

Trabajo Práctico I

10 / 04 / 2015

Métodos Numéricos

Aplicación de métodos básicos de Algebra de Palitos¹ para la resolución de sistemas de ecuaciones lineales.

Integrante	LU	Correo electrónico
Abdala, Leila	950/12	abdalaleila@gmail.com
Albertini, Alejandro	924/12	ale.dc@hotmail.com
Bayardo, Julian		julian@bayardo.com.ar
Niesz, Ignacio		ignacio.niesz@gmail.com

Resumen:

En el presente informe se presenta la implementación, desarrollo y análisis de resultados usados en la resolución de un problema práctico, con las herramientas estudiadas y provistas por la materia. El problema planteado consiste en encontrar, dado un parabrisas refrigerado atacado por sanguijuelas mutantes que liberan calor, la temperatura del centro del mismo y, de ser necesario, devolver la sanguijuela que, al ser eliminada, disminuya en mayor medida la temperatura de este centro.

Luego de realizamos una breve experimentación para analizar la calidad de las soluciones, performance de los algoritmos, realismo del modelado, etc.



Palabras claves:

Sanguijuela, Eliminación Gausiana, Factorización LU, Fórmula de Sherman-Morrison, Matriz Banda

¹ Lease algebra lineal.

Índice

1. Introducción	3
2. Desarrollo	4
2.1. Sobre la implementación	4
2.1.1. Pseudocódigo	7
2.2. Sobre la experimentación	8
3. Experimentación y discución	8
3.1. Experimento 1: Medición de performance respecto a la variación de la discretización	9
3.1.1. Caso A: ¿Ser o no ser?	9
3.1.2. Caso B: Una sanguijuela invoca otra sanguijuela	9
3.2. Experimento 2: Comparación de temperaturas en el punto critico	9
3.2.1. Caso A: ¡Mientras varie la granularidad, sirve!	9
3.2.2. Caso B: Buscando las sanguijuelas perdidas	9
3.3. Experimento 3: Comparación de performances para backtraking al aplicar Sherman-Morrison	10
3.3.1. Caso A: Sobre población de mini-sanguijuelas	10
4. Conclusiones generales	10
5. Bibliografía	10
6. Apéndice A: Enunciado	10
7. Apéndice B: Código fuente	14
7.1. BDouble.h	14
7.2. Matrix.h	16

1. Introducción

En este informe se detalla el diseño e implementación de algoritmos que modelan los métodos de Eliminación Gaussiana y factorización LU, para resolución de sistemas de ecuaciones lineales. También se presentan algoritmos para la aplicación de la fórmula de Sherman-Morrison y una estructura para el almacenamiento de matrices banda, con el fin de mejorar la complejidad espacial y la temporal. Luego de realizamos una breve experimentación para analizar la calidad de las soluciones, performance de los algoritmos, realismo del modelado, etc.

Esto se realiza al resolver un problema práctico mediante el modelado y discretización del mismo. Este problema consiste en, dado un parabrisas P que está siendo atacado por sanguijuelas mutantes, a las que llamaremos S_i , averiguar la temperatura del punto crítico que se halla en el centro de P. P cuenta con un sistema de refrigeración que mantiene el borde a temperatura constante -100°C . Cada sanguijuela S_i cuenta con una posición (x_i, y_i) , un radio r_i y una temperatura t_i . Si un punto del parabrisas, $P_{k,j}$, pertenece al área afectada por algún r_i y no pertenece al borde de P, la temperatura de dicho punto es t_i , donde por área afectada nos referimos a la circunferencia de radio r_i y centro (x_i, y_i) . Si pertenece a más de un radio, es decir $P_{k,j} \in r'_i, i \in (1..l)$, entonces su temperatura será $t_{k,j} = \max_{i \in (1..l)}(r'_i)$. Si el punto $P_{k,j}$ no pertenece a ninguno de los anteriores, su temperatura está determinada por la siguiente fórmula:

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

Además, si el punto crítico supera los 235°C , se desea saber si eliminando tan solo una sanguijuela, y cual es dicha sanguijuela, se puede lograr que la temperatura se reduzca por debajo de este margen, pues de lo contrario el parabrisas se romperá.

