# 数学软件——短学期课程
# Matlab 第一次作业


姓名：汪利军

学号：3140105707

班级：统计 1401


2016.07.08

# 目 录

# 1 按行三数组与全存储相互转换

## 1.1 全存储转换为三数组

### 1.1.1 算法分析

通过两个 for 循环将全矩阵转换为按行三数组存储模式，时间复杂度为 $O(n^2)$，运行结果见图 (1), 转换结果的正确性可以通过后续运算过程体现出来

```
>> tic;[r1,c1,e1] = myfull2sparse(Afull);toc;
Elapsed time is 5.029100 seconds.
```

Figure 1: 全存储转换为三数组

### 1.1.2 代码

```matlab
function [rowIdx, colIdx, entries]=myfull2sparse(A)
%% transforming full-matrix to sparse-matrix-by-row

[nrow, ncol] = size(A);
% Initialize
k = 1;
n = 1;
rowIdx = zeros;
colIdx = zeros;
entries = zeros;

for i = 1:nrow
    entries(k) = A(i,i);
    colIdx(n) = i;
    n = n + 1;
    k = k + 1;
```

```
    for j = 1:nrow
        if j == i
            continue;
        end
        if A(i,j) ~= 0
            entries(k) = A(i,j);
            colIdx(n) = j;
            n = n + 1;
            k = k + 1;
        end
    end
    rowIdx(i+1) = n;
end
rowIdx(1) = 1;
rowIdx(nrow+1) = ncol;
end
```

## 1.2 三数组转换为全存储

### 1.2.1 算法分析

通过两个 for 循环，其中一个 for 循环对行遍历，然后第二个 for 循环对该行非零元素 (含对角元) 进行遍历。假设每行非零元分布大体均匀，于是时间复杂度为 $O(n \times \frac{N}{n}) = O(N)$, 考虑极端情况，时间复杂度为 $O(n \times N)$。

运行结果如图 (2) 所示，转换结果的正确性可以通过后续运算过程体现出来。

```
>> tic;Afull = mysparse2full(r,c,e);toc;
Elapsed time is 0.011098 seconds.
```

Figure 2: 三数组存储转换为全存储

### 1.2.2 代码

```
function A = mysparse2full(rowIdx, colIdx, entries)
%% transform three-arrays sparse matrix to full matrix

nrow = size(rowIdx, 2) - 1;
ncol = rowIdx(nrow+1);
N = size(colIdx, 2);
A = zeros(nrow, ncol);
for i = 1:nrow
    j = rowIdx(i);
    if i==nrow
        for k = j:N
            A(i, colIdx(k)) = entries(k);
        end
        break;
    end
    for k = 1:(rowIdx(i+1) - rowIdx(i))
        A(i,colIdx(j)) = entries(j);
        j = j + 1;
    end
end
end
```

# 2 按行三数组与 Matlab 稀疏存储

## 2.1 Matlab 稀疏存储转换为按行三数组存储

### 2.1.1 算法分析

首先通过对非零行元素从小到大排序，然后对非零元按行进行遍历进而 online 转换为三数组存储, 对某行全空对对角元进行操作，因为是 online 的，所以可以判断时间复杂度为 $O(n)$

运行结果如图 (3) 所示，转换结果的正确性可以通过后续运算过程体现出来。

```
>> A = sprandsym(10000,0.00001);
>> tic;[r,c,e] = mymatsp2sp(A);toc;
Elapsed time is 0.003664 seconds.
```

Figure 3: Matlab 稀疏存储转换为行三数组存储

### 2.1.2 代码

```
function [rowIdx,colIdx,entries]=mymatsp2sp(A)
%% transform matlab sparse to three arrays
% A: full square matrix
% rowIdx : an array
% colIdx : an array
% entries : an array

[I,J,S] = find(A);
N = size(I, 1); % the number of non-zero elements
[nrow ncol] = size(A);

% sort by the first column
tmp = sortrows([I J S],1);
```

```
I = tmp(:,1);
J = tmp(:,2);
S = tmp(:,3);

% Initialize
rowIdx = zeros;
colIdx = zeros;
entries = zeros;
prow = 0;   % a pointer to the row
pcol = 1;   % a pointer to the column
diag = 0;   % record the diag element
k = 1;

while (k <= N)
    if prow == I(k)
        if I(k) == J(k)     % the diagonal element
            entries(diag) = S(k);
        else
            colIdx(pcol) = J(k);
            entries(pcol) = S(k);
            pcol = pcol + 1;
        end
        k = k + 1;
    else
        while (prow < I(k))
            prow = prow + 1;
            colIdx(pcol) = prow;
            entries(pcol) = 0;
            diag = pcol;
            rowIdx(prow) = pcol;
            pcol = pcol + 1;
```

```
            end
        end
    end
    while prow < nrow   % fill the rows which all elements are zero
        prow = prow + 1;
        colIdx(pcol) = prow;
        entries(pcol) = 0;
        rowIdx(prow) = pcol;
        pcol = pcol + 1;
    end
    rowIdx(nrow+1) = ncol;
    end
```

