MÉTODOS NUMÉRICOS PARA LA COMPUTACIÓN

**Tema 6: Matrices dispersas**

**2020/21**

X de diciembre de 2020

**Grupo 03:** José María Amusquívar Poppe y Prashant Jeswani Tejwani

Universidad de Las Palmas de Gran Canaria

Escuela de Ingeniería en Informática

Índice

[Actividad práctica 1 3](#_Toc58866292)

[Ejercicio 1 3](#_Toc58866293)

[Ejercicio 2 4](#_Toc58866294)

[Ejercicio 3 (optativo) 6](#_Toc58866295)

[Ejercicio 4 (optativo) 8](#_Toc58866296)

[Actividad práctica 2 9](#_Toc58866297)

[Ejercicio 1 9](#_Toc58866298)

[Ejercicio 2 (optativo) 9](#_Toc58866299)

[Actividad práctica 3 9](#_Toc58866300)

[Ejercicio 1 (optativo) 9](#_Toc58866301)

[Ejercicio 2 (optativo) 9](#_Toc58866302)

[Actividad práctica 4 9](#_Toc58866303)

[Ejercicio 1 (optativo) 9](#_Toc58866304)

[Referencias 10](#_Toc58866305)

# Actividad práctica 1

Se realiza la codificación COO y CSR en Matlab.

## Ejercicio 1

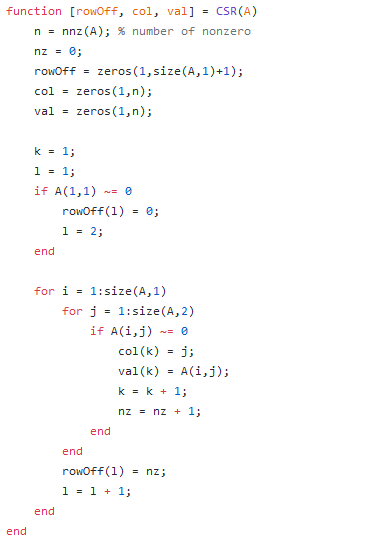
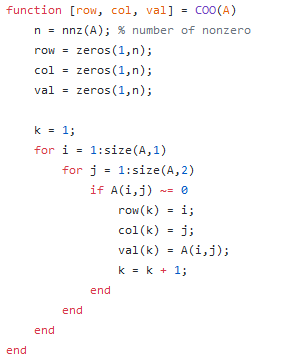
Se implementa dos funciones en Matlab que devuelvan la codificación COO y CSR para una matriz que se le pase como entrada.

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.

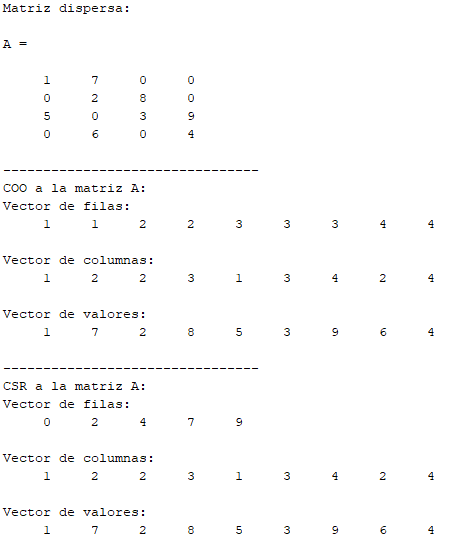
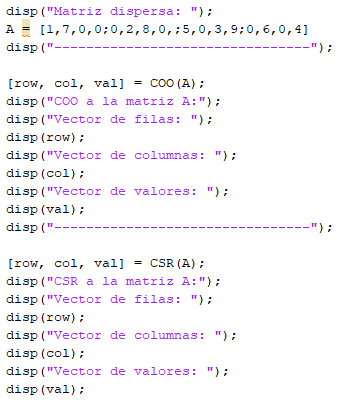
Como ejemplo utilizaremos la siguiente matriz 🡪 A =

