UNIVERSIDAD DE BUENOS AIRES

Facultad de Ciencias Exactas y Naturales
Departamento de Computación
Métodos Numéricos

RECUPERATORIO DEL TRABAJO PRÁCTICO NÚMERO 2

Eliminación del ruido sobre señales unidimensionales y bidimensionales.

Alumnos:

Izcovich, Sabrina | sizcovich@gmail.com
Otero, Fernando | fergabot@gmail.com
Vita, Sebastián | sebastian_vita@yahoo.com.ar

Palabras Clave:

PSNR - Señal - Transformada DCT - Ruido

Resumen:

Este trabajo consiste en un análisis sobre diversas señales unidimensionales y bidimensionales con ruido cuyo objetivo es extraer conclusiones en cuanto a la calidad de la señal o imagen recuperada dependiendo de la estrategia utilizada. Para ello, debimos resolver sistemas lineales del tipo $y = Mx$ alterando y de forma tal que, al recuperar $M\tilde{x} = \tilde{y}$, éste resulte con menos ruido. Dicha experimentación fue realizada tanto en vectores como en matrices (extendiendo la transformada DCT a señales de dos dimensiones). Las conclusiones fueron extraídas a partir de estudios de gráficos y experimentaciones explicitadas a continuación.

Índice

1. Introducción teórica

Para la realización del estudio de señales, utilizamos los siguientes conceptos:

- **Transformada Discreta del Coseno (DCT):** Esta herramienta consiste, en el plano continuo, en representar una función en la base de funciones $\mathcal{B} = \{1, \cos(x), \cos(2x), ...\}$. En el plano discreto, en cambio, la DCT se corresponde a un cambio de base: cada una de las funciones de la base $\mathcal{B}$ se discretiza en ciertos puntos pasando a ser una base de vectores en $\mathbb{R}^n$ (con n la dimensión del vector o señal a transformar).
- **Frecuencia:** Es una magnitud que mide el número de repeticiones por unidad de tiempo de cualquier fenómeno o suceso periódico. En nuestro caso, dicha magnitud resultó esencial a la hora de analizar señales pues resultó ser el parámetro de comparación por excelencia.
- **Peak Signal-to-Noise Ratio (PSNR):** Es una métrica 'perceptual' utilizada como forma para medir la calidad visual de la señal reconstruida $\tilde{x}$. Nos da la forma de medir la calidad de una imagen perturbada a través de su definición:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_x^2}{ECM} \right)$$

  donde MAX_x define el rango máximo de la señal (en caso de entradas de 8 bits sin signo, sería 255) y ECM es el *error cuadrático medio*, definido como: $\frac{1}{n} \sum_i (x_i - \tilde{x}_i)^2$, donde n es la cantidad de elementos de la señal, x es la señal original y $\tilde{x}$ es la señal recuperada. Cuanto mayor es el PSNR, mayor es la calidad de la imagen.
- **Factorización LU:** La descomposición LU es una forma de factorización de una matriz como el producto de una matriz triangular inferior y una superior. Dicha descomposición se usa en el análisis numérico para resolver sistemas de ecuaciones más eficientes o encontrar las matrices inversas.

2. Desarrollo

Análisis previo: En primer lugar, es necesario explicitar las variaciones entre ruidos de imagen y de sonido:

- Las señales sonoras recibidas por los sistemas de captura presentan variaciones continuas que pueden ser de dos tipos:
  - Variaciones reales de la señal. En general, variaciones de baja frecuencia.
  - Variaciones debidas a interferencias (llamadas ruido). Generalmente, son de alta frecuencia y se eliminan mediante filtrado. Esto es lo que utilizaremos a lo largo de nuestra experimentación.

- Las señales bidimensionales, en el caso de las imágenes, tienen tipos de ruido que consisten en variaciones que afectan el brillo y/o el color de las mismas. Los tipos de ruido posibles pueden ser, por ejemplo, el Ruido Impulsional en el que ciertos píxeles toman el valor de blanco o negro de forma aleatoria sin tener relación con los píxeles circundantes.
  Otro tipo de ruido a considerar es el ruido Gaussiano. Dicho ruido sigue una distribución normal, llamada también distribución Gaussiana, afectando el valor de cada píxel que conforma la imagen.

En este trabajo práctico, el objetivo es eliminar el ruido sobre una señal ruidosa $x \in \mathbb{R}^n$. Para ello debimos considerar distintas estrategias tanto para agregar como para quitar el ruido de manera tal que resultara interesante cualquier tipo de análisis posterior a las evaluaciones. En primer lugar, pensamos, por ejemplo, que sería buena idea alterar el brillo de las imágenes (por lo tanto aumentar el valor de cada píxel). Al idealizar dicho método, nos percatamos de que la saturación podría llevar a la no recuperación de la imagen original dado que dicho efecto puede generar monotonía en la imagen una vez que se alcanza cierto valor de blanco. Luego, decidimos descartar esta idea.
Por otro lado, también en el caso de imágenes, se nos ocurrió agregar ruido con el método de ruido impulsivo (conocido también como 'sal y pimienta'). La dificultad se presentaba a la hora de recuperar la imagen ya que no resultaba evidente la manipulación de la misma considerando que los píxeles ruidosos no tiene relación alguna con los píxeles circundantes. Estudiamos la opción de crear un programa que analizara píxel a píxel y que fuera reemplazando cada valor por el promedio de los píxeles circundantes al píxel en cuestión. Dicho programa resultaba acertado para cumplir el objetivo pero las predicciones eran evidentes y no pensamos que fuera una manera apropiada para obtener resultados inesperados e interesantes de los que hubiera que explicar su procedencia.

Luego de dichos análisis, nos decidimos por los métodos que nos resultaron más interesantes para extraer conclusiones relevantes:

Agregado de ruido:
En primer lugar, decidimos agregar ruido de dos maneras distintas: con Ruido Aditivo y Ruido Multiplicativo. De esta manera, pensamos en tomar vectores/matrices *random* y sumarlas/-multiplicarlas a nuestras matrices a alterar. Debido a que decidimos evaluar nuestros resultados en función de la cantidad de ruido agregado, nos percatamos de que el hecho de sumar o multiplicar ruido no variaba los resultados dado que en cada caso nos encargamos de especificar el rango entre el que éste se encontraría. Luego, decidimos utilizar únicamente Ruido Aditivo para realizar nuestro análisis.
Con el fin de no perder la señal original para poder recuperarla a futuro, utilizamos 60 como *random* máximo (usado para sumar ruido). De esta manera, nos aseguramos de que las señales no se distorsionaran de tal manera que nos resultaran inservibles para su recuperación. Los ruidos generados se mantuvieron en los siguientes intervalos:

- [-1, 1]
  [-5, 5]
  [-15, 15]
  [-30, 30]
  [-50, 50]
  [-60, 60]

Una vez obtenidos los vectores/matrices, nos limitamos a realizar las operaciones básicas necesarias para sumarlas y obtener una nueva señal con ruido.

Eliminación del ruido:
Para ambos tipos de señal, tanto unidimensional como bidimensional, utilizamos el método del valor umbral. Dicho método consiste en la segmentación gráfica de ciertos valores que no aportan datos significativos y cuya eliminación no genera la pérdida de información de la señal.

En el caso de la señal unidimensional, dicha herramienta resulta ser el nivel de presión

sonora con la mínima intensidad necesaria para que un sujeto lo detecte. A partir de ella, eliminamos la señal (igualándola a 0) que supera su valor. De esta manera, se consigue eliminar ruido poco perceptible por el oído humano pero molesto a la hora de escuchar una señal sonora o de visualizar una imagen. Dicho método mantiene las frecuencias altas que resultan ser las más apreciables por el oído humano.

Por otro lado, en el caso de la señal bidimensional, la idea del umbral se basa en distinguir en una imagen los objetos del fondo de los objetos del primer plano. Dichos elementos de la imagen se distinguen en el gráfico de la señal a través de los picos que presenta. Al segmentar dicho gráfico, los cambios no deberían ser demasiado significativos dado que lo que genera el umbral es la disminución del valor de ciertos píxeles. Esto afecta a la calidad de la imagen sin la pérdida del objeto ni su entorno.

Teniendo en cuenta la frecuencia de la señal, podemos afirmar que el ruido es oscilatorio y se relaciona con los valores más altos de la transformada. Por lo tanto, cuando borro el ruido mirando la frecuencia asumo que la señal original no tiene muchos saltos y que el ruido se acierta con saltos muy chiquitos por lo que conviene quedarse con los grandes que me van a dar la intensidad. La señal queda casi igual pues el ruido queda captado por los grandes valores.
Los valores elegidos para el umbral fueron considerados manualmente de acuerdo al gráfico de cada señal particular.

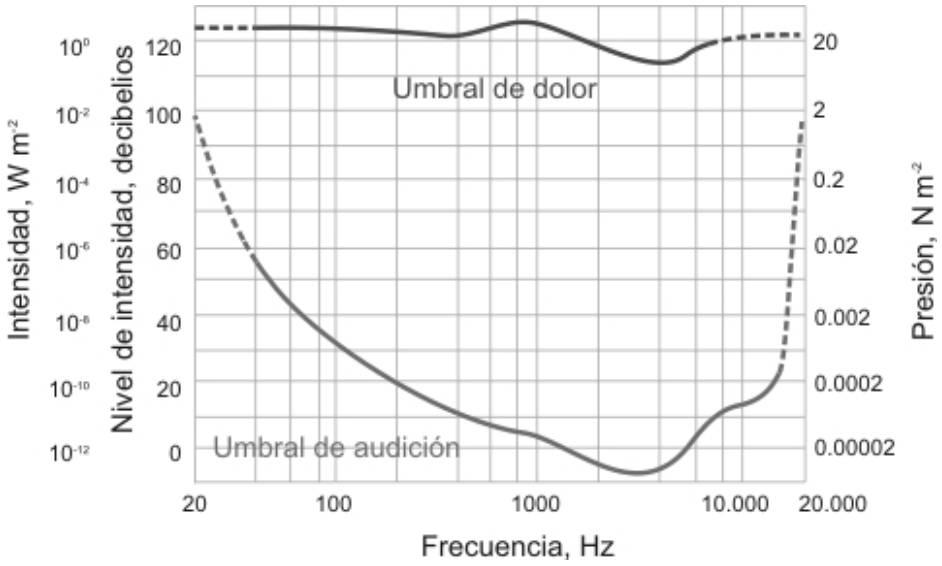

Figura 1: Umbral aplicado a ondas sonoras con respecto a la presión y nivel de intensidad.

Por otro lado, la otra manera de eliminar el ruido que tuvimos en cuenta fue una idea basada en la **Eliminación de frecuencias altas (interferencias) mediante filtrado de la señal**. Tal como lo explicita el nombre, procedimos a disminuir las frecuencias altas de la señal. Dicho método se basa en los ecualizadores que filtran, atenúan o eliminan frecuencias que molestan, ruidos o interferencias que se mezclan con el sonido. Debido a que, por lo general, los problemas ocurren en un rango determinado de frecuencias, este tipo de método nos pareció adecuado para quitar el ruido. Del mismo modo, pensamos que podría ser interesante aplicarlo a imágenes pues nos interesó experimentar en qué afectaría dicha aplicación en una señal bidimensional.

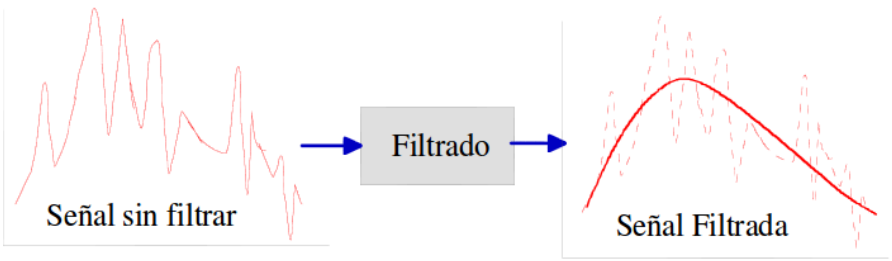

Figura 2: Eliminación de frecuencias altas (interferencias) mediante filtrado de la señal.

Un problema hallado en dicho procedimiento fue que, como consecuencia del muestreo de un sistema pueden crearse nuevas frecuencias inexistentes en la señal original, es decir frecuencias que no corresponden ni al ruido ni a la propia señal.

Considerando únicamente los métodos de eliminación del ruido, los resultados esperados al recuperar las señales eran gráficos cercanos a los originales dado que las alteraciones realizadas al agregarlo no debían otorgar diferencias extremas en los valores dados. Por lo tanto, el ruido esperado debía ser fácilmente perceptible pero no debía cambiar la señal de manera rotunda simplificando la eliminación del mismo.

Nuestro resultado esperado al recuperar la señal con el Método del Valor Umbral era que éste fuera muy similar a la señal original dado que esto fue lo ocurrido en los ejemplos probados en clase. En el caso de la eliminación de frecuencias altas, nuestro resultado esperado fue la recuperación de señales ligeramente más monótonas que las originales dado que los picos de mayor frecuencia habían sido reducidos. De todos modos, lo que esperamos encontrar fueron objetos más nítidos (en el caso de la imagen) y sonidos más puros dado que las frecuencias mayores son las más notorias por lo que el ruido es más evidente en ellas, buscando mimetizarlo con la imagen original.

Implementación: El paso siguiente consistió en programar en C++ los métodos de eliminación y agregado de ruido utilizadas, la función de la Transformada Discreta del Coseno, el PSNR y el programa para realizar las operaciones necesarias dado un archivo de texto con la cantidad de datos en la primera línea y los propios datos ASCII en la segunda. Del mismo modo, implementamos la factorización LU que nos permite reconstruir la señal de manera eficiente. Para implementar con matrices decidimos utilizar punteros a punteros dado que creamos nuestra matriz de forma dinámica y C++ no acepta que una función reciba una matriz con dimensiones desconocidas.

