16-12-2020

Métodos Numericos para la Computación

Entrega 6

4ºCurso

Grupo Prácticas 11

Alejandro Daniel Herrera Cardenes
Carlos Eduardo Pacichana Bastidas
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Indice

Practica 1 2
Descripción 2
Trabajo realizado 2
Practica 2 5
Descripción 5
Trabajo realizado 5
Optativos 8
Practica 1 8
Descripción 8
Practica 2 11
Descripción 11

Practica 1

Descripción

1. Programar dos funciones en Matlab/Octave que devuelvan la codificación COO y CSR para una matriz que se le pase como entrada:
   [row, col, val] = COO(A);
   [rowOff, col, val] = CSR(A);
2. Programar un generador de matrices aleatorias escasas en Matlab/Octave, tomando como entrada la densidad deseada de valores nulos para la matriz y el rango de valores.
   [A] = GenerateSparse(nRow, nCol, zDensity, vMin, vMax);

Trabajo realizado

Creamos una matriz 4x4 la cual usamos como entrada en las funciones COO y CSR para su codificación. Creamos funciones independientes para su uso más claro, las cuales pueden recibir matrices de cualquier tamaño y densidades de dispersión. Y para acabar generamos una matriz escasa con valores aleatorios.

Matlab

A continuación los códigos de los distintos apartados pero esta vez en ***Matlab***

```
%Apartado 1

X = [1 0 2 0; 0 1 1 1; 0 0 4 3 ;  0 0 4 3];
[f,c,v] = COO(X);
fprintf(" FORMATO COO \n")
fprintf(' Offset row :\n %s', sprintf('%d ', f))
fprintf('\n Indice de columnas :\n %s', sprintf('%d ', c))
fprintf('\n Valores :\n %s', sprintf('%d ', v))
[fl,cl,vl] = CSR(X);
fprintf(" \n \n FORMATO CSR \n")
fprintf(' Indice de filas :\n %s', sprintf('%d ', fl))
fprintf('\n Indice de columnas :\n %s', sprintf('%d ', cl))
fprintf('\n Valores :\n %s', sprintf('%d ', vl))
```

Código principal 1

```
function [rowOff, col, val] = CSR( A )
    NNZ = 0;
    iter = 1;
    for i=1:length(A)
        for e=1:length(A)
            if A(i,e) ~= 0
                NNZ = NNZ+1 ;
                col(iter) = e;
                val(iter) = A(i,e);
                iter = iter+1;
            end
        end
        rowOff(i+1) = NNZ;
    end
end
```

```
function [row, col, val] = COO( A )
    [row, col, val] = find(A);
end
```

Código principal 1.1

```
%Apartado 2
[a] = GenerateSparse(10,10,0.5,1,10);
```

Código principal 2

```
function [A] = GenerateSparse(nRow,nCol,zDensity,vMin,vMax)
    A = zeros(nRow,nCol);
    i=floor((rand(1)*nRow)+1);
    j=floor((rand(1)*nCol)+1);

    nElement = floor(nRow * nCol * zDensity);
    for e = 1:nElement
        while A(i,j)~=0
            i=floor((rand(1)*nRow)+1);
            j=floor((rand(1)*nCol)+1);
        end
        A(i,j) =( rand(1)* vMax )+ vMin;
    end
end
```

Código principal 2

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (***Resultado de ejecución 1***). Donde podemos ver los resultados de codificar la matriz definida en la imagen ***Código principal 1*** utilizando las funciones COO y CSR ***(Código principal 1.1)***

```
FORMATO COO
Indice de filas :
1 2 1 2 3 4 2 3 4
Indice de columnas :
1 2 3 3 3 3 4 4 4
Valores :
1 1 2 1 4 4 1 3 3

FORMATO CSR
Indice de filas :
0 2 5 7 9
Indice de columnas :
1 3 2 3 4 3 4 3 4
Valores :
1 2 1 1 1 4 3 4 3 >>
```

