

# MTX - 3.4

## C ANSI/C++ BLAS-3 API

Developed by Eng. Juan Camilo Gómez Cadavid MSc.

Macro/Function/Keyword	Description	MATLAB Eq.
Generation		
<b>matrix</b> VAR=NULL <b>mtxdef</b> (VAR)	Matrix MTX type definition Keyword. <b>matrix</b> keyword, defines unallocated variables. Always initialize matrix-type MTX variables to NULL Use <b>mtxdef</b> to define an allocated empty MATRIX by default.	VAR=[];
<b>M=mtx_new</b> (n,m)	Returns a matrix $M$ of $m \times n$ dimensions (initialized on 0)	M=zeros(n,m)
<b>M=mtx_init</b> (n,m,initval)	Returns a matrix $M$ of $m \times n$ dimensions initialized on <i>initval</i>	M=initval*ones(n,m)
<b>M=mtx_rand</b> (n,m)	Returns a matrix $M$ of $m \times n$ dimensions containing pseudorandom values drawn from the standard uniform distribution on the open interval (0,1)	M=rand(n,m)
<b>M=mtx_1dtomtx</b> (A)	Allocates a MTX matrix from the 1D array $A$	
<b>M=mtx_2dtomtx</b> (A)	Allocates a MTX matrix from the 2D array $A$	
<b>M=mtx_eye</b> (n,alpha)	Returns the $n \times n$ identity matrix $M = \alpha I$ . $\alpha$ is scalar and $I$ represents the identity matrix	M=alpha*eye(n)
<b>mtx_del</b> (M)	Releases the specified block of memory generated by M, back to the heap	clear M
<b>M=mtx_cpy</b> (A)	Generates a copy of matrix $A$ into $M$ .	M=A
<b>mtx_memcpy</b> (M,A)	Generates a copy of matrix $A$ into $M$ . (M is allocated, and has the same dimensions of $A$ )	M=A
Indexing		
<b>x=A-&gt;pos</b> [r][c]	Get the element at index <i>row-r</i> and <i>column-c</i>	x=A(r,c)
<b>A-&gt;pos</b> [r][c] = value	Set a single element at index <i>row-r</i> and <i>column-c</i>	A(r,c)=value
<b>M=mtx_getsubset</b> (A,r1,r2,c1,c2)	Get submatrix $M \leftarrow A$ ( $M$ from $A$ ), from rows $r1$ to $r2$ and columns $c1$ to $c2$	A(r1:r2,c1:c2)
<b>M=mtx_getrow</b> (A,r)	Get the $r$ -row from matrix $A$	M=A(r,:)
<b>M=mtx_getcol</b> (A,c)	Get the $c$ -column from matrix $A$	M=A(:,c)
<b>M=mtx_getrows</b> (A,r1,r2)	Get rows from matrix $A$ . From rows $r1$ to $r2$	M=A(r1:r2,:)
<b>M=mtx_getcols</b> (A,c1,c2)	Get columns from matrix $A$ . From columns $c1$ to $c2$	M=A(:,c1:c2)
<b>mxt_setsubset</b> (M,A,r1,r2,c1,c2)	Put submatrix $A$ into $M$ from row $f1$ to row $f2$ and cols $c1$ to col $c2$	M(r1:r2,c1:c2)=A

