# Métodos Numéricos

Primer Cuatrimestre de 2012

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

## Filtrado de imágenes resueltas con ecuaciones lineales.

Ecuaciones Lineales, Filtro de imagen

**Trabajo Práctico N°2**



| Integrante | LU | Correo electrónico |
|---|---|---|
| Mancuso, Emiliano | 597/07 | emiliano.mancuso@gmail.com |
| Mataloni, Alejandro | 706/07 | amataloni@gmail.com |
| Tolchinsky, Lucas | 591/07 | lucas.tolchinsky@gmail.com |



**Reservado para la catedra**

| Instancia | Docente | Nota |
|---|---|---|
| Primera entrega | | |
| Segunda entrega | | |

**Un problema típico al trabajar con imágenes digitales es la existencia de *ruido* en las mismas. En este trabajo desarrollamos un filtro para reducir el *ruido* en las imágenes, por el método de diferencias finitas.**

# Índice

# 1. Introducción Teórica

El objetivo de este trabajo práctico es reducir el ruido de una imagen. Para lograr esto se plantea minimizar una función que relaciona el *ruido* de la imagen original con la *suavidad* de la obtenida. Al derivar e igualar a cero esa función se obtiene una ecuación, que al discretizarse pixel por pixel, nos da una relación entre un pixel y sus cuatro adyacentes. A esto se lo conoce como *método de diferencias finitas*, que plantea un sistema de ecuaciones a resolver.

En este sistema, lidiamos con una ecuación por pixel, por lo tanto, si tenemos $N$ pixeles, tenemos una matriz de $N \times N$. De todas maneras, en cada ecuación sólo se trabaja con cinco variables: un pixel y sus adyacentes (salvo para los bordes, que más adelante explicamos su tratamiento). Por esta razón, cada fila de la matriz tiene cinco coeficientes, y el resto se completa con ceros. El resultado de esto es que la matriz resulta tener 5 bandas: la diagonal principal, dos bandas por debajo y dos bandas por arriba, donde cada banda representa un pixel adyacente al correspondiente en la diagonal principal.

La anatomía de esta matriz es importante ya que a la hora de operar con ella en nuestro trabajo, podemos asegurar que sólamente se ven afectados los valores que están entre la banda inferior y la banda superior. Este resultado será importante a la hora de guardar la matriz.

# 2. Desarrollo

Para llevar adelante el filtrado de las imágenes implementamos el método de diferencias finitas. En este problema, cada pixel de la imagen filtrada y sus cuatro pixeles adyacentes (llamados $u$), guardan una relación con el pixel correspondiente en la imagen original (llamado $\tilde{u}$).

$$- u_{i-1,j} - u_{i,j-1} + (4 + \lambda)u_{i,j} - u_{i,j+1} - u_{i+1,j} = \lambda \tilde{u}_{i,j} \tag{1}$$

Esto significa que por cada pixel de la imagen filtrada, tenemos una ecuación que involucra cinco variables.

En el caso de los bordes, lo que decidimos fue que dichas ecuaciones sólo involucren la variable del pixel correspondiente. De este modo al filtrarse, no cambia.

$$u_{i,j} = \tilde{u}_{i,j} \tag{2}$$

## 2.1. Estructura de representación

Teniendo dichas ecuaciones, el próximo paso consiste en obtener y guardar la matriz asociada al sistema.

Como se ve en (1) y (2), los coeficientes que acompañan las variables son, al principio, fijos.

De esta manera es sencillo representar cada fila de la matriz. El inconveniente es que dicha matriz tiene tamaño $N \times N$, donde $N$ es la cantidad de pixeles, y en su mayoría contiene ceros.

Con la idea de optimizar la memoria, el primer acercamiento que tuvimos fue no guardar los ceros de la matriz, sino solamente los valores. Lo que hicimos fue utilizar un Diccionario (stl::map) de dos dimensiones: (*fila*, *columna*, *valor*), y de esta manera no necesitamos guardar valores nulos.

Luego de implementar la solución al sistema de ecuaciones (discutido más adelante), notamos que el hecho de utilizar la biblioteca stl::map hacía que la complejidad del algoritmo fuese demasiado alta, teniendo por ejemplo imágenes de $64 \times 64$ que demoraban cerca de medio minuto en ser filtradas. Entendiendo que si bien la complejidad no es parte del espectro del trabajo práctico, el hecho de que el algoritmo sea lento es un obstáculo a la hora de realizar pruebas, y por este motivo decidimos pensar nuevamente la estructura.

Analizamos primero el hecho de que la matriz asociada es 5 bandas: la diagonal principal correspondiente al iésimo pixel de la imagen filtrada, y una banda por cada pixel adyacente (figura 1). Las primeras *n + 1* (con $n$ el ancho de la imagen) filas son filas de la matriz identidad, ya que corresponde con pixeles del borde superior y lo mismo sucede con las últimas *n + 1*, del borde inferior. Luego, cada vez que el pixel sea de un borde lateral, la matriz tendrá una fila identidad.

$$\begin{pmatrix}
1 & \cdots & & & & & & & \\
\cdots & 1 & \cdots & & & & & & \\
\cdots & & 1 & \cdots & & & & & \\
 & & & \ddots & & & & & \\
\cdots & -1 & \cdots & -1 & \lambda+4 & -1 & \cdots & -1 & \cdots \\
\cdots & & -1 & \cdots & -1 & \lambda+4 & -1 & \cdots & -1 \\
 & & & -1 & \cdots & -1 & \lambda+4 & -1 & \cdots \\
 & & & & & & \ddots & & \\
 & & & & & & & 1 & \cdots \\
 & & & & & & & 1 & \cdots \\
 & & & & & & & & 1
\end{pmatrix}$$

Figura 1: matriz de cinco bandas. Los $\cdots$ representan ceros

Aprovechando esto, notamos que los ceros por encima de la banda superior y debajo de la banda inferior nunca cambian, de modo que no es necesario guardarlos.

