Sparse Matrix Storage and Ordering

(EE521 Report-4)

Linli Jia
10/28/2019

Contents

1. Ordering test for 10x10 example

Taking the 10x10 example to test the script for ordering. The 10x10 example is shown in figure1.1.

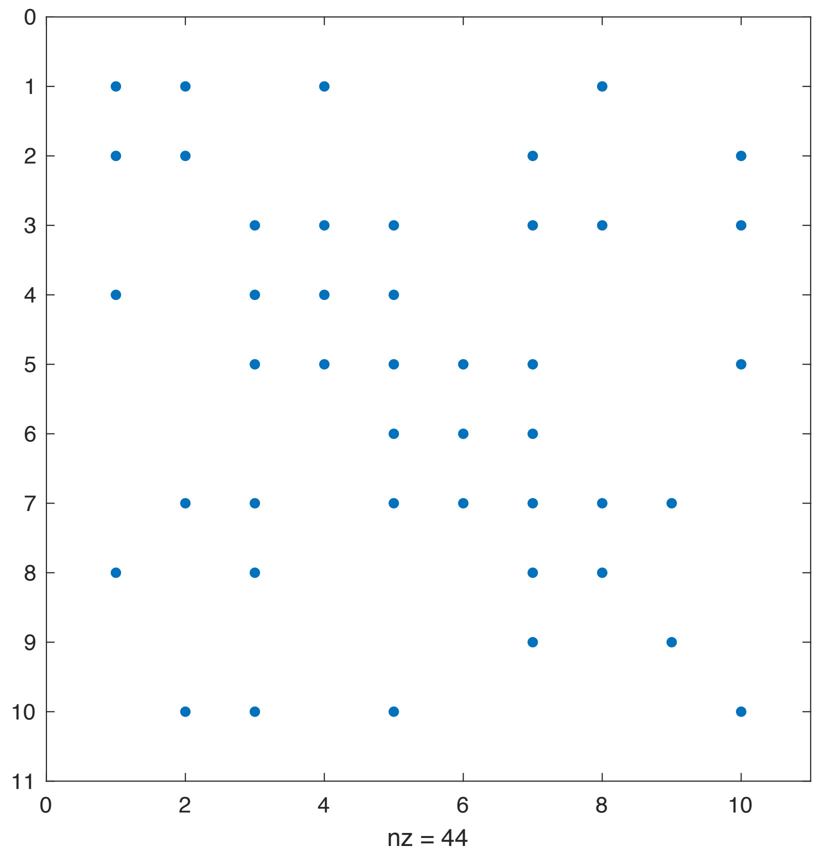

Figure 1.1 10x10 Matrix

1.1 Scheme 0

The script for scheme 0 is created as a function ’ [index_re_degrees]=Scheme0(A)’ shown in figure1.2, and the ordering result applying Scheme 0 is shown in figure 1.3, which is the same as the result given in the textbook[1].

```
function [Scheme0_order]=Scheme0(A)
% clc; clear all;
%
A=[1,1,0,1,0,0,0,1,0,0;1,1,0,0,0,0,1,0,0,1;0,0,1,1,1,0,1,1,0,1;1,0,1,1,1,0,0,
0,0,0;0,0,1,1,1,1,1,0,0,1;0,0,0,0,1,1,1,0,0,0;0,1,1,0,1,1,1,1,1,0;1,0,1,0,0,0
,1,1,0,0;0,0,0,0,0,0,1,0,1,0;0,1,1,0,1,0,0,0,0,1]
% spy(A)
degrees=sum(A~=0)-1;
k=unique(degrees);
m=length(k);
index_re_degrees=[];
for i=1:m
    index_re_degrees=[index_re_degrees,find(degrees==k(i))];
end
Scheme0_order=index_re_degrees;
end
```

Figure 1.2 Script for Scheme 0

```
ans =

     9     6     1     2     4     8    10     3     5     7
```

Figure 1.3 Ordering result applying Scheme 0

1.2 Scheme I

The script for scheme 1 is also created as a function ' [Scheme1_order]=Scheme1(A)' shown in figure1.4. In each iteration, degrees for the nodes are calculated, and the node with the least degree---'node ii' is selected to delete, which is represented by setting all the row and column elements of the node to zero. Besides, connections are built between the nodes that have a connection with 'node ii' but have no connection with each other. The ordering result applying Scheme 1 is shown in figure 1.5, which is also the same as the result given in the textbook [1].

```
function [Scheme1_order]=Scheme1(A)
% clc; clear all;
% 10 nodes example
%
A=[1,1,0,1,0,0,0,1,0,0;1,1,0,0,0,0,1,0,0,1;0,0,1,1,1,0,1,1,0,1;1,0,1,1,1,0,0,
0,0,0;0,0,1,1,1,1,1,0,0,1;0,0,0,0,1,1,1,0,0,0;0,1,1,0,1,1,1,1,1,0;1,0,1,0,0,0
,1,1,0,0;0,0,0,0,0,0,1,0,1,0;0,1,1,0,1,0,0,0,0,1]
degrees=sum(A~=0)-1;   %set the start value for degrees
Scheme1_order=[];
% A_ordered=A;
while max(degrees)>0
    index_re_degrees=[];
    k=unique(degrees);
    k_index=find(k>0);
    m=length(k_index);
    for i=1:m
        index_re_degrees=[index_re_degrees,find(degrees==k(k_index(i)))];
    end
    Scheme1_order=[Scheme1_order;index_re_degrees(1)];
    ii=index_re_degrees(1);
% find non-zero elements in the first of A_ordered which has the least degree
    A1_nnz=find(A(ii,:)~=0);
    if length(A1_nnz)>2
        for i=1:length(A1_nnz)-1
            for j=i+1:length(A1_nnz)
                if (A1_nnz(i)~=ii)&&(A1_nnz(j)~=ii)&&A(A1_nnz(i),
A1_nnz(j))==0  %build connection for other nodes
                    A(A1_nnz(i), A1_nnz(j))=1;
                    A(A1_nnz(j), A1_nnz(i))=1;
                end
            end
        end
    end
    A(ii,:)=0; %delete the node with the least degree
    A(:,ii)=0;
    degrees=sum(A~=0)-1;
end
    Scheme1_order=[Scheme1_order;index_re_degrees(end)];
end
```

