DEPARTAMENTO DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

"No creo que a él le gustará eso"

Metodos numericos
Primer Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Gastón Zanitti	058/10	gzanitti@gmail.com
Ricardo Colombo	156/08	ricardogcolombo@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción teórica

En este trabajo práctico intentaremos modelar y resolver el problema de una superficie a la que se le aplica calor en ciertos puntos, teniendo como condiciones además que los bordes permanecen a temperatura constante. Para modelar este problema utiizaremos la ecuación del calor:

$$\frac{\partial^2 T(x,y)}{\partial x^2} + \frac{\partial^2 T(x,y)}{\partial y^2} = 0. \tag{1}$$

Con esta ecuación diferencial es posible calcular la temperatura en cualquier punto de la superficie. Sin embargo, dado que queremos resolver el sistema bajo un modelo que no sea continuo, será necesario discretizar la ecuación diferencial de alguna manera adecuada. Para ello, discretizaremos esta superficie en segmentos de superficie discretos, e intentaremos resolver el sistema que se obtenga utilizando la computadora. Parece intuitivo pensar que mientras más pequeños sean los segmentos, el sistema se aproximará mas al sistema continuo, obteniendo así respuestas más proximas al mismo.
Exprensando esto en un lenguaje más riguroso, dados a y b el ancho y el alto de nuestra superficie, respectivamente, h la granularidad con la que discretizaremos, y valiendo que $a = m \times h$ y $b = n \times h$, obtendremos una grilla de $(n+1) \times (m+1)$ puntos (donde el punto $(0,0)$ se corresponde con el extremo inferior izquierdo).
Llamemos $t_{ij} = T(x_j, y_i)$ al valor (desconocido) de la función T en el punto $(x_j, y_i) = (ih, jh)$. La aproximación finita (que es posible gracias a la discretización realizada sobre el sistema) afirma que

$$t_{ij} \;=\; \frac{t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}}{4}. \tag{2}$$

De esta forma, es posible plantear un sistema en donde cada punto esté en función de otros y así resolver todas las ecuaciones nos dará la temperatura en el punto crítico.
Aprovechando la discretización del sistema, y que este es un problema lineal, lo modelaremos como un problema $Ax = b$ sobre el cual aplicaremos diversas técnicas para resolverlo, como eliminación gaussiana o descomposición LU, que nos permitirán resolver las incógnitas de una manera más comoda.
Dado que estamos discretizando un sistema continuo, puede ocurrir que el punto critico (que está definido como el centro exacto de la superficie) puede ocurrir que no coincida con ningún punto de la discretización. En este caso, calcularemos un punto próximo a este que sí esté en la discretización y lo consideraremos el punto crítico. Nuestro razonamiento consta en pensar que para una granularidad apropiada, este vecino es suficientemente cercano y se corresponde con el valor que debería coincidir con la discretización.

1.1. Entrada y salida de los algoritmos

Por una cuestión de simpleza, decidimos estandarizar la entrada y la salida de los programas que realizaremos. De esta manera, todos toman una instancia de un archivo de texto con el siguiente formato:

$$\begin{pmatrix} (a) & (b) & (h) & (d) \\ (x1) & (y1) & (r1) & (t1) \\ (x2) & (y2) & (r2) & (t2) \\ ... & & & \\ (xk) & (yk) & (rk) & (tk) \end{pmatrix}$$

Donde a y b son el ancho y el largo del parabrisas, h el tamaño de la discretización y k la cantidad de sanguijuelas del sistema.
Las k lineas siguientes corresponden a cada una de las sanguijuelas, tal que para la sanguijuela i, xi yi es la posición de la misma, ri es su radio y ti su temperatura.
La salida de todos los algoritmos contendrá un archivo que debe ser indicado por parámetro, donde por cada linea contendrá un indicador de la grilla i, j junto con el correspondiente valor de temperatura.
En el caso de los algoritmos de salvación (definidos más adelante), esta salida corresponde a la solución del problema habiendo quitado la sanguijuela que más reduce la temperatura en el punto crítico.
Además, se mostará por pantalla la información conveniente de cada algoritmo ejecutado.

2. Desarrollo

2.1. Sistema a resolver

Una vez obtenida la discretización de nuestro sistema y las posiciones donde se aplicará calor, planteamos las ecuaciones que nos permitirán resolverlo. Consideremos r al radio de alcance de los puntos donde se aplica calor, T_c su temperatura y C_1, ... , C_k los k puntos donde se aplica calor, con $C_i \in R^2$, $C_i = (x_i, y_i)$.
Sabemos que si $x = 0$ o $y = 0$ o $x = m$ o $y = n$ entonces la temperatura en ese punto será:

$$T(x, y) = -100$$

También que $\forall\, C_j$, $C_j = (x_j, y_j)$ / $\sqrt[2]{(x - x_j)^2 + (y - y_j)^2} \leq r$ la temperatura de ese punto será:

$$T(x, y) = T_c$$

Los puntos restantes, osea todos los (x, y) que no cumplen ninguna de estas condiciones tendrán que ser calculados con la ecuación planteada previamente en la introducción.
Planteandolo mas formalmente, lo que harémos será plantear $n \times m$ ecuaciones con $n \times m$ incógnitas. Cada una de estas ecuaciones representa un punto del parabrisas y cada uno de estos puntos vendrá definido de la manera siguiente:
Sea t_{ij} un punto de la grilla:

1. Si es parte del borde ($i = 0$ o $j = 0$ o $i = m$ o $i = n$) entonces su ecuación es $t_{ij} = -100$ por la temperatura por defecto que toma esta sección.

2. Si es un punto donde existe una sanguijuela, sea k la temperatura de la misma, entonces la ecuación será $t_{ij} = k$.

3. Si en cambio es un punto del interior del plano que no se corresponde con la posición de ninguna sanguijuela, su ecuación será, como mencionamos con anterioridad,

$$t_{ij} \;=\; \frac{t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}}{4}. \tag{3}$$

Con esta noción espacial asumiendo cada punto está condicionado tan solo por sus 4 vecinos, ahora se presenta el objetivo encontrar una manera de almacenar la matriz que sea conveniente y nos permita reducir el espacio en la matriz y el número de operaciones a realizar. En el siguiente apartado describiremos como ordenando las variables de manera inteligente, podremos guardar las incógnitas en una matriz banda.

