DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1: CQTDAH

10 / 4 / 2014 Métodos Numéricos

Grupo: Nombre Pendiente

Integrante	LU	Correo electrónico
Rama, Roberto	490/11	bertoski@gmail.com
Vanotti, Marco	229/10	mvanotti@dc.uba.ar
Ventura, Alejandro	249/11	vn2.amv@gmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Introducción

1.1. Objetivos

El objetivo principal del presente trabajo es la estimación de las temperaturas internas a la pared de un alto horno (horno de fundición de metales), para poder así determinar la posición aproximada de la isoterma de temperatura 'iso', y poder estudiar el estado y fiabilidad del horno. Se busca además comparar la eficiencia de distintos métodos computacionales para la obtención de dicha isoterma, para el caso en que hay que realizar una medición única de la temperatura del horno, o varias.

1.2. Introducción

El fenómeno físico de transmisión del calor es descripto detalladamente en la conocida ecuación del calor, para el caso en el que solo se desea conocer el equilibrio final de temperaturas de un sistema, se puede disponer de ecuaciones mas sencillas, estas ecuaciones de estado dependen de las características del sistema, y del tipo de coordenadas usadas. Nuestro caso consiste de un sistema físico en forma de 'disco', medido en coordenadas polares, esto es, con las variables r y θ. La ecuación que describe la temperatura para cada par (r, θ) en este sistema es una ecuación de segundo orden, en derivadas parciales:

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

Como los fenómenos y modelos físicos tienen una naturaleza continua mientras que los modelos computacionales tienen una naturaleza discreta, los mismos crean la necesidad de realizar un proceso de discretización sobre el modelo físico inicial y una interpolación para refinar la solución encontrada.
Para la discretización de la ecuación se utilizan formulas de diferencias finitas, que son versiones discretas de la definición continua de derivada. Se hacen aproximaciones discretas a la variable radio. Así se puede transformar una ecuación diferencial de estas características en un sistema de ecuaciones lineales.
Como método de interpolación, se puede utilizar una función lineal que una los puntos (r_i, t_i) y (r_{i+1}, t_{i+1}) del dominio. A esta función luego se le calcula la inversa, con lo cual para cada temperatura (fijado el ángulo), devuelve un valor que representa al radio (entre r_i y r_{i+1}) correspondido con esa temperatura.

Luego usaremos la discretización antes descripta para obtener un sistema de ecuaciones lineales que resolveremos mediante eliminación Gaussiana y factorización LU. La eliminación Gaussiana es un algoritmo para la resolución de sistemas de ecuaciones lineales de la forma $Ax = b$, donde A es una matriz que representa los coeficientes de las variables del sistema, x es el vector incógnita, y b el vector termino independiente. El método consiste en la obtención de una matriz triangular mediante la realización de operaciones elementales en una matriz que representa el sistema de ecuaciones. Una vez triangulada la matriz se procede a hacer la substitución hacia atrás (backward substitution). La complejidad del algoritmo es de $\mathcal{O}(n^3)$ sobre el tamaño de la matriz, si bien las operaciones de la substitución pueden realizarse solo en $\mathcal{O}(n^2)$.

Un método para optimizar el procedimiento cuando tenemos varias instancias (distintas mediciónes y por lo tanto distinto vector termino independiente) a resolver relacionadas por el mismo modelo físico (sin alterar la matriz característica) es la factorización LU (Lower, Upper) de la matriz. La misma consiste en escribir a la matriz inicial como el producto de una matriz triangular inferior y otra triangular superior, llamadas L y U respectivamente.
Con esto la ecuación vectorial del sistema inicial es equivalente a $LUx = b$, pudiendo escribirse como dos ecuaciónes distintas, llamando 'y' a 'Ux', como $Ly = b$ y $Ux = b$. Al ser las matrices de estas ecuaciónes, triangulares no hace falta la utilización de la eliminación Gaussiana, solamente la substitución hacia atrás. Con este método se espera conseguir una complejidad de $\mathcal{O}(n^2)$ una vez realizada la factorización, lo que podría mejorar el rendimiento, a la hora de realizar varias mediciones con un mismo modelo físico.

2. Desarrollo

2.1. Construcción del modelo

Sean $r_e, r_i \in \mathcal{R}$ el radio exterior de la pared del horno y el radio interno respectivamente. Llamaremos $T(r, \theta)$ 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 estado estacionario, esta temperatura satisface la ecuación de calor:

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

Si llamamos $T_i \in \mathcal{R}$ a la temperatura en el interior del horno y $T_e : [0, 2\pi] \to \mathcal{R}$ a la función de temperatura en el borde exterior del horno (correspondiente a los sensores) tendremos que $T(r, \theta) = T_i$ para todo punto (r, θ) con $r \leq r_i$ y que $T(r_e, \theta) = T_e(\theta)$ para todo punto (r_e, θ).

Para construir nuestro modelo y lograr una resolución computacional comenzaremos con una discretización del dominio del problema a resolver, ya que el mismo en la vida real tiene infinitos radios y ángulos. Consideraremos las siguientes particiones:

- $0 = \theta_0 < \theta_1 < ... < \theta_n = 2\pi$ en n **ángulos** discretos con $\theta_k - \theta_{k-1} = 2\pi/n = \Delta\theta$ para $k = 1, ..., n$.
- $r_i = r_0 < r_1 < ... < r_m = r_e$ en m **radios** discretos con $r_j - r_{j-1} = (r_i - r_e)/(m-1) = \Delta r$ para $j = 1, ..., m$.

Es importante remarcar que la discretización de los ángulos esta en limitada por la cantidad de sensores que tendremos en el horno real, no pudiendo superar a la cantidad de los mismos ya que serán nuestros datos de entrada.

Una vez discretizados los valores de las coordenadas polares el problema ahora consiste en determinar el valor de la función T en los puntos (r_j, θ_k). Llamemos $t_{jk} = T(r_j, \theta_k)$ al valor de la función T en el punto (r_j, θ_k), cuyos valores son desconocidos. Aproximaremos las derivadas de la ecuación (1) de la siguiente manera:

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

$$\frac{\partial T(r,\theta)}{\partial r}(r_j, \theta_k) \cong \frac{t_{j,k} - t_{j-1,k}}{\Delta r} \tag{3}$$

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

Si reemplazamos las aproximaciones (2), (3) y (4) en (1) obtendremos:

$$\frac{t_{j-1,k} - 2t_{j,k} + t_{j+1,k}}{(\Delta r)^2} + \frac{t_{j,k} - t_{j-1,k}}{r(\Delta r)} + \frac{t_{j,k-1} - 2t_{j,k} + t_{j,k+1}}{r^2(\Delta \theta)^2} = 0 \tag{5}$$

Para cada punto (r_j, θ_k), lo que nos deja con un sistema de ecuaciones lineales que modela el problema discretizado.

2.2. Idea de resolución

La idea es resolver el sistema de ecuaciones en un primer lugar representándolo en matrices y luego utilizando eliminación Gaussiana y factorización LU para resolver el mismo. El sistema quedará de la forma $At = b$ siendo t y b vectores columna en donde A contendrá a los coeficientes de la ecuación de calor (5), b contendrá los datos para resolver el sistema y t contendrá cada punto interno del horno correspondiente con la discretización.

Los vectores columna t y b tendrán ambos dimensión nm. La estructura de t estará conformada por los valores $t_{1,1}, t_{1,2}, ..., t_{1,n}, t_{2,1} ..., t_{m,n}$ dispuestos en forma de columna. Por otro lado, el vector b tendrá el valor 1500 en las primeras n posiciones (correspondientes al radio interno del horno), ceros en las siguientes $m(n-2)$ posiciones y los valores correspondientes a $T_e(\theta_1), T_e(\theta_2), ..., T_e(\theta_n)$ en las últimas n posiciones (correspondientes a los valores de los sensores).

