

# Guion 2: Compresión sin pérdida

## Código de Huffman

---

### Información sobre la entrega de la práctica

Las prácticas se entregarán en un único fichero comprimido Practica02ApellidoNombre.zip. El fichero contendrá:

- Las funciones de Matlab a realizar en ficheros .m con los nombres de las funciones que se indiquen en el guion.
- Los trozos de código a realizar, que se entregarán todos en los pasos correspondientes de un único fichero .m llamado Practica02ApellidoNombre.m . Este fichero lo crearás modificando el fichero .m Practica02MolinaRafael.m en el servidor.
- Las discusiones y respuestas solicitadas en el guion se entregarán en un único fichero pdf. El nombre del fichero será Practica02ApellidoNombre.pdf. Lo construirás editando Practica02MolinaRafael.doc y salvándolo en formato pdf.

---

En este guion estudiaremos el código de Huffman como método de compresión de datos sin pérdida, es decir, si se comprimen y descomprimen los datos se vuelven a obtener los datos originales.

### Sobre la Codificación Huffman

Puedes codificar ficheros de datos mediante codificación Huffman utilizando el software desarrollado por [Michael Dipperstein](#). En la página web de [Michael Dipperstein](#) se puede encontrar una clara explicación del algoritmo y las consideraciones de implementación del mismo. Aunque nosotros haremos las prácticas en Matlab te recomendamos que visites el sitio de [Michael Dipperstein](#), contiene unas páginas muy buenas sobre compresión.

### Codificación Huffman usando Matlab

#### Paso 1

Vamos a comenzar suponiendo que tenemos una fuente de la que conocemos los símbolos que genera y la probabilidad de cada una de ellos. Matlab nos permite generar el código de Huffman de una forma muy sencilla

```
A=[1:5];
p=[0.2 0.4 0.2 0.1 0.1];
D=huffmandict(A,p);
for i=1:length(A)
fprintf('Letra Alfabeto %s Código
      %s\n',num2str(D{i,1}),num2str(D{i,2}))
end
```

La salida de este trozo de código sería

```
Letra Alfabeto 1 Código 0 0 0
Letra Alfabeto 2 Código 1
Letra Alfabeto 3 Código 0 1
Letra Alfabeto 4 Código 0 0 1 1
Letra Alfabeto 5 Código 0 0 1 0
```

## Paso 2

Podemos utilizar otros símbolos para los elementos del alfabeto. Halla y muestra, siguiendo el esquema del paso anterior, el código de Huffman correspondiente al alfabeto {A,B,C,D,E} con probabilidades  $p=[0.2 \ 0.4 \ 0.2 \ 0.1 \ 0.1]$ . Cuya ejecución debe producir la salida que se muestra más abajo. Incluye el código en el paso 2 de Practica02ApellidoNombre.m

```
Letra Alfabeto A Código 0 0 0
Letra Alfabeto B Código 1
Letra Alfabeto C Código 0 1
Letra Alfabeto D Código 0 0 1 1
Letra Alfabeto E Código 0 0 1 0
```

## Paso 3

Podemos utilizar la función `huffmandict` para construir diccionarios de máxima o mínima varianza. El siguiente ejemplo construye el código de Huffman con mínima varianza para el alfabeto  $A=[1:5]$  y probabilidades  $p=[0.2 \ 0.4 \ 0.2 \ 0.1 \ 0.1]$ . Entiende las opciones de la función `huffmandict`.

```
A=[1:5];
p=[0.2 0.4 0.2 0.1 0.1];
D=huffmandict(A,p,2,'min');

for i=1:length(A)
    fprintf('Letra Alfabeto %s Código
    %s\n',num2str(D{i,1}),num2str(D{i,2}))
end
```

La salida de este código es

```
Letra Alfabeto 1 Código 1 1
Letra Alfabeto 2 Código 0 0
Letra Alfabeto 3 Código 1 0
Letra Alfabeto 4 Código 0 1 1
Letra Alfabeto 5 Código 0 1 0
```

