Métodos Numéricos

Primer Cuatrimestre - 2015

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

Trabajo Práctico I

Resolución de Sistemas de ecuaciones lineales utilizando métodos básicos de Álgebra Lineal

Integrante	LU	Correo electrónico
Cuneo, Christian	755/13	chriscuneo93@gmail.com

En este informe voy a desarrollar como solucione el problema planteado por la materia, en el cual se presentaba un gran sistema de ecuaciones (Ax = b), a resolver ya sea usando eliminación gaussiana o factorización LU, con este ultimo tenemos la ventaja que si, el sistema luego se altera en una fila o columna, nosotros podemos solucionarlo nuevamente sin tener que recalcular la factorización, utilizando la formula de Sherman-Morrison.

Eliminación Gaussiana **Factorización LU** **Matriz Banda** **Formula Sherman-Morrison**

Índice

1. Introducción Teórica

La resolución de este trabajo practico tiene como cimientos dos métodos muy importantes del álgebra lineal utilizados para resolver sistemas de ecuaciones, el primero, y el mas conocido, es el método de eliminación gaussiana; el segundo es el de la Factorizacion LU

1.1. Eliminación Gaussiana

La eliminación Gaussiana o eliminación de Gauss-Jordan, es un algoritmo del álgebra lineal para determinar las soluciones de un sistema de ecuaciones lineales. Un sistema de ecuaciones se resuelve por el método de Gauss cuando se obtienen sus soluciones mediante la reducción del sistema dado a otro equivalente en el que cada ecuación tiene una incógnita menos que la anterior. Cuando se aplica este proceso, la matriz resultante se conoce como: "forma escalonada". A continuación mostramos un algoritmo en pseudocodigo para llevar a la practica el proceso básico de eliminación gaussiana que acabamos de describir. En este algoritmo se supone que todos los elementos pivote son distintos de cero:

for $k = 1 \rightarrow (n-1)$ **do**
  for $i = k+1 \rightarrow n$ **do**
    $z \leftarrow \frac{a(i,k)}{a(k,k)}$
    $a(i,k) \leftarrow 0$
    for $j = k+1 \rightarrow n$ **do**
      $a(i,k) \leftarrow a(i,k) - z * a(k,j)$
    end for
  end for
end for

1.2. Derivadas Parciales

Las derivadas parciales están definidas como las derivadas de una función de más de una variable cuando cada una de ellas, salvo la variable de interés son consideradas como fijas durante la derivación. Como las derivadas en una variable, las derivadas parciales están definidas como el límite. Donde U es un subconjunto abierto de $\Re^n$ y $f : U \rightarrow \Re$ una función. Definimos derivada parcial de f en el punto $a = (a_1, ..., a_n) \in U$ con respecto a la i-ésima variable x_i como:

$$\frac{\delta f}{\delta x} = \delta_x f = f_x^{'}$$

Incluso si todas las derivadas parciales existen en el punto a, la función no necesariamente es continua en ese punto. Sin embargo, si todas las derivadas parciales existen alrededor de a y son continuas, entonces la función no sólo es continua sino además diferenciable cerca de a. En este caso, f es una función $C1$.

1.3. Diferencias finitas y ecuaciones diferenciales

Una ecuación diferencial es una ecuación en la que intervienen derivadas de una o más funciones. Dependiendo del n'umero de variables independientes respecto de las que se deriva, las ecuaciones diferenciales se dividen en:

- Ecuaciones diferenciales ordinarias: aquellas que contienen derivadas respecto a una sola variable independiente.
- Ecuaciones en derivadas parciales: aquellas que contienen derivadas respecto a dos o más variables.

La resolución de ecuaciones diferenciales es un tipo de problema matemático que consiste en buscar una función que cumpla una determinada ecuación diferencial. Se puede llevar a cabo mediante un método específico para la ecuación diferencial en cuestión o mediante una transformada. Dentro de los

métodos para la resolución de ecuaciones diferenciales se encuentra el Método de Diferencias Finitas, el cual es un método de carácter general que permite la resolución aproximada de ecuaciones diferenciales en derivadas parciales definidas en recintos finitos. Es de una gran sencillez conceptual y constituye un procedimiento muy adecuado para la resolución de una ecuación bidimensional como la ecuación del calor que planteamos previamente. Consiste en aproximar a la solución deseada a través ecuaciones diferenciales finitas para aproximar derivadas.

1.4. Factorización LU

La factorización LU de una matriz es una factorización que resume el proceso de eliminación gaussiana aplicado a la matriz y que es conveniente en terminos del número total de operaciones de punto flotante cuando se desea calcular la inversa de una matriz o cuando se resolverá una serie de sistemas de ecuaciones con una misma matriz de coeficientes. En la lectura, primeramente consideraremos la factorización LU sin intercambio basada en matrices elementales y que es conocida como de Doolittle y posteriormente veremos el algoritmo que da la factorización PA = LU. Con este metodo se factoriza la matriz como producto de una matriz triangular inferior y una triangular superior de esta forma:

$$\boxed{A = L \cdot U}$$

$$\begin{bmatrix} * & 0 & \cdots & 0 \\ * & * & \ddots & \vdots \\ * & * & * & 0 \\ * & * & * & * \end{bmatrix} \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ \vdots & \ddots & * & * \\ 0 & \cdots & 0 & * \end{bmatrix}.$$

Luego la resolución del sistema de ecuaciones Ax = b se encuentra de esta forma:

$$A \cdot x = b$$

$$L \cdot \underbrace{U \cdot x}_{y} = b \Rightarrow \begin{cases} L \cdot y = b \\ U \cdot x = y \end{cases}$$

Dado que las matrices L y U son triangulares, la resolución de los sistemas

- $L \cdot y = b$
- $U \cdot x = y$

tiene costo $O(n^2)$. De esta forma, si me cambian el vector b me ahorro de tener que volver a pagar $O(n^3)$ para encontrar una solución.

1.5. Sherman Morrison

Teniendo un sistema $Ax = b$, la formula Sherman Morrison nos permite encontrar $A'x = b$, sin tener que resolver de nuevo todo el sistema de ecuaciones, esto nos sirve fuertemente si ya tenemos la factorización LU de A, y solo sirve si A' se puede expresar como una modificación de A de esta manera: $A' = A + uv^t$ Entonces la inversa es:

$$(A')^{-1} = (A^{-1} + uv^t)^{-1} = A^{-1} - \frac{A^{-1}uv^tA^{-1}}{1 + v^tA^{-1}u}$$

Nosotros buscamos $A'x = b \Rightarrow x = (A')^{-1}b$, por la formula:

$$x = A^{-1}b - \frac{A^{-1}uv^tA^{-1}b}{1 + v^tA^{-1}u}$$

Entonces si tomamos $y = A^{-1}b$ y $z = A^{-1}$ como tenemos la LU de A, podemos resolver y e z en $O(n^2)$, luego reemplazando:

$$x = y - (\frac{v^ty}{1 + v^tz})z$$

2. Desarrollo

El problema presentado era el siguiente: uno tiene un parabrisas de vidrio rectangular al cual le están lanzando sanguijuelas robóticas con sopapas que aplican cierta temperatura al vidrio, que esta siendo refrigerado desde los bordes del mismo. Uno tiene que tener en cuenta que el parabrisas solo resiste hasta los 235^oC (medidos en el centro del parabrisas, que desde ahora llamaremos ”Punto Critico”), si esta medida se supera el vidrio estalla. El problema es entonces, sabiendo en que posicion del parabrisas estan pegadas estas sanguijuelas, calcular la temperatura de todo el parabrisas, en especial la del punto critico, y, al poder solo remover una sanguijuela, remover la que deje el parabrisas mas fresco en el punto critico. Para esto necesitamos cargar ciertos valores a la computadora, y como no hay memoria infinita, uno tiene que discretizar el parabrisas, es decir, elegir ciertos puntos metodologicamente para contemplar, en este caso estos puntos se seleccionaran cada una distancia h en el parabrisas, llamaremos a esta constante ”granularidad”. Luego de las sanguijuelas se tomara la posicion con respecto al limite inferior izquierdo del parabrisas, el radio de su sopapa y la temperatura que aplican. Para visualizarlo mejor se puede ver la siguiente figura:

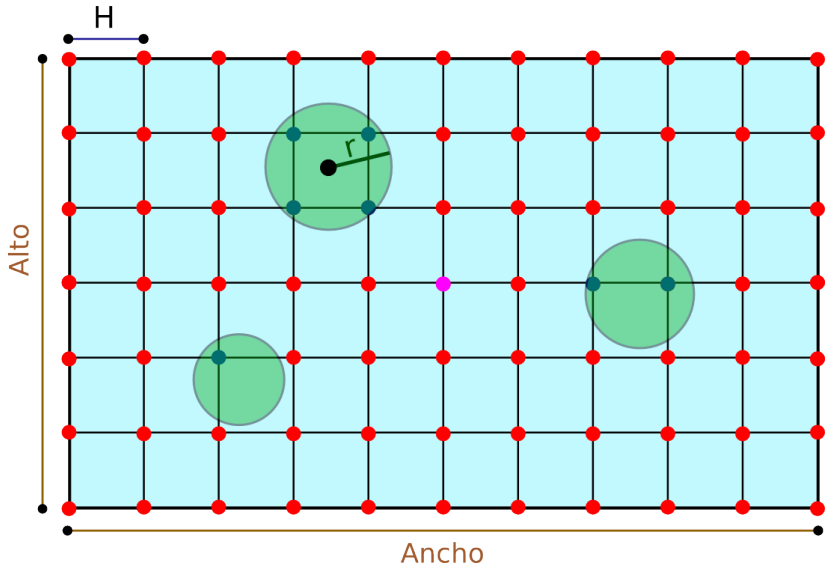


Figura 1: Discretización del parabrisas, y su información

Los puntos rojos son los puntos muestreados y procesados, los círculos verdes son las sanguijuelas (r es el radio) y H es la granularidad. Los puntos azules son los tocados por las sanguijuelas. El punto magenta es el punto critico.

Básicamente uno se encuentra en ese contexto y lo que tiene que hacer es calcular la temperatura del punto critico, que esta dada por la ecuación de calor de esta manera:

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

Que, al aproximar por diferencias finitas da:

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

Esto quiere decir que cada punto es el promedio de los que están alrededor (abajo, arriba, derecha e izquierda). Entonces, para calcular la temperatura del punto critico, necesitamos resolver la temperatura de todos los puntos, ya que todos los puntos dependen de los de alrededor, excepto los puntos con temperatura fija: los del borde con -100^oC y los tocados por sanguijuelas con la temperatura de su correspondiente sanguijuela.

Teniendo la ecuación anterior - que nos dice cual es la temperatura para un punto - uno tiene una ecuación por punto que resolver, estas formaran parte de un sistema de ecuaciones lineales de la forma

$Ax = b$ (con x incógnita). Por ejemplo, para un parabrisas de 4 x 3, con h = 1 como se ve en la siguiente figura:

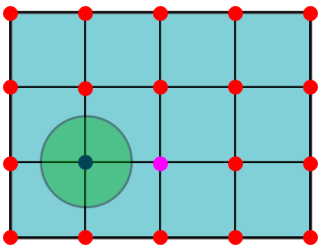


Figura 2: Ejemplo Parabrisas 4 x 3, h = 1

Para este contexto, el sistema seria el siguiente:

$$A = \begin{bmatrix}
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 & 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 & 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 & 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 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 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 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & -1 & \frac{1}{4} & 0 & 0 & 0 & \frac{1}{4} & 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 & 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 & 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
\end{bmatrix} \quad b = \begin{bmatrix}
-100 \\ -100 \\ -100 \\ -100 \\ -100 \\ -100 \\ 500 \\ 0 \\ 0 \\ -100 \\ -100 \\ 0 \\ 0 \\ 0 \\ -100 \\ -100 \\ -100 \\ -100 \\ -100 \\ -100
\end{bmatrix}$$

Como se puede ver, los puntos con valores fijos tienen en b su temperatura - ya sea -100^oC por ser borde o la temperatura de la sanguijuela, en esta caso 500^oC - y, como la ecuación para este punto sera solo si mismo, en A, esa fila, tendrá solo un 1 en su misma posición. Y los puntos a los que se les va a aplicar la ecuación de calor tendrán $\frac{1}{4}$ de cada punto que van a promediar y -1 de si mismo, ya que esa suma dará 0 -que es lo que esta en 'b'- cuando el valor de si mismo sea igual al promedio de sus vecinos.

$$t_{ij} = \frac{t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}}{4} \to 0 = \frac{t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}}{4} - t_{ij} \tag{3}$$

Observación: Algo muy importante que se puede ver a simple vista, es que A va a ser siempre cuadrada para cualquier ancho o alto de parabrisas del que se genero, y ademas A es una matriz banda (o multidiagonal), por lo tanto va a tener todos ceros fuera de las diagonales con información, entonces, pensando alguna estructura especial, uno puede almacenar la matriz ahorrando mucho espacio. Ademas, como se ve, los elementos de la diagonal son todos no nulos, por lo tanto se van a poder utilizar los métodos de resolución sin la necesidad se pivoteo

Ya teniendo el sistema, uno puede continuar de varias formas para resolverlo:

- Utilizar eliminación gaussiana para despejar todas las variables en función de una, y luego utilizar backward substitution para ir remplazando las variables y obtener el resultado.

- Utilizar factorización LU para descomponer a A en una matriz Lower y otra Upper, y luego con forward substitution para resolver $Ly = b$ y por ultimo backward substitution para resolver $Ux = y$ y obtener así el resultado.

Como el objetivo de esto es saber que sanguijuela remover para dejar lo mas fresco posible el punto critico, lo que hay que hacer es, generar A y b sin cada sanguijuela por separado, resolver el sistema, y quedarnos con la que resulte con menos temperatura en el punto critico. Si utilizara eliminación gaussiana para resolver cada sistema, tendría que reprocesar todo el sistema de nuevo para cada modificación a A o b, ya que, en este método, la triangulación de A va editando también el vector b, y los resultados son específicos para el sistema de ecuaciones que se utilizo.

En cambio, si utilizo la factorización LU, al ser la triangulación independiente del vector b, yo puedo utilizar la misma matriz L y U para resolver varios sistemas entre los que solo varía el vector b, esto lo podemos utilizar muy fuertemente si a pensamos un uso de la formula Sherman-Morrison, ya que, si la alteración de A se puede expresar como uv^t entonces podemos reutilizar L y U para resolver el sistema alterado!

En el problema planteado por el trabajo practico lo que se puede deducir acerca de esto, es que la única forma de que un cambio en A - al quitar una sanguijuela - se pueda expresar de la forma uv^t, es que solo se vea alterada una linea de A, eso puede suceder si la sanguijuela solo toca un punto, o que el .º exclusivo"de la sanguijuela quitada con el resto de las sanguijuelas solo toque un punto (en este caso se alteraría b en varios puntos, pero A solo en una fila). En cualquiera de los casos, el vector u tendría un 1 en la posición del punto alterado, y v^t seria de la forma:

$$v^t = \begin{bmatrix} \dots & 0 & \dots & \frac{1}{4} & \dots & 0 & \dots & \frac{1}{4} & -2 & \frac{1}{4} & \dots & 0 & \dots & \frac{1}{4} & \dots & 0 & \dots \end{bmatrix}$$

Este estaría generado especialmente dependiendo la fila que debe tocar y del parabrisas se esta procesando (esto ultimo altera el numero de diagonales que tiene la matriz banda). Para encontrarle la lógica, hay que pensar que uno quiere pasar de una ecuación que tenia una temperatura fija, osea con un 1 en el argumento de su propio punto, a una ecuación que cumple la ecuación de temperatura, el -2 es para pasar el 1 anterior a -1.

Luego aplicando Sherman-Morrison uno puede, simplemente utilizando L y U y haciendo las multiplicaciones vectoriales necesarias, resolver el sistema para un A y b alterados por la quita de una sanguijuela.

En este ejemplo, el resultado seria el siguiente (calculando solo las temperaturas, sin sacar ninguna sanguijuela):

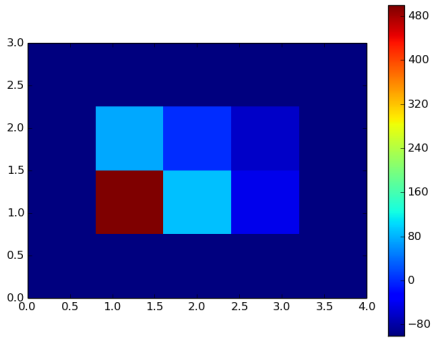


Figura 3: Resultado del ejemplo

2.1. Implementación

Al implementar, lo primero que se realizo fue la interfaz pedida por la cátedra, para la lectura de parámetros y datos, esto fue básico, se leyó todo a arrays para luego poder tenerlos listos para ser utilizados al generar el sistema de ecuaciones.