Como P tiene una superficie continua y dado que es imposible representar valores continuos en una PC, debemos discretizar nuestro sistema. Por lo tanto vamos a modelar la superficie de P de tal forma que P sea representado por una matriz de temperaturas. Es decir, solo consideraremos los puntos h-distantes para el modelado, donde h es un parámetro del problema. Solo serán afectados por una sanguijuela S_i los puntos h-distantes que estén incluidos en el radio de S_i . Si el radio de una sanguijuela S_i no afecta ningún punto de la discretización, esta no se tendrá en cuenta en la modelización del problema.

Una vez discretizado el problema de esta manera, podemos² estimar la ecuación (1) de la siguiente forma:

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

A partir de este modelo, despejando las ecuaciones de calor, obtenemos un sistema de ecuaciones lineales. Para la resolución del sistema resultante, vamos a aplicar dos métodos. Estos serán Eliminación Gaussiana y factorización LU.

En el caso de que debamos decidir qué sanguijuela eliminar, aplicaremos en primera instancia la técnica de Backtracking. Luego aplicaremos la fórmula de Sherman-Morrison, para optimizar.

Detallando los métodos utilizados, notamos que el método de Eliminación Gaussiana consta de dos pasos: en primer lugar, transforma el sistema original a uno equivalente donde la matriz es triangular superior, con costo $O(n^3)$ donde n es el número de incógnitas. Luego procede a resolver dicho sistema, con costo $O(n^2)$. Por otro lado, el método de descomposición LU descompone el sistema original $Ax = b$ en un sistema del tipo $LUX = b$, donde L es una matriz triangular inferior y U es una matriz triangular superior, con costo $O(n^3)$. Luego resuelve los sistemas $Ly = b$ y $Ux = y$ para obtener el valor de solución x , con costo $O(n^2)$ en cada caso. En términos asintóticos ambos métodos tienen una complejidad $O(n^3)$, sin embargo, la factorización LU presenta una ventaja por sobre EG. Si el sistema se debe resolver para un b diferente, no hace falta realizar nuevamente la factorización, sino que simplemente basta con resolver los dos sistemas triangulares usando las mismas matrices L y U lo que resulta en un costo cuadrático, en lugar del costo cúbico asociado a la Eliminación Gaussiana.

²Explicación al respecto en el apéndice A

Notese ademas que para las matrices de este problema se puede³ aplicar Eliminacion Gaussiana sin pivotear, o lo que es lo mismo, existe Factorización LU

2. Desarrollo

2.1. Sobre la implementación

En principio, y con la intencion de entender el problema en profundidad, planteamos un ejemplo que resolvimos manualmente. A partir de este ejemplo notamos un patron en el comportamiento del sistema. Por lo tanto recrearemos a continuación dicho ejemplo, para dar una noción clara de nuestra evolución en la comprensión de este problema particular.

Asumamos que recibimos un archivo de entrada con los siguientes parametros:

3 3 1 1
0.5 2.5 1 500

entonces el problema modelado es de la pinta⁶:

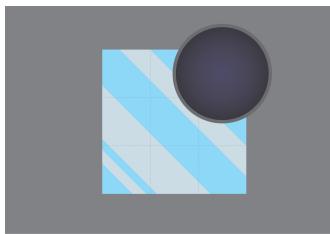


Figura 1: Imagen ilustrativa del parabrisas.

Luego lo representaremos como muestra el grafico a continuacion, donde cada intersección de las lineas representa una variable del modelo, de la cual debemos averiguar su temperatura.

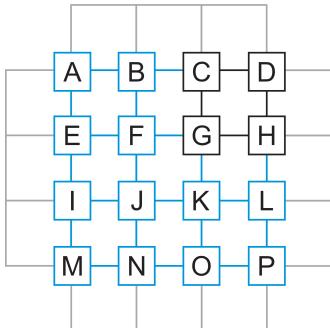


Figura 2: Representación visual del modelo discretizado.

Notese que, si bien C, D y H pertenecen al radio de una sanguijuela, al formar parte del borde estan acondicionados por el sistema de refrigeracion, por lo cual su temperatura es -100 en lugar de 500 .

Planteando la ecuacion de temperatura para cada variable y despejando, obtenemos el siguiente sistema de ecuaciones $Ax = B$:

$$a = -100 \quad (3)$$

$$b = -100 \quad (4)$$

$$c = -100 \quad (5)$$

$$d = -100 \quad (6)$$

$$e = -100 \quad (7)$$

³Para una demostracion de esta propiedad consulte a su docente amigo de Metodos Numericos.

