Mikhail Gaerlan
6 February 2018
MAT 226B Freund

Problem 1

(a)

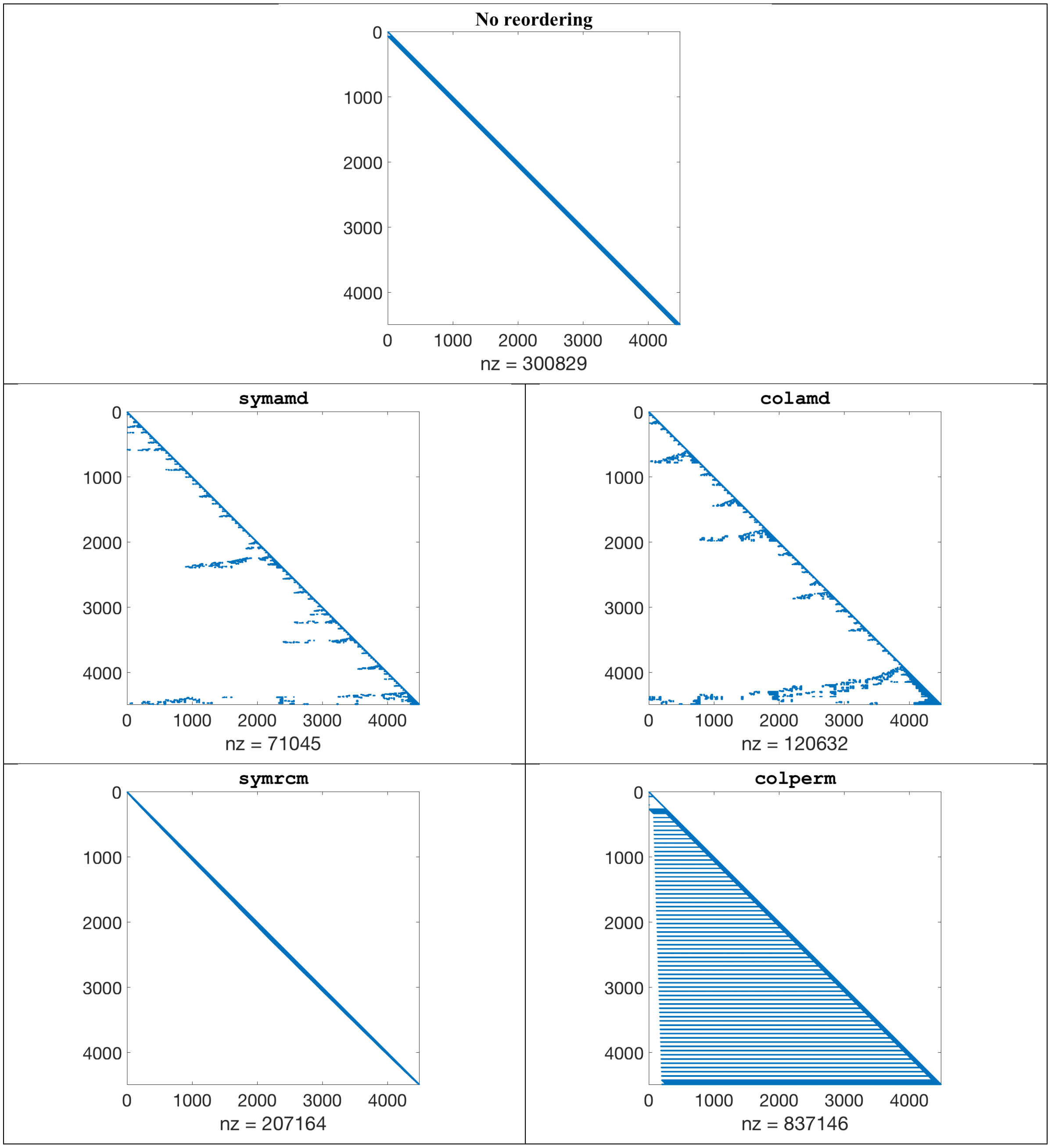
The following code was used to factor the 2D Laplacian:

```
m = 67;nnzs = zeros(5,1);A = make_2d_laplacian(m);

orderings = ["default","symamd","colamd","symrcm","colperm"];n=1;
for order = orderings
    if n == 1; p = 1:m^2;
    else; p = eval(strcat(order,"(A)")); end
    L = chol(A(p,p),'lower');
    nnzs(n) = nnz(L);spy(L);
    if n == 1; title("No reordering",'FontName','Times');
    else; title(order,'FontName','Courier'); end; set(gca,'FontSize',20);
    saveas(gcf,strcat("../Figures/",order),'png');n=n+1;
end; matrix2latex(nnzs,"../Tables/nnzs.tex",'alignment','r')
```