Por lo tanto, si bien la matriz asociada es de $N \times N$, alcanza con guardar una matriz de $N \times (2n+2)$, donde $n$ es el ancho de la imagen. Este último cálculo surge de que necesitamos guardar los coeficientes entre la banda inferior y la diagonal principal (n valores más uno de la diagonal), la diagonal principal y la banda superior (n valores más), y el dato correspondiente a cada fila.

Finalmente, logramos mejorar la complejidad del algoritmo a la vez que optimizamos la memoria utilizada para la representación.

## 2.2. Solución del sistema

A la hora de resolver el sistema de ecuaciones analizamos dos alternativas:

- Eliminación Gaussiana
- Factorización QR

Como la complejidad de los algoritmos es la misma, elegimos *eliminación Gaussiana* por que era un algoritmo ya conocido por nosotros y para el objetivo del TP las ventajas de la Factorización QR no harían diferencia. De hecho, dado que la matriz asociada es rala, y que debajo de la diagonal principal hay solamente dos bandas, la *eliminación Gaussiana* es conveniente porque solo tenemos 2 elementos a eliminar bajo cada diagonal.

En un principio, cuando contábamos con la estructura basada en Diccionarios, encontramos que recorrer cada fila de la columna a triangular en busca de los valores a eliminar resultaba lento, en especial al principio, cuando solamente contamos con a lo sumo dos valores bajo la diagonal principal. Esto nos llevó a implementar una estructura auxiliar para registrar en qué filas se encuentran los coeficientes de una columna dada.

Aprendimos luego que a medida que se triangula la matriz, la misma se va llenando y su densidad aumenta de modo tal que mantener esta estructura no resulta positivo para la complejidad. Este fue uno de los puntos claves que nos llevó a reconsiderar la estructura.

Por último, una vez que analizamos la anatomía de la matriz 5 bandas y dimos con la estructura óptima explicada anteriormente, para triangular sólo se requiere que en cada columna se recorra hasta la banda inferior, que se encuentra a $n$ valores de distancia. De esta manera no es necesaria otra estructura auxiliar.

## 2.3. Submuestreo y Sobremuestreo

Más allá de las optimizaciones que realicemos, la calidad de las fotografías supera ampliamente la capacidad de procesamiento y almacenamiento de nuestras computadoras, y por lo tanto necesitamos una estrategia para poder filtrar imágenes de estas dimensiones.

La técnica del **submuestreo** consiste en reducir la imagen dependiendo de un `factor de reducción`, aplicar el proceso de filtrado a la imagen de menor dimensión y luego restablecer la imagen a las dimensiones originales. Esto último se lo conoce como **sobremuestreo**.

Existiendo varias implementaciones para los algoritmos de **submuestreo**, por su simplicidad decidimos optar por eliminar filas y columnas correspondientes al parámetro recibido. Quiere decir, que nos quedamos con un pixel por cada **r** pixeles (siendo **r** el factor de reducción pasado por parámetro).

Entonces, si el $r = 1$ nos queda la imagen sin reducir. Si $r = 2$ filtramos una imagen que contiene los pixeles en filas o columnas pares (es decir ese pixel representa $r$ que es 2). Y así sucesivamente.

En las siguientes imágenes, marcamos en verde los pixeles que seleccionamos para trabajar, dependiendo el **r**.

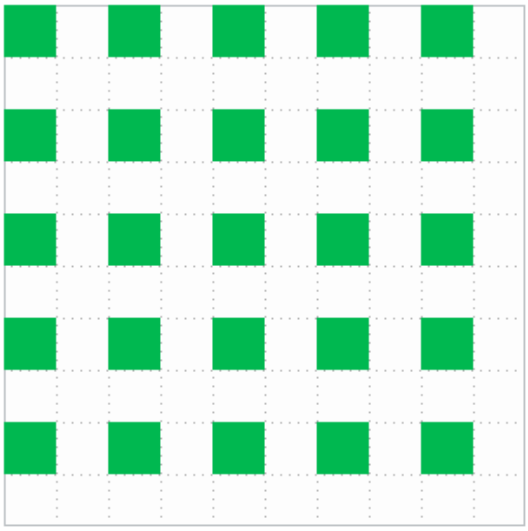

Figura 2: $r = 2$. Un pixel representa a 4

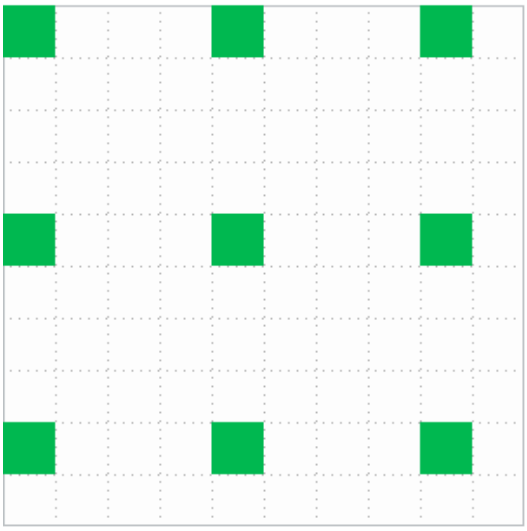

Figura 4: $r = 4$. Un pixel representa a 16

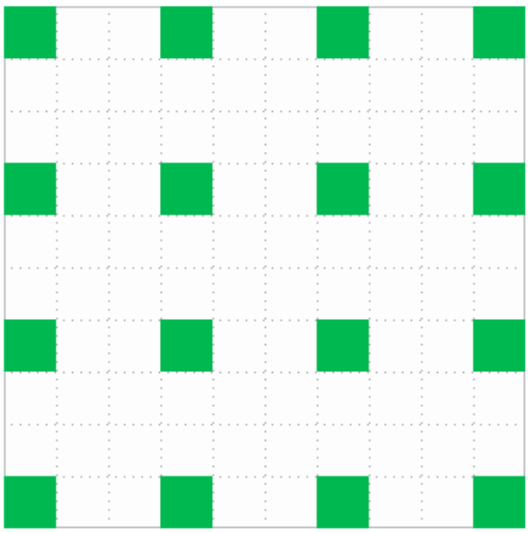

Figura 3: $r = 3$. Un pixel representa a 9

Elegimos mantener el parámetro como un número entero que representa la cantidad de filas y columnas que se van a eliminar en lugar de utilizar porcentajes pues al utilizar porcentajes, el algoritmo de reducción no sería tan sencillo.