Figure 1.4 Script for Scheme 1 ordering

```
ans =

     9     6     1    10     4     2     3     5     7     8
```

Figure 1.5 Ordering result applying Scheme I

1.3 Scheme II

The script for scheme II is also created as a function ' [Scheme2_order]=Scheme2(A)' shown in figure 1.6. 'Index_fills' vector stored all the alternative nodes for ordering, and it's keep updated until all of the nodes are ordered, which is represented by all of the elements in 'Index_fills' vector are deleted. In each iteration, fills for each alternative node are calculated, and the nodes with the lowest number of fills are chosen. If the node is unique, it is the node to be deleted. In case of a tie, degrees for those nodes are calculated and the one with the lowest degree and lowest natural ordering is selected to delete. Just as in Scheme1, the row and column elements of the node to be deleted are set to zero, and, connections are built between the nodes that have a connection with the node but have no connection with each other. The ordering result applying Scheme 2 is shown in figure 1.7, which is also the same as the result given in the textbook [1].

```
function [Scheme2_order]=Scheme2(A)
% clc; clear all;
% 10 nodes example
%
A=[1,1,0,1,0,0,0,1,0,0;1,1,0,0,0,0,1,0,0,1;0,0,1,1,1,0,1,1,0,1;1,0,1,1,1,0,0,
0,0,0;0,0,1,1,1,1,1,0,0,1;0,0,0,0,1,1,1,0,0,0;0,1,1,0,1,1,1,1,1,0;1,0,1,0,0,0
,1,1,0,0;0,0,0,0,0,0,1,0,1,0;0,1,1,0,1,0,0,0,0,1];
n=length(A);
Scheme2_order=[];
index_fills=(1:n);
  %% calculate fills for each node if eliminated
while ~isempty(index_fills)
    m=length(index_fills);  %each elimination alternative is considered
    fills=zeros(m,1);
    for i=1:m
        ii=index_fills(i);
        A1_nnz=find(A(ii,:)~=0);
        if length(A1_nnz)>2
            for p=1:length(A1_nnz)-1
                for q=p+1:length(A1_nnz)
                    if (A1_nnz(p)~=ii)&&(A1_nnz(q)~=ii)&&A(A1_nnz(p),
A1_nnz(q))==0  %build connection for other nodes
                    fills(i)=fills(i)+1;
                    end
                end
            end
        end
    end
    %% find the least fills
    k_fills=unique(fills);
    index_least_fills=find(fills==k_fills(1));
    if length(index_least_fills)==1
        kk=index_least_fills(1);
        node_eliminated=index_fills(kk);
    else
       degrees=[];
       for i=1:length(index_least_fills)
           ii=index_least_fills(i);
           kk=index_fills(ii); % find node number
           degrees=[degrees,sum(A(:,kk)~=0)-1];
       end
       k=unique(degrees);
       index_least_degrees=find(degrees==k(1));
       ii=index_least_degrees(1);
       jj=index_least_fills(ii);
       node_eliminated=index_fills(jj);
    end
    Scheme2_order=[Scheme2_order;node_eliminated];

    %% edges created
    A2_nnz=find(A(node_eliminated,:)~=0);
    if length(A2_nnz)>2
        for i=1:length(A2_nnz)-1
            ii=node_eliminated;
            for j=i+1:length(A2_nnz)
```

Figure 1.6   Script for Scheme II ordering

```
ans =

     9     6     4     8     2     1     3     5     7    10

fx >>
```

Figure 1.7 Ordering result applying Scheme II

2. Sparse storage

According to the ordering of nodes got from above calculation, the script 'Sparse_storage.m' stores all of the non-zero elements of a matrix, and obtain the corresponding vectors' FIC, FIR, NCOL, NROW, NIC, NIR', also vector b is reordered according to the new ordering. The function is
'[A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,b_ordered] =Sparse_storage(index_re_degrees,A,b)'.

The script 'Sparse_storage.m' is appended at the end of this report. Following matrix 'A' and vector 'b' are used for testing the script.

A=[11,12,13,14,15;21,22,0,0,0;31,0,33,0,0;41,0,0,44,0;51,0,0,0,55]

b= [1;2;0;4;5]

3. LU factorization and backward-forward substitution

3.1 Scripts for LU factorization

The function for LU factorization is

'[x]=LU_sparse(A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,b)'

and the script is appended at the end.