## Paso 4

En los ejemplos anteriores hemos supuesto que conocemos las probabilidades de cada símbolo. En condiciones reales estas probabilidades tendrán que ser estimadas/calculadas utilizando un fichero concreto o un conjunto de ficheros. Antes de ver como se estiman/calculan las probabilidades de cada símbolo recordemos una forma de generar símbolos. En el ejemplo siguiente simulamos la generación de 10 símbolos del alfabeto anterior

```
A=[1:5];
p=[0.2 0.4 0.2 0.1 0.1];
rng(0);
stream= randsrc(1,10,[A;p])
```

El resultado de este trozo de código es

```
4      5      1      5      3      1      2      2      5      5
```

Observa que podría no haber aparecido algún símbolo.

## Paso 5

Vamos ahora a leer un fichero del que estimaremos la probabilidad de cada letra de un alfabeto. Primero leemos el fichero Don Quijote de la Mancha - Miguel de Cervantes.txt y lo almacenamos en un vector fila de bytes

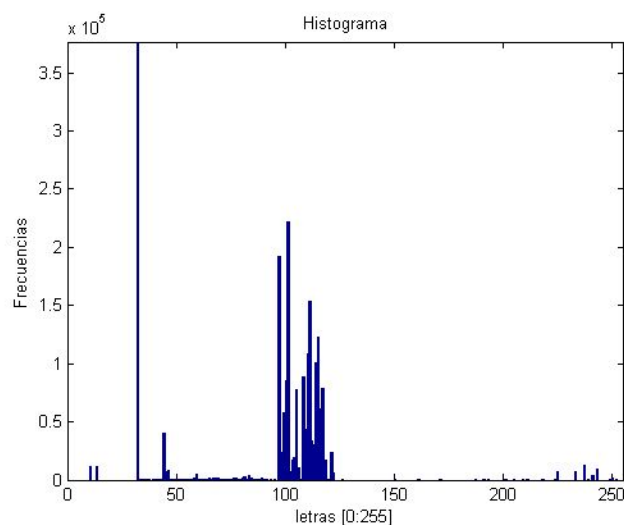
```
clc; clear all;
fichero= 'Don Quijote de la Mancha - Miguel de Cervantes.txt'
fid=fopen(fichero,'r')
seq=fread(fid,'*uint8');
fclose(fid);
seq=reshape(seq,1,length(seq)); %leídos datos en seq vector
fila
fprintf('Tamaño del fichero original en bytes
%d\n',numel(seq))
```

Tamaño del fichero original en bytes 2117266

## Paso 6

Calculamos su histograma y lo mostramos

```
letras=[0:255];  
histo=histc(seq,letras); %calculamos histograma e indices  
bar(letras,histo);  
axis('tight');  
xlabel('letras [0:255]');  
ylabel('Frecuencias');  
title('Histograma');  
xlabel('letras [0:255]');  
ylabel('Frecuencias');  
title('Histograma');
```



## Paso 7

De las 256 letras alfabeto solo usamos las del que aparecen en el fichero 'Don Quijote de la Mancha - Miguel de Cervantes.txt'. Hállalas y calcula sus probabilidades. Puedes utilizar la función `find`, por ejemplo `find(histo>0)` que devuelve las posiciones de la matriz `histo` que almacenan valores mayores que cero. No debes confundir las posiciones de las letras en `letras` que van de 1 a 255 con los valores de las letras que van de 0 a 255. Escribe el código correspondiente en el paso 7 de `Practica02ApellidoNombre.m`

## Paso 8

A continuación calculamos el diccionario para las letras usadas y sus probabilidades y codificamos la secuencia que teníamos suponiendo que has almacenado en `letras_usadas` las letras utilizadas (números en `[0:255]`) y sus probabilidades en `prob_letras_usadas`.