## 2.2 按行三数组存储转换为 Matlab 稀疏存储

### 2.2.1 算法分析

通过两个 for 循环，其中一个 for 循环对行遍历，另一个对每行的非零元 (含对角元) 历。设每行非零元分布大体均匀，于是时间复杂度为 $O(n \times \frac{N}{n}) = O(N)$, 考虑极端情况，时间复杂度为 $O(n \times N)$。

运行结果如图 (4) 所示，转换结果的正确性可以通过后续运算过程体现出来。

```
>> tic;Amatsp = mysp2matsp(r,c,e);toc;
Elapsed time is 0.740180 seconds.
```

Figure 4: 行三数组转换为 Matlab 稀疏存储

### 2.2.2 代码

```
function A=mysp2matsp(rowIdx,colIdx,entries)
%% transform three-arrays sparse matrix to matlab sparse matrix

```

```
nrow = size(rowIdx,2) - 1;
N = size(entries,2);
k = 1;
I = zeros;
J = zeros;
S = zeros;
for i = 1:nrow
    j = rowIdx(i);
    if i == nrow
        for p = j:N
            I(k) = i;
            J(k) = colIdx(p);
            S(k) = entries(p);
            k = k + 1;
        end
        break;
    end
    for p = 1:(rowIdx(i+1)-rowIdx(i))
        I(k) = i;
        J(k) = colIdx(j);
        S(k) = entries(j);
        j = j + 1;
        k = k + 1;
    end
end
for i = N:-1:1
    if S(i)==0
        I(i) = [];
        J(i) = [];
        S(i) = [];
    end
```

```
end
A=sparse(I,J,S);
end
```

# 3 在指定位置添加非零元素

## 3.1 算法分析

避免了使用 for 循环，采用矩阵运算 (代码 24,25 行), 故时间复杂度可以看作是 $O(1)$。值得说明的是，这个函数还可以向已有非零元进行加法计算，这是为了方便后续矩阵加矩阵做铺垫。

运行结果如图 (5) 所示, 通过判断 $C$ 与 $A$ 是否相等来检验正确性

```
>> A = sprandsym(10000,0.00001);
>> [r1,c1,e1] = mymatsp2sp(A);
>> tic;[r2,c2,e2] = myadd(r1,c1,e1,0.9999,2016,4032);toc;
Elapsed time is 0.001068 seconds.
>> tic;A(2016,4032)=0.9999;toc;
Elapsed time is 0.000656 seconds.
>> C = mysparse2full(r2,c2,e2);
>> C~=A

ans =

   All zero sparse: 10000-by-10000
```

Figure 5: 添加非零元

## 3.2 代码

```
1 function [ rowIdx,colIdx,entries] = myadd( rowIdx,colIdx,entries,a
    ,i,j)
```

```
%% add a not zero element at position (i, j)
% a: the element to be inserted
% i: the row index of element a
% j: the col index of element a
% rowIdx: an array
% colIdx: an array
% entries: an array

r = rowIdx(i); % first index in i row
N = size(colIdx, 2);% nonzero element numbers
nrow = size(rowIdx,2) - 1;% row numbers
if i ~= nrow    % t the number of the elements at i-th row
    t = rowIdx(i+1) - rowIdx(i);
else
    t = N - rowIdx(i) + 1;
end
colIdx = [colIdx, 1];
entries = [entries, 1];
if i == nrow
    colIdx(N+1) = j;
    entries(N+1) = a;
else
    colIdx((r+t+1):(N+1)) = colIdx((r+t):N);
    entries((r+t+1):(N+1)) = entries((r+t):N);
    colIdx(r+t) = j;
    entries(r+t) = a;
    rowIdx(i+1:nrow) = rowIdx(i+1:nrow) + 1;
end
%% the following is add a number to a non-zero element
if i ~= nrow
    s = r + t;
```

```
else
    s = N + 1;
end
for p = r:s-1
    if colIdx(p) == colIdx(s)
        entries(p) = entries(p) + entries(s);
        colIdx(s) =[];
        entries(s) = [];
        rowIdx(i+1:nrow) = rowIdx(i+1:nrow) - 1;
        break;
    end
end
end
```