Resultado de la ejecución 1

A continuación observamos el resultado de usar la función para generar matrices aleatorias dispersas que está en la imagen ***Código principal 2***. ***(Resultado de la ejecución 2)***

```
a =

    4.3452         0         0         0    2.2130    4.1877    3.3748         0   10.9294    2.4790
    3.5335         0         0    1.8678   10.1067    3.2133    9.1650    7.5095         0    8.1512
    1.9298         0         0         0         0    4.8195   10.6657         0    3.0528         0
    9.4269    8.9499         0   10.8792    7.6731         0         0    1.3261         0    5.5025
   10.2850         0    8.2115   10.1988         0         0         0         0         0         0
         0    2.8073         0         0         0         0    9.7861         0    9.7124         0
    7.4393    9.3471   10.6842    2.3032         0    5.5076         0    2.6504         0         0
         0         0    9.4901    5.8397         0    3.1949    6.4909    1.7910         0    8.9455
         0   10.9388    8.1212         0         0         0    3.5832   10.7046    3.6556    2.3418
         0         0         0    9.5752    6.1435   10.7621         0         0         0         0
```

Resultado de la ejecución 2

Practica 2

Descripción

Tomar un ejemplo de matriz utilizada en la práctica 1 y reproducir los resultados con las funciones MKL.

Trabajo realizado

Buscamos la equivalencia de las funciones CSR y COO en MKL en la documentación y con los parámetros permitidos realizamos el ejercicio.

Código Programa:

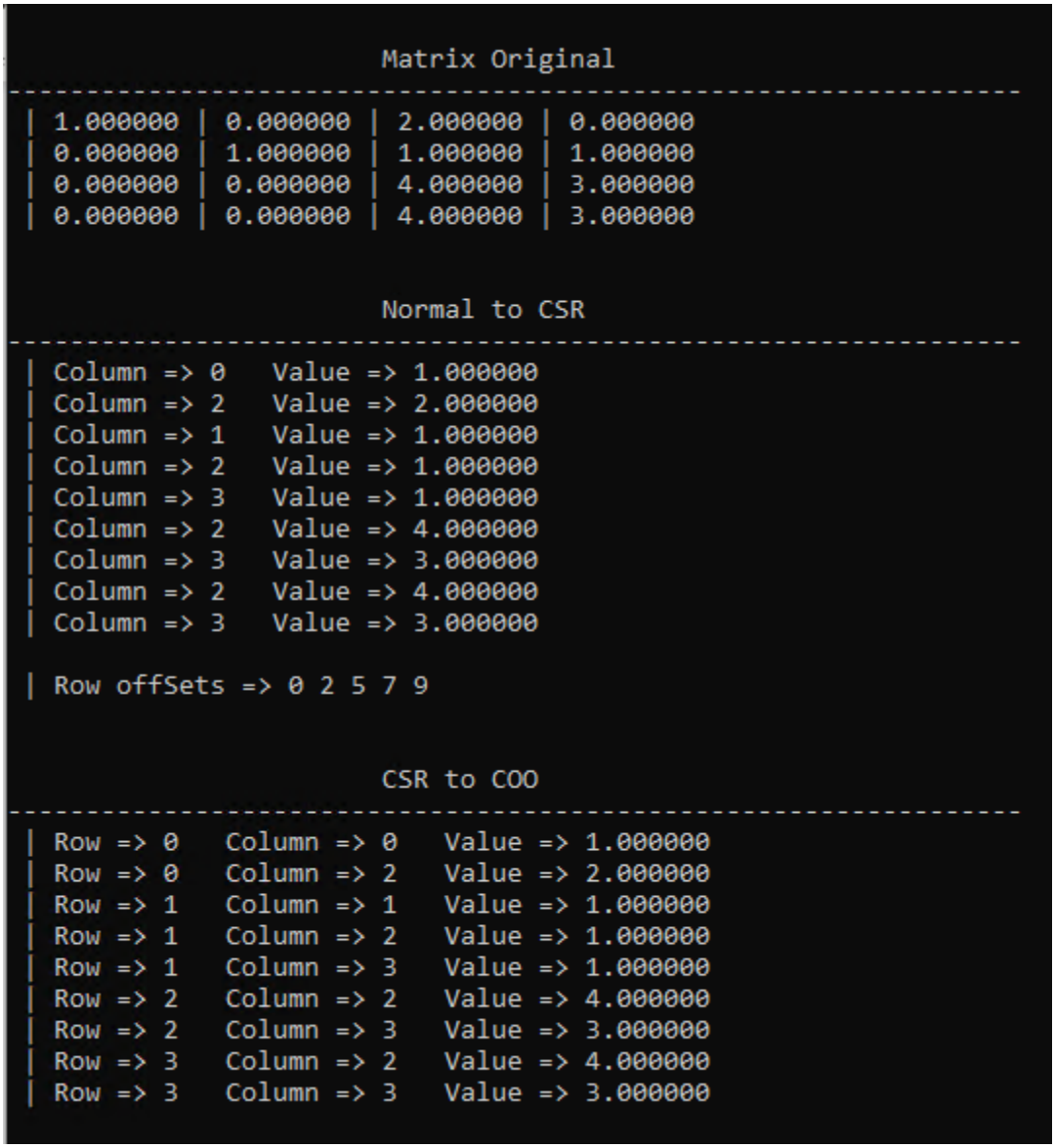
Los códigos de la librería ***MKL*** de los distintos apartados serán puesto a continuación con la referencia del apartado en el pie de la foto.

```c
int N = 4;

double X[16] = {1, 0, 2, 0,
                0, 1, 1, 1,
                0, 0, 4, 3,
                0, 0, 4, 3};

printf("\n\n\n");
printf("            Matrix Original");
printf("\n-----------------------------------------------------------------");
for (int e = 0; e < N * N; e++) {
    if (e % N == 0) {
        printf("\n | %f", X[e]);
    }
    else {
        printf(" | %f", X[e]);
    }
}

MKL_INT info;
MKL_INT m = 4;
MKL_INT n = 4;
MKL_INT NNZ = 9;
MKL_INT job[6] = {0, 0, 0, 2, NNZ, 1};

double* acsr = (double*)mkl_malloc(NNZ * sizeof(double), 64);
MKL_INT* aj = (MKL_INT*)calloc(NNZ, sizeof(MKL_INT));
MKL_INT* ai = (MKL_INT*)calloc(m+1, sizeof(MKL_INT));

#pragma warning(disable:4996)
mkl_ddnscsr(job, &m, &n, X, &m, acsr, aj, ai, &info);

printf("\n\n\n");
printf("            Normal to CSR");
printf("\n-----------------------------------------------------------------");

for (int i = 0; i < NNZ; i++) {
    printf("\n | Column => %i   Value => %f", aj[i], acsr[i]);
}

printf("\n\n | Row offSets =>");
for (int i = 0; i < m+1; i++) {
    printf(" %i", ai[i]);
}

double* acoo = (double*)mkl_malloc(NNZ * sizeof(double), 64);
MKL_INT* ir = (MKL_INT*)calloc(NNZ, sizeof(MKL_INT));
MKL_INT* jc = (MKL_INT*)calloc(m + 1, sizeof(MKL_INT));


#pragma warning(disable:4996)
mkl_dcsrcoo(job, &n, acsr, aj, ai, &NNZ, acoo, ir, jc, &info);

printf("\n\n\n");
printf("            CSR to COO");
printf("\n-----------------------------------------------------------------");
for (int i = 0; i < NNZ; i++) {
    printf("\n | Row => %i   Column => %i   Value => %f",ir[i], aj[i], acsr[i]);
}
```

Código principal 1

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (***Resultado de ejecución***).



Resultado de la ejecución

Optativos

Practica 1

Descripción

1. Caracterizar el ahorro promedio que se produce en almacenamiento en función de los diferentes valores de densidad del ejercicio 2.
2. Programar un generador de matrices aleatorias escasas simétricas.