El *main.cpp* consistió en un menú necesario para procesar los archivos de texto que serían, más tarde, utilizados por las funciones para resolver la DCT y proseguir con la modificación de la señal. Para realizarlo, nos limitamos a utilizar herramientas conocidas (if, while, etc.) y $printf$'s.
En *tcd.cpp*, escribimos las fórmulas utilizadas a lo largo del trabajo práctico. Tanto la función principal para hallar la DCT, el PSNR, como las funciones utilizadas para agregar y eliminar ruido como también las que nos permitieron volver a la señal original recuperada. Para que dichas fórmulas fueran legibles, utilizamos los vectores que se descomponen de la matriz $\mathcal{M}$ para realizar las operaciones necesarias y luego terminar por unirlas correctamente.

Modo de uso del programa: Nuestro programa corre en Windows 7 y Ubuntu. Para compilarlo, recomendamos usar g++. Para ejecutarlo es necesario compilar *tcd.cpp* y luego, *main.cpp*. Luego, se debe abrir el ejecutable del main.
Una vez abierta la interfaz del programa, se debe añadir el archivo de texto junto con su extensión cuyos valores quieran ser procesados. Luego, debe ser seleccionado el tipo de señal a transformar entre unidimensional y bidimensional. Posteriormente, debe elegirse entre agregar ruido sumando random o agregarlo multiplicando random. Por último, debe seleccionarse la función utilizada para quitar el ruido que utilizará la factorización LU para recuperar de manera aproximada la señal original.

Recuperación de resultados: Para verificar la correctitud de nuestro programa comparamos los resultados obtenidos con la solución del mismo proceso realizado en Matlab. De esta manera, corroboramos los algoritmos realizados y descartamos experimentos innecesarios. Al realizar varias iteraciones llegamos a la conclusión de que los resultados de nuestro programa resultaban satisfactorios para la mayoría de las pruebas realizadas.

3. Resultados

3.1. Señales unidimensionales

Para realizar las experimentaciones, utilizamos las señales otorgadas por la cátedra 'dopp1024.txt', 'g450.txt' y 'ramp1234.txt'. En primer lugar, graficamos la Transformada Coseno Discreta de los datos en ASCII de dichos archivos junto con el ruido a agregar. En el plano unidimensional, la aplicación de la Transformada puede ser representada como la multiplicación del vector de datos por la matriz de transformación. En cambio, en el plano bidimensional, los cálculos matriciales dificultan las operaciones. Para una simple visualización de los resultados, elegimos plotear las señales unidimensionales. En cambio, en el caso de las señales bidimensionales, preferimos mostrar su imagen característica para que los resultados obtenidos fueran fácilmente visualizables. Si bien nuestras pruebas fueron realizadas con los 6 intervalos de ruido mencionados anteriormente, elegimos utilizar el ruido en [-30,30] para realizar los gráficos dado que es el ruido medio y los demás se comportan de una manera similar dentro de sus propios rangos. Los gráficos que siguen representan a las dct de las señales unidimensionales originales en el plano de las frecuencias:

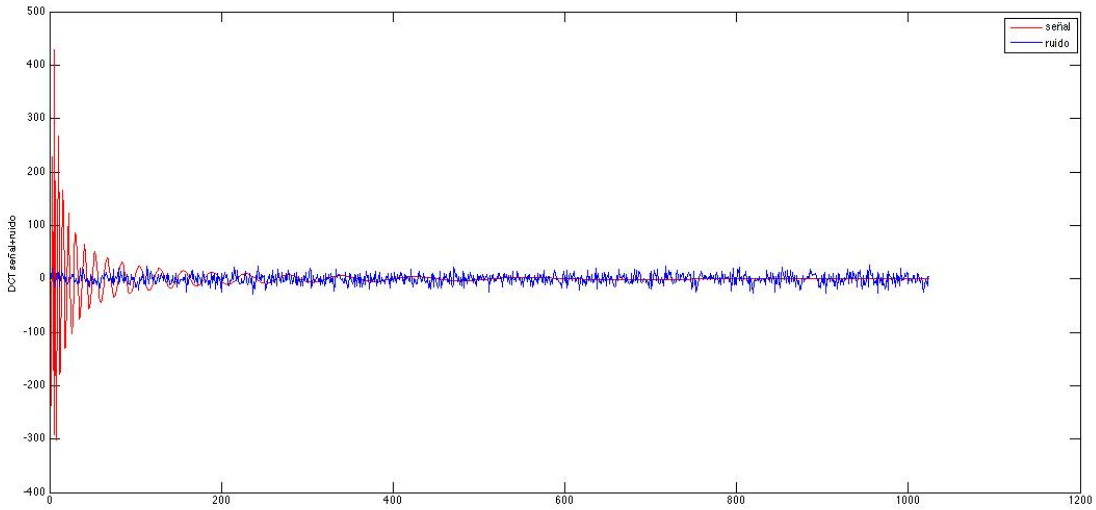

Figura 3: DCT de la señal dopp con 1024 datos y el ruido a agregar en el intervalo [-30, 30].

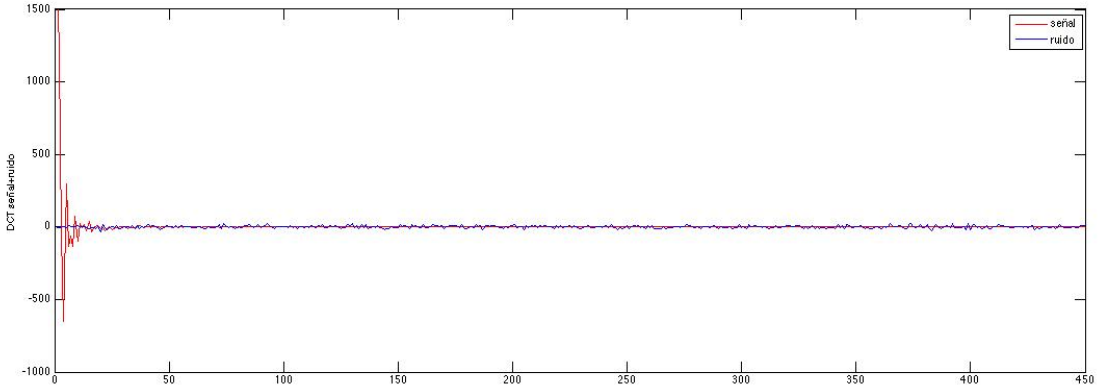

Figura 4: DCT de la señal g con 450 datos y el ruido a agregar en el intervalo [-30, 30].

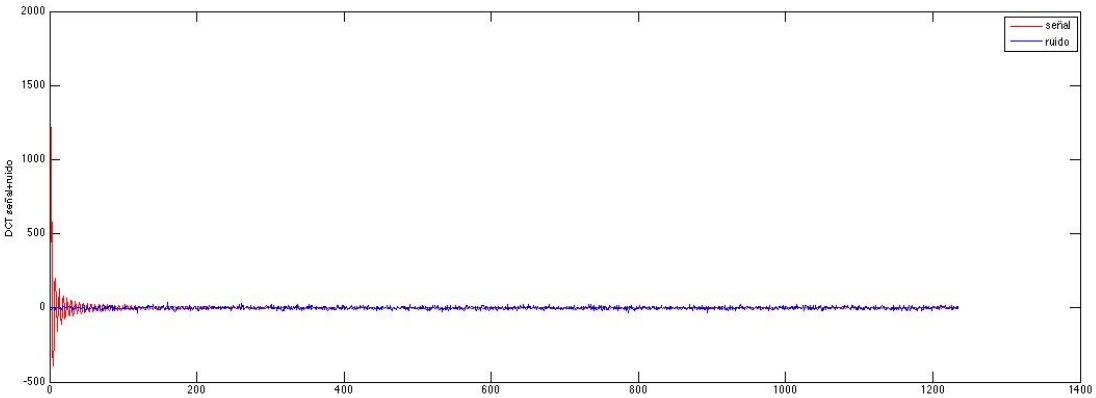

Figura 5: DCT de la señal ramp con 1234 datos y el ruido a agregar en el intervalo [-30, 30].

A continuación, proseguimos agregando el ruido. Para realizar ésto, decidimos sumarle los rangos de ruido agregados posteriormente a la señal original. Luego, calculamos la Transformada Coseno Discreta de las señales sumadas. Dado que los gráficos son semejantes a diferencia del alcance del ruido, preferimos dar uno a modo de prueba siendo el resto predecibles. Al igual que en el caso anterior, los gráficos son predecibles por lo que decidimos insertar uno a modo de referencia. Éstos resultaron de la siguiente manera:

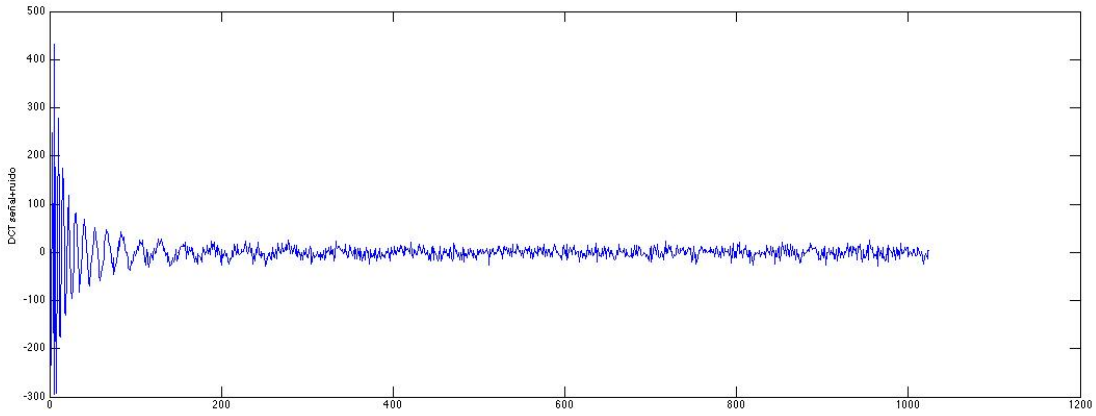

Figura 6: DCT de la señal original de dopp con 1024 datos + ruido [-30, 30].

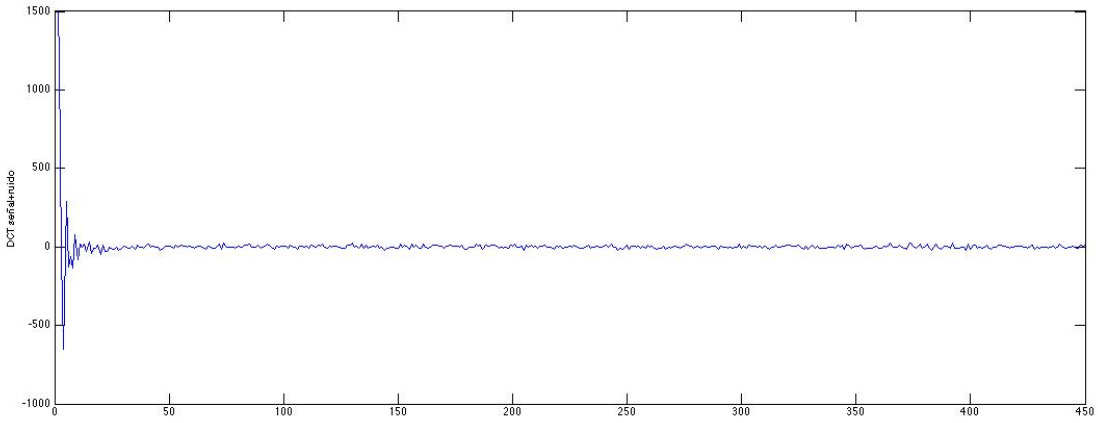

Figura 7: DCT de la señal original de g con 450 datos + ruido [-30, 30].

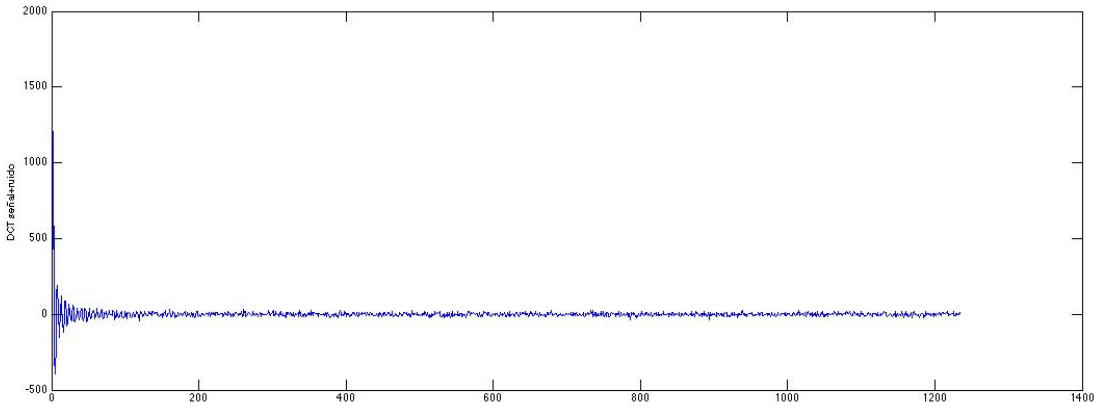

Figura 8: DCT de la señal original de ramp con 1234 datos + ruido [-30, 30].

Los gráficos fueron alterados en cuanto a longitud para que las frecuencias interesantes fueran visualizables.

A continuación, le aplicamos la primer función (que aplica el Método del Valor Umbral) a las señales de DCT con ruido agregado sumando random.
Luego de diversas pruebas con distintos valores de umbral, nos decidimos por el umbral más coherente para analizar. Por lo tanto, procesamos dichas señales igualando a cero los valores cuyo valor absoluto de su magnitud fuera menor al umbral que fijamos en 5000. En los siguientes gráficos, se puede observar que los valores absolutos mayores a dichas cotas se conservan tal como en la dct de la señal original, en cambio, los valores absolutos menores tienen ahora el valor de 0.

