Brian KYANJO
Homework 5
Math 566

1. Gaussian Elimination for a Structured matrix.

Determine the number of operations.

Addition	Subtraction	Multiplication	Division
0	$n-2$	$n-2$	$n-2$
0	$n-1$	$n-1$	$n-1$
0	$n-1$	$n-1$	$n-1$
0	$n-2$	$n-2$	$n-2$
+ n	n	n	n
n	$5n-6$	$5n-6$	$5n-6$

Total operations $= n + 5n-6 + 5n-6 + 5n-6$
$= \underline{(16n-18)}$ operations

No. 2

a) Derive a fast algorithm for solving these system using a similar approach done in class.

Consider

$$\begin{bmatrix} a_1 & c_1 & e_1 & & & \\ b_1 & a_2 & c_2 & e_2 & & \\ d_1 & b_2 & a_3 & c_3 & e_3 & \\ & d_2 & b_3 & a_4 & c_4 & e_4 \\ & & & & & \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ \end{bmatrix}$$

Step 1: Upper triangular form,

for $i=1$ to $n-2$ do

$b_i' = b_i / a_i$

$d_i' = \dfrac{d_i}{a_i}$

$a_{i+1} = a_{i+1} - b_i' c_i$

$c_{i+1} = c_{i+1} - b_i' e_i$

$b_{i+1} = b_{i+1} - d_i' c_i$

$a_{i+2} = a_{i+2} - d_i' e_i$

$f_{i+1} = f_{i+1} - b_i' f_i$

$f_{i+2} = f_{i+2} - d_i' f_i$

end for

$$a_n = a_n - \frac{b_{n-1}}{a_{n-1}} c_{n-1}$$

$$f_n = f_n - \frac{b_{n-1}}{a_{n-1}} f_{n-1}$$

$$x_n = \frac{f_n}{a_n}$$

$$x_{n-1} = \frac{(f_{n-1} - c_{n-1} x_n)}{a_{n-1}}$$

Step 2: Back substitution

for $i = n-2$ to -1 to 1 do

$$x_i = \frac{(f_i - c_i x_{i+1} - e_i x_{i+2})}{a_i}$$

end for

b) Determine the exact number of operations your algorithm requires for solving a general n-by-n penta diagonal system.

Additions	Subtractions	Multiplications	Divisions
$6(n-2)$	$6(n-2)$	$6(n-2)$	$2(n-2)$
0	3	3	4
0	$2(n-2)$	$2(n-2)$	$(n-2)$
+			
$6(n-2)$	$8n-13$	$8n-13$	$3n-2$

Total operations $= 6(n-2) + 8(n-13) + 8(n-13) + (3n-2)$
$= 25n - 202$ operations.

No. 4

Derive a procedure for computing the entries in $\tilde{R}$ and D for this new $A = \tilde{R}^T D \tilde{R}$ decomposition

Consider $A \in \mathbb{R}^{4\times4}$

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{12} & a_{22} & a_{23} & a_{24} \\ a_{13} & a_{23} & a_{33} & a_{34} \\ a_{14} & a_{24} & a_{34} & a_{44} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ r_{12} & 1 & 0 & 0 \\ r_{13} & r_{23} & 1 & 0 \\ r_{14} & r_{24} & r_{34} & 1 \end{bmatrix}}_{\tilde{R}^T} \underbrace{\begin{bmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 \\ 0 & 0 & d_3 & 0 \\ 0 & 0 & 0 & d_4 \end{bmatrix}}_{D} \underbrace{\begin{bmatrix} 1 & r_{12} & r_{13} & r_{14} \\ 0 & 1 & r_{23} & r_{24} \\ 0 & 0 & 1 & r_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\tilde{R}}$$

