



**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. Introduccion teorica	3
2. Desarrollo	4
2.1. Sistema a resolver	4
2.2. Matriz banda	4
2.3. Almacenamiento de la matriz banda	5
2.4. Eliminacion gaussiana sobre matriz banda	5
2.5. Descomposicion LU	6
2.6. No hay tiempo que perder	6
2.7. Implementacion Simple	7
2.8. Implementacion Sherman-Morrison	7
3. Discusion	9
3.1. Manipulación de estructura interna de la matriz	9
3.2. Eficiencia temporal	9
4. Resultados	10
4.1. Análisis de tiempos en función de los parametros de entrada	10
4.1.1. Ancho en función del tiempo	10
4.1.2. Largo en función del tiempo	11
4.1.3. Cantidad de sanguijuelas en función del tiempo	12
4.1.4. Granularidad en función del tiempo	13
4.2. Análisis de temperatura función de las discretizaciones	14
5. Conclusiones	17
5.1. Ventajas en el uso de una matriz banda	17
5.2. Discretización de un problema continuo	17
5.3. Pérdida de precisión en el uso de aritmética de punto flotante	17
5.4. Otras aplicaciones	17
6. Apendice	18

1. Introduccion teorica

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

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

Con esta ecuacion 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 ecuacion diferencial de alguna manera adecuada. Para ello, discretizaremos esta superficie en segmentos de superficie, que afectaran la presicion de las respuestas. Parece intuitivo pensar que mientras mas pequeños sean los segmentos, el sistema discreto mas se asemejará con el continuo, obteniendo así respuestas mas parecidas a este.

Aprovechando la discretizacion del sistema y que este es un problema lineal, lo modelizaremos como un problema $Ax = b$ sobre la cual aplicaremos diversas tecnicas para resolverlo, como eliminacion gaussiana o descomposicion LU, que nos permitiran resolver las incognitas de una manera mas comoda.

Para decirlo mas formalmente, 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 $(x_j, y_i) = (jh, ih)$. La aproximación finita (que es posible gracias a la discretización realizada sobre el sistema) afirma que

$$t_{ij} = \frac{t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}}{4}. \quad (2)$$

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.

Dado que estamos discretizando el sistema, puede ocurrir que el punto critico (que esta definido como el centro del parabrisas) puede ocurrir que no coincida con ningun punto de la discretización. En este caso, calcularemos un punto proximo a este que sí este en la discretización y lo consideraremos el punto critico. Nuestro razonamiento fue que para una granularidad apropiada, este vecino será suficientemente cercano y va a corresponderse con el valor que debería coincidir con la discretización.

2. Desarrollo

2.1. Sistema a resolver

Una vez obtenida la discretización de nuestro sistema y la posición de los puntos de calor en esta versión como se mencionó con anterioridad, planteamos las ecuaciones que nos permitirán resolverlo. Consideremos r al radio de alcance de los puntos de calor producidos por las sanguijuelas, T_c su temperatura y C_1, \dots, C_k los k puntos de calor, con $C_i \in R^2$, $C_i = (x_i, y_i)$.

Sabemos que si $x = 0 \vee y = 0 \vee x = m \vee y = n \Rightarrow T(x, y) = -100$.
También que si $\exists C_j$, $C_j = (x_j, y_j) / \sqrt{(x - x_j)^2 + (y - y_j)^2} \leq r \Rightarrow T(x, y) = T_c$.
Luego, nos falta definir $T(x, y)$ para todos los (x, y) que no son alcanzados por estas condiciones. Ahora plantearemos $n \times m$ ecuaciones con $n \times m$ incógnitas, que serán el sistema que luego resolveremos. Las incógnitas serán los $T(x, y)$, y sus constantes correspondientes a_{ij} .

Sea (α, β) un punto de la grilla:

1. Si es parte del borde, entonces su ecuación será $a_{ij} = -100$ por la temperatura por defecto que toma esta sección.
2. Si es un punto donde existe una sanguijuela, este toma el valor de la constante de temperatura otorgado por esta. $a_{ij} = k$ donde K representa la temperatura pasada como input para dicha sanguijuela.
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,

$$t_{ij} = \frac{t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}}{4}. \quad (3)$$

Teniendo en cuenta esta información, podremos despejar la incógnita para introducir dicha ecuación dentro de nuestra matriz, de la siguiente manera:

$$0 = -4t_{ij} + t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}. \quad (4)$$

logrando así igualar la ecuación a cero.

