# Actividad práctica 1

Se realiza la codificación COO y CSR en Matlab.

## Ejercicio 1

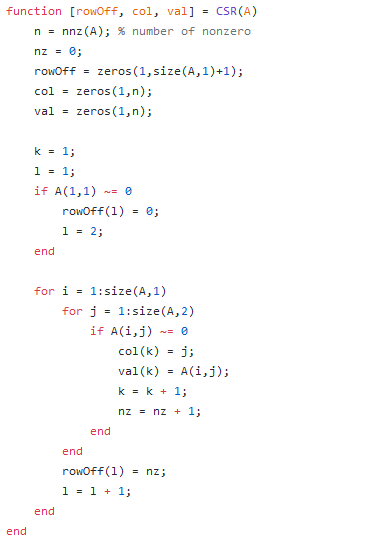
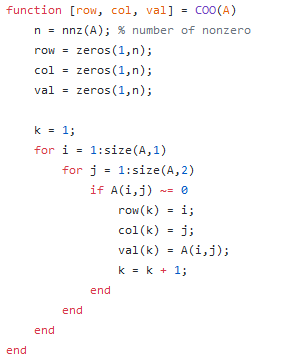
Se implementa dos funciones en Matlab que devuelvan la codificación COO y CSR para una matriz que se le pase como entrada.

Figura 2. Codificación CSR

Figura 1. Codificación COO

Para el método *COO* se obtiene el número de valores no nulos de la matriz mediante *nnz()* y se inicializan los vectores a cero. A continuación, se recorre la matriz con el fin de buscar los valores no nulos, a medida que se vayan encontrando se almacena el índice de fila en el vector de filas, el índice de columnas en el vector de columnas y el valor no nulo en el vector de valores.

Para el método *CSR* se obtiene el número de valores no nulos de la matriz mediante *nnz()* y se inicializan los vectores a cero (en este caso, la dimensión del vector de filas será el número de filas + 1 y se comprueba si el primer elemento de la matriz no es nulo). A continuación, se recorre la matriz con el fin de buscar los valores no nulos, a medida que se vayan encontrando se almacena el índice de columnas en el vector de columnas y el valor no nulo en el vector de valores. Al terminar de recorrer una fila se almacena en el vector de filas el número de elementos no nulos de la fila.

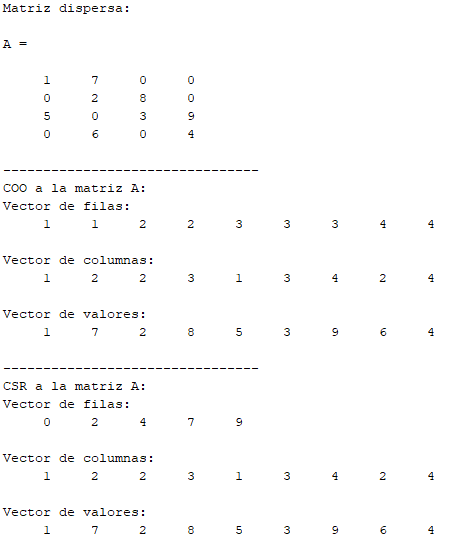
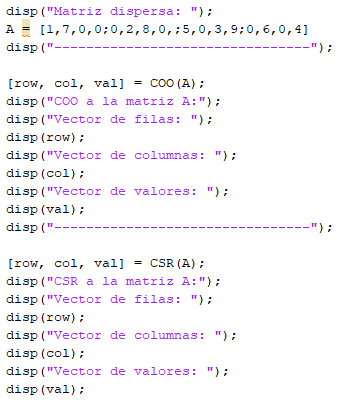
Como ejemplo utilizaremos la siguiente matriz 🡪 A =

Figura 4. Resultado obtenido

Figura 3. Ejemplo de codificación

## Ejercicio 2

Se programa un generador de matrices aleatorias escasas en Matlab, tomando como entrada la densidad deseada de valores nulos para la matriz y el rango de valores.