Step 1

$d_{11} = a_{11}$ and $r_{1j} d_{11} = a_{1j}$

$r_{1j} = \frac{1}{d_{11}}(a_{1j})$, $j = 2, 3, \ldots, n$

step 2

$a_{22} = r_{12}^2 d_{11} + d_{22}$ and $r_{12} r_{1j} d_{11} + r_{2j} d_{22} = a_{2j}$

$d_{22} = a_{22} - r_{12}^2 d_{11}$ $\qquad r_{2j} = \frac{1}{d_{22}}(a_{2j} - r_{12} r_{1j} d_{11})$,

$j = 3, 4, 5, \ldots, n$

Step 3

$a_{33} = r_{13}^2 d_{11} + r_{23}^2 d_{22} + d_{33}$

d_{33}

$$d_{33} = \left(a_{33} - (r_{13}^2 d_{11} + r_{23}^2 d_{22})\right) \quad \text{and} \quad a_{3j} = r_{13} r_{1j} d_{11} + r_{23} r_{2j} d_{22} + r_{3j} d_{33}$$

$$r_{3j} = \frac{1}{d_{33}} \left(a_{3j} - (r_{13} r_{1j} d_{11} + r_{23} r_{2j} d_{22})\right)$$

$$j = 4, 5, 6, \dots, n$$

Repeats

step(n)

$$d_{nn} = \left(a_{nn} - \left(r_{1n}^2 d_{11} + r_{2n}^2 d_{22} + r_{3n}^2 d_{33} + \dots + r_{n-1,n}^2 d_{n-1,n-1}\right)\right)$$

Pseudocode

$d_{11} = a_{11}$

for $k = 1$ to n do

  $d_{kk} = \left(a_{kk} - \sum_{i=1}^{k-1} r_{ik}^2 d_{ii}\right)$

  for $j = k+1$ to n do

    $r_{kj} = \frac{1}{d_{kk}} \left(a_{kj} - \left(\sum_{i=1}^{k-1} r_{ik} r_{ij} d_{ii}\right)\right)$

  end

end

No 5
(Sherman-Morrison formula)

(a) Suppose you have a fast algorithm for solving $Ay = c$. Explain how to use this algorithm to design a fast algorithm for solving $(A - uv^T)x = b$

- Assume the fast algorithm in solving $Ay = c$ costs $O(n^2)$.
- To solve $(A-uv^T)x = b$ in $O(n^2)$ time, we are going to do the following.
- Since $Ax - uv^T x = b$ differs from $Ay = c$ by just subtracting uv^T for some columns vectors u and v.
- The algorithm can be obtained using Sherman-Morrison formula as ~~form~~ follows.

$$\left.\begin{array}{ll} 1.\ \text{Solve} & Az = b \Rightarrow z = A^{-1}b \\ 2.\ \text{Solve} & Ay = u \Rightarrow y = A^{-1}u \end{array}\right\} O(n^2)$$

$$\left.\begin{array}{ll} 3.\ \text{Compute} & \alpha = v^T y \\ 4.\ \text{Compute} & \beta = v^T z \\ 5.\ \text{Compute} & x = z + \dfrac{\beta}{1-\alpha} y \end{array}\right\} O(n)$$

So $O(n^2) + O(n) = O(n^2)$

Hence the fast algorithm solves $(A-uv^T)x = b$ in $O(n^2)$, since A is already factored.

therefore from Sherman-Morrison formula

$x = z + \frac{\beta}{\alpha} y$

$$x = \left(A^{-1} + \left(\frac{A^{-1}uv^TA^{-1}}{1 - v^TA^{-1}u}\right)\right)b$$

$$x = (A - uv^T)^{-1}b$$

b) Identify the vectos u and v in ~~from~~ this system.

$$u = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad v = \begin{bmatrix} 2 \\ 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix}$$