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ \vdots & & \ddots & & & & \vdots \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & \dots & & t_{i,j-1} & t_{i,j} & \dots & \\ \vdots & & & \vdots & & & \vdots \\ 0 & \dots & & 0 & & \dots & 1 \end{pmatrix} \begin{pmatrix} t_{1,1} \\ t_{1,2} \\ \vdots \\ t_{1,n} \\ t_{2,1} \\ \vdots \\ t_{m,n} \end{pmatrix} = \begin{pmatrix} 1500 \\ 1500 \\ \vdots \\ 1500 \\ 0 \\ \vdots \\ T_e(\theta_n) \end{pmatrix}$$

La matriz A por otro lado será cuadrada de tamaño nm en cada dimensión. Las primeras n filas serán una identidad que se corresponderá con las temperaturas en el radio interno, que como son dato las conocemos. Las siguientes $m(n-2)$ filas corresponderán a las ecuaciones de calor por cada punto interno de la discretización. Cada ecuación tendrá nm términos (aunque la mayoría de ellos acompañados por un coeficiente nulo) por lo que la matriz tendrá nm columnas. Finalmente en las ultimas n filas de la matriz se encontrará la identidad que se corresponderá con las temperaturas externas, es decir aquellas registradas por los sensores.

Ahora solamente nos falta saber como llenar A. Si hacemos un manejo algebraico sobre la ecuación (5) juntando los elementos iguales nos queda que:

$$\implies \frac{t_{j-1,k}}{(\Delta r)^2} - \frac{2t_{j,k}}{(\Delta r)^2} + \frac{t_{j+1,k}}{(\Delta r)^2} + \frac{t_{j,k}}{r(\Delta r)} - \frac{t_{j-1,k}}{r(\Delta r)} + \frac{t_{j,k-1}}{r^2(\Delta\theta)^2} - \frac{2t_{j,k}}{r^2(\Delta\theta)^2} + \frac{t_{j,k+1}}{r^2(\Delta\theta)^2} = 0$$

$$\implies \frac{t_{j,k}}{r(\Delta r)} - \frac{2t_{j,k}}{(\Delta r)^2} - \frac{2t_{j,k}}{r^2(\Delta\theta)^2} + \frac{t_{j-1,k}}{(\Delta r)^2} - \frac{t_{j-1,k}}{r(\Delta r)} + \frac{t_{j+1,k}}{(\Delta r)^2} + \frac{t_{j,k-1}}{r^2(\Delta\theta)^2} + \frac{t_{j,k+1}}{r^2(\Delta\theta)^2} = 0$$

Es decir que me queda para la ecuación correspondiente a cada $t_{j,k}$:

$$c_1 = c_{j,k} = \frac{1}{r\Delta r} - \frac{2}{(\Delta r)^2} - \frac{2}{(r\Delta\theta)^2} \quad (6)$$

$$c_2 = c_{j-1,k} = \frac{1}{(\Delta r)^2} - \frac{1}{r\Delta r} \quad (7)$$

$$c_3 = c_{j+1,k} = \frac{1}{(\Delta r)^2} \quad (8)$$

$$c_4 = c_{j,k-1} = c_{j,k+1} = \frac{1}{(r\Delta\theta)^2} \quad (9)$$

$$r = r_i + (j-1)(\Delta r)$$

$$\Delta\theta = 2\pi/n$$

$$\Delta r = (r_i - r_e)/(m-1)$$

Finalmente para llenar A la inicializaremos en cero y luego nos moveremos por la diagonal. Si estamos en alguna de las dos áreas de identidad, es decir las primeras o últimas n filas, pondremos un 1. De lo contrario asignaremos el valor correspondiente a $c_{j,k}$ para el j y k apropiados y los otros 4 valores en las posiciones correspondientes. En los casos en modificamos j nos moveremos n columnas hacia la izquierda si restamos y hacia la derecha si sumamos. En los casos en los que $k+1$ o $k-1$ se salgan fuera del rango $[1, n]$ le aplicaremos módulo n, moviendonos así $n-1$ columnas hacia la izquierda o derecha respectivamente.

2.3. Algoritmos

```
1: function WithFifteenThetas(r_i, r_e, n, m, T_i, T_e)
2:     dr ← (r_e − r_i)/(m − 1)
3:     dt ← 2π/n
4:     Matrix b ∈ R^{nm×1}, A ∈ R^{nm×nm}
5:     for i ∈ [1, nm] do
6:         b[i][1] ← T_i[i]
7:         b[nm − i][1] ← T_e[i]
8:     end for
9:     j ← 1, k ← 1
10:    for i ∈ [1, nm] do
11:        if i ∈ [1, n] ∨ i ∈ [(n − 1)m, nm] then
12:            A[i][i] ← 1
13:        else
14:            A[i][i] ← c_1(j, r_i, dr, dt)
15:            A[i][i − n] ← c_2(j, r_i, dr, dt)
16:            A[i][i + n] ← c_3(j, r_i, dr, dt)
17:            if k = n then
18:                A[i][i − n + 1] ← c_4(j, r_i, dr, dt)
19:            else
20:                A[i][i + 1] ← c_4(j, r_i, dr, dt)
21:            end if
22:            if k = 1 then
23:                A[i][i + n − 1] ← c_4(j, r_i, dr, dt)
24:            else
25:                A[i][i − 1] ← c_4(j, r_i, dr, dt)
26:            end if
27:            k ← k + 1
28:            if k > n then
29:                j ← j + 1, k ← 1
30:            end if
31:        end if
32:    end for
33:    return solve(A, b)
34: end function
```

```
1: function BackSub(Matrix A, Matrix b)
2:     Matrix R ∈ R^{cols(A)×1}
3:     for i ∈ [A.n, 1] do
4:         if a_{i,i} ≠ 0 then
5:             r_{i,1} ← (b_{i,1} − A[i] × R)/a_{i,i}
6:         end if
7:     end for
8:     return R
9: end function
```

```
1: function ForwSub(Matrix A, Matrix b)
2:     Matrix R ∈ R^{cols(A)×1}
3:     for i ∈ [1, A.n] do
4:         if a_{i,i} ≠ 0 then
5:             r_{i,1} ← (b_{i,1} − A[i] × R)/a_{i,i}
6:         end if
7:     end for
8:     return R
9: end function
```

WithFifteenThetas es la función principal encargada de la resolución de una instancia del problema. Consideramos con fines prácticos que ya recibe los datos procesados del input por los parámetros de radio interno (r_i), radio externo (r_e), cantidad de ángulos (n), cantidad de radios (m), un vector de temperaturas internas (T_i) y un vector de temperaturas externas (T_e).

Entre las lineas 2 a 9 se inicializan los valores de Δr (dr) y $\Delta\theta$ (dt) y se carga el vector columna b con los valores provenientes de T_i y T_e, quedando en los n primeros lugares los valores de T_i y en los n últimos los valores de T_e. En la linea 9 se puede observar la inicialización de los contadores j y k. Entre las lineas 10 a 32 tenemos el bucle responsable de llenar la matriz A. Como explicamos en la sección anterior, la idea es moverse por la diagonal y si se encuentra en un área identidad asignar un 1 (lineas 11 y 12). De lo contrario, asignaremos el coeficiente c_1 a la posición de la diagonal y c_2, c_3 y c_4 a las columnas correspondientes de esa misma fila. Los coeficientes son calculados como se los muestra en las formulas (6), (7), (8) y (9).
Finalmente en la linea 33 se llama a la función solve que resuelve el sistema lineal. En nuestra implementación en ese momento se llama a solveGE (resolución por eliminación Gaussiana) o solveLU (resolución por factorización LU) dependiendo del modo en el que se este usando el programa. Para simplicidad del pseudocódigo decidimos dejar fuera dicho detalle ya que pertenece mas a la implementación. El llenado de la matriz A tiene una complejidad de $\mathcal{O}(nm)$.