Q-matrix which is also stored using sparse technique is firstly obtained through the function ‘[QVALUE,QNROW, QNCOL, QNIR, QNIC, QFIR, QFIC,alpha,beta] = Crout_Sparse(A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR)’, and the script is ’Crout_Sparse.m’. In this script, to call an element from the ‘A_ordered’ vector, the function ’sparse_element’ is used, as shown below in figure 3.1.

```
function[aij]=sparse_element(value,FIC,FIR,NCOL,NROW,NIC,NIR,i,j)
Index_value=FIR(i);
if Index_value==0  % FIR(i)=0,the i th row elements are all zeros and return
aij=0;
    aij = 0;
    return
else
    while Index_value~=0 %search all non-zero elements in ith row
        if NCOL(Index_value)==j
            aij=value(Index_value);
            return
        else
            Index_value=NIR(Index_value);
        end
        aij=0; % if not find a NCOL==j, then return 0
    end
end
```

Figure 3.1 Script for getting an element from a sparse matrix

To add an element into the ‘Q sparse matrix’, the function ’ sparse_add_element’ is used, and the script is appended at the end. In this script, the vectors’ NROW, NCOL, VALUE, NIR, NIC, FIR, FIC’ are updated. The updating of FIR and NIR is considered in the following logic:

(1) if FIR=0, then the added Q value would be the FIR in i^{th} row, and its next element would be 0;

(2) if FIR $\neq$0, the added Q value could locate at three kind of positions in a row, as shown below. Q is represented by the shape of triangle and ’x’ represents the first and the last value in a row. In the script, how to update the FIR

and NIR vectors is considered according to these conditions.

$$\Delta\ x \ldots \Delta \ldots x\Delta$$

Similarly, the conditions for FIC and NIC are considered according to whether FIC=0; if FIC$\neq$0, the new Q value may locate at the beginning, or the middle, or the end of a column.

3.2 Examples

3.2.1 Example 1

Ax=b; A=$\begin{bmatrix} 1 & 3 & 4 & 8 \\ 2 & 1 & 2 & 3 \\ 4 & 3 & 5 & 8 \\ 9 & 2 & 7 & 4 \end{bmatrix}$, b=$\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$. To get the Q sparse matrix for A matrix, and to find the solution vector x.

Applying the above scripts, following results are got:

QVALUE=[1;2;4;9;3;4;8;-5;-9;-25;1.2;2.6;-0.2;1;3;-6]

QFIC=[1;5;6;7]

QFIR=[1;2;3;4]

QNCOL=[1;1;1;1;2;3;4;2;2;2;3;4;3;3;4;4]

QNROW=[1;2;3;4;1;1;1;2;3;4;2;2;3;4;3;4]

QNIC=[2;3;4;0;8;11;12;9;10;0;13;15;14;0;16;0]

QNIR=[5;8;9;10;6;7;0;11;13;14;12;0;15;16;0;0]

x=$\begin{bmatrix} -0.5 \\ -5.5 \\ 1.5 \\ 1.5 \end{bmatrix}$

These results are exactly the same as the results given in the textbook. Thus, the program in this report for LU factorization and backward and forward substitution is valid.

3.2.2 Example 2

Determine number of fills, α, and β for the 10x10 example in different ordering, including no-ordering, scheme0-order, scheme1-order and scheme2-order separately. The script for this part is' main_alpha_beta_fills.m'