. No.1

```
%Gaussian elimination for a structured matrix
%Devise an efficient way to arrange the computations for solving an n-by-n
%linear system with non-zero entries in the coefficient matrix only in the
%first and last rows and columns and also in the two main diagonals.

clear all
close all

m = 9; n = 9;

%Sample matrix A, to check the algorithm
A = matrix(m,n);
%seed
rng('default')
s = rng
b = randn(n,1);

x = A\b

[a,x] = guas(A,b)


%Gauss elimination
function [a,x] = guas(A,b)

     a = [A,b];
    [n,m] = size(a);

    %forward sub
    for i = 2:n-1
           if n-i+1 ~= i
               temp = a(i,:)*a(n-i+1,i);
               a(n-i+1,:) = a(n-i+1,:) - temp/a(i,i);
           %else
              % break
           end
    end

    for j = 1:n-1
        a(n,:) = a(n,:) - a(j,:)*a(n,j)/a(j,j);
    end

    %swap first row with the last one
    temp = a(1,:);
    a(1,:) = a(n,:);
    a(n,:) = temp;

    for j = 2:n
        a(j,:) = a(j,:) - a(1,:)*a(j,1)/a(1,1);
    end

    for j = 2:n-1
        a(n,:)  = a(n,:) - a(j,:)*a(n,j)/a(j,j);
    end

    %back sub
    x = zeros(n,1);
    x(n) = a(n,m)/a(n,n);
```

```
    for i = n-1:-1:1
        temp = a(i,n)*x(n);
        x(i) = (a(i,m) - temp)/a(i,i);
    end
end

%unstructured matrix
function A = matrix(m,n)
    rng('default')
    s = rng
    A = diag(randn(n,1));
    A = fliplr(A);
    for j = 1:m
       for i = 1:n
           if i == j
                A(j,i) = randn(1);
           elseif i == 1
                A(j,i) = randn(1);
           elseif j == 1
                A(j,i) = randn(1);
           elseif i == n
                A(j,i) = randn(1);
           elseif j == n
                A(j,i) = randn(1);
           end
       end
    end

end
```

```
s =

  struct with fields:

     Type: 'twister'
     Seed: 0
    State: [625×1 uint32]


s =

  struct with fields:

     Type: 'twister'
     Seed: 0
    State: [625×1 uint32]


x =

    0.2973
   -5.7976
   -3.5566
   -3.4111
   11.4735
    1.7523
    2.4278
```

```
    3.8738
    3.7633


a =

  Columns 1 through 7

  -26.0704         0         0         0         0         0         0
         0    1.4172         0         0         0         0         0
         0         0    0.7172         0         0         0         0
         0         0         0    1.0347         0         0         0
         0         0         0         0    0.2939         0         0
         0         0         0         0         0   -0.0574         0
         0         0         0         0         0         0   -4.3098
    0.0000         0         0         0         0         0         0
   -0.0000         0         0         0         0         0         0

  Columns 8 through 10

         0    5.0366   11.2032
         0    2.7878    2.2750
         0    0.2824   -1.4881
         0    2.9355    7.5177
         0   -0.8459    0.1884
         0    0.1408    0.4293
         0    2.1265   -2.4607
   -1.1983    1.2050   -0.1074
         0    2.7722   10.4327


x =

    0.2973
   -5.7976
   -3.5566
   -3.4111
   11.4735
    1.7523
    2.4278
    3.8738
    3.7633
```

The algorithm i implemented works perfect with O(n), it produces the same results as the direct solver.