Trabajo realizado

Generamos 10 matrices escasas en las que en cada iteración aumentamos un 10% la densidad y comprobamos el número de elementos que se generan para cada codificación (COO y CSR). De esta manera podemos ver el coste que tendría almacenarlas en las dos codificaciones. Representamos dicho coste en una gráfica de líneas donde vemos que al aumentar la densidad aumenta el coste de almacenamiento.

Adaptamos el código de la función de generar matrices aleatorias de manera que además de que genere matrices aleatorias, genere matrices dispersas.

Código Programa:

```
%Optativo 1
nCol = 10;
nRow = 10;

for i=0.1:0.1:1
    [A] = GenerateSparse(nRow,nCol,i,1,10);
    [f1,c1,v1] = COO(A);
    [f2,c2,v2] = CSR(A);

    density(floor(i*10)) = i;
    cooSize(floor(i*10)) = length(f1)+length(c1)+length(v1);
    csrSize(floor(i*10)) = length(f2)+length(c2)+length(v2);
end


hold on;
plot(density,cooSize,density,csrSize);
legend("COO","CSR");
ylabel("Elementos");
xlabel("Densidad");
```

Código principal 1

```
%Optativo 2
[b] = GenerateSymmetricSparse(5,5,0.5,1,5);
b';
```

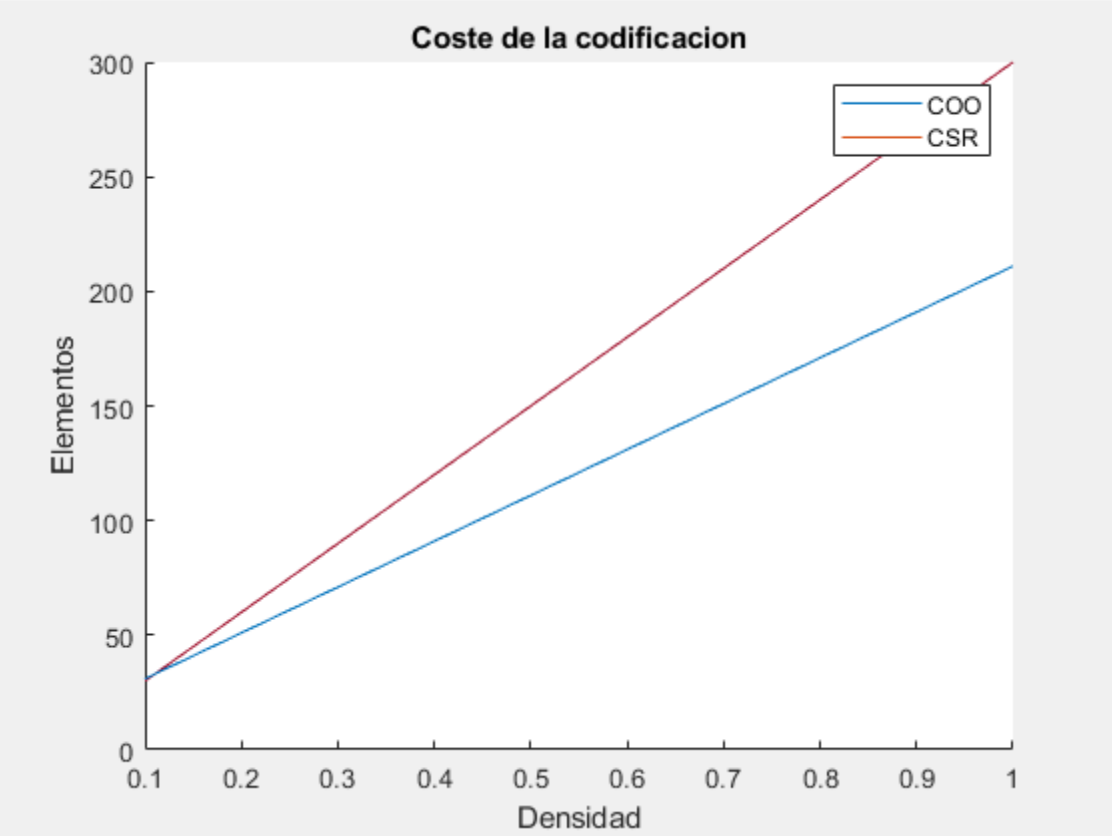
Código principal 2

```
function [A] = GenerateSymmetricSparse(nRow,nCol,zDensity,vMin,vMax)
    A = zeros(nRow,nCol);
    i=floor((rand(1)*nRow)+1);
    j=floor((rand(1)*nCol)+1);

    nElement = floor((nRow * nCol * zDensity)/2);
    for e = 1:nElement
        while A(i,j)~=0
            i=floor((rand(1)*nRow)+1);
            j=floor((rand(1)*nCol)+1);
        end
        A(i,j) = (rand(1)* vMax )+ vMin;
        A(j,i) = A(i,j);
    end
end
```