En el código anterior `letras_usadas` puede ser elementos en el rango `[0:255]` o los caracteres correspondientes. Mira la documentación de `huffmandict`, yo he utilizado números en `[0:255]`. Finalmente, muestra el número de letras utilizadas y la longitud de la codificación

```
[dict,avglen] = huffmandict(letras_usadas,prob_letras_usadas) ;
```

```
seq_codificada = huffmanenco(seq,dict);  
fprintf('Número de letras usadas %d\n',numel(letras_usadas))  
fprintf('Longitud de la secuencia codificada %d\n',  
length(seq_codificada))
```

debe devolver como resultados

Número de letras usadas 111

Longitud de la secuencia codificada 9333683

## Paso 9

¿Cómo guardaríamos el fichero codificado?. Podríamos utilizar una cabecera que tuviera las letras usadas (1 byte por letra) y sus probabilidades en formato double (8 bytes por letra) más una letra adicional, por ejemplo, 0 con probabilidad 0 (1 + 8 bytes más) para indicar que ya no tenemos más letras. A continuación guardaríamos la secuencia codificada que necesitaría

$\lceil \text{length}(\text{seq\_codificadas}/8) \rceil$  bytes

En nuestro ejemplo

```
tamagno_comprimido=(length(letras_usadas)+1)*1+ ...  
    (length(letras_usadas)+1)*8+ ...  
    ceil(length(seq_codificada)/8);  
fprintf('Tamaño fichero comprimido en bytes  
%d\n',tamagno_comprimido)
```

que produce

Tamaño fichero comprimido en bytes 1167719

Observa que no hemos discutido cómo terminar la decodificación. El uso de ceil nos dice que podemos tener unos bits al final del fichero codificado que no corresponden a ningún símbolo y que si los decodificamos nos habremos equivocado. Estos problemas los discutiremos al final del guión.

## Paso 10

¿Cómo reconstruiríamos el fichero comprimido?. Leeríamos la cabecera y de ella sacaríamos las letras\_usadas y prob\_letras\_usadas, después leeríamos la secuencia codificada que tendríamos que desempaquetar y convertir en una matriz de doubles que sólo contiene 0s y 1s y que llamaríamos seq\_codificada, esta seq\_codificada va a coincidir con la que obtuvimos en el paso 8. A continuación con letras\_usadas y prob\_letras\_usadas construimos el diccionario que sería el diccionario dict que obtuvimos en el paso 8 y finalmente decodificamos la secuencia usando

```
deco=huffmandeco(seq_codificada,dict);
```

## Paso 11

Muy importante, comprobamos que la secuencia original y decodificada son iguales

```
fprintf('¿Coinciden original y comprimido 1(S) 0 (N)?,
%d\n',...
       isequal(seq,uint8(deco)) )
```

¿Coinciden original y comprimido 1(S) 0 (N)?, 1

## Paso 12

Incluye en el paso 12 del fichero Practica02ApellidoNombre.pdf el tamaño del fichero original y el comprimido. Incluye también el factor de compresión.

[Escribe tus respuestas aquí.](#)

1. el tamaño original en bytes son 2117266 mientras que el comprimido es 1167719, el factor de compresion es  $2117266 * 2^3 / 116771 * 2^3 = 1.8131$

## Paso 13

Crea ahora la siguiente estructura

```
clc; clear all;
N=9;
Fichero=cell(N);
Fichero{1}='constitucion española.txt';
Fichero{2}='Fundacion e Imperio - Isaac Asimov.txt';
Fichero{3}='Cinco semanas en globo - Julio Verne.txt';
Fichero{4}='ptt1.pbm';
Fichero{5}='ptt4.pbm';
Fichero{6}='ptt8.pbm';
Fichero{7}='camera.pgm';
Fichero{8}='bird.pgm';
Fichero{9}='bridge.pgm';
fichero='ptt8.pbm';
```

## Paso 14

Para cada uno de los ficheros anteriores: 'constitucion española.txt', 'Fundacion e Imperio - Isaac Asimov.txt', 'Cinco semanas en globo - Julio Verne.txt', ptt1.pbm, ptt4.pbm, ptt8.pbm, camera.pgm, bird.pgm y bridge.pgm escribe en el paso 14 de Practica02ApellidoNombre.m el código Matlab que calcule el tamaño del fichero original, el comprimido usando el código de

Huffman y el factor de compresión obtenida en cada uno de los casos. El código Matlab debe comprobar también que los ficheros originales y descomprimidos son iguales.