Published with MATLAB® R2020b

NO.2

```
% Function isolves a pentadiagonal linear system
% input : a,b,c,d,e,and f
%output : solution of the system x

function [x] = pentadiagonal(a,b,c,d,e,f)

    n = length(a);
    bp = zeros(n-2,1);
    dp = zeros(n-2,1);
    x  = zeros(n,1);
    for i = 1:n-2
        bp(i) = b(i)/a(i);
        dp(i) = d(i)/a(i);
        %coefficients
        a(i+1) = a(i+1) - bp(i)*c(i);
        c(i+1) = c(i+1) - bp(i)*e(i);
        b(i+1) = b(i+1) - dp(i)*c(i);
        a(i+2) = a(i+2) - dp(i)*e(i);
        %left hand side
        f(i+1) = f(i+1) - bp(i)*f(i);
        f(i+2) = f(i+2) - dp(i)*f(i);
    end

    a(n) = a(n) - (b(n-1)/a(n-1))*c(n-1);
    f(n) = f(n) - (b(n-1)/a(n-1))*f(n-1);
    %backward subsitition
    x(n) = f(n)/a(n);
    x(n-1) = (f(n-1) - c(n-1)*x(n))/a(n-1);

    for i = n-2:-1:1
        x(i) = (f(i) - c(i)*x(i+1) - e(i)*x(i+2))/a(i);
    end
end
```

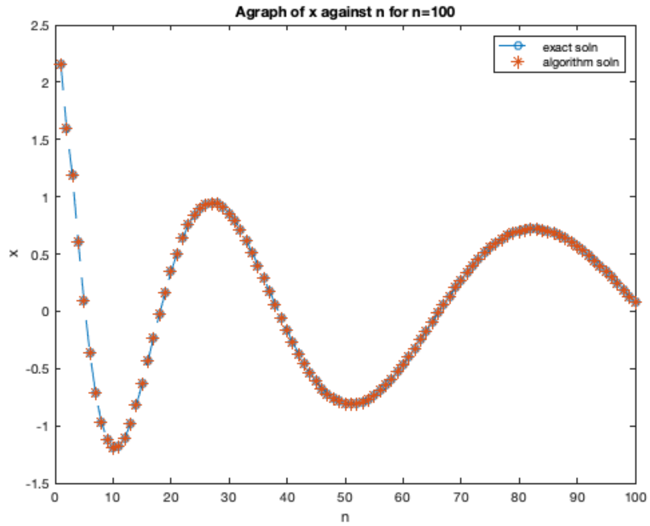
```
Not enough input arguments.

Error in pentadiagonal (line 8)
    n = length(a);
```