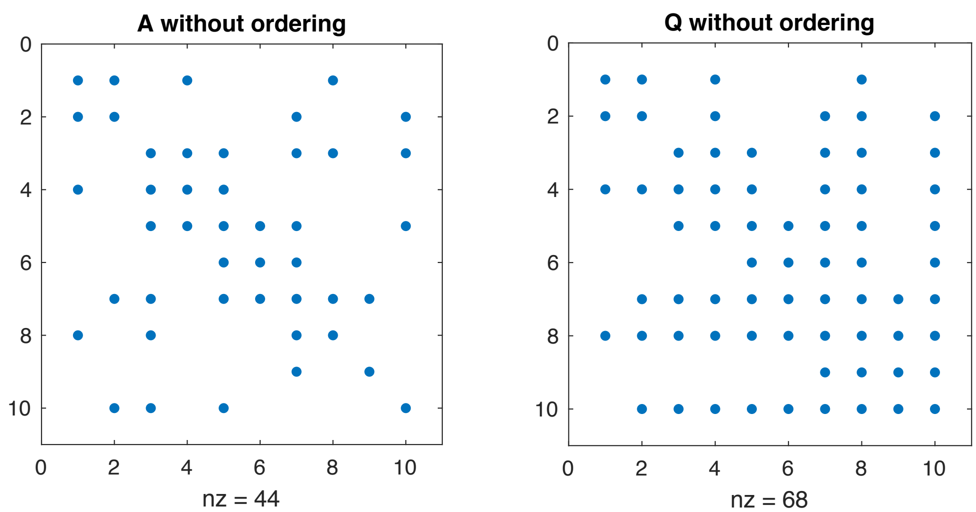

Figure 3.2 A and Q matrix without ordering

Without ordering, $\alpha = 134$, $\beta = 68$, fills=68-44=24

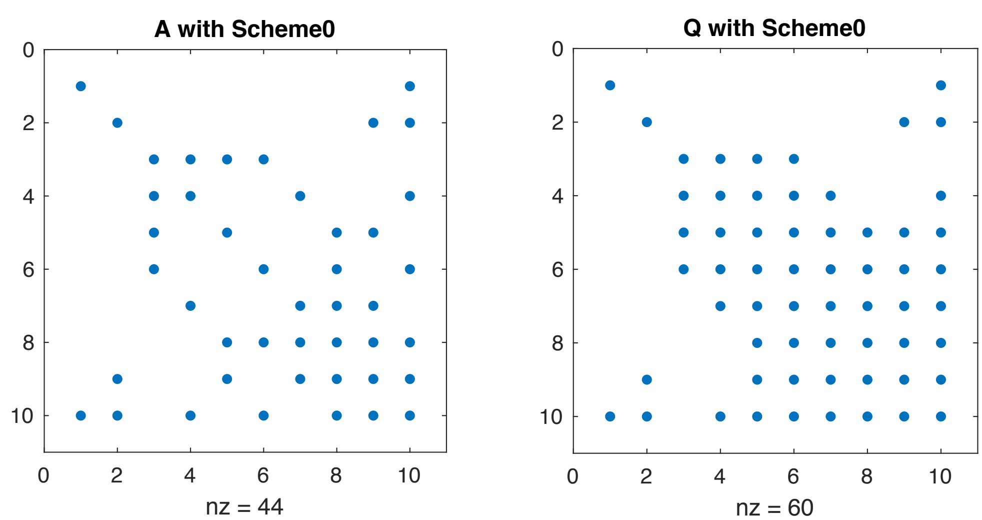

Figure 3.3 A and Q matrix with Scheme0 ordering

Applying Scheme0 ordering, $\alpha = 110$, $\beta = 60$, fills=60-44=16.

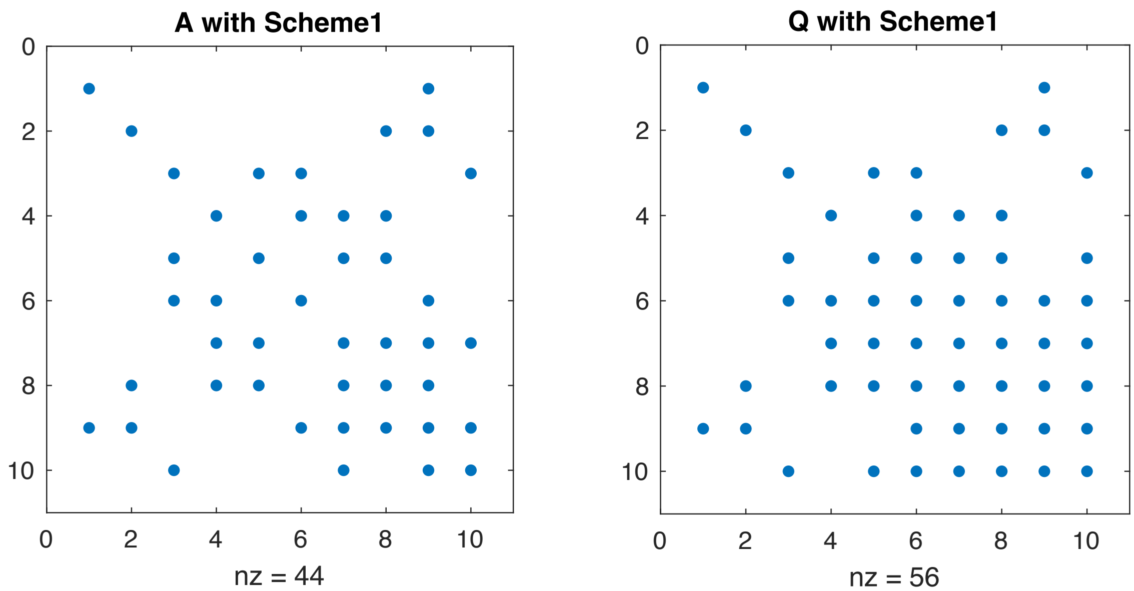

Figure 3.4 A and Q matrix with Scheme1 ordering

Applying Scheme1 ordering, $\alpha = 92$, $\beta = 56$, fills=56-44=12.