BackSub (backSubstitution) y **ForwSub** (forwardSubstitution) realizan la resolución de un sistema con una matriz A diagonal superior y diagonal inferior respectivamente. En **BackSub** en la linea 2 se inicializa con ceros el vector columna en donde se devolverá el resultado. Luego entre las lineas 3 a 7, se procede a iterar moviéndose por la diagonal. Si se encuentra un cero se saltea esa fila (linea 4), ya que de haberse encontrado toda la fila es cero. De lo contrario en la linea 5 se multiplica la fila i de A ($A[i]$) por el vector resultado (R), se lo resta a el dato correspondiente a esa ecuación y se lo divide por el coeficiente de la incógnita que se quiere despejar.
Si nos detenemos a observar esta última operación, podremos ver que lo que se esta haciendo es encontrar el valor de x_i despejando la ecuación $a_{i,i}x_i + A[i] \times R = b_{i,1}$ en donde $A[i] \times R$ es la suma de los x_i conocidos por sus constantes. En la linea 8 el algoritmo devuelve el resultado, que una vez terminadas las iteraciones contendrá los valores que resuelven el sistema ordenados de forma $x_1, x_2, ..., x_{cols(A)}$. **BackSub** tiene una complejidad de $\mathcal{O}(n^2)$.

ForwSub tiene un comportamiento isomorfo a **BackSub**, la única diferencia es que comienza despejando la primera ecuación en lugar de la última y sigue desde allí hasta la última.

```
1: function SolveGE(Matrix A, Matrix b)
2:     for i ∈ [1, A.m) do
3:         for z ∈ [i + 1, A.m] do
4:             A[z] ← A[z] − (a_{z,i}/a_{i,i}) · A[i]
5:             b[z] ← (−a_{z,i}/a_{i,i}) · b[i]
6:         end for
7:     end for
8:     return backSub(A, b)
9: end function
```

```
1: function SolveLU(Matrix A, Matrix b)
2:     Matrix LU ← decompLU(A)
3:     Matrix y ← forwSub(LU.first, b)
4:     return backSub(LU.second, y)
5: end function
```

```
1: function DecompLU(Matrix A)
2:     Matrix L ∈ R^{nm×nm}
3:     Matrix U ∈ R^{nm×nm}
4:     for i ∈ [1, A.m) do
5:         for z ∈ [i + 1, A.m] do
6:             l_{z,i} ← a_{z,i}/a_{i,i}
7:             U[z] ← A[z] − (a_{z,i}/a_{i,i}) · A[i]
8:         end for
9:         l_{i,i} ← 1
10:    end for
11:    L[A.m][A.m] ← 1
12:    return <L, U>
13: end function
```

SolveGe (solverGaussianElimination) utiliza la eliminación Gausseana para diagonalizar superiormente el sistema y luego lo resuelve utilizando backSubstitution. En nuestro algoritmo, en las lineas 2 y 3 a 6 y 7 se pueden observar los for principales del mismo. El primer for, cuyo contador es i, se encarga de apuntar a la fila que se esta usando para conseguir ceros en las demás. El segundo for, a quien el contador j le pertenece, se encarga de iterar por las filas inferiores a la fila i.
En cada iteración se le resta a la fila z un múltiplo de la fila i, de forma tal que se origine un cero en la posición (z, i) (linea 4). En la linea 5 se repite dicha operación con el vector columna b, de forma tal de conservar la concordancia con el sistema original. Una vez terminada la iteración completa del segundo for, los únicos lugares en donde habrán valores distintos de cero en la columna i, será en arriba de la diagonal. Finalmente con la matriz diagonalizada superiormente se utiliza el algoritmo de backSub para devolver una resolución al sistema. La complejidad de la eliminación Gausseana es de $\mathcal{O}(n^3 + n^2) \subseteq \mathcal{O}(n^3)$.

SolveLU utiliza la factorización LU para separar el sistema original en dos sistemas que juntos son equivalentes al original. Al separarlos, una de de las matrices queda diagonalizada de forma inferior (L) y la otra de forma superior (U). Luego se puede resolver el sistema utilizando forwardSubstitution para el sistema inferior y backSubstitution para el sistema superior. La ventaja de este método es que el vector columna b no es modificado en la diagonalización y por lo tanto podremos guardarnos las matrices L y U para resolver sistemas con un b distinto ahorrándonos el costo de la diagonalización.

La función **DecompLU** sirve para descomponer una matriz A en dos matrices L y U de forma tal que se cumpla que $L \cdot U = A$. Es básicamente una eliminación Gausseana que guarda en la matriz L los coeficientes que se van utilizando en cada paso (linea 6), además de colocarle en la diagonal todos unos (linea 9 y 11). Una vez terminada la diagonalización devuelve una tupla con ambas matrices.

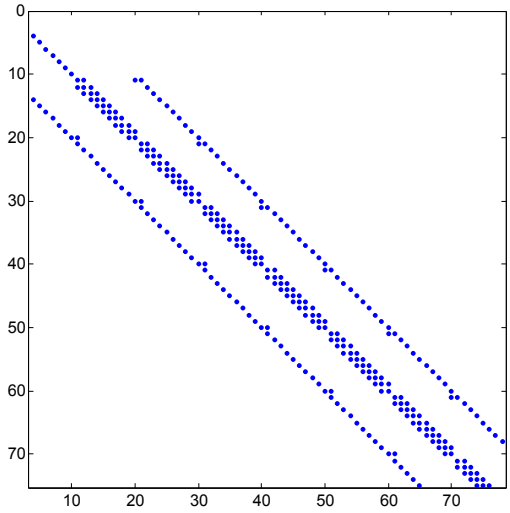


Figura 1: Realizando un spy en Matlab luego de llenar la matriz con el algoritmo propuesto anteriormente pudimos visualizar que su llenado era correcto. En la imagen se pueden observar aproximadamente las primeras 70 filas y columnas de la matriz, en las que se aprecia la identidad en las primeras filas y luego la forma característica que debería tener por las coordenadas polares involucradas y sus saltos de columnas al restarse o sumarse a alguna de ellas.

En nuestra implementación este algoritmo funciona modificando solamente A con el objetivo de ahorrar memoria. Esto se logra eliminando las lineas 11 y 9, haciendo las asignaciones a A en vez de a L y U en las lineas 6 y 7 y devolviendo A en vez de la tupla. Obviamente esta última modificación repercute también en la función **SolveLU**, en la cual hacemos modificaciones en los algoritmos de **ForwSub** y **BackSub** para que solamente se muevan por ciertos índices de A en cada caso, a forma de calcular correctamente la resolución de cada sistema sin la necesidad de utilizar dos matrices distintas. El motivo por el cual presentamos esta versión mas básica es porque es mas elegante y fácil de entender que su versión optimizada.

