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

Laboratorio de Métodos Numéricos
Primer Cuatrimestre 2014

Trabajo Práctico Número 1: Con 15 thetas *discretizo alto horno...*

Delgadillo Cárdenas Abel - LU: 74/12 - adelgadillo91@gmail.com
Gálvez Ramiro - admisión doctorado - ramirogalvez@gmal.com
Salinas Pablo Manuel - LU: 456/10 - salinas.pablom@gmail.com

Resumen

En el presente informe se presenta la implementación, desarrollo y análisis de resultados obtenidos al haber resuelto un problema práctico con las herramientas aprendidas en la materia. El problema planteado consitió en encontrar la isoterma 500^0C de un alto horno. Además de ahondar en cuestiones relacionadas al desarrollo, presentamos los resultados de experimentos que nos permitieron comprender de mejor manera cómo funciona el método de eliminación gaussiana y de de factorización LU. Sumado a los experimentos obligatorios planteados en la consigna, que nos permitieron entre otras cosas comprobar la superioridad de la factorización LU cuando se tienen que resolver más de una vez el sistema de ecuaciones, hicimos hincapié en estudiar cómo distintos métodos posibles para localizar la isoterma 500^oC dado un sistema ya resuelto pueden modificar los resultados obtenidos.

Palabras clave
Isoterma, Eliminación Gaussiana, Factorización LU

1. Introducción Teórica

En el presente informe se presenta la implementación, desarrollo y análisis de resultados obtenidos al haber resuelto un problema práctico con las herramientas aprendidas en la materia. En concreto, el objetivo planteado en las consignas del trabajo práctico implicó tener que implementar métodos de resolución de sistemas de ecuaciones lineales a un problema particular: estimar la ubicación de la isoterma de 500^oC de un alto horno.[1]

Siguiendo las indicaciones presentadas en el enunciado del trabajo práctico (presentado en el apéndice A) se consideró un corte horizontal del horno, tal como se puede observar en la siguiente imagen:

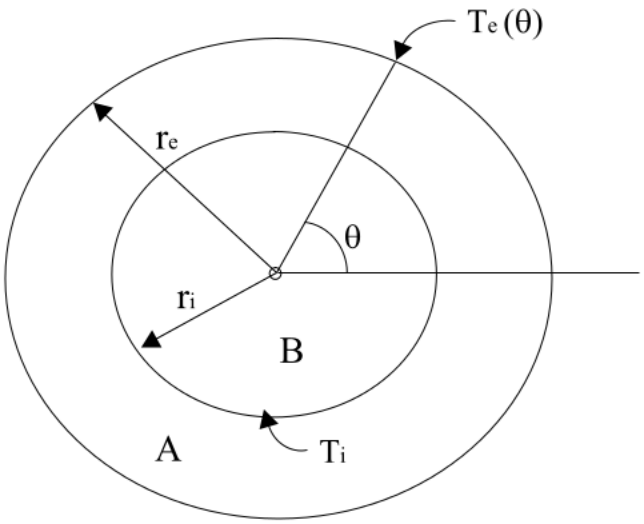

Figura 1: Sección circular del horno

Donde:

- Sector A: representa la pared del horno. Tanto el borde interno como externo de la misma son circulares. A su vez, mediante sensores ubicados en la parte externa del horno, se puede medir la temperatura de esta pared, la cual oscila habitualmente entre 50^oC y 200^oC.
- Sector B: representa el horno propiamente dicho, donde se funde el acero a altas temperaturas. Suponemos que la temperatura dentro del sector B es constante e igual a 1500^oC.
- r_i: representa el radio del sector B.
- r_e: representa el radio de la pared externa.
- $T_e(\theta) : [0, 2\pi] \to \mathbb{R}$: representa, para cada ángulo θ, la temperatura en el punto (r_e, θ).
- T_i: representa la temperatura en el interior del horno.
- $T(r, \theta)$: representa la temperatura en el punto dado por las coordenadas polares (r, θ), siendo r el radio y θ el ángulo polar de dicho punto.

[1] A modo de referencia, mayores detalles sobre la estructura y funcionamientos de los altos hornos pueden encontrarse en `http://es.wikipedia.org/wiki/Alto_horno`

A su vez, en base a información brindada por expertos, se sabe que en estado estacionario la temperatura del horno para un radio y ángulo determinado satisface la siguiente relación:

$$\frac{\partial^2 T(r,\theta)}{\partial r^2} + \frac{1}{r}\frac{\partial T(r,\theta)}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T(r,\theta)}{\partial \theta^2} = 0 \tag{1}$$

También sabemos que:

$$T(r,\theta) = T_i \quad \textit{para todo punto } (r,\theta) \textit{ con } r \leq r_i \tag{2}$$

$$T(r_e,\theta) = T_e(\theta) \quad \textit{para todo punto } (r_e,\theta) \tag{3}$$

El problema en derivadas parciales dado por la ecuación (1) con las condiciones de contorno (2) y (3), permite encontrar la función T de temperatura en el interior del horno (sector A), en función de los valores de temperatura en la pared interna y externa del horno.

A los fines de poder resolver computacionalmente este problema, se discretizó el dominio del problema (el sector A) en coordenadas polares. Consideando una partición $0 = \theta_1 < \theta_1 < ... < \theta_n = 2\pi$ de n ángulos discretos con $\theta_k - \theta_{k-1} = \Delta\theta$ para $k = 2, ..., n$, y una partición $r_i = r_0 < r_1 < ... < r_m = r_e$ de $m+1$ radios discretos con $r_j - r_{j-1} = \Delta r$ para $j = 1, ..., m$.

De este modo se quiere determinar el valor de la función T en los puntos de la discretización (r_j, θ_k) que se encuentren dentro del sector A. Denominamos $t_{j,k} = T(r_j, \theta_k)$ al valor (desconocido) de la función T en el punto (r_j, θ_k).

Para encontrar estos valores, se transforma la ecuación (1) en un conjunto de ecuaciones lineales sobre las incógnitas $t_{j,k}$, evaluando (1) en todos los puntos de la discretización que se encuentran dentro del sector A. Al hacer esta evaluación, aproximamos las derivadas parciales de T en (1) por medio de las siguientes fórmulas de diferencias finitas:

$$\frac{\partial^2 T(r,\theta)}{\partial r^2}(r_j,\theta_k) \cong \frac{t_{j-1,k} - 2t_{jk} + t_{j+1,k}}{(\Delta r)^2} \tag{4}$$

$$\frac{\partial T(r,\theta)}{\partial r}(r_j,\theta_k) \cong \frac{t_{j,k} - t_{j-1,k}}{\Delta r} \tag{5}$$

$$\frac{\partial^2 T(r,\theta)}{\partial \theta^2}(r_j,\theta_k) \cong \frac{t_{j,k-1} - 2t_{jk} + t_{j,k+1}}{(\Delta \theta)^2} \tag{6}$$

Reemplazando en la ecuación (1) se obtiene la siguiente igualdad aproximada:

$$\frac{t_{j-1,k} - 2t_{jk} + t_{j+1,k}}{(\Delta r)^2} + \frac{t_{j,k} - t_{j-1,k}}{\Delta r} + \frac{t_{j,k-1} - 2t_{jk} + t_{j,k+1}}{(\Delta \theta)^2} \cong 0 \tag{7}$$

Es importante destacar que, para $j = 0$ y $j = m+1$, los valores de las incógnitas $t_{j,k}$ son conocidos $\forall k = 1, ..., n$, ya que son los que se encuentran en las paredes interna y externa del horno. Al realizar este procedimiento, obtuvimos un sistema de ecuaciones lineales que modela el problema discretizado. El mismo contiene $n*(m+1)$ ecuaciones y $n*(m+1)$ incógnitas. Dichas incógnitas son aproximaciones de los valores de la función T en los puntos de la discretización.

Puesto que, dado este esquema de discretización, nos encontramos en presencia de un sistema de ecuaciones lineales, resolvimos el mismo aplicando dos de los métodos de resolución de sistemas de ecuaciones lineales vistos en clases: el algoritmo de Eliminación Gaussiana (sin pivoteo) (al que abreviamos en este informe "EG") y la factorización LU (al que abreviamos en este informe "LU").

En lo referido a los detalles de los métodos se tiene que el método EG consta de dos pasos: en primer lugar, lleva el sistema de ecuaciones original a un sistema equivalente donde la matriz es triangular superior (con costo $O(n^3)$ en donde n es el número de incógnitas) y luego procede a resolver dicho sistema (con costo $O(n^2)$). En cambio, 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 (esta descomposición tiene un costo $O(n^3)$), y luego resuelve los sistemas $Ly = b$ y $Ux = y$ (ambos con costo $O(n^2)$) para obtener el valor de solución x. En términos asintóticos ambos métodos tienen una complejidad $O(n^3)$; sin embargo LU presenta una ventaja por sobre EG, y es que si el sistema se debe resolver para un b diferente, no hace falta realizar nuevamente la factorización, y alcanza simplemente con resolver los dos sitemas triangulares usando las mismas matrices L y U (siendo el costo de resolver este sistema adicional $O(n^2)$). Para más detalles teóricos sobre las características de ambos métodos puede verse Burden *et. al.*

Igualmente, resulta importante señalar aquí que, como la matriz asociada al problema es diagonal dominante (no estricta) y sus columnas son linealmente independientes[2], el sistema de ecuaciones derivado de ella admite una descomposición LU o, lo que es equivalente, que se pueden hacer todas las iteraciones del método de triangulación de Gauss sin necesidad de pivotear en ningún paso. La demostración se describe en el Apéndice C. Este resultado fue muy importante a la hora de realizar el desarrollo, pues nos permitió evitar tener que implementar y testear técnicas de pivoteo de filas y columnas.