Con esta noción espacial de que cada punto está condicionado tan solo por sus 4 vecinos, ahora quisiéramos encontrar una manera de almacenarla 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 en los apartados anteriores, sabemos que todo punto 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 él, al punto que se encuentra debajo, al punto a la izquierda y a la derecha del mismo. Si ahora en la matriz de resolución del sistema lográsemos mantener a esos 4 puntos cerca de la diagonal, podríamos utilizar una matriz banda.

Ahora supongamos que tenemos un pequeño parabrisas de 3×3 que queremos resolver. Lo que descubrimos tras un arduo análisis es que si ordenáramos 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. nos quedará algo así:

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

Entonces la matriz resultante queda banda, con el tamaño de la banda igual a una columna del parabisas. (En este caso particular 3).

2.3. Almacenamiento de la matriz banda

Debido a que la representación matricial del problema que atacamos posee una estructura donde los elementos no nulos se concentran en la diagonal, pudimos representar esta matriz de $(n \times m)x(n \times m)$ usando otra estructura de $n \times (2m + 1)$ elementos, sabiendo que más allá de la banda diagonal de $p \times q$ elementos, el resto de la matriz se completa con 0.

Inicialmente utilizamos un arreglo unidimensional para almacenar la estructura, amparandonos en una característica presente en gcc y c99 que permite definir el tamaño de un arreglo en tiempo de ejecución. (Cuando lo corrimos en clang esto fallaba). Si bien esta estructura de datos es muy eficiente, decidimos movernos a la representación en un vector unidimensional, obteniendo mayor comodidad de manipulación. La diferencia en tamaño de la estructura del vector frente al arreglo es despreciable frente al almacenamiento general que requiere el problema.

2.4. Eliminacion gaussiana sobre matriz banda

Como primer forma de resolver el sistema de ecuaciones, se implemento un Gauss sobre el que se realizaron algunas mejoras. La idea general del algoritmo se mantiene, en el paso k diagonalizamos la columna k de la matriz A y todos los elementos en la columna k -esima a partir del $a_{k,k}$ seran 0.

La mejora que implementaremos será que, sabiendo que la matriz A es banda, a partir del elemento $a_{k+p,k}$ (donde p es el limite 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 estandar, 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 seran 0.

Escrito de manera mas formal el algoritmo será el siguiente:

TP1 1 void Gauss(matriz A, vector b)

- ```

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

```

De este algoritmo se desprende que este algoritmo tiene una mejor complejidad comparado con el Gauss standard, siendo la misma de  $O(n * p^2)$  siendo  $n$  la cantidad de incógnitas de nuestra matriz, y  $p$  el tamaño de la banda.

Una vez concluido 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 del plano discretizado. Aquí también es posible realizar una optimización, ya que en cada paso no es necesario ir hasta la última 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  $O(n * p)$ .

## 2.5. Descomposición LU

La segunda forma que desarrollamos para resolver el problema es realizando una descomposición LU a la matriz original. La ventaja de esto es que en caso de actualizar el vector  $b$ , el costo de volver a obtener los resultados se reduce en  $O(n^2)$ .

Dado que la descomposición LU es muy similar al algoritmo de eliminación gaussiana, este también puede ser optimizado para ser utilizado en una matriz banda. Las optimizaciones serán similares a las de Gauss. Expresándolo 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  $k=1\dots n$
  - 3:   Tomo el elemento  $a_{k,k}$  como pivote
  - 4:   Para  $i = k + 1, \dots k + p$
  - 5:     Para  $w = k + 1, \dots i + p$
  - 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.

Puede verse que siendo el algoritmo muy similar al de eliminación gaussiana, también posee la misma complejidad, esto es  $O(n * p^2)$ .

Este algoritmo será utilizado 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 parte del trabajo, buscaremos la posibilidad de disminuir la temperatura del punto crítico a través de la eliminación de una sanguijuela. Dado que esto nos afecta el sistema de ecuaciones, en primera instancia, será necesario recalcular todo el sistema 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 granularidad muy fina. Para esto, entonces, hacemos uso de una variación de la fórmula de Sherman–Morrison, evitando

replantear todo el sistema desde cero, permitiendo reutilizar informacion de la ejecucion con todas las sanguijuelas y disminuyendo la complejidad a una mas adecuada.

Comencemos definiendo de manera mas adecuada la implementacion mas simple.

## 2.7. Implementacion Simple

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

Mas formalizado en pseudocodigo:

---

**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 sera  $O(w * ((n * p^2) + (n * p)))$  que es lo mismo que  $O(w * (n * p^2))$

Ahora intentaremos mejorar esta complejidad a travez de la utlizacion de la formula de sherman morrison.

