, 1, 2, 5, 0, 0, 0, 0, 1};

float $matrix2[4] = {4, 1, 3, 1};

multiply4By1($matrix1, $matrix2)

# 6.3 Test Schedule - Multiply a (4x4) Matrix and a (4x1) Column Matrix

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Matrices | Expected  Result | Actual  Result | Pass / Fail |
| 1 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By1”  2) Run procedure in Maya  3) (4x4)\*(4x1) matrix multiplication output | matrix1  matrix2 |  |  | Pass |
| 2 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By1”  2) Run procedure in Maya  3) (4x4)\*(4x1) matrix multiplication output | matrix1  matrix2 |  |  | Pass |
| 3 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By1”  2) Run procedure in Maya  3) (4x4)\*(4x1) matrix multiplication output | matrix1  matrix2 |  |  | Pass |
| 4 | 1) Insert matrix1 and matrix2 values into procedure “multiply4By1”  2) Run procedure in Maya  3) (4x4)\*(4x1) matrix multiplication output | matrix1  matrix2 |  |  | Pass |

# 7. Stats - Mean, Median, Mode and Standard Deviation

Statistics are used to find a certain aspect of a data set, numbers in this case.

## 7.11 Mean

Mean is to find the average of the data set by totalling up all numbers involved and then dividing by the amount of numbers there were to add together at the start [7] [8].

Example: (Using number 1, 4, 7 and 9)

## 7.12 Code – Mean

// Procedure to calculate Mean

proc float mean(float $input[])

{

int $i = 0.0;

float $mean = 0.0;

// Find amount of numbers entered

int $size = size($input);

// Loop to add each number together

for($i; $i < $size; $i++)

{

$mean = $mean + $input[$i];

}

// Divide total number by amount of numbers

$mean = $mean / $size;

// Print result

print("\nMean:\n");

return $mean;

}

float $input[] = {1.3, 7.1, 5.6};

mean($input);

## 7.13 Test Schedule – Mean

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Values | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Insert values into procedure “mean”  2) Run procedure in Maya  3) Mean is output | {3.1, 0.3, 4.2, 5.7, 2.22, 10.5, 6.1} | 4.588571 | 4.588571 | Pass |
| 2 | 1) Insert values into procedure “mean”  2) Run procedure in Maya  3) Mean is output | {18.2, 15.1, 22.9, 14.6, 11.0, 5.1, 4.1, 0.03} | 11.37875 | 11.37875 | Pass |
| 3 | 1) Insert values into procedure “mean”  2) Run procedure in Maya  3) Mean is output | {4.0, 1.22, 11.4, 21.3, 9.8} | 9.544 | 9.544 | Pass |
| 4 | 1) Insert values into procedure “mean”  2) Run procedure in Maya  3) Mean is output | {0.01, 7.23, 4.6, 0.07, 1.5, -68.1} | -9.115 | -9.115 | Pass |

## 7.21 Median

The median of a list of numbers requires that the list be sorted in order from smallest to largest first so that the middle value can be determined. This middle value will become the median.

Once the numbers are sorted you must first check to see if the list is an even amount of values, if so then the two middle values must be taken and the mean found from those two values, the result is the median. If the amount of values is not even, then simply the middle number is median value [9] [8].

Example: odd amount of numbers: 1, 2, 3, 4, 5, 6, 7 - Median = 4

Even amount of numbers: 1 2 3 4 - Mean of 2 and 3 - Median = 2.5

## 7.22 Code – Median

// Procedure to calculate Median

proc float median(float $input[])

{

int $i;

int $j;

float $store;

float $median;

// Find amount of numbers entered

float $size = size($input);

// Sort number in order from smallest first

for($i; $i < $size; $i++)

{

for($j = 0; $j < $size - 1; $j++)

{

// If current value is larger than next, swap values

if($input[$j] >= $input[$j + 1])

{

$store = $input[$j];

$input[$j] = $input[$j + 1];

$input[$j + 1] = $store;

}

}

}

// Check if amount of values entered is even

// If even find two middle numbers and calculate mean

if($size % 2 == 0)

{

// Find midpoint of amount of numbers entered

int $store = ($size - 1) / 2;

// add midpoint number and next higher number

$median = $input[$store] + $input[$store + 1];

// calculate the mean(average) of the two numbers

$median = $median / 2.0;

}

// Else assign middle number to median value

else

{

// Find midpoint of amount of numbers entered

int $store = ($size - 1) / 2;

// Apply final answer to median value

$median = $input[$store];

}

return $median;