Para representar el sistema de ecuaciones, lo que se pensó fue realizar una estructura que se encargaría de almacenar todos los datos necesarios sobre el sistema y el contexto del problema. Para almacenar la

matriz A, aprovechando su característica de ser matriz banda, lo que se hizo fue almacenar solo las diagonales un array multidimensional de nxb, siendo n el n de la dimensión de A, y b el numero de diagonales o ancho de banda de la matriz. Luego, lo mejor de esta estructura, es que, desde el punto de vista externo, que la matriz sea banda es totalmente transparente, ya que el acceso a los datos se realiza a través de 'double get(int x, int y)' o 'void get(int x, int y, double d)', y esto se encargan de transformar la coordenada (x, y) a la posición correcta dentro del array multidimensional, esta función también va a reportar si se esta tratando de acceder a algún lugar que no se debería (fuera del ancho de banda), ya que esto no es necesario para los métodos que se van a utilizar (una salvaguarda mas que nada, para saber si algo no estaba andando bien durante el desarrollo).

El objetivo de esta estructura, con esas funciones, es poder utilizar el algoritmo clásico de eliminación gaussiana y factorización LU, sin tener que preocuparse por el almacenaje en banda de la matriz, y evitar así complicaciones innecesarias.

Continuando con la estructura de datos, tendrá espacios para almacenar la solución del sistema y el vector b.

Para almacenar los datos de las sanguijuelas de una forma eficiente, se van a guardar los datos de las mismas en tres lugares:

- std::vector<int>*sanguis : Un array de vectores, almacenara en el vector sanguis[i] los puntos que la sanguijuela i toca.
- double * tempSangui : Un array de temperaturas, almacenara en el valor tempSangui[i] la temperatura de la sanguijuela i.
- std::vector<int>*puntos : Un array de vectores, almacenara en el vector puntos[i] los indices de las sanguijuelas que tocan al punto i.

Esta organización de la información - que es redundante en el caso de 'puntos' y 'sanguis' - se aprovechara fuertemente al tener que modificar A al quitar una sanguijuela.

Lo primero que va a suceder luego de cargar los datos de entrada, es entrar a la función 'GenerarMatriz' que se encargara de generar el sistema de ecuaciones y completar la estructura de datos con los datos de entrada. Para llenar la información sobre las sanguijuelas se programo la función 'obtenerPuntosDeCalor' que se encarga de ver si una sanguijuela toca algún punto, para esto demarca un cuadrado con lado igual al doble del radio de la sanguijuela y con centro en el centro de la sanguijuela, luego, para cada punto de la discretización que se encuentre dentro de ese cuadrado va a usar Pitagoras para ver si este pertenece al circulo demarcado por la sanguijuela, ignorando los puntos que sean borde.

Por ultimo generara la matriz A y el vector b utilizando la información generada por 'obtenerPuntosDeCalor'.

Las únicas dificultades hasta este punto fue programar bien las funciones para leer la matriz y pensar las estructuras, esta ultima fue apareciendo a medida que se avanzo con el trabajo, ya que, por ejemplo, el método de almacenamiento de las sanguijuelas se genero por una necesidad que se dio posteriormente al hacer los algoritmos de eliminación de sanguijuelas

En este punto ya estaría almacenado todo lo necesario a procesar, y ya estaría el sistema de ecuaciones generado.

Luego se procederá dependiendo del método pasado por parámetro:

0. Se resuelve el sistema original utilizando el método de eliminación gaussiana junto con backward substitution.
1. Se resuelve el sistema original calculando la descomposición LU de A, luego se resuelve el sistema usando forward substitution y backward substitution.
2. Se decide que sanguijuela quitar para dejar el punto critico lo menos caliente posible, se utiliza el método básico de ir generando la matriz nuevamente para cada quita de sanguijuela, en mi caso decidí utilizar eliminación gaussiana para resolver los sistemas. Luego se devuelve el numero de sanguijuela a quitar y la solución del sistema sin esa sanguijuela.

3. También se decide que sanguijuela quitar para dejar el punto critico lo menos caliente posible, pero en este caso se usa el método de descomposición LU para que, si se dan las condiciones, se pueda utilizar la formula Sherman-Morrison para resolver los sistemas sin re-triangular el sistema.

2.1.1. Método 0

Para este caso se programó el algoritmo básico de eliminación gaussiana y backward substitution, ya que la matriz banda tiene la interfaz de una matriz común. Lo único que se tuvo en cuenta fue limitar los rangos de los ciclos for para no procesar los ceros que no van a cambiar, ya que seria un desperdicio de reloj del procesador. Se fijan limites condicionales para que solo se itere dentro de las bandas de la matriz.

Este algoritmo se probo con varias matrices a mano y comparando. Se presento cierta dificultad con el seteo de los limites, pero nada muy importante.

Funciones principales:

```
1 void eliminacionGauseana(Matriz* mat);
2 void backwardSubstitution(Matriz* mat);
```

2.1.2. Método 1

Aquí se programó de una manera muy similar el algoritmo para la descomposición LU, ya que estos son muy parecidos, lo único que cambia es que, en la descomposición LU nunca se toca el vector b, mientras que en eliminación gaussiana si, en cambio, en la factorización LU se va a calcular la matriz L a medida que se calcula la matriz U, U es la matriz que se calcula con el mismo algoritmo de la eliminación gaussiana.

Luego se programó el algoritmo de forward substitution para resolver y, que es igual al de backward substitution, pero itera de la forma opuesta. Por ultimo se va a utilizar el mismo algoritmo de backward substitution que eliminación gaussiana para resolver x y se imprime a archivo el resultado.

Funciones principales:

```
1 void descomposicionLU(Matriz * mat);
2 void fordwardSubstitution(Matriz * mat);
3 void backwardSubstitution(Matriz* mat);
```

2.1.3. Método 2

En este caso lo que se tuvo que diseñar fue el sistema de recalculado del sistema de ecuaciones para poder quitar una sanguijuela. En este momento es que se diseño la estructura para almacenar los datos de las sanguijuelas. Ya teniendo esa estructura, lo que se va a hacer es usar la función 'regenerarSinSangui' a la cual se le pasa el sistema y el indice de la sanguijuela a quitar.

Dicha función va utilizar el array de vectores 'sanguis' para ver que puntos esta sanguijuela toca, se va a pasar por estos puntos j y - utilizando el array de vectores 'puntos'- se va a ver si hay otras sanguijuelas que toquen ese punto, en caso positivo simplemente se va a cambiar el valor de $b[j]$ a la temperatura de la sanguijuela mas caliente que toca a ese punto (excluyendo a la sanguijuela quitada), caso contrario se va a colocar un 0 en $b[j]$ y se va a editar la fila $A[j]$ para colocar los 4 '$\frac{1}{4}$' en las posiciones correspondientes y el -1 en la columna j.

Luego de esto se resuelve el sistema y se guarda el resultado del punto critico. Luego se agrega nuevamente la sanguijuela quitada y se quita la siguiente, se resuelve el sistema, se compara con el resultado anterior, y así sucesivamente hasta que se pasa por todas las sanguijuelas, por ultimo se resuelve el sistema nuevamente para la sanguijuela que quedo como la mínima, y se imprime a archivo el sistema resuelto sin esa sanguijuela.

En este caso las dificultades se presentaron al querer regenerar la matriz, y se vio necesario generar las estructuras necesarias para lograrlo de una forma eficiente, y se logro un buen resultado.

Funciones principales:

```
1 int ultimaEsperanza(Matriz* mat, sanguijuela* sanguiList, int nsangui, double heightp,
      double widthp);
2 void regenerarSinSangui(Matriz * mat, int isangui);
3 void eliminacionGauseana(Matriz* mat);
4 void backwardSubstitution(Matriz* mat);
```

2.1.4. Método 3

Aquí vamos a utilizar la factorización LU para resolver el sistema de ecuaciones original que luego sera reutilizado.

Este ultimo método va a nuevamente iterar por las sanguijuelas, pero en este caso va utilizar una sub-función de 'regenerarSinSangui' llamada 'borrarSangui' que es la que se encarga de ver que puntos se van a liberar, osea que puntos pasan de tener una temperatura fija a utilizar la ecuación de calor, junto con esto va a editar a b correspondientemente, por ultimo va a devolver un set con solo los puntos que fueron liberados.

Luego de llamar a esta función, si la cantidad de puntos en el set que se devuelve es $<= 1$ - esto significa que se va a editar a lo sumo una fila de A - entonces se procede a generar el set u y v correspondiente y se va utilizar un algoritmo que aplica la formula de Sherman-Morrison para obtener el resultado utilizando el L y U previamente calculado.

En el caso que mas de una fila de A se vea afectada por la quita de la sanguijuela, el indice de esta sanguijuela se va a guardar en un vector para, luego de procesar todos los casos que se puedan con Sherman-Morrison, procesar de la misma forma que en el método 2 las sanguijuelas que no permiten cumplir las condiciones para aplicar Sherman-Morrison.

