

Implementation of LU Decomposition and Linear Solver using Matlab

Nasser M. Abbasi

sometime in 2009

Compiled on May 24, 2020 at 5:01am

Contents

1	introduction	1
2	examples	3
2.1	using nma_LU()	3
2.1.1	example 1	3
2.1.2	example 2	4
2.2	using nma_LinearSolve()	5
2.2.1	example 1	5
2.2.2	example 2	6
2.3	using nma_ForwardSub()	6
2.3.1	example 1	6
2.4	using nma_BackSub()	7
2.4.1	example 1	7
3	code listing	7
3.1	test script	7
3.2	nma_LU.m	7
3.3	nma_LinearSolve.m	10
3.4	nma_ForwardSub.m	11
3.5	nma_BackSub.m	13

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

2.1.2 example 2

```

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

L =
 1.000000000000000    0    0    0
 0.01212703756687    1.000000000000000    0    0
 0.07119243718995    0.20742768803520    1.000000000000000    0
 0.43394327408595    0.19377225100868    0.40879105345917    1.000000000000000

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 =
 1.000000000000000    0    0    0
 0.01212703756687    1.000000000000000    0    0
 0.07119243718995    0.20742768803520    1.000000000000000    0
 0.43394327408595    0.19377225100868    0.40879105345917    1.000000000000000

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

```
1 function result = test(iterations, matrixsize)
2 tic;
3 x = rand(matrixsize);
4 for i = 1:iterations
5     for j = 1:size(x(:,1))
6         y = abs(fft(x(j,:)));
7     end
8 end
9
10 toc
```

3.2 nma LU.m

```
1 function [L,U,P]=nma_LU(A,threshold)
2 %function [L,U,P]=nma_LU(A,threshold)
3 %
4 %does LU decomposition with permutation matrix for
5 %pivoting reorder. Supports threshold parameter.
6 %Similar to matlab lu function, but with little more control to the
7 %user as to when row exchanges should be made.
8 %
9 %INPUT:
10 % A: an nxn square matrix
11 % threshold:
12 % numerical positive value. row exchanges will be made
13 % only if the abs difference between the largest pivot and
14 % the current pivot is larger than this threshold.
```

```
15 %           Hence setting threshold to be 0 will cause a row exchange
16 %           anytime when there is a larger pivot. This is the DEFAULT
17 %           behaviour similar to Matlab lu().
18 %OUTPUT:
19 % P: nxn permutation matrix such that PA=LU
20 % L: unity lower triangular matrix
21 % U: unity upper triangular
22
23 % By Nasser M. Abbasi
24 % HW 4, MATH 501, CSUF
25 %
26 % LU decomposition with threshold support.
27 % Copyright (C) 2007 Nasser Abbasi
28 %
29 % This program is free software; you can redistribute it and/or
30 % modify it under the terms of the GNU General Public License
31 % as published by the Free Software Foundation; either version 2
32 % of the License, or (at your option) any later version.
33 %
34 % This program is distributed in the hope that it will be useful,
35 % but WITHOUT ANY WARRANTY; without even the implied warranty of
36 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
37 % GNU General Public License for more details.
38 %
39 % You should have received a copy of the GNU General Public License
40 % along with this program; if not, write to the Free Software
41 % Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
42
43 %EXAMPLE RUN
44 %
45 % >> A=rand(4);
46 % >> [L,U,P]=nma_LU(A,0)
47 %
48     if nargin ~=2
49         error '2 parameters are expected'
50     end
51
52     if ~isnumeric(A)
53         error 'input matrix A must be numeric'
54     end
55
56     if ~isnumeric(threshold)
57         error 'threshold must be numeric'
58     end
59
60     if threshold<0
61         error 'threshold must be positive'
```