La complejidad de **DecompLU** es de $\mathcal{O}(n^3)$, lo que desemboca en una complejidad de $\mathcal{O}(n^3 + 2n^2) \subseteq \mathcal{O}(n^3)$ para **SolveLU**. Sin embargo, es importante remarcar que si hacemos resoluciones del mismo sistema con la factorización LU solamente deberemos resolver los sistemas diagonalizados, por lo que el costo de **SolveLU** se reduce a $\mathcal{O}(2n^2) \subseteq \mathcal{O}(n^2)$ para las resoluciones sucesivas a la primera. Finalmente podemos ver que **WithFifteenThetas** tiene una complejidad total de $\mathcal{O}(nm + (nm)^3) \subseteq \mathcal{O}((nm)^3)$ con **SolveGE** y con la primera resolución de **SolveLU**. Sin embargo, las sucesivas resoluciones con **SolveLU** del mismo sistema tienen una complejidad de $\mathcal{O}((nm)^2)$.

```
1: function getIsoterm(Vector T, n, m, iso)
2:     Vector res ∈ R^n
3:     for k ∈ [0, n) do
4:         int best ← 0
5:         for j ∈ [0, m) do
6:             best ← j
7:             if T[j * n + k][0] < iso then
8:                 break
9:             end if
10:        end for
11:        double dr ← (r_e − r_i)/(m − 1)
12:        if best ≥ m then
13:            res_k ← r_e
14:            continue
15:        end if
16:        r_2 ← best
17:        r_1 ← best − 1
18:        t_1 = T[r_1 * n + k][0]
19:        t_2 = T[r_2 * n + k][0]
20:        res_k ← (isovalue − t_1) * (r_2 − r_1)/(t_2 − t_1) + r_1
21:        res_k ← res_k * dr
22:        res_k ← res_k + r_i
23:    end for
24:    return res
25: end function
```

Finalmente, **getIsoterm** es la función que calcula la curva isoterma. Toma como argumentos la matriz de temperaturas, la cantidad de ángulos y radios, y el valor de temperatura buscado. Luego devuelve un vector con los radios por los que pasa la curva para cada ángulo θ. En la linea 2 se encuentra el ciclo principal, que se encarga de recorrer los ángulos, para poder calcular el radio por el cual pasa la isoterma en cada ángulo por separado. El ciclo de la linea 5 se encarga de recorrer radialmente en dicho ángulo hasta que la temperatura sea menor que la isoterma buscada (aquí se asume que avanzando radialmente, las temperaturas solo pueden disminuir), con lo cual el radio buscado esta entre este radio, el j-ésimo y el anterior. Estos valores luego son guardados en la variable 'r_2' y 'r_1', y sus temperaturas en t_2 y t_1 respectivamente. Para interpolar se usa la inversa de una recta, que pasa por los puntos (r_1, t_1) y ($r2$, $t2$). Dicha recta, dentro de su dominio $[r_1, r_2]$, trabajaría como una función interpoladora que devuelve la temperatura, dado un radio. Pero al usar su inversa, sucede que al dar el valor de una temperatura (que en este caso sera siempre el de la isoterma), devuelve el radio del que provino. Este valor es asignado a res_k y luego multiplicado por delta r y adicionado a r_i para que represente el valor del radio real. Aquí termina el ciclo principal y se realiza el calculo nuevamente para el siguiente ángulo, hasta recorrerlos todos. La función retorna el vector de radios de la isoterma por el vector 'res'.

3. Experimentación

3.1. Instancias propuestas

En la siguiente sección analizaremos tres instancias distintas del sistema.
Estas consisten en:

- Un horno muy frío (temperatura interior: 1500, temperatura exterior: 30)
- Un horno muy caliente (temperatura interior: 1500, temperatura exterior: 480)
- Un horno averiado, que para la mitad superior esté muy caliente y para la mitad inferior muy frío

Para todos los casos, la isoterma buscada es de 500 grados.
La utilidad de estos tres casos es para mostrar que el tiempo de resolución del sistema no depende de los valores del mismo, sino de cómo se discretiza el horno. El tercer caso nos resulta interesante, además, para ver cómo varía la isoterma al tener tan marcada la diferencia de temperatura.

3.2. Análisis de isoterma

En la figura **??** podemos ver un gráfico de los hornos y su respectiva isoterma. Previamente explicamos cómo se calcula esta isoterma y en los gráficos podemos ver esto reflejado. Para el horno caliente, podemos observar que la isoterma está cerca de las paredes. Esto es porque las paredes externas tienen una temperatura de 480. Para el horno chico, vemos que la isoterma se encuentra más bien lejos de las paredes externas, pues estas se encuentran frías.

Figura 2: Izquierda: Horno Caliente, Derecha: Horno Frío

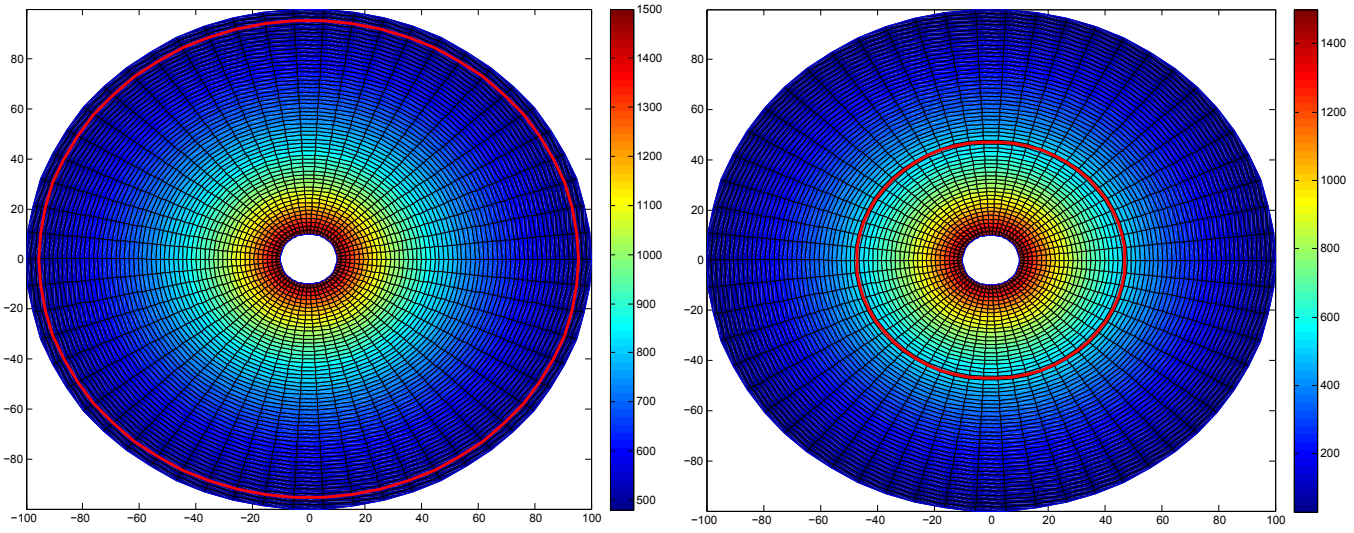
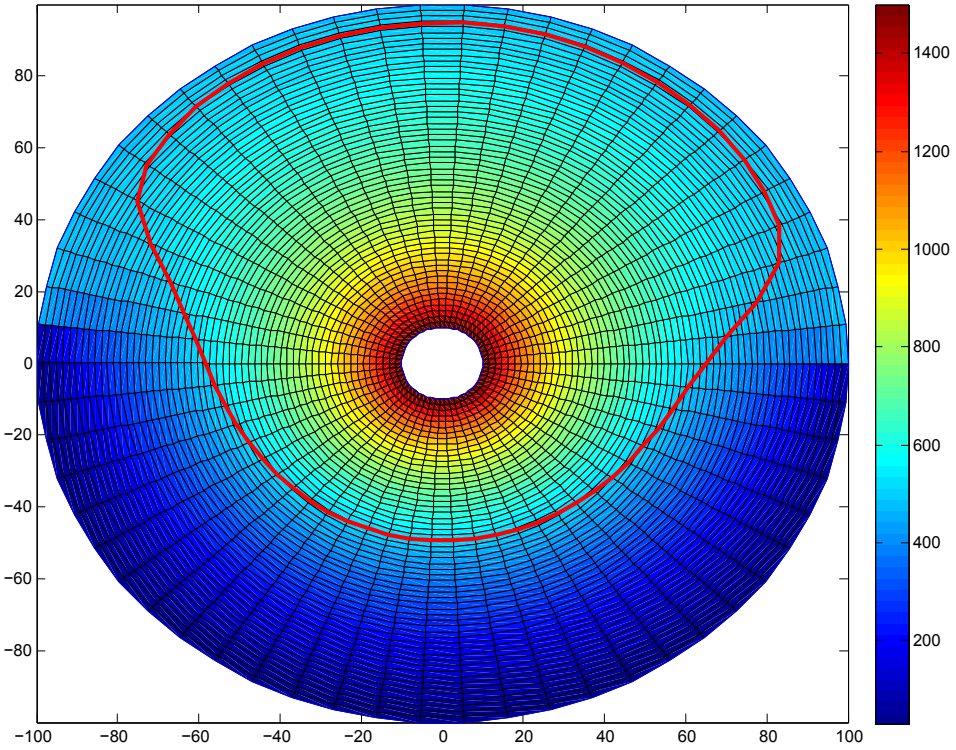