Por ejemplo, si queremos trabajar con el 50 % de la foto, solo deberíamos eliminar 1 por cada 2 filas, pero sin eliminar columnas. En otros casos, como por ejemplo el 25 % deberíamos eliminar una fila y una columna.

Como esta parte del Trabajo Práctico era un adicional y no el objetivo principal, decidimos mantener la implementación sencilla.

## 2.4. Saturación

Todas las imágenes con las que trabajamos, son en escala de grises reducida y por lo tanto es importante mantener la misma y ser cuidadosos a la hora de volcar los resultados en una imagen. Para que la escala de grises sea proporcional, seleccionamos el valor mínimo (el más oscuro), el máximo (el más claro) y a través de una regla de tres simple, transformamos todos los pixeles en valores que van de 0 a 255.

# 3. Discusión y Resultados

Una vez desarrollados los algoritmos, comenzamos a generar ruido a las imágenes entregadas por la cátedra para luego aplicar el filtro y comparar los resultados. Como el algoritmo demora bastante con imágenes grandes, iniciamos las pruebas fundamentales con imágenes de 128x128.

## 3.1. Generar y filtrar imágenes con ruido sin factor de reducción

Para empezar, comprobamos que nuestro algoritmo para generar ruido en imágenes no beneficiaba ni perjudicaba el filtro que luego aplicaríamos, para esto utilizamos una herramienta, **GIMP**, para introducir el ruido, aplicar el filtro y compararlo con el nuestro.

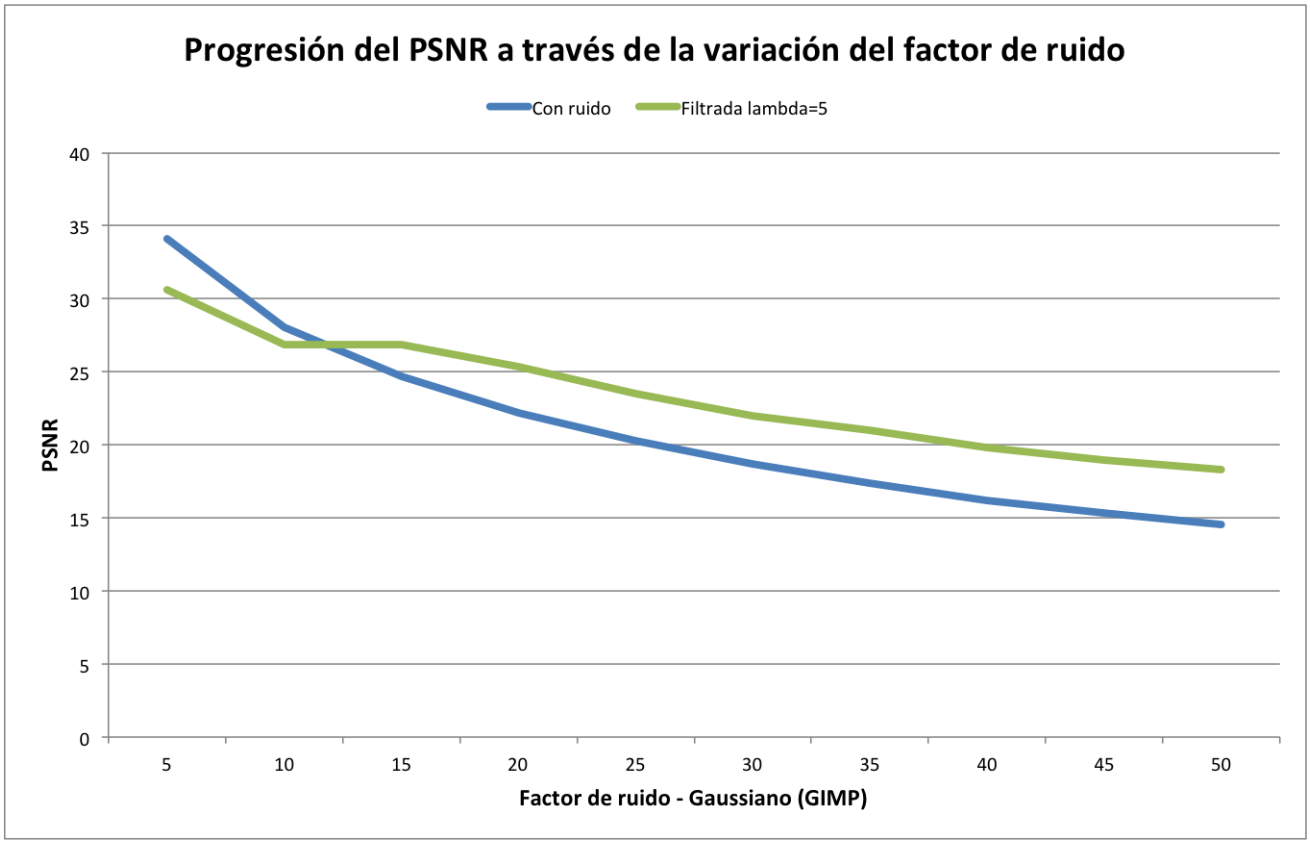


Figura 5: Imagen de prueba: Einsten. Valores del PSNR cuando filtramos la imagen con ruido, variando el factor de ruido introducido con el GIMP.

En la `figura 1` vemos la mejora que produce el filtro con respecto a la imagen con ruido para un $\lambda$ determinado. El $\lambda = 5$, nos resulto muy conveniente para nuestro algoritmo dado que la mayoría de veces nos da una amplia mejora.

Ahora que sabemos como se comporta con una imagen con ruido generado por una herramienta como el GIMP, nos enfocamos a comparar con nuestra propia implementación de inserción de ruido. En la `figura 2` vamos a ver que tiene un comportamiento similar, y de hecho utilizamos otros $\lambda$ para mostrar que no es un caso particular.

