Penta diagonal system solver in Matlab. Report for Math 501, CSUF

Nasser M. Abbasi

March 2007 Compiled on January 31, 2024 at 2:26am

Contents

1 Derive special Gauss-elimination

Derive special Gauss-elimination strategy to transfer the resulting penta-diagonal system into an upper semi-penta system

The following algorithm was designed and developed to handle a general banded A matrix to solve the problem of solving $Ax = b$.

This algorithm works on Matrices which contain only one band of specific width such as those found in tri-diagonal and penta-diagonal matrices.

The main idea of the algorithm is to locate submatrices within the main matrix A, so as to process those by applying the standard Gaussian elimination algorithm on them.

The algorithm locates these submatrices which are bounded below and to the right by the first zero entry. Starting at first pivot in $A(1, 1)$, looking down and locating the first zero entry to determine the lower bound, and then looking right from that location to locate the first zero entry. This determines the boundaries of the submatrix.

This process is repeated by shifting one row down and one column to the right, and each time a new submatrix boundaries are located as described above, and Gaussian elimination is called to process this new lower submatrix.

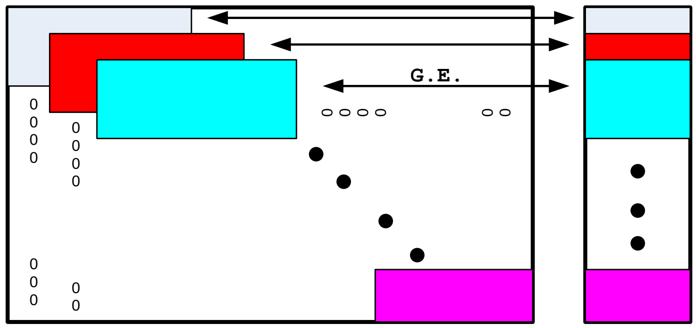
Hence we travel down the main matrix from the top left corner to the bottom right corner, processing small submatrices along the way. The b vector is updated all the time. Hence in each step, we create a new separate $Ax = b$ with its own A and b variables extracted from the original A and original b variables.

Notice that no data copying is involved, and the data is processed in-place.

The advantage of this algorithm is that it will work on any central banded Matrix A, tri-diagonal, penta-diagonal and larger bands.

The algorithm is illustrated in the following diagram

```
Gaussian Elimination is called repeatedly on the non-zero submatrices of the
original matrix
```




```
Overview of algorithm

  [A_part, b_part] = Get next non-empty submatrix from A

   WHILE  more LOOP
                CALL Gaussian Elimination on   A_part and b_part
                Updated A and b as a result of above call.
                [A_part,b_Part]=Get next non-empty submatrix from A
   END LOOP

   solution = CALL back substitution(A,b)
```

2 Derive special backward-substitution algorithm

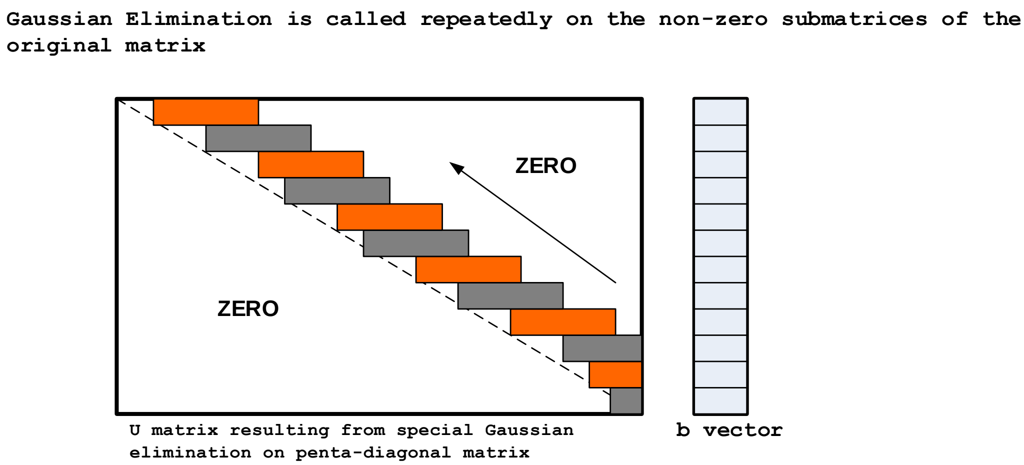
Derive special backward-substitution algorithm to solve the resulting upper semi-penta diagonal matrix.

The resulting U matrix from part(1) above is a banded matrix. Hence a special backward substitution algorithm was devised to take advantage of the sparseness of this U matrix.

Only the non-zero entries in each row are used to solve for x during the process of back substitution.