Figura 4. Ejemplo de matriz dispersa

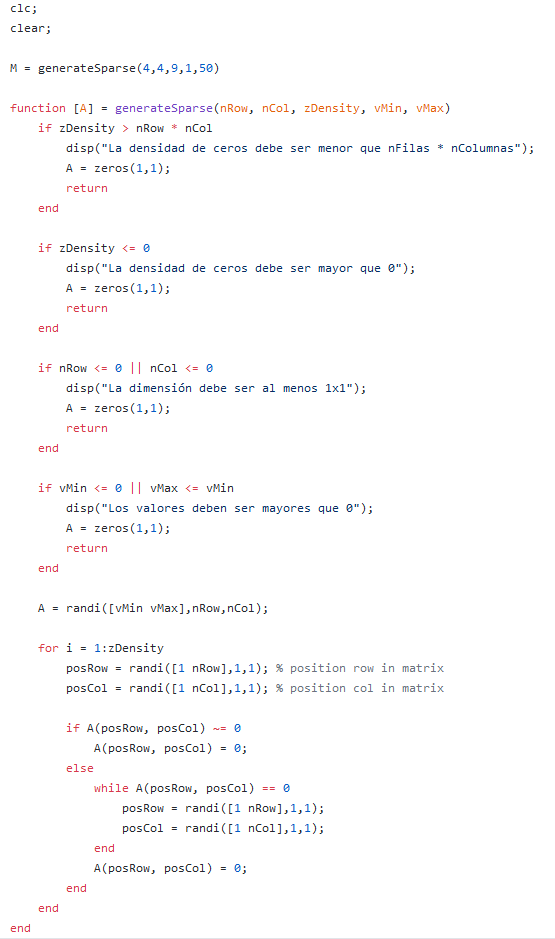
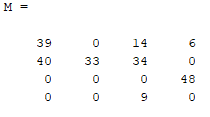


Figura 5. Generador de matrices dispersas

Figura 6. Resultado obtenido

El método toma los siguientes parámetros de entrada: *generateSparse* (nº de filas, nº de columnas, nº de valores nulos, rango de valores inferior, rango de valores superior). Primero se comprueba los parámetros que sean válidos y a continuación se genera una matriz de valores aleatorios del rango pasado por parámetro. Luego se itera tantas veces como números de ceros indicado, se genera 2 números aleatorios entre el rango de la dimensión de la matriz (los cuales será una posición aleatoria de la matriz) y se almacena un 0 en la posición.

## Ejercicio 3 (optativo)

A continuación, se caracteriza el ahorro promedio que se produce en almacenamiento en función de los diferentes valores de densidad del ejercicio 2. Para ello se crean matrices de 10x10 (y más adelante de 100x100) con densidades ascendentes hasta el número de elementos de la matriz menos uno (ya que sino en la última iteración se obtendría una matriz nula). Se pasa la misma codificar la misma matriz mediante COO y CSR, y a continuación se obtiene el número de bytes que ocupa la matriz (sin codificar), la matriz codificada en COO y CSR (se suman los bytes que ocupan los tres vectores) mediante la función *whos* de MATLAB.

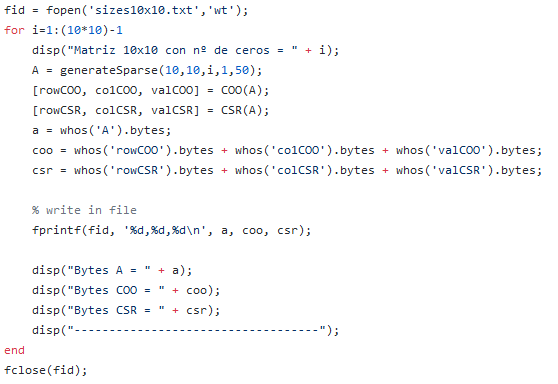


Figura 7. Cálculo de los bytes para distintas densidades según la codificación

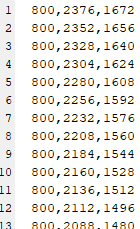
Estos valores se escriben en un fichero en el siguiente formato separados por comas:

Figura 8. Formato del fichero de salida

A continuación, se grafican los resultados obtenidos para analizar el ahorro que se produce en almacenamiento en función de los diferentes valores de densidad.