The number of nonzero entries are shown in the following vector:

$$\begin{bmatrix} \text{No reordering} \\ \texttt{symamd} \\ \texttt{colamd} \\ \texttt{symrcm} \\ \texttt{colperm} \end{bmatrix} = \begin{bmatrix} 300829 \\ 71045 \\ 120632 \\ 207164 \\ 837146 \end{bmatrix}.$$

The following table shows the output of `spy(L)`:



Using the following code, my computer ran out of storage after $i = 7$.

```
m0=67;i=0;
while 1
    fileID = fopen('../Tables/i.tex','w');
    fprintf(fileID,int2str(i));fclose(fileID);
    m=2^i*m0;A = make_2d_laplacian(m);
    p = symamd(A);
    L = chol(A(p,p),'lower');
    i = i + 1;
end
```

(b)

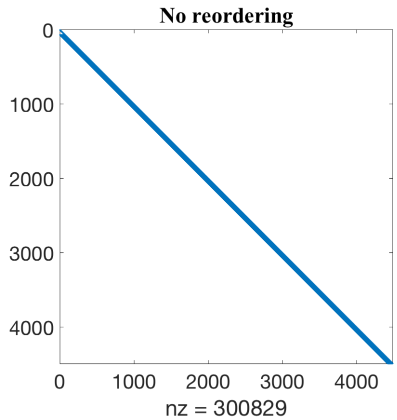
The following code was used to factor the 3D Laplacian:

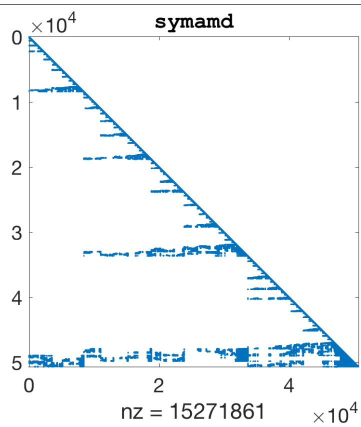
```
m = 37;nnzs = zeros(5,1);A = make_3d_laplacian(m);

orderings = ["default","symamd","colamd","symrcm","colperm"];n=1;
for order = orderings
    if n == 1; p = 1:m^2;
    else; p = eval(strcat(order,"(A)")); end
    L = chol(A(p,p),'lower');
    nnzs(n) = nnz(L);spy(L);
    if n == 1; title("No reordering",'FontName','Times');
    else; title(order,'FontName','Courier'); end; set(gca,'FontSize',20);
    saveas(gcf,strcat("../Figures/",order,"b"),'png');n=n+1;
end; matrix2latex(nnzs,"../Tables/nnzsb.tex",'alignment','r')
```

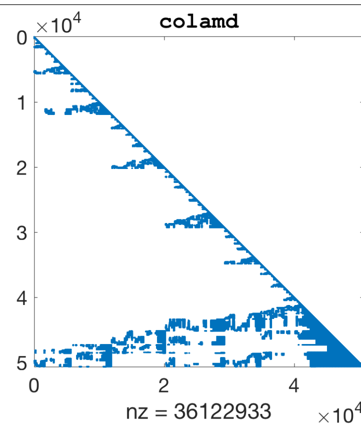
The number of nonzero entries are shown in the following vector:

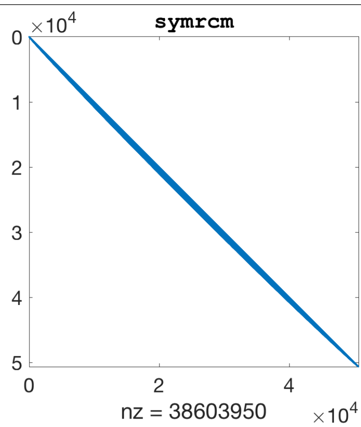
$$\begin{bmatrix} \text{No reordering} \\ \texttt{symamd} \\ \texttt{colamd} \\ \texttt{symrcm} \\ \texttt{colperm} \end{bmatrix} = \begin{bmatrix} 50689 \\ 15271861 \\ 36122933 \\ 38603950 \\ 222256905 \end{bmatrix}.$$

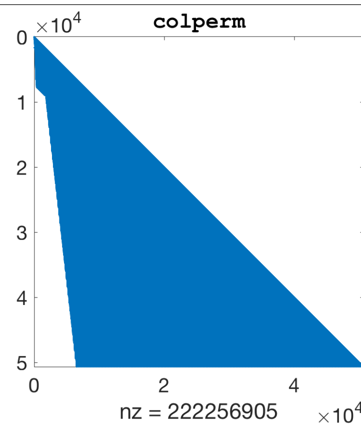
The following table shows the output of `spy(L)`:









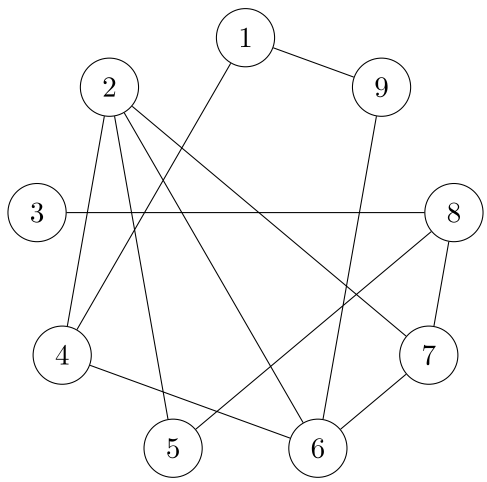


Using the following code, my computer ran out of storage after $i = 2$.

```
m0=37;i=0;
while 1
    fileID = fopen('../Tables/ib.tex','w');
    fprintf(fileID,int2str(i));fclose(fileID);
    m=2^i*m0;A = make_3d_laplacian(m);
    p = symamd(A);
    L = chol(A(p,p),'lower');
    i = i + 1;
end
```

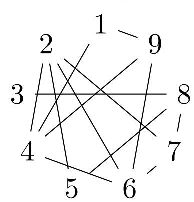
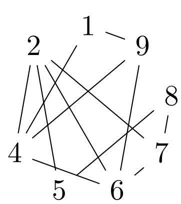
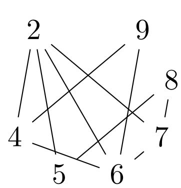
Problem 2

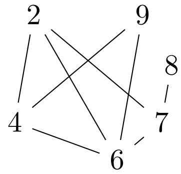
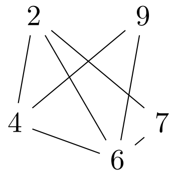
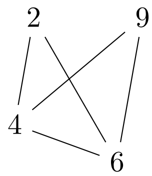
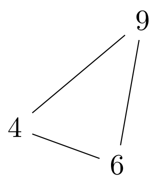
(a)



(b)

Since $a_{9,1} \neq 0, a_{4,1} \neq 0, a_{9,4} = 0$, fill-in element at $a_{9,4}$. Thus, the algorithm goes as follows

Step 0: Fill-in step 	$d = \begin{bmatrix} 1/2 & 1/4 & 1 & 1/4 & 1/2 & 1/4 & 1/3 & 1/3 & 1/3 \end{bmatrix}$
Step 1: Eliminate node 3. 	$p = \begin{bmatrix} 3 \end{bmatrix}$ $d = \begin{bmatrix} 1/2 & 1/4 & * & 1/4 & 1/2 & 1/4 & 1/3 & 1/2 & 1/3 \end{bmatrix}$
Step 2: Eliminate node 1. 	$p = \begin{bmatrix} 3 & 1 \end{bmatrix}$ $d = \begin{bmatrix} * & 1/4 & * & 1/3 & 1/2 & 1/4 & 1/3 & 1/2 & 1/2 \end{bmatrix}$

