



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Sanguijuelas

25 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	3
2. Introducción teórica	4
2.1. Algoritmo de eliminación gaussiana	4
2.2. Matriz banda	6
3. Desarrollo	7
3.1. Matriz Banda	7
3.1.1. Optimización espacial	7
3.1.2. Optimización temporal	8
3.1.3. Ejemplo	9
3.2. Eliminación Gaussiana	11
4. Soluciones	11
4.1. Punto Crítico	11
4.2. Solución Greedy	12
4.2.1. Solución Random	13
5. Experimentación Y Resultados	15
5.1. Introducción	15
5.2. Caso básico	15
5.2.1. Resultados Random	16
5.2.2. Resultados Greedy	18
5.3. Resultados Banda VS Gaussiano	19
5.4. Caso solución no es el mas cercano	19
6. Discusión	21
6.1. Espiral vs centrico	21
6.2. ¿Por que random?	21
6.3. Por tiempos no llegamos	21

7. Conclusiones	22
7.1. Random vs Greedy	22
8. Referencias	23

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 utilizada 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 consiste en restar multiples de filas entre sí, comenzando desde la primera, con el objetivo de lograr todos ceros por debajo de la diagonal y numeros distintos 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

$$F2 = F2 - 1/4 (F1) \text{ y } F3 = F3 - 1/2 (F1)$$

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

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 fácilmente 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

Analizando la matriz del sistema de ecuaciones que se genera a partir de la representación del Parabrisas observamos que presenta las características 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. En nuestro caso, al construir la matriz nos basamos en el tipo de celda, si es el borde frío o una sanguijuela el coeficiente respectivo será un 1 en la diagonal igualado a la temperatura correspondiente, pero en el caso de ser una celda vacía en la diagonal irá un -4, a los costados un 1 y representando la fila de "arriba" y "abajo" del parabrisas irán un 1 a m columnas de distancia de la diagonal. Finalmente la matriz resultante quedará con una banda de ancho $m+m$, con m el ancho del parabrisas.

Aca podemos ver un ejemplo de una matriz banda:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & 0 \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} & a_{67} \\ 0 & 0 & 0 & 0 & a_{75} & a_{76} & a_{77} \end{pmatrix}$$

Figura 1: Matriz banda 7x7

3. Desarrollo

3.1. Matriz Banda

Como explicamos en la introducción de la matriz banda, llegamos a la conclusión de que la matriz de ecuaciones de la representación del parabrisas tenía forma de la denominada "matriz bandajada" respectiva banda era de tamaño $m+m+1$ con m siendo el ancho del parabrisas. Este es un hecho importante ya que las matrices banda tienen características especiales de las cuales es posible la optimización temporal y espacial para resolver el sistema de ecuaciones con eliminación gaussiana.

3.1.1. Optimización espacial

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 quedaría de tamaño $(n*m)*(2*m+1)$.

El método para construir la representación optimizada de la matriz banda es simple, se guarda la diagonal en una matriz, 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 nos dimos cuenta que se podía representar la matriz de tal forma que presente una mejor optimización temporal, y consiste en no poner los coeficientes de las celdas vacías adyacentes si estas no son vacías, en tal caso al vector de resultados le restamos la temperatura de la misma, generando que luego no sea necesario considerar las posiciones no vacías para la eliminación gaussiana a la hora de resolver el sistema, y como consecuencia la no necesidad de intercambiar filas al triangular.

```

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

    si en esa posición hay una SANGUIJUELA:

        bandMatrix[pos][ancho] = 1;
        resultados[pos] = ts;

    si esa posición es borde FRI0:

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

    en caso de que esa posición sea VACIA:
        bandMatrix[pos][ancho] = -4;
        resultados[pos] = 0;

    si la izquierda no es vacía
        resultados[pos] -= temperatura del de la izquierda;
    else
        bandMatrix[pos][ancho-1] = 1;