2.2. Matriz banda

Por lo antedicho, sabemos que todo punto interno del parabrisas puede ser calculado en base a sus cuatro vecinos. Esto es, cualquier punto puede ser calculado en base al punto que se encuentra arriba de el, al punto que se encuentra debajo, al punto a la izquierda y a la derecha del mismo. Si logramos mantener en la matriz de resolución del sistema esos 4 puntos cercanos a la diagonal, podemos utilizar una matriz banda.
Ahora supongamos que tenemos un pequeño sistema parabrisas de 3×3 que queremos resolver. Lo que descubrimos tras un arduo análisis es que si ordenamos cada punto tal que la primera ecuación es para el punto $t_{1,1}$, la segunda para $t_{1,2}$, la tercera para $t_{1,3}$, la cuarta para $t_{2,1}$, etc. se forma una matriz con el siguiente contenido:

$$\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & -4 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\dots & & & & & & & & & & & & & & & & & & & & & & & & \dots
\end{pmatrix}
\begin{pmatrix} x_{0,0} \\ x_{0,1} \\ x_{0,2} \\ x_{0,3} \\ x_{0,4} \\ x_{1,0} \\ x_{1,1} \\ x_{1,2} \\ x_{1,3} \\ \dots \end{pmatrix}
=
\begin{pmatrix} -100 \\ -100 \\ -100 \\ -100 \\ -100 \\ -100 \\ 0 \\ 0 \\ 0 \\ \dots \end{pmatrix}$$

Donde todo $x_{i,j}$ con i o j igual a 0 ó 4 es uno de los bordes del parabrisas.
Entonces la matriz resultante queda banda, con el tamaño de la banda igual a una columna del parabrisas más las dos columnas que representan los bordes (en este caso las columnas 0 y 4).

2.3. Almacenamiento de la matriz banda

Debido a la representación matricial definida previamente donde logramos concentrar los elementos no nulos cerca en la diagonal, pudimos representar esta matriz de $(nxm)x(nxm)$ usando otra estructura de $nx(2m+1)$ elementos, sabiendo que más allá de la banda diagonal de pxq elementos, el resto de la matriz se completa con 0. Esto nos permitirá reducir tanto la complejidad espacial como la complejidad temporal de los algoritmos.
El comportamiento de la matriz banda fue encapsulado en una clase para que fuera más fácil de utilizar y transparente tanto para el usuario como para las clases que utilizan esta matriz para setear u obtener sus elementos, sin que este necesite conocer la manera exacta en la que se almacenan los valores internamente.

2.4. Eliminación gaussiana sobre matriz banda

Como primer forma de resolver el sistema de ecuaciones, se implemento el algoritmo de eliminación gaussiana sobre el que se realizaron algunas mejoras. La idea general del algoritmo se mantiene. En el paso k se realiza la diagonalización de la columna k de la matriz A y todos los elementos en la columna k-esima a partir del $a_{k,k}$ pasan a tener valor 0.
La mejora implementada consta en que, sabiendo que la matriz A es banda, a partir del elemento $a_{k+p,k}$ (donde p es el límite de la banda) los siguientes serán 0 y a partir del elemento $a_{k,k+p}$ también serán 0.
Luego, para la columna k no será necesario chequear hasta el final de la matriz, sino hasta el valor $k+p$.
Lo mismo pasa con las filas. En el gauss estándar, luego de diagonalizar una columna, todos los elementos de esa fila deben ser actualizados. Pero al trabajar con una matriz banda, solo es necesario actualizar los p valores que siguen a la diagonal, ya que tenemos asegurado que todos los valores siguientes son 0.
Expresando de manera más formal, el algoritmo es el siguiente:

Este algoritmo posee una mejor complejidad comparado con el Gauss estándar, siendo la misma de $O(n * p^2)$ con n la cantidad de incógnitas de nuestra matriz, y p el tamaño de la banda.
Una vez concluído el proceso de eliminación gaussiana sobre la matriz banda, utilizamos backwards substitution para resolver las ecuaciones y conseguir así el valor de cada uno de los puntos de calor

TP1 1 void Gauss(matriz A, vector b)

1: Para k=1...n
2: Tomo el elemento $a_{k,k}$ como pivote
3: Para $i = k + 1, ...k + p$
4: Para $j = k + 1, ...i + p$
5: $a_{i,j} = a_{i,j} - a_{k,j} * (a_{i,k}/a_{k,k})$
6: $a_{i,k} = 0$

del plano discretizado. Aquí también es posible realizar una optimización, ya que en cada paso no es necesario avanzar hasta la ultima columna sino hasta la columna donde sabemos que comienza la banda.
De esta manera habremos disminuido la complejidad del backwards substitution de $O(n^2)$ a una menor de $O(n * p)$.
Luego la resolución de este sistema tendrá una complejidad temporal de $O(n * p^2)$.

2.5. Descomposición LU

La segunda forma que desarrollamos para resolver el problema es realizando una descomposicion LU sobre matriz original.

Dado que la descomposicion LU es muy similar al algoritmo de eliminación gaussiana, este tambien puede ser optimizado para ser utilizado en una matriz banda. Las optimizaciones serán similares a las de la eliminación gaussiana. Expresandolo de manera algorítmica, la descomposición LU optimizada será así:

TP1 2 void Gauss(matriz A, vector b)

1: Inicializo una matriz L con unos en la diagonal
2: Para k=1...n
3: Tomo el elemento $a_{k,k}$ como pivote
4: Para $i = k + 1, ...k + p$
5: Para $w = k + 1, ...i + p$
6: $a_{i,j} = a_{i,j} - a_{k,j} * (a_{i,k}/a_{k,k})$
7: $a_{i,k} = 0$
8: $L_{i,k} = a_{k,j} * (a_{i,k}/a_{k,k}\ )$

Luego en la matriz A obtengo la matriz diagonal superior y en el L la diagonal inferior que, puede verse facilmente, también son banda, ya que los valores fuera de la banda nunca son modificados.
Puede verse que siendo el algoritmo muy similar al de eliminación gaussiana, tambien posee la misma complejidad, esto es $O(n * p^2)$.
Como ya habíamos dicho el backwards substitution para la matriz banda tiene una complejidad de $O(n * p)$ y de manera similar el foward substitution tiene la misma complejidad.
Luego la complejidad de este metodo tambien será de $O(n*p^2)$. La ventaja de este metodo con respecto al anterior es que en caso de actualizar el vector b, el costo de volver a obtener los resultados se reduce en $O(n * p)$, ya que solo es necesario realizar un backwars substitution y un foward substitution. Esta ventaja sera muy útil más adelante, cuando queramos invertir una matriz de manera barata para poder utilizar el algoritmo de Sherman Morrison.