2. Desarrollo

En esta sección presentamos los detalles referidos al desarrollo del trabajo práctico y a las decisiones que tomamos para llevarlo adelante.

Como requisito para resolver el trabajo práctico se pidió que los algoritmos que resuelven los sistemas de ecuaciones estuvieran escritos en C o C++. Optamos por escribirlos en C++, principalmente por estar más familirizados con él.

Antes de implementar los algoritmos, fue necesario modelar la matriz y vector de solución asociados al problema. Como ya teníamos una estructura determinada como input al problema, y como los sistemas se iban a resolver en C++, optamos también por modelar la matriz del problema y el vector de solución en C++. Dado que los tests dados por la cátedra eran relativamente grandes, en esta etapa armamos de manera manual archivos de entrada que presentasen el problema con menores valores de n y m_mas_1, de modo que pudiesemos inspeccionar imprimiendo desde C++ a pantalla la matriz y vector, y corroborar que los cálculos se estuviesen realizando bien. Una decisión que tomamos fue la de representar la matriz A del problema con una estructura de "*vector*" de "*vector*".

Una vez modelado el problema, implementamos las funciones de resolución de sistemas de ecuaciones lineales usando EG y LU, léase: una función que resuelve un sistema triangular inferior, una función que resuelve un sistema triangular superior, una función que realiza la eliminación gaussiana y otra función que realiza la factorización LU. La implementación de estas funciones fue relativamente simple, y la fuimos testeando en todo momento con sistemas pequeños para los cuales habíamos obtenido las soluciones en papel.

Una vez hecho esto, modificamos la función "main()" del programa para que leyera y guardara tanto los parámetros del programa (más detalles abajo) como los valores que contienen los archivos de entrada. Realizado esto pudimos efectuar y pasar los tests.

El código que modela y resuelve el sistema de ecuaciones (*main.cpp*), se presenta en el Apéndice B de este informe. Sin embargo consideramos oportuno detallar aquí qué *inputs* toma el programa (listados en el orden en que los recibe por linea de comandos, separados entre ellos por un espacio):

[2]Probar estas dos características para la matriz asociada el problema en cuestión escapa a los objetivos de este informe, por lo cual fuimos aconsejados de tomarlo como un resultados ciertos sobre los cuales podemos trabajar.

- Inputs Obligatorios:
  - Un string que indique la ubicación del archivo de entrada (por ejemplo: "*./infile.in*").
  - Un string que indique la ubicación del archivo de salida (por ejemplo: "*./outFile.out*").
  - Un entero que indique qué método se utilizará para resolver el sistema de ecuaciones (0 para EG y 1 para LU).
- Inputs Opcionales
  - Un string que indique la ubicación del archivo en donde se guardarán los detalles de la ubicación de la isoterma 500^0C (por ejemplo: "*./isoOut.out*").
  - Un entero que indique qué método se utilizará para calcular la isoterma (0 para el método del punto más cercano y 1 para interpolación lineal).

El programa produce las siguientes salidas:

- El archivo de salida (indicado como segundo parámetro) con la solución del sistema, es decir, la temperatura para cada punto de la discretización.
- El tiempo de ejecución (en segundos) en un archivo llamado *tiempoEjecucion.out* (más detalles abajo).
- La ubicación de la isoterma de 500°C en el archivo pasado como parámetro (en caso de haberle pasado este parámetro opcional. Si se ejecuta sin pasar este parametro, no generará esta salida)

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 $Ax = b$, 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 era 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 (como el ya mencionado ejemplo de la escritura de archivos) que distorsionaran la distancia relativa entre ellas.

Además de los experimentos que se piden realizar en la consigna de este trabajo práctico (la cual se presenta en el apéndice A), nos planteamos como objetivo adicional analizar cómo varían los resultados obtenidos si utilizamos distintas técnicas para calcular la ubicación de la isoterma 500°C. Por ello, implementamos dos métodos distintos para calcularla. El primero de ellos, que denominamos Método del Punto más Cercano, consiste en buscar, para cada ángulo θ_k de la discretización, el radio r_j para el cual el valor absoluto de la diferencia entre 500°C y $t_{r,j}$ es mínimo. El segundo, que denominamos Método de Interpolación Lineal, ajusta la ubicación de la isoterma usando una función continua (un polinomio de grado 1) de la siguiente manera: traza la recta entre los dos puntos más cercanos a 500°C y despeja el punto de dicha recta cuya imagen pasa por 500. Más adelante se comparará la precisión de los dos métodos.

Dado que, como se detallará más adelante, los experimentos requirieron realizar muchas corridas del programa con archivos de entrada diferentes (por ejemplo, variando los parámetros de discretización), consideramos necesario armar un programa que automatizara la elaboración de estos archivos. Por ello implementamos en *python* un programa llamado *armainfile.py* (el cual podemos hacer llegar a pedido) que arma archivos del tipo "*./infile.in*" en base a argumentos pasados por línea de comando (eventualmente este programa se llamó en los scripts de matlab que corrieron los experimentos). Los parámetros que recibe son: r_i, r_e, número de discretizaciones en radio (m_mas_1), número de discretizaciones en ángulo (n), valor de isotrema a estimar T_i, número de veces que se debe calcular la resolución de sistemas de ecuaciones (n_inst), un string que indique en dónde se guardará el archivo de salida (por ejemplo: "*./infile.in*") y para cada n_inst el valor de la temperatura interna del horno y el maximo y mínimo valor de temperatura de la pared externa del horno (por ejemplo si n_inst fuese 2 podrían pasarse los valores 1500 200 50 1500 50 50). La temperatura máxima y mínima de la pared

externa del horno son utilizadas por el programa para que en el archivo de texto resultante se escriban las temperaturas correspondientes al exterior del horno (para cada ángulo θ_j de la discretización) siguiendo la siguiente fórmula (definida para $0 \leq \theta_j \leq 2 * \pi$):

- Si $0 \leq \theta_j \leq \pi$ entonces $T(\theta_j) = 200 + (50 - 200)/\pi * \theta_j$
- Si $\pi < \theta_j < 2 * \pi$ entonces $T(\theta_j) = 50 + (200 - 50)/\pi * \theta_j$

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 modifican los parámetros de discretización (n y m_mas_1), nos enfocamos también en comparar el desempeño, en términos de estimación de la ubicación de la isoterma, usando el método del punto más cercano y de interpolación lineal. Una consideración general para todos los experimentos: consideramos $n = m\_mas\_1$.

Siguiendo las consignas del trabajo práctico, al momento de analizar la ubicación de la isoterma de 500^oC, trabajamos con dos estados diferentes del horno. En ambos casos tomamos $r_i = 10$, $r_e = 100$ y la temperatura interna igual a 1500^oC, pero difieren en que en el primer estado (al que denominamos "caso 1") asignamos para toda la pared exterior del horno una temperatura constante de 50^oC, mientras que en el segundo estado (al que denominamos "caso 2") consideramos que la temperatura del horno varía para cada ángulo θ_j de la discretización como se indica en la ecuación (8).[3]

Finalmente, corrimos los experimentos utilizando MatLab. Elegimos este programa principalmente porque permite obtener de una manera simple los gráficos que queríamos mostrar. A su vez, nuestras decisiones de implementación de "*main.cpp*" y "*armaInFile.py*" nos permitieron automatizar en gran medida el proceso de experimentación, por lo cual no nos demandó excesivo tiempo implementar esta etapa del desarrollo (sí lo demandó el tener que pensar qué experimentos realizar y cómo mostrar los resultados). Los experimentos de MatLab tienen como salida los gráficos que presentamos en la siguiente sección. Vale la pena mencionar que para la figura 2, la figura 4 y para las animaciones aprovechamos el script "*horno.m*" que nos entregó la cátedra (modificándolo levemente).

3. Resultados

Como se mencionó en la sección anterior, realizamos los experimentos de estimación de la ubicación de la isoterma de 500^oC para dos estados diferentes del horno. A continuación se presentan los resultados de los experimentos que llevamos adelante. En primer lugar presentamos los resultados referidos al análisis de cómo afectan la ubicación de la isoterma de 500^oC los diferentes estados del horno, los parámetros de discretización y el método de aproximación de su ubicación. Luego presentamos resultados referidos al tiempo de ejecución de los programas cuando n_inst es igual a 1. Finalmente presentamos resultados en donde estudiamos cómo se diferencian EG y LU a medida que varían los valores de n_inst y las condiciones de la pared externa del horno. Resulta importante destacar que no ahondaremos en interpretaciones de los resultados en esta sección y dejaremos esta tarea para la sección de discusión.

