Implementation of LU Decomposition and Linear Solver using Matlab

Nasser M. Abbasi

sometime in 2009 Compiled on January 30, 2024 at 11:36pm

Contents

1 introduction

This is MATLAB implementation for LU decomposition, forward substitution, backward substitution, and linear system solver.

The functions written are:

1. `nma_LU.m.txt` LU decomposition with partial pivoting with threshold support.
2. `nma_ForwardSub.m.txt` solves $Ly = b$ for y
3. `nma_BackSub.m.txt` solves $Ux = y$ for x
4. `nma_LinearSolve.m.txt` driver to solve $Ax = b$ for x using calling sequence $1 \rightarrow 2 \rightarrow 3$

Partial pivoting (P matrix) was added to the LU decomposition function. In addition, the LU function accepts an additional argument which allows the user more control on row exchange.

Matlab lu() function does row exchange once it encounters a pivot larger than the current pivot. This is a good thing to always try to do. But sometimes if the difference between the pivots is small, a user might not want this feature. Hence I added a "threshold" second parameter to the nma_LU.m function to indicate how large a difference should exist for a row exchange to occur.

A row exchange will always occur if the current pivot is zero and a non-zero pivot exist to do the exchange.

To get the same exact behavior as Matlab lu() simply make this parameter zero.

Below are examples calling the nma_LU, nma_ForwardSub.m, nma_BackSub.m and nma_LinearSolve.m.

In each example below, the output is verified against Matlab own functions

2 examples

2.1 using nma_LU()

2.1.1 example 1

```
>> A=[1 1 2;2 -1 1;1 2 0]
A =
     1     1     2
     2    -1     1
```

```
     1     2     0

>> [L,U,P]=nma_LU(A,0)
L =
   1.00000000000000                  0                  0
   0.50000000000000   1.00000000000000                  0
   0.50000000000000   0.60000000000000   1.00000000000000

U =
   2.00000000000000  -1.00000000000000   1.00000000000000
                  0   2.50000000000000  -0.50000000000000
                  0                  0   1.80000000000000
P =
     0     1     0
     0     0     1
     1     0     0

>> [L,U,P]=lu(A)

L =


   1.00000000000000                  0                  0
   0.50000000000000   1.00000000000000                  0
   0.50000000000000   0.60000000000000   1.00000000000000

U =
   2.00000000000000  -1.00000000000000   1.00000000000000
                  0   2.50000000000000  -0.50000000000000
                  0                  0   1.80000000000000

P =
     0     1     0
     0     0     1
     1     0     0
```

2.1.2 example 2

```
>> A=rand(4);
>> [L,U,P]=nma_LU(A,0)

L =
   1.00000000000000                  0                  0                  0
   0.01212703756687   1.00000000000000                  0                  0
   0.07119243718995   0.20742768803520   1.00000000000000                  0
   0.43394327408595   0.19377225100868   0.40879105345917   1.00000000000000

U =
   0.81316649730376   0.19872174266149   0.01527392702904   0.46599434167542
                  0   0.60138257315521   0.74660044907755   0.41299833684006
                  0                  0   0.11623493184110   0.32625386921423
                  0                  0                  0   0.51620218784594

P =
     0     0     1     0
     0     0     0     1
     1     0     0     0
     0     1     0     0

>> [L,U,P]=lu(A)
L =
   1.00000000000000                  0                  0                  0
   0.01212703756687   1.00000000000000                  0                  0
   0.07119243718995   0.20742768803520   1.00000000000000                  0
   0.43394327408595   0.19377225100868   0.40879105345917   1.00000000000000

U =
   0.81316649730376   0.19872174266149   0.01527392702904   0.46599434167542
                  0   0.60138257315521   0.74660044907755   0.41299833684006
                  0                  0   0.11623493184110   0.32625386921423
                  0                  0                  0   0.51620218784594

P =
     0     0     1     0
     0     0     0     1
     1     0     0     0
     0     1     0     0
```

```
>>
```

2.2 using nma_LinearSolve()

2.2.1 example 1

```
>> A=[1 1 2;2 -1 1;1 2 0]
A =
     1     1     2
     2    -1     1
     1     2     0

>> b=[1 2 1];

>> nma_LinearSolve(A,b)

ans =

     1
     0
     0

>> A\b(:)

ans =

     1
     0
     0
>>
```

2.2.2 example 2

```
>> A=rand(6);
>> b=rand(6,1);
>> nma_LinearSolve(A,b)

ans =

   0.59090034220622
  -0.56523444269280
   0.95687095978224
  -0.97248777153372
   1.00007995741472
   0.24035777097022

>> A\b(:)

ans =

   0.59090034220622
  -0.56523444269280
   0.95687095978223
  -0.97248777153372
   1.00007995741472
   0.24035777097022

>>
```