⁶Aclaramos, para los portadores de una imaginación poco delirante, que la circusferencia negra representa una sanguijuela mutante y el cuadrado celeste, al parabrisas.

$$f - \frac{b}{4} - \frac{e}{4} - \frac{g}{4} - \frac{j}{4} = 0 \quad (8)$$

$$g = 500 \quad (9)$$

$$h = -100 \quad (10)$$

$$i = -100 \quad (11)$$

$$j - \frac{f}{4} - \frac{i}{4} - \frac{k}{4} - \frac{n}{4} = 0 \quad (12)$$

$$k - \frac{g}{4} - \frac{j}{4} - \frac{l}{4} - \frac{o}{4} = 0 \quad (13)$$

$$l = -100 \quad (14)$$

$$m = -100 \quad (15)$$

$$n = -100 \quad (16)$$

$$o = -100 \quad (17)$$

$$p = -100 \quad (18)$$

del cual la matriz asociada A es la siguiente:

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
(3)	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
(4)	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
(5)	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
(6)	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
(7)	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
(8)	0	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	1	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	0	0	0	0	0	
(9)	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
(10)	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
(11)	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
(12)	0	0	0	0	0	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	1	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	0	
(13)	0	0	0	0	0	0	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	1	$-\frac{1}{4}$	0	0	$-\frac{1}{4}$	
(14)	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
(15)	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
(16)	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
(17)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
(18)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

siendo la ecuación (3) la resultante de despejar la ecuación de temperatura para la variable a , (4) para la variable b y así sucesivamente. Ordenandolo de esta manera, la matriz asociada A es una matriz banda de ancho y altura n , donde n es la cantidad de columnas de la matriz de temperaturas. Esto lo deducimos observando la figura 2 y la matriz A , ya que notamos que para averiguar el valor de la variable v_i son necesarios⁷ los valores de las variables $v_{i-n}, v_{i-1}, v_{i+1}, v_{i+n}$, donde n hace referencia a la cantidad de columnas. Notese que en las ecuaciones, o lo que es lo mismo, matriz asociada al sistema no se hicieron remplazos previos. Con esto nos referimos a no reemplazar en las ecuaciones de calor automáticamente los valores conocidos, como los del borde o los afectados por sanguijuelas. Un ejemplo de esto se daria al usar las ecuaciones 4, 7 y 9 para reemplazar los valores en la ecuación 8. Tomamos esta decisión porque consideramos que esos reemplazos son parte de la resolución total del sistema, por lo que deberían verse reflejados en la medición de tiempos. Sin embargo, si hicieramos los reemplazos en la generación de la matriz asociada al sistema, no podríamos medirlos con confiabilidad pues tendríamos la lectura de datos como outlier.

Dejaremos el ejemplo hasta aquí, pues la resolución del sistema no aporta información útil. Sin embargo, una vez que hallamos esta estructura procedimos a realizar la carga de datos de modo que la matriz asociada a nuestro sistema fuese una matriz banda- n . Cabe destacar, que mientras dos miembros del grupo buscaban comprender en profundidad las propiedades del

⁷Siempre y cuando v_i no pertenezca a un borde o esté cubierta por una sanguijuela.

modelo, los otros dos implementaron la estructura para la matriz banda⁸, y los algoritmos de Eliminación Gaussiana y Factorización LU. Ahora explicaremos el desarrollo del código.

El lenguaje empleado para la implementación es C++. Si bien la elección del lenguaje podría considerarse arbitraria, se debió principalmente a la cantidad de herramientas y librerías provistas por este lenguaje.

En primer instancia, creamos una clase llamada BDouble para representar decimales de precisión doble para guardar el error estimado acumulado. Esto nos sirvió a la hora de depurar el código. Luego creamos la clase Matrix, que representa a las matrices como un vector de vectores de BDouble. Además guarda la dimensión de la matriz, el ancho y alto de la banda. Para mejorar la complejidad espacial, solo guarda por fila aquellos coeficientes pertenecientes a la banda, o sea, aquellos que no valen necesariamente cero. Esta clase provee las funciones básicas para matriz, como consultar su dimensión o sumar múltiplos de filas entre sí, pero además, contiene las funciones gaussian_elimination, LU_factorization, backward_substitution y forward_substitution.