## 2.8. Implementacion Sherman-Morrison

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

$$x = y - \frac{z(v^t y)}{1 + v^t z}. \quad (5)$$

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 sangijuela:

*a)* Modifico el valor de  $b$  en el lugar de la sangijuela:  $O(1)$

*b)* Calculo  $x$  de la manera vista arriba:  $O(n^p)$

*c)* Restablezco los valores originales en  $b$ :  $O(1)$

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



### 3. Discusion

#### 3.1. 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 índices de la matriz original y los mapean a la representación interna que consta de un vector bidimensional, el problema pudo ser mitigado.

#### 3.2. 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.

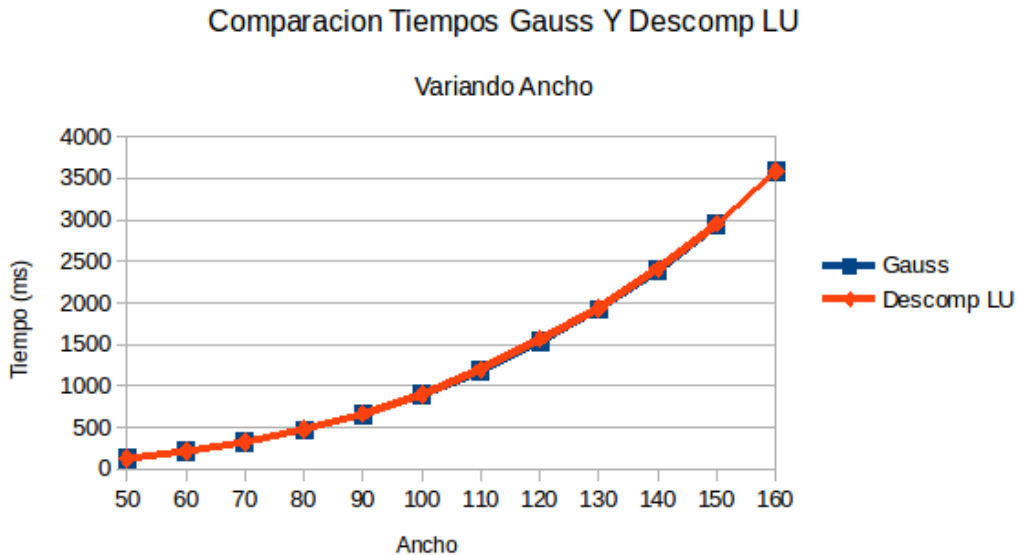
## 4. Resultados

### 4.1. Análisis de tiempos en función de los parametros de entrada

En esta seccion 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.

#### 4.1.1. Ancho en función del tiempo

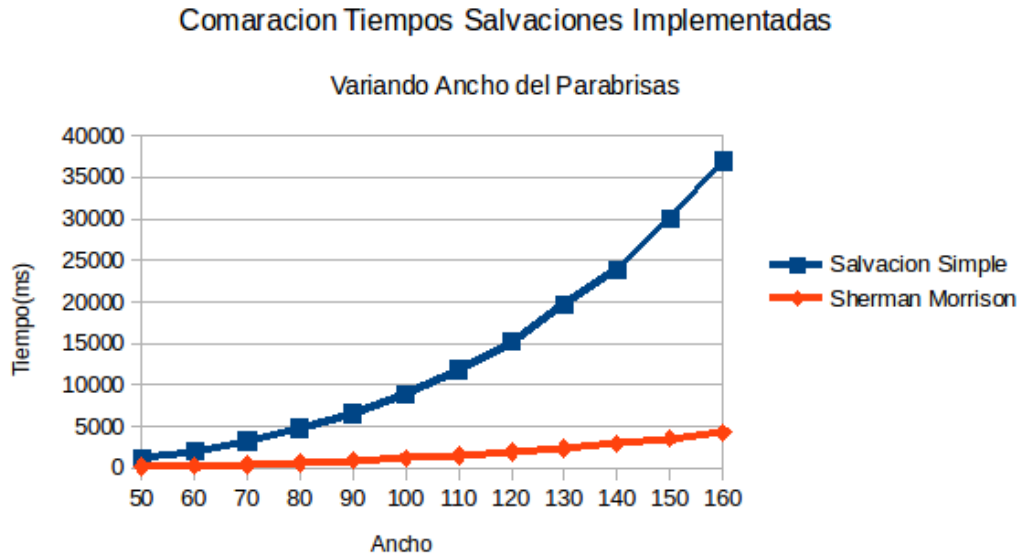
Para comenzar, tomaremos un parabrisas con 50 sanguijuelas, tal que estas solo toquen un punto de la discretización, y para una granularidad fija de 1,0 iremos variando el largo del parabrisas. De esta manera, comenzaremos con un parabrisas de  $50 \times 50$  luego uno de  $60 \times 50$  y así aumentando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



Sabemos que al aumentar el largo de el parabrisas de manera lineal, aumentará de manera lineal el numero de ecuaciones en nuestra matriz de resolución. Lo que puede observarse en este grafico es que con un aumento lineal del largo del parabrisas, el tiempo de ejecución aumenta de manera cuasi-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 (osea  $p$ ), era logico que al aumentar el largo, se obtuviera un aumento casi cuadrático en el tiempo de ejecución.