[3] Como ya se mencionó, este formato de archivo de entrada se pueden crear facilmanete en base al script "armaInFile.py".

3.1. Resultados Referidos a la Ubicación de la Isoterma 500^oC

3.1.1. Resultados Obtenidos para el Caso 1

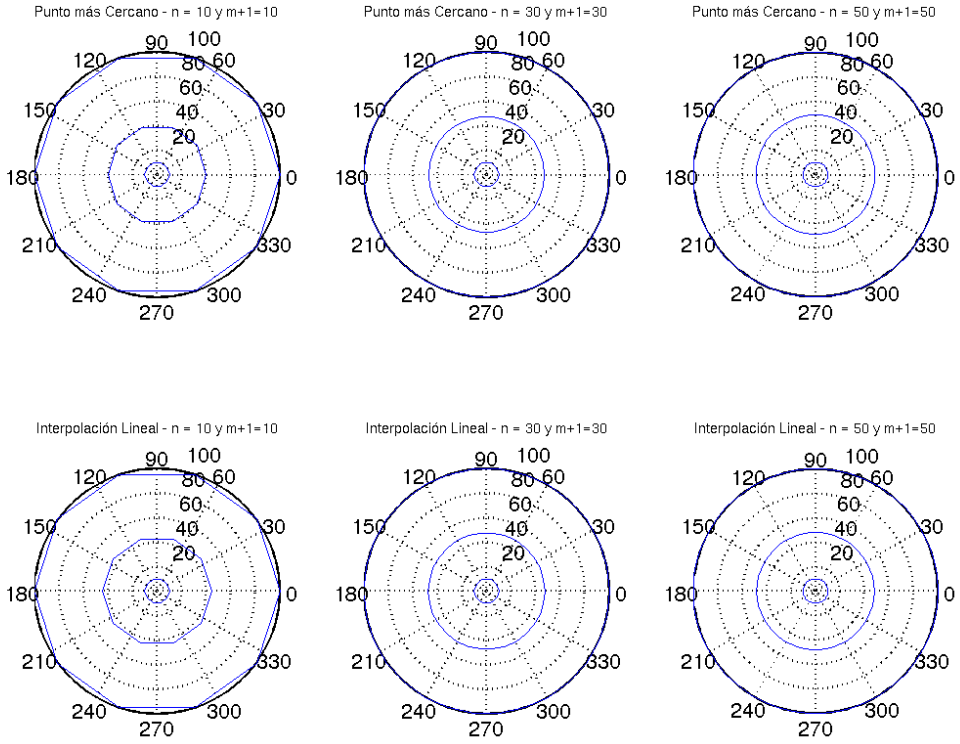

Figura 2: Ubicación de la Isoterma para el Caso 1

Como se puede apreciar en la Figura 2, para el caso 1 la isoterma tiene una forma circular. A su vez, tanto para el método del punto más cercano como para el método de interpolación lineal, pareciese que la isoterma de 500^0C se acerca a la pared exterior a medida que aumenta el valor de $n * m\_mas\_1$. Puesto que tuvimos interés en analizar con mayor detalle este comportamiento y que no resulta práctico repetir la misma figura un número excesivo de veces, optamos por plantear una medida que nos permitiese en una misma figura analizar con mayor detalle este comportamiento. La medida que utilizamos es, dada una isoterma calculada, cuál es la distancia entre la pared externa y el punto más cercano de la misma a la pared. En la figura 3 se presentan los resultados obtenidos de esta medida para $(n * m\_mas\_1)^i$ para i que va de 10 a 50 en pasos unitarios tanto para el método del punto más cercano como para el método de interpolación lineal.[4]

[4]Aun cuando la figura 2 contiene resultados interesantes, nos pareció oportuno corroborar nuestras intuiciones y analizar los resultados con mayor detalle en base a animaciones. Una animación que permite analizar cómo se modificar la ubicación de la isoterma 500^0C a medida que se discretiza con mayor detalle se puede bajar de `https://www.dropbox.com/s/vdmnzhmfy0vdh2y/IsotermasCaso01.wmv`

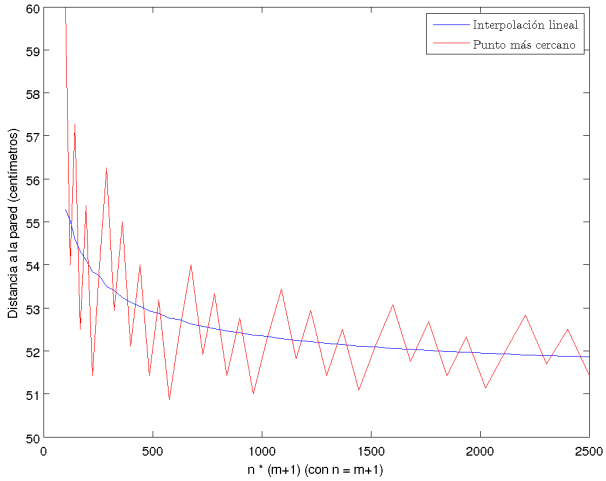

Figura 3: Curvas de Distancia Mínima a la Pared de las Isotermas Calculadas para el Caso 1

3.1.2. Resultados Obtenidos para el Caso 2

Como puede verse en la figura 4, para el caso 2 la isoterma ya no tiene una forma circular. A su vez, sigue pareciendo que tanto para el método del punto más cercano como para el método de interpolación lineal, la isoterma de 500^0C se acerca a la pared exterior a medida que aumenta el valor de $n*m\_mas\_1$, sin embargo en el caso del método del punto más cercano tiene un comportamiento aun más errático. En la figura 5 se presentan los resultados obtenidos de esta medida para $(n * m\_mas\_1)^i$, para i que va de 10 a 50 en pasos unitarios tanto para el método del punto más cercano como para el método de interpolación lineal.[5]

[5]Una animación que permite analizar cómo se modificar la ubicación de la isoterma 500^0C a medida que se discretiza con mayor detalle se puede bajar de `https://www.dropbox.com/s/u59779ewwltrwa5/IsotermasCaso02.wmv`

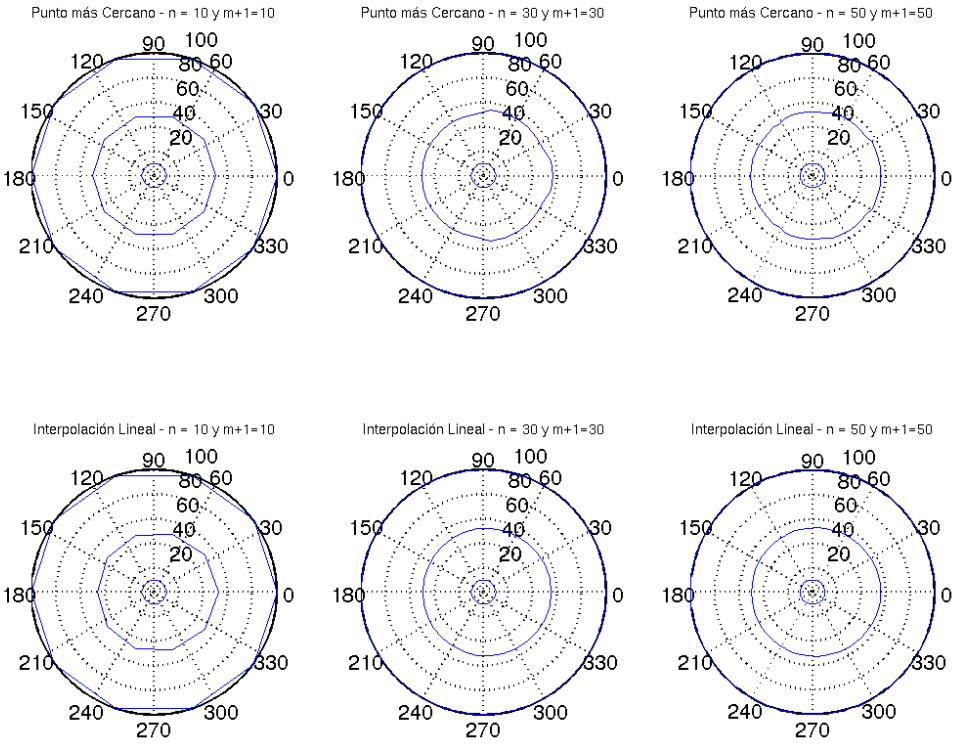

Figura 4: Ubicación de la Isoterma para el Caso 3

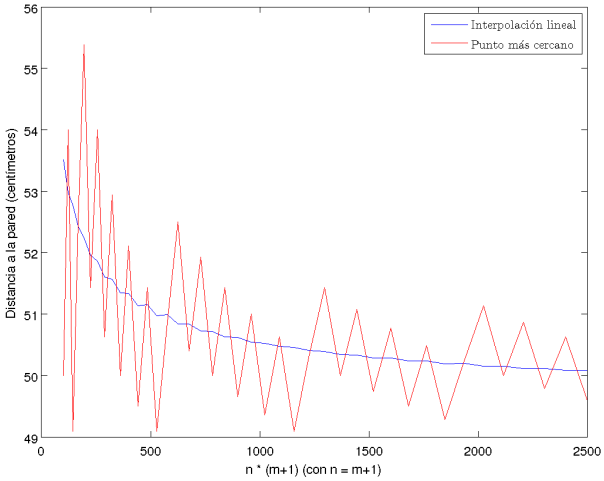

Figura 5: Curvas de Distancia Mínima a la Pared de las Isotermas Calculadas para el Caso 1