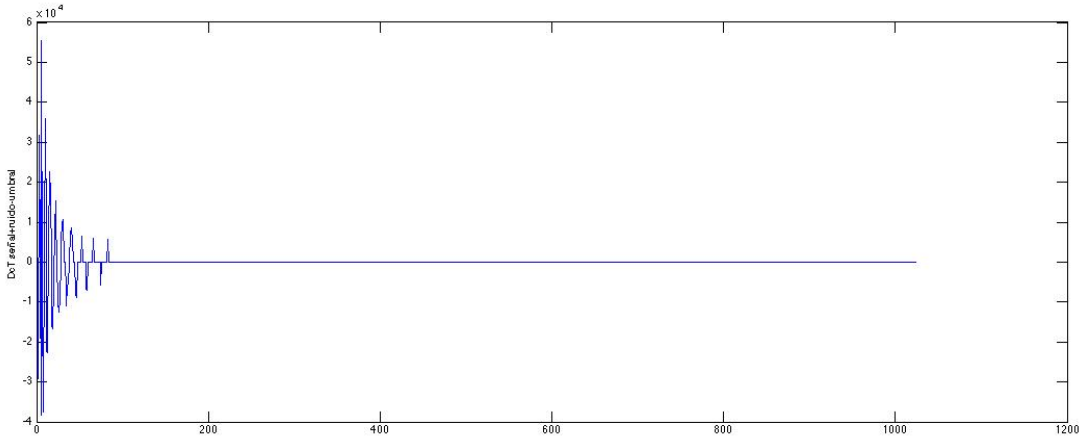

Figura 9: DCT de dopp1024 con umbral en 5000.

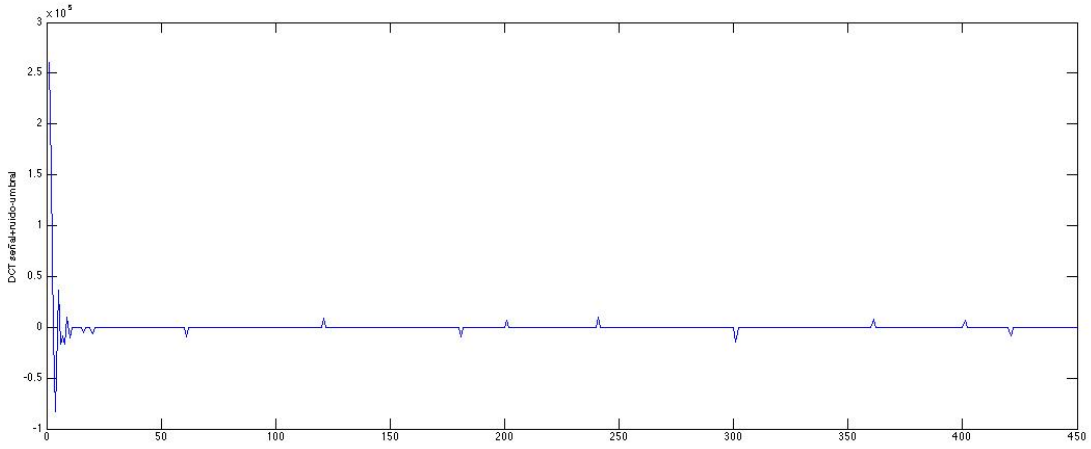

Figura 10: DCT de g450 con umbral en 5000.

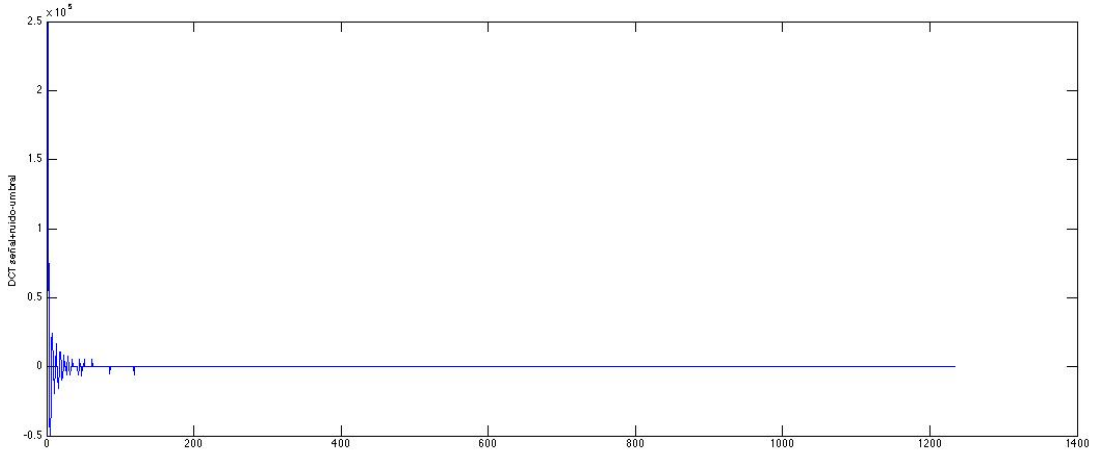

Figura 11: DCT de ramp1234 con umbral en 5000.

Lo que sigue consistió en aplicar la M correspondiente (para obtener $M * \tilde{x} = \tilde{y}$) para la vuelta a la señal original. En los gráficos siguientes, se puede ver cómo resultaron las señales en comparación con la señal original para cada rango de ruido.

- **dopp1024**

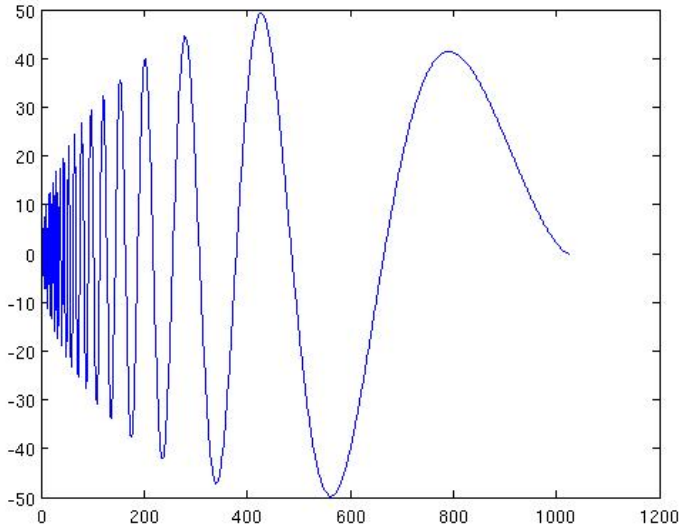

Figura 12: dopp1024 original.

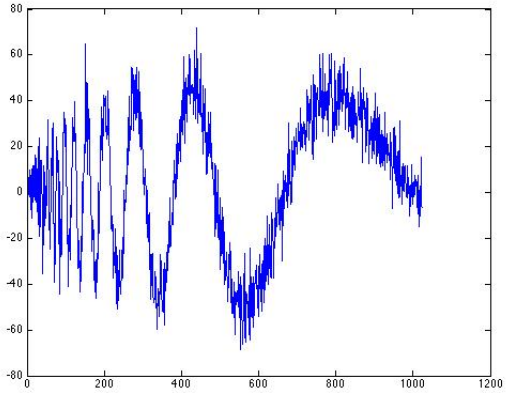

Figura 13: DCT de dopp1024 recuperada con un umbral de 1000

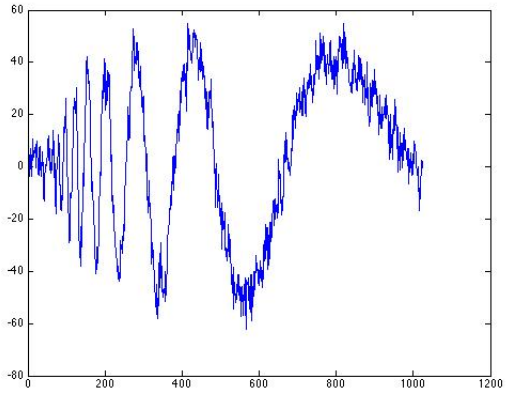

Figura 14: DCT de dopp1024 recuperada con un umbral de 3000

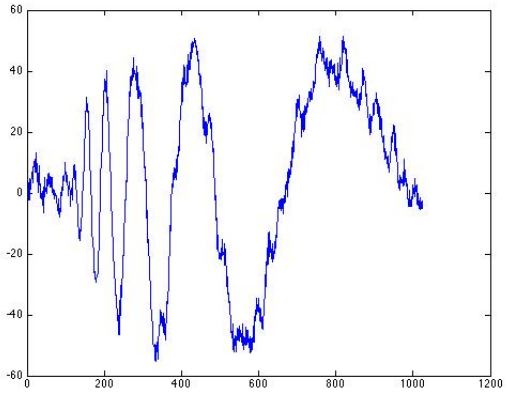

Figura 15: DCT de dopp1024 recuperada con un umbral de 5000

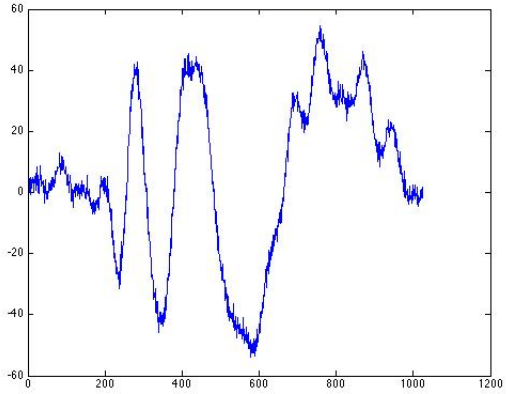

Figura 16: DCT de dopp1024 recuperada con un umbral de 10000

g450

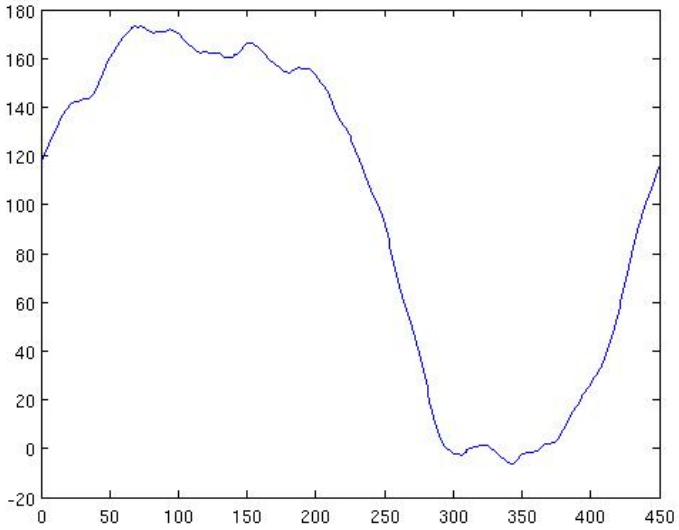

Figura 17: g450 original.

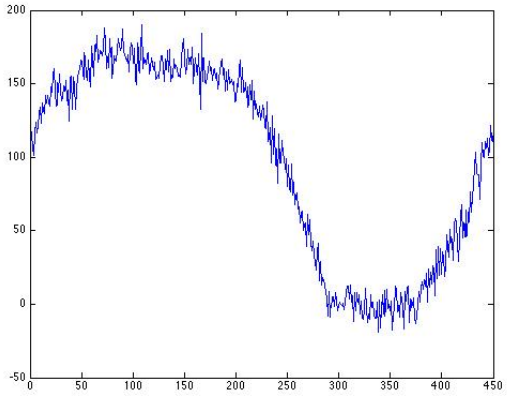

Figura 18: DCT de g450 recuperada con un umbral de 1000

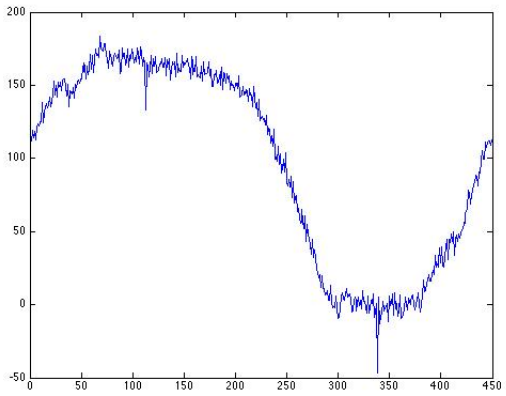

Figura 19: DCT de g450 recuperada con un umbral de 3000

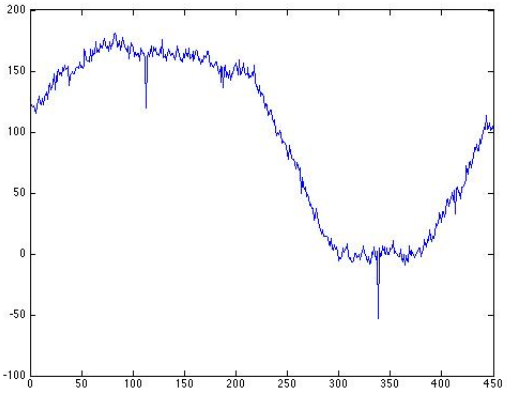

Figura 20: DCT de g450 recuperada con un umbral de 5000

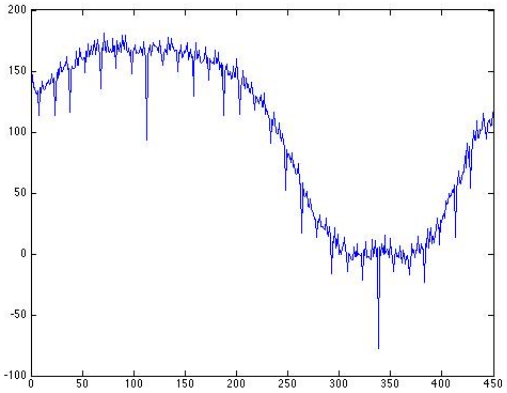

Figura 21: DCT de g450 recuperada con un umbral de 10000

ramp1234

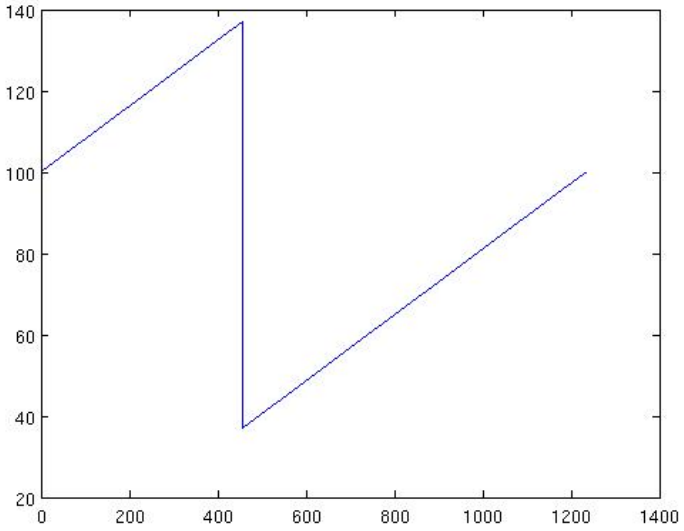

Figura 22: ramp1234 original.

