**MATRIX.h User Guide**

This document is sure to join the pantheon of exquisite works which embody the resilience, brilliance, and dare I say hubris, of the human spirit. I dedicate this very special work to my wife without whom this masterpiece would forever remain an unexplored and suppressed part of my vast intellect. I also dedicate this work to my future posterity (may it be numerous, but not too much). Finally, I would like to give you, fortunate reader, a word of caution: beyond the obvious risk of being dazed by the brilliance of my work, I have taken no care at all in avoiding memory leaks. This means you may use this class at your own risks. Let us begin.

**Creating a matrix**

**From a std::vector<>:**

The matrix object can be initialized with a 1D or 2D vector of type int, float, or double like so:

std::vector<std::vector<double>> myvector;

mat mymatrix(myvector);

Note that the vector must not be jagged. Using a jagged 2D vector will cause segmentation faults in other member functions.

**Identity matrix:**

The identity matrix I of size m x m can be generated in the following manner:

mat Identity(“I”, m);

mat Identity(“I”, m, m);

Note the second method requires the second and third arguments ‘m’ to be of the same value.

**Matrix of size m x n with all zeroes**

mat zeroMat(“zeros”, m, n);

**Matrix of size m x n with all values as double ‘a’**

mat aMat(a, m, n);

**Matrix of size m x n with all values uniformly randomly generated between double values ‘a’ and ‘b’**

mat randMat(“rand”, a, b, m, n);

**Matrix of size m x n with all values normally randomly generated between double values ‘a’ and ‘b’**

mat randMat(“randN”, a, b, m, n);

**Empty matrix (default constructor)**

mat myMat();

mat myMat;

Note this is also used when storing the result of a matrix operation, as will be demonstrated in the next sections.

**Matrix operations**

This section will assume the existence of 2 matrices: A and B, as well as a scalar k. The result of matrix operations will be stored in a new matrix C;

**Addition/Broadcasting::**

mat C = A + B;

Broadcasting requires B to be one dimensional and must match either the column or row number of A

**Subtraction/Broadcasting::**

mat C = A - B;

Broadcasting requires B to be one dimensional and must match either the column or row number of A

**Scalar multiplication:**

mat C = A \* k;

OR

mat C = k \* A ;

**Scalar division:**

mat C = A / k ;

**Vector division/Broadcasting:**

mat C = A / B ;

Broadcasting requires B to be one dimensional and must match either the column or row number of A

**Matrix multiplication:**

mat C = A \* B;

**Vector broadcasting/ Hadamard Product:**

mat C = A ^ B;

**Matrix comparison:**

A == B

Will return true or false depending on if the vectors contained in the matrix objects of A and B are identical or not.

**Assignment:**

A = B;

Will assign all member values of matrix B to matrix A.

**Indexing:**

A(i,j);

Will return the matrix value in row ‘i’ and column ‘j’. Note that because this class is built with std::vector, the matrix are also zero indexed.

**Utilities and other member functions**

**Matrix transpose:**

A.T();

Will return the transpose of matrix A.

**Accessor for rows and columns:**

A.rows();

A.cols();

Will return the number of rows and columns of Matrix A respectively.

**Print Matrix:**

A.print();

Will print matrix A to the console. If the matrix has more than 20 columns, it will only display the first 20 columns.

**Sum matrix rows:**

A.sum(“rows”)

Will sum all the elements in the matrix rows and return a 1D matrix of those sums.

**Sum matrix columns:**

A.sum(“cols”)

Will sum all the elements in the matrix columns and return a 1D matrix of those sums.

**Sum matrix:**

A.sum()

Will sum all the elements in the matrix and return a type double.

**Minimum element:**

A.min()

Returns the smallest element of A.

**Maximum element:**

A.max()

Returns the largest element of A.

**Exponential:**

A.xp()

Returns a matrix with the e^(element) of every element of A.