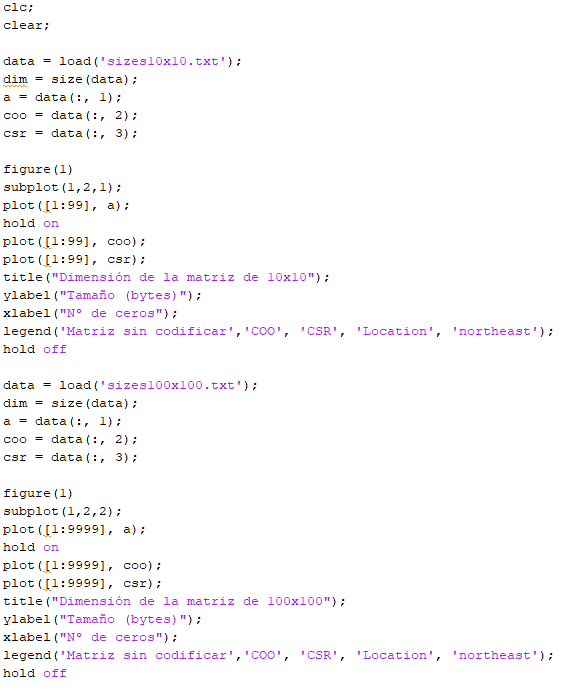


Figura 9. Fichero para obtener las gráficas

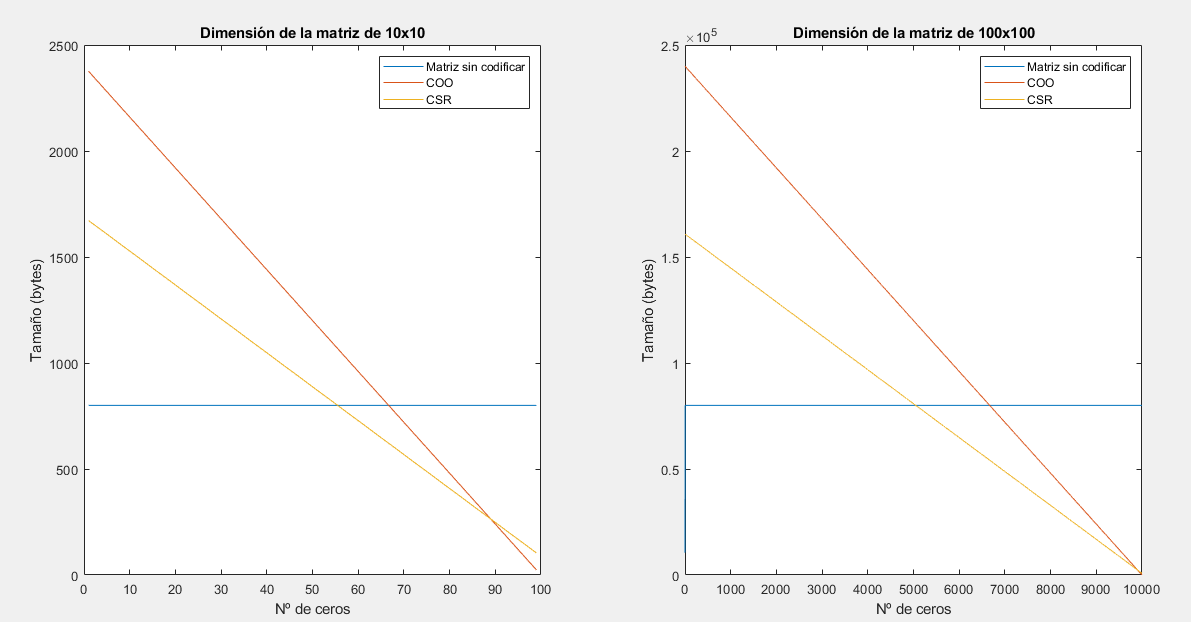
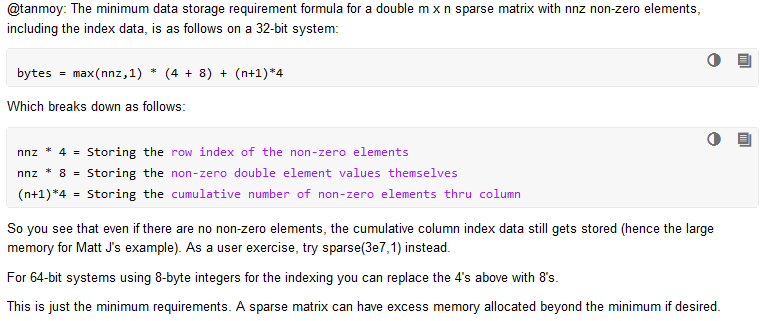
Observamos que ambas gráficas para distintos tamaños de matrices son iguales, pero en distinta escala. Se observa que para matrices de dimensión 10x10 el número de bytes sin codificar es constante (800 bytes y para dimensión de 100x100 el almacenamiento es de 80.000 bytes) mientras que para COO y CSR es decreciente a medida que aumenta el número de ceros en la matriz. Además, después de que la matriz esté formada por un 90% de ceros la codificación COO necesita menos almacenamiento que la CSR. Esto es así ya que, cuando hay un número muy pequeño de valores no nulos, la dimensión del vector de filas de la codificación COO será menor al vector de filas de la codificación CSR la cual es siempre es constante (número de filas + 1).