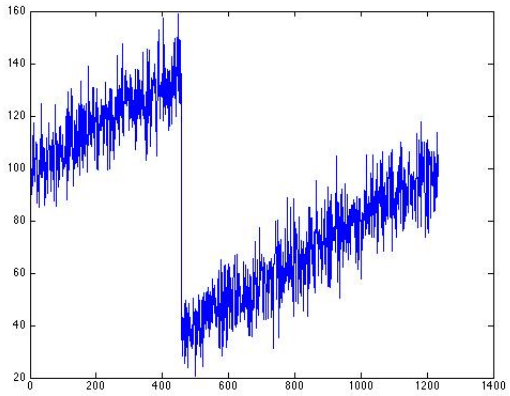

Figura 23: DCT de ramp1234 recuperada con un umbral de 1000

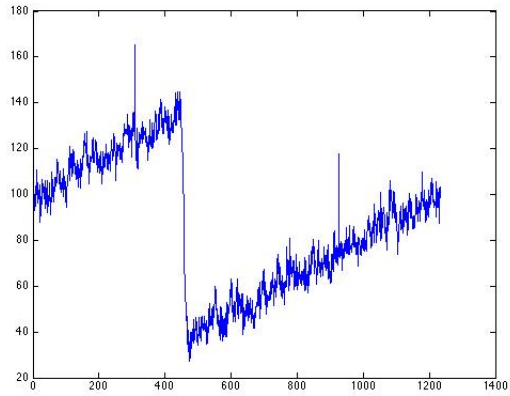

Figura 24: DCT de ramp1234 recuperada con un umbral de 3000

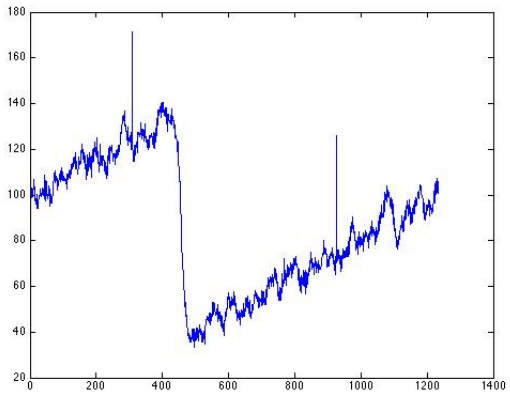

Figura 25: DCT de ramp1234 recuperada con un umbral de 5000

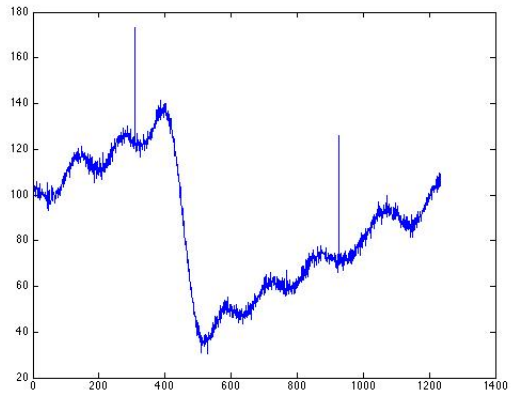

Figura 26: DCT de ramp1234 recuperada con un umbral de 10000

Luego, recuperamos los valores de los PSNR de acuerdo a la cantidad de ruido agregado para lograr obtener una relación lineal entre la cantidad de ruido agregado y la cantidad de ruido eliminado. Debido a que el PSNR es una métrica perceptual que divide el error cuadrático medio, un resultado grande (≥ 20) expone que las señales evaluadas se parecen. En cambio, si el resultado permanece entre números pequeños (menores que 5) nos dice que las señales no son semejantes. Los resultados obtenidos fueron los siguientes:

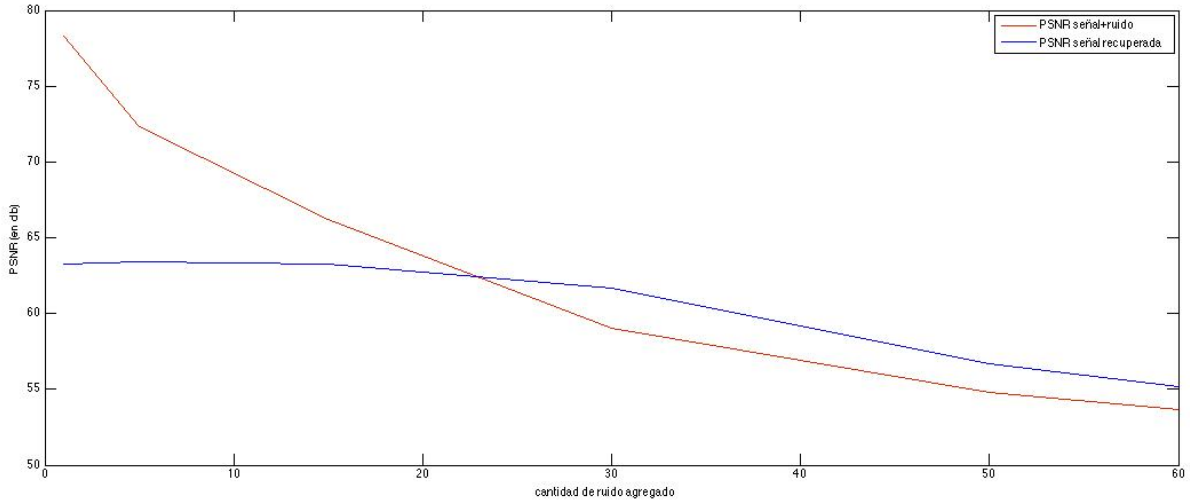

Figura 27: PSNR de la señal dopp1024+ruido y PSNR de la señal recuperada respecto de la cantidad de ruido agregado.

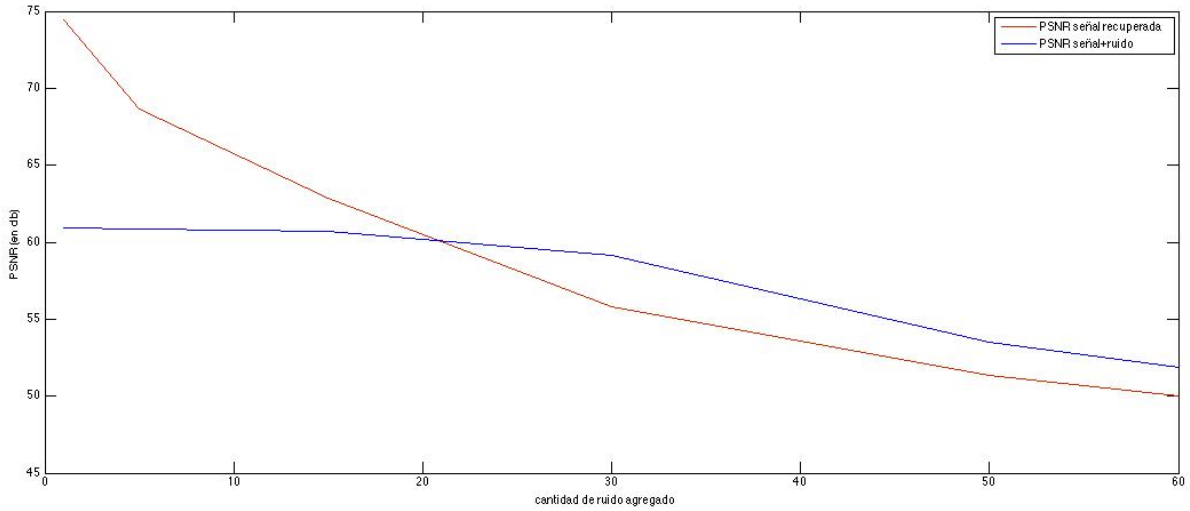

Figura 28: PSNR de la señal g450+ruido y PSNR de la señal recuperada respecto de la cantidad de ruido agregado.

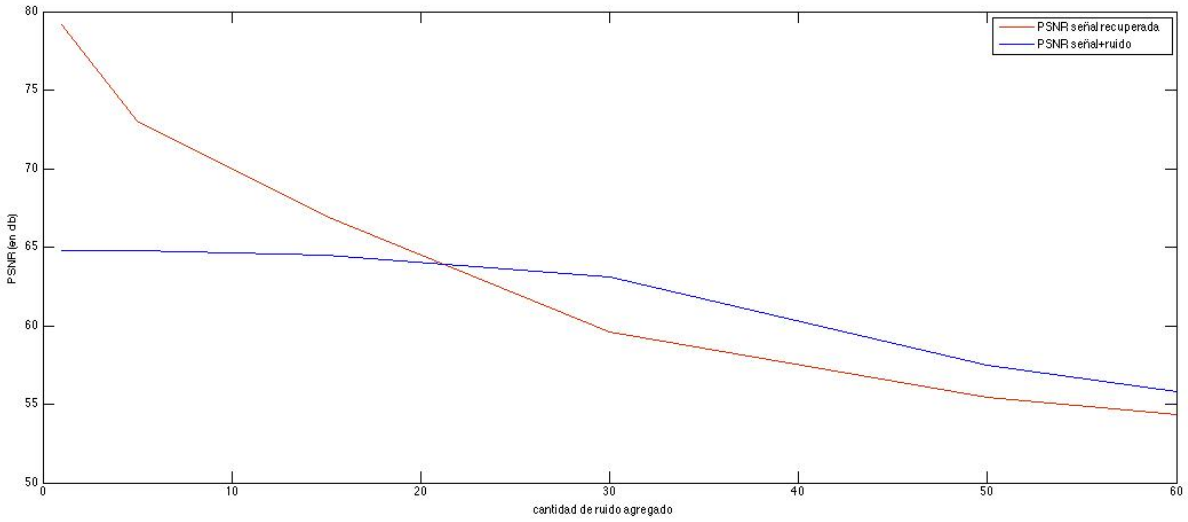

Figura 29: PSNR de la señal ramp1234+ruido y PSNR de la señal recuperada respecto de la cantidad de ruido agregado.

Luego, procedimos aplicando el segundo método 'Eliminación de frecuencias altas mediante filtrado de la señal'. En este caso, debimos evaluar diversos errores a partir de los cuales se reduciría parte de la frecuencia. Las pruebas realizadas consistieron en alternar la cota a partir de la que se reduciría la señal como también el valor que se le restaría a los valores de ésta que superaran dicha cota. Luego de distintas pruebas, notamos que la variación entre la señal con ruido y la señal recuperada no era realmente notoria en la mayoría de los casos. Los resultados obtenidos fueron los siguientes:

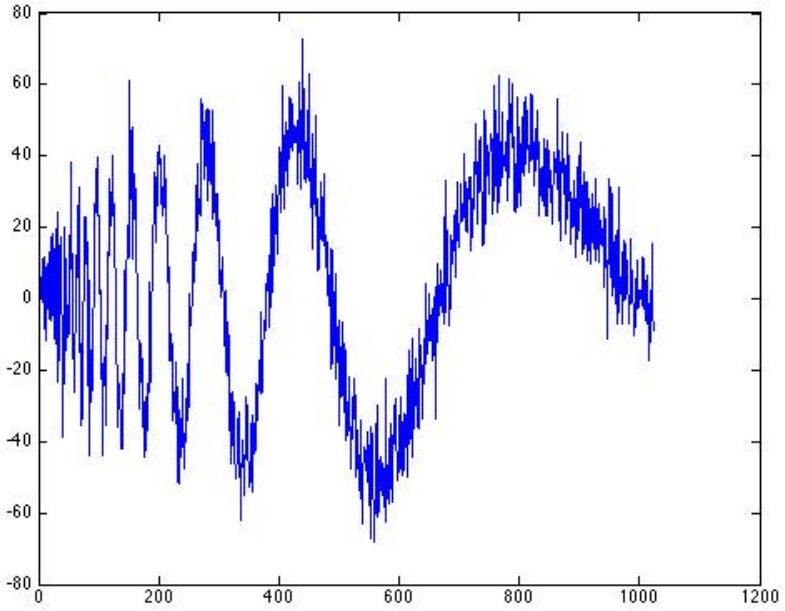

Figura 30: Señal dopp1024 recuperada utilizando una cota de 10000 y reduciendo las frecuencias de 50.

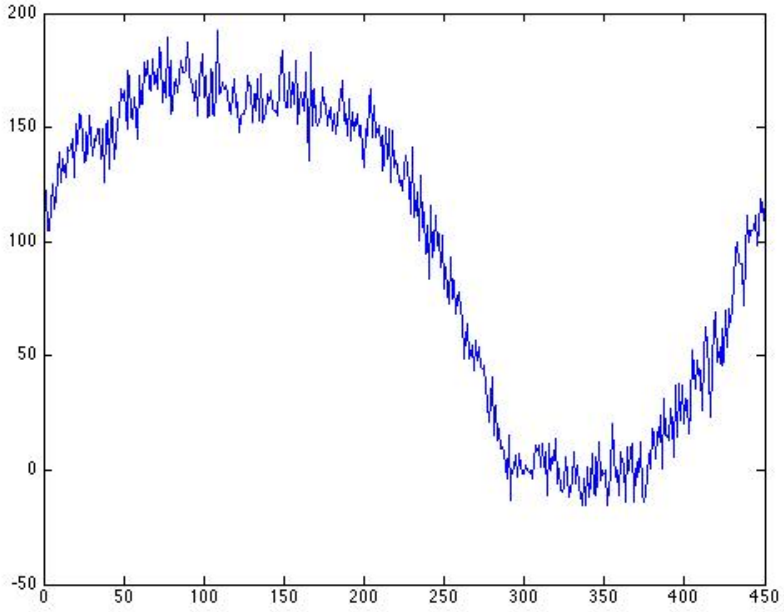

Figura 31: Señal g450 recuperada utilizando una cota de 10000 y reduciendo las frecuencias de 50.

