computeInverseMatrixProduct

Computes the product of an inverted matrix A represented by its Cholesky decomposed lower triangle matrix L and a vector b or even another matrix B. Solving runs column by column if a Matrix B is passed and computes the linear system to an intermediate result with the lower triangle matrix L (inner solving) and finaly in an outer solving with the transposed lower triangle matrix L and the intermediate result to the final product x of the inverted matrix and vector(s).

Syntax

```
x = computeInverseMatrixProduct(L, b)
```

Description

x = computeInverseMatrixProduct(L, b) performs the inverse matrix product of matrix A and a vector b or even another matrix B. Then product is formed column by column of B. The not inverted matrix A is represented by its lower triangle matrix L (Cholesky Decomposition).

Examples

```
A = [1.0, 0.9, 0.8;
     0.9, 1.0, 0.9;
     0.8, 0.9, 1.0];
L = decomposeChol(A);
b = = [5; 9; 0.5];
x = computeInverseMatrixProduct(L, b);
B = = [5   ,  9;
       0.5,  5;
       3   , -1];
X = computeInverseMatrixProduct(L, B);
```

Input Argurments

L is the lower triangle matrix of a matrix A.

b is a vector or matrix of real values.

Output Argurments

x is the product of the inverted matrix A and b.

Requirements

- Other m-files required: None
- Subfunctions: linsolve, mustBeLowerTriangle, mustBeFitSize
- MAT-files required: None

See Also

- decomposeChol
- linsolve


```
function x = computeInverseMatrixProduct(L, b)
    arguments
        % validate L as lower triangle matrix of size N x N
        L (:,:) double {mustBeReal, mustBeLowerTriangle(L)}
        % validate b as vecotr matrix with same row length as L
```

```
        b (:,:) double {mustBeReal, mustBeFitSize(L, b)}
    end

    % set linsolve option for inner (lower triangle) and outer (upper triangle)
    % solve, outer solve runs with intermediate result of inner solve
    opts1.LT = true;
    opts2.UT = true;

    % get size of b, if b is a matrix solve column by column
    [M, N] = size(b);

    % allocate memory for product result
    x = zeros(M, N);

    % solve column by column
    for n = 1:N
        % compute inner solve to intermediate result vecotor
        v = linsolve(L, b(:,n), opts1);

        % save final inverse product from outer solve
        x(:,n) = linsolve(L', v, opts2);
    end
end

% Custom validation functions
function mustBeLowerTriangle(L)
    % Test for lower triangle matrix
    if ~istril(L)
        eid = 'Matrix:notLowerTriangle';
        msg = 'Matrix is not lower triangle.';
        throwAsCaller(MException(eid,msg))
    end
    % Test for N x N
    if ~isequal(size(L,1), size(L, 2))
        eid = 'Size:notEqual';
        msg = 'L is not size of N x N.';
        throwAsCaller(MException(eid,msg))
    end
end

function mustBeFitSize(L, b)
    % Test for equal size
    if ~isequal(size(L,1), size(b, 1))
        eid = 'Size:notEqual';
        msg = 'Size of rows are not fitting.';
        throwAsCaller(MException(eid,msg))
    end
end
```

Published with MATLAB® R2020b