3.2. Resultados Referidos al Tiempo de Corrida para $n\_inst = 1$

Para analizar el tiempo de corrida de los algoritmos en función de los parámetros de discretización, diseñamos un experimento en el cual medimos cuánto demora en resolverse el sistema de ecuaciones para valores de n y m_mas_1 que van de 5 a 50 en pasos de 5. A su vez, como el tiempo de resolución de los sistemas es variable y depende de factores externos al programa, optamos por tomar como medida de duración para cada parametrización diferente, el promedio de 5 corridas. En la figura 6 se presentan los resultados obtenidos. [6]

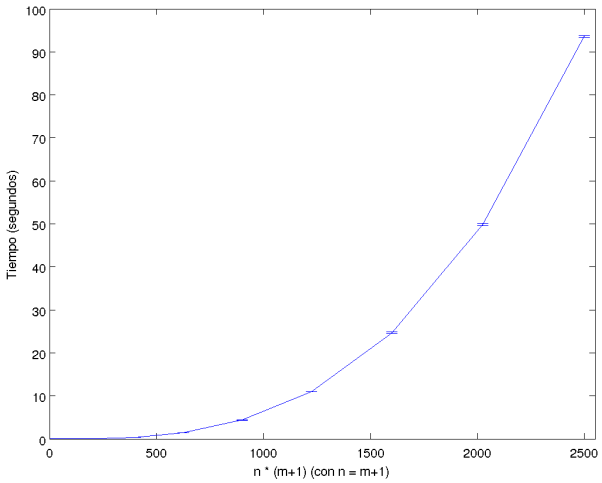

Figura 6: Tiempo de Corrida del Programa Para Distintos Valores de los Parámetros de Discretización

Resta decir que los resultados presentados se corresponden únicamente a corridas que resuelven el sistema haciendo uso de eliminación Gaussiana, pues, como se verá mejor en el siguiente experimento, cuando n_inst es igual a 1, la diferencia entre ambos métodos a medida que aumenta el número de ecuaciones deja de tener relevancia. [7]

[6] Como ya se mencionó, el tiempo que se mide es únicamente el tiempo de resolución de los sistemas

[7] Elaboramos una figura similar a la figura 6 en donde se presentaban también los resultados para el método LU, sin embargo observamos que las curvas se superponían de manera casi perfecta y decidimos no incorporarlo al informe.

3.3. Resultados Referidos al Tiempos de Corrida de Para $n\_inst >= 1$

Finalmente, nos enfocamos en analizar el tiempo que demora en resolverse el problema cuando $n\_inst >= 1$. En este caso diseñamos el experimento de manera tal que pudiéramos analizar cómo impacta la variación de n_inst sobre el tiempo de corrida y, a su vez, para ver si los parámetros de discretización impactan sobre los resultados obtenidos. La siguiente figura muestra los resultados obtenidos de nuestras corridas. Para poder mostrar de manera más clara los resultados obtenidos, la medida que mostramos es la diferencia entre el tiempo que le tomo resolver el sistema al método LU y el tiempo que le tomó resolver el mismo sistema al método de EG. Al igual que en el experimento anterior, los resultados que se presentan son el promedio de 5 corridas. En el panel izquierdo se puede ver cuánto demoran en resolverse los sistemas para valores de n_inst que van de 1 a 6 y para valores de n y m_mas_1 que van de 3 a 30 en pasos de 3, mientras que en el panel derecho presentamos para n y m_mas_1 igual a 30 cómo varía esta diferencia en función de n_inst. A su vez, para los casos en que n_inst es mayor a a 1, hacemos que a medida que aumenta el valor de n_inst aumente también la temperatura máxima de la pared externa del horno (siendo igual a 50^{o}C para el primer valor de n_inst e igual a 200^{o}C para el mayor valor que se esté probando).[8]

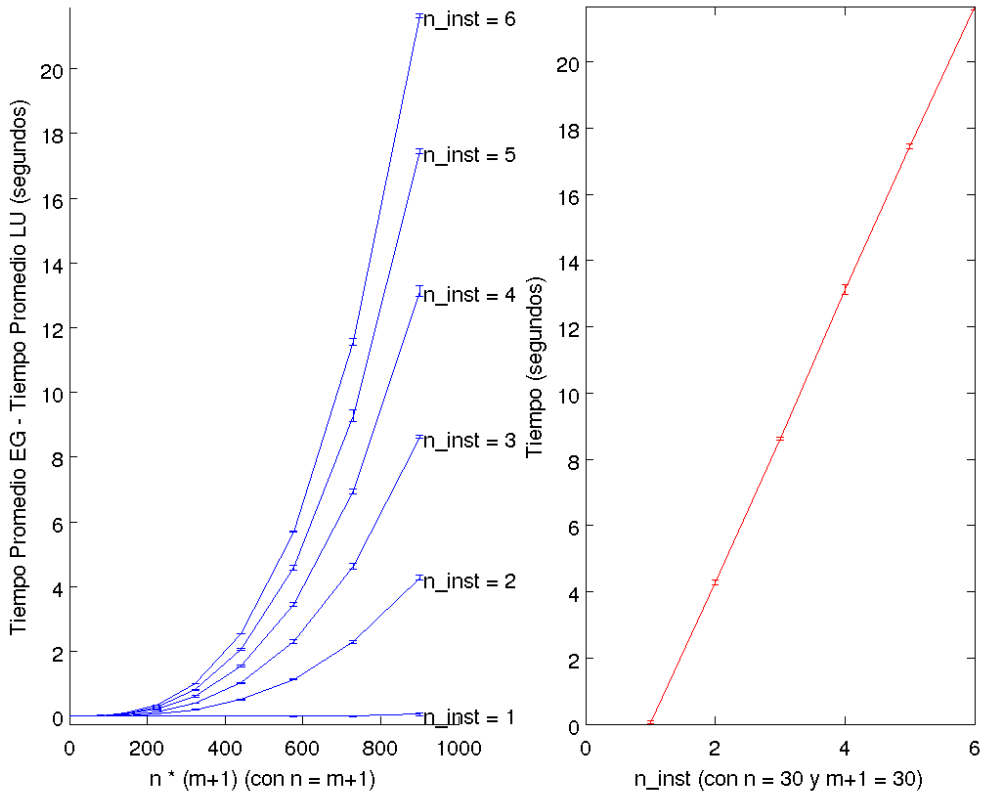

Figura 7: Comparación de Tiempos de Corrida de EG y LU para distintos valores de los Parámetros de Discretización y de n_inst

[8] Vale la pena mencionar no encontramos que este esquema de variación de la temperatura externa máxima modifique los tiempos de ejecución, que es la medida que en este experimento nos interesa evaluar.

4. Discusión

En esta sección analizaremos la información obtenida en Resultados. Para ello, propondremos una explicación o teoría a la cual creemos que los datos se ajustan y luego analizaremos si esto ocurre o no.

Respecto de los dos métodos implementados para calcular la isoterma, tuvimos la intuición (al momento de proponer los métodos y escribir el código) de que el comportamiento del Método del Punto Más Cercano resultaría un poco más errático que el de Interpolación Lineal. Esto se nos ocurrió por el hecho de que, si consideramos dos discretizaciones consecutivas (entendiendo esto como una discretización con n ángulos y n radios y otra con $n+1$ ángulos y radios, dejando fijas las condiciones de temperatura del horno) no es necesariamente cierto que una refine a la otra, es decir, que la discretización de $n+1$ ángulos y radios no necesariamente contiene a los puntos de la discretización de n ángulos y radios, con lo cual el cálculo del punto más cercano para las dos discretizaciones sucesivas podía devolver valores relativamente lejanos entre sí. Sin embargo, también teníamos la suposición de que dichos valores (haciendo un análisis asintótico respecto de la cantidad n de discretizaciones) debían estabilizarse y converger a la ubicación "real" de la isoterma del alto horno que estábamos modelando porque, en el infinito (teniendo todas las temperaturas reales de todos los puntos del horno) calcular el punto cuya temperatura está más cercana a 500^oC es exactamente el punto que está a 500^oC grados.