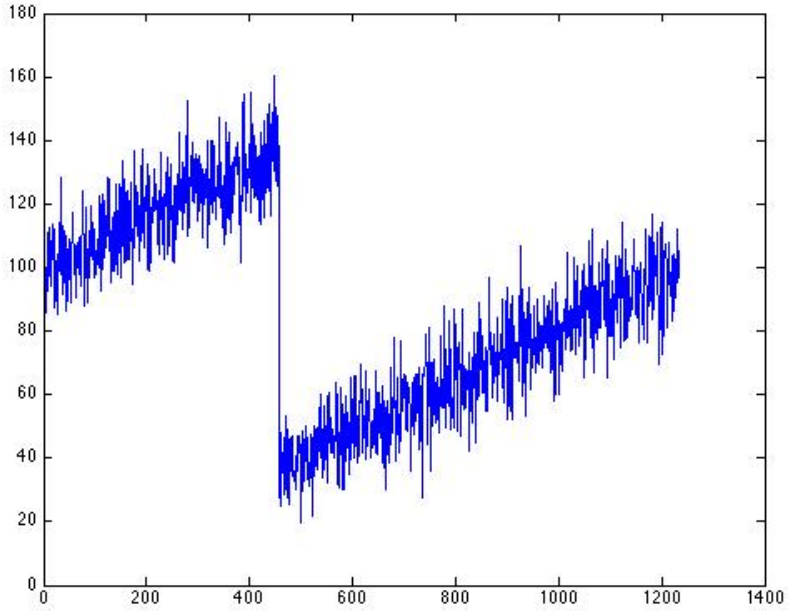

Figura 32: Señal ramp1234 recuperada utilizando una cota de 10000 y reduciendo las frecuencias de 50.

Para poder sacar conclusiones de los resultados obtenidos, decidimos realizar un gráfico en el que se pudiera visualizar la variación del PSNR en función de la cantidad de ruido agregado a la señal:

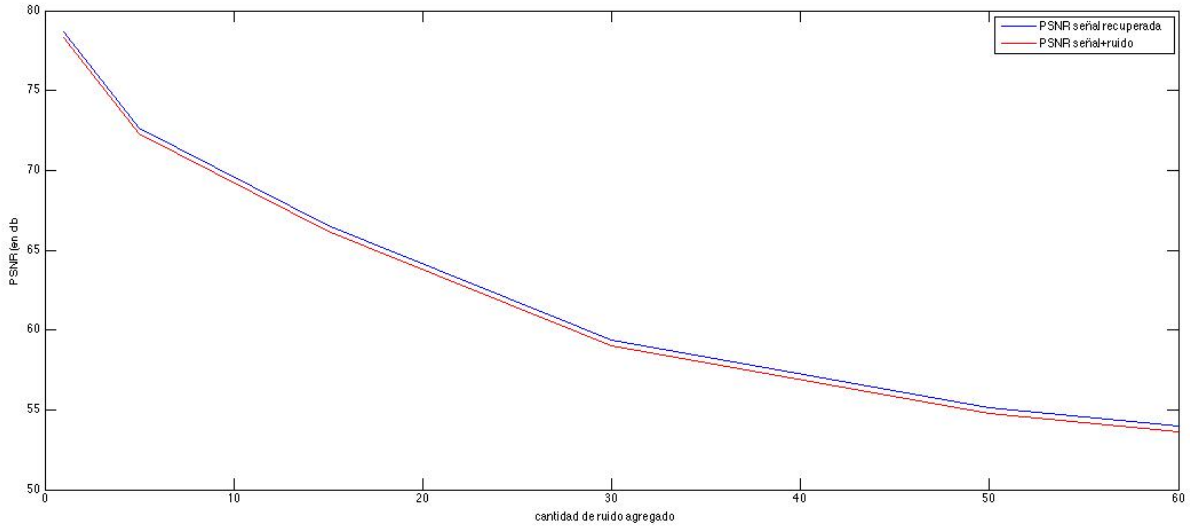

Figura 33: PSNR de la señal dopp1024+ruido y PSNR de la señal recuperada respecto de la cantidad de ruido agregado.

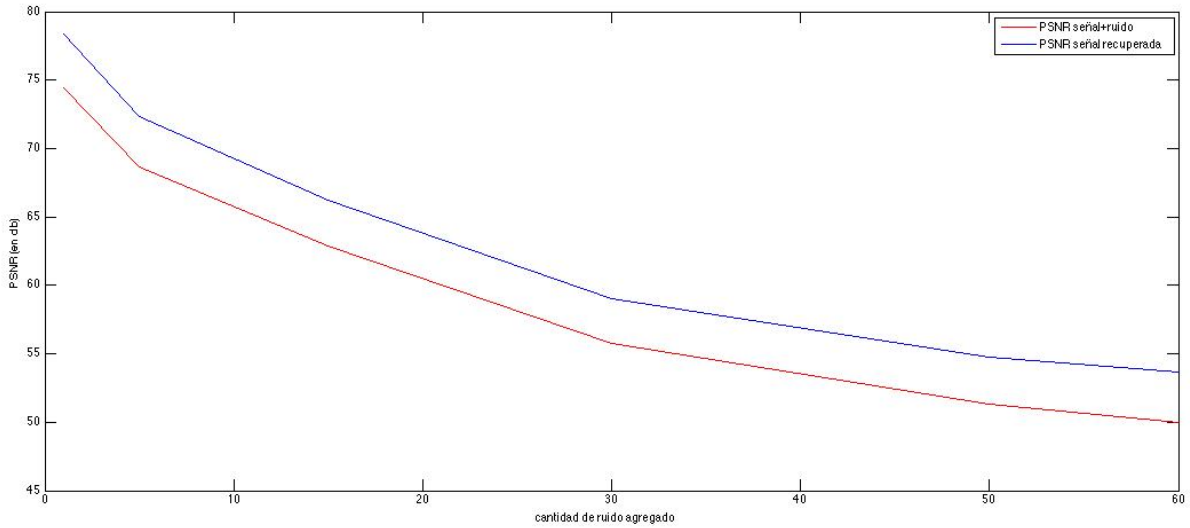

Figura 34: PSNR de la señal g450+ruido y PSNR de la señal recuperada respecto de la cantidad de ruido agregado.

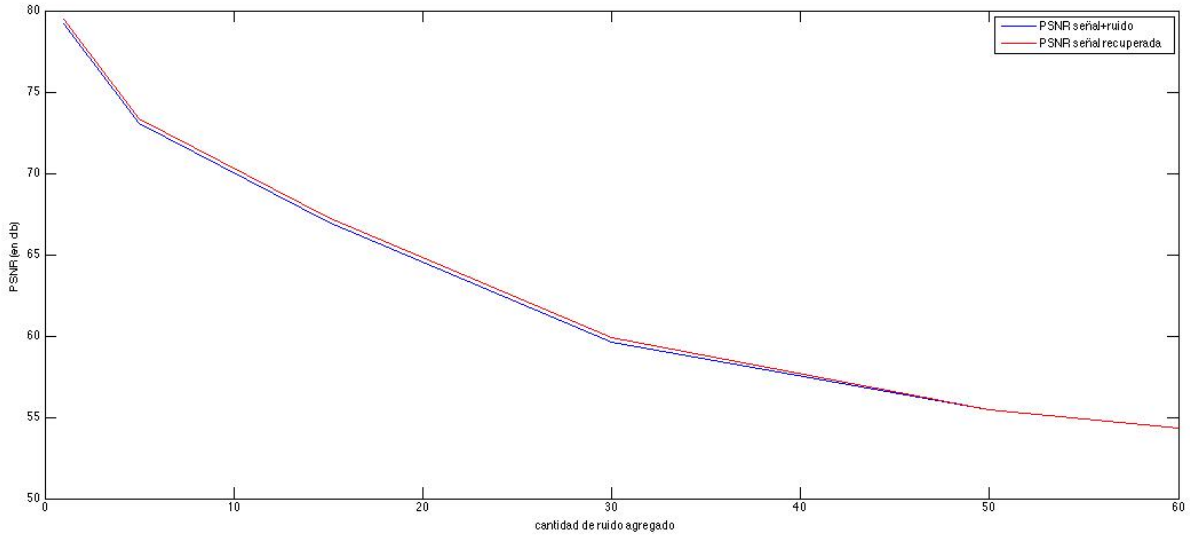

Figura 35: PSNR de la señal ramp1234+ruido y PSNR de la señal recuperada respecto de la cantidad de ruido agregado.

Se puede observar que los PSNR demuestran que el resultado es levemente mejor sin importar la cantidad de ruido agregado. Esto se debe a que al reducir la frecuencia de los valores más presentes en la señal (que es donde más ruido se preserva) la cantidad de ruido se aminora sin importar cuánto sea éste. Si bien este método no resulta eficaz, resulta interesante que, en la gran mayoría de los casos, siempre hay una leve mejoría.

3.2. Señales bidimensionales

Para las señales bidimensionales, utilizamos los mismos métodos para quitar el ruido. Primero, nos enfrentamos con la decisión de elegir un umbral que nos diera buenos resultados en la mayor parte de los casos. Para ello probamos, para una abundante combinación de cotas y cantidad de ruido, una enorme cantidad de experiencias. De este modo, se tuvieron en cuenta diferentes ruidos para una misma cota y se realizaron diversas variaciones de cada rango generado aleatoriamente.
El algoritmo utilizado es de la siguiente forma:

```
1: for cada cota do
2:     for cada ruido do
3:         resuelvo sistema con DCT
4:         calculoPSNR                ▷ Si el PSNR de la imagen recuperada es mayor que 20 lo guardo
5:     end for
6:     promedio los resultados en los que la imagen mejoró
7: end for
8: comparo los promedios y me quedo con la cota que mejor dio
```

Las señales bidimensionales utilizadas para la experimentación fueron las siguientes:

Para realizar las pruebas con imágenes, utilizamos los métodos de Umbral y Eliminación de Frecuencias Altas utilizando el mismo procedimiento que para señales unidimensionales alterando el código para que la matriz entera fuera procesada.
Realizamos nuestras pruebas con los umbrales 5000, 10000 y 500000.

Figura 36: Imagen 1.

Figura 37: Imagen 2.

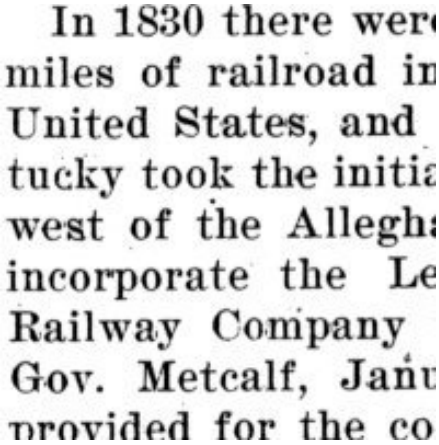

Figura 38: Imagen 3.

Los resultados de las pruebas realizadas con el Umbral son las siguientes imágenes:

Figura 39: Imagen 1 con ruido de [-60, 60], umbral de 5.000.

Figura 40: Imagen 1 recuperada.

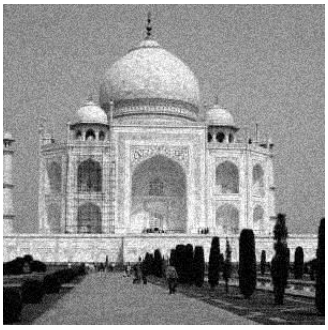

Figura 41: Imagen 2 con ruido de [-25, 25], umbral de 500.000.

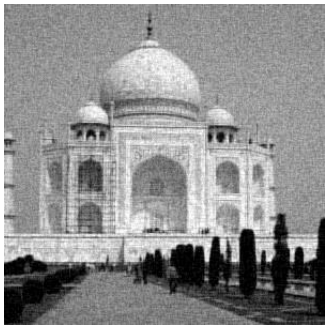

Figura 42: Imagen 2 recuperada.

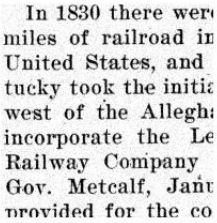
In 1830 there were
miles of railroad in
United States, and
tucky took the initia
west of the Allegha
incorporate the Le
Railway Company
Gov. Metcalf, Janu
provided for the co

Figura 43: Imagen 3 con ruido de [-60, 60], umbral de 5.000.

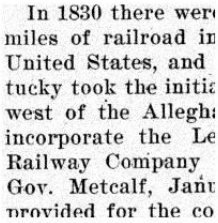
In 1830 there were
miles of railroad in
United States, and
tucky took the initia
west of the Allegha
incorporate the Le
Railway Company
Gov. Metcalf, Janu
provided for the co

Figura 44: Imagen 3 recuperada.

Los resultados de las pruebas realizadas con el método Eliminación de frecuencias altas son las siguientes imágenes:

Figura 45: Imagen 1 con ruido de [-60, 60], Eliminación de frecuencias altas de 10.000.

Figura 46: Imagen 1 recuperada.

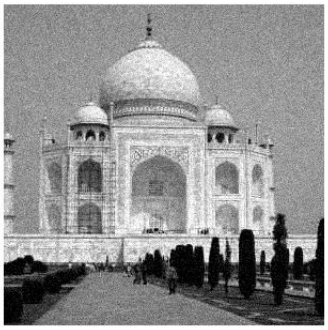

Figura 47: Imagen 2 con ruido de [-60, 60], Eliminación de frecuencias altas de 10.000.

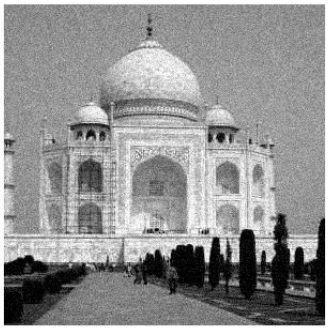

Figura 48: Imagen 2 recuperada.