Figura 3: Horno con parte inferior caliente y parte superior fría



Un caso más interesante es el de la figura **??**, donde la temperatura no es igual en todos los ángulos medidos. En el gráfico del horno se puede ver cómo la parte superior está más caliente que la inferior, y cómo la isoterma se va adaptando, moviendose entre casillas de colores muy similares, en la escala de la derecha podemos ver que el color por donde pasa la isoterma está muy cerca del color de la temperatura correspondiente a 500.

3.3. Análisis de performance

3.3.1. Medición de ciclos de procesador

Para realizar las mediciones y comparaciones, en vez de medir los tiempos de inicio y final, utilizamos un método sugerido por Intel [Intel-1][1], contando los ciclos de procesador utilizando la instrucción de procesador **RDTSC**. Tuvimos en cuenta las siguientes cosas:

- Todas las mediciones fueron realizadas 10 veces, y luego se promediaron los valores
- Se corrieron los experimentos en un sistema multicore
- En vez de realizar una llamada a función, utilizamos una macro que usa inline-assembly para diminuir el overhead de realizar la medición en sí.
- Creamos una barrera de ejecución con la instrucción lfence para evitar que el procesador ejecutara instrucciones fuera de orden (es decir, para evitar que el procesador ejecutara rdtsc antes de haber terminado de ejecutar el programa).

Consideramos que todos esos detalles nos brindan el nivel de precisión necesario para las mediciones del problema.

En un sistema multiproceso como Linux es posible que no todos los ciclos de procesador sean destinados a nuestro proceso, tratamos de mitigar eso realizando los experimentos varias veces y luego promediando los ciclos insumidos, también ayuda el hecho de ejecutar el programa en un sistema multicore.

Decidimos utilizar una macro inline assembly y rdtsc ya que hay un overhead significante en llamar a funciones, y obtener el tiempo. Es posible que al obtener el tiempo actual se realize una **syscall** y esto desaloje nuestro proceso[2]. Es cierto que para instancias grandes del problema, una diferencia de medio segundo es casi despreciable, pero consideramos que realizar las mediciones de esta forma disminuyen el error sin mucho esfuerzo.

Finalmente, la forma en la que utilizamos todo esto es: El programa cuenta los ciclos de reloj antes y despues de resolver el sistema (llamando a la macro que se puede encontrar en el archivo **tiempo.h**), los ciclos insumidos es el valor absoluto de la diferencia entre ciclos iniciales y finales. [3].

3.3.2. Benchmarks

Realizamos una serie de pruebas para analizar la performance de la resolución del sistema. Hicimos las pruebas en las tres instancias descriptas previamente. Realizamos los tests variando la cantidad de puntos de discretización que utilizamos, variando el n (cantidad de ángulos o sensores), el m (cantidad de radios) y n y m al mismo tiempo. Medimos los resultados contando los ciclos de procesador insumidos. Por cada test corrimos 10 repeticiones del mismo y luego promediamos los resultados, a modo de amortizar los ciclos de más por ser desalojados del scheduler.

Esperamos que cada uno de los distintos escenarios insuman cantidades muy cercanas de ciclos en su resolución, ya que los valores de los datos deberían ser irrelevantes para el tiempo de procesamiento. Además, el tiempo de procesamiento debería seguir un comportamiento cúbico en función de la entrada, ya que es esa es la complejidad del algoritmo utilizando eliminación Gausseana.

[1] El paper sugiere más técnicas además de las que utilizamos nosotros para mejorar el monitoreo de performance: Correr el programa en un módulo de kernel sin desalojo y medir cuánto overhead tiene la llamada a rdtsc. Eso es útil cuando se desea una granularidad extrema de menos de 1000 cicos de procesador, nosotros no necesitamos tanta granularidad

[2] A menos que el kernel implemente vDSO para la función de obtener el tiempo.

[3] notar que dado que usamos precisión de 64 bits, sólo es posible que haya overflow si la computadora estuvo prendida por más de 500 años

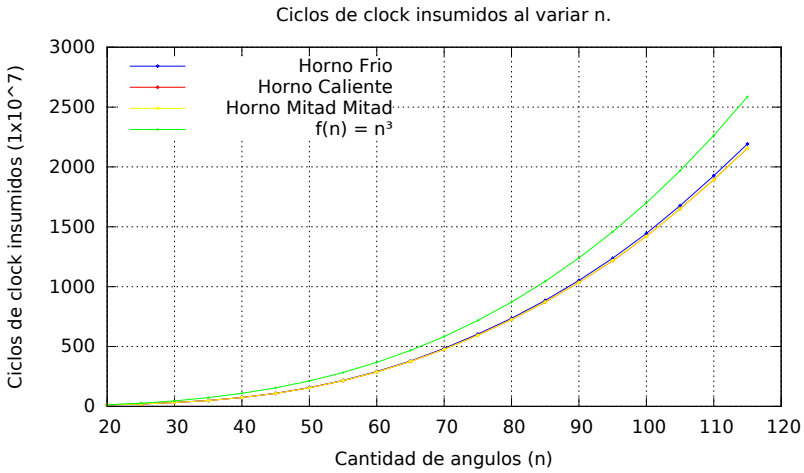


Figura 4: Se pueden observar los resultados de las primeras tres pruebas. Variamos n (cantidad de angulos o sensores) en los tres escenarios que presentamos anteriormente utilizando eliminación Gausseana para resolverlos. Como se puede visualizar, el insumo de ciclos es cúbico en función de la cantidad de ángulos y la resolución de los tres escenarios insumió una cantidad muy cercana de ciclos, que es lo que estabamos esperando.

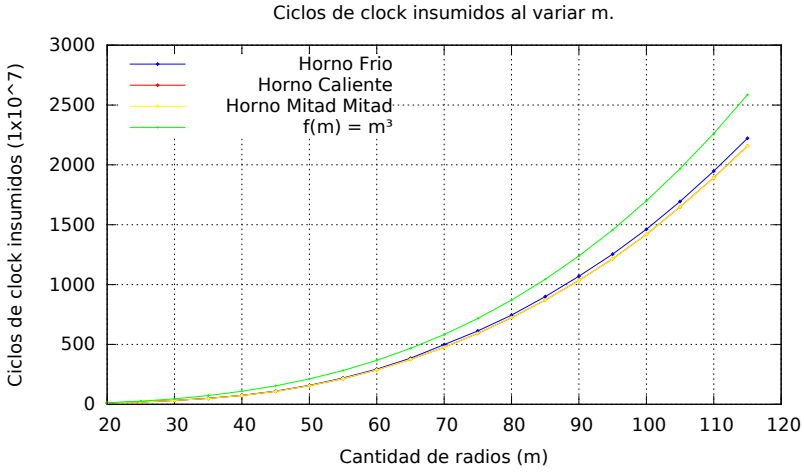


Figura 5: Al igual que la figura anterior, se puede observar exactamente el mismo comportamiento cuando solamente variamos m: un insumo de ciclos cúbico e insumos cercanos para los tres escenarios.

