



DEPARTAMENTO
DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

TP I - Métodos Numéricos

Métodos Numéricos

Grupo 2

Integrante	LU	Correo electrónico
Alejandro Danós	381/10	adp007@msn.com
Franco		
Fernando	56/09	yolibertino@gmail.com
Ana Sarriés	144/02	abarloventos@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		



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 Teórica	3
2. Desarrollo	3
2.1. Algoritmos	4
2.2. Tests	6
2.2.1. Test de isoterma en función de la granularidad de la discretización	6
2.2.2. Test de comparación Gauss vs LU	6
2.2.3. Test de isoterma en función de las condiciones de borde	6
2.2.4. Test de tiempo en función de la granularidad de la discretización	6
3. Resultados	7
3.1. Gauss vs LU	7
3.2. Granularidad vs Tiempo	9
4. Discusión	10
4.1. Análisis de comparación Gauss contra LU	10
5. Conclusiones	10
6. Apéndices	10
6.0.1. Demostración de la proposición	11
7. Referencias	12

1. Introducción Teórica

El presente informe se enfoca la resolución de sistemas matriciales del tipo $Ax = b$, donde A es una matriz inversible con coeficientes reales. El método clásico por excelencia para este tipo de problemas es la eliminación gaussiana que básicamente consiste en aplicar operaciones elementales de fila sobre la matriz A y el vector b para poder simplificar el sistema de ecuaciones original, obteniéndose un sistema triangulado que puede ser fácilmente resuelto sustituyendo de manera correcta las incógnitas del sistema. La eliminación gaussiana original puede (y en algunas ocasiones debe) realizar permutaciones de filas, pero como vamos a restringir nuestro estudio a matrices diagonal dominantes (para más información, ver demostración en el apéndice), se omitirá el pivoteo, y cada vez que se haga mención a dicho algoritmo se entenderá que es sin pivoteo.

El otro método estudiado en este informe es la factorización LU, que básicamente consiste en hallar una forma de A que sea igual a $L \cdot U$, donde U es triangular superior y L triangular inferior, de esta manera $Ax = b$ se convierte en $LUx = b$, y si consideramos $y = Ux$, la solución al sistema puede hallarse resolviendo primero $Ly = b$, y luego $Ux = y$. Esta factorización permite evitar tener que triangular la matriz A , cada vez que b es modificado. La desventaja de este método es que la matriz debe poder triangularse usando eliminación gaussiana sin pivoteo, pero afortunadamente en este informe trabajaremos con dichas matrices.

2. Desarrollo

Decidimos pensar al problema como un sistema lineal de ecuaciones o, equivalentemente, buscar el vector x que cumpla $Ax = b$, siendo éstas las siguientes:

- Matriz A : es una matriz cuadrada con cantidad de filas y de columnas igual a $n \times (m + 1)$ está dividida en 3 partes según las filas. Sean i, j tal que $1 \leq i, j \leq (n \times (m + 1))$.

- Caso** $i \leq n$ ó **Caso** $(n \times (m + 1)) - n < i$:

$$A_{ij} = \begin{cases} 1 & \text{si } i = j; \\ 0 & \text{si } i \neq j. \end{cases}$$

- Caso** $n < i \leq (n \times (m + 1)) - n$:

$$A_{ij} = \begin{cases} \frac{-2}{(\Delta r)^2} + \frac{1}{r \times \Delta r} - \frac{2}{r^2 \times (\Delta \theta)^2} & \text{si } i = j; \\ \frac{1}{(\Delta r)^2} - \frac{1}{r \times (\Delta r)} & \text{si } j = i - n; \\ \frac{1}{(\Delta r)^2} & \text{si } j = i + n; \\ \frac{1}{r^2 \times (\Delta \theta)^2} & \text{si } j = i - 1; \\ \frac{1}{r^2 \times (\Delta \theta)^2} & \text{si } j = i + 1; \\ 0 & \text{en otro caso.} \end{cases}$$

- Vector x : es un vector con $n \times (m + 1)$ incógnitas que representarían las temperaturas de los puntos en nuestra pared. Para que sea más fácil el cálculo y que sea consistente con lo propuesto en la matriz A , están ordenados de forma *alfabética* primero según el radio (r) y después según el ángulo (θ). Es decir, X_1 representa a $T(1,1)$, X_p representa a $T(p/n, p \% n)$, X_{n+1} a $T((p+1)/n, (p+1) \% n)$, etc.
- Vector b : es un vector con $n \times (m + 1)$ valores que representan lo que sabemos sobre las temperaturas, es decir, las temperaturas internas y externas y el resultado de las ecuaciones de calor. Los primeros y últimos n valores son las temperaturas internas y externas respectivamente. Los puntos intermedios entre ellos son todos 0s, el resultado de la ecuación de calor para ese punto. De esta forma, queda subdividido en las siguientes partes:
- submatrices inducidas de A en las que hay una matriz **Identidad**, los 2 casos en la primera definición de A_{ij} arriba, se igualaría el respectivo X_i con su temperatura fija. En los puntos de la *submatriz inducida de A* en los que no hay una Matriz **Identidad**, están igualados a 0 para aplicar la ecuación con derivadas con los multiplicadores de las incógnitas debidamente indicados por cada fila.