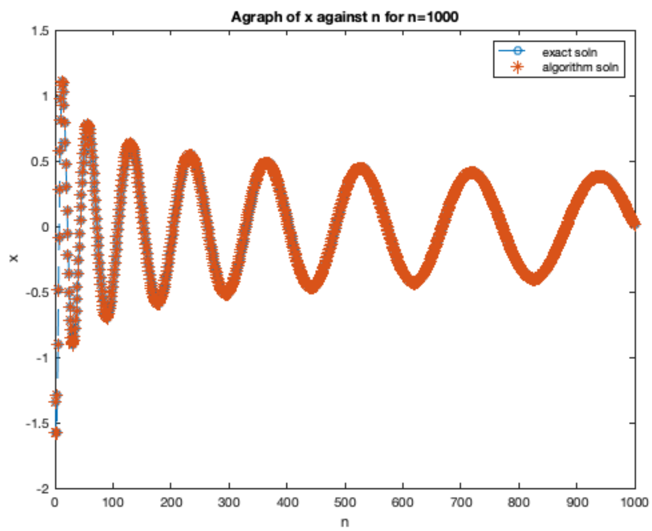
Published with MATLAB® R2020b

No.2d

```
fprintf('No2(d).\n\n');
% Test yoyr code from part (c)
clear all;
close all;

%dimensions
n1 = 100; n2 = 1000;

%solutions from the algorithm and using direct solver
[A1,xa1,f1] = algorithm(n1);
x1=A1\f1;

[A2,xa2,f2] = algorithm(n2);
x2=A2\f2;

%plotting
figure(1)
plot(x1,'--o')
hold on
plot(xa1,'*')
legend('exact soln','algorithm soln');
xlabel('n');ylabel('x');
title('Agraph of x against n for n=100')

figure(2)
%plotting
plot(x2,'--o')
hold on
plot(xa2,'*')
legend('exact soln','algorithm soln');
xlabel('n');ylabel('x');
title('Agraph of x against n for n=1000')

fprintf('As seen from the graphs above their is no difference between the exact solution from the solver \n and from the algorithm, hence it gives the

function [A,x,f] = algorithm(n)
    %coefficients
    i = [1:n]'; a = i;
    j = [1:n-1]'; b = -(j+1)/3; c = b;
    k = [1:n-2]'; d = -(k+2)/6; e = d;

    %RHS
    l = [3:n-2]'; fi(l-2) = 0;
    f = [1/2 1/6 fi 1/6 1/2]';

    [x] = pentadiagonal(a,b,c,d,e,f);

    %Coefficient matrix
    A = diag(d,-2) + diag(b,-1) + diag(a) + diag(c,1) + diag(e,2);
end
```

```
No2(d).

As seen from the graphs above their is no difference between the exact solution from the solver
 and from the algorithm, hence it gives the correct answer.
```





Published with MATLAB® R2020b

No.3

```
%Cholesky for sparse matrices
close all;
clear all;

fprintf('No.3(a)\n\n');
%Load the matrix
load bcsstk38;
A = Problem.A;

fprintf('Make a spy plot of the matrix A showing its sparsity pattern \n\n');
figure(1)
spy(A)
title('Sparsity pattern of A');
ylabel('n');xlabel('n');

fprintf('Compute the sparsity ratio of A\n\n');

sparsity_ratio = 1 - nnz(A)/numel(A);

fprintf('The sparsity ratio is %f \n\n',sparsity_ratio);

fprintf('No.3(b)\n\n');
%Compute the Cholesky decomposition of the matrix from part (a)
R = chol(A);

%Plot the sparsity pattern of the upper triangular matrix,R, from the decomposition.
figure(2)
spy(R)
title('Sparsity pattern of the cholesky decomposition of A');
ylabel('n');xlabel('n');

%Compute the amount of “fill-in” from the Cholesky decomposition.
fillin = nnz(R)/nnz(A);
fprintf('The fill-in is %f \n\n',fillin);

fprintf('No.3(c)\n\n');
s = symrcm(A); S=A(s,s);
R2 = chol(S);
fillin2 = nnz(R2)/nnz(S);
fprintf('The fill-in of the permuted A is %f \n\n',fillin2);

fprintf('The fill-in for the permuted matrix is small than that for the original A.\n\n')

figure(3)
spy(S)
title('Sparsity pattern of the Permuted A');
ylabel('n');xlabel('n');

figure(4)
spy(R2)
title('Sparsity pattern of Cholesky decomposition of the Permuted matrix A');
ylabel('n');xlabel('n');
```

```
No.3(a)

Make a spy plot of the matrix A showing its sparsity pattern
```