2.6. No hay tiempo que perder

En esta sección del trabajo, buscaremos la posibilidad de disminuir la temperatura del punto critico a través de la eliminación de una sanguijuela. Dado que esto afecta el sistema de ecuaciones, en primera instancia será necesario recalcular el sistema completo otra vez, lo que podría resultar muy costoso.
Plantear otra vez las ecuaciones, ahora con esta variación, y volver a calcular las incógnitas mediante el algoritmo de eliminación gaussiana tomaría otra vez complejidad $O(n * p^2)$. Teniendo en cuenta que es necesario calcular el sistema completo y luego una vez eliminando cada una de las sanguijuelas pegadas al parabrisas para saber si existe la posibilidad de salvarlo, hace que esta opción no sea la más adecuada, sobre todo en aquellos problemas donde el parabrisas presenta una gran cantidad de sanguijuelas.
Para esto, entonces, hacemos uso de una variación de la fórmula de Sherman–Morrison, evitando replantear todo el sistema desde cero, permitiendo reutilizar información de la ejecución con todas las sanguijuelas y disminuyendo la complejidad a una más adecuada.
Comencemos definiendo de manera más adecuada la implementación más simple.

2.7. Implementación simple

Como ya hemos adelantado, la idea detras de este algoritmo es la de quitar una sanguijuela del sistema, aplicarle eliminación Gaussiana a la matriz obtenida y utilizar backwads substitution para obtener la solución deseada.

Utilizando pseudocódigo, el procedimiento puede expresarse de la siguiente manera:

TP1 3 void Ultimo_disparo()

1: Para cada sanguijuela
2: La quito de la matriz A
3: Quito su b en 0
4: Aplico $Gauss(A, b)$
5: Aplico $Backwards\_substitution(A, b)$
6: Restablezco los valores originales en la matriz A y paso a la siguiente

Luego, siendo la cantidad de sanguijuelas w, el tamaño de la matriz de resolución n y el tamaño de la banda p. La complejidad de este algoritmo será $O(w * ((n * p^2) + (n * p)))$ que es lo mismo que $O(w * (n * p^2))$

Ahora intentaremos mejorar esta complejidad a través de la utlización de la fórmula de Sherman-Morrison.

2.8. Implementación Sherman-Morrison

Sean A la matriz original del problema y A^* la matriz modificada sin una de sus sanguijuelas, queremos resolver $A^* = A + uv^t$ donde uv^t representa la matriz que introduce los cambios debido a la modificación del sistema. Sea entonces $(A^*)x = b$. Resolvemos las siguientes ecuaciones, $Ay = b$ y $Az = u$ y obtenemos la nueva x a partir del siguiente cálculo:

$$x = y - \frac{z(v^t y)}{1 + v^t z}. \tag{4}$$

Así entonces, podemos obtener los resultados a partir de multiplicaciones vectoriales de complejidad lineal y evitamos volver a usar el algoritmo de eliminación gaussiana de complejidad cúbica.

Formalizando el algoritmo:

TP1 4 void Ultimo_disparo_Sherman_Morrison()

1: Realizo la descomposición LU de A
2: Para cada sanguijuela
3: Modifico el valor de b en el lugar de la sanguijuela
4: Calculo x de la manera vista arriba
5: Restablezco los valores originales en b y paso a la siguiente sanguijuela

Este algoritmo tendrá la siguiente complejidad:

1. Descomposición LU: $O(np^2)$
2. Luego para cada sanguijuela:
   a) Modifico el valor de b en el lugar de la sanguijuela: $O(1)$
   b) Calculo x utilizando el procedimiento mencionado anteriormente: $O(n * p)$
   c) Restablezco los valores originales en b: $O(1)$

Luego esto da una complejidad de $O(np^2 + wnp)$

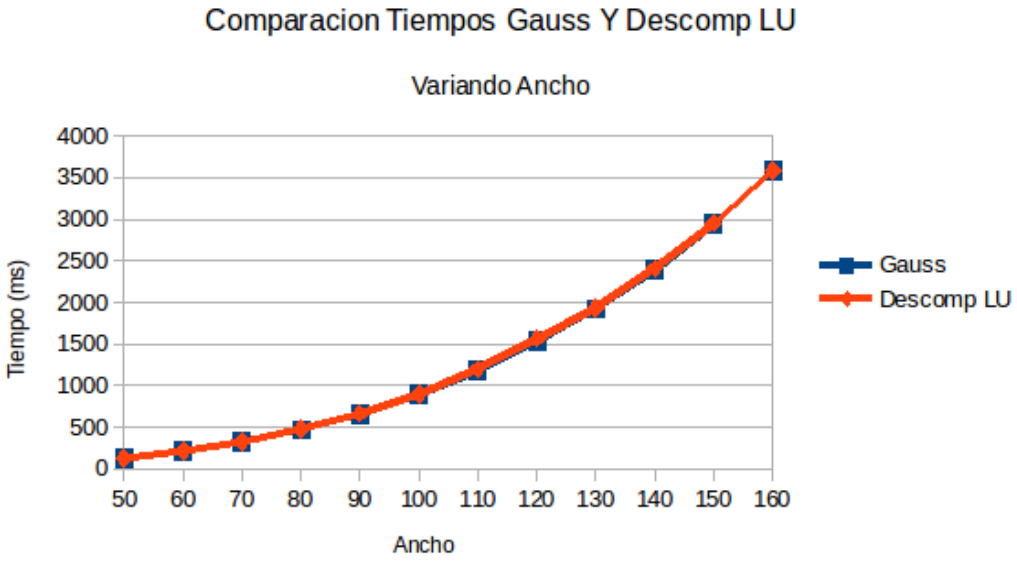
3. Experimentación

3.1. Análisis de tiempos en función de los parámetros de entrada

En esta sección analizaremos de manera experimental como varían los tiempos de ejecución de los algoritmos descriptos al variar el largo y el ancho de la matriz y la cantidad de sanguijuelas del sistema.