Step 3:	Eliminate node 5. 	$p = \begin{bmatrix} 3 & 1 & 5 \end{bmatrix}$ $d = \begin{bmatrix} * & 1/3 & * & 1/3 & * & 1/4 & 1/3 & 1 & 1/2 \end{bmatrix}$
Step 4:	Eliminate node 8. 	$p = \begin{bmatrix} 3 & 1 & 5 & 8 \end{bmatrix}$ $d = \begin{bmatrix} * & 1/3 & * & 1/3 & * & 1/4 & 1/2 & * & 1/2 \end{bmatrix}$
Step 5:	Eliminate node 7. 	$p = \begin{bmatrix} 3 & 1 & 5 & 8 & 7 \end{bmatrix}$ $d = \begin{bmatrix} * & 1/2 & * & 1/3 & * & 1/3 & * & * & 1/2 \end{bmatrix}$
Step 6:	Eliminate node 2. 	$p = \begin{bmatrix} 3 & 1 & 5 & 8 & 7 & 2 \end{bmatrix}$ $d = \begin{bmatrix} * & * & * & 1/2 & * & 1/2 & * & * & 1/2 \end{bmatrix}$
Step 7:	Eliminate node 4. 	$p = \begin{bmatrix} 3 & 1 & 5 & 8 & 7 & 2 & 4 \end{bmatrix}$ $d = \begin{bmatrix} * & * & * & * & * & 1 & * & * & 1 \end{bmatrix}$
Step 8:	Eliminate node 6.	$p = \begin{bmatrix} 3 & 1 & 5 & 8 & 7 & 2 & 4 & 6 \end{bmatrix}$
Final:	Eliminate node 9.	$p = \begin{bmatrix} 3 & 1 & 5 & 8 & 7 & 2 & 4 & 6 & 9 \end{bmatrix}$

(c)

$$A = \begin{bmatrix} * & 0 & 0 & * & 0 & 0 & 0 & 0 & 0 \\ 0 & * & 0 & 0 & 0 & 0 & * & 0 & * \\ 0 & 0 & * & * & 0 & * & 0 & 0 & 0 \\ * & 0 & * & * & * & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & * & * & * & 0 & * & 0 \\ 0 & 0 & * & 0 & * & * & * & * & 0 \\ 0 & * & 0 & 0 & 0 & * & * & * & 0 \\ 0 & 0 & 0 & 0 & * & * & * & * & * \\ 0 & * & 0 & 0 & 0 & 0 & 0 & * & * \end{bmatrix}$$

Problem 3

(a)

There would be 17 fill-in elements in positions $u_{2,1}$, $u_{2,2}$, $u_{2,4}$, $u_{4,1}$, $u_{4,2}$, $u_{4,4}$, $u_{5,1}$, $u_{5,2}$, $u_{5,4}$, $u_{7,1}$, $u_{7,4}$, $u_{10,1}$, $u_{10,2}$, $u_{10,4}$, $u_{11,1}$, $u_{11,2}$, $u_{11,4}$.

(b)

There would be a pivot at $u_{2,11}$. After the pivot,

$$U^{(k)} = \begin{bmatrix} * & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & * \\ 0 & * & * & 0 & * & 0 & 0 & 0 & 0 & 0 & * \\ 0 & 0 & 0 & * & * & 0 & 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & 0 & 0 & * & 0 & 0 & * & 0 & * \\ 0 & 0 & 0 & 0 & 0 & 0 & * & * & 0 & 0 & * \\ * & 0 & 0 & * & 0 & 0 & 0 & 0 & 0 & * & 0 \\ 0 & 0 & * & * & 0 & * & 0 & 0 & 0 & * & * \\ 0 & * & 0 & 0 & 0 & * & 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & * & * & 0 & * & 0 \\ 0 & 0 & 0 & * & 0 & 0 & 0 & * & 0 & 0 & * \\ * & 0 & 0 & 0 & 0 & * & 0 & 0 & 0 & 0 & * \end{bmatrix},$$

so there would only be one pivot at $u_{5,11}$.