Combinando estas dos suposiciones, se nos ocurrió que para el Método de Punto Más Cercano podría ocurrir que, si bien no fuera una convergencia suave, sí debería ocurrir que para valores grandes de n la ubicación calculada por este método estuviera contenida en una banda alrededor de la isoterma real. También pensamos que debería suceder algo análogo con el método de Interpolación Lineal, pero que, por varias razones, este sería más preciso y su convergencia a la ubicación de la isoterma sería más veloz. En primer lugar, pensamos que esto ocurriría porque basábamos la aproximación en funciones lineales que unían pares de puntos y de ahí despejaban el hipotético punto a 500°C, con lo cual, al aumentar la cantidad de puntos, sí ocurriría que al hacer cada discretización el cálculo de la isoterma sería más preciso que para la discretización anterior. Esto además daría como resultado una convergencia más suave, con menos saltos bruscos que el método anterior. Además, como cada aproximación de la ubicación de la isoterma sería (según nuestra intuición) mejor que la anterior, conjeturamos que necesitaríamos probar con valores más pequeños de n que en el método del punto más cercano para poder observar si esto efectivamente ocurría o no.

Después de correr los experimentos pudimos corroborar nuestras conjeturas iniciales. En efecto, se puede ver en los gráficos de las secciones 3.1.1 y 3.1.2 (figuras 1 y 3) que el cálculo de la ubicación de la isoterma con el método del punto más cercano da como resultado una curva con muchos más quiebres que la curva que resulta de hacer el cálculo de la isoterma con interpolación lineal. Además, las figuras 3 y 5 muestran, por un lado, cómo la distancia entre la pared externa y la isoterma se estabiliza mucho más rápido y la curva es mucho más suave en el caso de la interpolación lineal (lo cual a su vez implica que la ubicación de la isoterma en sí misma se estabiliza, ya que la pared externa está fija) y por otro lado que, si bien el gráfico del otro método tiene muchos picos y saltos repentinos en los valores, cuando la discretización se va haciendo cada vez más fina (esto es, cuando n es lo suficientemente grande) va quedando encerrado en una banda cada vez más pequeña alrededor de la curva que corresponde a la distancia entre la pared externa y la isoterma calculada con interpolación lineal, lo cual coincide con la intuición de que ambos convergen a la isoterma y que con interpolación lineal la convergencia es más veloz.

En lo referido al tiempo de ejecución de los algoritmos cuando n_inst es igual a 1, podemos ver tanto en la figura 6 como en la curva correspondiente a n_inst igual a 1 en el panel izquierdo de la figura 7, que a medida que aumenta el número de ecuaciones que se debe resolver el aumento que esto implica en términos de costo da lugar a una curva convexa. Esto se corresponde con el análisis de complejidad de ambos algoritmos que dice que la complejidad tanto de eliminación gaussiana como de factorización LU es del orden de $O((n + n\_mas\_1)^3)$.

En lo que se refiere a comparar los resultados obtenidos mediante eliminación gaussiana y factorización LU para valores de n_inst mayores a 1, en el panel izquierdo de la figura 7 se observa que el método

de factorización LU supera el método de eliminación gaussiana en términos de complejidad, pues sólo debe realizar el paso que tiene complejidad $O((n + n\_mas\_1)^3)$ una única vez, mientras que EG debe realizarlo n_inst veces. Es interesante analizar la forma que tienen las curvas para valores de n_inst mayores a 1, pues se observa que las mismas tienen un comportamiento explosivo a medida que aumenta la cantidad de ecuaciones. En base al análisis de la complejidad de ambos métodos sabemos que las diferencias de tiempos para LU y EG para valores de n_inst mayores a 1 debería converger a $O((n + n\_mas\_1)^3)$ pues el costo adicional que tiene EG en triangular la matriz más de una vez tiene este orden de complejidad. A su vez el panel izquierdo de la figura 7 nos muestra otro resultado interesante, en él se aprecia que para un valor de n y m_mas_1 dado, la diferencia de tiempo que existe entre resolver el sistema mediante el método LU y el método EG tiene sigue una tendencia lineal, esto quiere decir que a medida que se agrega una corrida adicional (es decir, a medida que n_inst aumenta en 1) el costo que se paga en tiempo crece a una tasa constante. Esto se debe a que los cálculos que debe realizar el método de eliminación gaussiana cuando aumenta en una unidad n_inst son constantes (se debe triangular el sistema y resolver el mismo una vez más), como también lo son los cálculos que debe realizar el método de factorización LU (debe resolver dos sistemas triangulares), siendo claramente menos costosos los cálculos adicionales que realiza el método LU por sobre los que realiza el método EG.

5. Conclusiones

En el presente informe presentamos nuestra resolución al primer trabajo práctico de métodos numéricos. En él detallamos las decisiones que tomamos para resolverlo, los objetivos que nos propusimos analizar y aspectos referidos al desarrollo. El mismo nos permitió experimentar de manera directa con los conceptos aprendidos en la parte teórica de la materia.

En concreto, los experimentos nos permitieron evidenciar en primera persona qué ventajas tiene la factorización LU por sobre la eliminación gaussiana al momento de tener que resolver más de una vez un sistema con una misma matriz A. Sumado a esto, el problema elegido también nos permitió realizar experimentos que nos ayudaron a fortalecer y desarrollar intuiciones referidas a los sistemas lineales de ecuaciones y a los métodos de discretización y linealización de sistemas de ecuaciones.

Un análisis adicional que llevamos adelante y que no estaba contemplado en las consignas de este trabajo práctico fue el de comparar diferentes métodos para obtener una isoterma, dada la solución a un sistema discretizado. En términos generales notamos que realizar interpolación lineal supera en gran medida el usar el punto más cercano como estimación; sin embargo, también vimos que esto se encuentra estrechamente asociado a las decisiones de discretización, siendo menor la ventaja de interpolación lineal a medida que se aumentan las ecuaciones del sistema.

Finalmente, dado lo acotado del tiempo para realizar este trabajo, hubo experimentos que nos hubiera gustado realizar y no pudimos llevar adelante. A modo de ejemplo: en nuestros experimentos nosostros supusimos siempre que $n = m\_mas\_1$, y creemos que podríamos haber obtenidos resultados interesantes si hubieramos diseñado experimentos que, dada una solución, fuesen aumentando solamente el valor de n, y luego repetir el experimento pero sólo aumentando el valor de m_mas_1 para, de este modo, comparar cómo se modifica la isoterma buscada en los dos casos, pues se ve que en las ecuaciones los términos asociados al ángulo y al radio de los puntos no son idénticos.

6. Apéndice A: Enunciado del trabajo práctico

Laboratorio de Métodos Numéricos - Primer Cuatrimestre 2014
Trabajo Práctico Número 1: Con 15 *thetas* discretizo alto horno...

Introducción

Consideremos la sección horizontal de un horno de acero cilíndrico, como en la Figura 1. El sector A es la pared del horno, y el sector B es el horno propiamente dicho, en el cual se funde el acero a temperaturas elevadas. Tanto el borde externo como el borde interno de la pared forman círculos. Suponemos que la temperatura del acero dentro del horno (o sea, dentro de B) es constante e igual a 1500^oC.

Tenemos sensores ubicados en la parte externa del horno para medir la temperatura de la pared externa del mismo, que habitualmente se encuentra entre 50^oC y 200^oC. El problema que debemos resolver consiste en estimar la isoterma de 500^oC dentro de la pared del horno, para estimar la resistencia de la misma. Si esta isoterma está demasiado cerca de la pared externa del horno, existe peligro de que la estructura externa de la pared colapse.

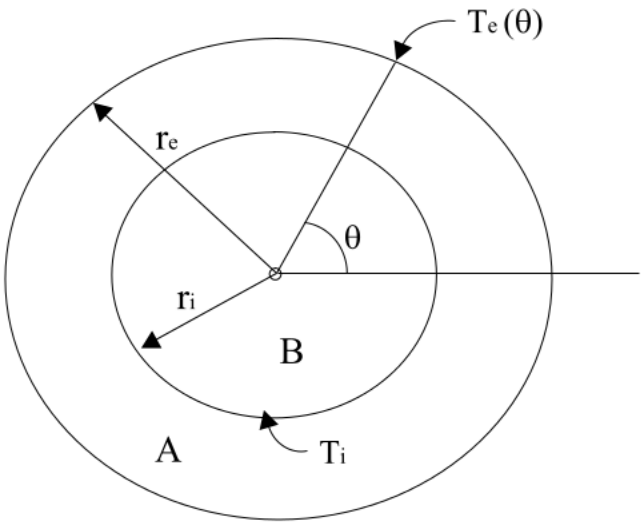

Figura 8: Sección circular del horno

El objetivo del taller es implementar un programa que calcule la isoterma solicitada, conociendo las dimensiones del horno y las mediciones de temperatura en la pared exterior.