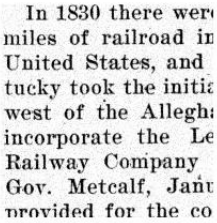

Figura 49: Imagen 2 con ruido de [-60, 60], Eliminación de frecuencias altas de 10.000.

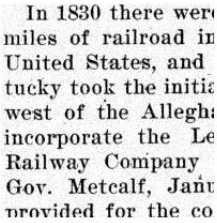

Figura 50: Imagen 2 recuperada.

Por último, calculamos los distintos PSNR de cada imagen con ruido y su recuperada. Para exhibir los valores de manera clara, decidimos realizar gráficos en los que se muestra la diferencia entre el PSNR de la imagen+ruido y el PSNR de la recuperada. Los resultados fueron los siguientes:

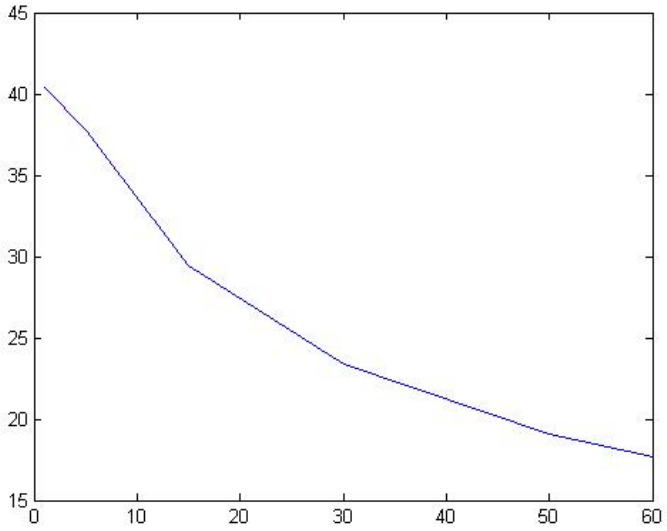

Figura 51: PSNRs para una cota de 100.000.

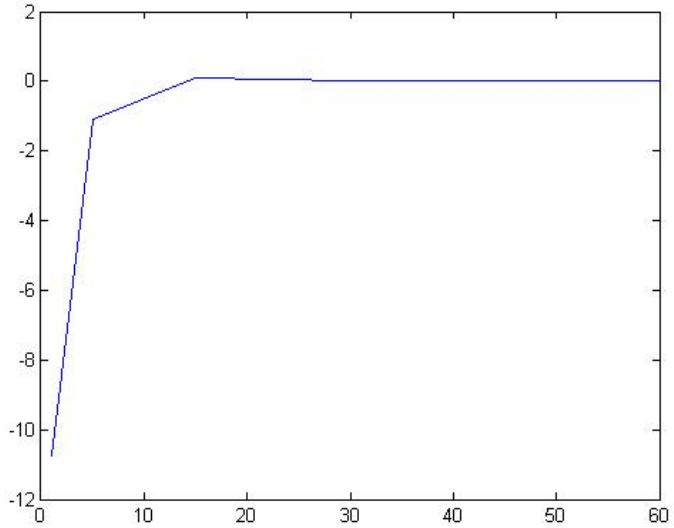

Figura 52: Diferencias de PSNR's para una cota de 100.000.

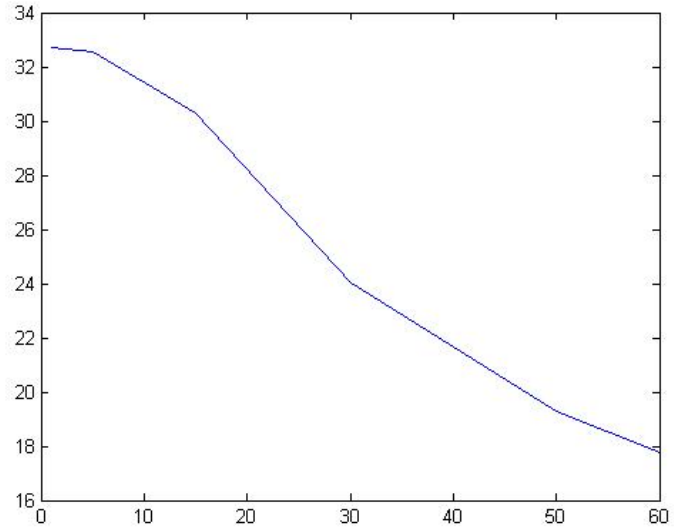

Figura 53: PSNR's para una cota de 300.000.

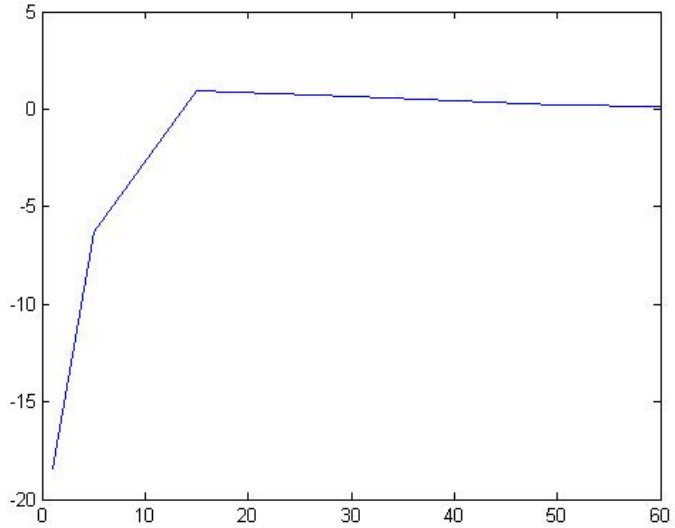

Figura 54: Diferencias de PSNR's para una cota de 300.000.

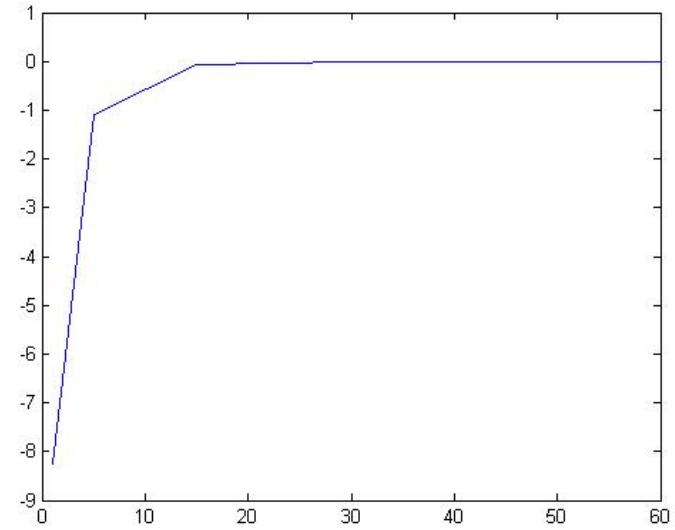

Figura 55: Diferencias de PSNRs para una cota de 100.000.

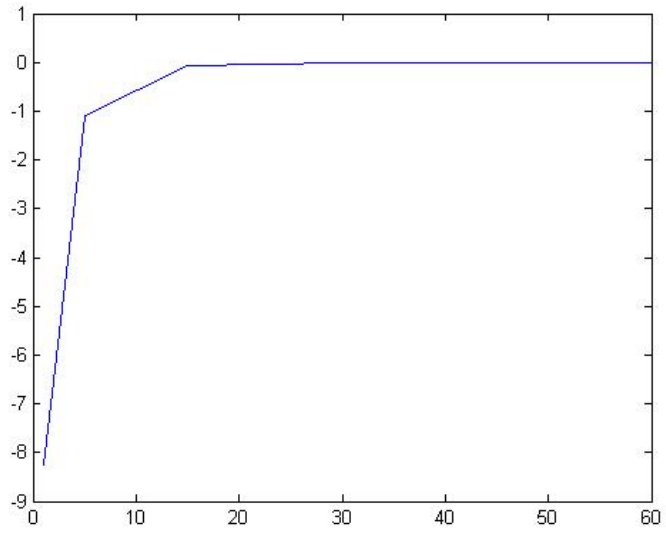

Figura 56: Diferencias de PSNR's para una cota de 100.000.

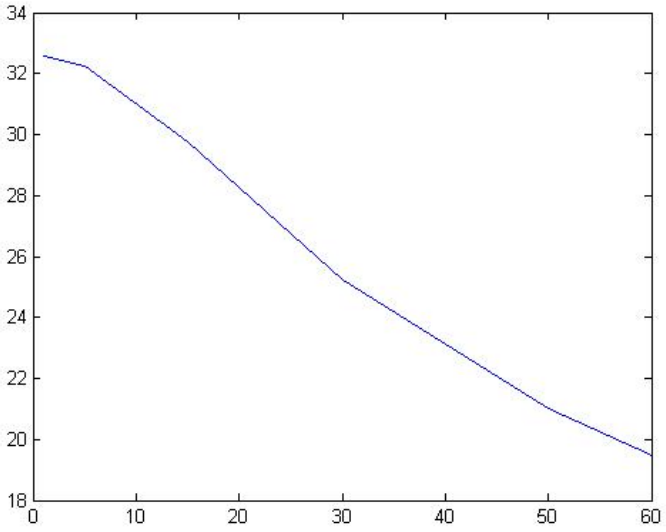

Figura 57: PSNR's para una cota de 300.000.

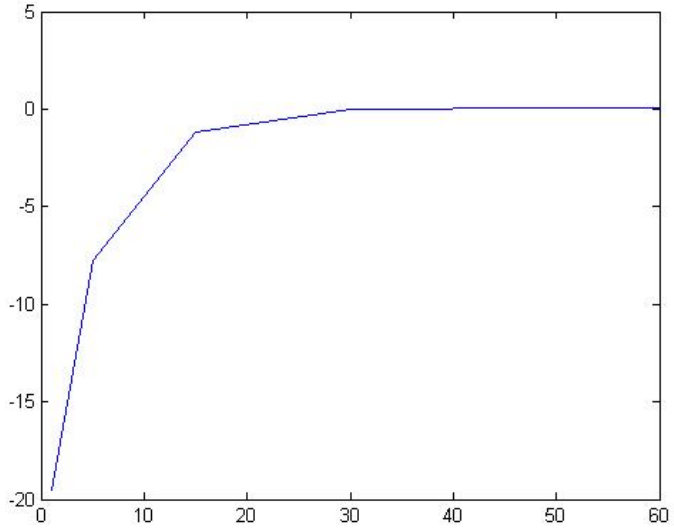

Figura 58: Diferencias de PSNR's para una cota de 300.000.

4. Discusión

Al realizar el gráfico de comparación entre la DCT y la frecuencia original (que es una biyección), extrajimos nuestras conclusiones en base al coeficiente de magnitud. Cuando dicho valor es alto, podemos afirmar que el primer vector de la base está siendo multiplicado por un valor alto, por lo tanto, hay unas presencia del vector. Al observarlo en una nueva base, no podemos extraer conclusión alguna con respecto al tiempo, sólo podemos obtener información sobre la frecuencia.

Tal como predicho en las secciones anteriores, pudimos observar, a partir de las DCT de las señales con ruido, que el agregado de ruido alteró levemente las señales sin deformarlas.

Analizando los gráficos del 12 al 26, en el caso de señales unidimensionales, se puede ver que el valor del umbral es uno de los mayores determinantes a la hora de quitarle el ruido a una señal con dicho método. Por ejemplo, para la señal dopp1024, podemos concluir que los umbrales de 1000 y 3000 no filtran enormemente el ruido pero no deforman la señal al recuperarla. Contrariamente, los umbrales de 5000 y 10000 son altamente efectivos para quitarle el ruido a una señal pero son más propensos a deformarla.
Por otro lado, en las señales referentes a g450, puede verse que el umbral más efectivo es el que tiene un valor de 5000. Si bien se pueden observar dos outliers que no se presentan en la señal original, la señal recuperada es la más aproximada a ésta.
Por último, de acuerdo a los distintos umbrales utilizados para la señal ramp1234, podemos observar que el más conveniente resulta el valor 10000.

A partir de los gráficos 27, 28 y 29, podemos concluir que cuando el ruido agregado excede el intervalo de [-20,20], la señal no puede recuperarse correctamente con el método umbral dando como resultado un PSNR menor al de la señal original con ruido. Esto ocurre pues las señales analizadas no se mueven en un rango de valores demasiado grandes por lo que el agregado de una gran cantidad de ruido lleva a la deformación de las mismas y a su dificultosa recuperación. Por otro lado, a menor ruido mayor recuperación. Esto nos permite afirmar que el método de umbral es muy eficiente siempre que la magnitud del ruido no supere en un 40 %

los valores de la señal original independiente del rango o forma de la señal utilizada.

En el caso del segundo método, podemos notar que los gráficos de recuperación de ruido (figuras 30, 31 y 32) no presentan una gran reducción del ruido, notamos que la cantidad de frecuencia suprimida no debía ser demasiado grande para no perder la información trascendente de la señal. Por otro lado, notamos que la cota a utilizar como tambien la cantidad de ruido a restar para lograr la mayor efectividad del método resultaba dependiente a cada tipo de señal y los valores entre los que éste se encontraba con lo cual el método permanece relativo a lo que se aplique. Por último, pudimos inferir, gracias a las distintas pruebas realizadas, que el método de 'Eliminación de frecuencias altas (interferencias) mediante filtrado de la señal' no es demasiado eficiente para quitar el ruido a señales que, por más de que haya una diferencia, ésta no resulta realmente notoria.