Ademas en este grafico se pueden observar que tanto los tiempos de la factorización LU como la de gauss son similares.

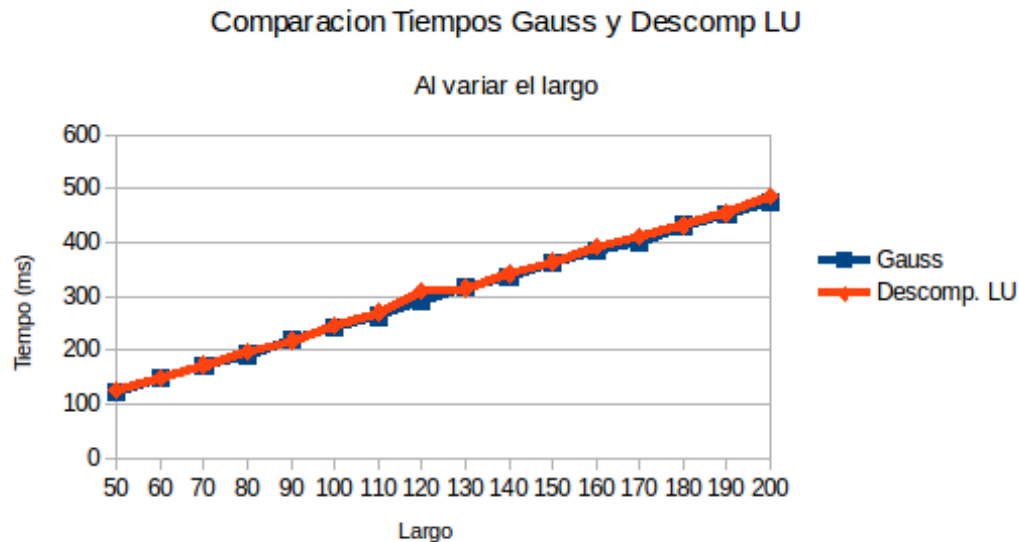
Ahora, utilizando la misma familia de parabrisas descripta anteriormente, veremos como se comportan ambos metodos de salvación. Dado que nos aseguramos que cada sanguijuela solo toque un punto de la discretización, nos aseguramos que podremos utilizar el metodo de Sherman Morrison. Para estos algoritmos, el grafico es el siguiente:



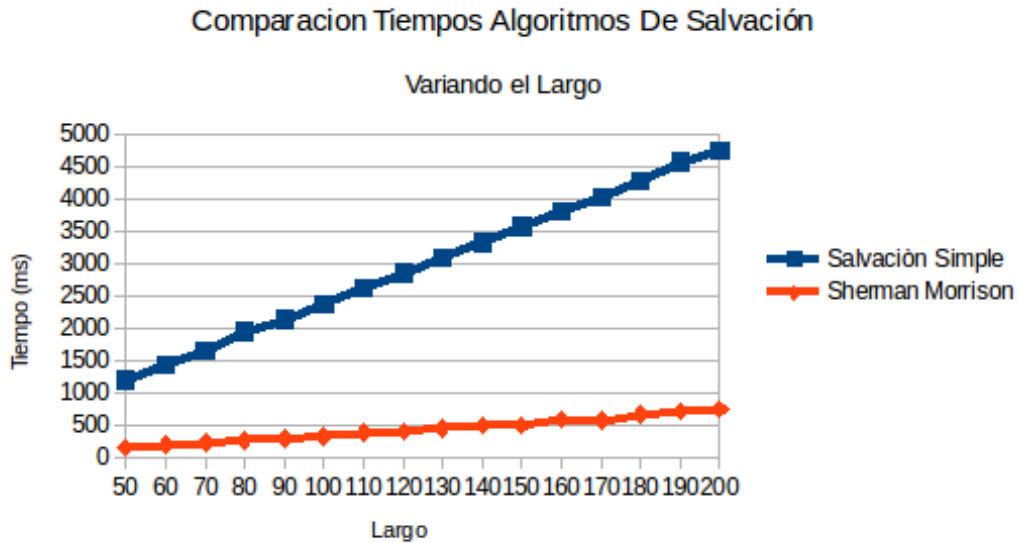
Como era de esperar, el algoritmo que utiliza la optimización de Sherman morrison es mucho mas rapido y escala mejor que la versión simple que debe calcular todo el sistema desde cero.