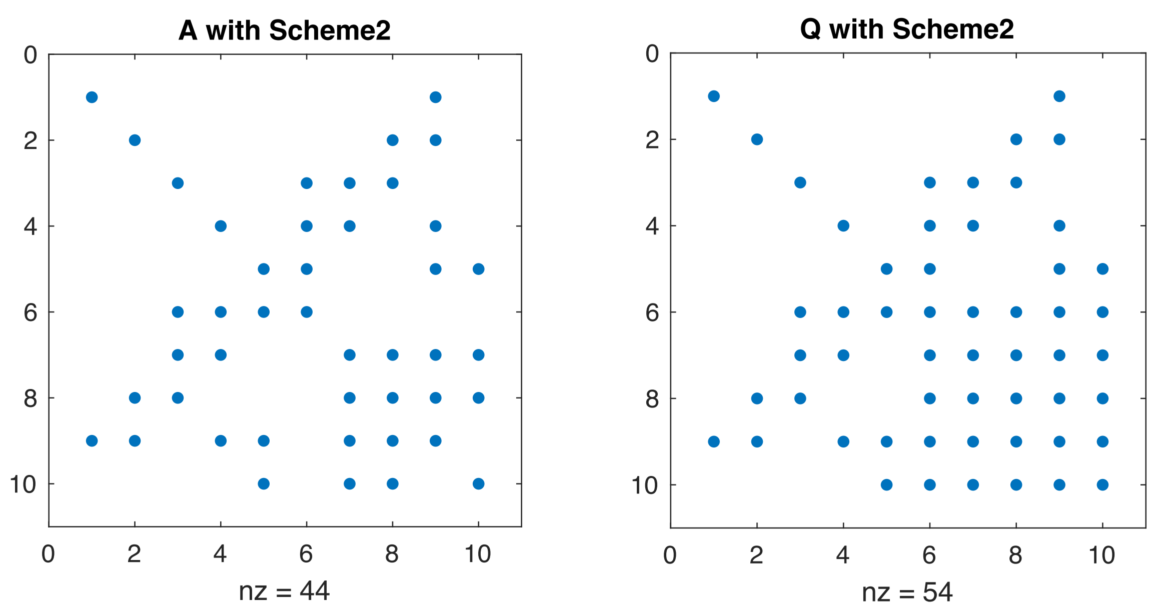

Figure 3.5 A and Q matrix with Scheme2 ordering

Applying Scheme2 ordering, $\alpha = 84$, $\beta = 54$, fills=54-44=10.

As we can see, the above reordering can effectively reduce the number of fills, α, β, and thus computational effort.

4. Applying in 14-bus-system Power Flow Calculation

4.1 Modification of Script

The part that has been revised in the script of power flow calculation is as shown below:

```
while (tol>tol_max)
    [P_cal,Q_cal]=cal_PQ(V_mag,Y_mag,Theta,V_Delta,No_of_Buses); %calculate the power at buses
    [dif_PQ]=difference_PQ(P_sch,Q_sch,P_cal,Q_cal,PQ,nPQ); % mismatches vector
    [J]=Jacobian_matrix(V_mag,P_cal,Q_cal,Y_mag,Theta, V_Delta,No_of_Buses,PQ,nPQ,B,G); %call the Jacobian matrix
    %dif_Voltage=inv(J)*dif_PQ; %get correction vector
    %% Tinney 0/1/2, so as to get the ordering of nodes
    [Scheme0_order]=Scheme0(J);
%     [Scheme1_order]=Scheme1(J);
%     [Scheme2_order]=Scheme2(J);

%% Sparse storage of Jacobian matrix
    [J_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,dif_PQ_ordered]=Sparse_storage(Scheme0_order,J,dif_PQ);
%     [J_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,dif_PQ_ordered]=Sparse_storage(Scheme1_order,J,dif_PQ);
%     [J_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,dif_PQ_ordered]=Sparse_storage(Scheme2_order,J,dif_PQ);
%% LU factorization

    [dif_Voltage_ordered]=LU_sparse(J_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,dif_PQ_ordered);
% [dif_Voltage]=LU_factor_PQ(J,dif_PQ);
%% get voltage vector in the original sequence

    dif_Voltage(Scheme0_order)=dif_Voltage_ordered;
%    dif_Voltage(Scheme1_order)=dif_Voltage_ordered;
%    dif_Voltage(Scheme2_order)=dif_Voltage_ordered;
```

Figure 4.1 The revising part in power flow calculation program

The power flow converges after three iterations, and the voltage results are shown in table 1.

```
Command Window
Congratulation,converge!, times of iteration=3.
>> V_result
```

Figure 4.2 Power flow calculation result

Table 1 Voltage comparison between final voltage and calculated voltage

V magnitude (p.u.)		V angle (degree)	
Final	Calculated	Final	Calculated
1.06	1.06	0	0
1.045	1.0450	-4.98	-4.9826
1.01	1.0100	-12.72	-12.7251
1.019	1.0177	-10.33	-10.3129
1.02	1.0195	-8.78	-8.7739

1.07	1.0700	-14.22	-14.2209
1.062	1.0615	-13.37	-13.3596
1.09	1.0900	-13.36	-13.3596
1.056	1.0559	-14.94	-14.9385
1.051	1.0510	-15.1	-15.0973
1.057	1.0569	-14.79	-14.7906
1.055	1.0552	-15.07	-15.0756
1.05	1.0504	-15.16	-15.1563
1.036	1.0355	-16.04	-16.0336

4.2 Alpha, Beta and Fills

Here, the Jacobian matrix got after power flow convergence and its corresponding Q matrix are used to calculate the number of α, β and fills. The results before ordering and after Scheme0 ordering, Scheme1 ordering, Scheme2 ordering are shown below:

Without ordering, $\alpha = 1726$, $\beta = 348$, fills=348-146=202, as shown in figure4.3.