```



```

si la de arriba no es vacia
    resultados[pos] -= temperatura del de arriba;
else
    bandMatrix[pos][0] = 1;

si la de la derecha no es vacia
    resultados[pos] -= temperatura del de la derecha;
else
    bandMatrix[pos][ancho+1] = 1;

si la de abajo no es vacia
    resultados[pos] -= temperatura del de abajo;
else
    bandMatrix[pos][ancho*2] = 1;

```

3.1.2. Optimización temporal

Para resolver el sistema de ecuaciones lo que se aplica es una eliminación gausseana optimizada y sin la necesidad de intercambiar filas al momento de generar la matriz triangular superior debido a la optimización que se aplicó en el momento de su creación. La eliminación gausseana fue optimizada teniendo en cuenta la forma de la matriz banda que se genera a partir del parabrisas y conociendo sus características, ya que al momento de resolverla detecta cuando la posición es no vacia y no necesita triangular en esa fila debido a que ya sabe que abajo son todos ceros. Por la forma que se almacena la matriz banda lo primero que se hace es triangular en diagonal a la izquierda entre las filas vacias, dando como resultado una matriz triangular superior, luego solo le restará hacer una back substitution optimizada, que consiste en realizar algo parecido a lo anterior pero esta vez de abajo hacia arriba, es decir, resta fila a fila "diagonalizando" en cada fila que se posiciona como sabe que solo tiene un coeficiente igualado a un resultado.

```

//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;

//BACK SUBSTITUTION
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 DIST

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

    multiplicador = actual / centro;

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

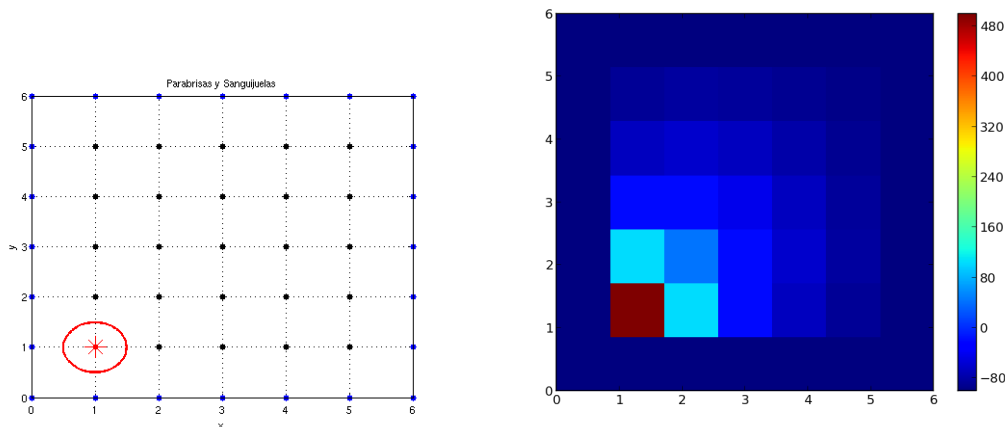
    lleno el parabrisas con las temperaturas actuales

```

3.1.3. Ejemplo

Aca podemos ver un ejemplo para ver como funciona lo explicado en esta sección. 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 así (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 $t_{ij} = (t_{i+1,j} + t_{i-1,j} + t_{i,j+1} + t_{i,j-1}) / 4$!!! ACA LATEXEAR !!! 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 debería quedar cada celda contigua en la matriz aplanada.

Pero en vez de eso al optimizar la matriz banda se genera esta matriz:

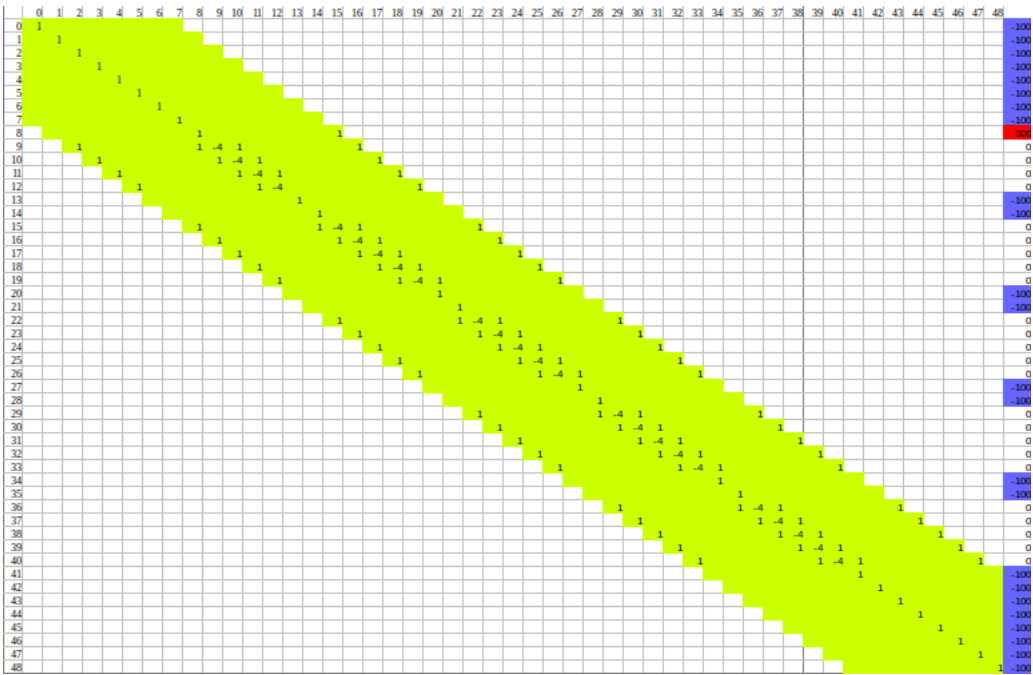


Figura 2: Matriz de ecuaciones del ejemplo

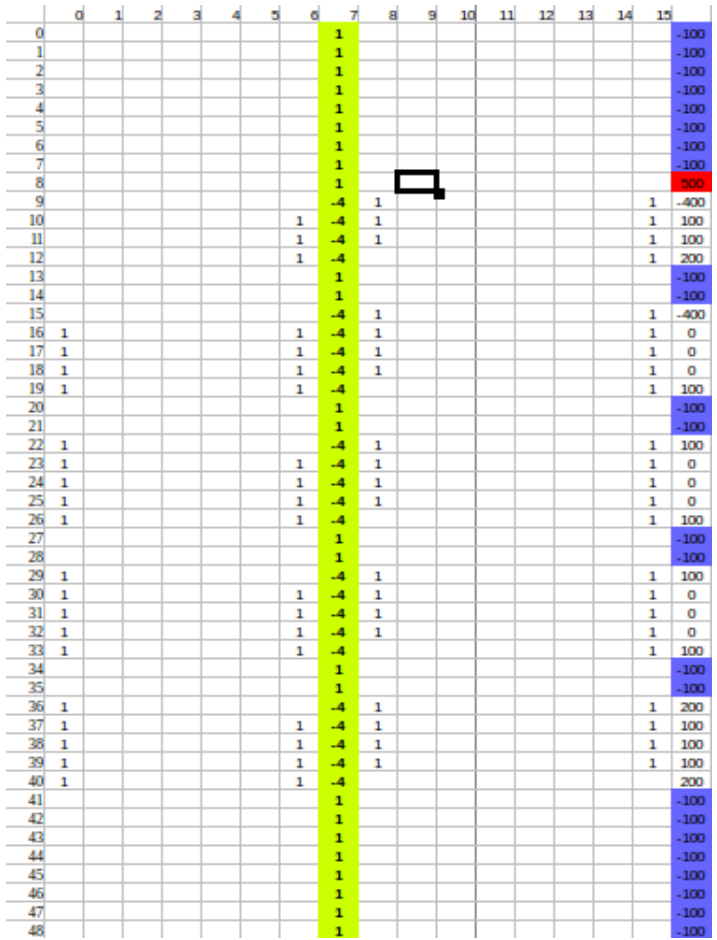


Figura 3: Matriz banda del ejemplo

3.2. 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()
        }
    }
}

```

4. Soluciones

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 ($\text{largo}/2$, $\text{ancho}/2$), 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 será un número entero, por lo tanto redondearemos para abajo y el punto crítico que consideraremos se priorizará teniendo en cuenta el que está 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 $O(n \cdot \log(n))$ en donde n es la cantidad de sanguijuelas.

Al ser una solución golosa (greedy), elige la siguiente sanguijuela siempre y cuando no se haya enfriado 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: $O(n \cdot \log(n) + n * RA)$. 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  $\leftarrow$  []
3: do
4:   leachesToRemove.add(this-¿removeFirstNotRemovedLeachOrderedCentrically())
5: while not isCooledDown()
6: return leachesToRemove;