Menos formalmente, sean $M_{i,j}$, $M_{i,i-n}$, $M_{i,i+n}$, $M_{i,i-1}$ y $M_{i,i+1}$ los multiplicadores en las filas de A “del medio” respectivamente según los enunciamos.

$$\begin{pmatrix} I & \cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots \\ \vdots & \ddots & & \cdots & 0 & \cdots & 0 & \cdots \\ \hline \vdots & \vdots & & \vdots & & \vdots & & \vdots \\ \vdots & M_{i,j-n} & \cdots & M_{i,j-1} & M_{i,j} & M_{i,j+1} & \cdots & M_{i,j+n} & \cdots \\ \vdots & \vdots & & \vdots & & \vdots & & & \vdots \\ \hline \cdots & \cdots & 0 & \cdots & 0 & \cdots & I & \cdots \\ \cdots & \cdots & 0 & \cdots & & \cdots & \vdots & \ddots \end{pmatrix}$$

2.1. Algoritmos

El objetivo principal del presente trabajo es resolver sistemas matriciales de la forma $Ax = b$, para el caso en que A sea una matriz inversible y diagonal dominante. Para poder resolver un sistema de ecuaciones en forma matricial, lo esencial es triangular la matriz para transformar el sistema, en principio complejo, en uno más simple que pueda ser resuelto mediante algún algoritmo sencillo. Los métodos elegidos y estudiados para la triangulación del sistema son el algoritmo de eliminación de Gauss-Jordan sin pivoteo y la factorización LU, mientras que para resolver el sistema triangulado se usaron *backward* y *forward substitution*. A continuación se muestran los pseudocódigos de los algoritmos implementados y la resolución de los sistemas de ecuaciones.

Algoritmo 2.1 gauss(Matriz A , vector b)

```

for  $i = 1$  hasta  $n - 1$  do
  if  $A_{ii} \neq 0$  then
    for  $j = i + 1$  hasta  $n$  do
       $m = A_{ji}/A_{ii}$ 
      for  $k = i$  hasta  $n$  do
         $A_{jk} = A_{jk} - m \cdot A_{ik}$ 
      end for
       $b_j = b_j - m \cdot b_i$ 
    end for
  end if
end for

```

Algoritmo 2.2 LU(Matriz A)

```

for  $i = 1$  hasta  $n$  do
  for  $j = i + 1$  hasta  $n - 1$  do
    if  $A_{ji} \neq 0$  then
       $m = A_{ji}/A_{ii}$ 
      for  $k = i$  hasta  $n$  do
         $A_{jk} = A_{jk} - m \cdot A_{ik}$ 
      end for
       $A_{ji} = m$ 
    end if
  end for
end for

```

Algoritmo 2.3 forwSubst(Matriz A , vector b , vector res , bool lu)

```
if  $lu$  then
  for  $i = 1$  hasta  $n$  do
     $auxVector = A_{ii}$ 
     $A_{ii} = 1$ 
  end for
end if
for  $i = 1$  hasta  $n$  do
   $acum = 0$ 
  for  $j = 1$  hasta  $j < i$  do
     $acum+ = res_j \cdot A_{ij}$ 
  end for
   $res_i = (b_i - acum)/A_{ii}$ 
end for
if  $lu$  then
  for  $i = 1$  hasta  $n$  do
     $A_{ii} = auxVector$ 
  end for
end if
```

Algoritmo 2.4 backSubst(Matriz A , vector b , vector res , bool lu)

```
if  $lu$  then
  for  $i = 1$  hasta  $n$  do
     $auxVector = A_{ii}$ 
     $A_{ii} = 1$ 
  end for
end if
for  $i = n$  hasta  $1$  do
   $acum = 0$ 
  for  $j = n$  hasta  $j > i$  do
     $acum+ = res_j \cdot A_{ij}$ 
  end for
   $res_i = (b_i - acum)/A_{ii}$ 
end for
if  $lu$  then
  for  $i = 1$  hasta  $n$  do
     $A_{ii} = auxVector$ 
  end for
end if
```

Algoritmo 2.5 resolverConGauss(Matriz A , vectores bes , vectores $reses$)

```
for  $i = 1$  hasta  $\#(bes)$  do
  gauss( $A, bes_i$ )
  backSubst( $A, bes_i, reses_i, false$ )
end for
```

Algoritmo 2.6 resolverConLU(Matriz A , vectores bes , vectores $reses$)

```
LU( $A$ )
for  $i = 1$  hasta  $\#(bes)$  do
  backSubst( $A, bes_i, aux, true$ )
  forwSubst( $A, aux, reses_i, true$ )
end for
```

Aclaraciones:

- Como se puede observar en el pseudocódigo hay ciertas optimizaciones que no afectan a la correctitud de los algoritmos.
- El código implementado permite usar pivoteo parcial, pero no será utilizado ni detallado en el informe.