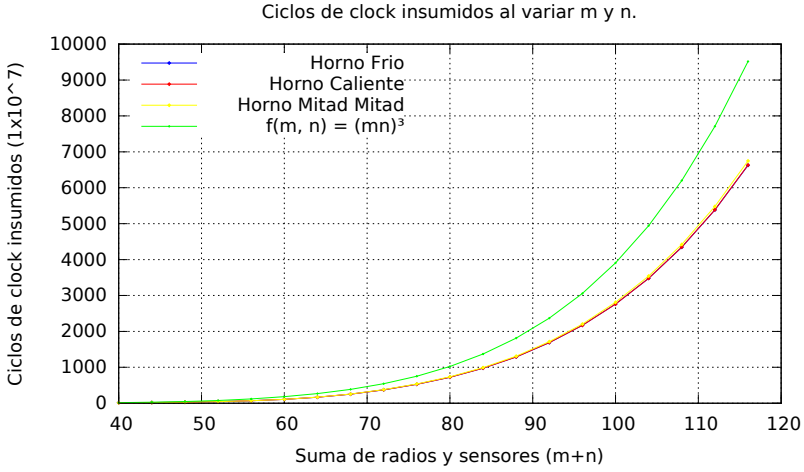


Figura 6: Esta figura, si bien sigue la misma tendencia que las anteriores, es un poco distinta. Debido a la variación de ambos parametros la curva es mucho más cócava que en las experimentaciones anteriores. Sin embargo, los tres escenarios siguen insumiendo una cantidad similar de ciclos en cada instancia, que es lo que esperabamos ver.

3.4. Factorización LU vs Eliminación Gaussiana

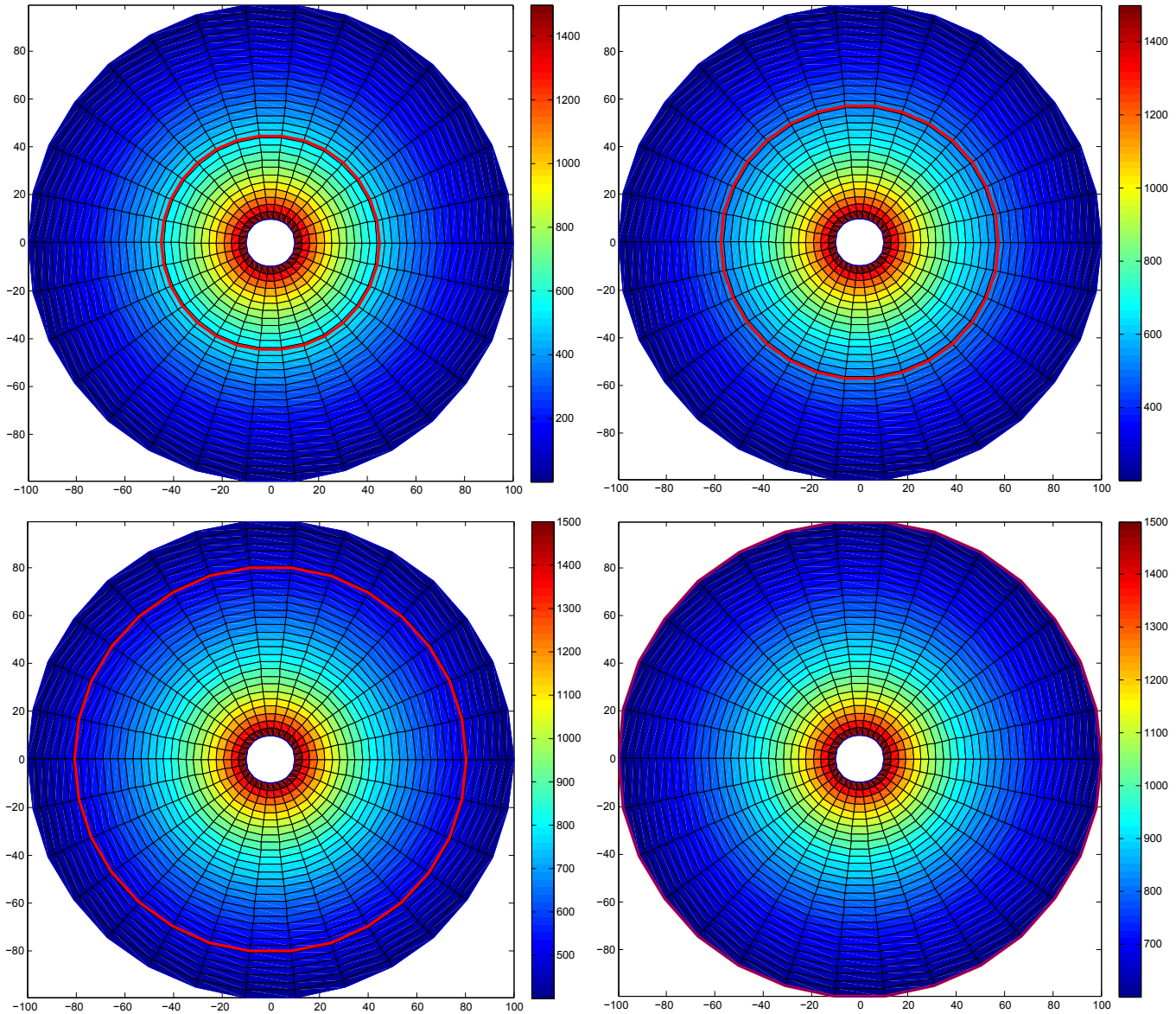
En esta sección compararemos la performance de los dos métodos de resolución utilizados. Dado que LU factoriza una sola vez la matriz, resolver el mismo sistema cambiando solo las mediciones implica un menor costo que resolver todo el sistema nuevamente (cosa que sucede con el método de eliminación Gaussiana). Basándonos en este razonamiento, esperamos conseguir una mejora de performance por parte del método LU cuando hay que resolver varias instancias del mismo sistema.

Proponemos dos experimentos, el primero consiste calcular la isoterma para muchas instancias del mismo horno, pero variando la temperatura de los sensores, simulando un efecto de 'paso del tiempo', en donde las paredes externas

del horno comienzan a calentarse desde un estado inicial con temperatura 0. La evolución del estado del horno se puede apreciar en la figura **??**. En la figura **??** mostramos la comparativa de ciclos insumidos por LU y Gauss, confirmando que conviene realizar LU cuando queremos analizar más de una instancia. Sin embargo, la cantidad de ciclos insumidos para la primer iteración es similar: el método LU debe factorizar la matriz y eso lleva casi lo mismo que el método de eliminiación gaussiana.

El segundo experimento trata de mostrar cómo varían la performance en función del tamaño del sistema y de las iteraciones. Simulamos varios sistemas como el anterior (un horno calentándose), pero en cada sistema aumentamos la cantidad de radios de la discretización y la cantidad de instancias. Con esto esperamos poder mostrar que luego de resolver la primer instancia, el proceso de resolución de las instancias siguientes es mucho más rápido con LU que con Gauss. Esto se puede ver en la figura **??**.

Figura 7: Evolución del horno en función del tiempo. Se ve cómo la isoterma de 500 grados se acerca hacia el borde del horno, hasta que lo supera.