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

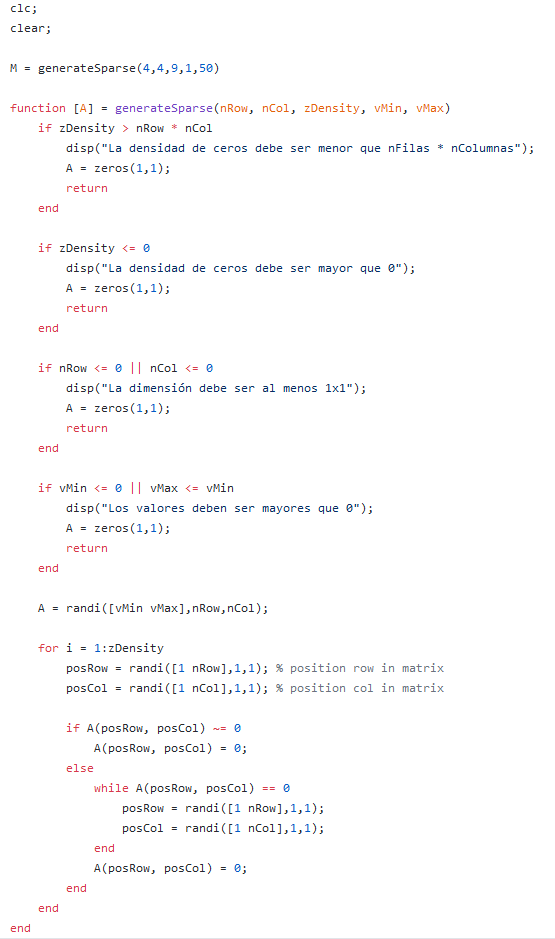
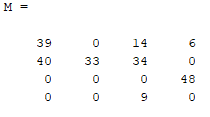


Figura 5. Generador de matrices dispersas

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.

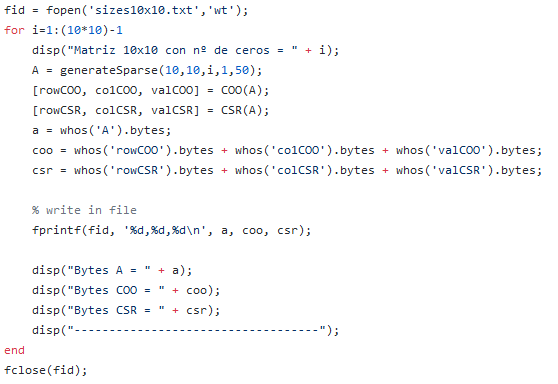


Figura 7. Cálculo de los bytes para distintas densidades según la codificación

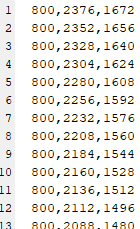
Estos valores se escriben en un fichero en el siguiente formato separados por comas:

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.