Luego de comparar todos los resultados, me quedo con el que deja el mínimo valor en el punto critico y se imprime esa solución.

Funciones principales:

```
1 int superLastHope(Matriz* mat, sanguijuela* sanguiList, int nsangui, double heightp, double
      widthp);
2 std::set<int> borrarSangui(Matriz * mat, int isangui);
3 void descomposicionLU(Matriz * mat);
4 void fordwardSubstitution(Matriz * mat);
5 void backwardSubstitution(Matriz* mat);
6 void sherman_morrison(Matriz * mat, double* u, double* v);
```

2.2. Experimentación

Para realizar las experimentaciones modifique el tp para que devuelva, por consola: el tiempo de ejecución, la temperatura del punto critico y la sanguijuela quitada (-1 si no aplica al método pedido). También se modifico para que, si se le manda un 4 parámetro, este lo tome como un override de la granularidad, ignorando la granularidad que indica el archivo '.in'. Luego elaboré un script, 'getdb.py' en python que va a correr el tp para los test que elija y va a guardar todos los datos de cada corrida en un csv para procesar luego.

El script va a correr los test para todos los h posibles para los test utilizados, y para cada granularidad va a correr los cuatro métodos. El script va a correr cada experimento varias veces y va el tiempo de ejecución que va a guardar para ese experimento sera la Media truncada en un 25 % para tener valores mas reales (la cantidad de veces que corre cada experimento esta seteada por la variable 'PRECISION').

El script es bastante sencillo, todo el contexto de prueba se encuentra en la carpeta 'src/Testing/' ahí se encuentra el script y la carpeta 'outputdata' donde estarán los 'testx.in' a probar y se guardara el csv resultado como 'resuts.csv'; luego, en la carpeta 'OUT-testx' se guardaran los outputs del tp con el nombre siguiendo la siguiente regla: 'granularidad_metodo.out' (en granularidad, el punto decimal se remplaza por un guion '-').

En mis experimentaciones decidí experimentar con los siguientes casos de test originales: test1.in, test2.in y test7.in, elegidos simplemente por gusto personal.

Tuve varias complicaciones durante la experimentación:

- Al principio no me había dado cuenta que no podía usar cualquier granularidad, y había empezado a hacer los experimentos empezando con h = 10 y en cada iteración restaba 0.1 hasta llegar a 0.3, esto luego lo cambie, calculando previamente los valores posibles, que se encuentran en el script.
- Me di cuenta, luego de dejar muchas horas corriendo los experimentos, que estaba había calculado mal el indice del punto critico en el vector solucion, por lo tanto los métodos de quitar las sanguijuelas no estaban funcionando como deberían, no comparaban el punto que deberían comparar, lo corregí y reinicie los experimentos.
- Se me acabo el espacio de mi carpeta de milagro.dc.uba.ar porque todavía tenia los test.csv y train.csv del TP2! No se había guardado el archivo de resultado, pero logre obtenerlo guardando el historial del tmux, ya que el script va mostrando por pantalla los resultados a medida que van pasando las experimentaciones.
- Me di cuenta, luego de pasar todas las experimentaciones correctamente, que no se me había ocurrido compilar con -O2, podría haber obtenido los resultados mucho mas rápido probablemente.

Las hipotesis planteadas en mi mente fueron:

- Resolver el sistema utilizando eliminación gaussiana tarda menos que la descomposición LU, ya que el ultimo tiene que hacer forward substitution. EQUIVOCADO: tarda mas, no pude deducir porque realmente.
- El método 3 tarda menos que el método 2 ya que, al programarlo, se ve que la cantidad de cálculos que se hacen son muchísimos menos si se cumplen las condiciones, si no se cumplen para ninguna sanguijuela entonces sera igual, ya que va a hacer lo mismo que el método 2. Por lo tanto el tiempo de ejecución del método 3 sera $<=$ al tiempo de ejecución del método 2. ACERTADO
- El método 3 va a ser menos eficiente a medida que baje la granularidad, ya que van a entrar mas puntos dentro de la superficie de una sanguijuela, entonces la probabilidad de que solo un punto se vea afectado por la eliminación de una sanguijuela va bajando. ACERTADO

3. Resultados

3.1. Experimento 1: Temperatura en función de la granularidad

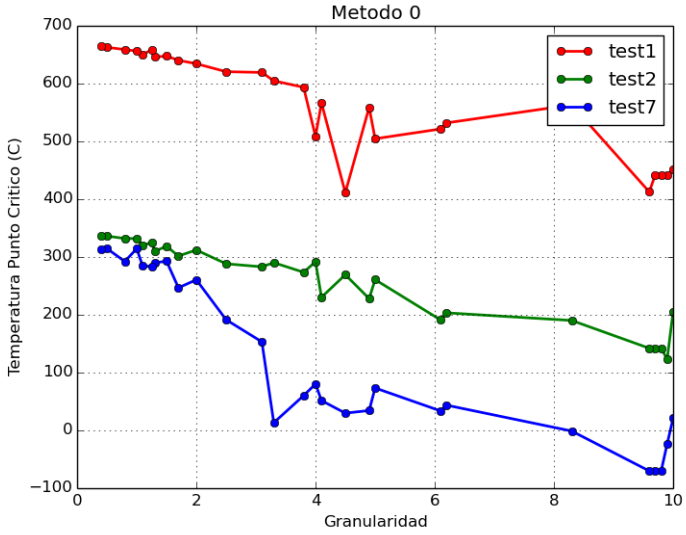
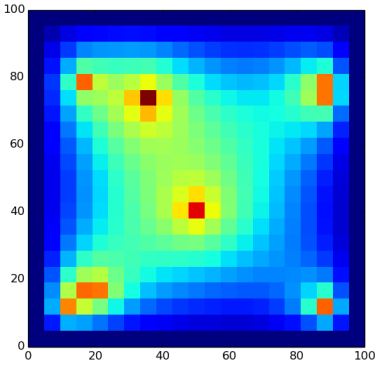
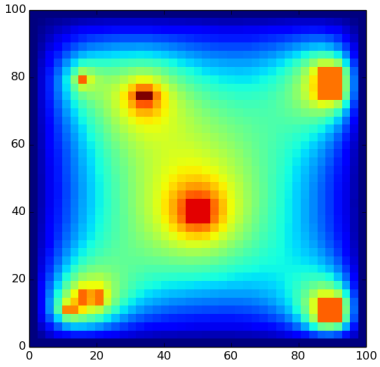


Figura 4: Resolución del Sistema de ecuaciones original

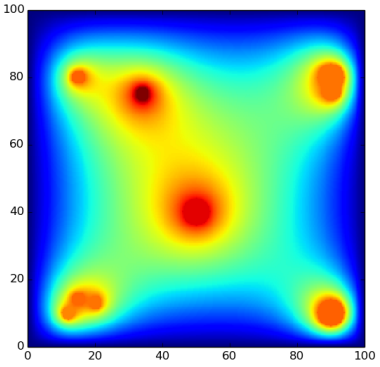
En esta experimentación se puede apreciar un muestreo que se va estabilizando a medida que decrece la granularidad, la causa de este fenómeno se podrá ver en las siguientes imágenes. Los picos que hay pueden suceder porque al cambiar la granularidad la grilla de puntos se va modificando también, tomando otros puntos de muestreo. Ahora vamos a ver, para cada test, como va cambiando la salida dependiendo de la granularidad:



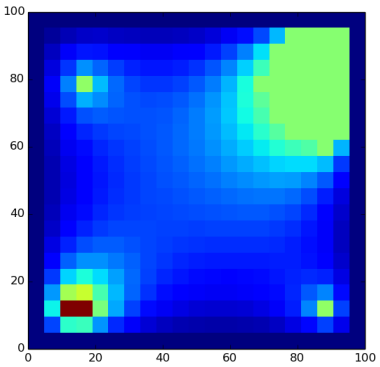
Test1 con granularidad 5



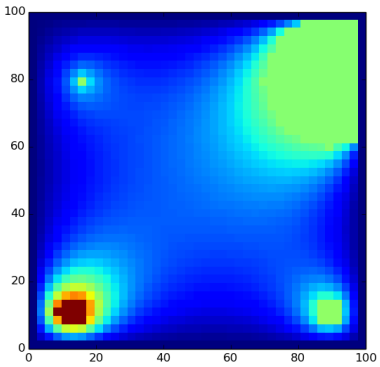
Test1 con granularidad 2.5



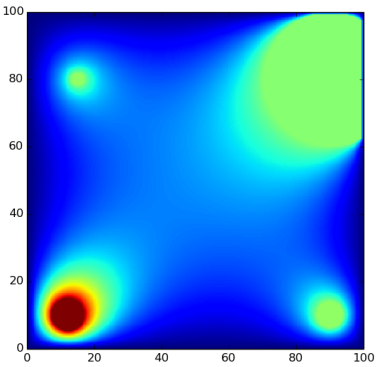
Test1 con granularidad 0.4



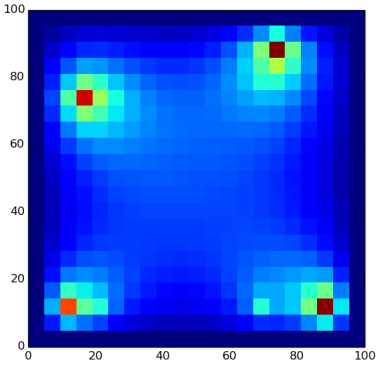
Test2 con granularidad 5



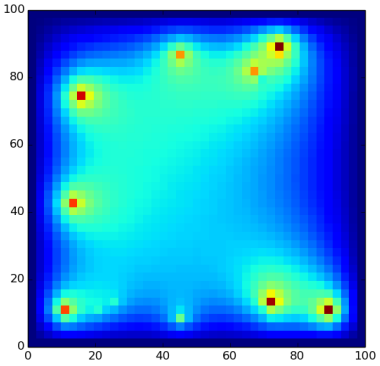
Test2 con granularidad 2.5



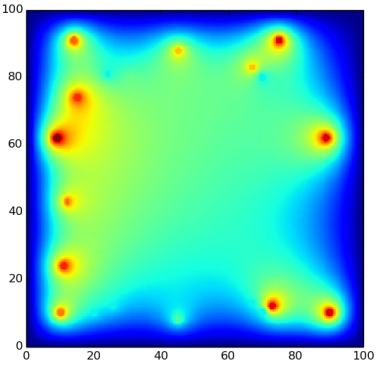
Test2 con granularidad 0.4



Test7 con granularidad 5



Test7 con granularidad 2.5



Test7 con granularidad 0.4

Aquí se ve muy claramente como se propaga menos el calor a medida que la granularidad es mayor, ya que al acercarse los puntos fríos a los calientes, por la ecuación de calor, al estar solo calculando un simple promedio, extremos de temperatura se ven muy afectados; y ademas, cuando ciertas sanguijuelas son muy pequeñas, estas se pierden entre los segmentos tan grandes de granularidad, como se ve en el Test7.

3.2. Experimento 2: Tiempo de computo en función de la granularidad

Ahora vamos a analizar como impacta la disminución de la granularidad en el tiempo de computo, esto tiene que mostrar un incremento del orden exponencial, ya que al disminuir por la mitad la granularidad, uno va a incrementar 4 veces la cantidad de puntos discretizados, esto generara una matriz A 16 veces mas grande que la anterior. Por supuesto, al ser una matriz banda el impacto es menor, pero es muy grande igual. Ahora veremos que dicen los datos:

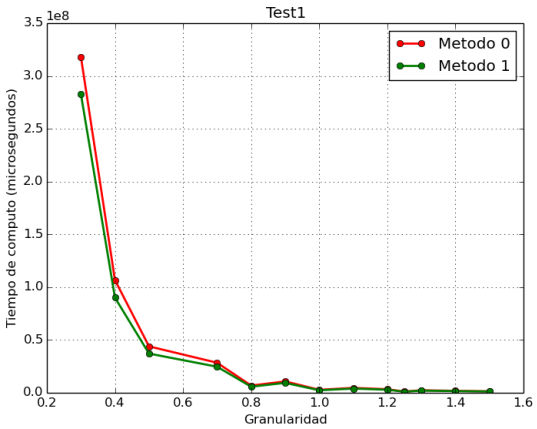


Figura 14: Resultados Test1

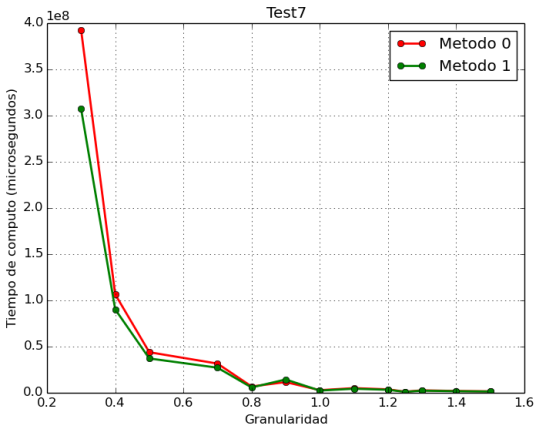


Figura 15: Resultados Test7

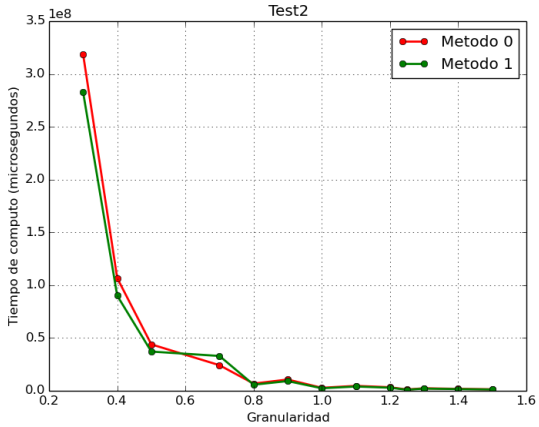


Figura 16: Resultados Test2

Es sorprenderte ver como realmente la diferencia entre Eliminación Gaussiana y Factorización LU no es tan remarcable, ya mientras veía los datos que salían de los experimentos parecía que si iba a haber una diferencia notable, pero estaba equivocado, la única diferencia de performance debería estar dada por el uso de dos algoritmos de substitution departe de la Factorización LU, que por los datos recolectados resulta ser un caso contra-intuitivo. Al sacar conclusiones, la única diferencia que podría llegar a notarse es el ciclo for que se realiza al comienzo del algoritmo de Eliminación Gaussiana, que copia los valores de b a x. Acerca de las ventajas, como ya vimos antes, LU tiene una ventaja muy grande a la hora de tener que resolver el mismo sistema de ecuaciones pero para otro b distinto, es ese caso te permite resolverlo simplemente corriendo el fordward substitution y luego el backward substitution, bajando la complejidad significativamente, de $O(n^3)$ a $O(n^2)$.

3.3. Experimento 3: Eliminación de Sanguijuelas

Por ultimo vamos a analizar la diferencia en performance al querer decidir que sanguijuela quitar para dejar el punto critico lo mas frió posible, para esto tenemos dos métodos, el primero en el cual se genera un sistema de ecuaciones con cada sanguijuela quitada por separado, y se resuelven utilizando el método de eliminación gaussiana; el segundo es descomponer la matriz A. del sistema original, como el producto de una matriz L y una U, y luego hacer lo posible para expresar los diferentes sistemas de ecuaciones sin las sanguijuelas como una modificación de una fila del sistema original, para poder utilizar la formula Sherman-Morrison y resolver el nuevo sistema en $O(n^2)$.

La intuición me decía que el incremento de performace del segundo método se iba a ver en las discretizaciones con granularidad mediana a alta, ya que, al ir decrementando la granularidad, las sopapas de las sanguijuelas iban a abarcar mas puntos, impidiendo poder expresar la modificación de la matriz A como la variación de una de sus filas nada mas. Veamos los resultados:

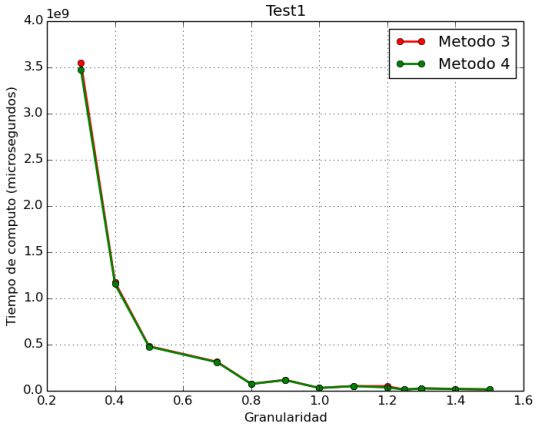


Figura 17: Resultados Test1

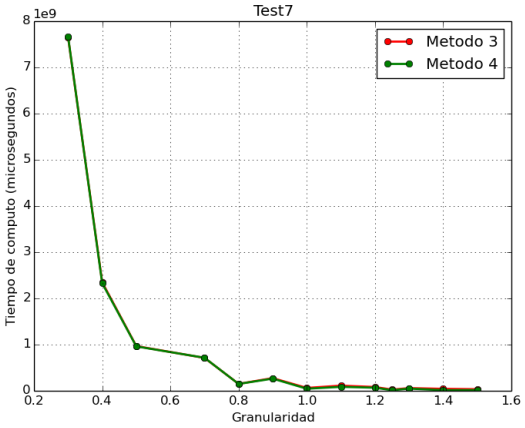


Figura 18: Resultados Test7

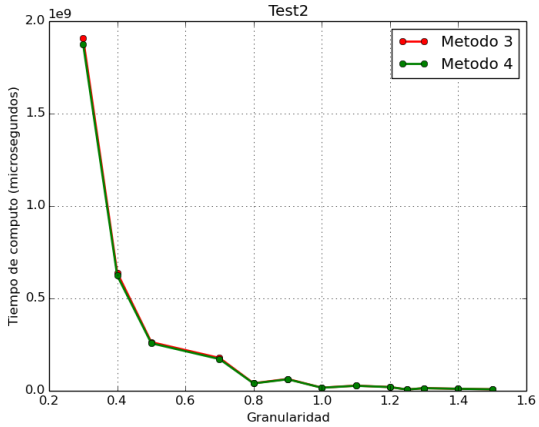


Figura 19: Resultados Test2

Nuevamente un resultado sorpresivo para mí, ya que esperaba una diferencia notable para las granularidades mayores, pero no, la diferencia es imperceptible, aun tomando mas foco en el sector de granularidades mayores:

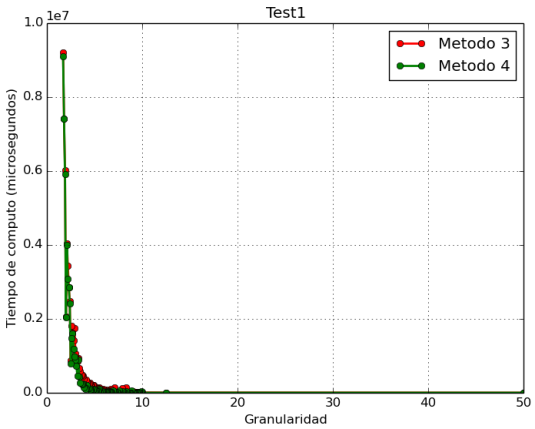


Figura 20: Resultados Test1

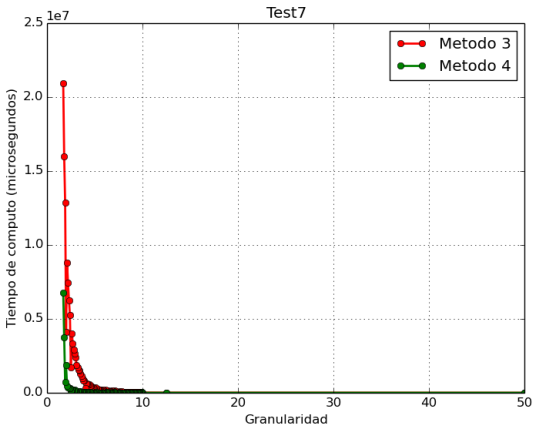


Figura 21: Resultados Test7

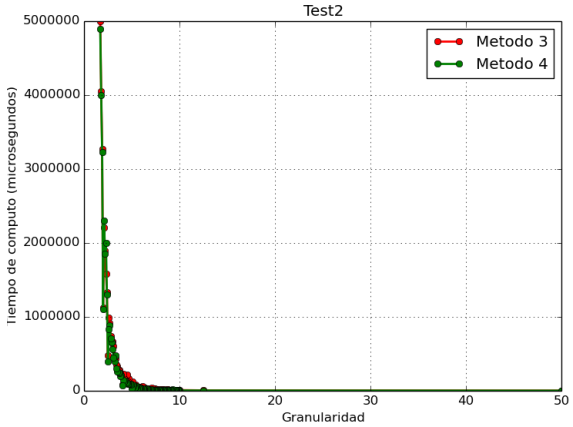


Figura 22: Resultados Test2

Se ve una diferencia ahora, pero esta tampoco es tan importante como esperaba, esto se da porque se tiene que dar un caso muy especial para poder expresar la modificación de la matriz A como la modificación de una sola de sus filas. Como en el caso de test7, que la superficie de las sopapas de las sanguijuelas son muy pequeñas, y por lo tanto no llegan a afectar a tantos puntos discretizados del sistema. Hay que tener en cuenta que ya, cambiando la ecuación de mas de un punto no se puede utilizar la formula de Sherman-Morrison, al menos no con la implementación que pensé yo

Ademas justamente lo importante de lograr mas performance es lograrlo con los casos que mas se tardan en resolver, ya a partir de la granularidad $>$a 2, el tiempo de resolución del sistema es tan pequeño que no se ve una diferencia notable.

Lo mejor que se podría hacer es pensar otra forma de poder aprovechar la formula Sherman-Morrison.

4. Conclusiones

5. Apendice: A

Laboratorio de Métodos Numéricos - Primer Cuatrimestre 2015
Trabajo Práctico Número 1: *"No creo que a él le gustará eso"*

Introducción

El afamado Capitán Guybrush Threepwood se encuentra nuevamente en problemas. El parabrisas de su nave El Pepino Marino está siendo atacado simultáneamente por varios dispositivos hostiles vulgarmente conocidos como *sanguijuelas mutantes*. Estos artefactos se adhieren a los parabrisas de las naves y llevan a cabo su ataque aplicando altas temperaturas sobre la superficie, con el objetivo de debilitar la resistencia del mismo y permitir así un ataque más mortífero. Cada sanguijuela consta de una *sopapa de ataque* circular, que se adhiere al parabrisas y aplica una temperatura constante sobre todos los puntos del parabrisas en contacto con la sopapa.

Para contrarrestar estas acciones hostiles, el Capitán Guybrush Threepwood cuenta con el sistema de refrigeración de la nave, que puede aplicar una temperatura constante de -100^{o}C a los bordes del parabrisas. El manual del usuario de la nave dice que si el punto central del parabrisas alcanza una temperatura de 235^{o}C, el parabrisas no resiste la temperatura y se destruye. Llamamos a este punto el *punto crítico* del parabrisas.

En caso de que el sistema de refrigeración no sea suficiente para salvar el punto crítico, nuestro Capitán Guybrush Threepwood tiene todavía una posibilidad adicional: puede destruir alguna de las sanguijuelas. Es importante destacar que solo puede destruir una de ellas, ya que la eliminación de muchas sanguijuelas consumiría energía vital para la finalización de la misión, y llevaría únicamente al fracaso de la misma. La situación es desesperante, y nuestro héroe debe tomar una rápida determinación: debe decidir, cuando sea posible, qué sanguijuela eliminar de modo tal que el parabrisas resista hasta alcanzar la base más cercana.

El modelo

Suponemos que el parabrisas es una placa rectangular de a metros de ancho y b metros de altura. Llamemos $T(x, y)$ a la temperatura en el punto dado por las coordenadas (x, y). En el estado estacionario, esta temperatura satisface la ecuación del calor:

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

La temperatura constante en los bordes queda definida por la siguiente ecuación:

$$T(x, y) = -100^o\text{C} \qquad \text{si } x = 0, a \text{ ó } y = 0, b. \tag{2}$$

De forma análoga es posible fijar la temperatura en aquellos puntos cubiertos por una sanguijuela, considerando T_s a la temperatura ejercida por las mismas.

El problema en derivadas parciales dado por la primera ecuación con las condiciones de contorno presentadas recientemente, permite encontrar la función T de temperatura en el parabrisas, en función de los datos mencionados en esta sección.

Para estimar la temperatura computacionalmente, consideramos la siguiente discretización del parabrisas: sea $h \in \mathbb{R}$ la granularidad de la discretización, de forma tal que $a = m \times h$ y $b = n \times h$, con $n, m \in \mathbb{N}$, obteniendo así una grilla de $(n+1) \times (m+1)$ puntos.

Luego, para $i = 0, 1, \ldots, n$ y $j = 0, 1, \ldots, m$, llamemos $t_{ij} = T(x_j, y_i)$ al valor (desconocido) de la función T en el punto $(x_j, y_i) = (ih, jh)$, donde el punto $(0, 0)$ se corresponde con el extremo inferior izquierdo del parabrisas. La aproximación por *diferencias finitas* de la ecuación del calor afirma que:

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

Es decir, la temperatura de cada punto de la grilla debe ser igual al promedio de las temperaturas de sus puntos vecinos en la grilla. Adicionalmente, conocemos la temperatura en los bordes, y los datos del problema permiten conocer la temperatura en los puntos que están en contacto con las sanguijuelas.