Completa en el paso 14 de Practica02ApellidoNombre.pdf la tabla siguiente, realiza una comparación crítica de los resultados obtenidos y explica la variación en el factor de compresión.

[Escribe tus respuestas aquí](#)

Fichero	Tamaño original	Tamaño Comprimido	FC
constitucion española.txt	112246	64095	1.751244
Fundacion e Imperio - Isaac Asimov.txt	461298	259641	1.776676
Cinco semanas en globo - Julio Verne.txt	487020	275634	1.766908
ptt1.pbm	513249	87435	5.870063
ptt4.pbm	513249	145557	3.526103
ptt8.pbm	513249	115572	4.440946
camera.pgm	65551	59961	1.093227
bird.pgm	65551	57069	1.148627
bridge.pgm	65551	65356	1.002984

## Paso 15

Vamos a usar el código de Huffman para comprimir una imagen agrupando de dos en dos sus valores. Lo haremos con una imagen sintética que convertiremos a una secuencia, `seq`

```
clc; close all; clear all;
M=256;
img=uint8(zeros(M));
img(:,129:M)=255;
imshow(img);
img16=uint16(img);
img2=255*img16(:,1:2:size(img,2))+img16(:,2:2:size(img,2));
[U,V]=size(img2);
seq=reshape(img2,1,U*V);
```

## Paso 16

Codifica la matriz `img`, que no agrupa los bytes de dos en dos, usando el código de Huffman, comprueba que coinciden la secuencia original y la decodificación de la comprimida, calcula el espacio que ocupa la imagen comprimida incluida la cabecera y el factor de compresión. Escribe el trozo de código en el paso 16 de Practica02ApellidoNombre.m.

Escribe en el paso 16 de Practica02ApellidoNombre.pdf el factor de compresión calculada.

[Escribe tus respuestas aquí](#)

1. El tamaño del fichero original es 65536 mientras que el fichero comprimido ocupa 8219 bytes, por tanto el factor de compresión es 7.973719.

## Paso 17

Codifica la matriz `img2`, que agrupa los bytes de dos en dos, usando el código de Huffman, comprueba que coinciden la secuencia original y la decodificación de la comprimida, calcula el espacio que ocupa la imagen comprimida incluida la cabecera y el factor de compresión. Escribe el trozo de código en el paso 17 de `Practica02ApellidoNombre.m`.

Escribe en el paso 17 de `Practica02ApellidoNombre.pdf` el factor de compresión calculada.

[Escribe tus respuestas aquí](#)

1. el tamaño del fichero original es 65536 mientras que el fichero comprimido ocupa 4126 bytes, por tanto el factor de la compresión es 15.8836

## Paso 18

¿Cuáles son los factores de compresión para las matrices `img2` e `img` obtenidas en los dos pasos anteriores?. Discute los resultados obtenidos en los dos pasos anteriores en el paso 18 de `Practica02ApellidoNombre.pdf`.

[Escribe tus respuestas aquí](#)

1. el factor de compresión en el primer caso es 7.973719 y el segundo es de 15.88366, este segundo valor es casi el doble que el primero, esto es porque los datos en el segundo caso son considerados de dos en dos, por tanto el factor de compresión es, si son valores no uniformes, el doble que si son tomados de uno en uno

## Paso 19

Utiliza ahora la imagen `goldhill.pgm`

```
close all; clear all;
img=imread('goldhill.pgm');
imshow(img);
img16=uint16(img);
img2=255*img16(:,1:2:size(img,2))+img16(:,2:2:size(img,2));
```

## Paso 20

Incluye en el paso 20 del fichero `Practica02ApellidoNombre.m` el código en Matlab que calcula los factores de compresión para estas nuevas matrices `img2` e `img` obtenidas a partir de la



imagen bird.pgm del paso anterior. Incluye el tamaño de la cabecera. Si no pudieses codificar alguna de las dos matrices explica por qué.