El Modelo

Sea $r_e \in \mathbb{R}$ el radio exterior de la pared y sea $r_i \in \mathbb{R}$ el radio interior de la pared. Llamemos $T(r, \theta)$ a la temperatura en el punto dado por las coordenadas polares (r, θ), siendo r el radio y θ el ángulo polar de dicho punto. En el estado estacionario, esta temperatura satisface la ecuación del calor:

$$\frac{\partial^2 T(r,\theta)}{\partial r^2} + \frac{1}{r}\frac{\partial T(r,\theta)}{\partial r} + \frac{1}{r^2}\frac{\partial^2 T(r,\theta)}{\partial \theta^2} = 0 \tag{8}$$

Si llamamos $T_i \in \mathbb{R}$ a la temperatura en el interior del horno (sector B) y $T_e : [0, 2\pi] \to \mathbb{R}$ a la función de temperatura en el borde exterior del horno (de modo tal que el punto (r_e, θ) tiene temperatura

$T_e(\theta)$), entonces tenemos que

$$T(r,\theta) = T_i \quad \textit{para todo punto } (r,\theta) \textit{ con } r \leq r_i \tag{9}$$

$$T(r_e,\theta) = T_e(\theta) \quad \textit{para todo punto } (r_e,\theta) \tag{10}$$

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 interior del horno (sector A), en función de los datos mencionados en esta sección.

Para resolver este problema computacionalmente, discretizamos el dominio del problema (el sector A) en coordenadas polares. Consideramos una partición $0 = \theta_0 < \theta_1 < ... < \theta_n = 2\pi$ en n ángulos discretos con $\theta_k - \theta_{k-1} = \Delta\theta$ para $k = 1, ..., n$, y una partición $r_i = r_0 < r_1 < ... < r_m = r_e$ en $m+1$ radios discretos con $r_j - r_{j-1} = \Delta r$ para $j = 1, ..., m$.

El problema ahora consiste en determinar el valor de la función T en los puntos de la discretización (r_j, θ_k) que se encuentren dentro del sector A. Llamemos $t_{jk} = T(r_j, \theta_k)$ al valor (desconocido) de la función T en el punto (r_j, θ_k).

Para encontrar estos valores, transformamos la ecuación (8) en un conjunto de ecuaciones lineales sobre las incógnitas t_{jk}, evaluando (8) en todos los puntos de la discretización que se encuentren dentro del sector A. Al hacer esta evaluación, aproximamos las derivadas parciales de T en (8) por medio de las siguientes fórmulas de diferencias finitas:

$$\frac{\partial^2 T(r,\theta)}{\partial r^2}(r_j,\theta_k) \cong \frac{t_{j-1,k} - 2t_{jk} + t_{j+1,k}}{(\Delta r)^2} \tag{11}$$

$$\frac{\partial T(r,\theta)}{\partial r}(r_j,\theta_k) \cong \frac{t_{j,k} - t_{j-1,k}}{\Delta r} \tag{12}$$

$$\frac{\partial^2 T(r,\theta)}{\partial \theta^2}(r_j,\theta_k) \cong \frac{t_{j,k-1} - 2t_{jk} + t_{j,k+1}}{(\Delta \theta)^2} \tag{13}$$

Es importante notar que los valores de las incógnitas son conocidos para los puntos que se encuentran sobre el borde exterior de la pared, y para los puntos que se encuentren dentro del sector B. Al realizar este procedimiento, obtenemos un sistema de ecuaciones lineales que modela el problema discretizado. La resolución de este sistema permite obtener una aproximación de los valores de la función T en los puntos de la discretización.

Enunciado

Se debe implementar un programa en `C` o `C++` que tome como entrada los parámetros del problema (r_i, r_e, $m+1$, n, valor de la isoterma buscada, T_i, $T_e(\theta)$) que calcule la temperatura dentro de la pared del horno utilizando el modelo propuesto en la sección anterior y que encuentre la isoterma buscada en función del resultado obtenido del sistema de ecuaciones. El método para determinar la posición de la isoterma queda a libre elección de cada grupo y debe ser explicado en detalle en el informe.

El programa debe formular el sistema obtenido a partir de las ecuaciones (1) - (6) y considerar dos métodos posibles para su resolución: mediante el algoritmo clásico de Eliminación Gaussiana y la Factorización LU. Finalmente, el programa escribirá en un archivo la solución obtenida con el formato especificado en la siguiente sección.

Como ya se ha visto en la materia, no es posible aplicar los métodos propuestos para la resolución a cualquier sistema de ecuaciones. Sin embargo, la matriz del sistema considerado en el presente trabajo

cumple con ser diagonal dominante (no estricto) y que sus filas son linealmente independientes. Luego, se pide demostrar el siguiente resultado e incluirlo en el informe:

Proposición 1 *Sea $A \in \mathbb{R}^{n \times n}$ la matriz obtenida para el sistema definido por (1)-(6). Demostrar que es posible aplicar Eliminación Gaussiana sin pivoteo.*[9]

En función de la experimentación, se pide como mínimo realizar los siguientes experimentos:

- Considerar al menos dos instancias de prueba, generando distintas discretizaciones para cada una de ellas y comparando la ubicación de la isoterma buscada respecto de la pared externa del horno. Se sugiere presentar gráficos de temperatura o curvas de nivel para los mismos, ya sea utilizando las herramientas provistas por la cátedra o implementando sus propias herramientas de graficación. Estudiar la proximidad de la isoterma buscada respecto de la pared exterior del horno en función de distintas granularidades de discretización y las condiciones de borde. Analizar también el tiempo de cómputo requerido en función de la granularidad de la discretización.
- Considerar un escenario similar al propuesto, pero donde las condiciones de borde (i.e., T_i y $T_e(\theta)$) cambian en distintos instantes de tiempo. En este caso, buscamos obtener la secuencias de estados de la temperatura en la pared del horno, y la respectiva ubicación de la isoterma especificada. Para ello, se considera una secuencia de *ninst* vectores con las condiciones de borde, y las temperaturas en cada estado es la solución del correspondiente sistema de ecuaciones. Se pide formular al menos un experimento de este tipo, aplicar los métodos de resolución propuestos de forma conveniente y compararlos en términos de tiempo total de cómputo requerido para distintos valores de *ninst*.

Finalmente, se deberá presentar un informe que incluya una descripción detallada de los métodos implementados y las decisiones tomadas, el método propuesto para el cálculo de la isoterma buscada y los experimentos realizados, junto con el correspondiente análisis y siguiendo las pautas definidas en el archivo `pautas.pdf`.

Programa y formato de archivos

Se deberán entregar los archivos fuentes que contengan la resolución del trabajo práctico. El ejecutable tomará tres parámetros por línea de comando, que serán el archivo de entrada, el archivo de salida, y el método a ejectutar (0 EG, 1 LU).

El archivo de entrada tendrá la siguiente estructura:

- La primera línea contendrá los valores r_i, r_e, $m+1$, n, *iso*, *ninst*, donde *iso* representa el valor de la isoterma buscada y *ninst* es la cantidad de instancias del problema a resolver para los parámetros dados.
- A continuación, el archivo contendrá *ninst* líneas, cada una de ellas con $2n$ valores, los primeros n indicando los valores de la temperatura en la pared interna, i.e., $T_i(\theta_0), T_i(\theta_1), \ldots, T_i(\theta_{n-1})$, seguidos de n valores de la temperatura en la pared externa, i.e., $T_e(\theta_0)$,$T_e(\theta_1)$,$\ldots$,$T_e(\theta_{n-1})$.

El archivo de salida obligatorio tendrá el vector solución del sistema reportando una componente del mismo por línea. En caso de $ninst > 1$, los vectores serán reportados uno debajo del otro.

Junto con el presente enunciado, se adjunta una serie de scripts hechos en `python` y un conjunto instancias de test que deberán ser utilizados para la compilación y un testeo básico de la implementación. Se recomienda leer el archivo `README.txt` con el detalle sobre su utilización.

Fechas de entrega

[9] Sugerencia: Notar que la matriz es diagonal dominante (no estrictamente) y analizar que sucede al aplicar un paso de Eliminación Gaussiana con los elementos de una fila.

- *Formato Electrónico:* Jueves 10 de Abril de 2014, hasta las 23:59 hs, enviando el trabajo (informe + código) a la dirección `metnum.lab@gmail.com`. El subject del email debe comenzar con el texto `[TP1]` seguido de la lista de apellidos de los integrantes del grupo.
- *Formato físico:* Viernes 11 de Abril de 2014, de 17:30 a 18:00 hs.

Importante: El horario es estricto. Los correos recibidos después de la hora indicada serán considerados re-entrega. Los grupos deben ser de exactamente 3 personas. Es indispensable que los trabajos pasen satisfactoriamente los casos de test provistos por la cátedra.