Compute the sparsity ratio of A

The sparsity ratio is 0.994490

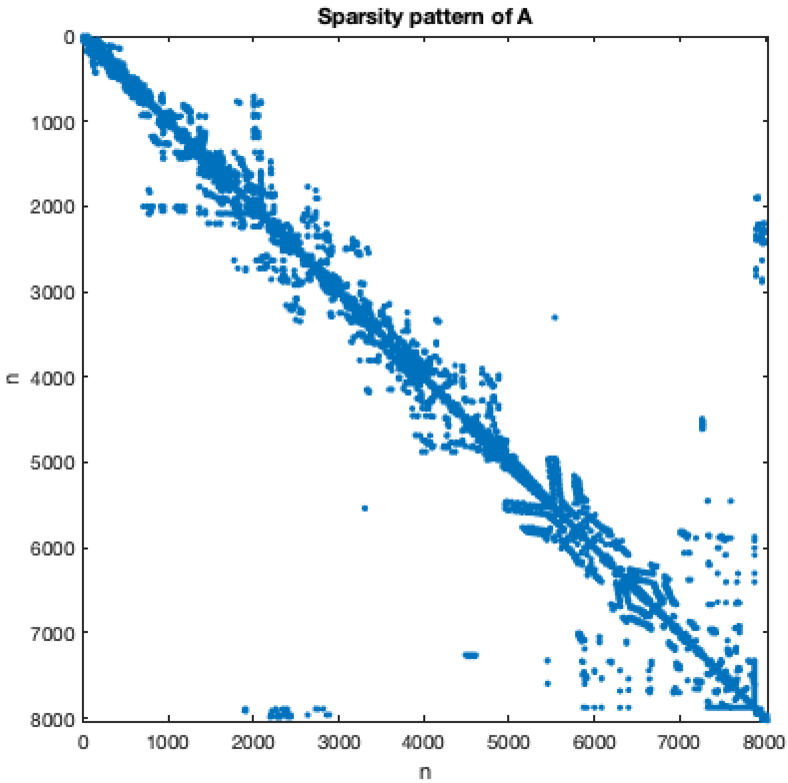
No.3(b)