- La igualdad por cero está definida por tolerancia.
- El pseudocódigo presenta abusos de notación y es una mezcla de varios lenguajes de programación y lenguaje natural.
- Algunos algoritmos pueden estar implementados en un mismo método.

2.2. Tests

En esta sección haremos una introducción a los diferentes tests que decidimos hacer en nuestro trabajo.

2.2.1. Test de isoterma en función de la granularidad de la discretización

La siguiente experimentación tiene la intención de analizar el comportamiento de la isoterma en función de la granularidad de la discretización. Tomaremos r_i , r_e , $T_i(\theta_j)$, $T_e(\theta_j)$ e *iso* constantes y apropiados para que se pueda apreciar con mayor claridad el comportamiento, la elección de los mismos fue tomada luego de varias pruebas. La cantidad de ángulos y radios tomada es la misma en todas las instancias, $m = n$. El objetivo es poder aislar el factor granularidad y ver de qué forma este afecta a la isoterma obtenida.

2.2.2. Test de comparación Gauss vs LU

La siguiente experimentación tiene la intención de determinar las presuntas ventajas en determinadas condiciones de utilizar factorización LU en lugar del algoritmo de eliminación de Gauss-Jordan para hallar la (o las) solución (es) a un sistema de ecuaciones lineales. Para realizar dicha experimentación se generaron instancias aleatorias con las mismas semillas, utilizando los archivos (o modificaciones de los mismos) *genTest.py* y *test.sh*, y se compararon los tiempos de cómputo en función de la cantidad de puntos del sistema y la cantidad de instancias a resolver.

El tiempo de cómputo para cada algoritmo fue medido con los métodos provistos por la cátedra (ubicados en *time.h*). El tiempo de cómputo total para cada tamaño fue calculado considerando solo los métodos de los algoritmos en cuestión más los necesarios para la resolución del sistema (*backward* y *forward substitution*). Se omitió el tiempo de los métodos que plantean al sistema por no ser considerados parte de los algoritmos de resolución de sistemas matriciales. Todo esto puede apreciarse en *main.cpp*.

Se realizaron varias ejecuciones de la experimentación considerando como valor final el promedio de dichas ejecuciones. Cabe aclarar que si bien las instancias son “aleatorias”, se usan las mismas tanto para LU como para Gauss porque se usa la misma semilla. La idea de esta experimentación es determinar si al cambiar las condiciones del entorno (o en términos más teóricos el vector b del sistema $Ax = b$) en forma “continua”, la factorización LU ahorra cálculos frente al método de eliminación de Gauss. Los tamaños de las matrices fueron fijados de manera que cubran el mayor espacio posible de instancias sin tener que caer en ejecuciones “eternas”. Por otro lado, en todas las matrices la cantidad de radios y ángulos es la misma, para que los tiempos de cómputo sean lo más equilibrados posibles.

2.2.3. Test de isoterma en función de las condiciones de borde

La siguiente experimentación tiene por objetivo principal analizar cómo se comporta el modelo implementado. Es decir, ver cómo refleja los cambios esperados en una situación dada del problema modelado. En un altohorno real r_i y r_e no sufren cambios, lo que varía constantemente son las $T_i(\theta_j)$ y las $T_e(\theta_j)$. Para simular esto y poder ver cómo varía la isoterma elegimos una buena discretización, la mayor que podamos costear en función de nuestra potencia de procesamiento, mantenemos r_i y r_e constantes y vamos variando las $T_i(\theta_j)$ y las $T_e(\theta_j)$ de diferentes formas para ver cómo se comporta la isoterma. Los casos que vamos a considerar son los siguientes:

1. aumenta $T_i(\theta_j)$ y $T_e(\theta_j)$ para todo j de la discretización.
2. aumenta $T_i(\theta_j)$ y se mantienen constante $T_e(\theta_j)$ para todo j de la discretización.
3. aumenta $T_i(\theta_j)$ y descende $T_e(\theta_j)$ para todo j de la discretización.

