MTX - 3.4

C ANSI/C++ BLAS-3 API

Developed by Eng. Juan Camilo Gómez Cadavid MSc.

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

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

M=mtx_prod(alpha,A,B)	Returns matrix Multiplication $M=\alpha AB$ α is scalar, A and B matrices.	M=alpha*A*B
ret=mtx_dgemm(alpha,A,TA,B,TB,beta,C)	Computes $C = \alpha AB + \beta C$ and related operations. α and β are scalars, and A, B and C are matrices. TA = 'T' or 't' → A is considered transposed (A^T). TB = 'T' or 't' → 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
M=mtx_kron(A,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 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=kron(A,B)
M=mtx_powui(A,n)	Returns A^n (Matrix power - n must be an unsigned int and A an square matrix)	M=A^n
M=mtx_ptpprod(A,B)	Returns $A.B$ (point by point product) (A and B must have the same dimensions)	M=A.*B
M=mtx_koper(A,Op,beta)	Returns standard scalar matrix operations. $C = A(Op)\beta$ Related operations (Op) = '+' , '-' , '*', '/' and '^' (power) Example: `M=mtx_koper(A,'+',beta);`	M=A+beta M=A-beta M=A*beta M=A/beta M=A^beta
M=mtx_inv(A)	Returns A^{-1} (Matrix inverse)	M=inv(A)
M=mtx_linsolve(A,B)	Returns $M= A^{-1}B$	M=inv(A)*B M=A\B
mtx_lu(L,U,A)	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] = lu(A)
mtx_svd(X,U,W,V)	Given a matrix $X_{(mxn)}$, this routine computes its singular value decomposition with the form $X = UWV^T$, where $U_{(mxn)}$, $W_{(nxn)}$ and $V_{(nxn)}$	[U,W,V]=svd(X)
Q=mtx_qr(A,R)	Orthogonal-triangular decomposition. (R is allocated previously and must have the same dimensions of A)	[Q,R]=qr(A)
X=mtx_sylvester(A,B,C)	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.	X=sylvester(A,B,C)

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

1) Solving the Sylvester Equation AX+XB=C

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

int main(void){
    mtx_assign_heap_wrappers(calloc, free); /*Assing the functions for heap memory allocation*/
    /*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

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

int main(void){
    mtx_assign_heap_wrappers(calloc, free); /*Assing the functions for heap memory allocation*/
    /*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_dispn(A,B,X);
    /*Release the heap*/
    mtx_del(A);
    mtx_del(B);
    mtx_del(X);
    return EXIT_SUCCESS;
}
```

3) Evaluate a simple matrix expression

```c
/*
 * 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>
#define MTX_PRINTOUT
#include "mtx.h"

int main(void){
    mtx_assign_heap_wrappers(calloc, free); /*Assing the functions for heap memory allocation*/
    /*Creating the matrices */
    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_dispn(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;
}
```