7. Apéndice B

En este apéndice incluiremos el código utilizado para construir, a partir de los parámetros de entrada del problema, la matriz A y el vector b , así como también los métodos empleados para resolver sistemas de ecuaciones lineales, las mediciones temporales de resolución de dichos sistemas y la escritura de los archivos de salida.

```
#include <iostream>
#include <vector>
#include <fstream>
#include <string>
#include <iomanip>
#include <boost/algorithm/string.hpp>
#include <boost/lexical_cast.hpp>
#include <stdlib.h>
#define _USE_MATH_DEFINES
#include <math.h>
#include <sys/time.h>
using namespace std;
using namespace boost;
typedef vector < vector < double > > matriz;
typedef vector <double> columna;

void gauss(matriz& A,columna& b){
  //Lleva al sistema Ax=b a uno equivalente Ux=d, donde U es triangular superior.
  double m_ji = 0;
  for(unsigned int i = 0; i < A.size() - 1; i++){
    for(unsigned int j = i + 1; j < A.size(); j++){
      m_ji = A[j][i] / A[i][i];
      b[j] = b[j] - m_ji*b[i];
      for(unsigned int k = i; k < A.size(); k++){
        A[j][k] = A[j][k] - m_ji*A[i][k];
      }
    }
  }
}


columna resolver_ts(const matriz& A, const columna& b){
//Resuelve un sistema lineal de la forma Ax = b, donde A es triangular superior.
  columna x;
  for(unsigned int i = 0; i < A.size(); i++){
    x.push_back(0);
  }
  double aux = 0;
  int n = b.size() - 1;
  for(int i = n; i >= 0; i--){
    aux = 0;
    if(i==n){
      aux = b[n] / A[n][n];
      x[n] = aux;
    }
    else{
      aux = b[i];
      for(int j = i + 1; j <= n;j++){
        aux = aux - A[i][j]*x[j];
```

```
      }
    aux = aux/A[i][i];
    x[i] = aux;
    }
  }
  return x;
}

columna resolver_ti(const matriz& A, const columna& b){
//Resuelve un sistema lineal de la forma Ax = b, donde A es triangular inferior.
  columna x;
  for(unsigned int i = 0; i < A.size(); i++){
    x.push_back(0);
  }
  double aux = 0;
  int n = b.size() - 1;
  for(int i = 0; i <= n; i++){
    if(i==0){
      aux = b[0] / A[0][0];
      x[0] = aux;
    }
    else{
      aux = b[i];
      for(unsigned int j = 0; j <= (i - 1); j++){
        aux = aux - A[i][j]*x[j];
      }
      x[i] = aux/A[i][i];
    }
  }
  return x;
}

void factorizacionLU(matriz& A, matriz& L){
  double m_ji = 0;
  for(unsigned int i = 0; i < A.size() - 1; i++){
    for(unsigned int j = i + 1; j < A.size(); j++){
      m_ji = A[j][i] / A[i][i];
      for(unsigned int k = i; k < A.size(); k++){
        A[j][k] = A[j][k] - m_ji*A[i][k];
      }
      L[j][i] = m_ji;
    }
  }
  for(unsigned int i = 0; i < A.size(); i++){
    L[i][i] = 1;
  }
}

columna vectorB(columna& vlsPared, unsigned int& n, unsigned int& m_mas_1){
// Toma las temperaturas en ambas paredes y arma el vector b
  columna out;
  out.resize(n * m_mas_1);
  for(unsigned int i = 0; i < vlsPared.size(); i++){
    if (i < n) {
      out[i] = vlsPared[i];
    }
    else {
      out[i %n + (m_mas_1 -1 ) * n] = vlsPared[i];
    }
  }
  return out;
}

matriz matrizA(unsigned int& n, unsigned int& m_mas_1, double& r_i, double& r_e){
  matriz out;
  out.resize(n * m_mas_1);
  for(unsigned int i = 0; i < out.size(); i++){
    out[i].resize(n * m_mas_1);
  }
  double dR = (r_e - r_i) / (m_mas_1 - 1);
  double dA = (2 * M_PI) / (n);
  for(unsigned int i = 0; i < out.size(); i++){
    if ((i < n)) {
```

```
      out[i][i] = 1;
    }
    else if (i > (n * m_mas_1) - 1 - n){
      out[i][i] = 1;
    }
    else {
      double dstR = r_i + (int) (i/n)*dR; // distancia del punto al centro.
      out[i][i] = 1/(dstR*dR) - 2*(1/(dR * dR) + 1/((dstR*dA) * (dstR*dA)));
      out[i][i + n] = 1/(dR*dR);
      out[i][i - n] = 1/(dR*dR) - 1/(dstR*dR);
      int jAnt;
      if((i %n) == 0){
        jAnt = i + (n-1);
      }
      else{
        jAnt = i - 1;
      }
      out[i][jAnt] = 1/(dstR * dA * dstR * dA);
      int jPost;
      if (((i + 1) %n) == 0){
        jPost = i - (n-1);
      }
      else {
        jPost = i + 1;
      }
      out[i][jPost] = 1/(dstR * dA * dstR * dA);
    }
  }
  return out;
}

columna isotermaInterLineal(double& iso, columna& x, unsigned int n,
unsigned int m_mas_1, double& r_e, double& r_i) {
  columna out;
  out.resize(n);
  double dR = (r_e - r_i) / (m_mas_1 - 1);
  for (unsigned int i = 0; i < n; i++){
    int j = 0;
    while (x[i + j*n] > iso && x[i + (j+1)*n] > iso){
      j++;
    }
  out[i] = (iso-x[i+j*n])*dR / (x[i+(j+1)*n]-x[i + j*n]) + (j*dR + r_i);
  }
  return out;
}

double modulo(double n){
  double res;
  if(n>=0){
    res = n;
  }
  else{
    res = (-1)*n;
  }
  return res;
}

columna isotermaPuntoMasCercano(double r_i, double r_e, double iso, int n,
int m_mas_1, columna temperaturas){
  double dR = (r_e-r_i)/(m_mas_1 - 1);
  columna isoterma;
  isoterma.resize(n);
  for(int i = 0; i < n; i++){
    unsigned int MenorDiferencia = 0;
    for(int j = 0; j < m_mas_1;j++){
      if(modulo(temperaturas[i+j*n] - iso) < modulo(temperaturas[i+MenorDiferencia*n])
          - iso){
        MenorDiferencia = j;
      }
    }
    isoterma[i] = r_i + (MenorDiferencia)*dR;
  }
  return isoterma;
```

```
}

int main(int argc, char *argv[]){
  char *inFile = argv[1];
  char *outFile = argv[2];
  int tipo = atoi(argv[3]);
  char *isoOutFile = argv[4];
  int tipoIso = atoi(argv[5]);
  string l;
  ifstream myfile(inFile);
  assert(myfile.is_open());
  getline(myfile,l);
  trim(l);
  vector< string > parametros;
  split(parametros, l, is_any_of("␣"), token_compress_on);
  double r_i = lexical_cast<double>(parametros[0]);
  double r_e = lexical_cast<double>(parametros[1]);
  unsigned int m_mas_1 = lexical_cast<int>(parametros[2]);
  unsigned int n = lexical_cast<int>(parametros[3]);
  double iso = lexical_cast<double>(parametros[4]);
  unsigned int n_inst = lexical_cast<int>(parametros[5]);
  ofstream salida;
  salida.open(outFile, ios::out);
  ofstream tiempoEjecucion;
  tiempoEjecucion.open("tiempoEjecucion.out", ios::out);
  ofstream salidaIso;
  if(argc > 5){
    salidaIso.open(isoOutFile, ios::out);
  }

  columna vlsPared;
  vlsPared.resize(2 * n);
  vector< string > vlsParedTmp;
  matriz A = matrizA(n, m_mas_1, r_i, r_e);
  matriz L;
  L.resize(A.size());
  for(unsigned int i = 0; i < L.size(); i++){
    L[i].resize(A.size());
  }

  double tiempoTotal = 0.0;
  timeval tic, toc;
  for (unsigned int inst = 0; inst < n_inst; inst++){
    getline(myfile,l);
    trim(l);
    split(vlsParedTmp, l, is_any_of("␣"), token_compress_on);
    for(unsigned int i = 0; i < vlsParedTmp.size(); i++){
      vlsPared[i] = 0;
      vlsPared[i] = lexical_cast<double>(vlsParedTmp[i]);
    }
    columna b = vectorB(vlsPared, n, m_mas_1);
    columna x;
    if(tipo == 0){
      if(inst > 0){
        matriz A = matrizA(n, m_mas_1, r_i, r_e);
      }
      gettimeofday(&tic, NULL);
      gauss(A, b);
      x = resolver_ts(A, b);
      gettimeofday(&toc, NULL);
    }
    else if(tipo == 1){
      columna y;
      if (inst == 0) {
        gettimeofday(&tic, NULL);
        factorizacionLU(A,L);
      }
      else{
        gettimeofday(&tic, NULL);
      }
      y = resolver_ti(L, b);
      x = resolver_ts(A, y);
      gettimeofday(&toc, NULL);
```

```
    }
    for (unsigned int j = 0; j < x.size(); j++) {
      salida << std::setprecision(15) << x[j] << endl;
    }
    if(argc > 5){
      columna isoterma;
      if(tipoIso==0){
        isoterma = isotermaPuntoMasCercano(r_i, r_e, iso, n, m_mas_1, x);
      }
      else{
        isoterma = isotermaInterLineal(iso, x, n, m_mas_1, r_e, r_i);
      }
      for(unsigned int s = 0; s < isoterma.size(); s++){
        salidaIso << std::setprecision(15) << isoterma[s] << endl;
      }
    }
      tiempoTotal = tiempoTotal + (1000000*(toc.tv_sec-tic.tv_sec)+(toc.tv_usec-tic.
          tv_usec))/1000000.0;
  }

  tiempoEjecucion << std::setprecision(15) << tiempoTotal << endl;
  salida.close();
  tiempoEjecucion.close();
  if(argc > 5){
    salidaIso.close();
  }
  return 0;
}
```