<b>mtx_setrow(M,A,r)</b>	Set the $r$ -row from matrix $M$ , with the row vector $A$	$M(r,:)=A$
<b>mtx_setcol(M,A,c)</b>	Set the $c$ -column from matrix $M$ , with the column vector $A$	$M(:,c)=A$
<b>M=mtx_vec(A)</b>	Return the matrix $A$ as single column vector	$M=A(:)$
<b>Basic information</b>		
<b>x=mtx_isempty(M)</b>	TRUE if $M$ is an empty matrix	<b>isempty(M)</b>
<b>x=mtx_isrow(M)</b>	TRUE if $M$ is a row vector	<b>isrow(M)</b>
<b>x=mtx_iscolumn(M)</b>	TRUE if $M$ is a column vector	<b>iscolumn(M)</b>
<b>x=mtx_isvector(M)</b>	TRUE if $M$ is vector	<b>isvector(M)</b>
<b>x=mtx_numel(M)</b>	Number of elements in matrix $M$	<b>x=numel(M)</b>
<b>x=mtx_length(M)</b>	Largest matrix dimension of $M$	<b>x=length(M)</b>
<b>x=mtx_det(M)</b>	Determinant of matrix $M$	<b>x=det(M)</b>
<b>x=mtx_trace(M)</b>	Trace of matrix $M$	<b>x=trace(M)</b>
<b>D=mtx_diag(M)</b>	Returns a column vector with all diagonal elements of $M$	<b>D=diag(M)</b>
<b>M=mtx_produ(A)</b>	Product of matrix elements - by columns	<b>M=prod(A)</b>
<b>x=mtx_cumprod(A)</b>	Total product of matrix elements	<b>x=prod(A(:))</b>
<b>M=mtx_sum(A)</b>	Sum of matrix elements - by columns	<b>M=sum(A)</b>
<b>x=mtx_cumsum(A)</b>	Total sum of matrix elements	<b>x=sum(A(:))</b>
<b>M=mtx_mean(A)</b>	Mean of matrix elements – by columns	<b>M=mean(A)</b>
<b>M=mtx_max(A)</b>	If $A$ is a vector, returns the largest element in $A$ . If $A$ is a matrix, treats the columns of $A$ as vectors, returning a row vector containing the maximum element from each column.	<b>M=max(A)</b>
<b>x=mtx_cummax(A)</b>	Largest element in matrix	<b>x=max(A(:))</b>
<b>M=mtx_min(A)</b>	If $A$ is a vector, returns the smallest element in $A$ . If $A$ is a matrix, treats the columns of $A$ as vectors, returning a row vector containing the smallest element from each column.	<b>M=min(A)</b>
<b>x=mtx_cummin(A)</b>	Smallest element in matrix	<b>x=min(A(:))</b>
<b>BLAS-Operations</b>		
<b>M=mtx_t(A)</b>	Returns the transpose of $A$	<b>M=A'</b>
<b>M=mtx_gadd(alpha,A,beta,B)</b>	Returns $M=\alpha A + \beta B$ $\alpha$ and $\beta$ are scalars, and $A$ and $B$ matrices.	<b>M=alpha*A+beta*B</b>