Elegimos estos tres casos porque creemos que cubren la mayoría de los comportamientos. A nivel teórico no hacemos distinción entre las $T_i(\theta_j)$ y las $T_e(\theta_j)$, por lo tanto tenemos dos componentes que pueden cambiar en un sentido u otro o mantenerse constantes. El caso 1 toma que las dos componentes cambian y en el mismo sentido. El caso 2 que una componente cambia y la otra permanece constante. Finalmente, el caso 3 considera la situación en la que las dos componentes cambian pero en sentidos inversos.

2.2.4. Test de tiempo en función de la granularidad de la discretización

La siguiente experimentación tiene por objetivo analizar cómo y cuánto influye la granularidad de la discretización en el tiempo de ejecución de nuestros algoritmos. Sólo utilizamos el método de Gauss-Jordan ya que no nos interesa comparar Gauss-Jordan con LU, hay otro test específico para eso. Para aislar la componente granularidad decidimos tomar r_i , r_e , $T_i(\theta_j)$, $T_e(\theta_j)$ e *iso* constantes. Lo único que varía entre las instancias del test es la cantidad de ángulos y radios en la que discretizamos el horno. Tomamos cantidades de ángulos y radios siempre iguales porque sólo nos

interesa la granularidad como cantidad de puntos. Es decir, que la cantidad de puntos en nuestra discretización va a ser:

$$puntos = m^2 = n^2 \quad (1)$$

Además, para cada cantidad de puntos calculamos 15 instancias y después sacamos promedio para tener una mejor aproximación al tiempo que debería tardar la ejecución del algoritmo y tratar de minimizar el error por las características de los sistemas operativos.

3. Resultados

3.1. Gauss vs LU

A continuación mostramos los resultados obtenidos para el test de comparación entre factorización LU y eliminación gaussiana, los tiempos de cómputo se muestran en segundos. Se muestran los resultados de matrices de 2500x2500, 900x900 y 100x100.

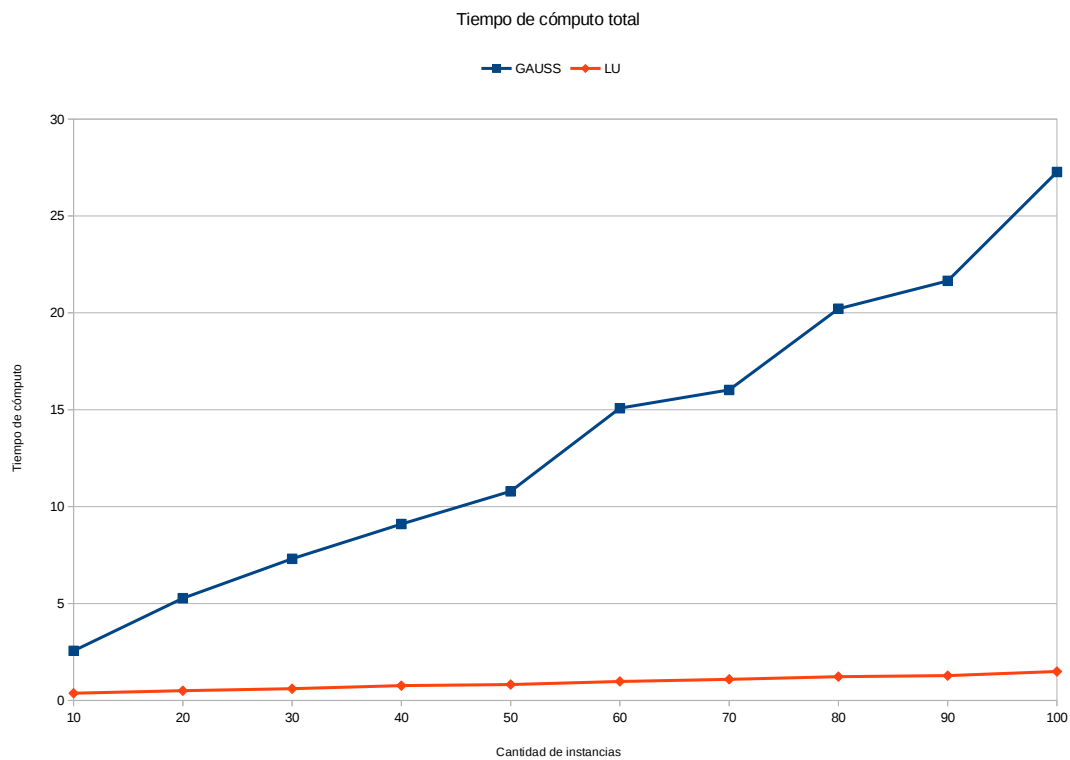


Figura 1: Resultados obtenidos usando matrices de 50 ángulos y 50 radios.