Si bien nuestra primera idea fue crear una clase aparte para estos métodos, y por claridad abstraernos en esta de la representación interna de la matriz, al hacerlo nos topamos con un problema en los header's del código. Al no poder descubrir el motivo de dicho problema, recurrimos a la catedra en busca de consejo. Aun así, no encontramos el motivo. Es por esto que incluimos la implementación de estos algoritmos en la clase Matrix.

Testeamos estos algoritmos con ejemplos pequeños escritos a mano, para verificar que se realizaban las cuentas correctamente. Notese que en este momento del desarrollo la carga de datos no estaba implementada, por lo que resultaba imposible correr los test provistos por la catedra.

Luego procedimos a crear la interfaz del problema en sí, es decir la entrada y salida de datos, la generación del sistema, etc. Establecimos como criterio usar el promedio de las temperaturas, de todos los puntos que estén incluidos en la circunferencia de radio h con centro en el punto crítico, para la estimación de la temperatura del mismo.

Para finalizar la implementación, codeamos un algoritmo de backtracking simple para calcular cuál sanguijuela eliminar, y un backtracking mejorado con la Fórmula de Sherman-Morrison.

La Fórmula de Sherman-Morrison nos permite conseguir un sistema equivalente al de $(A + B)^{-1}x = b$ si podemos encontrar dos vectores u y v tales que $uv^t = B$.

Sherman-Morrison nos proponen la siguiente Fórmula:

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

Pero sabemos que:

$$A^{-1}b = y \iff Ay = bA^{-1}u = z \iff Az = u$$

Suponiendo que ya resolvimos el sistema A antes de la modificación, tenemos su descomposición, $LU = A$. Por lo que basta resolver:

$$Ly_2 = y \text{ y } Lz_2 = z$$

Luego, $Uy = y_2$ y $Uz = z_2$. Una vez calculados y y z estamos en condiciones de resolver la expresión. Notemos que

$$\frac{v^t A^{-1}b}{1 + v^t A^{-1}u}$$
 es un escalar.

$$\text{Finalmente } x = y + \frac{v^t y}{1 + v^t z} z$$

En cuanto a uv^t recordemos que vamos a usar sherman-morrison para sistemas que cambiar una sola fila i , por lo que nos basta con conseguir que uv^t sea una matriz que modifique apropiadamente dicha fila. Por lo tanto tomamos a u como el i -ésimo vector de la base canónica, y a v como la modificación que necesitamos para transformar la fila i .

⁸Por dato del enunciado, en ese momento sabíamos que debíamos obtener una matriz banda a partir del sistema, si bien no sabíamos cómo organizar los datos para obtenerla.

2.1.1. Pseudocódigo

```

vector, solucion backwardsubstitution( Matriz m, vector b )
fijo i en la \'ultima fila de la matriz m para resolver el sistema de abajo para arriba
mientras
    si en la diagonal hay un 0 entonces
        hay infinitas soluciones
    sino
        empiezo a resolver el sistema de abajo para arriba
        tantas veces como el tama\~no de la banda superior

        cuando llego a la diagonal divido lo que tenia acumulado
        por el elemento de la diagonal

    si i es igual a 0 salgo del ciclo
devuelvo el vector x y el tipo de soluci\'on

```

```

vector, solucion forward_substitution( Matriz m, vector b )
fijo i en la primera fila de la matriz m para resolver el sistema de arriba para abajo
si en la diagonal hay un 0 entonces
    hay infinitas soluciones
sino
    empiezo a resolver el sistema de arriba para abajo
    tantas veces como el tama\~no de la banda inferior

    cuando llego a la diagonal divido lo que tenia acumulado
    por el elemento de la diagonal

devuelvo el vector x y el tipo de soluci\'on

```

```

<vector, soluci\'on> eliminacion_gaussiana( Matriz A, vector b )

diagonal es el m\'inimo entre las columnas de A y las filas de A
recorro tantas veces como dice el n\'umero diagonal

recorro tantas veces como el tamano de la banda inferior,
busco el coeficiente y empiezo a triangular para resolver el sistema

me fijo que el coeficiente no sea 0

el coeficiente es A[i,d]/[d,d]
eso me sirve para que cuando triangulo me quede 0 en donde estoy parado
y asi me queda la matriz triangular superior.

Realizamos la resta a toda la fila.

hago backward_substitution de la Matriz A y con el vector b y devuelvo el resultado