<b>M=mtx_prod(alpha,A,B)</b>	Returns matrix Multiplication $M=\alpha AB$ $\alpha$ is scalar, $A$ and $B$ matrices.	$M=\alpha A*B$
<b>ret=mtx_dgemm(alpha,A,TA,B,TB,beta,C)</b>	Computes $C = \alpha AB + \beta C$ and related operations. $\alpha$ and $\beta$ are scalars, and $A$ , $B$ and $C$ are matrices. $TA = \text{'T' or 't'}$ $\rightarrow A$ is considered transposed ( $A^T$ ). $TB = \text{'T' or 't'}$ $\rightarrow B$ is considered transposed ( $B^T$ ). On success, returns (0) and matrix $C$ is overwritten, otherwise returns (-1).	$C = \alpha A*B + \beta C$ $C = \alpha A'*B + \beta C$ $C = \alpha A*B' + \beta C$ $C = \alpha A'*B' + \beta C$
<b>M=mtx_kron(A,B)</b>	Returns the Kronecker tensor product of matrices $A$ and $B$ , $M=A\oplus B$ . If $A$ is an m-by-n matrix and $B$ is a p-by-q matrix, then $\text{mtx\_kron}(A,B)$ is an m*p-by-n*q matrix formed by taking all possible products between the elements of $A$ and the matrix $B$ .	$M=\text{kron}(A,B)$
<b>M=mtx_powui(A,n)</b>	Returns $A^n$ (Matrix power - n must be an unsigned int and $A$ an square matrix)	$M=A^n$
<b>M=mtx_ptpprod(A,B)</b>	Returns $A.B$ (point by point product) ( $A$ and $B$ must have the same dimensions)	$M=A.*B$
<b>M=mtx_koper(A,Op,beta)</b>	Returns standard scalar matrix operations. $C = A(Op)\beta$ Related operations (Op) = '+', '-', '*', '/' and '^' (power) Example: <b>M=mtx_koper(A,'+',beta);</b>	$M=A+\beta$ $M=A-\beta$ $M=A*\beta$ $M=A/\beta$ $M=A^\beta$
<b>M=mtx_inv(A)</b>	Returns $A^{-1}$ (Matrix inverse)	$M=\text{inv}(A)$
<b>M=mtx_linsolve(A,B)</b>	Returns $M= A^{-1}B$	$M=\text{inv}(A)*B$ $M=A\backslash B$
<b>mtx_lu(L,U,A)</b>	Expresses the matrix $A$ , as the product of two essentially triangular matrices, one of them, a permutation of a lower triangular matrix and the other an upper triangular matrix. ( $L$ and $U$ , must be allocated previously and must have the same dimensions of $A$ )	$[L,U] = \text{lu}(A)$
<b>Q=mtx_qr(A,R)</b>	Orthogonal-triangular decomposition. ( $R$ is allocated previously and must have the same dimensions of $A$ )	$[Q,R]=\text{qr}(A)$
<b>X=mtx_sylvester(A,B,C)</b>	Solves the Sylvester equation $AX + XB = C$ , where $A$ is a m-by-m matrix, $B$ is a n-by-n matrix, and $X$ and $C$ are m-by-n matrices. The equation has a unique solution when the eigenvalues of $A$ and $-B$ are distinct. In terms of the Kronecker tensor product, , the equation is solved using: $[I\oplus A + B^T\oplus I] X(:) = C(:)$	$X=\text{sylvester}(A,B,C)$

	where $I$ is the identity matrix, and $X(:)$ and $C(:)$ denote the matrices $X$ and $C$ as single column vectors.	
<b>M=mtx_rpinv(A,tol)</b>	Returns the Moore-Penrose right pseudoinverse (tol = tolerance)	
<b>M=mtx_lpinv(A,tol)</b>	Returns the Moore-Penrose left pseudoinverse (tol = tolerance)	
<b>M=mtx_expm(A, alpha)</b>	Returns matrix exponential $e^{\alpha A}$ . <b>mtx_expm</b> uses the Padé approximation with scaling and squaring.	<b>M=expm(alpha*A)</b>
<b>M=mtx_fxptp(A, fx)</b>	Returns function $fx$ evaluated for each element of matrix $A$ Examples: <b>M=mtx_fxptp(A, sin);</b> → Element-wise sine <b>M=mtx_fxptp(A, fabs);</b> → Element-wise absolute value	<b>M=fx(A)</b>
<b>Manipulation</b>		
<b>M=mtx_vcat(A,B)</b>	Returns $[A ; B]$ (Vertical concatenation - append by columns)	<b>M=[A;B]</b>
<b>M=mtx_hcat(A,B)</b>	Returns $[A , B]$ (Horizontal concatenation - append by rows)	<b>M=[A,B]</b>
<b>M=mtx_vcatn(A,B,C,...,X,Y,Z)</b>	Return $[A ; B ; C ; \dots , X ; Y ; Z]$ (Vertical concatenation - append by columns)	<b>M=[A;B;C;...;X;Y;Z]</b>
<b>M=mtx_hcatn(A,B,C,...,X,Y,Z)</b>	Return $[A , B , C , \dots , X , Y , Z]$ (Horizontal concatenation - append by rows)	<b>M=[A,B,C,...,X,Y,Z]</b>
<b>Visualization</b>		
<b>mtx_disp(A)</b>	Display matrix $A$	<b>disp(A)</b>
<b>mtx_show(A)</b>	Display matrix with variable name	