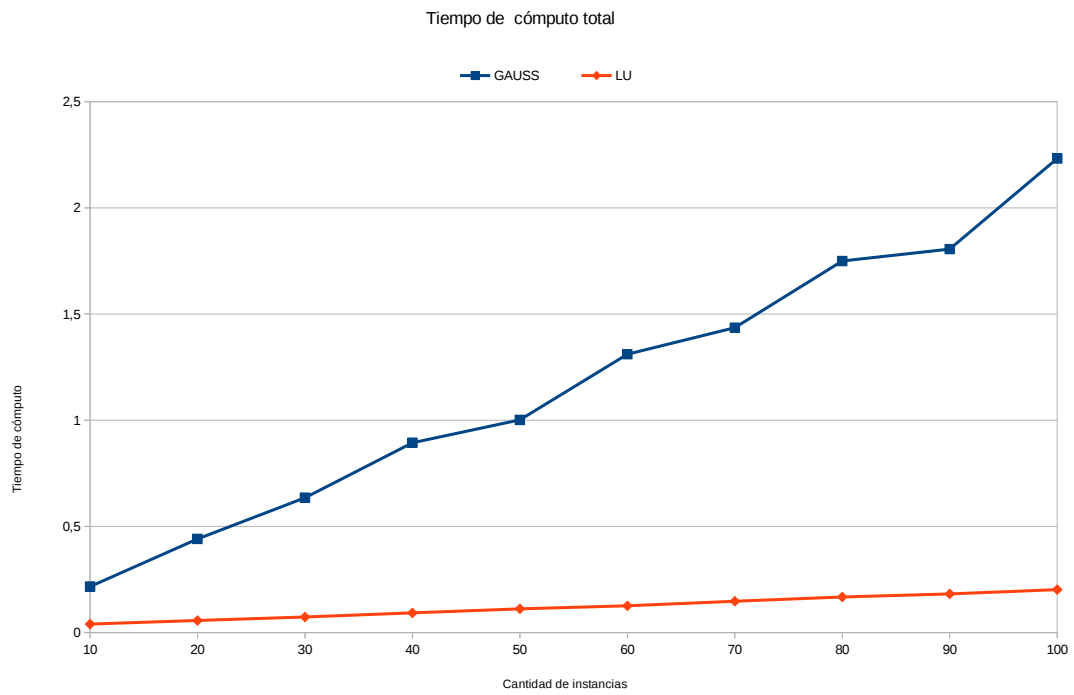


Figura 2: Resultados obtenidos usando matrices de 30 ángulos y 30 radios.

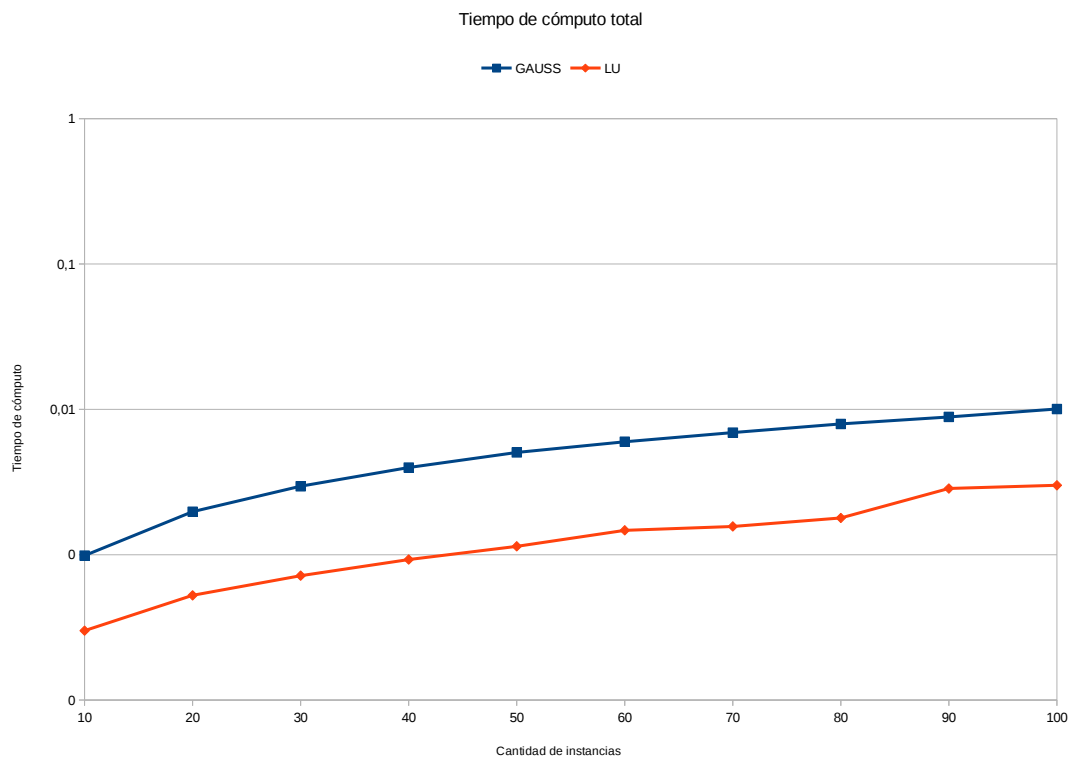


Figura 3: Resultados obtenidos usando matrices de 10 ángulos y 10 radios.