#### 4.1.2. 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.



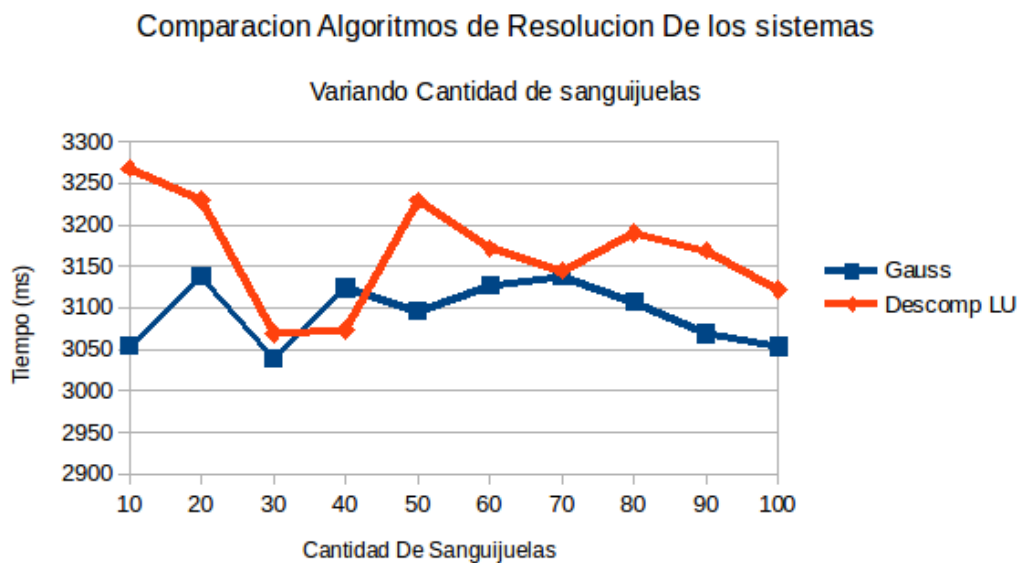
En este caso los tiempos de ejecución crecen de manera estrictamente lineal. Esto se debe que a diferencia del ancho, el largo no interviene en el calculo del tamaño de la banda de la matriz de resolución. Luego al aumentar el largo, solo aumenta la cantidad de incognitas  $n$ . Y aplicando el mismo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



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

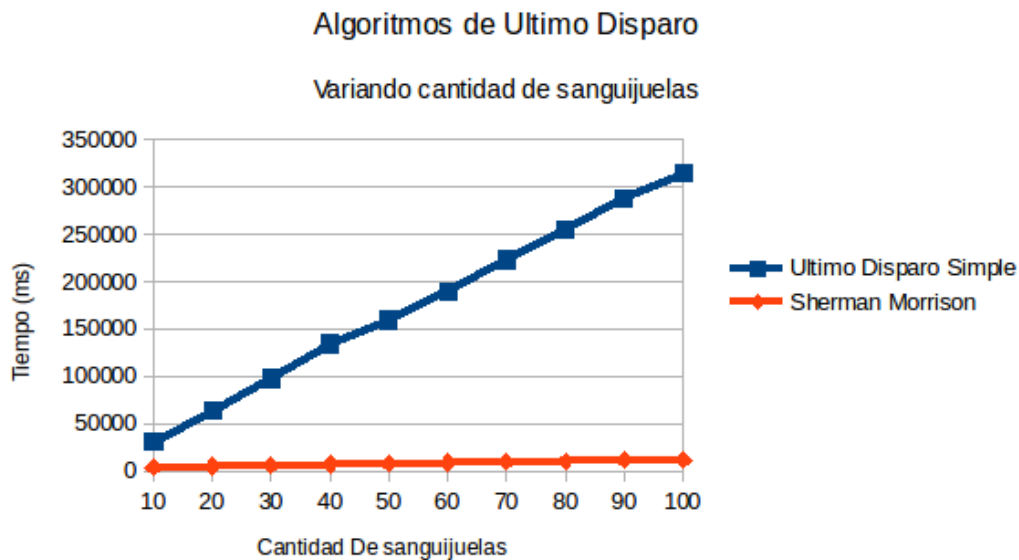
#### 4.1.3. Cantidad de sanguijuelas en función del tiempo

Para el siguiente experimento, variaremos la cantidad de sanguijuelas y dejaremos fija tanto la granularidad como el largo y el ancho del parabrisas. Nuevamente, por una cuestión de simplicidad las sanguijuelas solo tocarán un punto de la discretización. Para el experimento tomamos un parabrisas de  $100 \times 100$ , 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 Descomposición LU, se obtuvo el siguiente gráfico.