Discute los resultados obtenidos en el paso 20 de Practica02ApellidoNombre.pdf.

[Escribe tus respuestas aquí](#)

1. Sobre la img el tamaño original es de 131072 bytes y al comprimir resulta 57179 por tanto el factor de compresión es de 2.29231, mientras que sobre la img2 el tamaño original es de 65536 bytes y al intentar comprimirlo resulta un tamaño de 80468 por lo que no se ha comprimido nada, al contrario ha aumentado el tamaño, su factor de compresión es 0.81444

## Paso 21

Lee la secuencia miss\_am.yuv, muéstrala y extrae la luminancia de los dos primeros fotogramas

```
clc, close all; clear all;
file_name = 'miss_am.yuv';
file_format = 'QCIF_PAL';
num_of_frames = 30;
[yuv_movie, yuv_array] = readYUV(file_name, num_of_frames, ...
file_format);
imshow(yuv_movie)
fotograma1=yuv_array(:,:,1,1);
fotograma2=yuv_array(:,:,1,2);
figure; imshow(fotograma1);
figure; imshow(fotograma2);
```

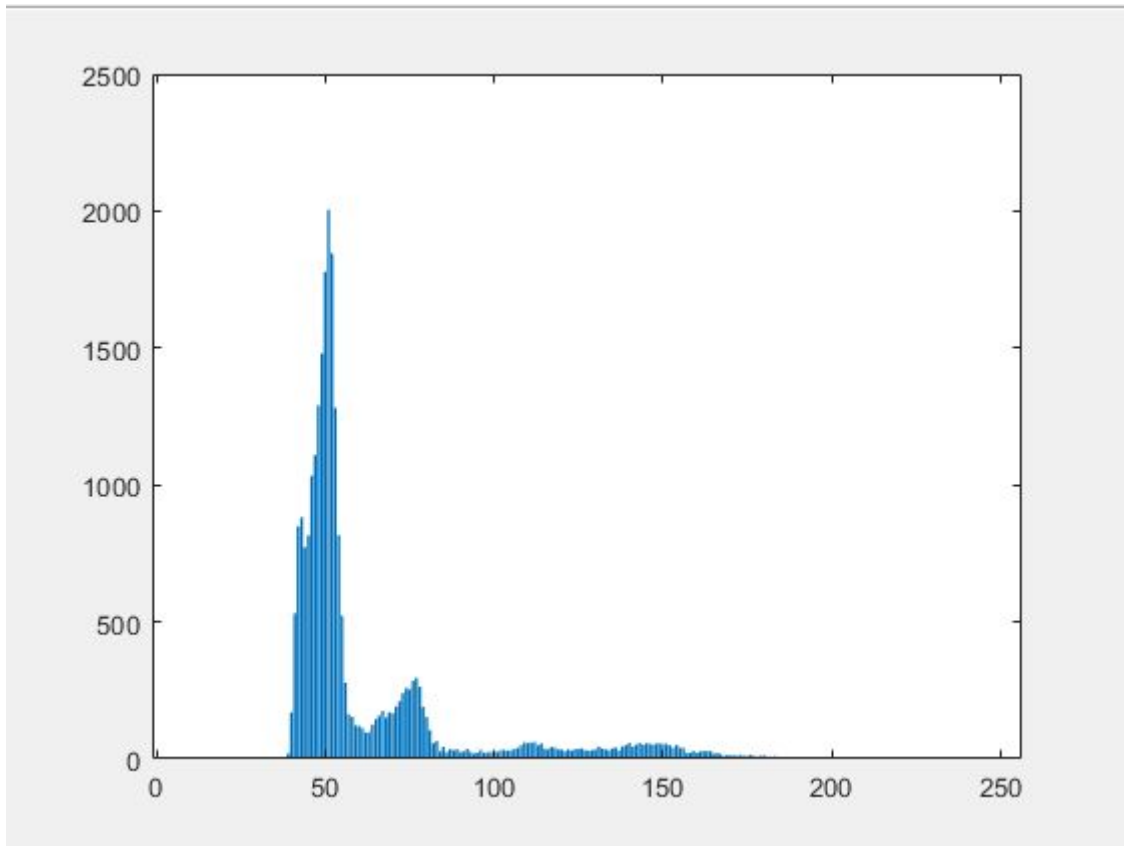
## Paso 22 //No coincide el valor

Utiliza el código de Huffman para codificar la matriz fotograma2. Escribe en el paso 22 de Practica02ApellidoNombre.m el código en Matlab para calcular el histograma de la imagen original, el tamaño de la matriz comprimida, incluida la cabecera, y su factor de compresión. Comprueba que la matriz original y la decodificación de la comprimida coinciden.

Incluye el histograma, el tamaño de la matriz original y el de la comprimida (con cabecera) en el paso 22 de Practica02ApellidoNombre.pdf. Escribe también el factor de compresión

[Escribe tus respuestas aquí](#)

1. el tamaño original de fotograma 2 es de 25344, mientras que el tamaño comprimido es 19245 y el factor de compresion es 1.316