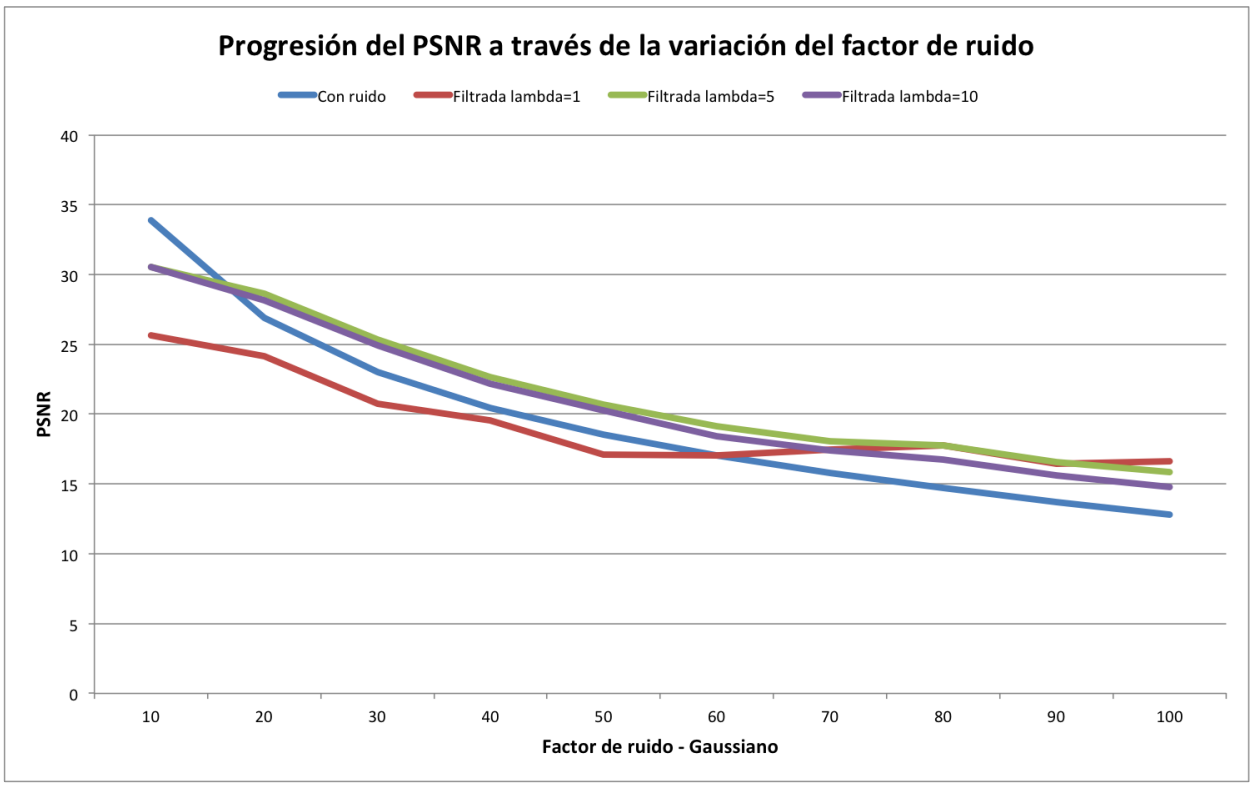


Figura 6: Imagen de prueba: Einsten. Valores del PSNR cuando filtramos la imagen con ruido introducido con nuestro algoritmo.

Mientras observamos el comportamiento partiendo de diferentes ruidos, notamos cierta relación entre el ruido y el $\lambda$. Tomamos otra imagen y rehicimos todas las pruebas, y volcamos los resultados en el gráfico de la `figura 7`.