Problem 4

(a)

The following function was used to find the J,I, and V_L matrix for the compressed sparse column format for a lower-triangular matrix

```
function [J,I,V] = comp_l_tri(L)
%Find the compressed sparse column format of a lower triangular matrix
%   Input  - L  lower triangular matrix
%   Output - J  row indices
%            I  column pointers
%            V  nonzero entries
SL = tril(L,-1);
[J,~,V] = find(SL); I = ones(size(SL,1),1);
for k = 2:length(I); I(k) = I(k-1) + nnz(SL(:,k-1)); end
end
```

The following code was used on the `small_ex.mat` example

```
load('small_ex.mat');
[J,I,V] = comp_l_tri(L);
matrix2latex(J,"../Tables/smallexj.tex",'alignment','r')
matrix2latex(I,"../Tables/smallexi.tex",'alignment','r')
matrix2latex(V,"../Tables/smallexv.tex",'alignment','r','format','%-.15e')
```

and the following code was used for the `large_ex.mat` example

```
load('large_ex.mat');
[J,I,V] = comp_l_tri(L);
matrix2latex([I(50000);I(100000);I(150000);I(200000);I(250000)],"../Tables/largeex.tex",'alignment','r')
```

The output for the `small_ex.mat` example is

$$J = \begin{bmatrix} \texttt{1} \\ \texttt{3} \\ \texttt{4} \\ \texttt{9} \\ \texttt{2} \\ \texttt{4} \\ \texttt{5} \\ \texttt{8} \\ \texttt{9} \\ \texttt{3} \\ \texttt{5} \\ \texttt{7} \\ \texttt{8} \\ \texttt{4} \\ \texttt{6} \\ \texttt{7} \\ \texttt{8} \\ \texttt{9} \\ \texttt{5} \\ \texttt{8} \\ \texttt{9} \\ \texttt{6} \\ \texttt{7} \\ \texttt{8} \\ \texttt{9} \\ \texttt{7} \\ \texttt{8} \\ \texttt{8} \\ \texttt{10} \\ \texttt{9} \\ \texttt{10} \end{bmatrix}, \quad I = \begin{bmatrix} \texttt{1} \\ \texttt{5} \\ \texttt{10} \\ \texttt{14} \\ \texttt{19} \\ \texttt{22} \\ \texttt{26} \\ \texttt{28} \\ \texttt{30} \\ \texttt{31} \\ \texttt{32} \end{bmatrix}, \quad V_L = \begin{bmatrix} \texttt{1.000000000000000e+00} \\ \texttt{8.127286465304295e-01} \\ \texttt{2.578478773046358e-01} \\ \texttt{-7.969322223753756e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{-8.907667695526849e-01} \\ \texttt{2.565826406430549e-03} \\ \texttt{-1.365576562315056e-01} \\ \texttt{9.951206990243782e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{-2.869666020396466e-02} \\ \texttt{7.888955111347864e-01} \\ \texttt{-7.249068104658705e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{8.547124499962497e-01} \\ \texttt{8.349876648322339e-01} \\ \texttt{4.271480231886315e-01} \\ \texttt{2.366747672438800e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{8.720546533795395e-01} \\ \texttt{-7.504519186790148e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{2.929548648516276e-01} \\ \texttt{6.663039713385901e-01} \\ \texttt{-2.034355435624491e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{6.704410209562610e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{1.045232337167099e-01} \\ \texttt{1.000000000000000e+00} \\ \texttt{1.000000000000000e+00} \end{bmatrix}$$

and `large_ex.mat` example is

$$\begin{bmatrix} I(50000) \\ I(100000) \\ I(150000) \\ I(200000) \\ I(250000) \end{bmatrix} = \begin{bmatrix} \texttt{376955} \\ \texttt{1762353} \\ \texttt{4716326} \\ \texttt{12466445} \\ \texttt{15513326} \end{bmatrix}.$$

(b)

The following function was used to find the J,I, and V_L matrix for the compressed sparse column format for an upper-triangular matrix

```
function [J,I,V] = comp_u_tri(U)
%Find the compressed sparse column format of a upper triangular matrix
%   Input  - U  upper triangular matrix
%   Output - J  row indices
%            I  column pointers
%            V  nonzero entries
[J,~,V] = find(U); I = ones(size(U,1)+1,1);
for k = 2:length(I); I(k) = I(k-1) + nnz(U(:,k-1)); end
end
```