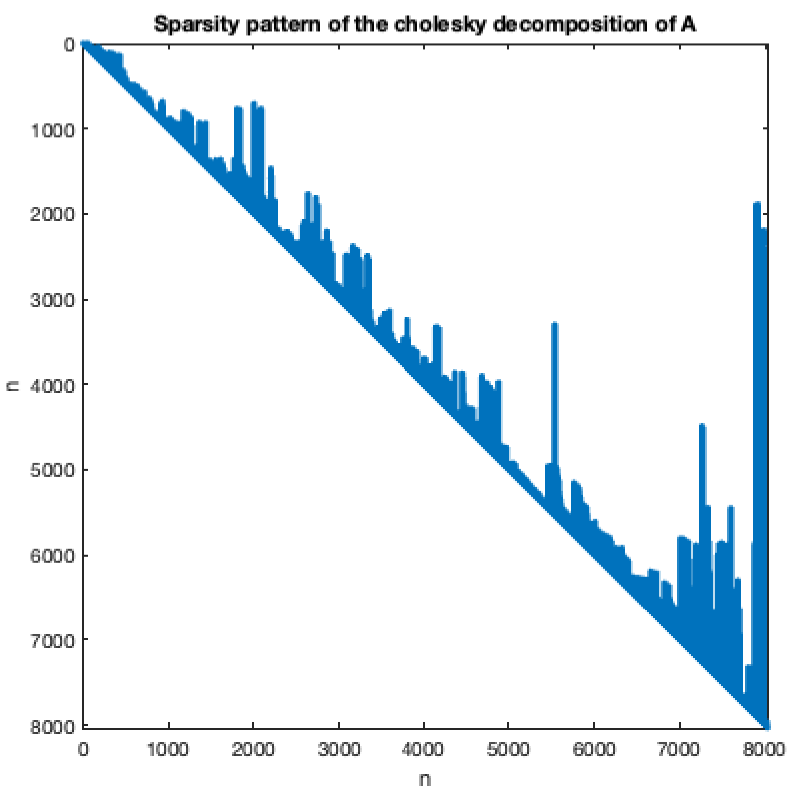
The fill-in is 4.738485

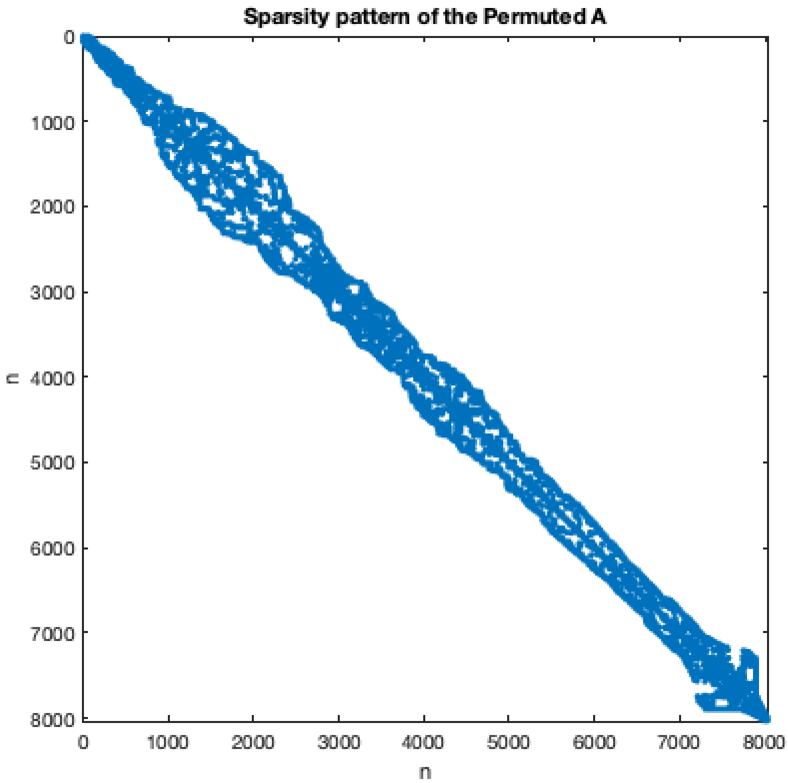
No.3(c)

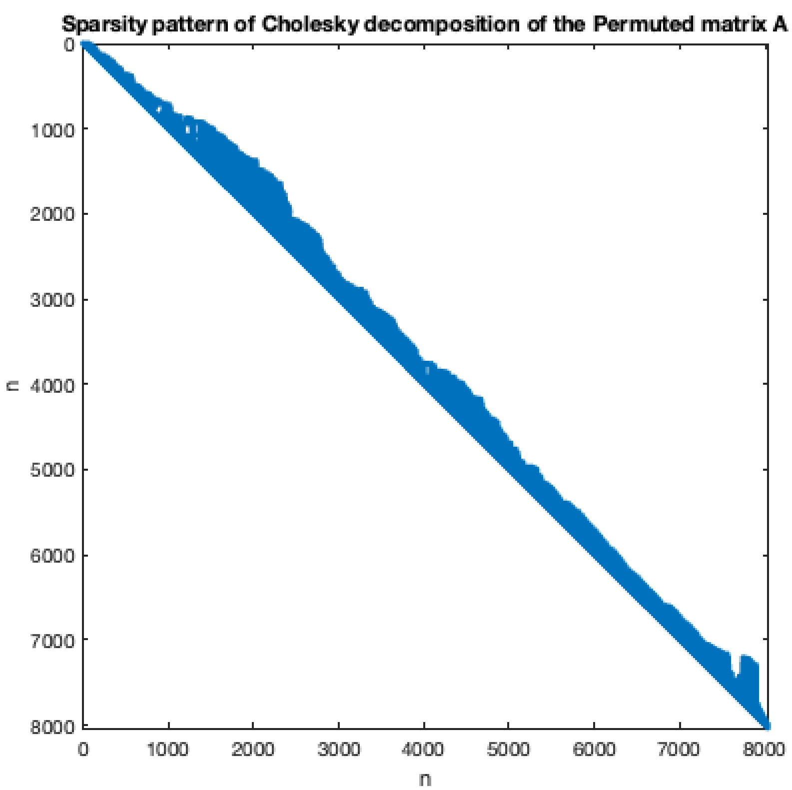
The fill-in of the permuted A is 4.038452

The fill-in for the permuted matrix is small than that for the original A.