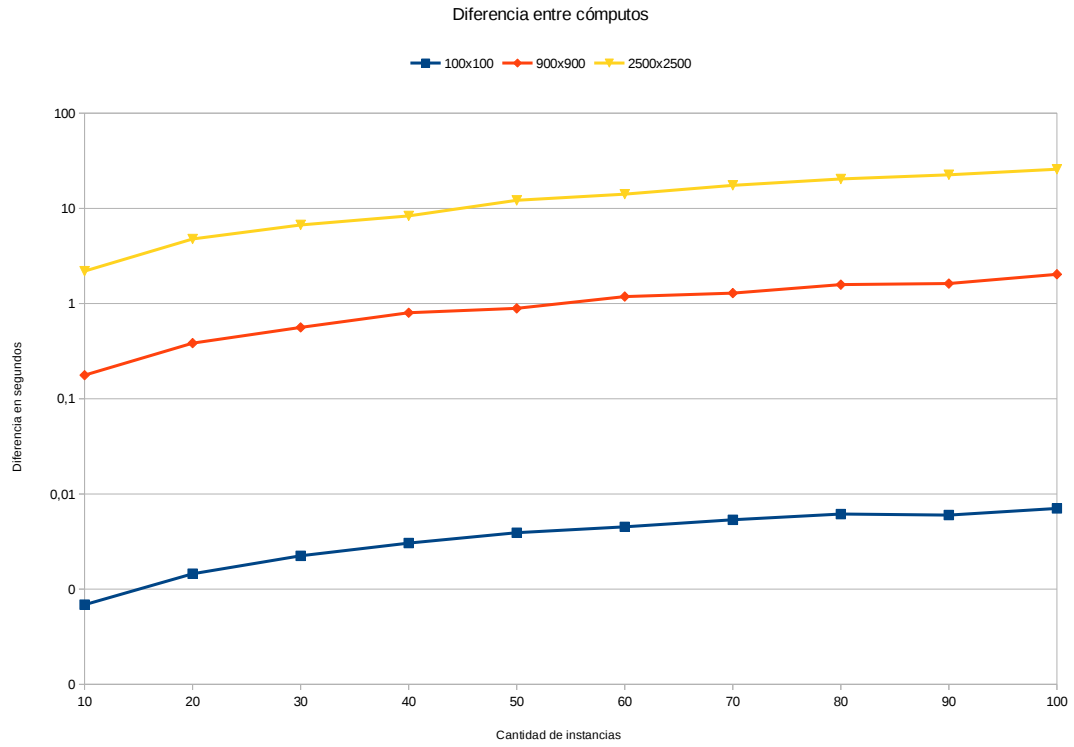


Figura 4: Diferencia entre el tiempo consumido por LU y Gauss de los gráficos anteriores. Uso de escala logarítmica.