The following code was used on the `small_ex.mat` example

```
load('small_ex.mat');
[J,I,V] = comp_u_tri(U);
matrix2latex(J,"../Tables/smallexuj.tex",'alignment','r')
matrix2latex(I,"../Tables/smallexui.tex",'alignment','r')
matrix2latex(V,"../Tables/smallexuv.tex",'alignment','r','format','%-.15e'
    )
```

and the following code was used for the `large_ex.mat` example

```
load('large_ex.mat');
[J,I,V] = comp_u_tri(U);
matrix2latex([I(50000);I(100000);I(150000);I(200000);I(250000)],"../Tables
    /largeexu.tex",'alignment','r')
```

The output for the `small_ex.mat` example is

$$
J = \begin{bmatrix} \texttt{1} \\ \texttt{2} \\ \texttt{1} \\ \texttt{3} \\ \texttt{1} \\ \texttt{2} \\ \texttt{4} \\ \texttt{2} \\ \texttt{3} \\ \texttt{5} \\ \texttt{4} \\ \texttt{6} \\ \texttt{3} \\ \texttt{4} \\ \texttt{6} \\ \texttt{7} \\ \texttt{2} \\ \texttt{3} \\ \texttt{4} \\ \texttt{5} \\ \texttt{6} \\ \texttt{7} \\ \texttt{8} \\ \texttt{1} \\ \texttt{2} \\ \texttt{4} \\ \texttt{5} \\ \texttt{6} \\ \texttt{9} \\ \texttt{8} \\ \texttt{10} \end{bmatrix}, \quad I = \begin{bmatrix} \texttt{1} \\ \texttt{2} \\ \texttt{4} \\ \texttt{7} \\ \texttt{10} \\ \texttt{12} \\ \texttt{16} \\ \texttt{23} \\ \texttt{29} \\ \texttt{31} \\ \texttt{32} \end{bmatrix}, \quad V_L = \begin{bmatrix} \texttt{-7.227965685152800e-01} \\ \texttt{1.764187707789873e-01} \\ \texttt{-2.676863990901244e-01} \\ \texttt{6.135190893222113e-01} \\ \texttt{7.561571552311852e-03} \\ \texttt{-2.081132255329154e-02} \\ \texttt{7.540974467700878e-01} \\ \texttt{-2.937163741220890e-01} \\ \texttt{-1.011128868565034e-01} \\ \texttt{9.270605736868538e-01} \\ \texttt{-9.154044041709144e-01} \\ \texttt{9.459166682812694e-01} \\ \texttt{-6.215863137552480e-01} \\ \texttt{3.342406000801499e-01} \\ \texttt{1.728792293578358e-01} \\ \texttt{3.502248328102306e-01} \\ \texttt{-2.779559016106783e-01} \\ \texttt{2.405568541421710e-01} \\ \texttt{6.223017702005704e-01} \\ \texttt{-9.614850451717172e-01} \\ \texttt{-8.322529834342003e-01} \\ \texttt{9.496033343697807e-01} \\ \texttt{3.026990648307066e-01} \\ \texttt{-5.375243676712955e-01} \\ \texttt{-1.930177137508202e-01} \\ \texttt{-7.559589634957409e-01} \\ \texttt{-4.631223572054337e-01} \\ \texttt{-4.843076597747906e-01} \\ \texttt{-3.366695225147414e-01} \\ \texttt{-6.955319742741071e-01} \\ \texttt{-3.039846805677731e-01} \end{bmatrix}
$$

and `large_ex.mat` example is

$$
\begin{bmatrix} I(50000) \\ I(100000) \\ I(150000) \\ I(200000) \\ I(250000) \end{bmatrix} = \begin{bmatrix} \texttt{209895} \\ \texttt{1442487} \\ \texttt{4084327} \\ \texttt{14906857} \\ \texttt{17772355} \end{bmatrix}.
$$

(c)