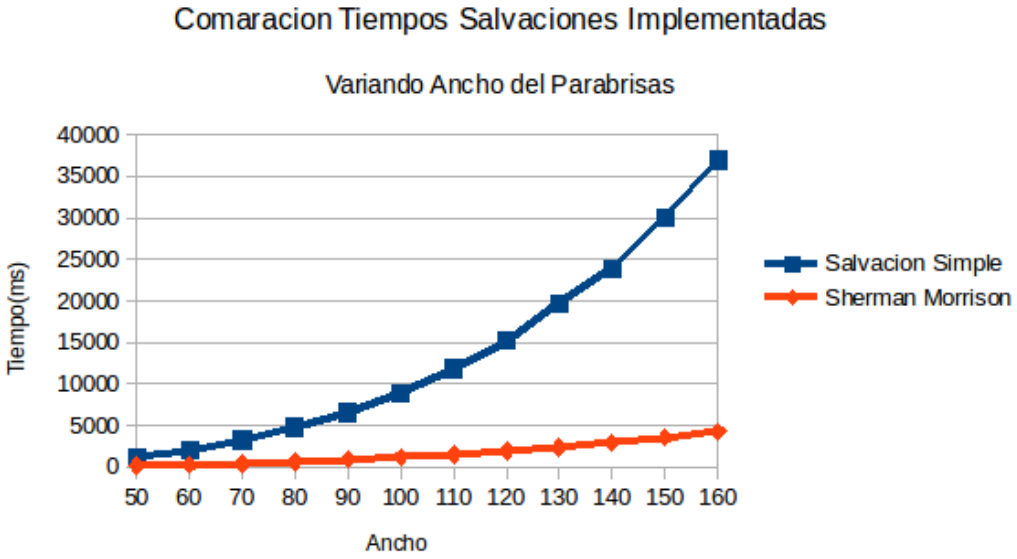
3.1.1. Ancho en función del tiempo

Para comenzar, tomaremos un parabrisas con 50 sanguijuelas tal que estas solo afecten un punto de la discretización, y para una granularidad fija de 1,0 iremos variando el largo del parabrisas. De esta manera, comenzaremos con un parabrisas de 50×50 luego uno de 60×50 y asi aumentando de manera lineal ambos parámetros hasta llegar a un parabrisas de 100×50. Resolveremos cada uno de estos sistemas utilizando ambos metodos implementados (Gauss y Descomposición LU). Los resultados obtenidos pueden verse en el siguente gráfico:



Sabemos que al aumentar el largo del parabrisas de manera lineal, aumentará también de manera lineal el número de ecuaciones en nuestra matriz de resolución. Lo que puede observarse en este gráfico es que con un aumento lineal del largo del parabrisas, el tiempo de ejecución aumenta de manera semi-lineal. Esto era esperable ya que sabemos que tanto el gauss como la descomposición LU tienen una complejidad igual a $O(n * p^2)$, y dado que en nuestro modelado, utilizamos el largo del parabrisas para definir el tamaño de la banda en la matriz de resolución (es decir p), resulta lógico que al aumentar el largo, se obtuviera un aumento casi cuadrático en el tiempo de ejecución.
Además, en este gráfico se pueden observar que tanto los tiempos de la factorización LU como la de gauss son similares.
Ahora, utilizando la misma familia de parabrisas descripta anteriormente, veremos como se comportan ambos métodos de salvación.

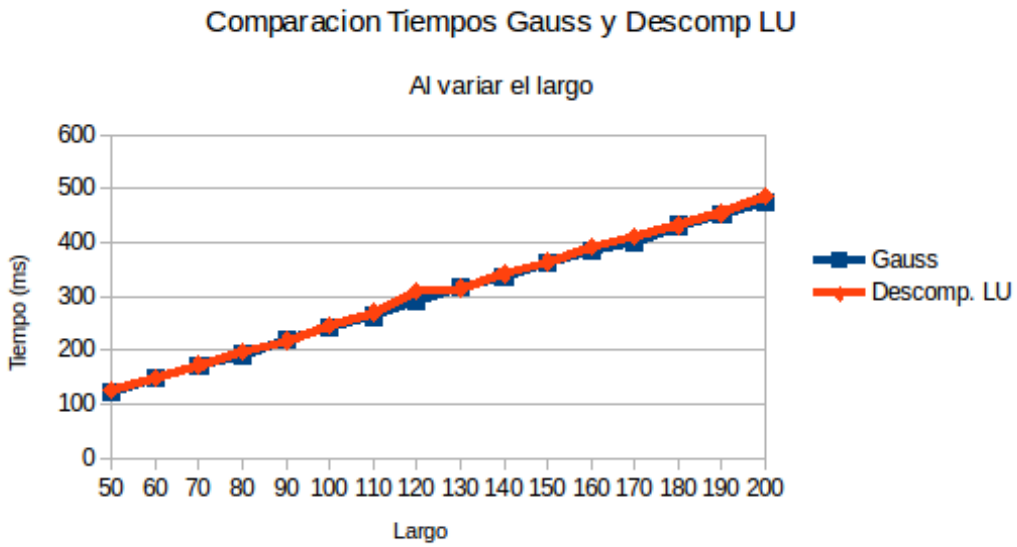
Dado que nos aseguramos que cada sanguijuela solo toque un punto de la discretización, nos aseguramos que podremos utilizar el metodo de Sherman-Morrison. Para estos algoritmos, el gráfico es el siguiente:



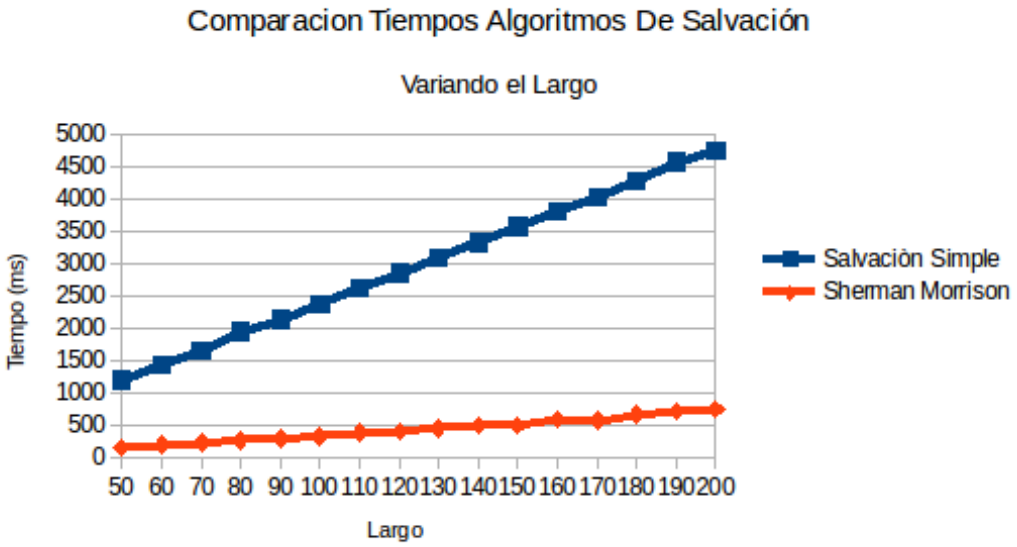
Como era de esperar, el algoritmo que utiliza la optimización de Sherman-Morrison es mucho más rapido y escala mejor que la versión simple que debe calcular todo el sistema desde cero, ademas el Sherman-Morrison realiza operaciones sobre vectores y escalares haciendo la diferencia en el modo de comparacion respecto al método anterior.

3.1.2. Largo en función del tiempo

Ahora, analizaremos que sucede dejando fijo el ancho y variando el largo del parabrisas. Las condiciones son las mismas que en el test anterior, solo que ahora el ancho permaence constante igual a 50 y se varía el largo de 50 a 100.



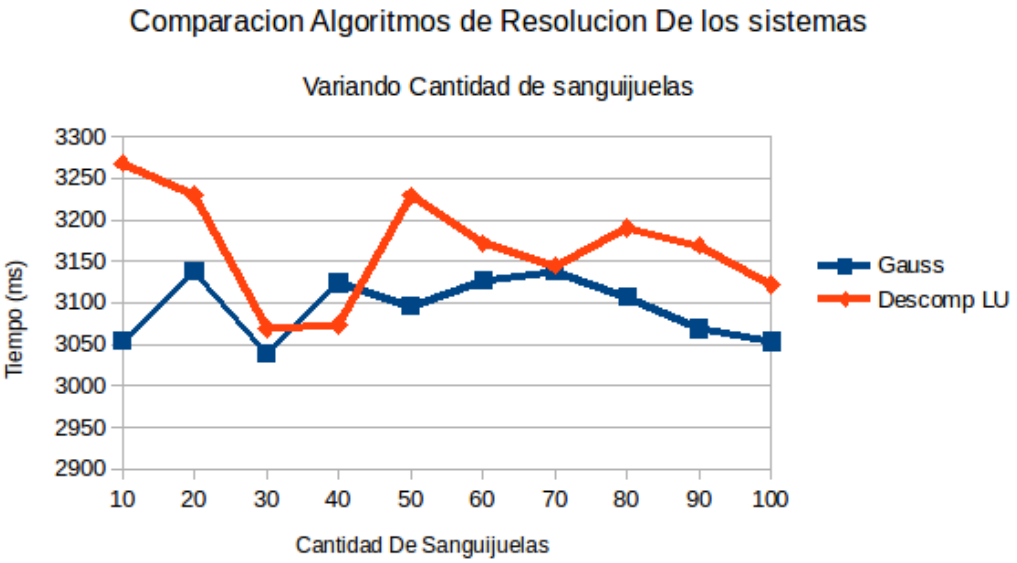
En este caso los tiempos de ejecución crecen de manera estrictamente lineal. Esto se debe que a diferencia del ancho, el largo no interviene en el cálculo del tamaño de la banda de la matriz de resolución. Luego, al aumentar el largo, solo aumenta la cantidad de incógnitas n.
Aplicando el mismo experimento para los dos métodos de salvación:



Al igual que lo dicho en la sección anterior, aqui también se pueden ver las ventajas de utilizar Sherman-Morrison. Además en este caso también puede apreciarse el comportamiento lineal de ambos algoritmos al variar el largo del sistema.

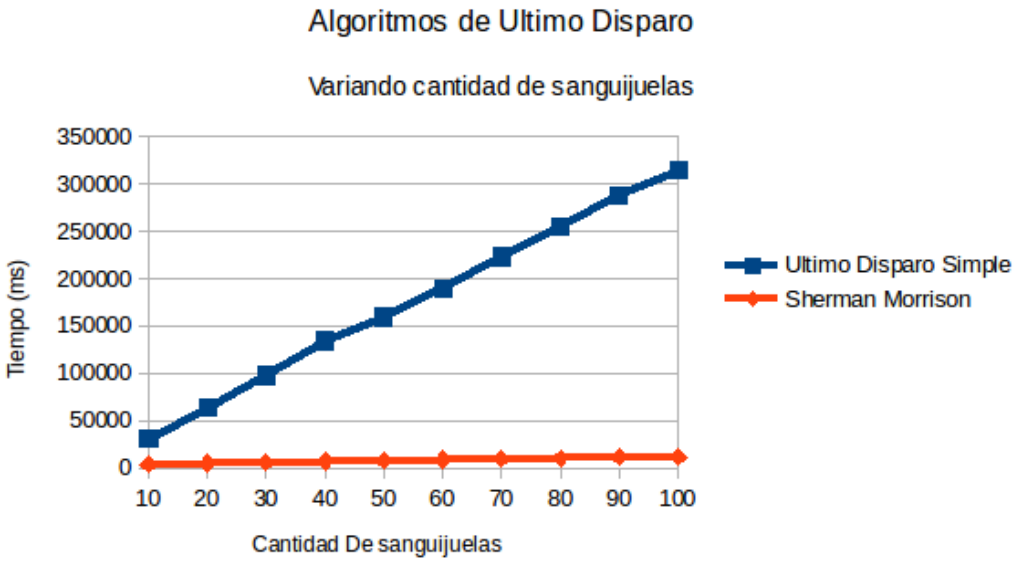
3.1.3. Cantidad de sanguijuelas en función del tiempo

Para el siguente experimento, variamos la cantidad de sanguijuelas y dejamos fija tanto la granularidad como el largo y el ancho del parabrisas. Nuevamente, por una cuestión de simplicidad, las sanguijuelas solo afectan un punto de la discretización. Para el experimento tomamos un parabrisas de 100×100, una granularidad igual a 1,0, y variamos la cantidad de sanguijuelas desde 10 hasta 100. Resolviendo el sistema con el algoritmo de Gauss y Descomposicion LU, se obtuvo el siguiente grafico.