```

```

Matrix L, Matrix U LU_factorization( Matriz A )

creo la matriz L y U que van a tener el mismo tamaño que la matriz A
seteo en 1 las diagonales de las matrices L y U

Me fijo que no haya un 0 en la posicion (0, 0) de la matriz U
porque sino no puedo factorizar

seteo la primer fila de la matriz U y la primer columna de la matriz L

los elementos de la banda de la matriz U van a ser los mismos
que de la banda de la matriz A

Acabo resuelvo el sistema para la fila i de la matriz y lo hago tantas veces
como el mínimo entre el tamaño de la banda inferior y superior.

Voy resolviendo el sistema restando la fila i y acumulando en U(i,i).
Mientras voy guardando los coeficientes en la Matriz L

Divido lo que habrá en la diagonal de la matriz U
por que lo que habrá en la diagonal de la matriz L.

Si el resultado es 0 entonces no podemos factorizar.

Completo las matrices U y L.

seteo la ultima posición de la Matriz U
con el elemento (N-1, N-1) de la matriz A.

devuelvo la matriz L triangular inferior con los resultados
y la matriz U triangulada y triangular superior.

```

2.2. Sobre la experimentación

Para medir los tiempos de resolución, utilizamos la función `gettimeofday`, que permite medir con facilidad el lapso de tiempo entre dos momentos de la ejecución del programa. Decidimos medir únicamente los tiempos que estuvieran estrictamente relacionados a la resolución del sistema, dejando afuera, por ejemplo, el tiempo empleado para escribir las soluciones en archivos de salida. La idea de fondo que nos llevó a medir de esta manera fue la de intentar que las magnitudes de los tiempos empleados por los métodos de triangulación de Gauss y LU no se vieran afectadas por tiempos mucho mayores que distorsionaran la distancia relativa entre ellas.

Una vez finalizada la etapa de implementación y testeo de los algoritmos, procedimos a realizar una serie de experimentos que nos permitieran evaluar el comportamiento de los mismos. Además de analizar cómo se modifican los resultados a medida que se modifica el parámetro de discretización h , definimos algoritmos para comparar el desempeño, en términos de la estimación de temperaturas. Por comodidad, a la hora de diseñar los experimentos, definimos todos los parámetros con forma cuadrada.

Siguiendo la consigna del TP, planteamos cuatro casos diferentes para analizar como varía la temperatura en el punto crítico al variar la discretización. Estos casos no fueron generados al azar, sino que para optimizar el tiempo de computo, creamos a mano cada instancia para realizar por caso un experimento fijo. Con esto queremos decir que, además de analizar la variación de la temperatura en función de la granularidad, planteamos para cada experimento una hipótesis aparte, que corroboraremos o refutaremos luego de analizar los resultados.

3. Experimentación y discusión

Presentaremos en esta sección la experimentación y discusión juntas, separadas por experimentos. Primero daremos una breve descripción de la instancia experimental, seguido de lo

que esperamos observar, es decir, nuestra hipótesis. A continuación mostraremos los resultados obtenidos y nuestro análisis respecto a porque obtuvimos dichos resultados. Decidimos darle este formato porque creímos mas declarativo mantener toda la información junta.

3.1. Experimento 1: Medición de performance respecto a la variación de la discretización

3.1.1. Caso A: ¿Ser o no ser?

Este experimento presenta un caso genérico de parabrisas, donde por genérico nos referimos a que seleccionamos la ubicación, radio y temperatura de manera aleatoria entre un grupo de valores predeterminados. Se corre para distintos tamaño de discretización. La idea de este experimento es responder la siguiente pregunta: ¿Terminar hoy o modelar bien? Con esto nos referimos a si es preferible sacrificar tiempo de computo en pos de modelar mejor, o si por el contrario es mejor alejarse un poco del modelo real, pero obteniendo resultados en un tiempo razonable. Por supuesto, es obvio que esta decisión depende en mayor medida del uso que se le quiere dar a los resultados y no es nuestra intención conseguir una respuesta genérica a esta pregunta, sino analizar la relación entre estos conceptos logrando así una mejor comprensión del problema.

Nuestra hipótesis es que para h muy chicos, menores al 5 % de la longitud original, el tiempo de computo aumentara demaciado, sin obtener ninguna mejora significativa en la modelización. Claro que para esta hipótesis estamos asumiendo no hallarnos en un caso extremo como, por ejemplo, tener gran cantidad de sanguijuelas microscópicas con temperaturas similares a la del sol o plutón.