# 4 剔零压缩存储

## 4.1 算法分析

避免了 for 循环，采用矩阵向量运算，于是可以将时间复杂度看成是 $O(1)$。运行结果如下图 (6) 所示, 通过时间来看，这与上述时间复杂度的判断是一致的，而且结果是正确的。

```
>> A = sprandsym(10000,0.00001);
>> [r1,c1,e1] = mymatsp2sp(A);
>> tic;[r2,c2,e2] = myzero(r1,c1,e1,2016,4032);toc;
Elapsed time is 0.000090 seconds.
>> tic;A(2016,4032)=0;toc;
Elapsed time is 0.000307 seconds.
>> C = mysparse2full(r2,c2,e2);
>> C~=A

ans =

   All zero sparse: 10000-by-10000
```

Figure 6: 剔零压缩存储

## 4.2 代码

```
function [ rowIdx,colIdx,entries] = myzero( rowIdx,colIdx,entries, i,j)
%% change the non-zero element to zero
% i: the non-zero element row's index
% j: the non-zero element column's index

% whatever the diagonal element is we return the same matrix
if i == j
    return;
end

N = size(colIdx,2);
nrow = size(rowIdx,2) - 1;
```

```
r = rowIdx(i);
if i ~= nrow                    % t is the number of elements at i-
    th row
    t = rowIdx(i+1) - rowIdx(i);
else
    t = N - rowIdx(i) + 1;
end
k = find(colIdx(r:(r+t-1)) == j);
if isempty(k)
    return;
end
k = k + r -1;
colIdx(k) = [];
entries(k) = [];
if i ~= nrow
    rowIdx(i+1:nrow) = rowIdx(i+1:nrow) - 1;
end
end
```

# 5 矩阵加法

## 5.1 算法分析

对第二个矩阵的非零元进行遍历，利用 myadd.m 先把第二个矩阵的每一个非零元插入第一个矩阵中，这里有两种情况，一是添加的元素在矩阵一中不为零（或是对角元），这种就相当于简单的在矩阵一中添加非零元；第二种情况是矩阵二中的元素在矩阵一中对应的元素非零 (或为对角元)，亦即在 entries1 中有相应的元素，这是利用在 myadd.m 中后一段代码便可以实现加法，同时还可以实现压缩存储。

又因为 myadd.m 的时间复杂度为 $O(1)$，myzero.m 的时间复杂度为 $O(1)$, 则此时时间复杂度为 $O(N_2)$，其中，$N_2$ 为第二个矩阵的非零元个数。

运行结果如下图 (8) 所示，通过判断 $C1$ 与 $C$ 是否相等，来判断运算结果的正确性。

```
>> A = sprandsym(10000,0.00001);
>> B = sprandsym(10000,0.00001);
>> [r1,c1,e1] = mymatsp2sp(A);
>> [r2,c2,e2] = mymatsp2sp(B);
>> tic;[r3,c3,e3] = myplus(r1,c1,e1,r2,c2,e2);toc;
Elapsed time is 1.917180 seconds.
>> C1 = mysparse2full(r3,c3,e3);
>> tic;C=A+B;toc;
Elapsed time is 0.000564 seconds.
>> C1~=C

ans =

   All zero sparse: 10000-by-10000
```

Figure 7: 矩阵加法

## 5.2 代码

```
function [ rowIdx1,colIdx1,entries1] = myplus( rowIdx1,colIdx1,
    entries1,rowIdx2,colIdx2,entries2)
%% one matrix plus one matrix

nrow1 = size(rowIdx1, 2) - 1;
ncol1 = rowIdx1(nrow1+1);
nrow2 = size(rowIdx2, 2) - 1;
ncol2 = rowIdx2(nrow2+1);
if nrow1 == nrow2 && ncol1 == ncol2
N = size(colIdx2,2);
```

```
prow = 1;
for i = 1:N
    if i == rowIdx2(prow+1) && prow < nrow2
        prow = prow + 1;
    end
    [rowIdx1,colIdx1,entries1] = myadd( rowIdx1,colIdx1,entries1,
       entries2(i),prow,colIdx2(i));
end
else
    disp('matrix dimensions must agree!!!');
end
end
```