```

62     end
63
64     [nRow,nCol]=size(A);
65     if nRow ~= nCol
66         error 'Matrix must be square'
67     end
68
69     %*****
70     %* Internal functions to flip
71     %* rows for row pivoting
72     %*****
73     function flipRows()
74         [c,I]=max(abs(A(n:end,n)));
75         I=I+(n-1);
76         tmp=A(n,:);
77         A(n,:)=A(I,:);
78         A(I,:)=tmp;
79
80         %make sure we also flip the L matrix rows to keep in sync
81         tmp=L(n,:);
82         L(n,:)=L(I,:);
83         L(I,:)=tmp;
84
85         %now make the elementary matrix for this move
86         E(n,:)=0;
87         E(n,I)=1;
88         E(I,:)=0;
89         E(I,n)=1;
90     end
91
92     P=diag(ones(nRow,1));
93     U=zeros(nRow);
94     L=zeros(nRow);
95
96     for n=1:nRow-1
97         currentPivot=A(n,n);
98
99         E=diag(ones(nRow,1));
100
101         maxPivot=max(A(n+1:end,n));
102         if abs(currentPivot)<eps %zero, do row exchange always
103             if abs(maxPivot)<eps % not possible to exchange
104                 error 'unable to complete LU decomposition, bad A'
105             else
106                 flipRows();
107             end
108         else %not a zero pivot, but still can exchange, check threshold

```

```

109     if abs(currentPivot)<abs(maxPivot)
110         if abs(currentPivot-maxPivot)>=threshold
111             flipRows();
112         end
113     end
114 end
115
116 P=P*E; %update the perumtation matrix
117
118 for i=n+1:nRow
119     L(i,n)=A(i,n)/A(n,n);
120     A(i,n)=0;
121     for j=n+1:nRow
122         A(i,j)=A(i,j)-L(i,n)*A(n,j);
123     end
124 end
125 end
126
127 L=L+diag(ones(nRow,1));
128 P=P';
129 U=A; % that is all
130 end

```

3.3 nma LinearSolve.m

```

1 function x=nma_LinearSolve(A,b)
2 %function x=nma_LinearSolve(A,b)
3 %
4 % Solves Ax=b
5 %
6 %
7 %INPUT:
8 % A: an nxn square matrix
9 % b: vector of size n
10 %
11 %OUTPUT:
12 % x: the solution to A*x=b
13
14 % By Nasser M. Abbasi
15 % HW 4, MATH 501, CSUF
16 %
17 % Linear Solver. Solves Ax=b
18 % Copyright (C) 2007 Nasser Abbasi
19 %
20 % This program is free software; you can redistribute it and/or
21 % modify it under the terms of the GNU General Public License
22 % as published by the Free Software Foundation; either version 2

```

```

23 % of the License, or (at your option) any later version.
24 %
25 % This program is distributed in the hope that it will be useful,
26 % but WITHOUT ANY WARRANTY; without even the implied warranty of
27 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
28 % GNU General Public License for more details.
29 %
30 % You should have received a copy of the GNU General Public License
31 % along with this program; if not, write to the Free Software
32 % Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
33
34     if nargin ~= 2
35         error 'Only two inputs are required'
36     end
37
38     if ~(isnumeric(A)&isnumeric(b))
39         error 'input must be numeric'
40     end
41
42     [nRow,nCol]=size(b);
43     if nRow>1 & nCol>1
44         error 'b must be a vector'
45     end
46
47     [nRow,nCol]=size(A);
48     if nRow ~= nCol
49         error 'Matrix A must be square'
50     end
51
52     if length(b) ~= nRow
53         error 'b length does not match A matrix dimension'
54     end
55
56     b=b(:);
57
58     [L,U,P] = nma_LU(A,0);
59     y      = nma_ForwardSub(L,P*b);
60     x      = nma_BackSub(U,y);
61
62 end

```

3.4 nma_ForwardSub.m

```

1 function y=nma_ForwardSub(L,w)
2 %function y=nma_ForwardSub(L,w)
3 %
4 %Forward substitution that solves the lower triangular system

```