This is in comparison with the standard back substitution routine written for solving general $Ax = b$, which processed all entries in the upper triangular matrix regardless if the entries contain zero or not.

The following diagram illustrates the algorithm



Overview of special backsub algorithm

```
X(n)=b(n)/U(n,n)

For i=N-1:-1:1
  locate the subrow within row I with non-zero entries
  Use this sub-row to solve for x(i) by multiplying only non-zero entries
end
```

Figure 1: diagram illustrating the algorithm

3 Long operation counts

Elimination process: each submatrix is of size 3×5. There are $n-1$ such submatrices.

Each submatrix requires 2 divisions (for the multipliers), and 6 multiplications. (3 per row, we have 2 rows). Hence each submatrix requires 8 ops. Hence the total is $10(n-1)$

For the special backsub: each x requires 2 multiplications and one division, and there are $(n-1)$ rows to process. Also there is the first division for the $x(n)$.

Hence the total is $2(n-1)+1$.

Adding the elimination process with the backsub, we obtain

$$8(n-1)+2(n-1)+1 = 10(n-1)+1$$

4 Implementation in Matlab

The file `nma_pentaSolve.m` is Main driver. Called to solve $Ax = b$ when A is penta-banded. Locates matrices within the main matrix A and calls `nma_gaussian_elimination` on each problem.

The file `nma_backSub.m` is special back substitution for special banded U matrices

The file `nma_gaussian_elemination.m` is Gaussian elimination routine

The file `nma_penta_test.m` is script that calls `nma_pentaSolve` repeatedly with different A, b and compares the results with Matlab's "\" solver to check for correctness.

Appendix has source code listing.

5 Part 5 Test code

A Matlab script was written which tested the above implementation using different A, b input. Each test was verified against Matlab "\" solver. The following is the output of the test.

```
==========>test 1
A =
    15    -2    -6     0
    -2    12    -4    -4
    -6    -4    19    -9
     0    -1    -9    21
b =
   300     0     0     0
=======>our result
x =
  27.16548702392990
  11.42568250758342
  14.10515672396360
   6.58914728682170
=======>Matlab result
ans =

  27.16548702392989
  11.42568250758342
  14.10515672396360
   6.58914728682170
```

```
==========>test 2
A =
    15     8    -6     0     0     0
    -2    12    -4    -4     0     0
    -6    -4    19    -9     4     0
     0    -1    -9    21     6     7
     0     0     9    10    11     8
     0     0     0    10    -2     2

b =
   300     0     0     0     1     2

=======>our result
x =
  1.0e+002 *

   0.14783336170627
   0.22913057224154
   0.17509083392106
   0.43838420195044
   0.60556360806440
  -1.57635740168778
=======>Matlab result
ans =
  1.0e+002 *

   0.14783336170627
   0.22913057224154
   0.17509083392106
   0.43838420195044
   0.60556360806440
  -1.57635740168779

==========>test 3
A =
    15     8    -6     0     0     0     0
    -2    12    -4    -4     0     0     0
    -6    -4    19    -9     4     0     0
     0    -1    -9    21     6     7     0
     0     0     9    10    11     8     3
     0     0     0    10    -2     2     4
```

```
     0     0     0     0    -2     2     4

b =
   300     0     0     0     1     2     6
=======>our result
x =
  22.60711151041847
  10.45280071875325
  20.45484640105050
  -0.40000000000000
 -53.69705242060892
  75.01839040740875
 -62.85772141400877
=======>Matlab result
ans =
  22.60711151041846
  10.45280071875324
  20.45484640105048
  -0.40000000000000
 -53.69705242060884
  75.01839040740865
 -62.85772141400874

==========>test 4
A =
    15     8
     0    12
b =
   300     0
=======>our result
x =
    20
     0
=======>Matlab result
ans =
    20
     0

==========>test 5
A =
    15     8    -6     0     0     0     0     0
```

```
   -2    12    -4    -4     0     0     0     0
   -6    -4    19    -9     4     0     0     0
    0    -1    -9    21     6     7     0     0
    0     0     9    10    11     8     3     0
    0     0     0    10    -2     2     4     3
    0     0     0     0    -2     2     4     7
    0     0     0     0     0     4     8     9

b =
   300     0     0     0     1     2     6    10
=======>our result
x =
  1.0e+002 *
   0.13293913687916
   0.13247435914403
   0.00898032105660
   0.32197318793590
   1.01366621229967
  -1.80430528168548
  -0.00214695022696
   0.81493296983974

=======>Matlab result

ans =

  1.0e+002 *

   0.13293913687916
   0.13247435914403
   0.00898032105660
   0.32197318793589
   1.01366621229967
  -1.80430528168548
  -0.00214695022696
   0.81493296983974
```