Como vemos, no se muestra ningún patrón visible al modificar la cantidad de sanguijuelas del sistema. Esto es porque la cantidad de incógnitas continúa siendo la misma.

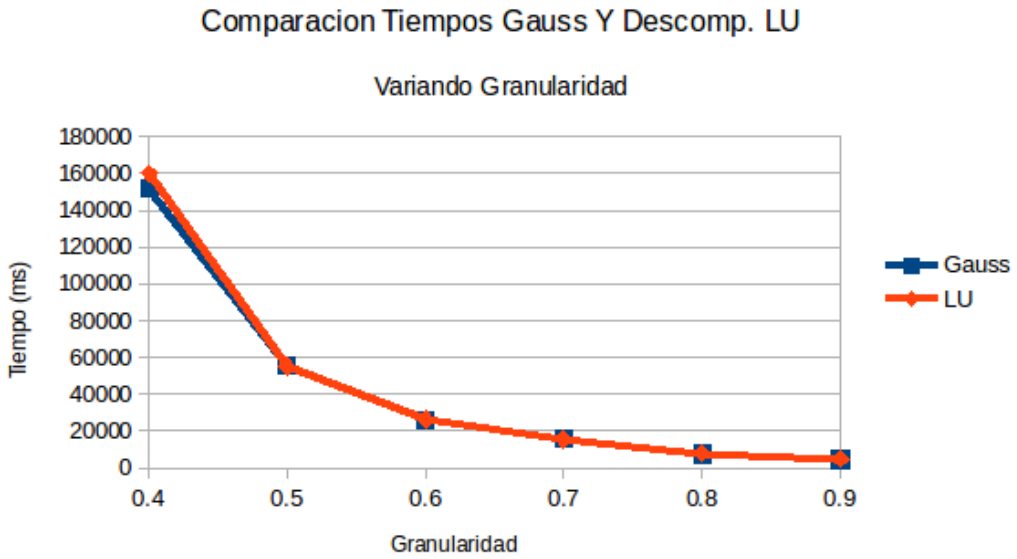
Ahora, utilizando el mismo experimento para el problema del último disparo, resolviendo este parabrisas con ambos algoritmos, se obtuvo este gráfico:



En este gráfico se puede apreciar como si bien ambos algoritmos dependen de manera lineal de la cantidad de sanguijuelas, el algoritmo de Sherman-Morrison presenta una mejor velocidad y una mejor escalabilidad.

3.1.4. Granularidad en función del tiempo

Por ultimo, veremos como afecta variar la granularidad de la discretización para ver como se ve afectada la performance. Para este experimento se dejan fijos el largo y el ancho, iguales a 100, la cantidad de sanguijuelas iguales a 5 y se varía la granularidad desde 0,4 hasta 0,9, aumentando de 0,1 en cada paso. Se obtiene lo siguiente:

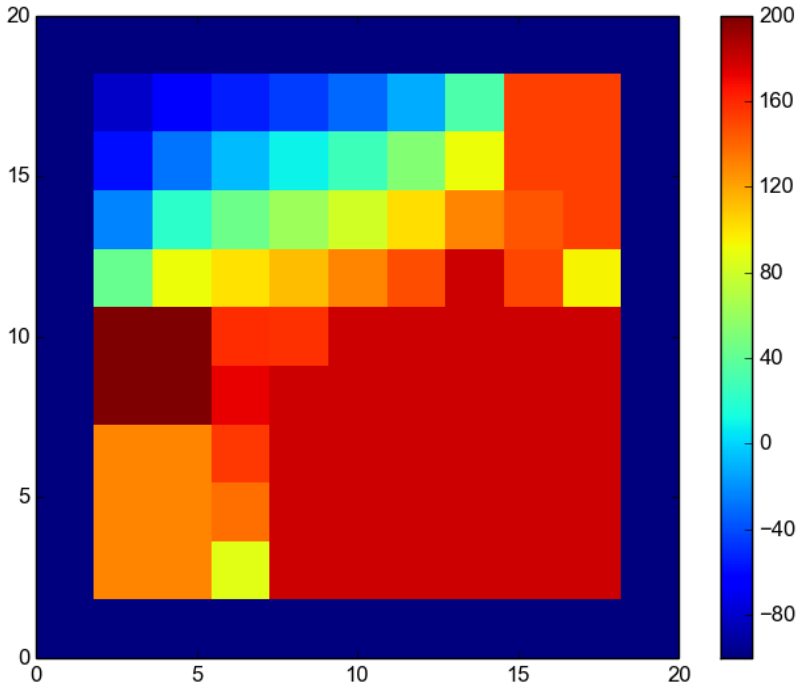


En el gráfico puede observarse que una disminución lineal de la granularidad produce un aumento cuadrático en el tiempo de ejecución. Esto se debe a que tanto la cantidad de filas y la cantidad de columnas viene dado por el largo/ancho del parabrisas, dividido por granularidad. Dado que el tamaño de nuestra matriz de resolución del problema viene dado por Cantidad De filas $\times$ Cantidad De Columnas esto será lo mismo que (Largo $\times$ Ancho)/granularidad^2. En esta fórmula puede verse claramente que disminuir la granularidad de manera lineal produce un aumento cuadrático en el número de incógnitas de nuestro problema.

