DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Sanguijuelas

26 de noviembre de 2014

Métodos Numéricos
Trabajo Práctico Nro. 1

Integrante	LU	Correo electrónico
Martin Carreiro	45/10	martin301290@gmail.com
Kevin Kujawski	459/10	kevinkuja@gmail.com
Juan Manuel Ortíz de Zárate	403/10	jmanuoz@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. Resumen

Mediante el manejo de matrices se buscará modelar y resolver un problema de la realidad. Cómo la realidad está compuesta de infinitas variables dicha modelización implicará una inevitable discretización. Es decir trabajar con una cantidad acotada de variables del problema (sólo las relevantes). Si bien el problema en cuestión consiste en decidir cual/es de las sanguijuelas que están adheridas al parabrisas se debe eliminar, la modelización del mismo no es trivial. Es más, podría decirse que este proceso es mucho más complejo y costoso que la solución en sí. ¿Por qué? porque la creación de la matriz que represente al parabrisas y el cálculo de las temperaturas en cada punto, si no se usa buen método, podría llegar a demorar mucho tiempo.

Por esto es que a lo largo de este tp haremos mucho foco en como calcular las temperaturas, como optimizar el espacio ocupado por la matriz obtenida y como optimizar lo más posible todas las operaciones matriciales.

2. Introducción teórica

Dado que la modelización utlizada para este problema es un sistema de ecuaciones y que en base a algunos datos que nos dan (posición de las saniguijuelas, dimensiones y temperaturas de los bordes) tenemos que calcular los valores de muchas celdas, lo que hay que resolver es un sistema de ecuaciones. Lo primero que se nos viene a la mente cuando tenemos esto es el método de eliminación gaussiano. Este transforma la matriz de coeficientes en una matriz triangular superior y luego mediante back substitution se pueden obtener todos sus valores. Si bien este método no funciona siempre veremos que para nuestro caso puede otorgarnos soluciones aceptables.

Veamos ahora más en detalle como funciona cada uno de estos algoritmos.

2.1. Algoritmo de eliminación gaussiana

Este algoritmo consisnte en restar multiplos de filas entre sí, comenzando desde la primera, con el objetivo de lograr todos ceros por debajo de la diagonal y numeros disintos de cero en la diagonal, es decir una matriz triangular superior. En el caso que me quedase un 0 en la diagonal el algoritmo comprende la opción de pivoteo (mas adelante demostraremos que en realidad no necesitamos) que consiste en intercambiar filas entre sí para que esto no suceda.

Veamos ahora algún ejemplo concreto con la siguiente matriz de 3×3:

$$\begin{pmatrix} 4 & 8 & 9 \\ 1 & 5 & 2 \\ 2 & 3 & 1 \end{pmatrix}$$

Hago

F2 = F2 - 1/4 (F1) y F3 = F3 - 1/2 (F1)

$$\begin{pmatrix} 4 & 8 & 12 \\ 0 & 3 & -1 \\ 0 & -1 & -5 \end{pmatrix}$$

y ahora

F3 = F3 - -1/3 (F2)

$$\begin{pmatrix} 4 & 8 & 12 \\ 0 & 3 & -1 \\ 0 & 0 & -31/3 \end{pmatrix}$$

Si entendemos la matriz original como un sistema de ecuaciones con 3 variables igualadas a distintos resultados, como podemos ver a continuación:

$$\begin{pmatrix} 4x & 8y & 9z \\ 1x & 5y & 2z \\ 2x & 3y & 1z \end{pmatrix}$$

Con esta nueva matriz triangulada podríamos calcular facilmente el valor de cada variable ya que de la

última ecuación podemos despejar facilmente el valor de Z, reemplazarlo en las otras ecuaciones y así calcular las otras variables.

Por último cabe destacar que la complejidad computacional de esta trinagulación es de O(n^3). Lo cual puede llegar a un cálculo sumamente extenso si nuestro n es considerablemente importante. Por esto es que decidimos pensar un poco mas la situación y encontrarle la vuelta para resolver en un tiempo razonable.

2.2. Matriz banda

Matriz banda es un caso especial de las matrices que ocurre cuando estas concentran toda su información cerca de la diagonal, más especificamente, la matriz tiene en su mayoría elementos nulos y los que no son nulos se concentrar a izquierda y derecha de la diagonal de la matriz, siendo el máximo de estas distancias la que determinaran el ancho de la banda con que se denomine a la matriz y buscandose que las mismas sean relativamente pequeñas en comparación del ancho de la matriz. Por ejemplo, si a lo sumo tiene p elementos no nulos a la izquierda y q a la derecha, esta matriz tendrá una banda de $p + q + 1$. Hay casos particulares para denominar estas matrices, por ejemplo, cuando $p = q = 0$ esta se denomina diagonal, y en caso de $p = q = 1$ es una tridiagonal. La ventaja principal que proporcina este tipo de matriz es la optimización a la hora de su almacenamiento, ya que como la mayoría de los elementos son nulos y los no nulos tiene un ancho fijo, es posible guardarlos en una matriz mucho menor con solo el tamaño de la banda como ancho, ahorrando gran cantidad de memoria.

Aca podemos ver un ejemplo de una matriz banda:

Matriz banda tridiagonal

$$\begin{pmatrix} d & q & 0 & 0 & 0 & 0 \\ p & d & q & 0 & 0 & 0 \\ 0 & p & d & q & 0 & 0 \\ 0 & 0 & p & d & q & 0 \\ 0 & 0 & 0 & p & d & q \\ 0 & 0 & 0 & 0 & p & d \end{pmatrix}$$

En el ejemplo se puede ver como se forma la banda sobre la diagonal, con la mayoría de elementos nulos fuera de la misma. Por lo tanto para ahorrar espacio se puede optimizar su almacenamiento de la siguiente forma:

Almacenamiento sólo de la banda de la matriz original

$$\begin{pmatrix} 0 & d & q \\ p & d & q \\ p & d & q \\ p & d & q \\ p & d & q \\ p & d & 0 \end{pmatrix}$$

En la sección Desarrollo mostraremos como aprovecharemos las ventajas de este tipo de estructura para resolver el problema planteado de una forma más eficiente.