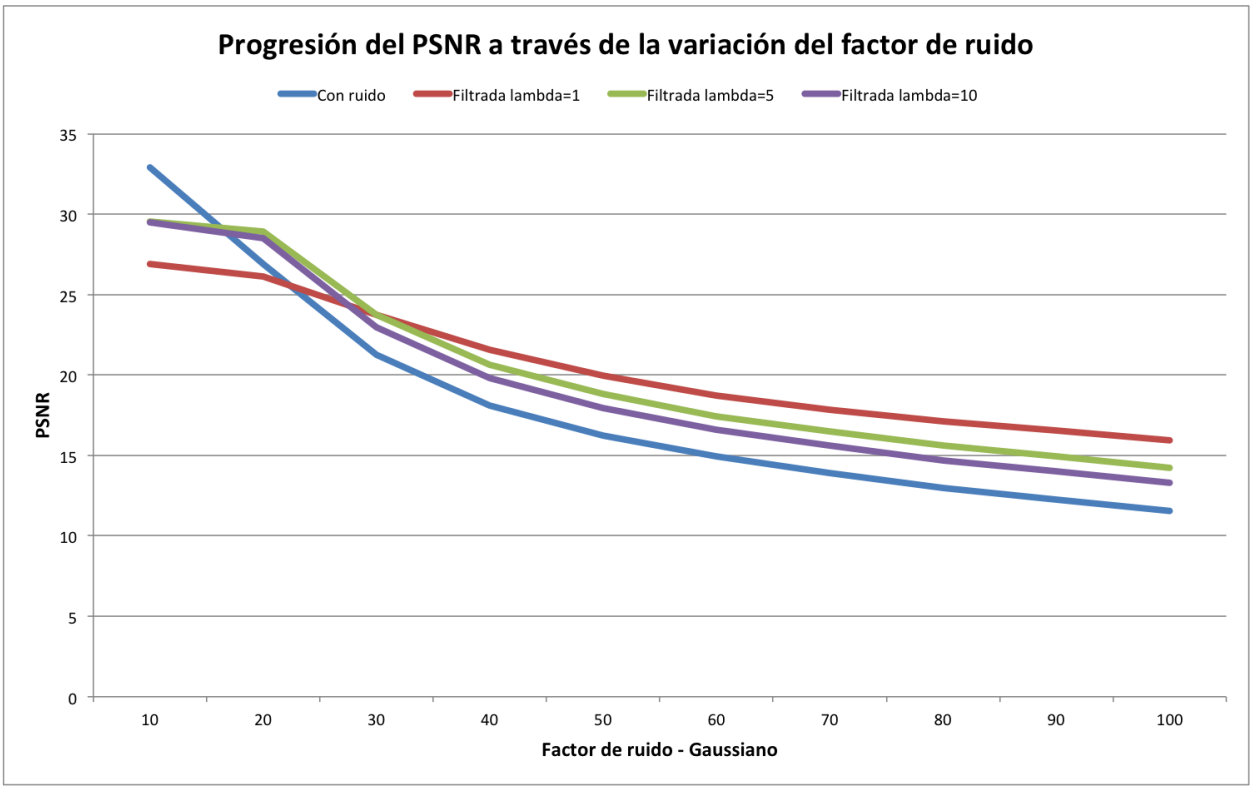


Figura 7: Imagen de prueba: Blond. Valores del PSNR cuando filtramos la imagen con ruido, variando $\lambda$

Una vez más, encontramos la relación entre los parámetros. Para imágenes con poco ruido, nuestro filtro genera imágenes de mejor calidad con $\lambda$ más grandes. Y cuando el factor de *ruido* es mayor, conviene seleccionar un $\lambda$ más chico.

La explicación para esto la encontramos en la fórmula que buscamos optimizar:

$$\Pi = \int_\Omega \frac{\lambda}{2} |u - \tilde{u}|^2 + \frac{1}{2} \|\nabla u\|^2 d\Omega$$

Vemos que $\lambda$ es un valor que afecta sólo al primer término, que mide el *peso* del ruido en la imagen original. Por lo tanto cuanto más chico es este valor menos "importancia"tiene. Por eso creemos que para grandes valores de *factor de ruido* es conveniente reducir el *peso* del mismo para ayudar al filtrado. De la misma manera cuando el *factor de ruido* es bajo darle mayor relevancia ayuda la detección del mismo.

Es interesante denotar que la manera en la que sucede esta tendencia depende de la imagen que se filtra. En la `figura 7` vemos que con un *factor de ruido* entre 20 y 30, las curvas se van acomodando de manera de que a partir de este punto la mejor calidad se logra con $\lambda = 1$. Sin embargo, en la `figura 6`, que corresponde a una imagen diferente, tomando $\lambda = 1$ obtenemos un PSNR menor al resto incluso en factores de ruido cercanos a 70.

Sin embargo, continuamos con las pruebas con otra imagen, y esta vez variando progresivamente el $\lambda$. Los resultados que obtuvimos fueron los siguientes:

| $\lambda$ | PSNR | | |
|---|---|---|---|
| | F.R. 10 | F.R. 50 | F.R. 100 |
| **Ruido** | **32.89** | **18.88** | **19.95** |
| **1** | **21.73** | **25.80** | **18.50** |
| 2 | 22.19 | 24.40 | 17.35 |
| 3 | 22.24 | 23.32 | 16.51 |
| 4 | 22.46 | 22.54 | 15.90 |
| **5** | **22.61** | **21.96** | **15.43** |
| 6 | 22.71 | 21.51 | 15.06 |
| 7 | 22.78 | 21.16 | 14.76 |
| 8 | 22.87 | 20.86 | 14.52 |
| 9 | 22.94 | 20.66 | 14.31 |
| **10** | **22.99** | **20.49** | **14.14** |
| 11 | 23.03 | 20.34 | 13.99 |
| 12 | 23.06 | 20.21 | 13.86 |
| 13 | 23.08 | 20.09 | 13.74 |
| 14 | 23.10 | 19.99 | 13.64 |
| 15 | 23.12 | 19.89 | 13.55 |
| 16 | 23.13 | 19.81 | 13.47 |
| 17 | 23.14 | 19.74 | 13.39 |
| 18 | 23.15 | 19.67 | 13.33 |
| 19 | 23.16 | 19.63 | 13.27 |
| 20 | 23.16 | 19.58 | 13.21 |

Como podemos ver, acertamos con la suposición que para imágenes con poco ruido, nuestro filtro genera imágenes de mejor calidad con $\lambda$ más grandes. Y cuando el factor de *ruido* es mayor, conviene seleccionar un $\lambda$ más chico.

Habiendo verificado que esta tendencia se mantiene, lo que suponemos es que siendo que $\lambda$ mide la importancia relativa entre el ruido de la imagen sin filtrar y la suavidad de la imagen obtenida, el punto donde las curvas se estabilizan depende enteramente de la imagen que se esté tratando. Analizando la tabla vista arriba podemos confirmarlo: para $\lambda = 1$ y *factor de ruido* igual a 10 el PSNR es menor incluso que el de la imagen ruidosa, sin embargo cuando pasamos a *factor de ruido* 50 notamos una mejora mayor que para $\lambda = 5$ y $\lambda = 10$. Luego, para *factor de ruido* 100, la relación se mantiene.

Resumiendo, en la `figura 6` el equilibrio se concreta para valores de *factor de ruido* mayores a 80, para la `figura 7` de 30 en adelante y en la tabla anterior en algún momento entre *factor de ruido* 10 y 50. Cada uno de estos casos fue realizado con imágenes distintas y concluimos que esta es la razón del fenómeno.

## 3.2. Factor de reducción. Ventajas y desventajas

Un filtro para limpiar imágenes es realmente útil si permite trabajar con imágenes grandes y no sólo eso, sino también poder controlar la relación tiempocalidad y este es el puntapié para nuestras siguientes pruebas.

Tomamos un conjunto de imágenes de distintos tamaños y le introdujimos ruido para luego aplicarle el filtro, pero esta vez aplicando la técnica de **submuestreo** y **sobremuestreo**.

El factor de reducción trae como desventaja la pérdida de calidad de la imagen, a medida que este factor crece. Es de esperar pues estamos eliminando mucha *información* de la imagen por cada fila y columna que eliminamos.