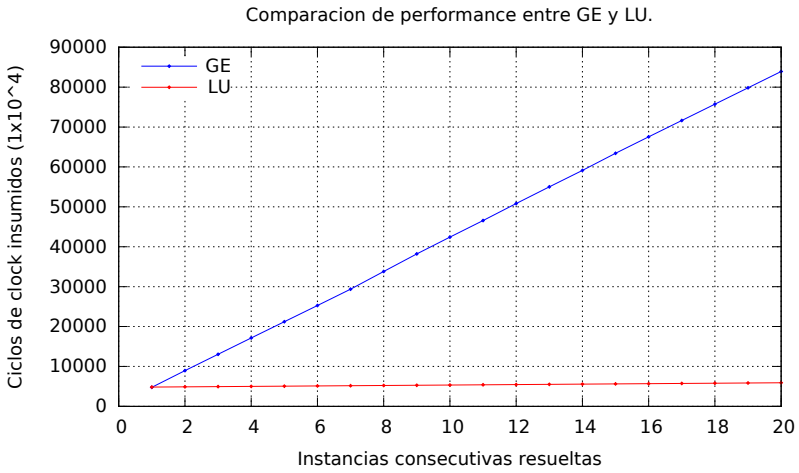


Figura 8: Se puede observar la comparación de performance entre eliminación Gausseana (GE) y factorización LU con 20 instancias seguidas del mismo sistema. Como se visualiza, en la primera resolución del sistema el insumo de ciclos es casi el mismo, mientras que en las consecuentes la factorización LU toma mucha ventaja con respecto a la eliminación Gausseana. Los resultados muestran un crecimiento lineal en función de la cantidad de instancias, ya que al haber fijado el n y el m el insumo de ciclos en la resolución de una instancia es constante.

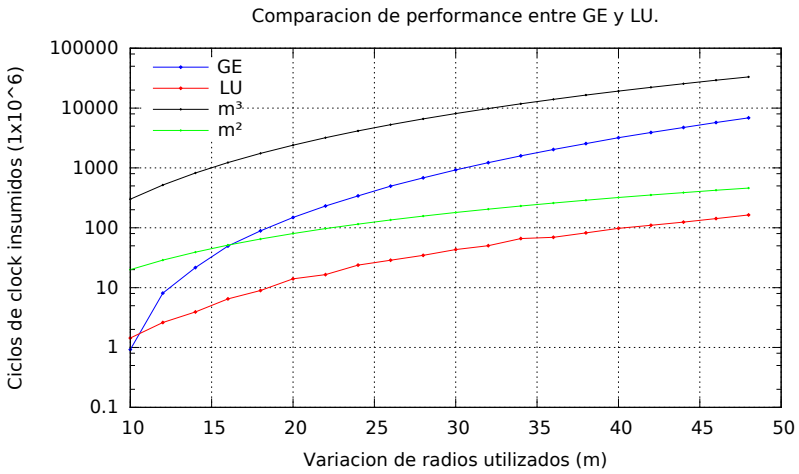


Figura 9: En esta última experimentación intentamos realizar una comparativa que muestre mas claramente la diferencia de complejidad entre eliminación Gausseana y factorización LU. Para ello realizamos 20 iteraciones en las cuales $n = 10$ para todas y m era incrementado de a dos unidades en cada iteración (con un valor inicial de 10). A la par de los incrementos de m era incrementado $ninst$ de forma tal de que en las primeras iteraciones se pudiese observar con el algoritmo de LU un comportamiento mas cercano a una complejidad de $\mathcal{O}(n^3)$, mientras que en las ultimas iteraciones donde $ninst > 100$ el factor n^2 tomaría mas importancia que la primera pero única resolución con complejidad n^3.

4. Demostración

Proposición: Sea $A \in \mathbb{R}^{nxn}$ la matriz obtenida para el sistema definido por $(6)-(9)$. Es posible aplicar eliminación Gaussiana sin pivoteo.

4.1. Aclaraciones

En las siguientes demostraciones, se usará la notación $A^{(k)}$ para referirse a la matriz resultante de aplicar k pasos de eliminación gaussiana en A, para algún $k \in \mathbb{N}$

4.2. Demostración de la propiedad

Lema 0: Sea $A \in \mathbb{R}^{nxn}$ una matriz cuyas filas son *linealmente independientes* (LI), no existe una columna de A con elementos todos nulos.

Lema 1: Sea $A \in \mathbb{R}^{nxn}$ una matriz cuyas filas son LI, vale que $\forall i, j \in [1, n], i \neq j$ la matriz resultante de restar un múltiplo cualquiera de la fila i a la fila j tambien es LI.

Lema 2: Sea $A \in \mathbb{R}^{nxn}$ una matriz DDNE cuyas filas son LI, se puede hacer un paso de eliminación Gaussiana sin pivote.

Lema 3: Sea $A \in \mathbb{R}^{nxn}$ una matriz DDNE cuyas filas son LI, la submatriz resultante de aplicar un paso de la eliminación Gaussiana es DDNE.

Demostración: Sea A una matriz definida por $(6) - (9)$. Vale que A es una matriz cuadrada que es *diagonal dominante no estricta* (DDNE) cuyas filas son LI.

Luego, por el **Lema 0**, **Lema 1** y el **Lema 2** puedo aplicar un paso de la eliminación gaussiana conservando la independencia lineal. Por el **Lema 3** vale que la matriz resultante sigue siendo DDNE y LI, cumpliendo de nuevo las hipótesis originales. Aplicando este paso n veces, se obtiene la matriz $A^{(n)}$ sin pivotear. ■

4.2.1. Demostración del Lema 0

Sea $A \in \mathbb{R}^n xn$ una matriz cuyas filas son linealmente independientes.

Asumamos que A tiene una columna de elementos todos nulos. Entonces, A^t tiene una fila de elementos todos nulos.

Por lo tanto $det(A^t) = 0$ [Hefferon-1]$\Rightarrow$ $det(A) = 0$ [Jeronimo-1]$\Rightarrow$ Las filas de A son linealmente dependientes[Jeronimo-2].

El absurdo provino de suponer que la matriz A, teniendo filas linealmente independientes, puede tener una columna de elementos todos nulos.

4.2.2. Demostración del Lema 1

Sea $A \in \mathbb{R}^{nxn}$. Que sus filas sean LI implica que:

$$\lambda_1 f_1 + ... + \lambda_n f_n = 0 \Leftrightarrow \lambda_p = 0, \forall p \in [1..n] \tag{10}$$

Supongamos ahora que realizo la operación de restarle a la fila j-ésima de A, un múltiplo de la fila i-ésima de A. Una combinación lineal de las filas de la matriz resultante, llamémosla A', podría escribirse en función de las filas originales como:

$$\lambda_1 f_1 + ... + \lambda_i f_i + ... + \lambda_j f_j - k\lambda_i f_i + ... + \lambda_n f_n = \tag{11}$$

$$\lambda_1 f_1 + ... + (\lambda_i - k) f_i + ... + \lambda_j f_j + ... + \lambda_n f_n = \tag{12}$$

Si ahora tomamos $\lambda'_i = \lambda_i - k$ como coeficiente, podemos obtener la siguiente expresión, que es equivalente a la que obtuvimos originalmente de los vectores de A.

$$\lambda_1 f_1 + ... + \lambda'_i f_i + ... + \lambda_n f_n = \Leftrightarrow \lambda' = 0 \wedge \lambda_i = 0 \forall \lambda \in [1..n] \tag{13}$$

Con lo cual si las filas de A son linealmente independientes, las de A' también lo son.

4.2.3. Demostración del Lema 2

Si A es DDNE, entonces $\forall i, j \in [1, n], |a_{jj}| \geq |a_{ij}|$.

Como la matriz tiene filas LI y es cuadrada, por **Lema 0**, no existe una columna de elementos todos nulos.

Luego, $\forall j \in [1, n] \exists i_0 \in [1, n] / |a_{i_0 j}| > 0$, finalmente, como es DDNE $\forall i \in [1, n], |a_{ii}| \geq |a_{i_0 j}| > 0$.