3. Desarrollo

3.1. Eliminación Gaussiana

A modo de comparación recreamos el clásico método de resolución de Eliminación Gaussiana que consta de la eliminación progresiva de variables en el sistema de ecuaciones, hasta tener sólo una ecuación con una incógnita. Una vez resuelta esta, se procede por sustitución regresiva hasta obtener los valores de todas las variables.

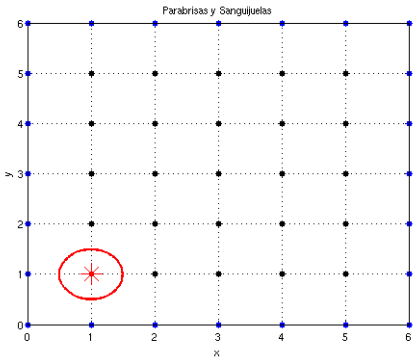
```
Class Windshield{
    resolveByGaussianElimination(){
         gaussianElimination();
         backSubstitution();
    }
    gaussianElimination(){
        for rows k already been reduced
            iMax = findPivot();
            Swap rows k and imax;
            Force 0's in column A[k+1..n-1][k];
    }
    backSubstitution(){
        for row from last to first:
            calculateFromNextOneExceptForLast()
    }
}
```

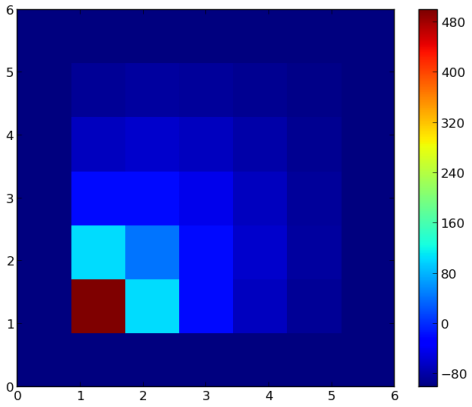
3.2. Matriz Banda

En la sección anterior presentamos una forma de resolver nuestro problema representando el parabrisas como un sistema de ecuaciones y aplicando Eliminación Gaussiana. En este sección propondremos una optimización a esto, pero para esto primero necesitamos ver con qué cosas estamos tratando y para eso veamos el siguiente ejemplo:

En este caso es un parabrisas de 6x6 granulado en 1 y una sanguijuela de radio 1 con temperatura de 500 en la primera posición.

El parabrisas se vería representado asi (hay que tener en cuenta que en realidad hay que espejarlo horizontalmente ya que para nosotros el eje de coordenadas se encuentra arriba a la izquierda):





Planteando el parabrisas como un sistema de ecuaciones sin tener en cuenta que es banda y por consiguiente sin realizar ninguna optimización al respecto, la matriz quedaría como el siguiente gráfico, donde en cada fila se representa todo el parabrisas como un vector $'aplanado'$ pero hace referencia a una en particular. Por lo tanto la diagonal de la matriz representa al coeficiente de esa posición en el parabrisas. En base a lo que explicamos anteriormente y como la temperatura de cada celda vacía es el promedio de las 4 contiguas, al plantear la ecuación:

$$t_{ij} = (t_{i+1j} + t_{ij+1} + t_{i-1j} + t_{ij-1})/4$$

$$4t_{ij} = t_{i+1j} + t_{ij+1} + t_{i-1j} + t_{ij-1}$$

$$-4t_{ij} + t_{i+1j} + t_{ij+1} + t_{i-1j} + t_{ij-1} = 0$$

Nos queda que los coeficientes de la celda actual es -4 y de las contiguas un 1, igualando todo a cero. Por lo tanto en cada fila de la matriz calculamos la posición que deberia queda cada celda contigua en la matriz aplanada, quedando el sistema de ecuaciones de la siguiente forma:
(donde los espacios vacios representan blancos)