The following functions were written to solve the equations $Lc = b$ and $Ux = c$ respectively

```
function c = solve_l(J,I,V,b)
%Solve Lc = b where L is unit lower-triangular
%    Input   - J   row indices
%              I   pointers
%              V   nonzero entries
%              b   right-hand side
%    Output  - c   solution
c = b;
for k = 1:(length(I)-1)
    indices = I(k):(I(k+1)-1); rows = J(indices);
    c(rows) = c(rows) - V(indices)*c(k);
end
end
```

```
function x = solve_u(J,I,V,c)
%Solve Ux = c where U is a nonsingular upper-triangular
%    Input   - J   row indices
%              I   pointers
%              V   nonzero entries
%              c   right-hand side
%    Output  - x   solution
x = c;
for k = (length(I)-1):-1:2
    indices = I(k):(I(k+1)-1);rows = J(indices);
    x(k) = x(k)/V(indices(end));
    x(rows(1:end-1)) = x(rows(1:end-1)) - V(indices(1:end-1))*x(k);
end; x(1) = x(1)/V(1);
end
```

The following code was used on the `small_ex.mat` example

```
load('small_ex.mat');
[J,I,V] = comp_l_tri(L);
c = solve_l(J,I,V,b);
[J,I,V] = comp_u_tri(U);
x = solve_u(J,I,V,c)
matrix2latex(x,"../Tables/smallexsolve.tex",'alignment','r','format','
    %-.15e')
```

and the following code was used for the `large_ex.mat` example

```
load('large_ex.mat');
[J,I,V] = comp_l_tri(L);
c = solve_l(J,I,V,b);
[J,I,V] = comp_u_tri(U);
x = solve_u(J,I,V,c);
matrix2latex([x(50000);x(100000);x(150000);x(200000);x(250000)],"../Tables
    /largeexsolve.tex",'alignment','r','format','%-.15e')
```

The output for the `small_ex.mat` example is

$$x = \begin{bmatrix} \texttt{-7.368997733798020e+00} \\ \texttt{-3.412065799871883e+01} \\ \texttt{2.493451030044967e+01} \\ \texttt{-2.682899840821154e+01} \\ \texttt{-9.180874775319783e+00} \\ \texttt{-1.237568442840570e+01} \\ \texttt{2.593887743663491e+01} \\ \texttt{-6.717855384133224e+00} \\ \texttt{-4.296837993924318e+00} \\ \texttt{-1.097654778394844e+00} \end{bmatrix}$$

and `large_ex.mat` example is

$$\begin{bmatrix} x(50000) \\ x(100000) \\ x(150000) \\ x(200000) \\ x(250000) \end{bmatrix} = \begin{bmatrix} \texttt{-7.306985715493668e+04} \\ \texttt{-5.686028360745258e+05} \\ \texttt{5.850981452463222e+04} \\ \texttt{-6.238599598901578e+04} \\ \texttt{4.381939017807725e+06} \end{bmatrix}.$$

Problem 5

The following code was used.

```
for k = ["large_ex1","large_ex2"]
    clearvars -except k
    load(strcat(k,".mat"));n=length(b);
    for perm = ["default","colamd","colperm"]
        for scaling = ["default","scaling"]
            if perm == "default"; p0=1:size(A,1);
            else; p0 = eval(strcat(perm,"(A)")); end
            p0i(p0)=1:n;D=speye(n,n);
            if scaling == "default";[L,U,p,q]=lu(A(:,p0),'vector');
            else; [L,U,p,q,D] = lu(A(:,p0),'vector'); end; qi(q)=1:n;
            c = D\b; c = c(p);
            [J,I,V] = comp_l_tri(L); v = solve_l(J,I,V,c);
            [J,I,V] = comp_u_tri(U); x = solve_u(J,I,V,v);
            x = x(qi); x=x(p0i);
            matrix2latex([nnz(L);nnz(U);norm(b-A*x)/norm(b);x(1);x(30000);
               x(70000);x(140000);x(200002)],strcat("../Tables/",k,"_",
               scaling,"_",perm,".tex"),'alignment','r','format','%-.15e')
        end
    end
end
```

The output for `large_ex1.mat`

Ex. 1: (i)