}

// Enter number to find the Median

// A comma between each number is required

float $input[] = {1.1, 10.0, 4.3, 0.3, 0.6, 7.1, 3.2, 3.3, 3.2};

median($input);

## 7.23 Test Schedule – Median

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Values | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Insert values into procedure “median”  2) Run procedure in Maya  3) Median is output | {3.1, 0.3, 4.2, 5.7, 2.22, 10.5, 6.1} | 4.2 | 4.2 | Pass |
| 2 | 1) Insert values into procedure “median”  2) Run procedure in Maya  3) Median is output | {18.2, 15.1, 22.9, 14.6, 11.0, 5.1, 4.1, 0.03} | 12.8 | 12.8 | Pass |
| 3 | 1) Insert values into procedure “median”  2) Run procedure in Maya  3) Median is output | {4.0, 1.22, 11.4, 21.3, 9.8} | 9.8 | 9.8 | Pass |
| 4 | 1) Insert values into procedure “median”  2) Run procedure in Maya  3) Median is output | {0.01, 7.23, 4.6, 0.07, 1.5, -68.1} | 0.785 | 0.785 | Pass |

## 7.31 Mode

The mode is the value that occurs the most frequently in a data set.

To work out the mode of a set of values, you have to put the values in order, and then count how many values are identical and output the value that reoccurs the highest amount of times [10] [8].

Example: 1, 1, 4, 4, 7, 7, 7, 8, 9 – mode = 7

## 7.32 Code – Mode

// Procedure to find the mode

proc string mode(float $input[])

{

int $i = 0;

int $j = 0;

int $count1 = 1;

int $count2 = 1;

float $mode = $input[0];

float $store;

// Find amount of numbers entered

float $size = size($input);

// Sort number in order from smallest first

for($i; $i < $size; $i++)

{

for($j = 0; $j < $size - 1; $j++)

{

// If current value is larger than next, swap values

if($input[$j] >= $input[$j + 1])

{

$store = $input[$j];

$input[$j] = $input[$j + 1];

$input[$j + 1] = $store;

}

}

}

// For loop to count and store if number has more than one value

$i = 0;

for($i; $i < $size; $i++)

{

// Check if current value is the same as next value

if($input[$i] == $input[$i + 1])

{

// Add 1 to counter if the numbers are the same

$count1 = $count1 + 1;

// If amount of identical numbers is greater than last number checked

if($count1 > $count2)

{

// set $count2 to higher value

$count2 = $count1;

// Set mode as current array value

$mode = $input[$i];

}

}

// Reset counter

else

$count1 = 1;

}

// Check to see if there was a valid mode

if($count2 == 1)

{

// Assign mode display the below message

$mode = print("This set of numbers has no mode. Each number occurs only once.\n");

}

$printer = $mode;

return $printer;

}

float $input[] = {10.2, 1.1, 1.1, 1.1, 0.3, 1.3, 0.3, 0.6, 3.5, 3.6};

mode($input);

## 7.33 Test Schedule – Mode

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Values | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Insert values into procedure “mode”  2) Run procedure in Maya  3) Mode is output | {5.7, 3.1, 6.1, 0.3, 4.2, 5.7, 2.22, 10.5, 6.1, 5.7} | 5.7 | 5.7 | Pass |
| 2 | 1) Insert values into procedure “mode”  2) Run procedure in Maya  3) Mode is output | {18.2, 5.1, 0.03, 0.03, 15.1, 22.9, 14.6, 11.0, 5.1, 4.1, 0.03} | 0.03 | 0.03 | Pass |
| 3 | 1) Insert values into procedure “mode”  2) Run procedure in Maya  3) Mode is output | {4.0, 1.22, 11.4, 21.3, 9.8, 4.0} | 4 | 4 | Pass |
| 4 | 1) Insert values into procedure “mode”  2) Run procedure in Maya  3) Mode is output | {-68.1, 0.01, 7.23, 4.6, 0.07, -68.1, 1.5, 4.6, -68.1} | -68.1 | -68.1 | Pass |

## 7.41 Standard Deviation

Standard deviation is a way to work out how spread apart from each other your values are.

First you need to work out the variance between the values. This uses the following method. Find the mean of the values, subtract the mean and square the result for each value, then finally find the mean of this figure, this will result in the variance of the values. The final step is to find the square root of the variance, and hence the Standard Deviation [11] [12].