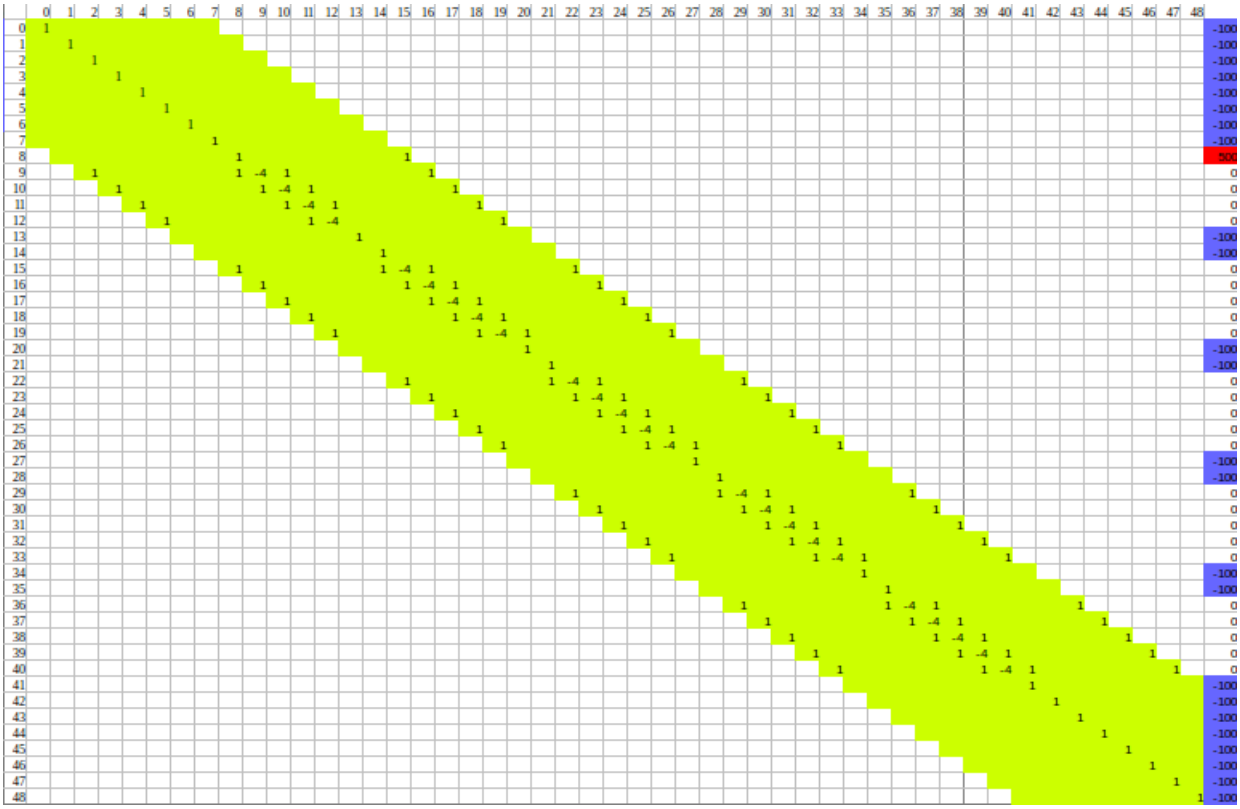


Figura 1: Matriz de ecuaciones del ejemplo

Analizando la matriz del sistema de ecuaciones que se genera a partir de la representación del Parabrisas observamos que presenta las caracteristicas de un tipo de matriz llamada "*banda*", la cual se distingue por tener todas las celdas en 0 salvo en la diagonal ensanchada, es decir, que p celdas a la izquierda y q a la derecha de la diagonal a lo sumo tienen algunos o todos los valores distintos de 0. Este es un hecho importante ya que las matrices banda tienen caracteristicas especiales de las cuales es posible la optimización temporal y espacial para resolver el sistema de ecuaciones con eliminación gausseana.

3.2.1. Optimización

El problema de representar la matriz original es que la mayoría de los valores son 0 y solo importan los elementos de la banda de la matriz, por lo tanto una forma de optimizar espacialmente es solo guardar la banda, con lo que se logra reducir considerablemente el espacio en esta representación, ya que suponiendo que se tiene un parabrisas de n filas y m columnas, el tamaño de la matriz original sería de $(n*m)^2$, mientras que con la optimización de matriz banda queda de tamaño $(n*m)*(2*m+1) \approx (n*m^2)$.

El método para construir la representación optimizada de la matriz banda es simple, se guarda la banda en una matriz pero rotandola de forma que quede vertical, quedando en el centro los elementos de la diagonal dependiendo de lo que haya en esa posición, ya que en el caso de que sea vacía esta deberá tener coeficientes de las posiciones adyacentes. Además se adiciona una columna más para representar la temperatura en esa posición. Aúnque uno podría pensar que ya que solo hay 4 elementos no nulos máximo en cada fila se podria optimizar guardando esos 4 coeficientes, pero debido a que luego se le aplicará el método de Back Substitution de la Eliminación Gausseana como método para resolver el sistema, esto hará que se generen valores no nulos dentro de la banda, por lo tanto el tamaño de esta matriz deberá ser lo suficiente como para poder trabajar sobre él y almacenar los valores parciales. La matriz banda tendra hasta un maximo de 5 valores no nulos por fila dependiendo el tipo de posición de la misma, ya que si es la posición hace

referencia al borde frío, esta fila tendra un 1 como valor central y un -100 en la columna de resultados, esto se deduce a partir de la expresión

$$T(x, y) = -100^o\text{C} \qquad \text{si } x = 0, a \text{ ó } y = 0, b. \tag{1}$$

El caso para el cual la posición contenga una sanguijuela será similar al de frío, ya que deberá respetar la ecuación

$$T(x, y) = temp_{sanguijuela} \tag{2}$$

Y por último, en base a las ecuaciones que deben respetar las posiciones vacías vistas anteriormente, en la posición central que representa a la posición actual quedará un -4 y en las de arriba-abajo-izquierda-derecha un 1 en cada una, siendo referenciadas en la matriz banda por las posiciones de la izquierda y derecha de la central, y las que estan a distancia de la banda, por lo tanto, en las puntas de la fila de matriz, y al final de todo un 0 en la columna de resultado. Por lo tanto, cada fila que represente a una posición vacía tendra una distribución inicial de la siguiente forma:

$$\begin{pmatrix} 1 & 0 & \dots & 0 & 1 & -4 & 1 & 0 & \dots & 0 & 1 \end{pmatrix}$$

El algoritmo consta de dos etapas, la construcción de la matriz que representara la banda y la resolución del problema utilizando la misma.

Construcción de la matriz banda

```
por cada posición del parabrisas //O(N*M)
       pos = fila posicion + columna posicion * ancho;

              si en esa posición hay una SANGUIJUELA:

                    matrizBanda[pos][ancho] = 1;
                    resultados[pos]         = temp_sanguijuela;

              si esa posición es borde FRIO:

                    matrizBanda[pos][ancho] = 1;
                    resultados[pos]         = -100;

              en caso de que esa posición sea VACIA:

                    matrizBanda[pos][ancho]   = -4;
                    matrizBanda[pos][ancho-1] = 1;
                    matrizBanda[pos][0]       = 1;
                    matrizBanda[pos][ancho+1] = 1;
                    matrizBanda[pos][ancho*2] = 1;
                    resultados[pos]           = 0;
```