Esto significa que los elementos de la diagonal son no nulos, y, por lo tanto, se puede realizar un paso de la eliminación Gaussiana sin necesidad de pivotear.

4.2.4. Demostración del Lema 3

Quiero ver que al hacer eliminación Gaussiana, la submatriz resultante no pierde su propiedad de DDNE. Dada una matriz $A^{(0)} \in \mathbb{R}^{nxn}$ y que se realizó un paso de la factorización de Gauss, siendo la matriz resultante nombrada $A^{(1)}$. En tal caso, un elemento de A^1 se escribe:

$$|a_{ij}^{(1)}| = |a_{ij}^{(0)} - \frac{a_{i1}^{(0)}}{a_{11}^{(0)}} a_{1j}^{(0)}|$$

Y la propiedad que queremos probar se escribe:

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

De ahora en mas se evitara usar los supraindices por cuestiones de simplicidad. Reemplazando (10) en (11) vale que

$$\sum_{i=2, i \neq j}^{n} |a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j}| \leq$$

$$\sum_{i=2, i \neq j}^{n} |a_{ij}| + \sum_{i=2, i \neq j}^{n} |\frac{a_{i1}}{a_{11}} a_{1j}| =$$

$$-|a_{1j}| + \sum_{i=1, i \neq j}^{n} |a_{ij}| + \sum_{i=2, i \neq j}^{n} |\frac{a_{i1}}{a_{11}} a_{1j}|$$

Pero como $A^{(0)}$ es DDNE

$$\sum_{i=1, i \neq j}^{n} |a_{ij}| \leq |a_{jj}| \Rightarrow (18) \leq -|a_{1j}| + |a_{jj}| + \sum_{i=2, i \neq j}^{n} |\frac{a_{i1}}{a_{11}} a_{1j}| =$$

$$|-a_{1j}| + |a_{jj}| + |\frac{a_{1i}}{a_{11}}| \sum_{i=2, i \neq j}^{n} |a_{i1}|$$

Otra vez, usando que $A^{(0)}$ es DDNE

$$\sum_{i=2}^{n} |a_{i1}| \leq |a_{11}| \Rightarrow$$

$$\sum_{i=2, i\neq j}^{n} |a_{i1}| + |a_{j1}| \leq |a_{11}|$$

$$\sum_{i=2, i\neq j}^{n} |a_{i1}| \leq |a_{11}| - |a_{j1}|$$

Esto implica que

$$(20) \leq -|a_{ij}| + |a_{jj}| + |\frac{a_{1j}}{a_{11}}|(|a_{11}| - |a_{j1}|)$$

$$= |a_{jj}| - |\frac{a_{1j}}{a_{11}}||a_{j1}| \leq |a_{jj} - \frac{a_{1j}}{a11}a_{j1}| = |a_{jj}^{(1)}|$$

5. Conclusiones

Mediante los resultados obtenidos por la experimentación, pudimos concluir que la factorización LU es igual (para los casos de una instancia) o más performante (para los casos de mas de una instancia) que la eliminación Gausseana.

El análisis de performance deja en evidencia que efectivamente la eliminación Gausseana tiene una complejidad de $\mathcal{O}((nm)^3)$ para toda iteración mientras que la factorización LU tiene una complejidad de $\mathcal{O}((nm)^3)$ para la primera resolución mientras que para sucesivas resoluciones tiene una complejidad de $\mathcal{O}((nm)^2)$.

En un sistema de control de temperaturas real, en donde tendríamos que dar una estimación a cada instante de la temperatura del horno y del calculo de la isoterma para poder prevenir accidentes, la factorización LU supone una mejor manera de hacerlo. ya que como el sistema no cambiara la cantidad de sensores (ya que son piezas de hardware y demandaría una reinstalación de los mismos) tampoco lo hará la matriz asociada al sistema. Es por ello que hasta podríamos hacer el calculo de la matriz de antemano, de alguna forma hardcodearlo para que todo funcione aun mas rápido y luego resolver el sistema y dar las estimaciones cuando los datos de los sensores lleguen.

6. Apéndice A: Implementación

6.1. Implementación en C++

Proveemos un programa de computadora para resolver instancias de este problema, mediante distintos métodos, a saber, Eliminación gaussiana y Factorización LU. El código fuente del programa se encuentra adjunto a este documento.

El código está en lenguaje C++11 con una extensión en Assembler de la arquitectura Intel x86-64 para contar los ciclos de procesador insumidos, esto fue utilizado en la sección Experimentación y allí se encuentra una explicación.

Para generar el binario se necesita:

- GNU Make para compilar y generar el binario
- El compilador g++ de GNU, versión 4.6.3 (Se puede utilizar otro compilador, pero este debe soportar el estándar de C++11 y soporte inline-assembly con sintaxis Intel.

Utilizando GNU Make se puede generar un binario compatible con arquitecturas Intel x86-64. Para ello, hay que ejecutar:

```
make main
```

Esto generará un archivo ejecutable llamado **tp**, para ejecutarlo, se deben brindar los siguientes parámetros:

- **archivo_in** el nombre de un archivo que describa un sistema con un formato parseable, el mismo será descripto más adelante (formato entrada).
- **archivo_out** nombre del archivo donde se desee guardar el sistema resuelto, con un formato que también será descripto más adelante (formato salida)
- **modo** Modo de resolución 0 para Gauss, 1 para LU
- **archivo_iso** (opcional) nombre del archivo en donde se desee guardar la información de la isoterma, con un formato que será descripto más adelante (formato isoterma)

Por ejemplo, se puede ejecutar el binario como:

```
./tp archivo.in archivo.out 1 archivo.isoterma
```

Donde archivo.in es un archivo existente y archivo.out y archivo.isoterma pueden no existir y serán sobreescritos.

6.1.1. Formatos de Archivos

La especificación del formato de entrada y formato salida se encuentra en el Apéndice B - Enunciado. La especificación del formato de la isoterma es: para un horno de n sensores y k instancias, un archivo con $k * n$ líneas, con un número real en cada línea, donde las línea i contiene el radio de la isoterma para la instancia $[\frac{i}{n}]$ del sensor $i\,\%n$ (que representa 'el resto de dividir a i por n).

6.1.2. Scripts para MATLAB y Python

También proveemos scripts para resolver el sistema en MATLAB y Python. Estos scripts fueron utilizados para prototipar el programa en C++ y encotrar errores en las operaciones. Los mismos se encuentran dentro de la carpeta 'exp' del directorio de archivos provisto. Para ejecutar estos archivos es necesario poseer los programas MATLAB y Python 2.7 (también la librería scipy).

6.1.3. Generación de Experimentos

Proveemos también los scripts utilizados para generar y correr los experimentos. Estos se encuentran dentro de la carpeta 'experimentacion'. Se necesita poseer bash alguna shell compatible y el programa Go, compilador oficial del lenguaje de programación Go (algunos de los programas fueron hechos en este lenguaje). Es necesario compilar el tp y los programas y ponerlos en las carpetas donde están los scripts.

7. Bibliografía

[Intel-1]:
titulo: How to Benchmark Code Execution Times
on Intel IA-32 and IA-64 Instruction Set Architectures
url:
http://www.intel.com/content/dam/www/public/
us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf

[Jeronimo-1]:
titulo: Algebra Lineal
autores:Gabriela Jeronimo, Juan Sabia y Susana Tesauri
ISSN: 1851-1317
Capítulo 5, pág 115.

[Jeronimo-2]:
titulo: Algebra Lineal
autores:Gabriela Jeronimo, Juan Sabia y Susana Tesauri
ISSN: 1851-1317
Capítulo 5, pág 119.

[Hefferon-1]:
titulo: Linear Algebra
autor: James Hefferon
url: http://joshua.smcvt.edu/linearalgebra
Capítulo 4, pág 322.