# 6 矩阵减法

## 6.1 算法分析

矩阵减法直接利用上述矩阵加法，故其时间复杂度也为 $O(N_2)$。运行结果如下图 (8) 所示

```
>> tic;[r4,c4,e4] = myminus(r1,c1,e1,r2,c2,e2);toc;
Elapsed time is 1.963895 seconds.
>> tic;D1=A+B;toc;
Elapsed time is 0.000622 seconds.
>> tic;D1=A-B;toc;
Elapsed time is 0.000906 seconds.
>> D2 = mysparse2full(r4,c4,e4);
>> D1~=D2

ans =

   All zero sparse: 10000-by-10000
```

Figure 8: 矩阵减法

### 6.2 代码

```
function [ rowIdx1,colIdx1,entries1] = myminus( rowIdx1,colIdx1,entries1,rowIdx2,colIdx2,entries2)
%% one matrix minus another matrix
entries2 = -1 * entries2;
[ rowIdx1,colIdx1,entries1] = myplus( rowIdx1,colIdx1,entries1,rowIdx2,colIdx2,entries2);
```

## 7 矩阵乘向量

### 7.1 算法分析

通过两个 for 循环，其中一个 for 循环对行遍历，另一个对每行的非零元 (含对角元) 遍历。设每行非零元分布大体均匀，于是时间复杂度为 $O(n \times \frac{N}{n}) = O(N)$, 考虑极端情况，时间复杂度

为 $O(n \times N)$

对于 $A \times b$，$A$ 为 $10^4$ 阶方阵，$b$ 为 $1 \times 10^4$ 的行向量，结果如下图所示，可见虽然运算速度远不及 matlab，但仍属于可接受的范围，而且运算结果是正确的。

```
>> [r,c,e] = mymatsp2sp(A);
>> A = sprandsym(10000,0.0001);
>> B = sprandsym(10000,0.0001);
>> b = B(:,1);
>> tic;C1 = A*b;toc;
Elapsed time is 0.000581 seconds.
>> [r,c,e] = mymatsp2sp(A);
>> tic;C2 = mymultivector(r,c,e,b);toc
Elapsed time is 0.086166 seconds.
>> C1~=C2

ans =

   All zero sparse: 10000-by-1
```

## 7.2 代码

```
function Vector= mymultivector(rowIdx,colIdx,entries,vector)
%% a matrix multiply a vector

```

```
nrow = size(rowIdx, 2) - 1;
N = size(colIdx, 2); % the number of non-zero element(include
    diagonal elements)

nvector = size(vector, 1);
if nvector == nrow
    Vector = zeros(nrow,1);
else
    disp('Inner matrix dimensions must agree.');
    return;
end
for i = 1:nrow
    r = rowIdx(i);      %  t is the number of non-zero elements at
        i-th row
    if i == nrow
        t = N - rowIdx(i) + 1;
    else
        t = rowIdx(i+1) - rowIdx(i);
    end
    for k = r:r+t-1
        Vector(i,1) = entries(k)*vector(colIdx(k),1) + Vector(i,1)
            ;
    end
end
end
```

# 8 矩阵乘矩阵

## 8.1 算法分析

考虑矩阵 $A \times B$, 对矩阵 $A$ 的行进行遍历，对矩阵的每一行中的非零元 (含对角元)，对于在矩阵 $B$ 中的对应的行，找出非零元所在的列组成 nonzerocol2, 这样矩阵乘法只需要对矩阵 $A$ 的每一行和 nonzerocol2 中的列进行运算，在非零元分布相对均匀时情况下，时间复杂度为 $O(n \times \frac{N_1}{n} \times \frac{N_2}{n}) = O(\frac{N_1 N_2}{n})$, 极端情况下，时间复杂度为 $O(n \times N_1 \times N_2)$

对于一万乘以一万，稀疏度为 $10^{-4}$ 的矩阵，运行结果如下图 (9) 所示。可见虽然效率跟 matlab 相比完全不在一个量级，但至少说得过去，毕竟 matlab 有一堆数学家在研究算法。

```
>> A = sprandsym(10000,0.0001);
B = sprandsym(10000,0.0001);
[r1,c1,e1] = mymatsp2sp(A);
[r2,c2,e2] = mymatsp2sp(B);
>> tic;
[r3,c3,e3] = mymulti2(r1,c1,e1,r2,c2,e2);
toc;
Elapsed time is 2.915097 seconds.
>> tic;
C = A*B;
toc;
Elapsed time is 0.012742 seconds.
```