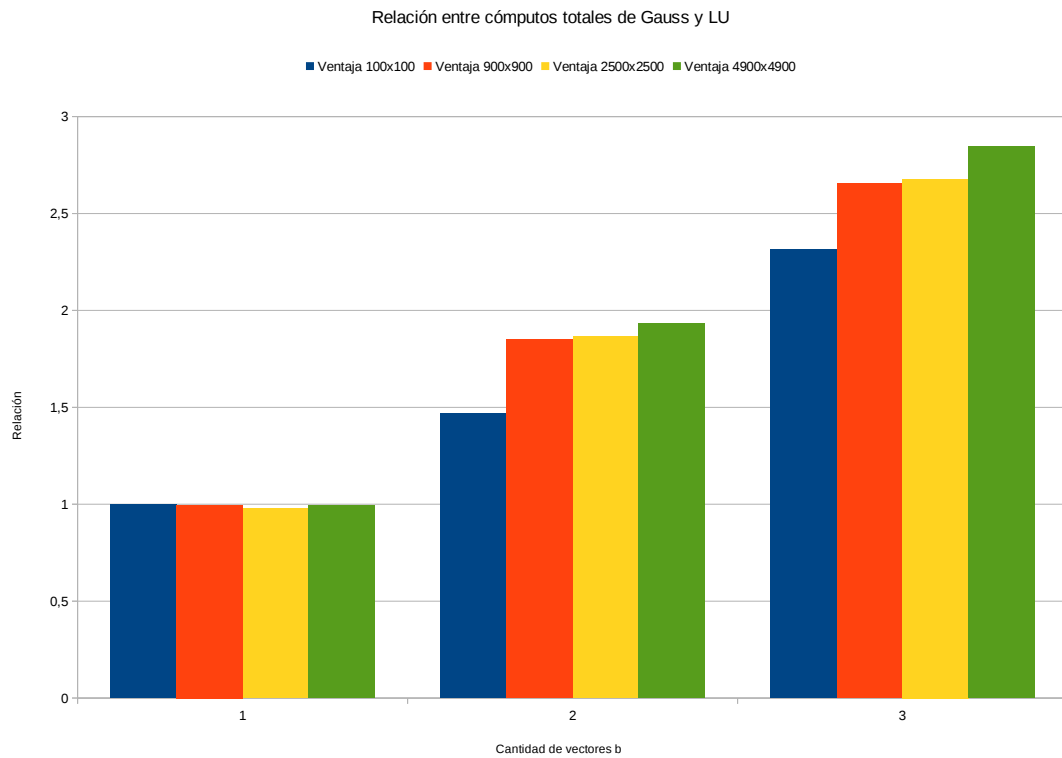


Figura 5: Relación entre el tiempo total consumido por Gauss y LU en función de la cantidad de vectores b.

3.2. Granularidad vs Tiempo

Aquí veremos cómo afectaría el tiempo cambiar la granularidad. Corrimos el algoritmo de Gauss para granularidades desde 50^2 hasta 104^2 con 5 instancias iguales para cada uno y calculamos el promedio de tiempo de ejecución para cada una.

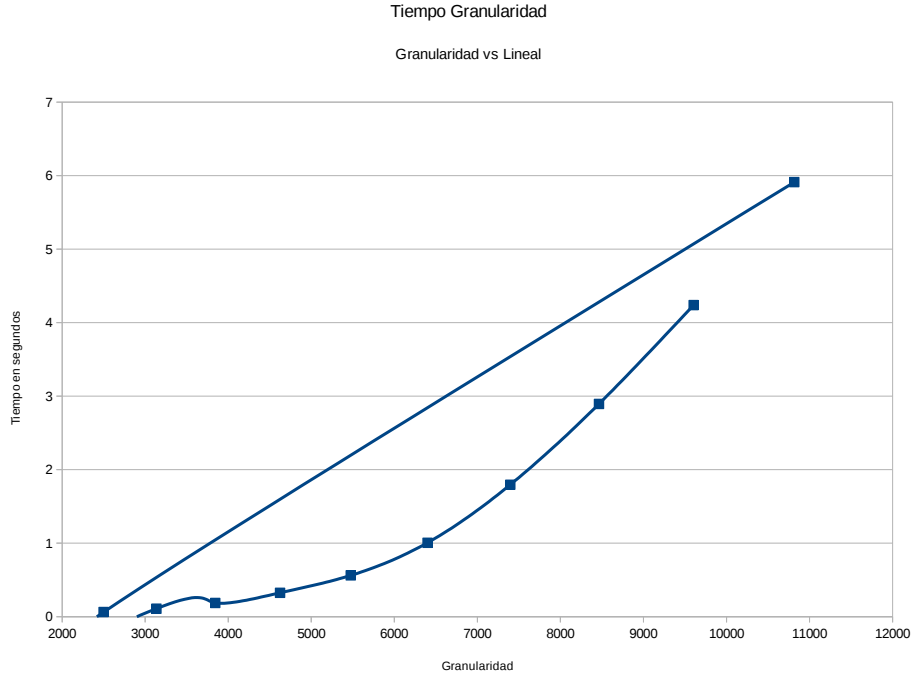


Figura 6: Tiempo de aumentar la granularidad vs una función lineal.

4. Discusión

4.1. Análisis de comparación Gauss contra LU

Como se puede apreciar en los gráficos 1, 2 y 3, al variar la cantidad de instancias el tiempo de cómputo de la factorización LU fue muy inferior al tiempo de la eliminación gaussiana, y esta diferencia se hace sumamente notoria a medida que la cantidad de vectores b (a partir de ahora simplemente “instancias”) a calcular aumenta. En el gráfico 4 se puede observar como la diferencia es proporcional tanto a la cantidad de instancias como al tamaño de las matrices, es decir, al aumentar la cantidad de instancias y/o el tamaño de la matriz LU aumenta la diferencia con respecto a Gauss. Por otro lado en 5 se puede determinar que basta con resolver el sistema de ecuaciones con sólo dos vectores b distintos para poder ya apreciar una ventaja efectiva sobre la eliminación gaussiana y que para el caso en que solo haya un sólo vector b , los tiempos de cómputo son prácticamente idénticos, es decir la *forward substitution* extra no agrega *overhead* al cálculo de la solución. También es posible notar que para los diferentes tamaños matriciales, la relación se mantiene para una misma cantidad de instancias, y que dicha relación muestra un incremento significativo a medida que se incrementa la cantidad de instancias, por lo que se puede determinar que el factor más importante que influye sobre la diferencia de rendimiento entre eliminación gaussiana es la cantidad de instancias y no tanto el tamaño de la matriz.