2.3 using nma_ForwardSub()

2.3.1 example 1

```
>> [L,U,P]=nma_LU(A,0);
>> nma_ForwardSub(L,b)

ans =

   0.83849604493808
   0.36727512318587
   0.12405626870025
```

```
  -0.14539724685973
   0.17813906538571
  -0.19809655526705
```

2.4 using nma_BackSub()

2.4.1 example 1

```
>> nma_BackSub(U,ans)

ans =

   0.29867870305809
  -0.84855613142087
   0.48347828223154
  -1.68311779577975
   1.49928530116874
   1.53825192677360
```

3 code listing

3.1 test script

```
function result = test(iterations, matrixsize)
tic;
x = rand(matrixsize);
for i = 1:iterations
    for j = 1:size(x(:,1))
        y = abs(fft(x(j,:)));
    end
end

toc
```

3.2 nma_LU.m

```
function [L,U,P]=nma_LU(A,threshold)
%function [L,U,P]=nma_LU(A,threshold)
%
%does LU decomposition with permutation matrix for
%pivoting reorder. Supports threasold parameter.
%Similar to matlab lu function, but with little more control to the
%user as to when row exhanges should be made.
%
%INPUT:
%  A:  an nxn square matrix
%  threshold:
%             numerical positive value. row exchanges will be made
%             only if the abs difference between the largest pivot and
%             the current pivot is larger than this threshold.
%             Hence setting threashold to be 0 will cause a row exchange
%             anytime when there is a larger pivot. This is the DEFAULT
%             behaviour similar to Matlab lu().
%OUTPUT:
%  P: nxn permutation matrix such that PA=LU
%  L: unity lower triangular matrix
%  U: unity upper triangular

% By Nasser M. Abbasi
% HW 4, MATH 501, CSUF
%
% LU decomposition with threshold support.
% Copyright (C) 2007  Nasser Abbasi
%
% This program is free software; you can redistribute it and/or
% modify it under the terms of the GNU General Public License
% as published by the Free Software Foundation; either version 2
% of the License, or (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
```

```
% along with this program; if not, write to the Free Software
% Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.

%EXAMPLE RUN
%
% >> A=rand(4);
% >> [L,U,P]=nma_LU(A,0)
%
    if nargin ~=2
       error '2 parameters are expected'
    end

    if ~isnumeric(A)
        error 'input matrix A must be numeric'
    end

    if ~isnumeric(threshold)
        error 'threshold must be numeric'
    end

    if  threshold<0
        error 'threshold must be positive'
    end

    [nRow,nCol]=size(A);
    if nRow ~= nCol
        error 'Matrix must be square'
    end

    %*****************************
    %* Internal functions to flip
    %* rows for row pivoting
    %*****************************
    function flipRows()
        [c,I]=max(abs(A(n:end,n)));
        I=I+(n-1);
        tmp=A(n,:);
        A(n,:)=A(I,:);
        A(I,:)=tmp;

        %make sure we also flip the L matrix rows to keep in sync
```

```
    tmp=L(n,:);
    L(n,:)=L(I,:);
    L(I,:)=tmp;

    %now make the elementary matrix for this move
    E(n,:)=0;
    E(n,I)=1;
    E(I,:)=0;
    E(I,n)=1;
end

P=diag(ones(nRow,1));
U=zeros(nRow);
L=zeros(nRow);

for n=1:nRow-1
    currentPivot=A(n,n);

    E=diag(ones(nRow,1));

    maxPivot=max(A(n+1:end,n));
    if abs(currentPivot)<eps  %zero, do row exchage always
       if abs(maxPivot)<eps  % not possible to exchange
          error 'unable to complete LU decomposition, bad A'
       else
          flipRows();
       end
    else %not a zero pivot, but still can exchange, check threshold
      if abs(currentPivot)<abs(maxPivot)
         if abs(currentPivot-maxPivot)>=threshold
            flipRows();
         end
      end
    end

    P=P*E;  %update the perumtation matrix

    for i=n+1:nRow
        L(i,n)=A(i,n)/A(n,n);
        A(i,n)=0;
        for j=n+1:nRow
```

```
                A(i,j)=A(i,j)-L(i,n)*A(n,j);
            end
        end
    end

    L=L+diag(ones(nRow,1));
    P=P';
    U=A;  % that is all
end
```