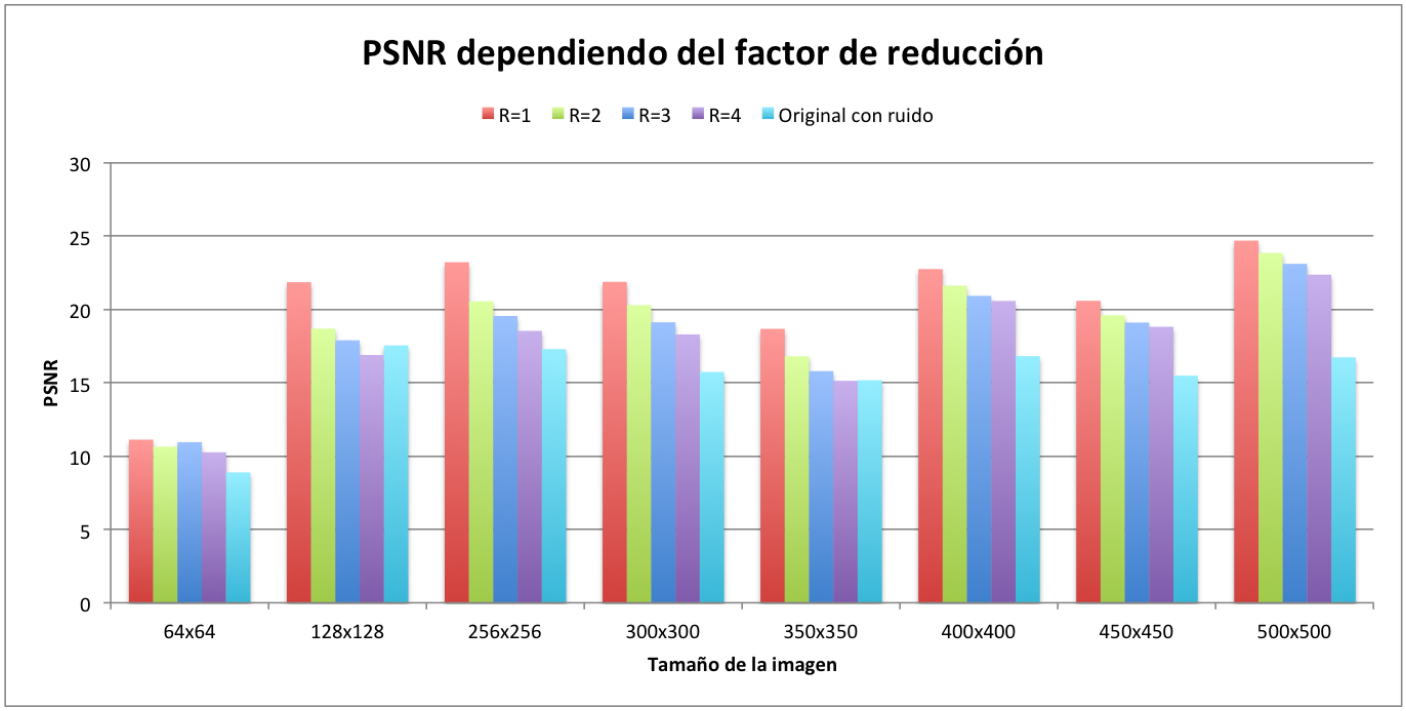


Figura 8: PSNR variando el factor de reducción. A medida que agrandamos el factor de reducción se pierde calidad.

En la `figura 8` marcamos los valores de los PSNR obtenidos y se ve claramente que seleccionando un factor de reducción más grande se pierde calidad. Si bien aumentar este parámetro en imágenes pequeñas, no sirve para obtener buenos resultados, si nos permite extraer conclusiones del comportamiento, que es nuestro caso. Debido a una cuestión de Hardware, las imágenes que superaban el tamaño de 600x600 no pudimos terminar de correrlas, sobretodo porque ya las superiores a 512x512 demoraban más de 20 minutos cada una!

En la `figura 9` graficamos los tiempos que demoramos en filtrar las imágenes utilizadas para la prueba anterior, y podemos ver como el tiempo crece exponencialmente.