Código principal 2.2

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (***Resultado de ejecución***).



Resultado de la ejecución 1

```
b =

     0     0    3.3709     0    2.7126
     0     0         0     0    5.3277
3.3709     0         0     0    3.2762
     0     0         0  1.3058     0
2.7126  5.3277    3.2762     0    5.9720


ans =

     0     0    3.3709     0    2.7126
     0     0         0     0    5.3277
3.3709     0         0     0    3.2762
     0     0         0  1.3058     0
2.7126  5.3277    3.2762     0    5.9720
```

Resultado de la ejecución 2

Practica 2

Descripción

Probar conversiones a otros formatos.

Trabajo realizado

Usamos el cambio de CSR a BSR. Esta rutina convierte una matriz cuadrada dispersa A almacenado en el formato de fila dispersa comprimida (CSR) (variación de 3 matrices) al formato de fila dispersa de bloque (BSR) y viceversa.

Y usamos CSR a CSC que es bastante parecida solo que la CSR se almacena en filas y la rutina CSC se almacena en columnas.

Código Programa:

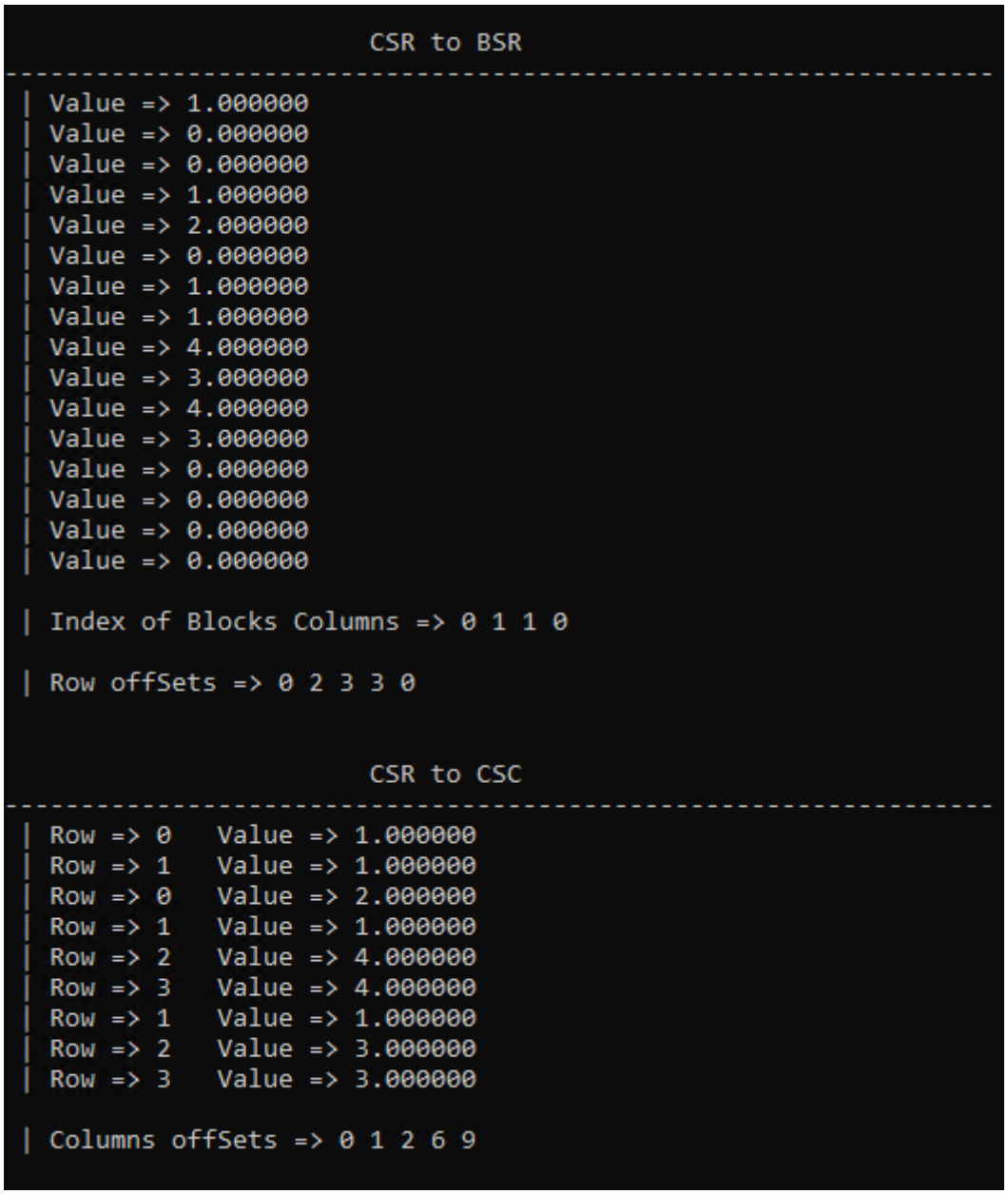
Los códigos de la librería ***MKL*** de los distintos apartados serán puesto a continuación con la referencia del apartado en el pie de la foto.

```
//Practica 2 optativo 3

MKL_INT mblk =  2;
MKL_INT mBSR = 5;
MKL_INT ldabsr = 4;

double* absr = (double*)mkl_malloc(ldabsr * mblk * mblk *sizeof(double), 64);