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.

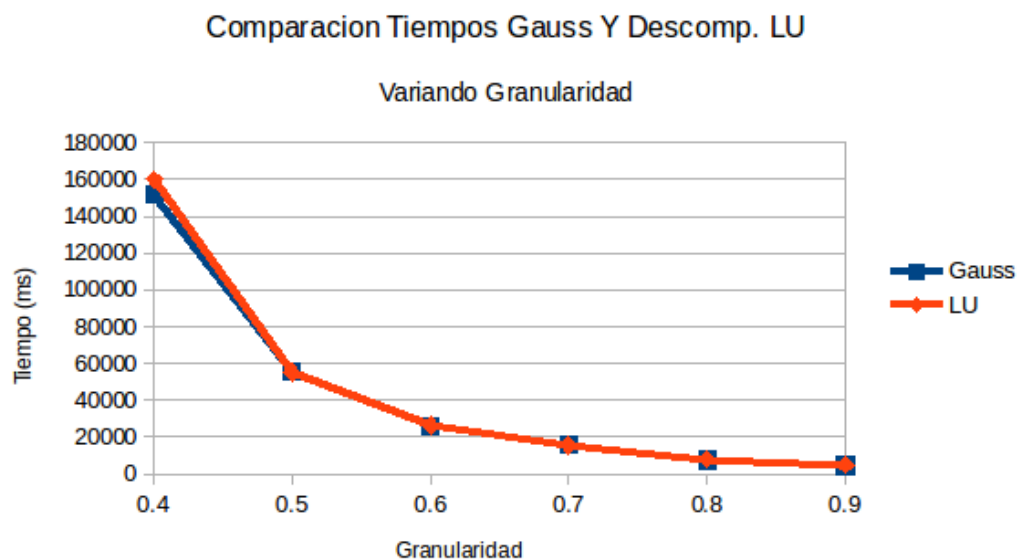
Ahora utilizando el mismo experimento para el problema del ultimo disparo, resolviendo este parabrisas con ambos algoritmos, se obtuvo este grafico:



En este grafico 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.

#### 4.1.4. 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 dejarán fijos el largo y el ancho, iguales a 100, la cantidad de sanguijuelas iguales a 5 y se variará la granularidad desde 0,4 hasta 0,9, aumentando de 0,1 en cada paso. Lo obtenido es lo siguiente:



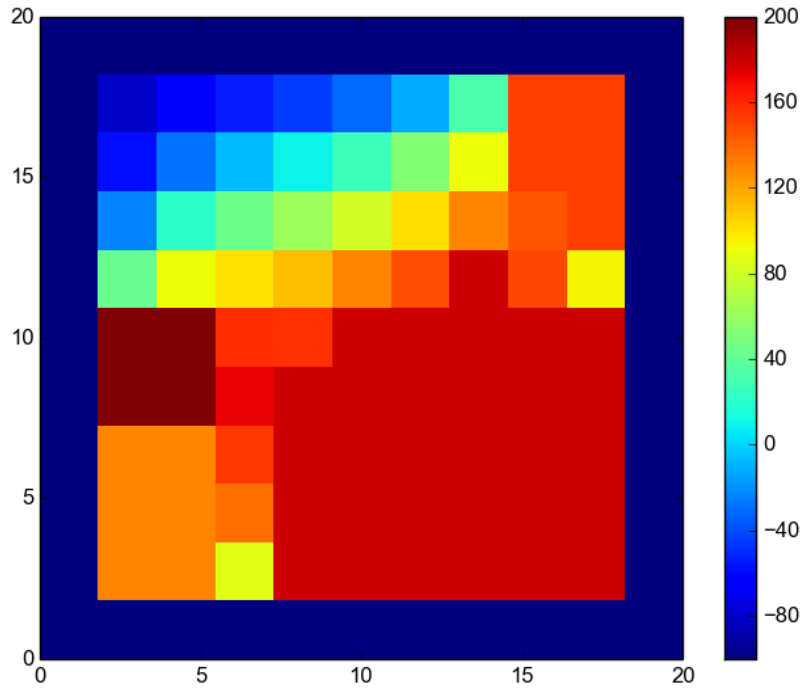
En el grafico puede verse que una disminucion lineal de la granularidad produce un aumento cuadratico en el tiempo de ejecuci3n. Esto se debe a que tanto la cantidad de filas y la cantidad de columnas viene dado por el largo/ancho del paravrisas, dividido por la granularidad. Dado que el tama1o de nuestra matriz de resoluci3n del problema viene dado por Cantidad De filas  $\times$  Cantidad De Columnas esto ser1a lo mismo que  $(\text{Largo} \times \text{Ancho})/\text{granularidad}^2$ . En esta formula puede verse claramente que disminuir la granularidad de manera lineal produce un aumento cuadratico en el numero de incognitas de nuestro problema.