Para el caso de las señales bidimensionales, los resultados obtenidos a partir del primer método fueron que los umbrales debían hallarse en un rango que iba desde los 500.000 hasta los 8.000.000 db dependiendo del ruido. Al superar dicho valor, la imagen recuperada perdía definición mimetizándose de manera tal a que dejaran de distinguirse los elementos presentes en la misma.
El umbral que mejor promedió según este algoritmo fue 3.000.000, dando como resultado un PSNR aproximado de 23 en la imagen restaurada para la mayoría de los casos.
Por otra parte, observamos que una cota de 100.000 o menos podía arruinar la imagen en vez de mejorarla. Así como una cota chica podía no mejorar en nada a la imagen.
Visualmente, puede notarse una leve mejoría en las imágenes recuperadas con el método umbral. El ruido ya no se ve como píxeles aislados unos de otros sino que la imagen se encuentra más límpida. Dicho resultado puede comprobarse con el incremento del PSNR visibles en las Figuras 51 a 58.
Dichos PSNR nos permitieron determinar que, si bien existe un umbral con mejor promedio de PSNR en la imagen recuperada, debe tenerse en cuenta que a menor cantidad de ruido menor debe ser la cota para lograr mayor eficacia.
En el caso del segundo método, las imágenes recuperadas y las imágenes con ruido resultaron altamente similares. Si bien la mejora no es visual, ésta puede ser comprobada con los gráficos del PSNR (que se comportaron del mismo modo que para señales unidimensionales).

Por lo tanto, podemos decir que, por un lado, para filtrar correctamente una señal (de una o dos dimensiones) la función y la cota de la misma deben estar estrechamente relacionadas con la cantidad y el tipo de ruido agregado. Para el caso del ruido, hemos intentado encontrar buenas cotas que resultaran buenas para la mayoría de los casos. Sin embargo, los resultados muestran que esto no siempre es suficiente. Algunas cotas que se comportan bien con la diferencia de PSNR (siendo el PSNR de la imagen restaurada mayor que el PSNR de la imagen con ruido) no presentan diferencias considerables, esto es, que la diferencia entre los dos PSNR es menor a 0.5.

5. Conclusiones

A modo de resumen, nuestro trabajo consistió en el agregado de ruido a señales unidimensionales y bidimensionales para que luego, éstas fueran recuperadas y volvieran lo máximo posible a su estado original. Para ello, utilizamos el método de Ruido Aditivo para agregar ruido y los métodos de Umbral y Eliminación de frecuencias altas mediante filtrado para ambos tipos de señales. Luego de una gran cantidad de pruebas, logramos encontrar resultados que efectivizaran los métodos otorgando señales más limpias de ruido y similares a las originales. Para el primer método, llegamos a la conclusión de que un umbral a partir de 800 mejora la

señal cuando el ruido no exceda en valor al 40 % del valor más alto de la señal. Caso contrario, el método la distorsiona devolviendo una señal diferente a la deseada. Dichos resultados fueron verificados por el valor del PSNR de la señal con ruido y de la restaurada, que, cuando el ruido no era demasiado grande, marcaba mejorías de hasta 6 db.

Por otro lado, el segundo método otorgó resultados que no reflejaron una gran mejoría en las señales (el PSNR resultó 0,20 db mayor que el de la señal con ruido). Sin embargo, dichas mejoras permanecieron lineales de acuerdo al aumento del ruido por lo que éste método resulta mejor que el primero en los casos en los que el ruido es demasiado grande.

Nuestras conclusiones, en base a lo estudiado en las secciones anteriores, es que para lograr una señal bien filtrada con una función umbral, la cota debe ser elegida en función al tipo e intensidad del ruido. En nuestro caso, una gran cantidad de ruido aditivo no permite la correcta recuperación de la señal. Por otro lado, si el ruido agregado es menor que el umbral elegido, la funcion empeorará la señal. El análisis de los métodos mostró que la función umbral es efectiva para el ruido aditivo si la cota se encuentra cercana al promedio de ruido agregado. Por otro lado, el método de eliminación de frecuencias altas mostró no dar resultados demasiado eficientes, otorgando señales recuperadas altamente similares a las señales con ruido.

6. Apéndices

6.1. A

Laboratorio de Métodos Numéricos - Primer Cuatrimestre 2013
Trabajo Práctico Número 2

Introducción

La Transformada Discreta del Coseno (DCT, por sus siglas en inglés) es una herramienta que nos permite representar cualquier señal en el plano de las frecuencias. Dado que es utilizada por el estándar de compresión de imágenes JPEG y formato de video MPEG, se encuentra implementada en más lugares de lo que pensamos: en cada cámara digital o teléfono móvil. La DCT no solo tiene aplicaciones al mundo de la compresión (donde los valores transformados pueden ser codificados de forma eficiente), sino también al procesamiento: el análisis de qué frecuencias están presentes en las señales es esencial en ciertos contextos de aplicación.

La idea intuitiva de esta transformada, en el plano continuo, consiste en representar una función $f : \mathbb{R} \to \mathbb{R}$ en la base de funciones $\mathcal{B} = \{1, \cos(x), \cos(2x), ...\}$. En el plano discreto, la DCT se corresponde a un cambio de base: cada una de las funciones de la base $\mathcal{B}$ se discretiza en ciertos puntos pasando a ser una base de vectores en $\mathbb{R}^n$, (donde n es la dimensión del vector o señal a transformar). Es decir, dado un vector o señal $x \in \mathbb{R}^n$ existe una matriz $M \in \mathbb{R}^{n \times n}$ de cambio de base que define la transformada DCT, donde $y = Mx$ es el vector o señal transformado al espacio de frecuencias por la DCT (ver apéndice A). Esta operación es fácilmente extensible a señales de dos dimensiones (ver apéndice A.1).

Enunciado

El objetivo del trabajo es eliminar ruido sobre una señal ruidosa $x \in \mathbb{R}^n$. Para ello se realiza el siguiente proceso:

1. $y := Mx$ [Transformar usando Ec. (1) de Ap. A]

2. $\tilde{y} := f(y)$ [Modificar]
3. Resolver $M\tilde{x} = \tilde{y}$ [Reconstruir]

Una forma de medir la calidad visual de la señal reconstruida $\tilde{x}$, es a través del PSNR (*Peak Signal-to-Noise Ratio*). EL PSNR es una métrica 'perceptual' (acorde a lo que perciben los humanos) y nos da una forma de medir la calidad de una imagen perturbada, siempre y cuando se cuente con la señal original. Cuanto mayor es el PSNR, mayor es la calidad de la imagen. La unidad de medida es el decibel (db) y se considera que una diferencia de 0.5 db ya es notada por la vista humana. El PSNR se define como:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_x^2}{ECM}\right)$$

donde MAX_x define el rango máximo de la señal (en caso de entradas de 8 bits sin signo, sería 255) y ECM es el *error cuadrático medio*, definido como: $\frac{1}{n}\sum_i (x_i - \tilde{x}_i)^2$, donde n es la cantidad de elementos de la señal, x es la señal original y $\tilde{x}$ es la señal recuperada.

En la implementación realizada deben llevar a cabo los siguientes experimentos:

- Para varias señales con distintos niveles de ruido, se deberán experimentar con al menos 2 estrategias (definiendo f de forma conveniente) para modificar la señal transformada y (paso 2) con el objetivo de que la señal recuperada $\tilde{x}$ contenga menos ruido; se deberán extraer conclusiones en cuanto a la calidad de la señal recuperada, en función de la estrategia utilizada.
- Se deberán repetir los anteriores experimentos también sobre imágenes adaptando el método para aplicar la transformada DCT en dos dimensiones según se explica en la apéndice A.1.
- **(Opcional)** Se deberá analizar la aplicación de la DCT 'por bloques' sobre imágenes. Por ejemplo, si tenemos una imagen de 64×64 píxeles podemos subdividirla en: 4 bloques de 32×32, o 16 bloques de 16×16, o 64 bloques de 8×8, y aplicar la DCT en 2D sobre cada uno de los bloques (considerar un tamaño mínimo de 8×8 para cada bloque).
  Elegir una estrategia utilizada para señales unidimiensionales y sacar conclusiones respondiendo a los siguientes interrogantes (realizando experimentos que justifiquen la respuesta): ¿Es lo mismo eliminar ruido sobre la imagen entera que de a bloques? ¿Qué forma es más conveniente en cuanto a la calidad visual? ¿Qué forma es más rápida?

Formatos de archivos de entrada

Las señales serán leídas de un archivo de texto en cuya primer línea figuran la cantidad de datos y en la línea siguiente se encuentran los datos en ASCII separados por espacios. Para leer y escribir imágenes sugerimos utilizar el formato *raw* binario `.pgm`[1]. El mismo es muy sencillo de implementar y compatible con muchos gestores de fotos[2] y Matlab.

Fecha de entrega:

- *Formato electrónico:* jueves 16 de mayo de 2013, hasta las 23:59 hs., enviando el trabajo (informe+código) a `metnum.lab@gmail.com`. El subject del email debe comenzar con el texto `[TP2]` seguido de la lista de apellidos de los integrantes del grupo.
- *Formato físico:* viernes 17 de mayo de 2013, de 18 a 20hs (en la clase del labo).

[1] http://netpbm.sourceforge.net/doc/pgm.html
[2] XnView http://www.xnview.com/

A. Transformada Coseno Discreta

Para generar la matriz $M \in \mathbb{R}^{n \times n}$ que define la transformada de Coseno Discreta definimos:

- Vector de frecuencias: $g = \begin{pmatrix} 0 \\ 1 \\ \vdots \\ n-1 \end{pmatrix}$
- Vector de muestreo: $s = \frac{\pi}{n} \begin{pmatrix} \frac{1}{2} \\ 1 + \frac{1}{2} \\ \vdots \\ (n-1) + \frac{1}{2} \end{pmatrix}$
- Constante de normalización: $C(k) = \begin{cases} \sqrt{\frac{1}{n}} & k = 1 \\ \sqrt{\frac{2}{n}} & k > 1 \end{cases}$

Siendo $T = \cos(g \cdot s^t)$ la matriz resultante de aplicarle el coseno a cada elemento de la matriz $g \cdot s^t$, finalmente definimos, $\widehat{M}_{i,j} = C(i) \cdot T_{i,j}$

Para obtener una versión entera de la transformación que define la matriz M, la cual será aplicada a señales (o vectores) enteras en el rango $[0, q]$, definimos:

$$M = \left\lfloor \frac{q\widehat{M} + 1}{2} \right\rfloor \tag{1}$$

donde $\lfloor \cdot \rfloor$ indica la parte entera inferior[3]. (Es decir, escalamos los elementos de la matriz M por $q/2$ y luego redondeamos los valores.)

A.1. Extensión a 2D

Dada una matriz $B \in \mathbb{R}^{n \times n}$, podemos extender fácilmente la transformada DCT a señales de dos dimensiones. Para ello, aplicamos la transformación por filas y por columnas: $M\,B\,M^t$

A.2. B

Implementación de nuestro programa:

main.cpp

```
/**

1) Decido si leo un sonido o una imagen
2) Leo el archivo ( z = seal )
3) Agrego ruido ( R(z) = x )
4) Transformo con la DCT ( y = M * x  y = M * x * Mt )
5) Modifico ( f(y) = y' )
6) Reconstruyo ( Mx' = y' )
7) Comparo (PSNR(z, x'))
8) Liberar memoria

*/
```

[3] El redondeo de un número m pude definirse como $\lfloor m + 1/2 \rfloor$. Luego, M se define como el redondeo de $\widehat{M} \cdot (q/2)$.

```
#include <iostream>
#include <sstream>
#include <cstdio>
#include <cmath>
#include <fstream>
#include <string>
#include <cstring>
#include <list>
#include <ctype.h>
#include "Matriz.h"

using namespace std;

int main()
{
    Matriz *signal;
    double Vmax = -200;
    int sonido = 0;

    printf("-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
    printf("\nBienvenido al programa revisado del TP2 de MetNum, primer cuatrimestre 201
    cout<<"Alumnos"<<endl;
    cout<<"             Izcovich, Sabrina. LU: 550/11"<<endl;
    cout<<"             Otero, Fernando. LU: 424/11"<<endl;
    cout<<"             Vita, Sebastian. LU: 149/11"<<endl;
    string archivo;
    char respuesta = 'n';

    int cantMuestra = 0;
    ifstream entrada;
/** 1 */
    while(sonido >2 || sonido < 1)
    {
                cout << "Ingrese si quiere trabajar sobre una imagen o sonido"<<endl;
                cout << "1 = sonido" << endl;
                cout << "2 = imagen" << endl;
                cin >> sonido;
        }

/** 2 */
    while (tolower(respuesta) == 'n')
    {
        cout << "\n Ingrese el archivo donde este la muestra (sin olvidar su extension)\
        cin >> archivo;
        char * cArchivo = new char [archivo.length()+1];
        strcpy(cArchivo, archivo.c_str());

        entrada.open(cArchivo);
        if(entrada.good())
        {
            respuesta = 's';
            entrada >> cantMuestra;
            if(sonido == 1)
            {
                signal = new Matriz(cantMuestra,1);
            }
            else
            {
```

```
                signal = new Matriz(cantMuestra,cantMuestra);
            }

            int j = 0;
            double aux;
            for(int i = 0; i < cantMuestra; i++)
            {
                if(sonido == 2) //IMAGEN
                {
                    for(j = 0; j < cantMuestra; j++)
                    {
                        entrada >> aux;
                        signal->set(i, j, aux);
                        if(aux > Vmax)
                            Vmax = aux;
                    }
                }
                else    //SONIDO
                {
                    entrada >> aux;
                    signal->set(i, j, aux);
                    if(aux > Vmax)
                        Vmax = aux;
                }
            }
        }
        else
        {
            printf("\nNo se cargo el archivo. Salir? (s/n)\n");
            cin >> respuesta;
            if(tolower(respuesta) == 's')
            {
                entrada.close();
                return 0;
            }
        }
    }
    //TENGO LA SEAL (SONIDO O IMAGEN)
    Matriz *M = new Matriz();
    *M = M->DCT(Vmax,cantMuestra);

/** 3 */
    Matriz *R = new Matriz();
    *R = signal->ruidoSuma();

/** Testeo */
    cout << "PSNR de la seal con ruido sin filtrar: ";
    cout << signal->PSNR(*R,Vmax) << endl;

/** 4 */
    *R = *M * *R;
    if(sonido == 2)
        *R = *R * M->traspuesta();

    respuesta = 's';
    while(tolower(respuesta) == 's')
    {
    /** 5 */
        int efe = 0;
```

```
        while(efe <1 || efe >2)
        {
            cout << "Ingrese cual de las 2 funciones quiere aplicar para sacar el ruido:
            cout << "1 = Aplanar" << endl;
            cout << "2 = Restar umbral" << endl;
            cin >> efe;
        }
        int umbral = -1;
        while(umbral < 0)
        {
            cout << "Ingrese el valor borde para la funcion elegida: " << endl;
            cin >> umbral;
        }
        if(efe == 1)
            R->umbral(umbral);
        else
            R->menosUmbral(umbral);

            R->imprimir();

    /** 6 */
    *R = M->resolverGauss(*R);

    /** 7 */
        cout << "PSNR de la seal con ruido luego de tratarla: ";
        cout << signal->PSNR(*R,Vmax) << endl << endl;

        respuesta = 't';
        while(tolower(respuesta) != 'n' && tolower(respuesta) != 's')
        {
            cout << "Quiere volver a ingresar la funcion? (s/n)" << endl;
            cin >> respuesta;
        }
    }
/** 8 */
    delete M;
    delete R;
    delete signal;

        return 0;
}
```