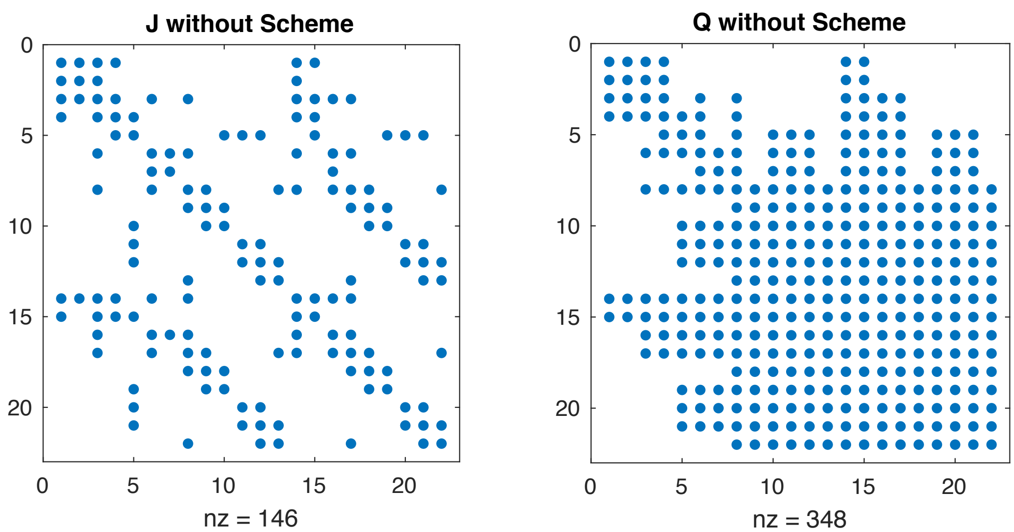

Figure 4.3 J and Q matrix without ordering

With Scheme 0, $\alpha = 338$, $\beta = 166$, fills=166-146=20, as shown in figure4.4.

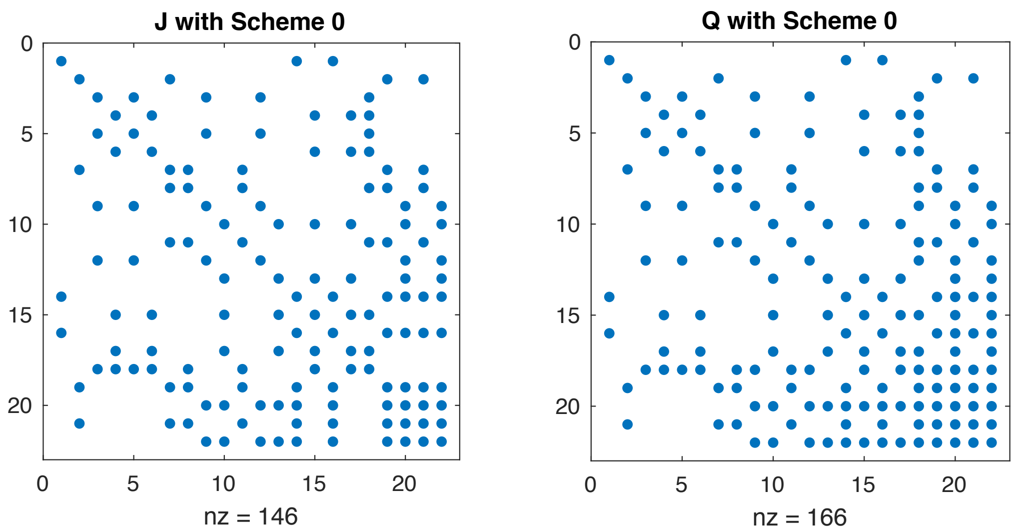

Figure 4.4 J and Q matrix with Scheme 0

With Scheme 1, $\alpha = 320$, $\beta = 162$, fills=162-146=16, as shown in figure4.5.

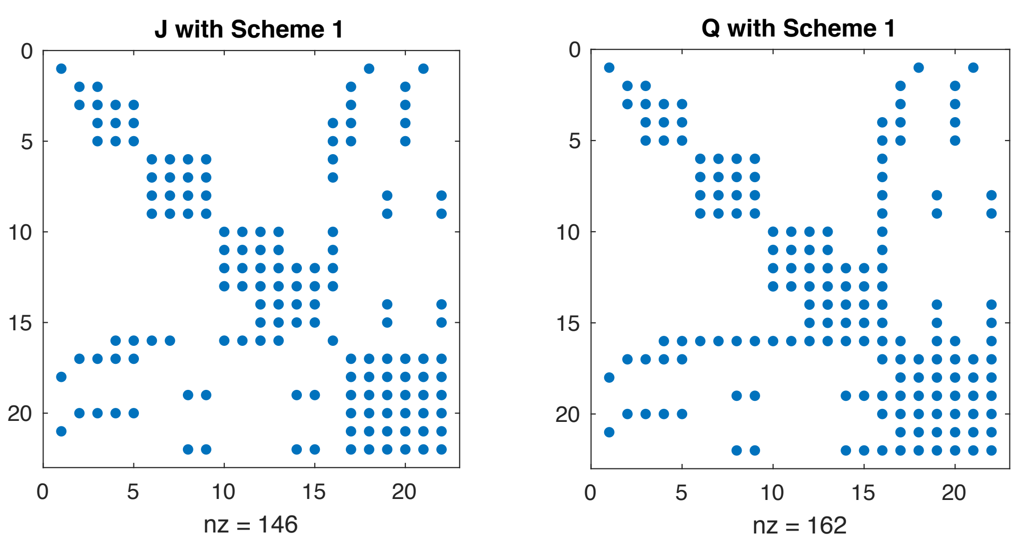

Figure 4.5 J and Q matrix with Scheme 1

With Scheme 2, $\alpha = 320$, $\beta = 162$, fills=162-146=16, as shown in figure4.6.