3.3 nma_LinearSolve.m

```
function x=nma_LinearSolve(A,b)
%function x=nma_LinearSolve(A,b)
%
% Solves Ax=b
%
%
%INPUT:
%  A:  an nxn square matrix
%  b:  vector of size n
%
%OUTPUT:
%  x: the solution to A*x=b

% By Nasser M. Abbasi
% HW 4, MATH 501, CSUF
%
% Linear Solver. Solves Ax=b
% Copyright (C) 2007  Nasser Abbasi
%
% This program is free software; you can redistribute it and/or
% modify it under the terms of the GNU General Public License
% as published by the Free Software Foundation; either version 2
% of the License, or (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
```

```
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, write to the Free Software
% Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.

    if nargin ~= 2
       error 'Only two inputs are required'
    end

    if ~(isnumeric(A)&isnumeric(b))
        error 'input must be numeric'
    end

    [nRow,nCol]=size(b);
    if nRow>1 & nCol>1
        error 'b must be a vector'
    end

    [nRow,nCol]=size(A);
    if nRow ~= nCol
        error 'Matrix A must be square'
    end

    if length(b) ~= nRow
       error 'b length does not match A matrix dimension'
    end

    b=b(:);

    [L,U,P] = nma_LU(A,0);
    y       = nma_ForwardSub(L,P*b);
    x       = nma_BackSub(U,y);

end
```

3.4 nma_ForwardSub.m

```
function y=nma_ForwardSub(L,w)
%function y=nma_ForwardSub(L,w)
%
%Forward substitution that solves the lower triangular system
%Ly=w for y
%
%
%INPUT:
%  L:  an nxn L, lower tirangular square matrix
%  w:  vector of size n
%
%OUTPUT:
%  y: the solution to L*y=w

% By Nasser M. Abbasi
% HW 4, MATH 501, CSUF
%
% LU decomposition with threshold support.
% Copyright (C) 2007  Nasser Abbasi
%
% This program is free software; you can redistribute it and/or
% modify it under the terms of the GNU General Public License
% as published by the Free Software Foundation; either version 2
% of the License, or (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, write to the Free Software
% Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
% 02110-1301, USA.


%EXAMPLE RUN
% >> [L,U,P]=nma_LU(A,0);
% >> nma_ForwardSub(L,b)
```

```
%
% ans =
%
%    0.83849604493808
%    0.36727512318587
%    0.12405626870025
%   -0.14539724685973
%    0.17813906538571
%   -0.19809655526705
% % >>

    if nargin ~= 2
       error 'Only two inputs are required'
    end

    if ~(isnumeric(L)&isnumeric(w))
        error 'input must be numeric'
    end

    [nRow,nCol]=size(w);
    if nRow>1 & nCol>1
        error 'w must be a vector not a matrix'
    end

    [nRow,nCol]=size(L);
    if nRow ~= nCol
        error 'Matrix L must be square'
    end

    if length(w) ~= nRow
        error 'w length does not match L matrix dimension'
    end

    y=zeros(nRow,1);
    y(1)=w(1)/L(1,1);

    w=w(:);

    for n=2:nRow
        y(n)=( w(n) - L(n,1:n-1)*y(1:n-1) ) / L(n,n);
    end
```

```
end
```

3.5 nma_BackSub.m

```
function x=nma_BackSub(U,v)
%function x=nma_BackSub(U,v)
%
%Backward substitution that solves the lower triangular system
%Ux=v for x
%
%
%INPUT:
%  U:  an nxn U, upper tirangular square matrix
%  v:  vector of size n
%
%OUTPUT:
%  x: the solution to U*x=v

% By Nasser M. Abbasi
% HW 4, MATH 501, CSUF
%
% Backsubstition matlab function.
% Copyright (C) 2007  Nasser Abbasi
%
% This program is free software; you can redistribute it and/or
% modify it under the terms of the GNU General Public License
% as published by the Free Software Foundation; either version 2
% of the License, or (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, write to the Free Software
% Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA.

    if nargin ~= 2
```

```
      error 'Only two inputs are required'
    end

    if ~(isnumeric(U)&isnumeric(v))
        error 'input must be numeric'
    end

    [nRow,nCol]=size(v);
    if nRow>1 & nCol>1
        error 'v must be a vector not a matrix'
    end

    [nRow,nCol]=size(U);
    if nRow ~= nCol
        error 'Matrix U must be square'
    end

    if length(v) ~= nRow
      error 'v length does not match U matrix dimension'
    end

    x=zeros(nRow,1);
    x(nRow)=v(nRow)/U(nRow,nRow);

    v=v(:);

    for n=(nRow-1):-1:1
        x(n)=(v(n)-(U(n,n+1:end)*x(n+1:end))) / U(n,n);
    end

end
```