Continuando con el ejemplo que propusimos cuando empezamos esta sección, la matriz banda resultante sería la siguiente:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0								1									-100
1								1									-100
2								1									-100
3								1									-100
4								1									-100
5								1									-100
6								1									-100
7								1									-100
8								1									500
9	1						1	-4	1							1	0
10	1						1	-4	1							1	0
11	1						1	-4	1							1	0
12	1						1	-4	1							1	0
13								1									-100
14								1									-100
15	1						1	-4	1							1	0
16	1						1	-4	1							1	0
17	1						1	-4	1							1	0
18	1						1	-4	1							1	0
19	1						1	-4	1							1	0
20								1									-100
21								1									-100
22	1						1	-4	1							1	0
23	1						1	-4	1							1	0
24	1						1	-4	1							1	0
25	1						1	-4	1							1	0
26	1						1	-4	1							1	0
27								1									-100
28								1									-100
29	1						1	-4	1							1	0
30	1						1	-4	1							1	0
31	1						1	-4	1							1	0
32	1						1	-4	1							1	0
33	1						1	-4	1							1	0
34								1									-100
35								1									-100
36	1						1	-4	1							1	0
37	1						1	-4	1							1	0
38	1						1	-4	1							1	0
39	1						1	-4	1							1	0
40	1						1	-4	1							1	0
41								1									-100
42								1									-100
43								1									-100
44								1									-100
45								1									-100
46								1									-100
47								1									-100
48								1									-100

Figura 2: Matriz banda del ejemplo

Aquí se puede observar mejor como se distribuyen los coeficientes de la matriz original que pertenecen a la banda sobre esta nueva representación de ella, y como se "*verticaliza*" la misma. Como sucedió anteriormente, los espacios en blanco representan los valores nulos.

Hasta este punto tenemos construida la representación de la matriz original en una forma que optimice el almacenamiento de la misma, ahora solo queda resolverla de una forma también optima.

Resolución del sistema

Este algoritmo se basa basicamente en el algoritmo de eliminación gausseana para triangular y luego realizar un back sustitution para obtener el valor de los coeficientes. Lo que es remarcable es el hecho de como trabaja en la matriz banda vertical como si fuese la banda en diagonal de la matriz original.

```
//PRIMERO LA HAGO TRIANGULAR SUPERIOR
PARA CADA FILA DE LA BANDA, DE ARRIBA A ABAJO

   SI NO ES VACIO SE QUE ABAJO HAY TODO CERO, CONTINUO
   SI ES VACIA DIAGONALIZO:

          centro = bandMatrix[i][n];
          actual = bandMatrix[i+h][n-h];
          multiplicador = actual / centro;

          fila[i+h] - fila[i];

          resultado[i+h] -= resultado[i] * multiplicador;

 // EN ESTE PUNTO YA TENGO LA MATRIZ TRIANGULADA SUPERIORMENTE, AHORA SOLO FALTA REALIZAR BACK SU
 PARA CADA FILA DE LA BANDA, DE ABAJO A ARRIBA
        fila = i / n;
        for( int h = 1; h <= n; h++){
        // COMO ES BANDA ME FIJO SI EN LA DIAGONAL IZQ INF HAY DISTINTO DE 0 PARA PIVOTEAR

             centro = bandMatrix[i][n];
             actual = bandMatrix[i-h][n+h];

             multiplicador = actual / centro;

             fila[i-h] - fila[i];
             resultado[i-h] -= resultado[i]* multiplicador;

             // y por ùltimo actualizo los valores del parabrisas con las temperaturas actuales

             Parabrisas[i/ancho +1][j % ancho].temp = actual
```

4. Eliminar sanguijuelas

La resolución del problema de evitar el punto crítico en el centro del parabrisas consiste en la eliminación de sanguijuelas. Suponiendo la existencia de n sanguijuelas, los cantidad de subconjuntos a probar son 2^n. Es un número que crece rápidamente y si bien se puede resolver por fuerza bruta (ayudados por Backtracking y derivados), suponemos que encontrar el mínimo tiene un orden exponencial en la cantidad de sanguijuelas.

Finalmente, planteamos dos tipos de solución a modo de comparación. Estos serán detallados más adelante. Ambas son heurísticas por lo mencionado anteriormente.

4.1. Punto Crítico

En un parabrisas real el punto crítico se determina por el punto exacto del medio del parabrisas, en la posición (largo/2, ancho/2), pero en nuestra estructura de datos que representa al parabrisas esta puede no llegar a tener un punto exacto dependiendo de la discretización que tenga el mismo. Por lo tanto tenemos dos casos, el caso "trivial" es hay un punto exacto donde cae el punto crítico en el parabrisas, en ese caso elegimos ese. En el caso contrario, cuando tomamos la mitad del largo y ancho del parabrisas para elegir el punto crítico este no sera un número entero, por lo tanto redondearemos para abajo y el punto crítico que consideraremos se priorizara teniendo el cuenta el que esta más arriba a la izquierda.

4.2. Solución Greedy

Esta solución consiste en ir seleccionando las sanguijuelas más cercanas al punto medio hasta que el punto central esté por debajo de 235 C°. Al no ser una solución exacta es posible que eliminar la más cercana al centro no pertenezca a la mejor solución, pero es intuitivo suponer que la sanguijuela que más cause calor al punto medio esté cerca del mismo.
Lo primero que se realiza es ordenar las sanguijuelas según su distancia al centro. Para esto utilizamos la función *sort*() de C++ que utiliza como comparador el valor pre-calculado de distancia al centro. El orden de esta parte es $O(n.log(n))$ en donde n es la cantidad de sanguijuelas.
Al ser una solución golosa (greedy), elige la siguiente sanguijuela siempre y cuando no se haya enfríado el punto medio y haya salido del estado crítico. La elección está dada por la sanguijuela más cerca todavía no elegida en una previa iteración.
Suponiendo que la complejidad de rehacer el cálculo es RA, la complejidad final es: $O(n.log(n) + n * RA)$. Notar que esa última n es en el peor caso que tengamos que eliminar todas las sanguijuelas, pero debería ser un $k < n$.