## 4.2. An1lisis de temperatura funci3n de las discretizaciones

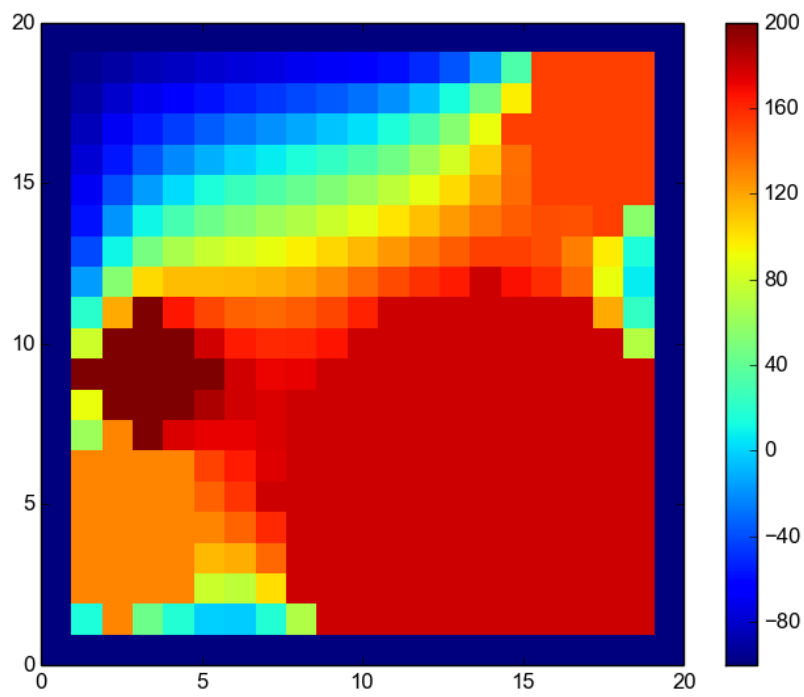
A continuaci3n presentamos el an1lisis y las conclusiones obtenidas de acuerdo a la experimentaci3n realizada. Como pudimos ver en el apartado anterior, una mayor granularidad en la discretizaci3n hace escalar r1pidamente la cantidad de ecuaciones y variables involucradas, con lo que el tiempo de los algoritmos escala r1pidamente.

Partiendo de la base de que un aumento en la granularidad permite una mejor representaci3n de las sanguijuelas (son circulos), notamos que con este aumento y mejora en la representaci3n, las temperaturas del punto cr1tico disminuyen considerablemente. Teniendo en cuenta esta informaci3n, queda claro como una baja granularidad impacta directamente sobre la precisi3n de los resultados (a expensas, como vimos con anterioridad, de los tiempos de c3mputo).

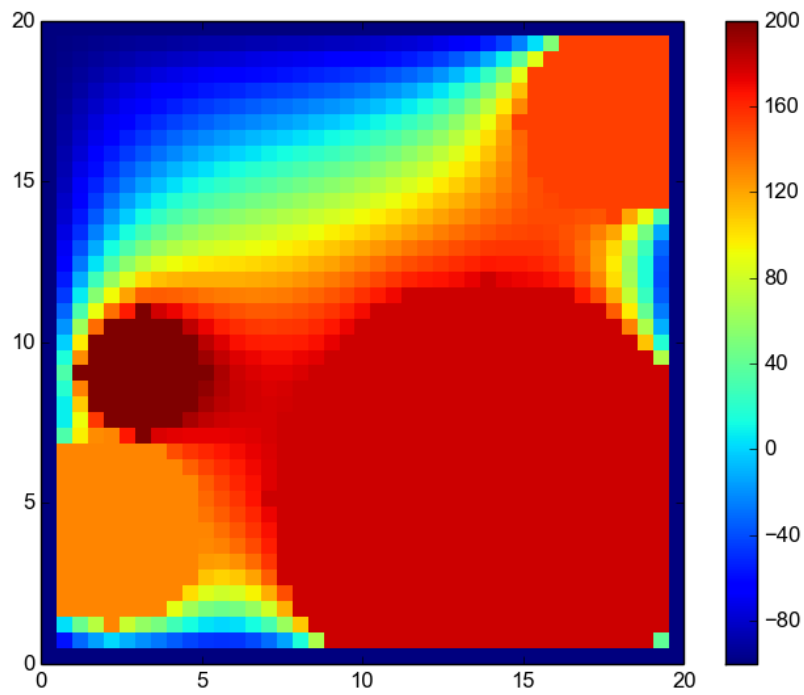
- Matriz  $20 \times 20$ , granularidad 2.