Sparsity pattern of A
n
n


Sparsity pattern of the cholesky decomposition of A
n
n





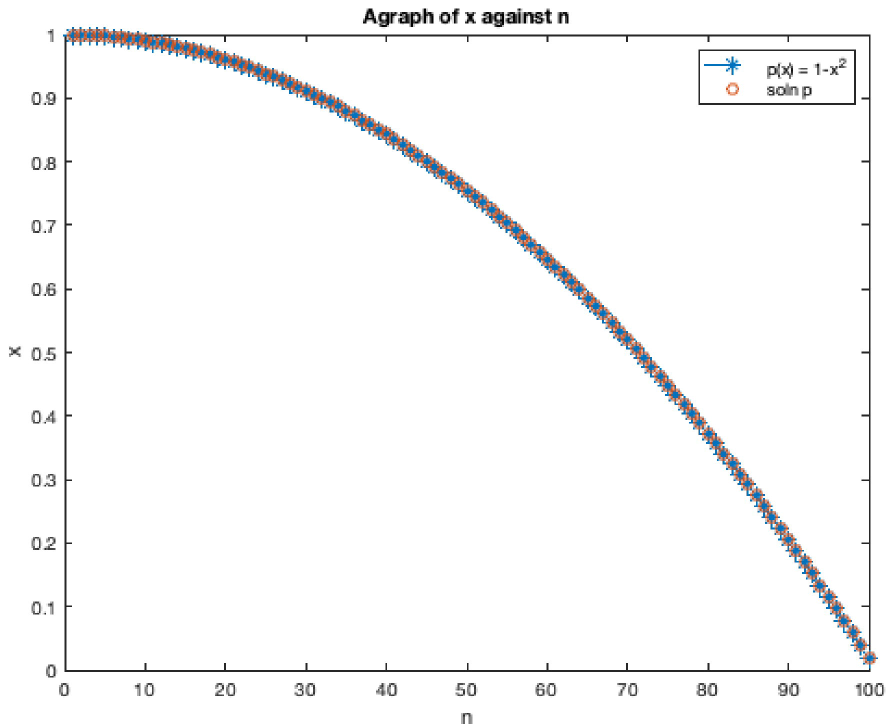
Published with MATLAB® R2020b

No.5

```
fprintf('No.5(c)\n\n');
close all;
clear all;
% Write a code that solves the linear system
%function f(x)
f = @(x) -2*x.^0;

beta = 0; gama = 2/3;
n = 100;

h = 1/(n+1);

j = [1:n]';
xj = j*h;

%RHS b
b1 = (h^2)*f(xj(1)) - ((2*gama)/h);
jj = [2:n-1]'; bj = (h^2)*f(xj(jj));
bn = (h^2)*f(xj(n)) - beta;
b = [b1;bj;bn];

%tridiagonal matrix A
aa = -2; bb = 1; cc = 1;
A = diag(aa*ones(1,n)) + diag(bb*ones(1,n-1),1) + diag(cc*ones(1,n-1),-1);
A = sparse(A);

%vectors u and v
u = zeros(n,1); u(1) = 1;
v = zeros(n,1);
for i = 1:n
   v(i) = 2;
end

p = algorithm(A,u,v,b);
%x1 = (A - u*v')\b
px = @(x) 1 - x.^2;

%plotting
plot(px(xj),'--*')
hold on
plot(p,'o')
legend('p(x) = 1-x^2','soln p');
xlabel('n');ylabel('x');
title('Agraph of x against n')

fprintf('According to the plot above the solution converges to p(x) = 1 - x^2 as n tends to infinity\n\n');

function p = algorithm(A,u,v,b)
    %solve Az = b
    z = A\b;
    %solve Ay = b
    y = A\u;
    %compute alpha
    alpha = v'*y;
    %compute beta
    beta = v'*z;
    %compute x
    if alpha == 0
        exit
    else
        p = z + (beta/(1-alpha)).*y;
    end
end
```

No.5(c)

According to the plot above the solution converges to p(x) = 1 - x^2 as n tends to infinity



Published with MATLAB® R2020b