```

Algorithm 2 *isCooledDown()*

```

1: recalculateByBandMatrix();
2: return (matrix.centerPoint ;Ts)  $\triangleright$  TS es la temperatura máxima que soporta el centro

```

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 crítico 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 not isCooledDown()  
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 dependientes a los inputs y los procesos posible.

5.2. Caso básico

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

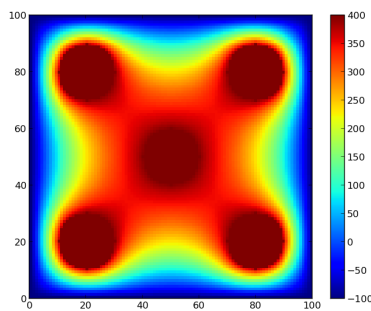


Figura 4: Granularidad 1

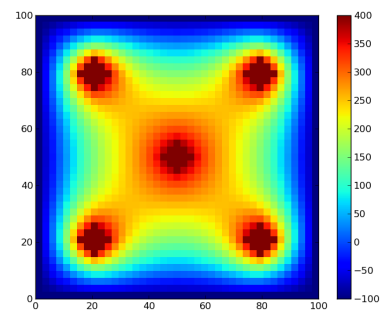


Figura 5: Granularidad 2.5

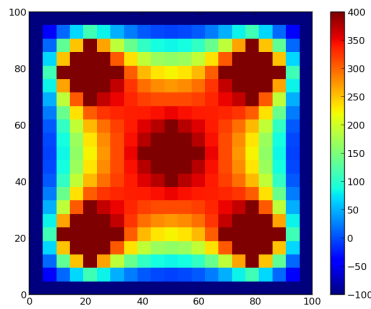


Figura 6: Granularidad 5

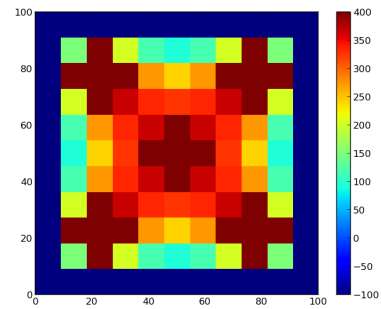


Figura 7: Granularidad 10

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°. Lo observaremos con 4 discretizaciones distintas para tener un mas amplio panorama.