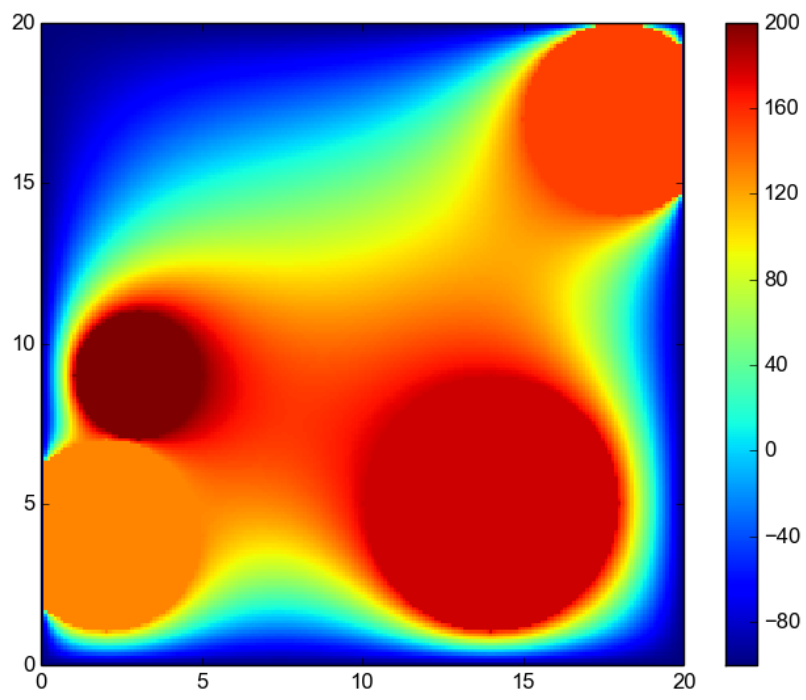
- Matriz  $20 \times 20$ , granularidad 1.



- Matriz  $20 \times 20$ , granularidad 0,5.



- Matriz  $20 \times 20$ , granularidad 0,1.





## 5. Conclusiones

### 5.1. Ventajas en el uso de una matriz banda

Una particularidad del sistema de ecuaciones planteado es que el valor de cada elemento del vector de resolución depende exclusivamente de, como máximo, otros 4 valores que además son cercanos al elemento a resolver, llamando cercano a un elemento que se encuentra a no más de una fila de distancia, dejando una gran cantidad de elementos (la gran mayoría en 0). Este problema en particular plantea una matriz banda con una cantidad de elementos no nulos mucho menor a aquellos que tienen un valor no nulo. Siendo más precisos la matriz tiene  $(n \times m) \times (n \times m)$  elementos, con menos de  $5 \times (m \times n)$  elementos no nulos. Gracias a eso es que pudimos realocar los valores en un espacio físico muy por debajo de lo que la representación matricial de un caso general requiere, mapeando los valores distintos de 0 a la matriz física y los valores nulos siendo entregados sin ser alojados en ninguna matriz.

### 5.2. Discretización de un problema continuo

Durante la resolución del problema pudimos ver que cambiando el  $h$ , es decir, la medida de discretización del continuo del parabrisas, obtenemos diferentes grados de error. A mayor  $h$  esperamos un error más pronunciado. Una posible definición de *Probabilidad* es el nivel de conocimiento sobre el sistema. Análogamente podemos decir que el nivel de discretización de un sistema continuo nos da una idea del conocimiento sobre el mismo.

Por otro lado, una mayor discretización del sistema impacta directamente sobre los tiempos de cómputo en su resolución. Queda entonces a criterio del desarrollador valorar ambos parámetros y balancear entre soluciones más precisas y tiempos de resolución más cortos. Esta elección está condicionada por los problemas acarreados provenientes de la resolución a través de computadoras, que no tienen la posibilidad de representar todos los números de la recta real, generando problemas de precisión.

### 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 estándar 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 imprecisión propia de la computadora al realizar los cálculos.

### 5.4. 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 imagen 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. Apendice