Figure 9: $10^4 \times 10^4$ 稀疏度为 $10^{-4}$

下图 (10) 展现了运算结果的正确性，因为没有元素是不相等的。

```
>> C1 = mysparse2full(r3,c3,e3);
>> C~=C1

ans =

   All zero sparse: 10000-by-10000
```

Figure 10: 检验结果是否正确

```
>> A = sprandsym(100000,0.00001);
B = sprandsym(100000,0.00001);
[r1,c1,e1] = mymatsp2sp(A);
[r2,c2,e2] = mymatsp2sp(B);
>> tic;
[r3,c3,e3] = mymulti2(r1,c1,e1,r2,c2,e2);
toc;
Elapsed time is 28.769804 seconds.
>> tic;
C = A*B;
toc;
Elapsed time is 0.017300 seconds.
```

Figure 11: 挑战十万级别

挑战十万乘十万阶的矩阵发现还是可以运行的, 虽然跟 matlab 的运行速度相差更大。

## 8.2 代码

```
function [ RowIdx,ColIdx,Entries] = mymulti2( rowIdx1,colIdx1,
    entries1,rowIdx2,colIdx2,entries2)
N1 = size(colIdx1, 2);
```

```
N2 = size(colIdx2, 2);

nrow1 = size(rowIdx1,2) - 1;
nrow2 = size(rowIdx2,2) - 1;
if nrow1 ~= nrow2
    disp('Inner matrix dimensions must agree.');
    return;
end
RowIdx = zeros;
ColIdx = zeros;
Entries = zeros;
pcol = 1;
for i = 1:nrow1
    RowIdx(i) = pcol;
    r = rowIdx1(i);
    if i == nrow1
        t = N1 - rowIdx1(i) + 1;
    else
        t = rowIdx1(i+1) - rowIdx1(i);
    end
    % find the nonzero column of matrix 2
    % at the rows which corresponding to
    % the nonzero column of matrix 1
    nonzerocol2 = [];
    for k = r:r+t-1
        % k is the index of the colIdx1
        % sum{(i,j)*(j,entries1(k))}
        % t2 is the number of element at colIdx(k)-th row in
           matrix 2
        r2 = rowIdx2(colIdx1(k));% the index of column at the
           colIdx1(k)-th row in matrix 2
```

```
        if colIdx1(k) == nrow2
            t2 = N2 - rowIdx2(colIdx1(k)) + 1;
        else
            t2 = rowIdx2(colIdx1(k)+1) - rowIdx2(colIdx1(k));
        end
        for k2 = r2:(r2+t2-1)
            nonzerocol2 = union(nonzerocol2, colIdx2(k2));
        end
    end
    % ans(i, nonzerocol2(j)) = sum{(i,:)*(:,nonzerocol2(j))}
    diag = find(nonzerocol2 == i);
    nonzerocol2(diag) = [];
    nonzerocol2 = [i nonzerocol2];
    for j = nonzerocol2
        % k is the index of colIdx1
        tmpsum = 0;
        for k = r:r+t-1
            % colIdx(k) is the column of element which is nonzero
            % (i,colIdx(k))' = (colIdx(k),i)
            % tmpcol is the index of row in matrix 2
            %tmpcol = find(rowIdx2(1:nrow1) == colIdx1(k));%%pay attention!!!! the last element in rowIdx2 is the number of columns
            %if isempty(tmpcol)
            % the corresponding position's value is zero
            %   continue;
            %end
            % tmpt is the number of element at tmpcol-th row in matrix 2
            if colIdx1(k) == nrow2;
                tmpt = N2 + 1 - rowIdx2(colIdx1(k));
```

```
            else
                tmpt = rowIdx2(colIdx1(k)+1) - rowIdx2(colIdx1(k));
            end
            curcol = find(colIdx2(rowIdx2(colIdx1(k)):(rowIdx2(colIdx1(k))+tmpt-1)) == j);
            if isempty(curcol)
            % (colIdx1(k),i)'s value is entries2(curcol)
            % (i,colIdx1(k))'s value is entries1(k)
                tmpsum = tmpsum;
            else
                curcol = curcol + rowIdx2(colIdx1(k)) - 1;
                tmpsum = tmpsum + entries1(k)*entries2(curcol);
            end
        end
        ColIdx(pcol) = j;
        Entries(pcol) = tmpsum;
        pcol = pcol + 1;
    end
end
RowIdx(nrow1+1) = nrow1;
end
```