Algorithm 1 Solución Greedy

```
1: orderLeachesByDistanceToCenter()
2: leachesToRemove ← []
3: do
4:     leachesToRemove.add(this-¿removeFirstNotRemovedLeachOrderedCentrically())
5: while notisCooledDown()
6: return leachesToRemove;
```

Algorithm 2 isCooledDown()

```
1: recalculateByBandMatrix();
2: return (matrix.centerPoint ¡Ts)        ▷ TSeslatemperaturamaximaquesoportaelcentro
```

Algorithm 3 orderLeachesByDistanceToCenter()

1: *sort(leaches).by(lambda —leach,otherLeach— leach.distanceToCenter ¡otherLeach.distanceToCenter);*
2: *return (matrix.centerPoint ¡Ts); //TS es la temperatura máxima que soporta el centro*

4.2.1. Solución Random

En esta solución seleccionamos de nuestro array de posiciones de sanguijuelas (que es el array recibido por parametro, osea con las posiciones sin discretizar) una al azar y la eliminamos. Luego ejecutamos de nuevo el cálculo de las temperaturas y chequeamos si el punto crìtico esta por debajo del 235 C°.Si no lo está, elegimos otra al azar y repetimos el proceso hasta que lo esté.

Algorithm 4 RandomSolution

1: **do**
2: this-¿randomKill()
3: **while** *notisCooledDown*()
4: *return leachesToRemove;*

Algorithm 5 randomKill()

1: *randomRemoveFrom(allLeaches);* ▷ All Leaches is loaded with matrix
2: *recalculateByBandMatrix();*

5. Experimentación Y Resultados

5.1. Introducción

Realizamos los experimentos en tres computadoras con procesadores similares y se trató de mantener cualquier otro programa cerrado. Igualmente cada conjunto de test fue resuelto en la misma máquina para que las diferencias sean lo más dependendientes a los inputs y los procesos posible.

5.2. Caso básico

A continuación plantearemos distintos casos de parabrisas con distintas granularidades que utlizaremos durante las pruebas.

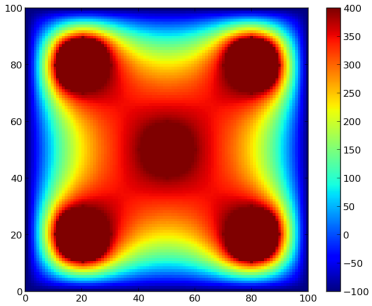


Figura 3: Granularidad 1

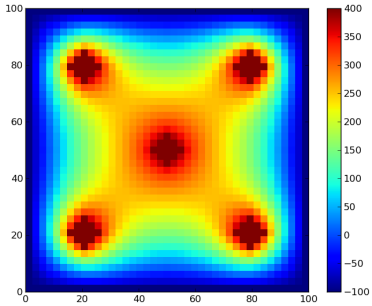


Figura 4: Granularidad 2.5

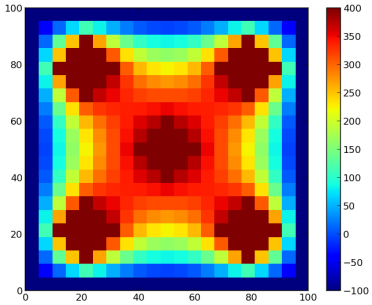


Figura 5: Granularidad 5

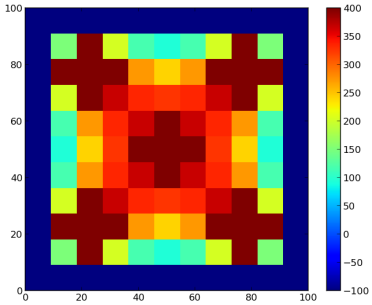


Figura 6: Granularidad 10

5.2.1. Resultados Random

A continuación observaremos un caso en la que la única sanguijuela que se debe eliminar es la del centro. Esta se encuentra exactamente en el punto central. Adicionalmente hay otras 4 sanguijuelas en las puntas. Matar a estas no soluciona nada ya que la del centro esta aplicando una temperatura constante de 400 C^{o}. Lo observaremos con 4 discretizaciones distintas para tener un mas amplio panorama.

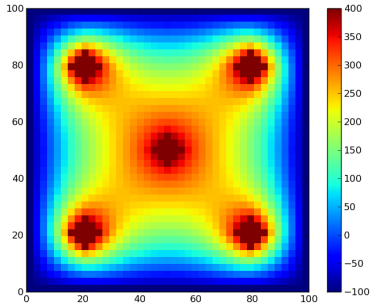


Figura 7: Granularidad 2.5

Ahora veamos que obtenemos al aplicarle distintas veces el algoritmo de solución random explicado en el desarrollo (4.2.1).

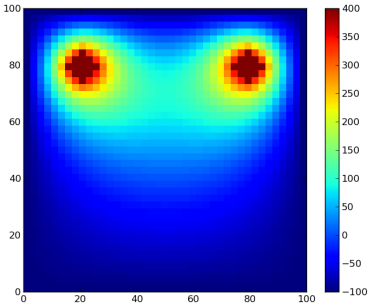


Figura 8: Resultado primer corrida

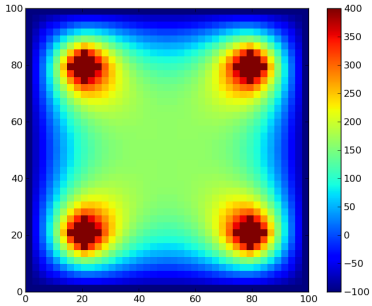


Figura 9: Resultado segunda corrida

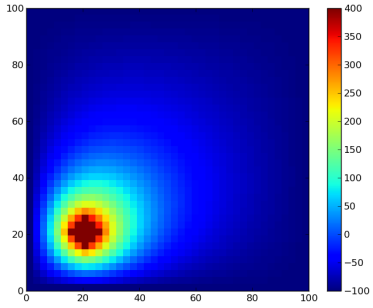


Figura 10: Resultado tercer corrida

Como podemos observar la solución óptima (la que menos sanguijuelas elimina), es la segunda. Pero como esta solución es completamente random en la primera corrida mata 2 sanguijuelas antes de elegir la correcta

y en la tercera 4. Sólo en la segunda elije en el primer intento la sanguijuela correcta. Se aplicó también para todas las granularidades antes mencionadas y se observó el mismo comportamiento erratico a la hora de matar sanguijuelas.