Enunciado

Se debe implementar un programa en `C` o `C++` que tome como entrada los parámetros del problema (a, b, h, junto con las posiciones, radio y temperatura de las sanguijuelas) y calcule la temperatura en el parabrisas utilizando el modelo propuesto en la sección anterior.

Para resolver este problema, se deberá formular un sistema de ecuaciones lineales que permita calcular la temperatura en todos los puntos de la grilla que discretiza el parabrisas, e implementar el método de Eliminación Gaussiana (EG) para resolver este sistema particular. Dependiendo de la granularidad utilizada en la discretización, el sistema de ecuaciones resultante para este problema puede ser muy grande. Luego, es importante plantear el sistema de ecuaciones de forma tal que posea cierta estructura (i.e., una matriz banda), con el objetivo de explotar esta característica tanto desde la *complejidad espacial* como *temporal* del algoritmo. Además de la estructura banda, en determinados casos es posible utilizar la descomposición LU del sistema para acelerar el cómputo de evaluar qué sucedería si una sanguijuela fuese eliminada.

En función de la implementación, como mínimo se pide:

1. *Explotando la estructura:* Representar la matriz del sistema aprovechando la estructura banda de la misma, haciendo hincapié en la complejidad espacial. Realizar las modificaciones necesarias del algoritmo de EG clásico para que aproveche la estructura banda de la matriz, e implementarlo.
2. *Factorización LU:* Implementar un algoritmo que permita calcular la descomposición LU de la matriz, aprovechando la estructura banda de la misma. Además, se deben implementar los algoritmos de *forward/back substitution* para resolver el sistema utilizando la factorización LU. Discutir alternativas de implementación buscando minimizar el espacio utilizado.
3. *Última Esperanza:* Implementar un algoritmo para decidir si es posible salvar el parabrisas de la destrucción mediante la eliminación de *una* sanguijuela. En caso de ser posible, el algoritmo debe determinar la *mejor* sanguijuela a eliminar, (i.e., aquella que al ser removida genera la menor temperatura en el punto crítico).
4. *No hay tiempo que perder:* Existen casos donde la eliminación de una sanguijuela modifica *levemente* el sistema, alterando solo una *fila* de este. En estos casos, es posible utilizar la factorización LU y lo que se conoce como fórmula de Sherman–Morrison [1]:

$$(A + uv^t)^{-1} = A^{-1} - \frac{A^{-1}uv^tA^{-1}}{1 + v^tA^{-1}u}. \quad (4)$$

para acelerar el cómputo de evaluar la temperatura de eliminar una sanguijuela. Identificar estos casos, analizar cómo se modifica el sistema, e implementar una versión del algoritmo propuesto en 3 que explote esta propiedad cuando sea posible.

En función de la experimentación, como mínimo debe realizarse lo siguiente:

- Considerar al menos tres instancias originales de prueba, generando discretizaciones variando la granularidad para cada una de ellas y comparando el valor de la temperatura en el punto crítico. Se sugiere presentar gráficos de temperatura para los mismos, ya sea utilizando las herramientas provistas por la cátedra o implementando sus propias herramientas de visualización.
- Analizar el tiempo de cómputo requerido en función de la granularidad de la discretización, buscando un compromiso entre la calidad de la solución obtenida y el tiempo de cómputo requerido. Comparar los resultados obtenidos para alguna de las variantes propuestas en 1 y 2, y analizar ventajas y desventajas de ambos esquemas. ¿Cómo impacta en los resultados la elección del esquema 1 y 2? ¿Por qué?
- Estudiar el comportamiento del método propuesto para la estimación de la temperatura en el punto crítico y para la eliminación de sanguijuelas, comparando las variantes propuestas en 3 y 4.

Finalmente, se deberá presentar un informe que incluya una descripción detallada de los desarrollos teóricos propuestos, métodos implementados y las decisiones tomadas, incluyendo las estructuras utilizadas para representar la matriz banda y los experimentos realizados, junto con el correspondiente análisis y siguiendo las pautas definidas en el archivo `pautas.pdf`.

Opcional:

- Al aplicar Eliminación Gaussiana sobre el sistema asociado a este problema en particular, ¿Es necesario pivotear? ¿Por qué?
- ¿Cómo modificaría el algoritmo propuesto en 3 para aprovechar la fórmula de Sherman–Morrison [1] para cualquier tipo de sanguijuela?

Programa y formato de archivos

El programa debe tomar tres parámetros (y en ese orden): el archivo de entrada, el archivo de salida y el método a ejecutar, (*0:* Eliminación Gaussiana banda, *1:* Factorización LU explotando la propiedad banda y utilizando forward/back substitution, *2:* Algoritmo de eliminación de sanguijuela simple, *3:* Algoritmo de eliminación de sanguijuela mejorado, usando la fórmula de Sherman–Morrison cuando sea posible).

El archivo de entrada contiene los datos del problema (tamaño del parabrisas, ubicación, radio y temperatura de las sanguijuelas) desde un archivo de texto con el siguiente formato:

```
(a)  (b)  (h)  (k)
(x1) (y1) (r1) (t1)
(x2) (y2) (r2) (t2)
```

```
...
(xk) (yk) (rk) (tk)
```

En esta descripción, $a \in \mathbb{R}_+$ y $b \in \mathbb{R}_+$ representan el ancho y largo en metros del parabrisas, respectivamente. De acuerdo con la descripción de la discretización del parabrisas, h es la longitud de cada intervalo de discretización, obteniendo como resultado una grilla de $n+1 \in \mathbb{N}$ filas y $m+1 \in \mathbb{N}$ columnas. Además, $k \in \mathbb{N}$ es la cantidad de sanguijuelas. Finalmente, para $i = 1, \ldots, k$, el par (x_i, y_i) representa la ubicación de la i-ésima sanguijuela en el parabrisas, suponiendo que el punto $(0, 0)$ corresponde al extremo inferior izquierdo del mismo, mientras que $r_i \in \mathbb{R}_+$ representa el radio de la sopapa de ataque de la sanguijuela (en metros), y $t_i \in \mathbb{R}$ es la temperatura de dicha sopapa.

El archivo de salida contendrá los valores de la temperatura en cada punto de la discretización utilizando la información original del problema (es decir, antes de aplicar el método de remoción de sanguijuela), y será utilizado para realizar un testeo parcial de correctitud de la implementación. El formato del archivo de salida contendrá, una por línea, el indicador de cada posición de la grilla i, j junto con el correspondiente valor de temperatura. A modo de ejemplo, a continuación se muestran cómo se reportan los valores de temperatura para las posiciones $(3, 19)$, $(3, 20)$, $(4, 0)$ y $(4, 1)$.

```
...
3   19  -92.90878
3   20  -100.00000
4   0   -100.00000
4   1   60.03427
...
```

El programa debe ser compilado, ejecutado y testeado utilizando los *scripts* de *python* que acompañan este informe. Estos permiten ejecutar los tests provistos por la cátedra, incluyendo la evaluación de los resultados obtenidos e informando si los mismos son correctos o no. Es requisito que el código entregado pase satisfactoriamente los casos de tests provistos para su posterior corrección. Junto con los archivos podrán encontrar un archivo `README` que explica la utilización de los mismos.

Sobre la entrega

- FORMATO ELECTRÓNICO: Jueves 09 de Abril de 2015, hasta las 23:59 hs., enviando el trabajo (`informe` + `código`) a `metnum.lab@gmail.com`. El `asunto` del email debe comenzar con el texto `[TP1]` seguido de la lista de apellidos de los integrantes del grupo. Ejemplo: `[TP1] Acevedo, Miranda, Montero`
- FORMATO FÍSICO: Viernes 05 de Abril de 2015, de 17:30 a 18:00 hs.

Referencias

[1] Golub, G. H. and Van Loan, C. F. Matrix Computations, 3rd ed. Baltimore, MD: Johns Hopkins, p. 51, 1996.