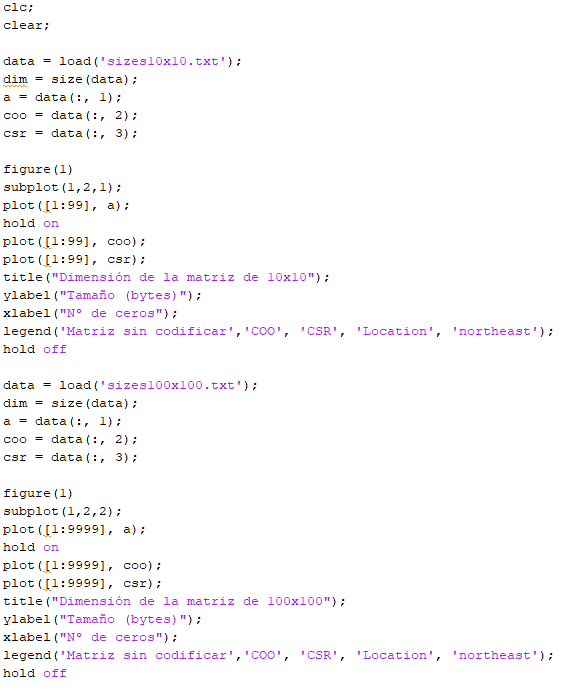


Figura 9. Fichero para obtener las gráficas

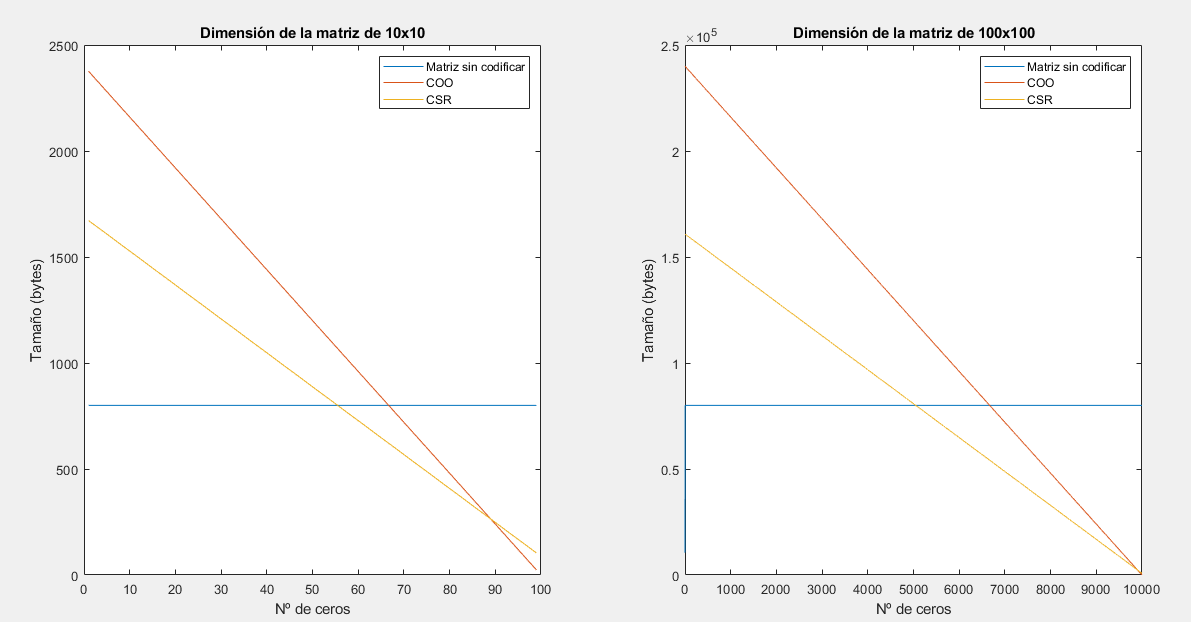
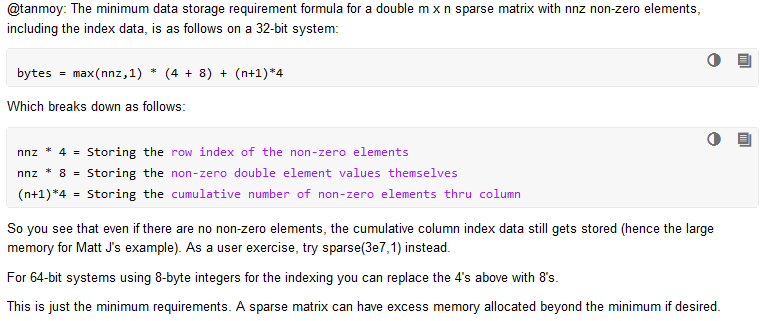
Observamos que ambas gráficas para distintos tamaños de matrices son iguales, pero en distinta escala. Se observa que para matrices de dimensión 10x10 el número de bytes sin codificar es constante (800 bytes y para dimensión de 100x100 el almacenamiento es de 80.000 bytes) mientras que para COO y CSR 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)

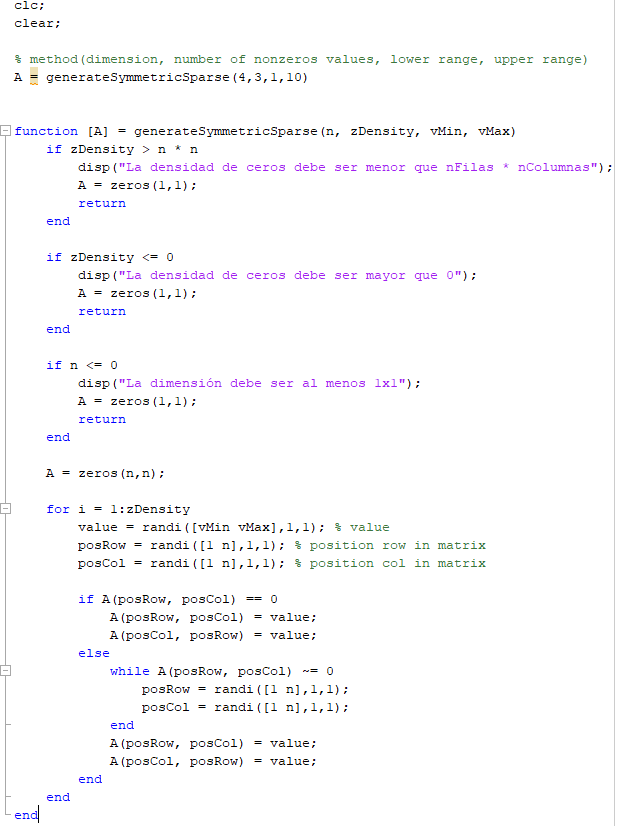
Se implementa un generador de matrices aleatorias escasas simétricas. Para ello se sigue prácticamente la misma manera de implementar el ejercicio 2 aunque los parámetros son ligeramente distintos. Las matrices simétricas deben ser cuadradas por lo que se le pasa como parámetros de entrada: *generateSymmetricSparse* (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:

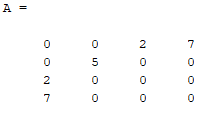


Figura 12. Resultado obtenido

# Actividad práctica 2

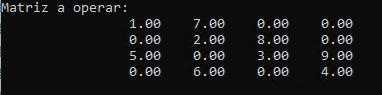
Se realiza la codificación dispersa con MKL en C++. La matriz a utilizar en esta práctica es la misma que se ha utilizado en el primer apartado (Figura 13).

Figura 13. Matriz a comprimir usando los distintos formatos.

## Ejercicio 1

Para poder realizar las operaciones de compresión “CSR” y “COO”, se ha empleado funciones de la librería “MKL”.

### CSR

El primero que se ha calculado es la compresión “CSR”, empleando para ello la función:

Figura 14. Función que comprime una matriz escasa en formato "CSR".

De la figura anterior se puede apreciar el uso de la instrucción “pragma warning()”. Esta función se ha utilizado para desactivar una advertencia de “Visual Studio”, que informaba que la función usada debajo estaba en desuso, por lo que era imposible ejecutar el código sin desactivarlo.

Para que esta función se ejecute correctamente, es necesario pasarle como parámetros una serie de datos como las dimensiones de la matriz “m” y “n” (usando “&” para pasarle su dirección de memoria), “matrixOver” corresponde con la propia matriz escasa acompañado de su “lda”, los siguientes tres parámetros se corresponden con variables donde se almacenarán los datos resultados, “valuesOutCSR” es el vector donde se guardarán los valores distintos de cero de la matriz, “ja” e “ia” también son vectores que almacenarán los índices de las columnas donde se sitúan los elementos distinto de cero, y el número de elementos distinto de cero que contiene cada fila, respectivamente; y el último parámetro de la función es un entero que guarda el resultado de la llamada, tanto si ha sido exitosa como si ha sido errónea.

El parámetro “job” es el más especial de todos estos, pues éste es un vector de enteros de 6 elementos, en el que cada uno especifica una configuración de compresión. El primer elemento especifica si se desea convertir de matriz escasa a formato CSR (un 0) o viceversa (un 1); el segundo y tercer elemento especifica si los índices de las matrices empiezan en 0 (un 0) o en 1 (un 1); el cuarto elemento configura la porción de matriz almacenada, en este caso un 2 especifica que se desea almacenar toda la matriz; el quinto elemento recibe el número de elementos distinto de cero de la matriz (“nnz” fue calculado con una función específica); y el último elemento configura si se desea almacenar los resultados en los vectores “ja” e “ia”.

Figura 15. Inicialización del vector "job" para el formato "CSR".

Destacar que, dado que el vector “job” varía según el tipo de compresión que se realice, se ha utilizado un método que se encarga de modificar el vector interno “job” con los nuevos valores, justo antes de llamar a cualquier método de compresión (Figura 15).

Una vez se ha llamado al método de la figura 14, se procede a imprimir los valores de los tres vectores resultado para comprobar que son correctos:

############FIGURA de vectores CSR###########

### COO

El segundo tipo de compresión que se ha realizado es el “COO”, utilizando para ello la función:

Figura 16. Función que comprime una matriz escasa en formato "COO".

Algo que destacar de esta función es que, para poder llamarla, primero se ha de realizar la compresión “CSR”, obteniendo así sus tres vectores resultados. Pues, al igual que esta función”, ésta recibe una serie de parámetros similares tales como la dimensión de la matriz (“n”); los tres vectores obtenidos mediante la compresión “CSR”; el número de elementos no nulos de la matriz (“nnz”); un vector que almacenará los valores distintos de cero (“valuesOutCOO”), por lo que este vector debe ser análogo al vector “valuesOutCSR”; también recibe dos vectores que almacenarán las coordenadas de estos últimos valores, “rows” las filas y “cols” las columnas, por lo que el vector “cols” debe ser igual que el vector “ja”; y también recibe un entero con el código de salida.

En este tipo de compresión, el vector “job” es parecido al vector utilizado en “CSR”, con la única diferencia que en la última posición del vector debe situarse un 3, con esto se consigue que los tres vectores sean escritos con los datos obtenidos.