MKL_INT* jab = (MKL_INT*)calloc(ldabsr, sizeof(MKL_INT));
MKL_INT* iab = (MKL_INT*)calloc(mBSR + 1, sizeof(MKL_INT));
MKL_INT job1[6] = {0, 0, 0, 0, NNZ, 1};

#pragma warning(disable:4996)
mkl_dcsrbsr(job1, &mBSR, &mblk, &ldabsr, acsr, aj, ai, absr, jab, iab, &info);

printf("\n\n\n");
printf("            CSR to BSR");
printf("\n-----------------------------------------------------------------");

for (int i = 0; i < ldabsr * mblk * mblk; i++) {
    printf("\n | Value => %f", absr[i]);
}

printf("\n\n | Index of Blocks Columns =>");
for (int i = 0; i < ldabsr; i++) {
    printf(" %i", jab[i]);
}

printf("\n\n | Row offSets =>");
for (int i = 0; i < mBSR; i++) {
    printf(" %i", iab[i]);
}


double* acsc = (double*)mkl_malloc(NNZ * NNZ * sizeof(double), 64);
MKL_INT* ja1 = (MKL_INT*)calloc(NNZ, sizeof(MKL_INT));
MKL_INT* ia1 = (MKL_INT*)calloc(m + 1, sizeof(MKL_INT));

mkl_dcsrcsc(job, &n, acsr, aj, ai, acsc, ja1, ia1, &info);

printf("\n\n\n");
printf("            CSR to CSC");
printf("\n-----------------------------------------------------------------");
for (int i = 0; i < NNZ; i++) {
    printf("\n | Row => %i   Value => %f", ja1[i], acsc[i]);
}

printf("\n\n | Columns offSets =>");
for (int i = 0; i < m + 1; i++) {
    printf(" %i", ia1[i]);
}
printf("\n\n\n");
```

Código principal 1

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (***Resultado de ejecución***).



Resultado de la ejecución