```
5 %Ly=w for y
6 %
7 %
8 %INPUT:
9 % L:  an nxn L, lower tirangular square matrix
10 % w:  vector of size n
11 %
12 %OUTPUT:
13 % y:  the solution to L*y=w
14
15 % By Nasser M. Abbasi
16 % HW 4, MATH 501, CSUF
17 %
18 % LU decomposition with threshold support.
19 % Copyright (C) 2007  Nasser Abbasi
20 %
21 % This program is free software; you can redistribute it and/or
22 % modify it under the terms of the GNU General Public License
23 % as published by the Free Software Foundation; either version 2
24 % of the License, or (at your option) any later version.
25 %
26 % This program is distributed in the hope that it will be useful,
27 % but WITHOUT ANY WARRANTY; without even the implied warranty of
28 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
29 % GNU General Public License for more details.
30 %
31 % You should have received a copy of the GNU General Public License
32 % along with this program; if not, write to the Free Software
33 % Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
34 % 02110-1301, USA.
35
36
37 %EXAMPLE RUN
38 % >> [L,U,P]=nma_LU(A,0);
39 % >> nma_ForwardSub(L,b)
40 %
41 % ans =
42 %
43 %    0.83849604493808
44 %    0.36727512318587
45 %    0.12405626870025
46 %   -0.14539724685973
47 %    0.17813906538571
48 %   -0.19809655526705
49 % % >>
50
51     if nargin ~= 2
```

```

52     error 'Only two inputs are required'
53 end
54
55 if ~(isnumeric(L)&isnumeric(w))
56     error 'input must be numeric'
57 end
58
59 [nRow,nCol]=size(w);
60 if nRow>1 & nCol>1
61     error 'w must be a vector not a matrix'
62 end
63
64 [nRow,nCol]=size(L);
65 if nRow ~= nCol
66     error 'Matrix L must be square'
67 end
68
69 if length(w) ~= nRow
70     error 'w length does not match L matrix dimension'
71 end
72
73 y=zeros(nRow,1);
74 y(1)=w(1)/L(1,1);
75
76 w=w(:);
77
78 for n=2:nRow
79     y(n)=( w(n) - L(n,1:n-1)*y(1:n-1) ) / L(n,n);
80 end
81
82 end

```

3.5 nma BackSub.m

```

1 function x=nma_BackSub(U,v)
2 %function x=nma_BackSub(U,v)
3 %
4 %Backward substitution that solves the lower triangular system
5 %Ux=v for x
6 %
7 %
8 %INPUT:
9 % U:  an nxn U, upper tirangular square matrix
10 % v:  vector of size n
11 %
12 %OUTPUT:
13 % x:  the solution to U*x=v

```

```
14
15 % By Nasser M. Abbasi
16 % HW 4, MATH 501, CSUF
17 %
18 % Backsubstitution matlab function.
19 % Copyright (C) 2007 Nasser Abbasi
20 %
21 % This program is free software; you can redistribute it and/or
22 % modify it under the terms of the GNU General Public License
23 % as published by the Free Software Foundation; either version 2
24 % of the License, or (at your option) any later version.
25 %
26 % This program is distributed in the hope that it will be useful,
27 % but WITHOUT ANY WARRANTY; without even the implied warranty of
28 % MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
29 % GNU General Public License for more details.
30 %
31 % You should have received a copy of the GNU General Public License
32 % along with this program; if not, write to the Free Software
33 % Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA.
34
35     if nargin ~= 2
36         error 'Only two inputs are required'
37     end
38
39     if ~(isnumeric(U)&isnumeric(v))
40         error 'input must be numeric'
41     end
42
43     [nRow,nCol]=size(v);
44     if nRow>1 & nCol>1
45         error 'v must be a vector not a matrix'
46     end
47
48     [nRow,nCol]=size(U);
49     if nRow ~= nCol
50         error 'Matrix U must be square'
51     end
52
53     if length(v) ~= nRow
54         error 'v length does not match U matrix dimension'
55     end
56
57     x=zeros(nRow,1);
58     x(nRow)=v(nRow)/U(nRow,nRow);
59
60     v=v(:);
```

```
61  
62     for n=(nRow-1):-1:1  
63         x(n)=(v(n)-(U(n,n+1:end)*x(n+1:end))) / U(n,n);  
64     end  
65  
66 end
```