5.2.2. Resultados Greedy

Veamos ahora para la misma matriz como se comporta nuestro otro algoritmo.

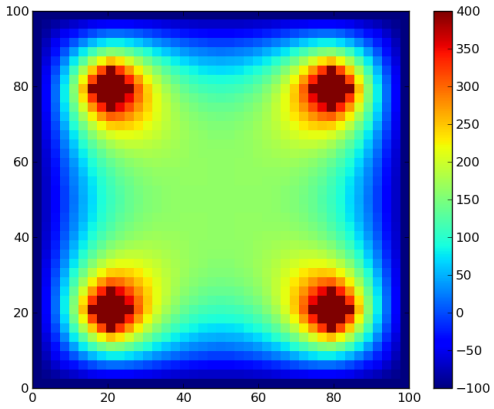


Figura 11: Resultado corrida con greedy

En todas las corridas obtuvimos el mismo resultado y demoraron lo mismo. Era de esperarse, ya que efectivamente en este caso, la sanguijuela a eliminar era la del medio.

5.3. Resultados Banda VS Gaussiano

Veamos ahora que realmente valió la pena la mejora al saber que era matriz banda en cuanto a tiempos en comparación con la eliminación gaussiana. Anotamos que agregar un análisis de resultados no vale la pena ya que son el mismo

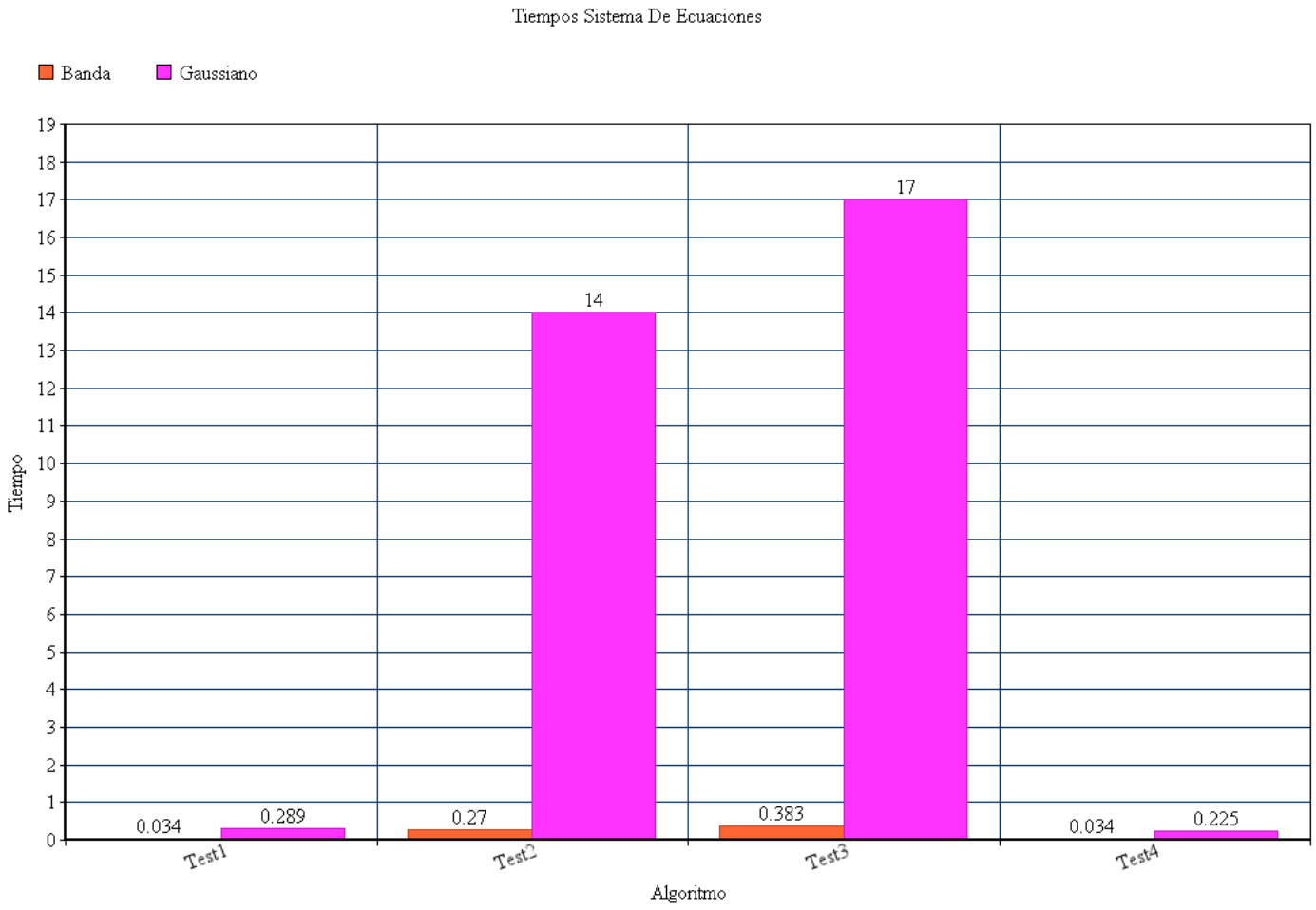


Figura 12: Gauss VS Banda

5.4. Caso solución no es el mas cercano

A continuación analizaremos un caso en el cual se debe eliminar una sanguijuela que no es la más cercana para efectivamente poder reducir la temperatura en punto crítico. Los datos exactos de este test son:
Medidas: 100 x 100
discretización:1
radio: 0.9
tempreratura sanguijuelas:450
Posiciónes sanguijuelas:(49, 38);(57, 60);(57, 37);(39, 37);(36, 47)