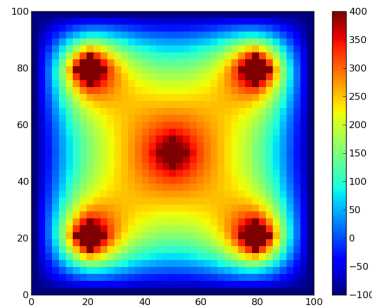


Figura 8: Granularidad 2.5

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

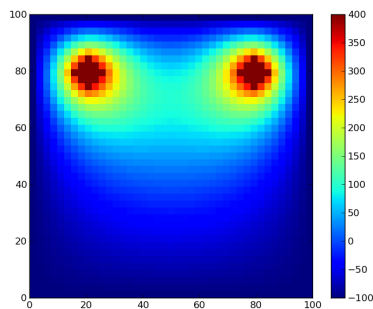


Figura 9: Resultado primer corrida

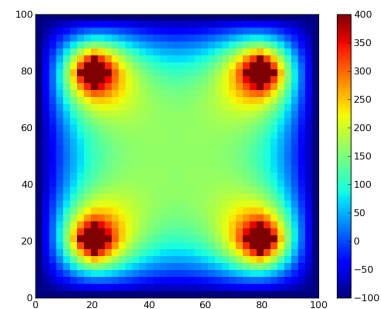


Figura 10: Resultado segunda corrida

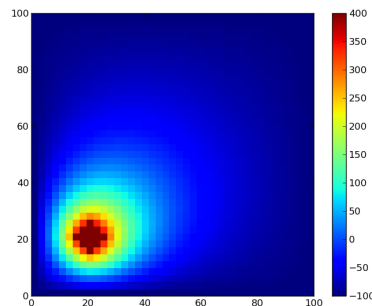


Figura 11: Resultado tercer corrida

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 elige en el primer intento la sanguijuela correcta. Se aplicó también para todas las granularidades antes mencionadas y se observó el mismo comportamiento errático a la hora de matar sanguijuelas.