Figura 17. Inicialización del vector "job" para el formato "COO".

Una vez procesada esta llamada, se procede a imprimir los resultados y verificar que estos sean adecuados y correctos:

############FIGURA de vectores CSR###########

## Ejercicio 2 (optativo)

Para este apartado se ha realizado la conversión a dos tipos distintos, empleando la misma metodología que el apartado anterior.

### SKY LINE

Para realizar este apartado, se ha empleado la función de “MKL”:

Figura 18. Función que comprime una matriz escasa en un formato “SKY LINE”.

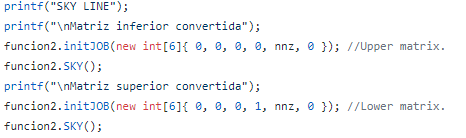
Al igual la compresión “COO”, este tipo de compresión requiere haber realizado una compresión a formato “CSR” antes. Los parámetros son similares a los explicados con anterioridad, con la diferencia que los resultados de esta conversión se almacenan en dos vectores, “asky” y “pointers”. Antes de explicar estos dos vectores, se debe explicar la configuración que contiene el vector “job”, el cual tiene la siguiente forma:

Figura 19. Inicialización del vector "job" para conversión a formato "SKY LINE".

El vector “job” tiene la misma estructura que los mencionados anteriormente con la única diferencia que el cuarto elemento especifica si se quiere comprimir el triángulo superior de la matriz (un 0) o el inferior (un 1); y, además, para almacenar todos los resultados en los dos vectores, el último elemento debe ser igual a 0.

Pues bien, esta función devuelve dos vectores, si el cuarto elemento de “job” es igual a 0, el vector “asky” guardará aquellos valores distintos de cero que se encuentren en, y por encima de, la diagonal principal, recorriéndolos por columnas, de arriba hacia abajo, y respetando aquellos ceros que se encuentren entre valores no nulos. Entonces, el vector “pointers” representa todos los índices de aquel primer elemento de cada nueva columna almacenada en el vector “asky”. Por tanto, si el cuarto elemento del vector “job” es igual a 1, entonces se almacena en el vector “asky” aquellos valores no nulos que se encuentren en, y por debajo, de la diagonal principal, además de su respectivo vector “pointers”; por ello se realizan dos llamadas a la misma función (Figura 19).

Por ejemplo, teniendo la siguiente matriz de ejemplo:

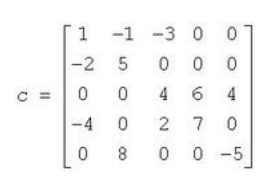
Se obtiene que sus vectores para el triángulo superior son (con indexación igual 1):

Figura 20. Matriz escasa de ejemplo para compresión "SKY LINE".

Y para el triángulo inferior (con indexación igual 1):

Una vez llamada la función, se procede a imprimir los resultados y corroborar su validez:

############FIGURA de vectores CSR###########

### CSC

Este tipo de conversión emplea el mismo mecanismo que la conversión CSR, ya que simplemente, en lugar de recorrer la matriz por filas, se recorre por columnas, por lo que el orden de almacenamiento es distinto. La función que se emplea es:

Figura 21. Función que comprime una matriz escasa en formato "CSC".

Tal como se puede apreciar, los parámetros que recibe esta función son idénticos a los de la función “CSR”, entendiendo que para la función “CSR” se requiere la matriz escasa, mientras que para esta función simplemente los tres vectores resultados de la compresión “CSR”. También, su vector “job” es mucho más simple que cualquier otro, pues requiere de pocas configuraciones:

Figura 22. Inicialización del vector "job" para el formaro "CSC".

De este vector, el único elemento distinto a los anteriores vistos es el último, pues éste configura si se desea guardar los tres vectores resultados (un 1) o no (un 0).

A continuación, se representan los resultados de esta conversión para su respectiva verificación:

############FIGURA de vectores CSR###########

# Actividad práctica 3

## Ejercicio 1 (optativo)

Se realiza una operación de multiplicación de matrices escasas grandes utilizando BLAS, repitiendo y utilizando *SparseBLAS*. Se comparar el tiempo invertido.

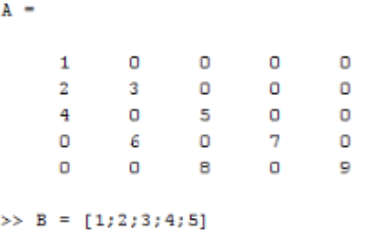
## Ejercicio 2 (optativo)

Se estima la ganancia en GFLOPS resultante.

# 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