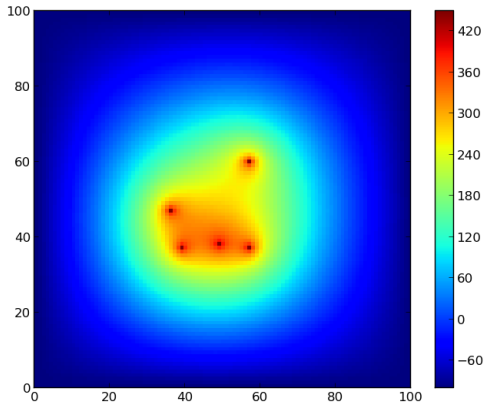


Figura 13: Parabrisas granularidad 1

Como podemos observar en las imágenes que están debajo, la solución greedy no es la óptima ya que mató a una sanguijuela extra. Esto es debido a que la principal radiación de calor era producida por el conjunto de sanguijuelas que están debajo del punto crítico y no por la mas cercana. Si eliminamos la sanguijuela central de aquel conjunto la radiación de calor emitida por él disminuye drásticamente en el punto crítico produciendo que la temperatura esté por debajo de los 235 grados sin necesidad de eliminar a las mas cercana.

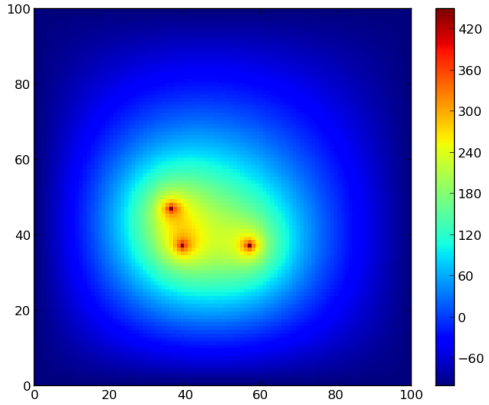


Figura 14: Solución Greedy

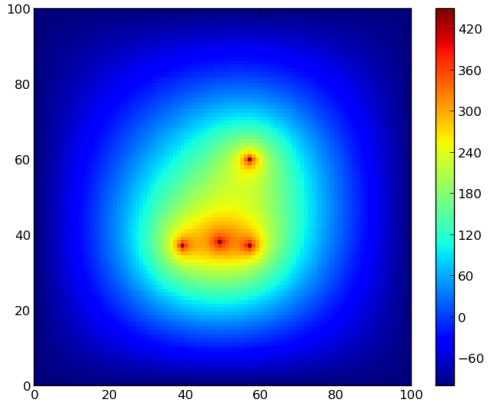


Figura 15: Solución Random 1

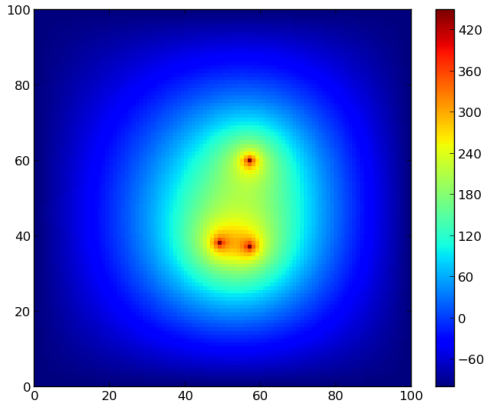


Figura 16: Solución random 2

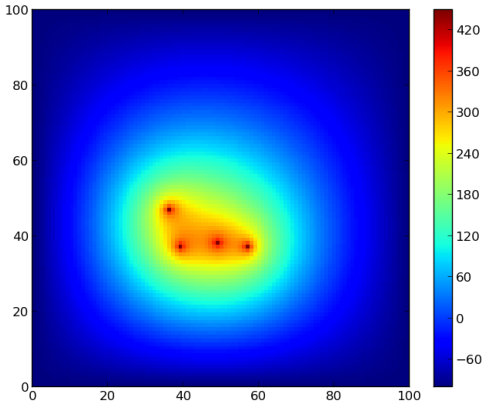


Figura 17: Solución random 3

El greedy mató primero a la sanguijuela de la posición (49,38) ya que era la mas cercana. Sin embargo si mataba primero la (57,60) la temperatura en el punto crítico disminuía lo suficiente. Esto pudimos corroborarlo al correr el test nuevamente sin esta última sangijuela y comprobando que tanto el greedy como random no mataban ninguna sangijuela ya que el punto crítica estaba por debajo de 235 grados celcius. El random por otro lado nos dió 3 resultados distintos de los cuales dos son válidos. Cabe destacar sin embargo que el erroneo nos lo devolvió 2 veces, por lo que podemos decir que tuvo un 50 % de efectividad.

5.4.1. Variando la granularidad

Como vimos anteriormente, la temperatura en el punto crítico varía según la granularidad que le demos. Veamos entonces que pasa si aumentamos la granularidad (0.8 en vez de 1).