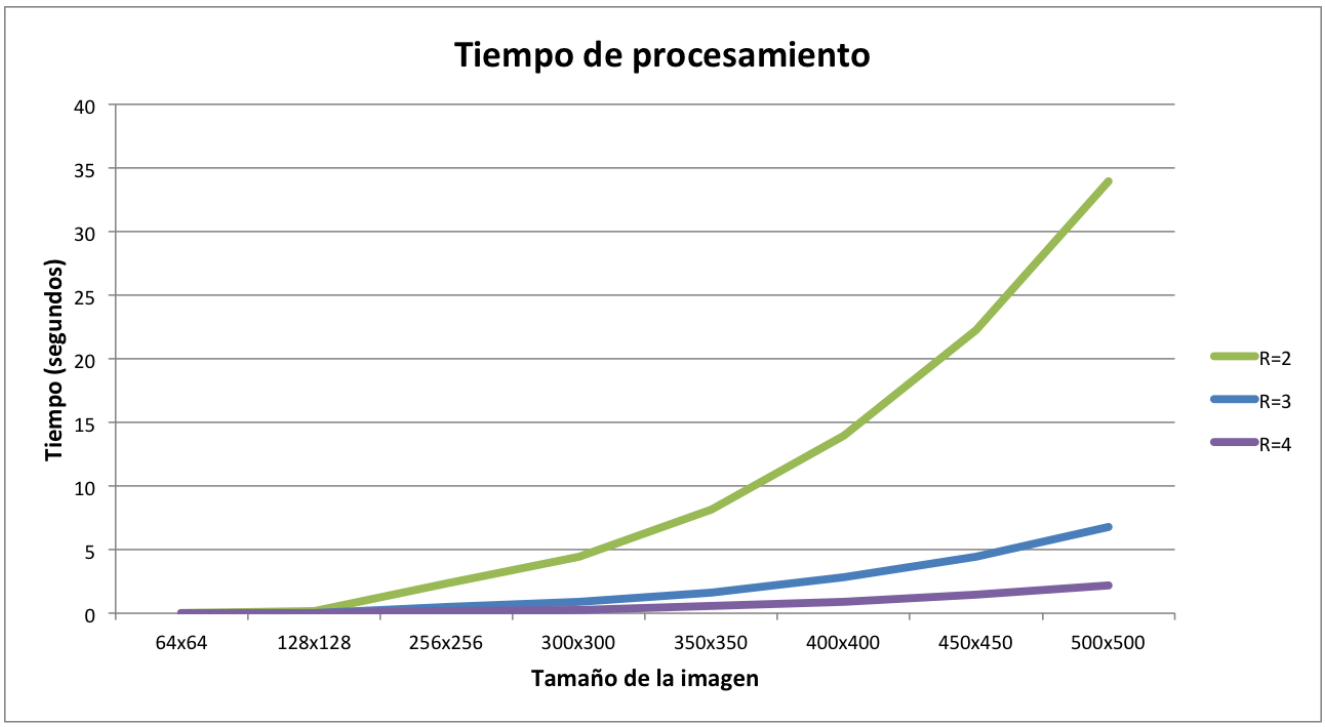


Figura 9: El tiempo se reduce considerablemente cuando agrandamos el factor de reducción, sacrificando calidad en la imágen pero ahorrando tiempo de computo.

Notemos ahora la relación que hay entre el *tiempo de procesamiento* y la calidad obtenida con *factor de reduccion*. En la `figura 8` se puede observar que en los casos de imágenes chicas, de $64 \times 64$ a $256 \times 256$, tener un *factor de reduccion* igual a 1 (es decir, sin reducción) nos da un PSNR varios puntos mayor que reduciendo la imágen, sin embargo para imágenes más grandes la diferencia no es tan significativa. Si tenemos en cuenta el tiempo de procesamiento necesario para esto, en la `figura 9` vemos que utilizar un *factor de reduccion* igual a 3 baja considerablemente el *tiempo de procesamiento*, por lo tanto para imágenes grandes es aceptable utilizar dicho factor dado que el tiempo que se gana es bastante respecto de la poca calidad que se pierde.

En la `figura 9` no graficamos el tiempo de procesamiento cuando no se la reduce porque la diferencia en los valores era muy grande y no se podía apreciar la diferencia entre los otros factores.

Pero como los tiempos de las pruebas los habíamos tomado, los mostramos en esta tabla a parte.

*Tiempo (segundos) para el algoritmo con factor de reducción 1*

| Tamaño | Tiempo (segundos) |
|---|---|
| 64x64 | 00.148 |
| 128x128 | 02.36 |
| 256x256 | 37.21 |
| 300x300 | 69.82 |
| 350x350 | 128.00 |
| 400x400 | 458.00 |
| 450x450 | 1020.00 |
| 500x500 | 11405.00 |

Sin embargo, sacamos provecho de la técnica de reducción y decidimos probar nuestra implementación en una imagen de 2500x2500, anticipándonos del tiempo que iba a demorar utilizamos distintos factores de reducción (6, 8, 10, 16). En este caso tuvimos que sacrificar calidad ya que con una imagen tan grande no nos alcanzaba la memoria.

Por las pruebas anteriores, sabíamos que una imagen de 500x500 tardaba por lo menos 20 minutos, entonces calculamos el factor mínimo de la siguiente manera

$$\frac{2500}{500} < r \tag{3}$$

A partir de los factores encontrados, analizamos los tiempos y la calidad de los resultados.

*Tiempo (segundos) para la imagen de 2500x2500 con ruido*

| Factor | PSNR | Tiempo (segundos) |
|---|---|---|
| 6 | 20.89 | 519 |
| 8 | 19.99 | 83 |
| 10 | 19.27 | 33 |
| 16 | 17.94 | 5.46 |

La imagen que filtramos le agregamos ruido con el **GIMP**, porque tuvimos un problema con nuestro generador para esta imagen que no pudimos resolver. El PSNR de esta imagen con ruido es de 15,05.

# 4. Conclusiones

Nuestra primera impresión fue que el sistema de ecuaciones era muy grande y requeriría mucho memoria si volcábamos el problema exactamente en la computadora.

Lo que nos motivo es tener que pensar una estructura especial para resolver el problema, dadas las características del mismo. Y no solo fue una estructura sino un debate entre 2 estructuras a las cuales después estuvimos agregando pequeñas optimizaciones para mejorar el algoritmo aún más.

La técnica de **submuestreo** la encontramos fascinante, ya que gracias a ella podemos filtrar imágenes de tamaños muy grandes que de otra manera tardarían años en terminar. Es como una especie de heurística aplicada a imágenes y sistemas lineales de ecuaciones.

Nos sorprendimos al encontrar que la resolución de sistemas lineales de ecuaciones tenían una aplicación real y útil no sólo en el campo académico.

## 4.1. Pendientes

Nos quedo pendiente realizar pruebas con imágenes con otro tipo de ruido, ya sea Multiplicativo, Salt & Pepper, u otro.

Con respecto a **Salt & Pepper** hicimos unas pruebas básicas para demostrar que nuestro filtro no producía mejoras tan importantes como las de ruido **Gaussiano**. Por el contrario a lo que esperábamos, las imágenes filtradas tenían una calidad muy parecida a las provenientes de ruido Gaussiano. Suponemos que esto se debe a nuestra función de saturación que interviene en el filtrado.

# 5. Apéndices

## 5.1. A - Enunciado

Un problema típico que se encuentra al trabajar con imágenes digitales es la existencia de "ruido" en las mismas. En pocas palabras, podemos decir que el ruido ocurre cuando el valor de uno o más píxeles de la imagen, no se corresponden con la realidad. La mayoría de las veces, esto se debe a la calidad del equipo electrónico utilizado para tomar las fotografías, o bien a posibles perturbaciones introducidas al momento de transmitir la información. Un caso muy común de imágenes con ruido son las fotografías satelitales.