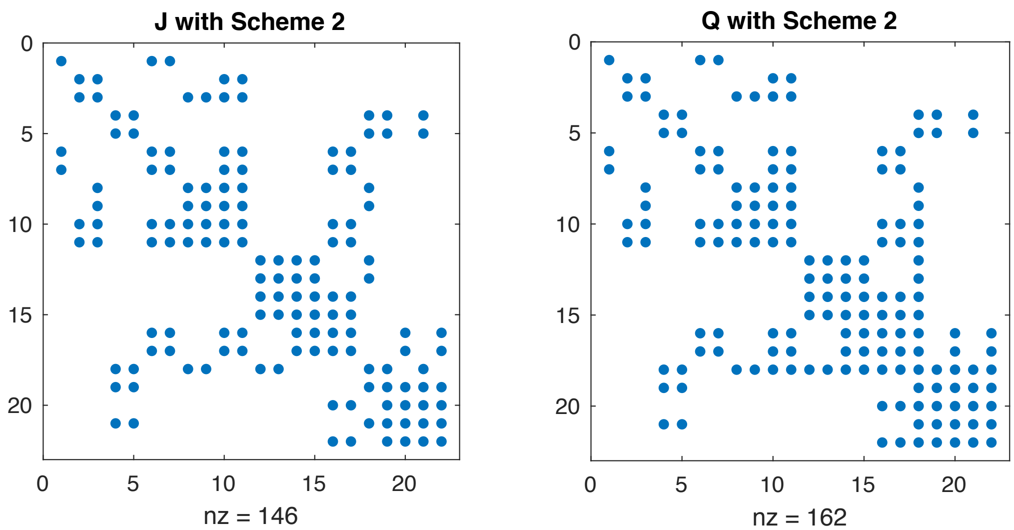

Figure 4.6 J and Q matrix with Scheme 2

As we can see, ordering of Jacobian matrix reduces the number of α, β,and fills effectively, which means a save of calculation effort. But compared with Scheme 1, Scheme 2 does not make much improvement.

5. Reference

[1] Mariesa L. Crow. Computational Methods for Electric Power Systems,3rd edition. CRC Press,2015.

6. APPENDIX

6.1 Sparse_storage.m

```
function
[A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,b_ordered]=Sparse_storage(index_re_degre
es, A,b)
% clc; clear all;
% A=[11 12 13 14 15;21 22 0 0 0;31 0 33 0 0;41 0 0 44 0;51 0 0 0 55];
% b=[1;2;0;4;5];
% A=[1 3 4 8;2 1 2 3;4 3 5 8;9 2 7 4];
% b=[1;1;1;1];
% [index_re_degrees]=Scheme0(A);

%% storage the matrix using sparse technique
A_ordered=[]; % the non zero elements in A matrix
% b_ordered=[];
n=length(A);
NROW=[];
NCOL=[];
for i=1:n
    ii=index_re_degrees(i);
    for j=1:n
        jj=index_re_degrees(j);
        if A(ii,jj)~=0
            A_ordered=[A_ordered; A(ii,jj)];
            NROW=[NROW;i];
            NCOL=[NCOL;j];
        end
    end
end
nnz=length(A_ordered);
NIR=zeros(nnz,1);
NIC=zeros(nnz,1);
for i=1:nnz-1
    if NROW(i+1)==NROW(i)
       NIR(i)=i+1;
    end
end

for i=1:nnz-1
    for j=i+1:nnz
        if NCOL(j)==NCOL(i)
            NIC(i)=j;
            break
        end
    end
end

FIR=zeros(n,1);
FIC=zeros(n,1);
for i=1:n
    for j=1:nnz
        if NROW(j)==i
        FIR(i)=j;
        break
        end
    end
```

```
end
for i=1:n
    for j=1:nnz
        if NCOL(j)==i
        FIC(i)=j;
        break
        end
    end
end
%% order b vector
b_ordered=[];
for i=1:n
    ii=index_re_degrees(i);
    b_ordered=[b_ordered; b(ii)];
end
end
```

6.2 LU_Sparse.m

```
function [x]=LU_sparse(A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,b)

 [QVALUE,QNROW, QNCOL, QNIR, QNIC, QFIR, QFIC,alpha,beta] =
Crout_Sparse(A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR);
n=length(QFIR);
%% get vector y by forward substitution
y=zeros(n,1);
for k=1:n
    y_1=0;
    for j=1:k-1
        [Q_kj]=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,k,j);
        y_1= y_1+Q_kj*y(j);
    end
    [Q_kk]=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,k,k);
    y(k)=(b(k)-y_1)/Q_kk;
end
%% get the solution vector x by backward substitution
x=zeros(n,1);
for k=n:-1:1
    x_1=0;
    for j=k+1:n
         [Q_kj]=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,k,j);
         x_1=x_1+Q_kj*x(j);
    end
    x(k)=y(k)-x_1;
end
end
```