tcd.cpp

```
#include <cmath>
#include <stdlib.h>
#include "tcd.h"

#define pi 3.14159265359

using namespace std;

double** vectorDeFrecuencias (unsigned int n){
        double** res = create_vector(n);
        for (unsigned int i = 0; i < n; i++)
                *res[i] = double(i);
        return res;
}
```

```
double** vectorDeMuestreo (unsigned int n){
        double** res = create_vector(n);
        for (unsigned int i = 0; i < n; i++){
                *res[i] = double(1 + (i/2));
                *res[i] = double(*res[i] * double(pi/n));
        }
        return res;
}

double** mSombrero (unsigned int n){
        double **g = vectorDeFrecuencias(n);
        double **s = vectorDeMuestreo(n);
        double **res = create_matrix(n, n);
        for (unsigned int i = 0; i < n; i++)
                for (unsigned int j = 0; j < n; j++)
                {
                        double t = *g[i] * *s[j];
                        if(i == 1)
                                res[i][j] = sqrt(1/n) * cos(t);
                        else
                                res[i][j] = sqrt(2/n) * cos(t);
                }
        return res;
}

double** M (int q, unsigned int n)
{
        double **mS = mSombrero(n);

        double **res = create_matrix(n, n);
        for (unsigned int i = 0; i < n; i++)
                for (unsigned int j = 0; j < n; j++)
                {
                        res[i][j] = floor((q * mS[i][j] + 1)/2);
                }
        return res;
}

double** multMatrices(double **a, double **b, unsigned int colsA, unsigned int rowsA, ur
{
    //colsA = rowsB
        double **res = create_matrix(colsB, rowsA);
        for (unsigned int i = 0; i < colsB; i++)
                for (unsigned int j = 0; j < rowsA; j++)
                {
                        res[i][j]= a[i][0] * b[0][j];
                        for(unsigned int k = 1; k < colsA; k++)
                        {
                                res[i][j] += a[i][k] * b[k][j];
                        }
                }
        return res;
}
double** multMatrizVector(double **a, double **b, unsigned int n)
{
        double **res = create_matrix(n, n);
        for (unsigned int i = 0; i < n; i++)
                for (unsigned int j = 0; j < n; j++)
```

```
                {
                        res[i][j]= a[i][0] * *b[0];
                        for(unsigned int k = 1; k < n; k++)
                        {
                                res[i][j] += a[k][j] * *b[k];
                        }
                }
        return res;
}

double** traspuesta(double **m, unsigned int n)
{
        double **res = create_matrix(n, n);
        for (unsigned int i = 0; i < n; i++)
                for (unsigned int j = 0; j < n; j++)
                {
                        res[j][i] = m[i][j];
                }
        return res;
}

double** resolverSistema(double **M, unsigned int n, double **vectord)
{
    int** pivot = create_vectorInt(n);
    double** result = create_vector(n);
    vueltaMatrizModificada(M, vectord, pivot, result, n);
    destroy_vectorInt(pivot, n);
    return result;
}

double** resolverSistema2D(double **M, unsigned int n, double **matriz)
{
    int** pivot = create_vectorInt(n);
    double** elemIesimo = create_vector(n);
    double** resultIesimo = create_vector(n);
    double** result = create_matrix(n, n);
    for(unsigned int j = 0; j < n; j++)
    {
        for(unsigned int i = 0; i < n; i++)
        {
            *elemIesimo[i] = matriz[i][j];
        }
        vueltaMatrizModificada(M, elemIesimo, pivot, resultIesimo, n);
        for(unsigned int i = 0; i < n; i++)
        {
            matriz[i][j] = *resultIesimo[i];
        }
    }
    destroy_vectorInt(pivot, n);
    destroy_vector(elemIesimo, n);
    destroy_vector(resultIesimo, n);
    return result;
}


double** ruidoSuma(double **signal, unsigned int n)
{
        double** res = create_vector(n);
        srand (n);
```

```
        for(unsigned int i=0; i<n; i++)
        {
                int ruido = rand() % 500;
                *res[i]= *signal[i] + double(ruido)/100; //random entre 0 y 0.499
        }
        return res;
}

double** ruidoSumaM(double **matriz, unsigned int cols, unsigned int rows)
{
        double** res = create_matrix(cols, rows);
        srand (cols);
        for(unsigned int i=0; i<cols; i++)
        {
        for(unsigned int j=0; i<rows; i++)
        {
            int ruido = rand() % 500;
            res[i][j]= matriz[i][j] + double(ruido)/100; //random entre 0 y 0.499
        }
        }

        return res;
}

double** ruidoMult(double **signal, unsigned int n)
{
        double** res = create_vector(n);
        srand (n);
        for(unsigned int i=0; i<n; i++)
        {
                int ruido = rand() % 500;
                *res[i]= *signal[i] * double(ruido)/100; //random entre 0 y 0.499
        }
        return res;
}

double** ruidoMultm(double **matriz, unsigned int cols, unsigned int rows)
{
        double** res = create_matrix(cols, rows);
        srand (cols);
        for(unsigned int i=0; i<cols; i++)
        {
            for(unsigned int j=0; j<rows; j++)
        {
            int ruido = rand() % 500;
            res[i][j]= matriz[i][j] * double(ruido)/100; //random entre 0 y 0.499
        }
        }
        return res;
}

void fUmbral(double **signal, unsigned int n, int umbral)
{
        for(unsigned int i = 0; i<n; i++)
        {
                if(abs(*signal[i])<umbral)
                {
                        *signal[i] = 0;
                }
```

```
        }
}

void fMenosUmbral(double **signal, unsigned int n, int umbral){
    for (unsigned int i = 0; i<n; ++i) {
        if (*signal[i] > umbral) {
            *signal[i]-=umbral;
        }
    }
}

void fUmbralm(double **matriz, unsigned int cols, unsigned int rows, int umbral)
{
        for(unsigned int i = 0; i<cols; i++)
        {
                for(unsigned int j = 0; j<rows; j++)
        {
            if(abs(matriz[i][j])<umbral)
            {
                matriz[i][j] = 0;
            }
        }
        }
}

void fMenosUmbralm(double **matriz, unsigned int cols, unsigned int rows, int umbral){
    for (unsigned int i = 0; i<cols; ++i) {
        for (unsigned int j = 0; j<rows; ++j) {
            if (matriz[i][j] > umbral) {
                matriz[i][j]-=umbral;
            }
        }
    }
}


double PSNR(double **signal, double **conRuido, unsigned int n, double Vmax)
{
        double ecm = ECM(signal, conRuido, n);
        Vmax *=Vmax;
        if(ecm != 0)
        return 10 * log10(Vmax/ecm);
    else
        return 9999999999999;
}

double ECM(double **signal, double **conRuido, unsigned int n)
{
        double sum = 0;
        for(unsigned int i = 0; i < n; i++)
        {
                sum+=pow(*signal[i] - *conRuido[i],2);
        }

        return sum/n;
}

double PSNRm(double **matriz, double **conRuido, unsigned int cols, unsigned int rows, d
{
```

```
        double ecm = ECMm(matriz, conRuido, cols, rows);
        Vmax *=Vmax;
        if(ecm != 0)
        return 10 * log10(Vmax/ecm);
    else
        return 9999999999999;
}

double ECMm(double **matriz, double **conRuido, unsigned int cols, unsigned int rows)
{
        double sum = 0;
        for(unsigned int i = 0; i < cols; i++)
        {
            for(unsigned int j = 0; j < rows; j++)
        {
            sum+=pow(matriz[i] - conRuido[i],2);
        }
        }
        return sum/(cols*rows);
}

double** create_matrix(unsigned int cols, unsigned int rows) {
        double **bar = (double**)malloc(rows * sizeof(double*));
    for (size_t i = 0; i < rows; ++i)
    {
        bar[i] = (double*)malloc(cols * sizeof(double));
    }
    return bar;
}

void destroy_matrix(double **matrix, unsigned int rows) {
    for (size_t i = 0; i < rows; i++) {
        free(matrix[i]);
    }
    free(matrix);
}

double** create_vector(unsigned int n) {
        unsigned int rows = n, cols = 1;
        double **bar = (double**)malloc(rows * sizeof(double*));
    for (size_t i = 0; i < rows; ++i)
    {
        bar[i] = (double*)malloc(cols * sizeof(double));
    }
    return bar;
}

void destroy_vector(double **vector, unsigned int rows) {
    free(vector);
}

int** create_vectorInt(unsigned int n) {
        unsigned int rows = n, cols = 1;
        int **bar = (int**)malloc(rows * sizeof(int*));
    for (size_t i = 0; i < rows; ++i)
    {
        bar[i] = (int*)malloc(cols * sizeof(int));
    }
    return bar;
```

```
}

void destroy_vectorInt(int **vector, unsigned int rows) {
    free(vector);
}

int LU_conpivote(double **A, int **pivot, int n)
{
   int i, j, k;
   double *p_k, *p_row, *p_col;
   double max;

   for (k = 0, p_k = *A; k < n; p_k += n, k++) {

      *pivot[k] = k;
      max = fabs( *(p_k + k) );
      for (j = k + 1, p_row = p_k + n; j < n; j++, p_row += n) {
         if ( max < fabs(*(p_row + k)) ) {
            max = fabs(*(p_row + k));
            *pivot[k] = j;
            p_col = p_row;
         }
      }

      if (*pivot[k] != k)
         for (j = 0; j < n; j++) {
            max = *(p_k + j);
            *(p_k + j) = *(p_col + j);
            *(p_col + j) = max;
         }

      if ( *(p_k + k) == 0.0 ) return -1;

      for (i = k+1, p_row = p_k + n; i < n; p_row += n, i++) {
         *(p_row + k) /= *(p_k + k);
      }

      for (i = k+1, p_row = p_k + n; i < n; p_row += n, i++)
         for (j = k+1; j < n; j++)
            *(p_row + j) -= *(p_row + k) * *(p_k + j);

   }

   return 0;
}

int vueltaMatrizModificada(double **A, double** B, int** pivot, double** x, int n)
{
   int i, k;
   double *p_k;
   double dum;
    LU_conpivote(A, pivot, n);

   for (k = 0, p_k = *A; k < n; p_k += n, k++) {
      if (*pivot[k] != k) {dum = *B[k]; *B[k] = *B[*pivot[k]]; *B[*pivot[k]] = dum; }
      *x[k] = *B[k];
      for (i = 0; i < k; i++) *x[k] -= *x[i] * *(p_k + i);
   }
```

```
    for (k = n-1, p_k = *A + n*(n-1); k >= 0; k--, p_k -= n) {
        if (*pivot[k] != k) {dum = *B[k]; *B[k] = *B[*pivot[k]]; *B[*pivot[k]] = dum; }
        for (i = k + 1; i < n; i++) *x[k] -= *x[i] * *(p_k + i);
        if (*(p_k + k) == 0.0) return -1;
        *x[k] /= *(p_k + k);
    }

    return 0;
}
```

tcd.h

```
#ifndef TCD_
#define TCD_

#include <iostream>
#include <cmath>
using namespace std;

double** M (int q,unsigned int n);
double** mult(double **a, double **b);
double** traspuesta(double **m, unsigned int n);
//void traspuesta(double **m, unsigned int n);
double** multMatrices(double **a, double **b, unsigned int colsA, unsigned int rowsA, un
double** multMatrizVector(double **a, double **b, unsigned int n);
double** resolverSistema(double **M, unsigned int n, double **vectord); // en
vector devuelve el resultado
double** resolverSistema2D(double **M, unsigned int n, double **matriz); // en
matriz devuelve el resultado
double** factorizar(double **L, double **U, double **A, unsigned int n); //devuelve
la matriz P
double** ruidoSuma(double **signal, unsigned int n);
double** ruidoSumaM(double **signal, unsigned int cols, unsigned int rows);
double** ruidoMult(double **signal, unsigned int n);
void fUmbral(double **signal, unsigned int n, int umbral);
void fMenosUmbral(double **signal, unsigned int n, int umbral);
void fUmbralm(double **signal, unsigned int cols, unsigned int rows, int umbral);
void fMenosUmbralm(double **signal, unsigned int cols, unsigned int rows, int umbral);
double PSNR(double **signal, double **conRuido, unsigned int n, double max);
double PSNRm(double **signal, double **conRuido, unsigned int cols, unsigned int rows, d
double ECM(double **signal, double **conRuido, unsigned int n);
double ECMm(double **signal, double **conRuido, unsigned int cols, unsigned int rows);

double** create_matrix(unsigned int cols, unsigned int rows);
void destroy_matrix(double **matrix, unsigned int rows);
double** create_vector(unsigned int n);
void destroy_vector(double **vectord, unsigned int rows);

int** create_vectorInt(unsigned int n);
void destroy_vectorInt(int **vectord, unsigned int rows);


int vueltaMatrizModificada(double **A, double** B, int** pivot, double** x, int n);
int LU_conpivote(double **A, int **pivot, int n);

#endif
```

B. Referencias

- BURDEN, RICHARD L. ; Análisis numérico, 7ma ed. 2002. México, Thomson Learning.
- http://planetmath.org/discretecosinetransform
- http://www4.ujaen.es/ satorres/practicas/practica2.pdf
- http://www.eueti.uvigo.es/files/material_docente/478/enmascaramiento.pdf