6. Apendice: B

Estructura principal del sistema:

```
struct Matriz
{
  double** mat; //Matriz entera, o Upper en LU
  double** L;   //Lower en LU
  int n;       //Cantidad de coeficientes. La matriz seria de nxn
  int ancho;    //Cantidad de puntos de ancho en la discretizacion del parabrisas. Alto = n/ancho
  double * valores; //b en Ax = b
  double * solution; //x en Ax = b
  bool solved;  //Si esta resuelta
  double h;    //precision de discretizacion del parabrisas
  int pto;     //Indice de la temperatura en el Punto Critico. El x en valores[x]
  int sanguipre;  //Sanguijuela borrada para restaurarla. Si nunca se borro es -1
  double * minSolution; //Solucion minima al sacar sanguis
  //Datos de sanguijuelas: sanguis[i] y tempSangui[i] corresponden a la sanguijuela i
  std::vector<int> *sanguis; //Guarda para cada sanguijuela que puntos toca
  double * tempSangui;  //Guarda la temperatura de cada sanguijuela
  std::vector<int> *puntos; //Para cada punto del parabrisas guarda que sanguijuelas (indice) lo tocan

public:
  ~Matriz();
  double get(int y, int x){
    int xf = ancho - y + x;
    return (xf < 0 || xf > ancho * 2) ? 0 : this->mat[y][xf];
  }

  void set(int y, int x, double d){
    int xf = ancho - y + x;
    if (xf < 0 || xf > ancho * 2){
      std::cout << "Set choto y= " << y << " x= " << x << std::endl;
    }
    else{
      this->mat[y][xf] = d;
    }
  }
  double getL(int y, int x){
    int xf = ancho - y + x;
    return (xf < 0 || xf > ancho * 2) ? 0 : this->L[y][xf];
  }

  void setL(int y, int x, double d){
    int xf = ancho - y + x;
    if (xf < 0 || xf > ancho * 2){
      std::cout << "Set choto L y= " << y << " x= " << x << std::endl;
    }
    else {
      this->L[y][xf] = d;

    }
  }
};
```

Eliminación gaussiana:

```
void eliminacionGauseana(Matriz* mat){
  int i, j, pivote = 0;
  int filas = mat->n;
  int columnas = mat->n;
  int ancho = mat->ancho;
  long double multiplicador;
  for (int x = 0; x < columnas; x++){ mat->solution[x] = mat->valores[x]; }
  for (pivote = 0; pivote < filas - 1; ++pivote){
    int endi;
    if (pivote + ancho + 1 <= mat->n) { endi = pivote + ancho + 1; }
    else { endi = mat->n; }
    for (i = pivote + 1; i < endi; i++){
      multiplicador = mat->get(i, pivote) / mat->get(pivote, pivote);
      int endj;
      if (i + ancho + 1 <= mat->n) { endj = i + ancho + 1; }
      else { endj = mat->n; }
      for (j = pivote; j < endj; j++){
        mat->set(i, j, mat->get(i, j) - (multiplicador * mat->get(pivote, j)));
      }
      mat->solution[i] = mat->solution[i] - multiplicador*mat->solution[pivote];
    }
  }
}
```

Descomposición LU:

```
void descomposicionLU(Matriz * mat){
    int i, j, pivote = 0;
  int n = mat->n;
  int ancho = mat->ancho;
  double multi;
  for (pivote = 0; pivote < n - 1; ++pivote){
    int endi;
    if (pivote + ancho + 1 <= mat->n) { endi = pivote + ancho + 1; }
    else { endi = mat->n; }
    for (i = pivote + 1; i < endi; i++){
      multi = mat->get(i,pivote) / mat->get(pivote,pivote);
      int endj;
      if (i + ancho + 1 <= mat->n) { endj = i + ancho + 1; }
      else { endj = mat->n; }
      for (j = pivote; j < endj; j++){
        mat->set(i,j,mat->get(i,j) - (multi * mat->get(pivote, j)));
      }
      mat->setL(i, pivote, multi);
    }
  }
  setDiagonalUnos(mat);
}
```

Forward y backward substitution:

```
void fordwardSubstitution(Matriz * mat){
  double y;
  int n = mat->n;
  int ji;
  for(int i = 0; i < n; i++){
      y = mat->valores[i];
    if (i - mat->ancho < 0) { ji = 0; }
```

```
    else { ji = i - mat->ancho; }   //Adaptacion a banda
      for (int j = ji; j < i ; j++){
          y = y - mat->getL(i, j)* mat->solution[j];
      }
      mat->solution[i] = y/mat->getL(i, i);
    }
}

void backwardSubstitution(Matriz* mat){
  double y;
  int je;
  for (int i = mat->n - 1; i >= 0; i--){
    y = mat->solution[i];
    if (i + mat->ancho + 1 > mat->n) { je = mat->n; }
    else { je = i + 1 + mat->ancho; } //Adaptacion a banda
    for (int j = i + 1; j < je; j++){
      y = y - mat->get(i, j)* mat->solution[j];
    }
    mat->solution[i] = y / mat->get(i, i);
  }
  mat->solved = true;
}
```

Sherman-Morrison:

```
void sherman_morrison(Matriz * mat, double* u, double* v) {
  int n = mat->n;
    //Sherman-Morrison A' = (A + uv')
  double * b = new double[mat->n];
  for (int i = 0; i < n; i++)
    b[i] = mat->valores[i];
    //From Sherman-Morrison
    // A^-1 b = y <=> Ay = b

  fordwardSubstitution(mat);
  backwardSubstitution(mat);
  double * y = new double[mat->n];
  for (int i = 0; i < n; i++)
    y[i] = mat->solution[i];

    // A^-1 u = z <=> Az = u

  for (int i = 0; i < n; i++)
    mat->valores[i] = u[i];

  fordwardSubstitution(mat);
  backwardSubstitution(mat);
  double * z = new double[mat->n];
  for (int i = 0; i < n; i++)
    z[i] = mat->solution[i];

    //Finally mat->solution = y - z * [(v' y)/(1 + v' z)]

  double vty = 0;
  for (int i = 0; i < n; i++)
    vty += v[i] * y[i];

  double vtz = 0;
  for (int i = 0; i < n; i++)
```

```
35      vty += v[i] * z[i];
36
37    double a = vty / (vtz + 1);
38    for (int i = 0; i < n; i++)
39      mat->solution[i] = y[i] - (z[i] * a);
40
41    for (int i = 0; i < n; i++)
42      mat->valores[i] = b[i];
43
44    delete[] y;
45    delete[] z;
46    delete[] b;
47  }
```

Scripts Experimentos:

```
1  import sys
2  import re
3  import os
4  from os.path import basename
5  from numpy import mean
6  import shutil
7  import commands
8  import subprocess
9
10 #Para correr en milagro y no perder los datos!
11 #outpdir = os.path.expanduser('~') + "/outputdata/"
12 outpdir = "./outputdata/"
13 bindir = "../"
14 inpdir = "./outputdata/"
15 PRECISION = 10
16 tests = ["test1", "test2", "test7"]
17 granuls = [50.0, 12.5, 10.0, 6.25, 5.0, 4.0, 2.5, 2.0, 1.25, 1.0, 0.8, 0.5, 0.4, 0.25]
18 #granuls = [float(x)/10 for x in range(100,20,-1)]
19 startp = 10
20
21 def main():
22   if not os.path.exists(outpdir):
23     os.makedirs(outpdir)
24   global startp
25   if len(sys.argv) == 2:
26     startp = float(sys.argv[1])
27   process()
28
29 def trimmean(arr, percent):
30   n = len(arr)
31   k = int(round(n*(float(percent)/100)/2))
32   return mean(arr[k+1:n-k])
33
34
35 def process():
36     done = 0
37     total = len(granuls) * 3 * 4
38     ou = open(outpdir + "results.csv", "w")
39     ou.write("testN, Method, Granularity, Time, tempPoint, RemovedSangui" +'\n')
40     #Para cada implementacion
41     for impl in tests:
42       boutdir = outpdir + "OUT-" + impl + "/"
43       if not os.path.exists(boutdir):
```

```
44          os.makedirs(boutdir)
45      for granul in granuls:
46        param4 = str(granul)
47        for test in tests:
48          boutdir = outpdir + "OUT-" + test + "/"
49          param1 = inpdir + test +".in"
50          for method in range(0,4):
51            param2 = boutdir + str(granul).replace('.','-') + "_" + str(method) + ".out"
52            param3 = str(method)
53
54            time = []
55            for x in range(0,PRECISION+1):
56              outp = subprocess.check_output([ bindir +"tp", param1, param2, param3,
                      param4]).split(',')
57              if (len(outp) < 3): continue
58              microseconds = int(outp[0])
59              if (microseconds != 0):
60                  time.append(microseconds)
61            done += 1
62            print("G="+ param4 + " Set=" + test + " Method=" + param3 + ' | ' + str(done) +
                  "of" + str(total) + " | Process " +
                  "{0:.2f}".format(float(done*100)/float(total)) + "%" + " | " + str(outp))
63            ou.write(test + ',' + str(method) + ',' + str(granul) + ',' +
                  str(int(trimmean(sorted(time), 0.25))) + ',' + outp[1] + ',' + outp[2] + "\n")
64
65      ou.close()
66
67  if __name__ == '__main__':
68    main()
```