## MTX Examples

### 1) Solving the Sylvester Equation $AX+XB=C$

```
/* Solve Sylvester Equation  $A^*X + X^*B = C$  with 4-by-2 Output */
#include <stdio.h>
#include <stdlib.h>
#include "mtx.h"

int main(void){
    /*Create a 4-by-4 coefficient matrix, A, and 2-by-2 coefficient matrix, B.*/
    double a[4][4]= {
        1,    0,    2,    3,
        4,    1,    0,    2,
        0,    5,    5,    6,
        1,    7,    9,    0,
    };
    double b[2][2]= {
        0,    -1,
        1,    0,
    };
    /*Define C as a 4-by-2 matrix to match the corresponding sizes of A and B*/
    double c[4][2]={
        1,    0,
        2,    0,
        0,    3,
        1,    1,
    };
    /*Create a mtx-type variables that points to the 2-dimensional arrays*/
    matrix A = mtx_2dtomtx(a);
    matrix B = mtx_2dtomtx(b);
    matrix C = mtx_2dtomtx(c);

    matrix X = NULL;
    /*Use the sylvester function to solve the Sylvester equation for these values of A, B, and C.*/
    X = mtx_sylvester(A,B,C);
    /*Display the matrices A,B,C and the solution X*/
    mtx_dispn(A,B,C,X);
    /*Release the heap*/
    mtx_del(A);
    mtx_del(B);
    mtx_del(C);
    mtx_del(X);
    return EXIT_SUCCESS;
}
```

## 2) Solving the linear system $AX=B$

```
/* Solving the linear equations A*x=b using LAPACK */
#include <stdio.h>
#include <stdlib.h>
#include "mtx.h"

int main(void){
    /*Define the equations using the matrix notation*/
    double a[3][3]= {
        3.1,    1.3,    -5.7,
        1.0,    -6.9,    5.8,
        3.4,    7.2,    -8.8,
    };
    double b[3]= {
        -1.3,
        -0.1,
        1.8,
    };
    /*Create a mtx-type variables that points to the 2-dimensional arrays*/
    matrix A = mtx_2dtomtx(a);
    matrix B = mtx_1dtomtx(b);

    matrix X = NULL;
    /*find solution using MTX routine mtx_linsolve*/
    X = mtx_linsolve(A,B);
    /*Display the matrices A,B and the solution X*/
    mtx_disp(A,B,X);
    /*Release the heap*/
    mtx_del(A);
    mtx_del(B);
    mtx_del(X);
    return EXIT_SUCCESS;
}
```

3) Evaluates a simple matrix expression

```
/*
 * Evaluate the expression  $X = (A*B)/(alpha + B'*A*B)$ 
 * A is a 4-square random matrix and B is a 4-column random matrix
 * alpha is scalar
 */
#include <stdio.h>
#include <stdlib.h>
#include "mtx.h"

int main(void){
    /*Define the using the matrix notation*/
    matrix A = mtx_rand(4,4);
    matrix B = mtx_rand(4,1);
    matrix X = NULL; //holds the expression
    matrix AB = NULL; //holds the auxiliar A*B result
    matrix BtAB = mtx_new(1,1); // holds the auxiliar B'*A*B result
    double alpha = 0.8;

    /*Compute the operations*/
    AB = mtx_prod(1.0, A,B); // AB = A*B
    mtx_dgemm(1.0, B, 't', AB, '.', 0, BtAB); // compute B'*A*B using the generalized matrix product routine mtx_dgemm
    X = mtx_koper(AB, '/', (alpha + BtAB->pos[0][0]) );
    /*Display the matrices*/
    mtx_disp(A,B,AB,BtAB,X);
    /*Release the heap*/
    mtx_del(A);
    mtx_del(B);
    mtx_del(AB);
    mtx_del(BtAB);
    mtx_del(X);
    return EXIT_SUCCESS;
}
```