6 Appendix

6.1 Code listing

6.1.1 nma_backsolve.m

```
%by Nasser Abbasi. Feb 27,2007
function x=nma_backsolve(A,b)

[n,nCols]=size(A);

x=zeros(n,1);

x(n)=b(n)/A(n,n);
x(n-1)=(b(n-1)-A(n-1,n)*x(n))/A(n-1,n-1);

for k=n-2:-1:1
   x(n-2)=(   b(n-2)-  A(n-2,n-1)*x(n-1) - A(n-2,n)*x(n)  )/A(n-2,n-2);
end


end
```

6.1.2 nma_penta.m

```
%by Nasser M. Abbasi. Feb 27,2007
function [A,b]=nma_penta(A,b)
[nRows,nCols]=size(A);

d=diag(A);

s=zeros(nRows-1,1);
for i=2:nRows
    s(i-1)=A(i,i-1);
end

u=zeros(nRows-2,1);
for i=3:nRows
    u(i-2)=A(i,i-2);
end
```

```
t=zeros(nRows-1,1);
for i=1:nRows-1
    t(i)=A(i,i+1);
end

v=zeros(nRows-2,1);
for i=1:nRows-2
    v(i)=A(i,i+2);
end


j=0;
for k=2:nRows
   j=j+1;
   m=s(j)/d(j);
   b(k)=b(k)-m*b(k-1);

   A(k,k-1)=A(k,k-1)-m*A(k-1,k-1);

   s(j)=A(k,k-1);

   A(k,k)=d(j+1)-m*t(j);
   d(j+1)=A(k,k);

   if k+1<=nRows
     A(k,k+1)=t(j+1)-m*v(j);
     t(j+1)=A(k,k+1);
   end

   if k+1<=nRows
     m=u(j)/d(j);
     A(k+1,k-1)=A(k+1,k-1)-m*A(k,k-1);
     u(j)=A(k+1,k-1);

     A(k+1,k)=s(j+1)-m*d(j+1);
     s(j+1)=A(k+1,k);

     A(k+1,k+1)=d(j+2)-m*t(j+1);
     d(j+2)=A(k+1,k+1);
```

```
      if k+2<nRows
        A(k+1,k+2)=t(j+2)-m*v(j+1);
        t(j+2)=A(k+1,k+2);
      end

    end

end


end
```

6.1.3 nma_pentaSolve.m

```
function x=nma_pentaSolve(A,b)
%function x=nma_pentaSolve(A,b)
%Solves a penta-diagonal Ax=b system
%
%INPUT:
%  A  an nxn Matrix
%  b  an vector of length n
%
%OUTPUT
%  x  vector of length n, the solution for Ax=b
%
%Algorithm Overview:
% The matrix is banded matrix. Due to this, saving in processing
% is achived by only processing the non-zero elements along the band.
% see full report for more details
%by Nasser M. Abbasi. Feb 27,2007

%March 7, 2007

    if nargin ~=2
        error('2 arguments required');
    end

    if ~isnumeric(A)|~isnumeric(b)
        error('A and b must be numeric');
    end
```

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

    b=b(:);

    [b_nRow,b_nCol]=size(b);
    if b_nCol>1
        error('b must be a vector');
    end

    if b_nRow~=nRow
        error('b must be same size as A');
    end

    [U,b_new]=penta_elimination(A,b);
    x=nma_pentaBackSub(U,b_new);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% The algorithm checks for the band width, and process elements
% within the band as using standard G.E. This saves processing
% time compared with the G.E. which process the whole matrix.
%
% Since the matrix is sparse, we save time by processing only the
% non-zero band.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [A,b]=penta_elimination(A,b)

    TRUE=1;
    FALSE=0;

    [nRow,nCol]=size(A);
    pivot=1;
    more_bands=TRUE;

    while more_bands
```

```
        r=find(abs(A(pivot:end,pivot))>4*eps);
        r=r+(pivot-1);  %find is relative to current band, so adjust to A
        end_row=r(end);
        r=find(abs(A(end,:))>4*eps);
        end_column=r(end);

        [U,b_new]=nma_gaussian_elimination...
            (A(pivot:end_row,pivot:end_column),b(pivot:end_row));
        A(pivot:end_row,pivot:end_column)=U;
        b(pivot:end_row)=b_new;

        if pivot==nRow
            more_bands=FALSE;
        else
            pivot=pivot+1;
        end
    end

end
```

6.1.4 nma_gaussian_elimination.m

```
function [A,b]=nma_gaussian_elimination(A,b)
%function [U,b_new]=nma_gaussian_elimination(A,b)
%
%Perform Gaussian Elimination on a matrix and the
%updated the corresponding b matrix as well.
%
%This function is used as part of the routines I written
%during work on Math 501, at CSFU, winter 2007
%
%INPUT
%  A: matrix. does not have to be square
%  b: vector. from Ax=b
%
%OUTPUT:
%  U: upper triangular matrix
%  b_new: updated b vector
%
%by Nasser Abbasi. Feb 27,2007
```