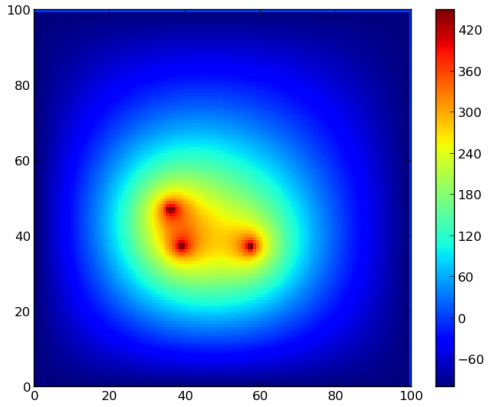


Figura 18: Solución Greedy

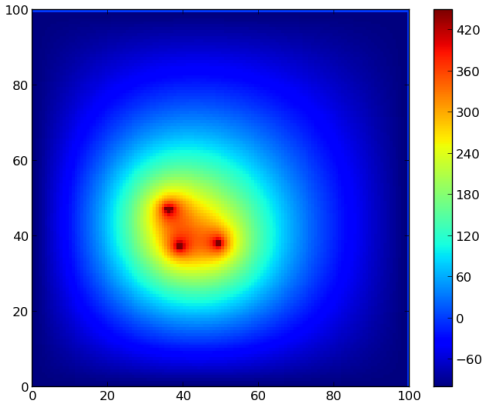


Figura 19: Solución real

Como le aumentamos la granularidad aumentó la temperatura del punto central de tal forma que provocó que tampoco bastase eliminando una sola sangijuela, obteniendo así la misma cantidad de sangijuelas tanto en al solución real como en la greedy.

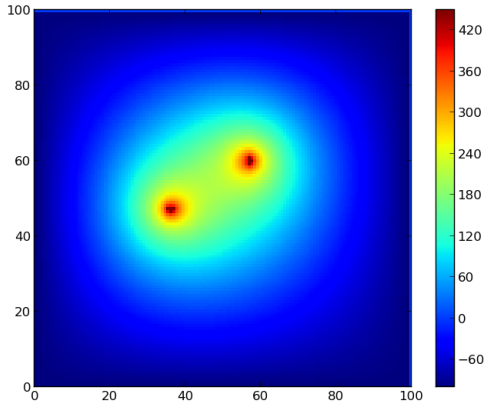


Figura 20: Solución Greedy

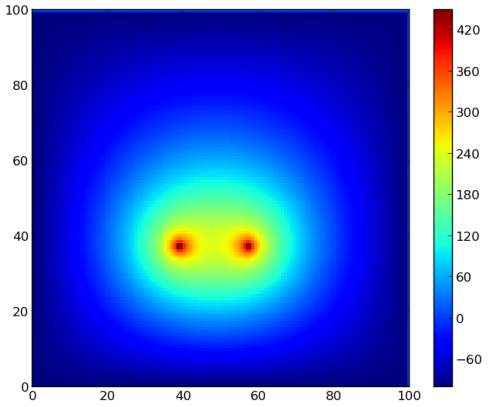


Figura 21: Solución real

6. Discusión

6.1. Espiral vs centrico

Durante el desarrollo del algoritmo Greedy + LocalSearch, nuestro primer acercamiento fue recorrer la matriz de forma espiral. El primer problema que encontramos fue que teníamos un cálculo muy grande para determinar dado un punto del parabrisas, qué sanguijuela era y que recorríamos partes de la matriz que no importaban (en especial los bordes). A su vez, tendíamos a alejarnos del objetivo principal que era preocuparnos por el centro. Es por eso que decidimos ordenar las sanguijuealas según su distancia al centro. Para ahorrar tiempo computacional, este dato está calculado en el momento de la creación de la sanguijuela para su posterior ordenamiento.

6.2. ¿Por que random?

Cuando empezamos a pensar otras soluciones alternativas a la golosa. Pensamos primero en soluciones totalmente suboptimas, como ir sacando desde el borde hacia el centro. Pero considerabamos que era tan malo que quitaba cualquier tipo de anàlisis serio. Luego también pensamos en ir sacando una del centro y una del borde alternadamente, pero no tenía tampoco algún justificativo conceptual alguno. Ahí fué entonces cuando pensamos en la solución random. Que claramente no responde a ningún compartamiento definido. Nos resultó mejor esto ya que las otras opciones pensadas o directamente estaban mal propuestas a proposito o no tenían sentido alguno

6.3. Por tiempos no llegamos

Una mejora que quisimos agregar que no llegamos por tiempos fue no utilizar los bordes y las sanguijuelas, y solo tenerlas como coeficientes reemplazados en el resultado de las celdas que necesiten de los adyacentes para su cálculo. Hubo una realización parcial en el desarrollo de la matriz banda

7. Conclusiones

7.1. Random vs Greedy

Como observamos en los resultados para el caso en el que la sanguijuela/s se encuentran en el medio claramente es mejor el greedy, ya que siempre mata a la sanguijuela correcta a diferencia del random que sólo lo hizo en uno de los 3 intentos. Por otro lado si tenemos en cuenta los tiempos que pueden llegar a demorar recalcular las temperaturas, esto le da todavìa una mayor importancia, ya que se reduce considerablemente el tiempo de ejecución.

Por otro lado en el segundo caso, donde es un poco mas conflictivo para la lógica de ir sacando los del medio, no sólo sigue siendo claramente mejor que el random sino que ademàs también da la mejor por solución. Por lo tanto podemos concluir que no sólo es un mejor algoritmo que el random sino que además se comporta de forma óptima para los casos que pensabamos iban a ser mas conflictivos.

8. Referencias

- Una matriz de n por m elementos, es una matriz cuadrada si el número de filas es igual al número columnas, es decir, n = m y se dice, entonces que la matriz es de orden n
- En matemáticas una matriz se le llama matriz banda cuando es una matriz donde los valores no nulos son confinados en un entorno de la diagonal principal, formando una banda de valores no nulos que completan la diagonal principal de la matriz y más diagonales en cada uno de sus costados.
- Un algoritmo Greedy (también conocido como ávido, devorador o goloso) es aquel que, para resolver un determinado problema, sigue una heurística consistente en elegir la opción óptima en cada paso local con la esperanza de llegar a una solución general óptima. Este esquema algorítmico es el que menos dificultades plantea a la hora de diseñar y comprobar su funcionamiento.