Se puede pensar el problema de filtrar una imagen con ruido como la minimización del siguiente funcional:

$$\Pi = \int_\Omega \frac{\lambda}{2} |u - \tilde{u}|^2 + \frac{1}{2} \|\nabla u\|^2 d\Omega, \tag{4}$$

donde $u : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ describe la imagen filtrada y $\tilde{u} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}$ la imagen a filtrar (con ruido). De esta manera, el primer término *pesa* cuanto ruido tiene $\tilde{u}$ y el segundo *pesa* la suavidad de la imagen obtenida. La constante $\lambda$ controla la importancia relativa de los dos términos.

La minimización del funcional de la ecuación (4) da lugar a la siguiente ecuación diferencial:

$$\lambda (u - \tilde{u}) - \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = 0. \tag{5}$$

La solución de la ecuación (5) que representa la imagen filtrada se puede aproximar de manera discreta utilizando el método de diferencias finitas, lo cual conduce al siguiente sistema de ecuaciones:

$$\lambda u_{i,j} - (u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j}) = \lambda \tilde{u_{i,j}} \tag{6}$$

donde ahora $u, \tilde{u} : \Omega \subset \mathbb{Z}^2 \to [0 \dots 255]$ son la versiones discretas de la imagen filtrada y la imagen original, respectivamente. Viendo la imagen $u$ como una matriz, $i, j$ son los índices de fila y columna de cada elemento (píxel) de la matriz, donde el 0 es representado por el color negro y el 255 por el blanco[1].

**Enunciado**

El objetivo principal de este trabajo práctico es implementar un programa para eliminar (o reducir) el ruido en imágenes digitales. Para ello, el programa deberá tomar como entrada una imagen (supuestamente con ruido) y resolver la ecuación (5) por el método de diferencias finitas (resolviendo el sistema de ecuaciones dado por las ecuaciones (6)). Finalmente, el programa deberá devolver la versión filtrada de la imágen. La constante $\lambda$ involucrada en las ecuaciones debe ser un parámetro del programa de manera tal que se pueda luego experimentar con ella.

Adicionalmente, el programa implementado deberá ser capaz de procesar imágenes de gran tamaño. Para ello, se pide implementar una funcionalidad extra que permita reducir las imágenes antes de ser procesadas y que, luego del proceso, revierta esta reducción para lograr así una imagen de las dimensiones originales. El *factor de reducción* a utilizar debe ser un parámetro del programa. Por ejemplo, si se desea procesar una imagen de 5 *megapíxeles*[2], puede ocurrir que el tiempo de proceso necesario exceda lo que uno está dispuesto a esperar, con lo cual sería posible reducir la imagen con un cierto factor de reducción y aplicar el proceso de filtrado a una imagen de menores dimensiones. Obviamente, la salida del programa deberá invertir este proceso para retornar una imagen de dimensiones idénticas a la imagen original. Estos procesos llevan el nombre de *submuestreo* (la reducción) y *sobremuestreo* (la ampliación) y existen muchas formas de realizarlos. La manera de realizarlos para este trabajo queda a criterio del grupo.

Tanto el valor de la constante $\lambda$ como el factor de reducción tendrán un fuerte impacto en la calidad de las imágenes obtenidas. El factor de reducción impactará también en los tiempos de ejecución. Para medir estos impactos, se deberá realizar una experimentación computacional cuyos resultados deberán ser plasmados en el informe del trabajo.

**Experimentación**

Una forma de medir la calidad visual de las imágenes filtradas, 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 imagen 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:

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

[1] Este modelo de filtrado de imágenes se puede extender a imágenes color RGB, repitiendo el proceso descripto para cada componente de color.

[2] Un megapíxel equivale a un millón de píxeles.

donde $MAX_u$ define el rango máximo de la imagen (para nuestro caso sería 255) y ECM es el *error cuadrático medio*, definido como:

$$\frac{1}{N}\sum_{i,j}(u^0_{i,j} - u_{i,j})^2$$

donde $N$ es la cantidad de píxeles de la imagen, $u^0$ es la imagen original y $u$ es la imagen perturbada (o en nuestro caso, la imagen recuperada).

La experimentación propuesta para el presente trabajo práctico consiste en analizar la calidad de las imágenes reconstruídas y los tiempos de ejecución en función de:

- el nivel de ruido en la imagen,
- la constante $\lambda$ y
- el factor de reducción.

Dado que para medir la calidad se requiere contar con la imagen original, se deberán utilizar imágenes *ruidosas* generadas artificialmente (por ejemplo, sumando o restando a los píxeles de la imagen original valores generados aleatoriamente con distribución uniforme).

**Formatos de archivos de entrada**

Para leer y escribir imágenes sugerimos utilizar el formato *raw* binario `.pgm`[3]. El mismo es muy sencillo de implementar y compatible con muchos gestores de fotos[4] y Matlab.

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

## 5.2. B - Cómo compilar y usar el TP

El directorio del TP contiene un Makefile, con lo cual para compilarlo basta sólamente con ejecutar **make**. Los binarios generados son:

- **tp** Lee la imagen, plantea el sistema de equaciones, aplica el algoritmo de Gauss y resuelve el sistema. Luego graba la nueva imagen

  **-l** Lambda.

  **-f** Ruta de la foto con ruido.

  **-o** Ruta de la foto filtrada.

  **-r** Factor de reducción [Entero].

- **psnr** Calcula el PSNR de las fotos pasadas como parámetro.

  **-c** Ruta de la foto sin ruido.

  **-n** Ruta de la foto con ruido.

- **generateNoise** Agrega ruido a la foto

  **-M** es el método a ejecutar y se mapean del siguiente modo:
  - 0 para Salt & Pepper
  - 1 para Gaussian

  **-f** Ruta de la foto a agregar ruido.

  **-o** Ruta de la foto con ruido.

  **-p** Parámetro P para **Salt & Pepper**

  **-q** Parámetro Q para **Salt & Pepper**

  **-r** Factor para **Gaussian**