computeTransposeInverseProduct

Computes the both side product of an inverted Matrix A and a vector b (left product) and the transposed of b (right product). If matrix B is passed instead of a vector b the computation is done column by column of B. The matrix A is represented by its Cholesky decomposed lower triangle matrix L. The computation is optimized so it does a linear solve with lower triangle matrix to intermediate result vector. The final both side product is now the transpose of intermediate result multyplied with intermediate results itself. So a outer linear solve is not needed anymore.

Syntax

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

Description

x = computeTransposeInverseProduct(L, b) linear solve the equation system to a intermediate result and get the both side transpose inverse product of A and b by multiply the transpose intermediate result with intermediate resulst itself.

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 = computeTransposeInverseProduct(L, b);
B = = [5   ,  9;
       0.5,  5;
       3   , -1];
X = computeTransposeInverseProduct(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 both side product of the inverted matrix A and b and transposed b.

Requirements

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

See Also

- decomposeChol
- linsolve


```
function x = computeTransposeInverseProduct(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 to solve with lower triangle matrix
    opts.LT = true;

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

    % allocate memory for intermediate result
    v = zeros(M, N);

    % solve column by column
    for n=1:N
        % save to intermediate result columns
        v(:,n) = linsolve(L, b(:,n), opts);
    end

    % get final product by multiply transposed intermediate result with itself
    x = v' * v;
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