An efficient algorithm to find certain elements of inverse of a sparse matrix

Farzad Mahfouzi

In this note our aim is to find certain elements of the inverse of a symmetric sparse matrix. When the size of the matrix is too large one might not be interested to know all elements of the inverse of the matrix. In fact it's not efficient at all to find all elements of inverse of the matrix and then filter out the required elements. Instead we can see in the following that for the case of sparse matrices it's possible to find efficiently only required blocks of the inverse of matrix. The idea is to reorder the indices of the matrix such that at the end we'll have a block tridiagonal matrix(in general with different dimensions). Then we use a recursive method and try to find the required blocks of the inverse which is in fact first block of the matrix. Therefore in this method one should first specify the elements that are required. One can do this by introducing a vector array $x=[x_1,x_2,...,x_n]$, where x_i shows the index of the required elements. Therefore if the size of whole **symmetric** matrix **A** is N, x_i would lie between 1 and N. As the first step we construct an n by n matrix using these elements and put it as first block of our block diagonal matrix **B**. Then we try to construct the whole block diagonal matrix **B**, using this base. In order to do this we try to find the next block by finding the indices that are connected directly to first block. For example if $x=x_1=1$ then , all js such that **A**(1,j) are nonzero will give me the indices of my next block (i.e. $x'=[j_1,j_2,...]$). In order to do this in Matlab we wrote following function

```
function x=neighbours_sites(ix,iy,m)
yp=find(ix==m);
for j=1:size(yp)
    x(j)=iy(yp(j));
end
```

where

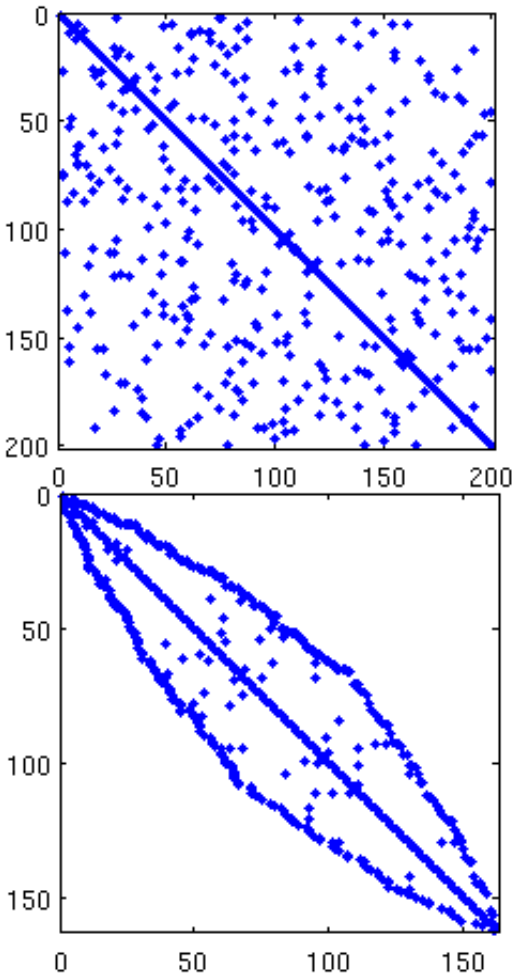
```
[ix,iy,z]=find(A);
```

m in this function is index of one site and x would be in general an array showing all indices coupled to index m. Now we can use this function and find indices of the next block using following Matlab function;

```
function [x1,fs1,nfs1]=next_block(ix,iy,x,fs,nfs)
Nb=size(x,2);
fs1=fs;
nfs1=nfs;
Counter=0;
for j=1:Nb
    nx=neighbours_sites(ix,iy,x(j));
    for k=1:size(nx,2)
        new=1;
        for l=1:nfs1
            if(nx(k)==fs1(l))
                new=0;
            end
        end
        if(new==1)
            Counter=Counter+1;
            nfs1=nfs1+1;
            fs1(nfs1)=nx(k);
            x1(Counter)=nx(k);
        end
    end
end
if(Counter==0)
    x1=0;
end
```

'fs' in this function represents found sites (indices) and 'nfs' is the number of found sites. We have introduced such variables to put only new indices for the next block. As a result x1 in this function gives indices of the next block. Now using this function we can also put as input the indices of second block and get the indices of third block. Continuing this procedure we can construct the whole matrix as blocked matrix. In fact in this method we find a new ordering for our original matrix **A** which is now in a form of block diagonal matrix. As the following code we present the function that performs this procedure.

```
function [T1,H,T2,s]=sparse_to_block(A,x)
[ix,iy,z]=find(A);
Nt=size(A,1);
fs=zeros(Nt);
Nb=size(x,2);
nfs=Nb;
fs(1:nfs)=x(:);
l_n=1;
s(l_n)=Nb;
while nfs<Nt
    nfs1=nfs;
    [x1,fs,nfs]=next_cell(ix,iy,x,fs,nfs);
    if(nfs1==nfs)
        break;
    end
    Nb1=size(x1,2);
    for j=1:Nb
        for k=1:Nb
            H(j,k,l_n)=A(x(j),x(k));
        end
        for k=1:Nb1
            T1(j,k,l_n)=A(x(j),x1(k));
            T2(k,j,l_n)=A(x1(k),x(j));
        end
    end
    l_n=l_n+1;
    x=x1;
    Nb=Nb1;
    s(l_n)=Nb;
end
for j=1:Nb
    for k=1:Nb
        H(j,k,l_n)=A(x(j),x(k));
    end
end
```



The outputs of this function are three arrays of matrices which form upper diagonal blocks (**T**1) and diagonal blocks (**H**) and lower diagonal blocks (**T**2) and a one dimensional array s which represents the dimensions of the block matrices. As an example we consider a 200 by 200 random sparse matrix shown in upper picture and put x=[1,2,3]. Then as output of the above function we see a matrix with the structure shown in lower picture. Now we use this three blocked diagonal matrix and using following function try to find the elements of inverse of first block. We can see the result of direct inversion and our method in the following picture taken from Matlab environment which are completely in agreement.

```
function invA=invb(T1,H,T2,s)
Nz=size(H,3);
N=size(H,1);
if(s==0)
    s(1:Nz)=N;
end
i=1:s(Nz);
B(i,i)=inv(H(i,i,Nz));
for k=1:Nz-2
    i=1:s(Nz-k);
    j=1:s(Nz-k+1);
    B(i,i)=inv(H(i,i,Nz-k)-T1(i,j,Nz-k)*B(j,j)*T2(j,i,Nz-k));
end
i=1:s(1);
j=1:s(2);
invA(i,i)=inv(H(i,i,1)-T1(i,j,1)*B(j,j)*T2(j,i,1));
```

```
>> B=inv(Hamil);
>> B=B(1:3,1:3)

B =

    0.0637         0         0
         0   -0.0107    0.0347
         0    0.0347   -0.1033

>> B1=invb(T1,H,T2,s)

B1 =

    0.0637         0         0
         0   -0.0107    0.0347
         0    0.0347   -0.1033
```