4.2. Análisis de comparación de granularidad y tiempo

Como era de esperarse, el tiempo de ejecución es creciente con la granularidad. Lo que sí descubrimos en este test es que el tiempo parece crecer más que linealmente dado que la pendiente de la función que parecerían describir los datos es mayor que la de la función lineal que usamos para comparar. Esto en un principio parecería intuitivo dado que la complejidad de nuestros algoritmos depende de la matriz A en la ecuación $Ax=b$, y las dimensiones de ésta dependen de la granularidad, pero de la granularidad al cuadrado. Por ende, mientras más crece la granularidad más crece la dimensión de la matriz, pero cuadráticamente.

5. Conclusiones

6. Apéndices

6.0.1. Demostración de la proposición

Proposición:

Sea $A \in K^{n \times n}$ con $K = \mathbb{R}$ o \mathbb{C} una matriz diagonal dominante e inversible entonces es posible aplicar eliminación gaussiana sin pivoteo.

Notación útil:

$A(i|j) :=$ Es la submatriz de A que se obtiene de eliminar las filas de 1 a i y las columnas de 1 a j .

$A^{(k)} :=$ Es la matriz obtenida luego de realizar k pasos de eliminación gaussiana sobre las filas de A .

Demostración:

Como A es diagonal dominante, se tiene que $|A_{ii}| \geq \sum_{j=1, i \neq j}^n |A_{ij}|$ para todo $j \neq i$, por lo tanto debe suceder que $A_{11} \neq 0$, si así no fuera, se tiene que $0 = |A_{11}| \geq |A_{1j}| \geq 0$ para todo $j \neq 2$ a n , por lo que $|A_{1j}| = 0$, y esto sucede si y solo si $A_{1j} = 0$, es decir, la columna 1 es nula y por lo tanto A no es inversible, pero A es inversible y en consecuencia $A_{11} \neq 0$. Teniendo en cuenta que $A_{11} \neq 0$, veamos ahora que al realizar eliminación gaussiana sobre A , la matriz $A(1|1)^{(1)}$ resulta ser diagonal dominante: Hay que probar que para todo j vale:

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

Tenemos que:

$$\sum_{i \geq 2, i \neq j}^n |A_{ij}^{(1)}| = \sum_{i \geq 2, i \neq j}^n \left| A_{ij} - \frac{A_{1j}A_{i1}}{A_{11}} \right| \leq \sum_{i \geq 2, i \neq j}^n |A_{ij}| + \left| \frac{A_{1j}A_{i1}}{A_{11}} \right| = \sum_{i \geq 1, i \neq j}^n |A_{ij}| - |A_{1j}| + \left| \frac{A_{1j}}{A_{11}} \right| \left(\sum_{i \geq 2}^n |A_{i1}| - |A_{j1}| \right)$$

y como A es diagonal dominante:

$$\sum_{i \geq 1, i \neq j}^n |A_{ij}| \leq |A_{jj}| \quad \text{y} \quad \sum_{i \geq 2}^n |A_{i1}| \leq |A_{11}|$$

Finalmente:

$$\begin{aligned} \sum_{i \geq 1, i \neq j}^n |A_{ij}| - |A_{1j}| + \left| \frac{A_{1j}}{A_{11}} \right| \left(\sum_{i \geq 2}^n |A_{i1}| - |A_{j1}| \right) &\leq |A_{jj}| - |A_{1j}| + \left| \frac{A_{1j}}{A_{11}} \right| (|A_{11}| - |A_{j1}|) = \\ |A_{jj}| - \left| \frac{A_{1j}A_{j1}}{A_{11}} \right| &\leq \left| A_{jj} - \frac{A_{1j}A_{j1}}{A_{11}} \right| = |A_{jj}^{(1)}| \end{aligned}$$

Por lo tanto $A(1|1)^{(1)}$ es diagonal dominante e inversible porque la matriz original lo era (si no lo fuera se podría triangular obteniendo al menos una fila con ceros, pero como A ya tiene ceros en la primer columna, existiendo una triangulación de A con una fila nula, lo que es un absurdo pues A ya era inversible), por lo que aplicando la demostración anterior para $A(1|1)^{(1)}$, se tiene que también es posible realizar el paso 2 de gauss sin pivoteo. Aplicando la demostración anterior recursivamente se obtiene que las matrices $A(2|2)^{(2)}, \dots, A(k|k)^{(k)}$ son todas diagonal dominantes e inversible, donde k es el último paso de la eliminación gaussiana, demostrando lo pedido.

7. Referencias