5.2.2. Resultados Greedy

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

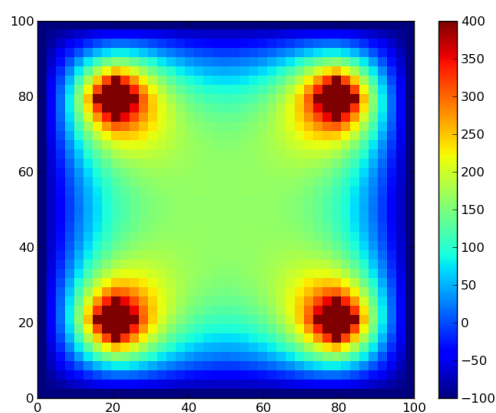


Figura 12: Resultado corrida con greedy

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

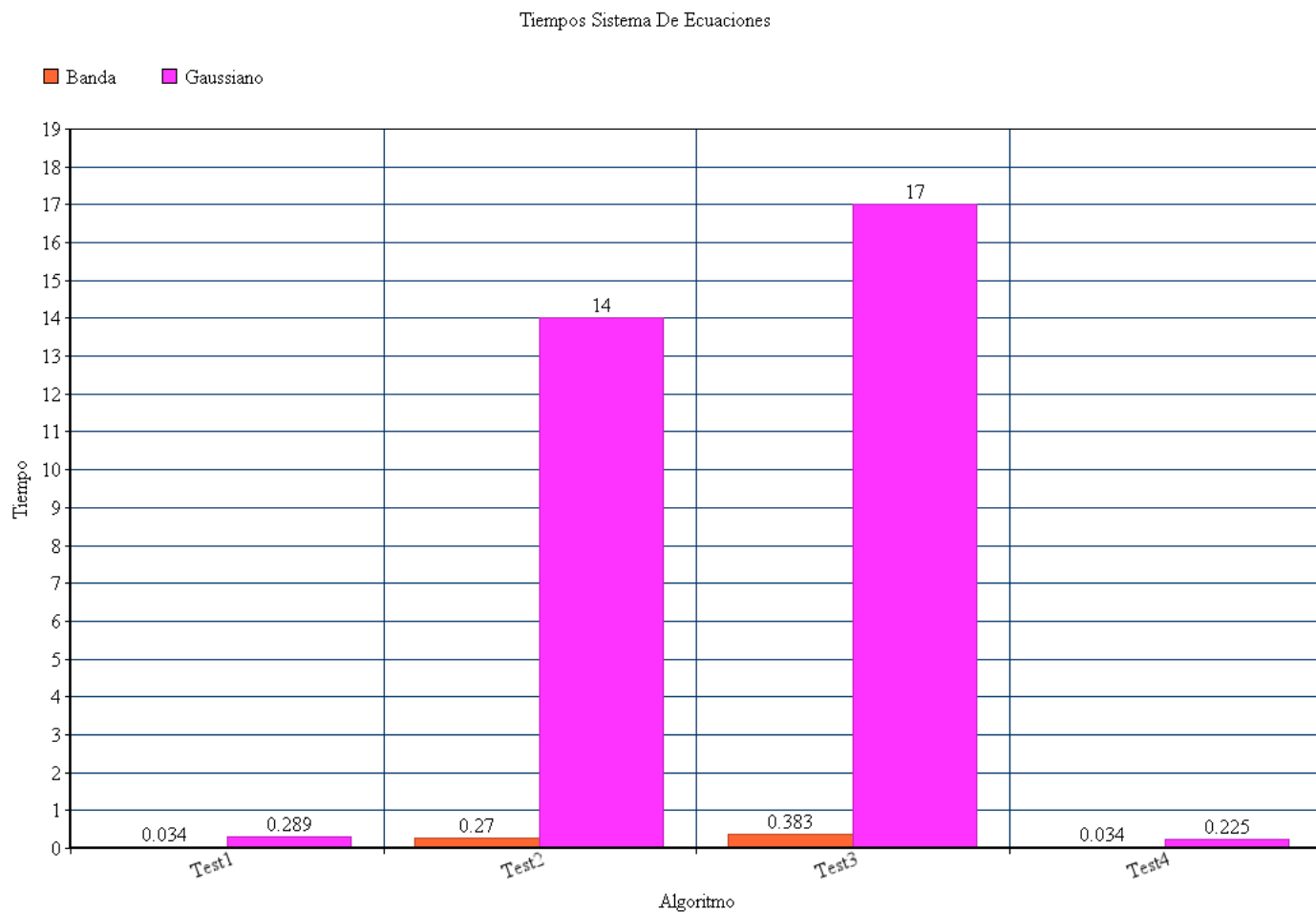


Figura 13: Gauss VS Banda

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.

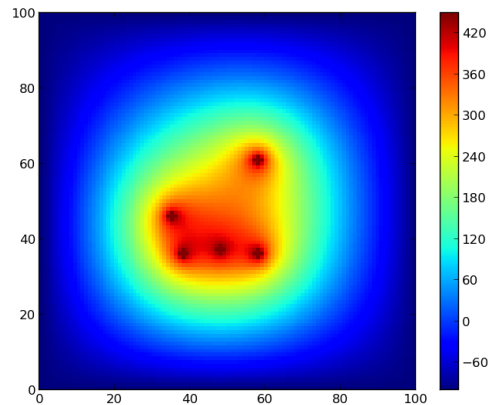


Figura 14: Parabrisas granularidad 1

Como podemos observar en las imágenes que estan 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 cercanas.

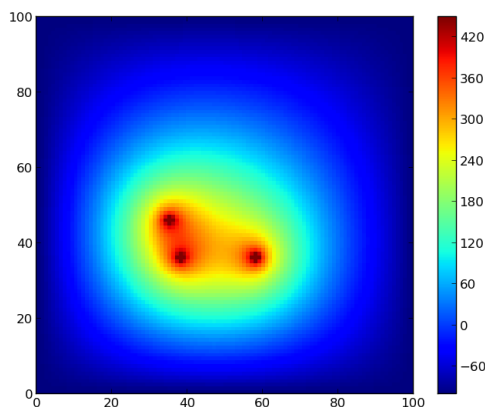


Figura 15: Solución Greedy

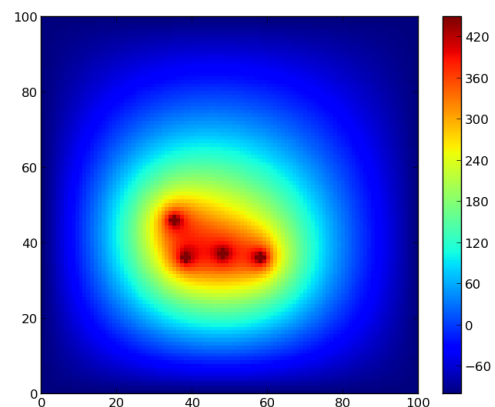


Figura 16: Solución real

Los datos exactos de este test son:

Medidas: 100 x 100 discretización:1 radio: 1.40 temperatura sanguijuelas:450 Posiciones sanguijuelas:
48 37 58 61 58 36 38 36 35 46

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 recorriamos 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 análisis 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 necesitan 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 recalculando las temperaturas, esto le da todavía 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 además 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 pensábamos 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 de 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.