	Without scaling	With scaling
`nnz`(L)	4.816814200000000e+07	2.607293400000000e+07
`nnz`(U)	7.515763800000000e+07	4.119203700000000e+07
$\|\|b-Ax\|\|_2/\|\|b\|\|_2$	2.828029206810877e-12	6.790398425331222e-14
$x(1)$	3.724469259680818e-01	3.724469259680818e-01
$x(30000)$	-9.486769919656040e-01	-9.486769948691244e-01
$x(70000)$	5.609874735302742e-02	5.609874645819704e-02
$x(140000)$	5.158011280782167e-02	5.158011140559619e-02
$x(200002)$	3.492542265989640e-02	3.492542265861234e-02

Ex. 1: (ii)

	Without scaling	With scaling
`nnz`(L)	4.810847900000000e+07	2.308868800000000e+07
`nnz`(U)	1.027379140000000e+08	3.763303400000000e+07
$\|\|b-Ax\|\|_2/\|\|b\|\|_2$	1.067252484791601e-11	8.116366886591767e-14
$x(1)$	3.724469259680818e-01	3.724469259680818e-01
$x(30000)$	-9.486769975350847e-01	-9.486769973752108e-01
$x(70000)$	5.609874791210737e-02	5.609874620224603e-02
$x(140000)$	5.158011150167822e-02	5.158011103064770e-02
$x(200002)$	3.492542265822163e-02	3.492542265822110e-02

Ex. 1: (iii)

	Without scaling	With scaling
`nnz`(L)	4.129774200000000e+07	2.296480200000000e+07
`nnz`(U)	8.125790400000000e+07	3.368607300000000e+07
$\|\|b-Ax\|\|_2/\|\|b\|\|_2$	2.374530280936845e-14	5.125021168696811e-14
$x(1)$	3.724469259680818e-01	3.724469259680818e-01
$x(30000)$	-9.486769978902629e-01	-9.486769983414199e-01
$x(70000)$	5.609874623542198e-02	5.609874579025122e-02
$x(140000)$	5.158011102075557e-02	5.158011071012165e-02
$x(200002)$	3.492542265976455e-02	3.492542265839199e-02

The output for `large_ex2.mat`

Ex. 2: (i)

	Without scaling	With scaling
nnz(L)	3.188442500000000e+07	3.033252700000000e+07
nnz(U)	4.444986200000000e+07	4.354803300000000e+07
$\|\|b-Ax\|\|_2/\|\|b\|\|_2$	1.650718270551992e-16	2.582444141156649e-14
$x(1)$	-5.870862540210242e-01	-5.870862540907285e-01
$x(30000)$	-9.401644284862181e-01	-9.401644284864308e-01
$x(70000)$	6.591485582451860e-02	6.591485582454848e-02
$x(140000)$	8.626100798318471e-01	8.626097937227244e-01
$x(200002)$	-2.718282595836798e-02	-2.718282681590964e-02

Ex. 2: (ii)

	Without scaling	With scaling
nnz(L)	2.717166000000000e+07	2.652711600000000e+07
nnz(U)	4.037528300000000e+07	3.958078600000000e+07
$\|\|b-Ax\|\|_2/\|\|b\|\|_2$	1.714118657717020e-16	6.751642453866837e-14
$x(1)$	-5.870862541863805e-01	-5.870862540897915e-01
$x(30000)$	-9.401644284839612e-01	-9.401644284861428e-01
$x(70000)$	6.591485582468770e-02	6.591485582458421e-02
$x(140000)$	8.626105435293228e-01	8.626099989587699e-01
$x(200002)$	-2.718282600362049e-02	-2.718282655860763e-02

Ex. 2: (iii)

	Without scaling	With scaling
nnz(L)	3.188122100000000e+07	3.028285400000000e+07
nnz(U)	4.445873300000000e+07	4.349419700000000e+07
$\|\|b-Ax\|\|_2/\|\|b\|\|_2$	1.688597557702839e-16	2.817217290485666e-14
$x(1)$	-5.870862540540591e-01	-5.870862540766120e-01
$x(30000)$	-9.401644284873146e-01	-9.401644284863432e-01
$x(70000)$	6.591485582456863e-02	6.591485582440532e-02
$x(140000)$	8.626094027729604e-01	8.626095802218049e-01
$x(200002)$	-2.718282610437903e-02	-2.718282640176180e-02