Example: (using 10, 4, 3, 6 and 9)

Find the Mean:

Then subtract the mean from each value and square the result:

Then add values up and find mean from the result:

Square root the result to find standard deviation:

## 7.42 Code – Standard Deviation

// Procedure to calculate Standard Deviation

proc float standardDeviation(float $input[])

{

int $i = 0;

// Find amount of numbers entered

int $size = size($input);

// Find the mean

float $StanDev = mean($input);

// Find the variance

for($i; $i < $size; $i++)

{

// Take away the mean from each value to find the difference

$input[$i] = $input[$i] - $StanDev;

// Square each value's result

$input[$i] = $input[$i] \* $input[$i];

}

// Find the mean and squareroot

$StanDev = sqrt(mean($input));

return $StanDev;

}

float $input[] = {10.2, 0.4, 1.3, 0.6};

standardDeviation($input);

## 7.43 Test Schedule – Standard Deviation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | Values | Expected  Result | Actual  Result | Pass /  Fail |
| 1 | 1) Insert values into procedure “standardDeviation”  2) Run procedure in Maya  3) Standard Deviation is output | {3.1, 0.3, 4.2, 5.7, 2.22, 10.5, 6.1} | 3.043341 | 3.043341 | Pass |
| 2 | 1) Insert values into procedure “mean”  2) Run procedure in Maya  3) Mean is output | {18.2, 15.1, 22.9, 14.6, 11.0, 5.1, 4.1, 0.03} | 7.28726 | 7.28726 | Pass |
| 3 | 1) Insert values into procedure “mean”  2) Run procedure in Maya  3) Mean is output | {4.0, 1.22, 11.4, 21.3, 9.8} | 6.953254 | 6.953254 | Pass |
| 4 | 1) Insert values into procedure “mean”  2) Run procedure in Maya  3) Mean is output | {0.01, 7.23, 4.6, 0.07, 1.5, -68.1} | 26.504083 | 26.504083 | Pass |

# Conclusion

This report has looked at how to create part of a Maths Library which includes several mathematical procedures including: Working with Vectors, Matrices and Statistics. This report was aimed at creating functions to be used in relation to games programming. All test schedules were a pass, so all code has been proven to output the correct values.

# References

|  |  |
| --- | --- |
| [1] | K. A. Stroud and D. J. Booth, Engineering Mathematics, 7th ed., Hampshire: Palgrave Macmillan, 2013, p. 531. |
| [2] | “Unit Vector,” 2014. [Online]. Available: https://www.mathsisfun.com/algebra/vector-unit.html. [Accessed 14 December 2015]. |
| [3] | E. Stapel, “The Distance Formula,” Purplemath, 2012. [Online]. Available: http://www.purplemath.com/modules/distform.htm. [Accessed 14 December 2015]. |
| [4] | D. Rosen, “Linear algebra for game developers ~ part 2,” 3 July 2009. [Online]. Available: http://blog.wolfire.com/2009/07/linear-algebra-for-game-developers-part-2/. [Accessed 14 December 2015]. |
| [5] | K. A. Stroud and D. J. Booth, Engineering Mathematics, 7th ed., Hampshire: Palgrave Macmillan, 2013, p. 534. |
| [6] | K. A. Stroud and D. J. Booth, Engineering Mathematics, 7th ed., Hampshire: Palgrave Macmillan, 2013, p. 493. |
| [7] | R. Pierce, “How to Calculate the Mean Value,” Math Is Fun, 21 August 2015. [Online]. Available: http://www.mathsisfun.com/mean.html. [Accessed 16 December 2015]. |
| [8] | E. Furey, “Mean Median Mode Calculator,” calculatorsoup.com, [Online]. Available: http://www.calculatorsoup.com/calculators/statistics/mean-median-mode.php. [Accessed 16, 17, 18 December 2015]. |
| [9] | R. Pierce, “How to Find the Median Value,” Math Is Fun, 10 June 2015. [Online]. Available: http://www.mathsisfun.com/median.html. [Accessed 17 December 2015]. |
| [10] | K. A. Stroud and D. J. Booth, Engineering Mathematics, 7th ed., Hampshire: Palgrave Macmillan, 2013, p. 1059. |
| [11] | R. Pierce, “Standard Deviation and Variance,” Math Is Fun, 5 June 2014. [Online]. Available: http://www.mathsisfun.com/data/standard-deviation.html. [Accessed 18 December 2015]. |
| [12] | R. Pierce, “Standard Deviation Calculator,” Math Is Fun, 13 September 2015. [Online]. Available: http://www.mathsisfun.com/data/standard-deviation-calculator.html. [Accessed 18 December 215]. |