```
    if nargin ~=2
        error('2 arguments required');
    end

    if ~isnumeric(A)|~isnumeric(b)
        error('A and b must be numeric');
    end

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

    if b_nRow~=nRow
        error('b must be same size as A');
    end

    pivot=A(1,1);
    [nRow,nCol]=size(A);
    for i=2:nRow
        multiplier=A(i,1)/pivot;
        b(i)=b(i)-multiplier*b(1);
        for j=1:nCol
            A(i,j)=A(i,j)-multiplier*A(1,j);
        end
    end

end
```

6.1.5 nma_pentaBackSub.m

```
function x=nma_pentaBackSub(U,b)
%function x=nma_pentaBackSub(U,b)
%
%Does backsub on an upper diagonal U type matrix
%special backsub in that this function can
%detected banded U and will only process the non-zero
%entries in its backsub process
```

```
%
%INPUT:
%  U: special upper triangular matrix. Can be the
%     result of doing special Gaussian elimination on penta-diagonal
%  b: the rhs in the Ux=b
%
%OUTPUT: the solution to Ux=b
%
%by Nasser M. Abbasi. Feb 27,2007

    if nargin ~=2
        error('2 arguments required');
    end

    if ~isnumeric(U)|~isnumeric(b)
        error('U and b must be numeric');
    end

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

    if b_nRow~=nRow
        error('b must be same size as U');
    end

    [nRow,nCol]=size(U);
    n=nRow;
    x=zeros(n,1);
    x(n)=b(n)/U(n,n);

    for i=n-1:-1:1
        r=find(abs(U(i,:))>4*eps);
        row=U(i,r(1):r(end));
        x(i)=(b(i)-row(2:end)*x(i+1:i+length(row)-1))/row(1);
    end

end
```

6.1.6 penta_test.m

```
% This scipt was written to test penta solver for
% MATH 501 computer assignment  CSUF
%by Nasser M. Abbasi. Feb 27,2007

clear all;
format long;

test=1;
fprintf('==========>test %d\n',test);
A=[15  -2 -6    0;
   -2  12 -4   -4;
   -6  -4 19   -9;
   0   -1 -9   21]
b=[300 0 0 0]
fprintf('=======>our result\n');
x=nma_pentaSolve(A,b)
fprintf('=======>Matlab result\n');
A\b'

test=test+1;
fprintf('==========>test %d\n',test);
A=[15   8 -6   0   0   0;
   -2  12 -4  -4   0   0;
   -6  -4 19  -9   4   0;
    0  -1 -9  21   6   7;
    0   0  9  10  11   8;
    0   0  0  10  -2   2]
b=[300 0 0 0 1 2]
fprintf('=======>our result\n');
x=nma_pentaSolve(A,b)
fprintf('=======>Matlab result\n');
A\b'


test=test+1;
fprintf('==========>test %d\n',test);
A=[15   8 -6   0   0   0  0;
   -2  12 -4  -4   0   0  0;
   -6  -4 19  -9   4   0  0;
```

```
    0  -1 -9  21   6   7  0;
    0   0  9  10  11   8  3;
    0   0  0  10  -2   2  4;
    0   0  0  0   -2   2  4]
b=[300 0 0 0 1 2 6]
fprintf('=======>our result\n');
x=nma_pentaSolve(A,b)
fprintf('=======>Matlab result\n');
A\b'

test=test+1;
fprintf('==========>test %d\n',test);
A=[15   8;
   0  12]
b=[300 0]
fprintf('=======>our result\n');
x=nma_pentaSolve(A,b)
fprintf('=======>Matlab result\n');
A\b'


test=test+1;
fprintf('==========>test %d\n',test);
A=[15   8 -6   0   0   0  0  0;
   -2  12 -4  -4   0   0  0  0;
   -6  -4 19  -9   4   0  0  0;
    0  -1 -9  21   6   7  0  0;
    0   0  9  10  11   8  3  0;
    0   0  0  10  -2   2  4  3;
    0   0  0  0   -2   2  4  7;
    0   0  0  0   0    4  8  9]
b=[300 0 0 0 1 2 6 10]
fprintf('=======>our result\n');
x=nma_pentaSolve(A,b)
fprintf('=======>Matlab result\n');
A\b'
```

6.2 Code download

1. nma_backsolve.m
2. nma_penta.m
3. nma_pentaSolve.m
4. nma_gaussian_elimination.m
5. nma_pentaBackSub.m
6. penta_test.m