3.1.2. Caso B: Una sanguijuela invoca otra sanguijuela

En este caso, a diferencia de los demás donde variamos solo el h , fijamos la discretización y el tamaño. La variable en este experimento es la cantidad de sanguijuelas. Planteamos este experimento porque nos parecio interesante ver como se modifica la performance de la eliminación Gaussiana y la factorización LU, a medida que fijamos mayor cantidad de constantes. Esto es porque suponemos que, al tener menos coeficientes la matriz, backward y forward deberian ejecutar mas rápido, lo que implica que la ejecucion con mayor cantidad de sanguijuelas debería ser mas la mas rápida. Esa es nuestra hipótesis en este experimento.

3.2. Experimento 2: Comparación de temperaturas en el punto crítico

3.2.1. Caso A: ¡Mientras varie la granularidad, sirve!

Para realizar este experimento, el de comparación de puntos críticos con tres instancias, usaremos como instancia todos los demás experimentos que varien la granularidad. Esto hace honor a la famosa frase “el tiempo es oro” y como no disponemos mucho de ninguno, reutilizaremos las corridas de otros experimentos. En general, esperamos que al achicar el h la temperatura del punto crítico varie, pero no esperamos que aumente o disminuya siempre. Al aumentar la granularidad por un lado vamos a incluir en el modelo nuevas sanguijuelas, lo que potencialmente haria que la temperatura del punto crítico, y de los demás puntos, aumente. Sin embargo, cuando agregamos puntos tambien estamos aumentando la “dissipación” del calor, con lo que nos referimos a que los sucesivos cálculos de promedio bajan la temperatura a medida que te alejas de una sanguijuela. Es por eso que esperamos que las variaciones sean tanto positivas como negativas. Y remarcamos que una hipótesis concreta de la variación de la temperatura, debe provenir del análisis de una instancia concreta, por lo que no se puede crear una hipótesis genérica.

3.2.2. Caso B: Buscando las sanguijuelas perdidas

Este experimento fue diseñado para modelar un caso que se comporte “mal” para un h grande. Es decir, modelamos este experimento para que las sanguijuelas pequeñas, que son aquellas de mayor temperatura, estén distribuidas de modo que no afecten el modelo para discretizaciones grandes. Realizamos este experimento como un caso particular de los decriptos en el A, para mostrar una hipótesis concreta. En este caso, como al aumentar el h van a surgir

nuevas sanguijuelas, lo que esperamos es que la temperatura del punto critico, y en general, aumente. Esto es porque creemos que la aparicion de nuevas sanguijuelas es mas significativo que el aumento de disipacion, cuando este aumento es relativamente pequeño, es decir, del orden del 50-25 %.

3.3. Experimento 3: Comparación de performances para backtraking al aplicar Sherman-Morrison

3.3.1. Caso A: Sobre población de mini-sanguijuelas

Este experimento es un caso extremo para probar el metodo que usa la formula de Sherman-Morrison. Todas las sanguijuelas si afectan un punto es exclusivamente uno, en todas las discretizaciones planteadas. La hipotesis es que el metodo 4, que aplica la Formula de Sherman-Morrison tiene mejor performance temporal.

Como deceamos observar la diferencia de performance entre el 3 y 4 metodo para los casos de sanguijuelas pequeñas, decidimos no introducir sanguijuelas que afecten mas de un punto. Esto es porque las tecnicas de backtraking son, en general, de muy mala performance temporal y disponemos de un tiempo acotado para finalizar la experimentación. Por esto, nos restringimos a los casos particulares que nos competen, dejando para como proyecto a futuro la experimentación en casos generados de manera completamente aleatoria. Esto incluye los experimentos previos.

4. Conclusiones generales

5. Bibliografia

R. Burden y J.D.Faires. Análisis numérico, International Thomson Editors, 1998.

Heath, M. T. Scientific computing: an introductory survey, pág. 53. Boston, McGraw-Hill, 2002.