Figura 10. Gráficas obtenidas para matrices de tamaño 10x10 y 100x100

Por lo que podemos decir que a partir de un umbral merecerá la pena codificar una matriz que esté formada por un número de ceros significativo. Por ejemplo, para el caso de la matriz de dimensión 10x10, cuando más de 55% de la matriz esté formada por ceros se ahorrará almacenamiento si se codifica mediante CSR y cuando sea mayor que 70% mediante COO.

(BUSCAR o si no lo entiendo pos para la basura)



## Ejercicio 4 (optativo)

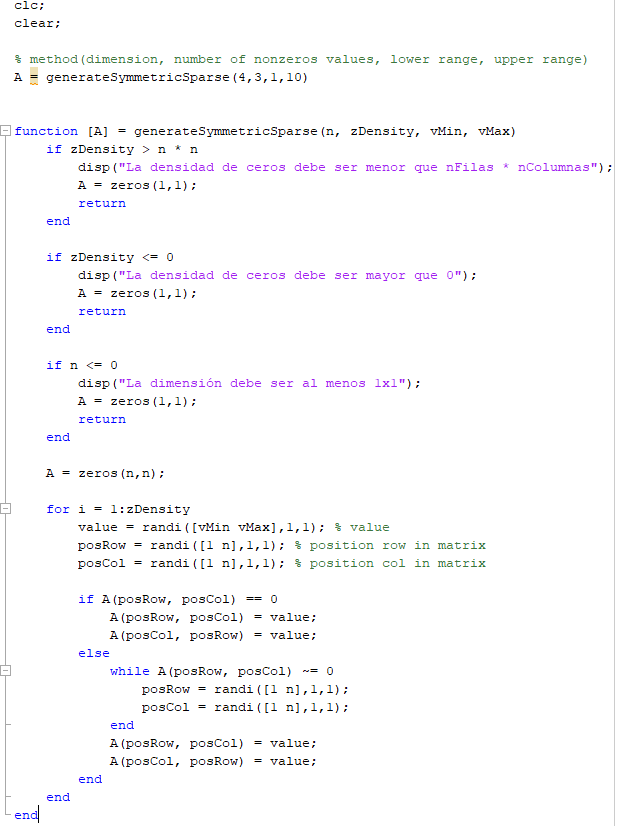
Se implementa un generador de matrices aleatorias escasas simétricas. Para ello se sigue prácticamente la misma manera de implementar el ejercicio 2 aunque los parámetros son ligeramente distintos. Las matrices simétricas deben ser cuadradas por lo que se le pasa como parámetros de entrada: *generateSymmetricSparse* (dimensión de la matriz, nº de valores únicos no nulos, rango de valores inferior, rango de valores superior). Se comprueban estos parámetros al principio del método.

Figura 11. Generador de matrices dispersas simétricas

Se inicializa una matriz de ceros y se itera tantas veces como el número de valores no nulos se pasa como parámtero. Se genera una posición aleatoria para almacenar un valor (también aleatorio) en dos posiciones que no hayan valores nulos. Se muestra el resultado obtenido con una dimensión de matriz 4x4, tres valores únicos no nulos con valores en un rango de 1-10:

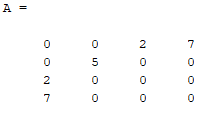
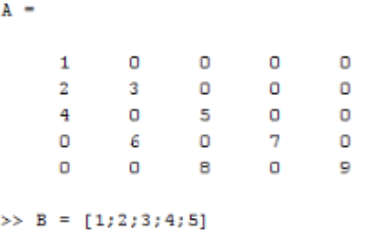


Figura 12. Resultado obtenido

# Actividad práctica 4

## Ejercicio 1 (optativo)

Se resuelve el siguiente sistema Ax = b aplicando la metodología DSS:



# Referencias

*ULPGC*. (s.f.). Obtenido de https://ncvt-aep.ulpgc.es/cv/ulpgctp21/pluginfile.php/412003/mod\_resource/content/9/6%20Matrices%20dispersas.pdf