8. Apéndice C

8.1. Esquema de la demostración

Sea A $\in \mathbb{R}^{n \times n}$ la matriz obtenida para el sistema de ecuaciones. Queremos probar que se puede aplicar el algoritmo de eliminación gaussiana sin pivoteo. Para probar esto, usaremos como hipótesis extra (facilitada por la cátedra en el enunciado del trabajo práctico) el hecho de que la matriz A es diagonal dominante (no estricto) y que sus filas son linealmente independientes. Dividiremos la demostración en varias partes:

- $a_{1,1} \neq 0$
- Si definimos $A^{(1)}$ como la matriz que resulta de ejecutar el primer paso del algoritmo de Gauss sobre la matriz A, el siguiente paso es probar que la submatriz de $\mathbb{R}^{n-1 \times n-1}$ que resulta de eliminar la primer fila y la primer columna de $A^{(1)}$ es diagonal dominante (no estricto) y tiene filas linealmente independientes. Es importante señalar que este paso implicará que pueden hacerse todas las iteraciones del método de Gauss, ya que quedará demostrado que si se parte de una matriz diagonal dominante (no estricto) con filas linealmente independientes, entonces se puede hacer una iteración y la matriz (de una dimensión menos) que hay que seguir triangulando preservará las dos propiedades (diagonal dominante, filas l.i.) que permitieron hacer la iteración anterior.

8.2. Demostración

- En primer lugar, queremos ver que $a_{1,1} \neq 0$. Procedemos por el absurdo. Supongamos $a_{1,1} = 0$. Por ser diagonal dominante no estricto, tenemos que $|a_{1,1}| \geq \sum_{j=2}^{n} |a_{1,j}|$, y por la igualdad anterior, tenemos que $\sum_{j=2}^{n} |a_{1,j}| \leq 0$. Como cada término de la suma es positivo, tenemos que $|a_{1,j}| = 0$, $\forall j = 2, .., n \implies a_{1,j} = 0$, $\forall j = 2, .., n$. Como además $a_{1,1} = 0$, tenemos que toda la primer fila de la matriz es nula, lo cual es absurdo pues, por hipótesis, las filas de A eran linealmente independientes. Luego, $a_{1,1} \neq 0$.

- Probado el punto anterior, se puede proceder con el primer paso de la triangulación de Gauss. Sea $A^{(1)}$ la matriz previamente definida. Queremos probar que la submatriz de $\mathbb{R}^{n-1\times n-1}$ que resulta de eliminar la primer fila y la primer columna de $A^{(1)}$ es diagonal dominante (no estricto) y tiene filas linealmente independientes.

  Veamos primero que es diagonal dominante:

  Sea $i \geq 2$. Queremos ver que $\sum_{j=2,j\neq i}^{n} |a_{i,j}^{(1)}| \leq |a_{i,i}^{(1)}|$. (Notar que la suma empieza desde $j = 2$ pues, como ya hemos mencionado, estamos considerando la submatriz de $A^{(1)}$ sin la primera fila ni la primer columna). Por cómo se define el algoritmo de Gauss, tenemos que:

$$a_{i,j}^{(1)} = a_{i,j} - \frac{a_{1,j} * a_{i,1}}{a_{1,1}}. \tag{14}$$

  Reemplazando en el término general de la sumatoria, obtenemos:

$$\sum_{j=2,j\neq i}^{n} |a_{i,j}^{(1)}| = \sum_{j=2,j\neq i}^{n} |a_{i,j} - \frac{a_{1,j} * a_{i,1}}{a_{1,1}}| \tag{15}$$

  Aplicando la desigualdad triangular:

$$\sum_{j=2,j\neq i}^{n} |a_{i,j} - \frac{a_{1,j} * a_{i,1}}{a_{1,1}}| \leq \sum_{j=2,j\neq i}^{n} |a_{i,j}| + \sum_{j=2,j\neq i}^{n} \frac{|a_{1,j} * a_{i,1}|}{|a_{1,1}|} \tag{16}$$

  Sumando y restando $|a_{i,1}|$:

$$\sum_{j=2,j\neq i}^{n} |a_{i,j}| + \sum_{j=2,j\neq i}^{n} \frac{|a_{1,j} * a_{i,1}|}{|a_{1,1}|} = \sum_{j=2,j\neq i}^{n} |a_{i,j}| + |a_{i,1}| - |a_{i,1}| + \sum_{j=2,j\neq i}^{n} \frac{|a_{1,j} * a_{i,1}|}{|a_{1,1}|} \tag{17}$$

  Agrupando convenientemente los términos:

$$\sum_{j=2,j\neq i}^{n} |a_{i,j}| + |a_{i,1}| - |a_{i,1}| + \sum_{j=2,j\neq i}^{n} \frac{|a_{1,j} * a_{i,1}|}{|a_{1,1}|} = \sum_{j=1,j\neq i}^{n} |a_{i,j}| - |a_{i,1}| + \sum_{j=2,j\neq i}^{n} \frac{|a_{1,j} * a_{i,1}|}{|a_{1,1}|} \tag{18}$$

  Usando que A es diagonal dominante (en el caso particular de la fila i):

$$\sum_{j=1,j\neq i}^{n} |a_{i,j}| - |a_{i,1}| + \sum_{j=2,j\neq i}^{n} \frac{|a_{1,j} * a_{i,1}|}{|a_{1,1}|} \leq |a_{i,i}| - |a_{i,1}| + \frac{|a_{i,1}|}{|a_{1,1}|} * \sum_{j=2,j\neq i}^{n} |a_{1,j}| \tag{19}$$

  Aplicando la misma propiedad, esta vez para la fila 1, obtenemos:

$$\sum_{j=1,j\neq i}^{n} |a_{1,j}| \leq |a_{1,1}| \implies \sum_{j=2,j\neq i}^{n} |a_{1,j}| \leq |a_{1,1}| - |a_{1,i}| \tag{20}$$

  Aplicando la desigualdad de (20) en (19), nos queda:

$$|a_{i,i}| - |a_{i,1}| + \frac{|a_{i,1}|}{|a_{1,1}|} * \sum_{j=2,j\neq i}^{n} |a_{1,j}| \leq |a_{i,i}| - |a_{i,1}| + \frac{|a_{i,1}|}{|a_{1,1}|} * (|a_{1,1}| - |a_{1,i}|) \tag{21}$$

  Aplicando la propiedad distributiva sobre el último término y cancelando términos, obtenemos:

$$|a_{i,i}| - |a_{i,1}| + \frac{|a_{i,1}|}{|a_{1,1}|} * (|a_{1,1}| - |a_{1,i}|) = |a_{i,i}| - \frac{|a_{i,1}|}{|a_{1,1}|} * |a_{1,i}| \tag{22}$$

  Por propiedad del valor absoluto:

$$|a_{i,i}| - \frac{|a_{i,1}|}{|a_{1,1}|} * |a_{1,i}| \leq |a_{i,i} - \frac{a_{i,1} * a_{1,i}}{a_{1,1}}| \tag{23}$$

Pero esa última expresión es justamente $|a_{i,i}^{(1)}|$, por lo cual, mirando ambos extremos de la desigualdad que acabamos de desarrollar:

$$\sum_{j=2, j\neq i}^{n} |a_{i,j}^{(1)}| \leq |a_{i,i}^{(1)}| \tag{24}$$

Como queríamos probar.

Ver que las $n-1$ filas de $\mathbb{R}^{n-1}$ (las que forma la matriz $A^{(1)}$ sin su primer fila y columna) son linealmente independientes es inmediato. Sea B $\in \mathbb{R}^{n-1 \times n-1}$ la matriz $A^{(1)}$ luego de eliminarle la primer fila y columna. Quiero ver que B tiene filas linealmente independientes, o equivalentemente, que su determinante es distinto de cero. Se puede calcular el determinante de $A^{(1)}$ como el producto de los determinantes de dos bloques, donde uno de ellos es $a_{1,1}$ y el otro es el de la matriz B (usando fuertemente que debajo de $a_{1,1}$ hay ceros, en otro caso esto sería falso). El determinante de $A^{(1)}$ es distinto de cero, pues es $A^{(1)}$ es el resultado del producto entre A (inversible, por hipótesis) y matrices elementales (también inversibles), por lo cual se obtiene que:

$$det(A^{(1)}) = (a_{1,1} * det(B)) \neq 0 \implies det(B) \neq 0 \tag{25}$$

Como queríamos probar.

Esto completa la demostración. Hemos probado que partiendo de una matriz diagonal dominante no estricto cuyas filas son linealmente independientes, se puede hacer el primer paso de la Triangulación Gaussiana, y la submatriz sobre la cual hay que aplicar el segundo paso del algoritmo también cumple con tener filas linealmente independientes y ser diagonal dominante no estricto, con lo cual se puede proceder recursivamente, haciendo de un paso por vez (teniendo ahora probado que para un tamaño menos el siguiente paso también se podrá hacer) hasta haber triangulado completamente la matriz original.

9. Referencias

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