6. Apendice A: Enunciado

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 sangujuela 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°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°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 sangujuela 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:

$$\frac{\partial^2 T(x, y)}{\partial x^2} + \frac{\partial^2 T(x, y)}{\partial y^2} = 0. \quad (19)$$

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

$$T(x, y) = -100^\circ\text{C} \quad \text{si } x = 0, a \text{ ó } y = 0, b. \quad (20)$$

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 $i = 0, 1, \dots, n$ y $j = 0, 1, \dots, m$, 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:

$$t_{ij} = \frac{t_{i-1,j} + t_{i+1,j} + t_{i,j-1} + t_{i,j+1}}{4}. \quad (21)$$

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. *Explorando 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]:

$$(A + uv^t)^{-1} = A^{-1} - \frac{A^{-1}uv^tA^{-1}}{1 + v^tA^{-1}u}. \quad (22)$$

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 $i = 1, \dots, k$, 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.

7. Apendice B: Código fuente

7.1. BDouble.h

```
#ifndef _TP1_BDOUBLE_H_
#define _TP1_BDOUBLE_H_

#include <cmath>
#include <limits>
#include <utility>

class BDouble {
public:
    BDouble() : x(0.0) { }
    BDouble(double x) : x(x) { }
    BDouble(float x): x(x) { }
    BDouble(const BDouble &d) : x(d.x) { }

    double get() const { return this->x; }

    BDouble &operator=(const BDouble &rhs) {
        this->x = rhs.x;
        return *this;
    }

    BDouble &operator+=(const BDouble &rhs) {
        this->x += rhs.x;
        return *this;
    }

    BDouble &operator-=(const BDouble &rhs) {
        this->x -= rhs.x;
        return *this;
    }

    BDouble &operator*=(const BDouble &rhs) {
        this->x *= rhs.x;
        return *this;
    }

    BDouble &operator/=(const BDouble &rhs) {
        this->x /= rhs.x;
        return *this;
    }

    BDouble &operator=(const double &rhs) {
        this->x = rhs;
        return *this;
    }

    BDouble &operator+=(const double &rhs) {
        this->x += rhs;
        return *this;
    }

    BDouble &operator-=(const double &rhs) {
        this->x -= rhs;
        return *this;
    }
}
```

```
BDouble &operator*=(const double &rhs) {
    this->x *= rhs;
    return *this;
}

BDouble &operator/=(const double &rhs) {
    this->x /= rhs;
    return *this;
}

BDouble &operator++() {
    this->x++;
    return *this;
}

BDouble &operator--() {
    this->x--;
    return *this;
}

bool operator==(const BDouble &rhs) const {
    return std::fabs(this->x - rhs.x) < std::numeric_limits<double>::epsilon();
}

bool operator<(const BDouble &rhs) const {
    return rhs.x - this->x < std::numeric_limits<double>::epsilon();
}

bool operator!=(const BDouble &m) const {
    return !(*this == m);
}

bool operator==(const double &rhs) const {
    return std::fabs(this->x - rhs) < std::numeric_limits<double>::epsilon();
}

bool operator<(const double &rhs) const {
    return rhs - this->x < std::numeric_limits<double>::epsilon();
}

bool operator!=(const double &m) const {
    return !(*this == m);
}

private:
    double x;
};

BDouble operator+(const BDouble &lhs, const BDouble &rhs) {
    BDouble output(lhs);
    output += rhs;
    return output;
}

BDouble operator-(const BDouble &lhs, const BDouble &rhs) {
    BDouble output(lhs);
    output -= rhs;
    return output;
}
```

```
BDouble operator*(const BDouble &lhs, const BDouble &rhs) {
    BDouble output(lhs);
    output *= rhs;
    return output;
}

BDouble operator/(const BDouble &lhs, const BDouble &rhs) {
    BDouble output(lhs);
    output /= rhs;
    return output;
}

BDouble operator+(const BDouble &lhs, const double &rhs) {
    return lhs + BDouble(rhs);
}

BDouble operator-(const BDouble &lhs, const double &rhs) {
    return lhs - BDouble(rhs);
}

BDouble operator*(const BDouble &lhs, const double &rhs) {
    return lhs * BDouble(rhs);
}

BDouble operator/(const BDouble &lhs, const double &rhs) {
    return lhs / BDouble(rhs);
}

std::ostream &operator<<(std::ostream &os, const BDouble &m) {
    os << m.get();
    return os;
}

const BDouble zero = BDouble(0.0);

#endif // _TP1_BOXEDDOUBLE_H_
```

7.2. Matrix.h

No lo voy a agregar hasta el ultimo jodido minutos. (?)