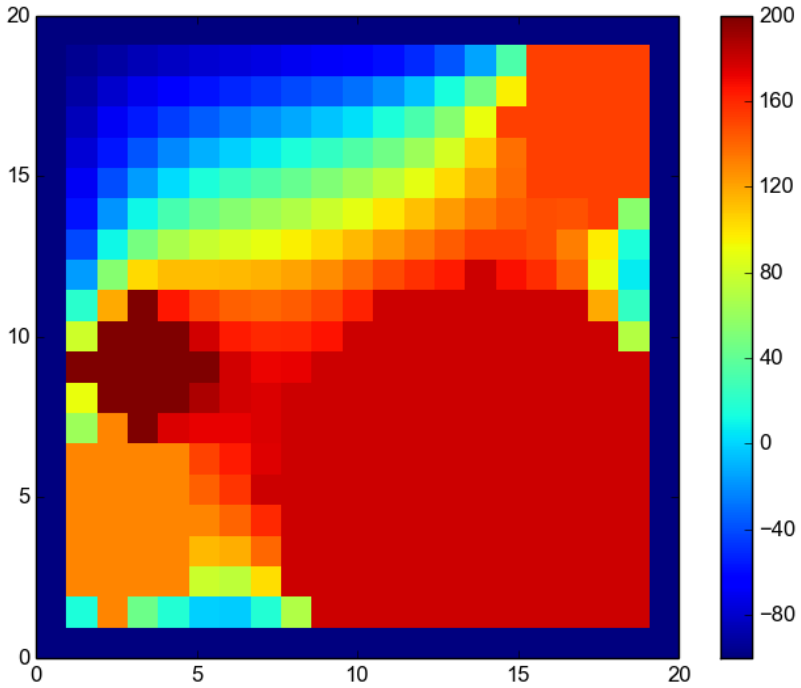
3.2. Análisis de temperatura en función de las discretizaciones

Partiendo de la base de que un aumento en la granularidad permite una mejor representación de las sanguijuelas (son circulos), notamos que con este aumento y mejora en la representación, el problema discreto parece al menos perceptualmente para el ojo humano, volverse continuo. Teniendo en cuenta esta información, queda claro como una baja granularidad impacta directamente sobre la precisión de los resultados (a expensas, como vimos con anterioridad, de los tiempos de cómputo).
En el siguiente experimento, mostramos como varían los resultados al variar la granularidad:

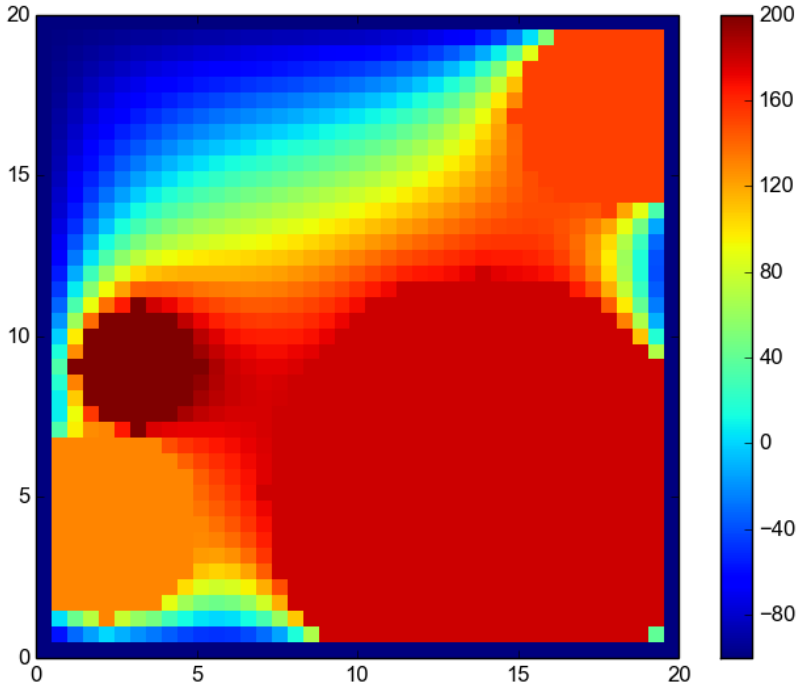
- Matriz 20×20, granularidad 2.



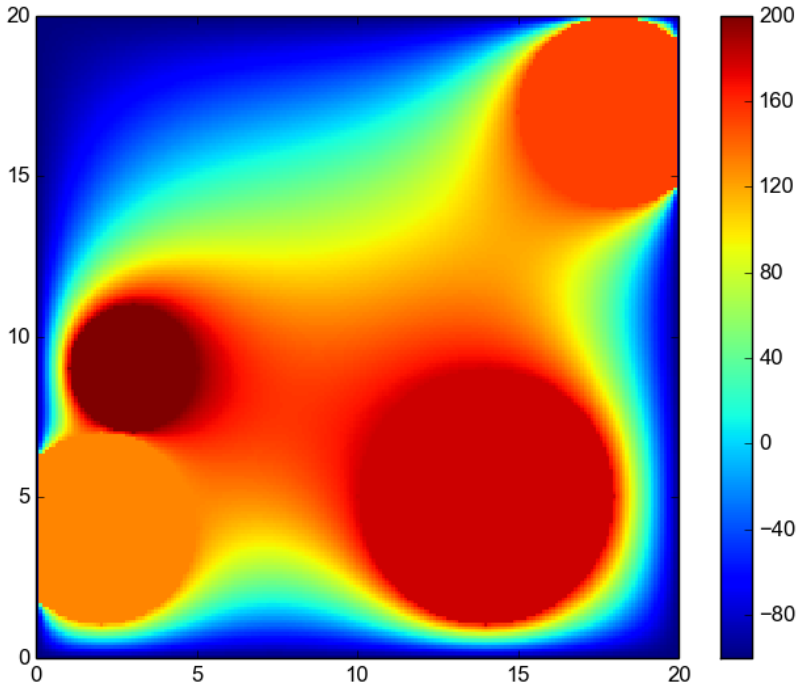
- Matriz 20×20, granularidad 1.



- Matriz 20×20, granularidad 0,5.



- Matriz 20×20, granularidad 0,1.