6.3 Crout_Sparse.m

```
function [QVALUE,QNROW, QNCOL, QNIR, QNIC, QFIR, QFIC,alpha,beta] =
Crout_Sparse(A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR)

n=length(FIC); % get the dimention of A matrix
QNROW=[];
QNCOL=[];
QVALUE=[];
QNIR=[];
QNIC=[];
```

```
QFIR=zeros(1,n);
QFIC=zeros(1,n);
%% to calculate alpha
alpha_col=zeros(1,n);
alpha_row=zeros(1,n);
alpha=0;

% elements of jth column in Q matrix
for j=1:n
    for k=j:n

[A_ordered_kj]=sparse_element(A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,k,j); %
call an element in A_ordered
        Qtemp_kj=0;
        for i=1:j-1
            Q_ki=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,k,i);
            Q_ij=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,i,j);
            Qtemp_kj=Qtemp_kj+Q_ki*Q_ij;
        end
        Q_kj=A_ordered_kj-Qtemp_kj;
        if Q_kj~=0
            [QNROW, QNCOL, QVALUE, QNIR, QNIC, QFIR, QFIC] =
sparse_add_element(Q_kj, k, j, QNROW, QNCOL, QVALUE, QNIR, QNIC, QFIR, QFIC);
            alpha_col(j)=alpha_col(j)+1;
        end
    end
    % elements of jth row in Q matrix
    Q_jj=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,j,j);
    if Q_jj~=0
        for k=j+1:n
            Qtemp_jk=0;

[A_ordered_jk]=sparse_element(A_ordered,FIC,FIR,NCOL,NROW,NIC,NIR,j,k);
            for i=1:j-1

Q_ji=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,j,i);

Q_ik=sparse_element(QVALUE,QFIC,QFIR,QNCOL,QNROW,QNIC,QNIR,i,k);
                Qtemp_jk=Qtemp_jk+Q_ji*Q_ik;
            end
            Q_jk=(A_ordered_jk-Qtemp_jk)/Q_jj;
            if Q_jk~=0
            [QNROW, QNCOL, QVALUE, QNIR, QNIC, QFIR, QFIC] =
sparse_add_element(Q_jk, j,k, QNROW, QNCOL, QVALUE, QNIR, QNIC, QFIR, QFIC);
            alpha_row(j)=alpha_row(j)+1;
            end
        end
    end
end
%% calculate alpha and beta
for j=1:n
    alpha=alpha+alpha_col(j)*alpha_row(j);
end
beta=length(QVALUE);
end
```

6.4 Sparse_add_element.m

```
function [NROW, NCOL, VALUE, NIR, NIC, FIR, FIC] =
sparse_add_element(aij,i,j, NROW, NCOL, VALUE, NIR, NIC, FIR, FIC)
% %% for test this script
% clc; clear all;
% A=[1 3 4 8;2 1 2 3;4 3 5 8;9 2 7 4];
% % b=[1;1;1;1];
% [VALUE,FIC,FIR,NCOL,NROW,NIC,NIR]=Scheme0_storage(A)

%% add elements to VALUE, NROW, NCOL
n_VALUE=length(VALUE);
VALUE=[VALUE;aij];
NROW=[NROW;i];
NCOL=[NCOL;j];
num_aij=n_VALUE+1; % the number of aij in VALUE vector
index_R=FIR(i);
index_C=FIC(j);

%% update FIR,NIR
if FIR(i)==0  %originally, all of the elements in ith row are zero, then aij
would be the first element in ith row
    FIR(i)=num_aij;
    NIR(num_aij)=0;
else %FIR(i)~=0
    if j<NCOL(index_R) % aij locates at the left side of FIR(i)
        FIR(i)=num_aij;
        NIR(num_aij)=index_R;
    else %j>NCOL(index_R)
    while index_R~=0  % loop until 'index=the last element of ith row'
        if NIR(index_R)==0 %index_R points to the last element of ith row
            NIR(index_R)=num_aij;
            NIR(num_aij)=0;
            break
        elseif (NCOL(index_R)<j)&&(j<NCOL(NIR(index_R))) %aij locates between
two non-zero elements
            NIR(num_aij)=NIR(index_R);
            NIR(index_R)=num_aij;
            break
        end
            index_R=NIR(index_R);
    end
    end
end
%% update FIC NIC
if FIC(j)==0  %originally, all of the elements in jth row are zero, then aij
would be the first element in jth column
    FIC(j)=num_aij;
    NIC(num_aij)=0;
else %FIC(j)~=0
    if i<NROW(index_C) % aij locates above of FIC(i)
        FIC(j)=num_aij;
        NIC(num_aij)=index_C;
    else %i>NROW(index_C)
    while index_C~=0  % loop until 'index=the last element of jth column'
        if NIC(index_C)==0 %index_C points to the last element of jth column
            NIC(index_C)=num_aij;
            NIC(num_aij)=0;
            break
```

```
        elseif (NROW(index_C)<i)&&(i<NROW(NIC(index_C))) %aij locates between
two non-zero elements
            NIC(num_aij)=NIC(index_C);
            NIC(index_R)=num_aij;
            break
        end
            index_C=NIC(index_C);
    end
    end
end
end
```