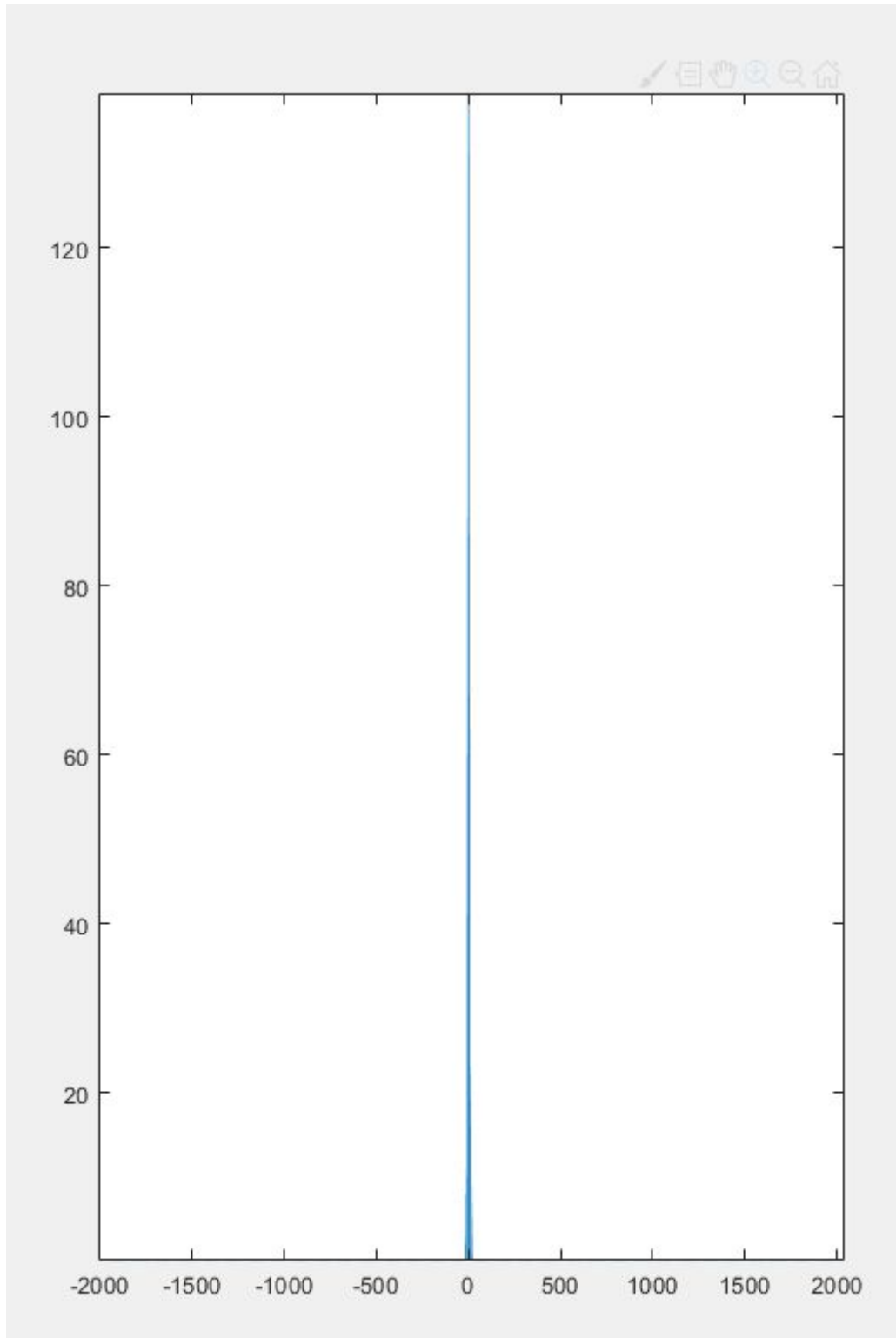
### Paso 23

Como puedes comprobar la nueva representación de la imagen ocupa menos espacio que la imagen original pero la reducción del tamaño no es grande. Escribe código en Matlab que dibuje el histograma de `fotograma2-fotograma1`, codifique la matriz diferencia usando Huffman, compruebe que `fotograma2` coincide con su reconstrucción y calcule el tamaño de `fotograma2-fotograma1` comprimido, incluida su cabecera. Incluye el código en el paso 23 de `Practica02ApellidoNombre.m`.

Incluye el histograma de `fotograma2-fotograma1`, y el tamaño de la matriz comprimida (con cabecera) en el paso 23 de `Practica02ApellidoNombre.pdf`. Escribe también el factor de compresión.

[Escribe tus respuestas aquí](#)

1. el tamaño de la matriz comprimida es 9152 y su factor de compresión es de 5.53846



## Paso 24

Compara en el paso 24 del fichero Practica02ApellidoNombre.pdf los tamaños de las matrices fotograma2 y fotograma2-fotograma1 comprimidos obtenidos en los pasos 22 y 23. ¿A qué se debe la mejora?

[Escribe tus respuestas aquí](#)

1. El tamaño comprimido de fotograma 2 es 19245 mientras que el de la diferencia es de 9152, esta disminución de tamaño se debe a que la diferencia entre fotograma 2 y fotograma 1 resulta valores muy cercanos al 0, debido a la gran dependencia entre los dos fotogramas. Esto hace que la compresión de las diferencias sea mejor en las diferencias.

## Paso 25

En el paso 9 nos dejamos pendiente detectar que habíamos decodificado toda la secuencia de 0s y 1s. Discute en el paso 25 de Practica02ApellidoNombre.pdf posibles soluciones a este problema.

[Escribe tu respuesta aquí](#)

1. Como hemos realizado en las anteriores prácticas, una de las soluciones consistiría en comprobar este resultado coincide con el original

## Paso 26

En el paso 9 definimos una cabecera. Se te ocurre alguna(s) mejora(s) que ahorre(n) espacio?. Incluye tu respuesta en el paso 26 de Practica02ApellidoNombre.pdf.

[Escribe tu respuesta aquí](#)

1. Otra de las formas de definir las cabeceras, en la que se ahorra mas espacio se basa en la representación canónica del árbol, teniendo solo que mandar el tamaño de manera ordenada.