Puede notarse a simple vista que una granularidad más pequeña produce resultados que aproximan más exactamente a lo que uno creería es el resultado real de la ecuación diferencial de la que partimos en la introducción, ya que para una granularidad de 2 pueden verse grandes bloques discretos, pero para la discretización de 0,1 puede verse que estos bloques desaparecen y puede notarse una cierta s̈uavidad¨ en el resultado obtenido, propio de una función continua y derivable. Esto puede verse en el gráfico ya que la tonalidad va oscureciendo hasta llegar a rojo generando circulos grandes, de ser chicos estaríamos viendo un pico en la función y no sería derivable.
Además, al disminuir la granularidad, está claro que la cantidad de incógnitas cercanas al punto crítico aumenta, por lo que, en caso de no poder tomarlo de manera exacta, al menos podremos tomar un vecino muy próximo a este por lo que tendemos a pensar que la presición del resultado tambien aumentará por ese lado.

4. Discusión

4.1. Manipulación de estructura interna de la matriz

Al utilizar una estructura de menor espacio para representar la matriz del sistema, debimos lidiar con problemas de indexación. Gracias a una abstracción útil de la matriz bajo la clase $MatrizB$, entregando los métodos públicos $getVal$ y $setVal$, que toman indices de la matriz original y los mapean a la respresentación interna que consta de un vector bidimensional, el problema pudo ser mitigado.

4.2. Eficiencia temporal

La resolución de este sistema de ecuaciones implica una gran cantidad de operaciones aritméticas intervinientes. Trabajamos entonces en técnicas para disminuir el costo computacional sobre el algoritmo base de eliminación gaussiana a través de la explotación de características particulares del problema. Algunas técnicas, como la factorización LU, permiten una resolución más rápida del problema y funcionan para cualquier entrada. En cambio, otras técnicas aplican a un subconjunto de las posibles entradas.

Para la detección de sanguijuelas que modifican levemente el sistema utilizamos la fórmula de Sherman-Morrison, obteniendo la variación del sistema de modo más eficiente que en el caso general que requiere de rehacer los cálculos de eliminación gaussiana.

Otra técnica que pensamos y finalmente no implementamos para detectar si eliminar una sanguijuela salva el parabrisas en el caso en que exista una sanguijuela cuyo radio de acción contiene al punto crítico es el siguiente:

1. Si la sanguijuela posee una temperatura menor a 235, eliminando cualquier otra el sistema va a seguir debajo de los 235 grados.
2. Si la sanguijuela se encuentra a una temperatura igual o mayor a 235, probamos eliminarla.
3. Si eliminando la sanguijuela se baja de los 235 grados, eliminandola logramos salvar el parabrisas, si no es cierto, no hay ninguna sanguijuela que pueda ayudar ya que la sanguijuela recién probada ejerce una temperatura directamente sobre el punto crítico.

5. Conclusiones

5.1. Ventajas en el uso de una matriz banda

Como ya discutimos en el desarollo, pudimos plantear el problema a resolver como una matriz banda. Gracias a esto es que pudimos realocar los valores en un espacio físico muy por debajo de lo que la representación matricial de un caso general requiere, ahorrando memoria, ya que fue suficiente con almacenar los valores dentro de la banda, y ahorrando tiempo de cómputo, ya que como también discutimos en la sección de desarrollo, nos limitamos a aplicar los algoritmos en la banda, obteniendo complejidades teóricas más pequeñas que aquellas que se obtienen con el algoritmo estándar.

5.2. Discretización de un problema continuo

En el apartado de experimentación pudimos ver que tomando un valor de discretización muy grande, el problema que comenzo siendo continuo, pasa a ser discreto, pudiendose ver a simple vista los bloques que fueron discretizados. A medida que la discretización se vuelve mas pequeña el problema empieza a tener un comportamiento más continuo (aunque sea de manera perceptual, ya que la solución sigue siendo discreta), lo que sugiere que aumentar la discretización aumenta la presición de todo el sistema.

Sin embargo esto viene con un tradeoff que resulta que para una discretización mas pequeña, y por lo tanto exacta, el tiempo de cómputo aumenta sustancialmente. Luego a la hora de simular este problema deberá prestarse sumo cuidado a esta relación tiempo-presición dependiendo del problema en particular y el grado de confianza que se desea obtener.

5.3. Pérdida de precisión en el uso de aritmética de punto flotante

Así como la separación de los puntos del sistema queda hasta cierto punto a criterio de quien resuelve el problema, existe una gran limitación debido a la representación de los números de punto flotante en la computadora. Si bien C++ es independiente a la arquitectura, las arquitecturas modernas suelen utilizar el estandar IEEE754 para representar la recta numérica. Sabemos que realizando operaciones de punto flotante perdemos precisión, si bien esta puede ser calculada y acotada.
Entonces es así como la resolución del problema se ve afectado no solo por la reducción de un subespacio $\mathbb{R}2$ en un conjunto finito de puntos, sino también por la impresición propia de la computadora al realizar los cómputos.

5.4. Optimizaciones Algebraicas

Comparando ambos metodos de salvación del parabrisas pudimos ver como utilizar propiedades algebraicas y ordenar las operaciones de manera inteligente permite obtener tanto complejidades teoricas tanto como tiempos experimentales mejores. Más se noto la diferencia en los tiempos para el Sherman-Morisson ya que al realizar el producto de vectores de la manera correcta generamos menor cantidad de operaciones acelerando el cálculo de los valores, por quedarnos con un producto de vector por escalar, cambiando el orden de los productos notamos que generamos una matriz haciendo que la cantidad de operaciones se multiplique.
Luego, podemos concluir que para poder resolver un problema de manera óptima es importante valerse de las propiedades especificas de las matrices con las que se trabaja, así también como la teoría del álgebra lineal.

5.5. Otras aplicaciones

Vemos como este tipo de problema donde el valor de cada elemento de un espacio en 2 dimensiones (o más) depende del valor de elementos cercanos puede aplicarse en diversas áreas, por ejemplo en el procesamiento de imágenes, donde el suavizado en el zoom de una imágen se puede realizar a través del promedio de los vecinos de cada pixel. También es de posible aplicación para la propagación de ondas en distintos medios. Es interesante ver también como algunas técnicas genéricas ayudan a la velocidad de cómputo y luego se pueden realizar optimizaciones dentro del dominio de cada sistema en particular, como fue en nuestro caso Sherman-Morrison para casos donde el sistema varía levemente.

6. Apéndice