ANÁLISIS TEÓRICO Y COMPUTACIONAL SOBRE MATRICES ESPARZAS

CESAR FERNANDO BOLAÑOS REVELO

FACULTAD DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE MATEMÁTICAS Y ESTADÍSTICA
UNIVERSIDAD DE NARIÑO
SAN JUAN DE PASTO
2016

ANÁLISIS TEÓRICO Y COMPUTACIONAL SOBRE MATRICES ESPARZAS

CESAR FERNANDO BOLAÑOS REVELO

Trabajo presentado como requisito parcial para optar al título de Licenciado en Matemáticas

Asesora
Catalina María Rúa Álvarez
Doctora en Matemáticas Aplicadas

FACULTAD DE CIENCIAS EXACTAS Y NATURALES
DEPARTAMENTO DE MATEMÁTICAS Y ESTADÍSTICA
UNIVERSIDAD DE NARIÑO
SAN JUAN DE PASTO
2016

Nota de Responsabilidad

Todas las ideas y conclusiones aportadas en el siguiente trabajo son responsabilidad exclusiva del autor.

Artículo 1^{ro} del Acuerdo No. 324 de octubre 11 de 1966 emanado por el Honorable Consejo Directivo de la Universidad de Nariño.

Nota de Aceptación

Catalína María Rúa Álvarez
Directora de Tesis

Eduardo Ibargüen Mondragón
Jurado

Paul E. Castillo
Jurado

San Juan de Pasto, noviembre 24 de 2016

Agradecimientos

Quiero expresar mis más profundos agradecimientos a aquellos que apoyaron mi proceso formativo para la realización y culminación de mi trabajo de grado.

En la parte espiritual le agradezco a Dios, por brindarme la sabiduría, la paciencia, la constancia y la tolerancia necesaria para corregir mis errores y aprender de ellos para hacer las cosas de una manera adecuada.

A mi familia que me apoyó en este proceso, tanto moral como económicamente para poder estudiar mi carrera y terminar este trabajo satisfactoriamente. Especialmente a mi padre forjador y ejemplo de constancia.

A la Universidad de Nariño; mi alma mater que me acogió todos estos años, y me permitió forjar valores y adquirir conocimientos para salir adelante y me brindó los espacios necesarios para la realización de mi trabajo.

A los profesores del departamento de matemáticas y estadística de la Universidad de Nariño, que hicieron parte de mi proceso de formación académica.

A los jurados de tesis el Dr. Paul Castillo y al Dr. Eduardo Ibargüen por las recomendaciones y los consejos recibidos en la revisión de mi trabajo de grado.

Finalmente a mi asesora de Tesis la Dra. Catalina M. Rúa; mis más sinceros agradecimientos por su acompañamiento, su tiempo, sus sugerencias y su apoyo teórico y práctico en todo este proceso.

Cesar Fernando Bolaños Revelo

Universidad de Nariño
Octubre 12 de 2016.

Resumen

Una gran cantidad de problemas científicos pueden ser modelados matemáticamente y aunque no todos los modelos cuentan con una solución analítica, gracias al avance de las herramientas tecnológicas, la simulación numérica se ha convertido en una excelente opción para obtener aproximaciones de estos modelos. Sin embargo, se presentan situaciones en las cuales el tiempo o la memoria empleada en los cómputos es tan alta que se debe tener un cuidado especial. En este trabajo se realiza un estudio tanto teórico como computacional en matrices que intervienen en diversas aplicaciones, que se caracterizan por ser de gran escala y tener un gran número de elementos nulos, denominadas matrices esparzas. Para lograr este fin, se hace una recopilación de algunos de los principales conceptos en el estudio de matrices, como sus propiedades, relación con los sistemas lineales y métodos numéricos para aproximar la solución de sistemas lineales, particularmente haciendo énfasis en los métodos del gradiente. En este sentido se enfatiza en el estudio de esquemas de almacenamiento, estructuras de datos y en la eficiencia computacional en operaciones para matrices esparzas. Con la ayuda de los software MATLAB y DEV-C++, se realizan pruebas numéricas para determinar y comparar los tiempos de computo, la memoria empleada y para verificar que los resultados numéricos que se producen son lo suficientemente próximos a los resultados esperados. Las matrices esparzas usadas se relacionan con diversas aplicaciones reales y han sido adquiridas de los recursos electrónicos Matrix Market y de la base de datos de la Universidad de Florida.

Abstract

A great quantity of scientific problems can be mathematically modeled and although all the models do not have an analytic solution, thanks to the advance of technological tools, the numerical simulation became an excellent option to obtain the desired approximation. However, there are situations in which the time or the memory used in the computations is so high that a special attention is needed. In this work, a theoretical as same as computational study in matrixes that appear in a variety of applications with large scales and several zeros entries named sparse matrixes are developed. To achieve this aim, a compilation of the main concepts related with matrixes, some properties, linear systems and numerical methods to approximate its solutions, emphasizing particularly on the gradient´s methods is done. In this sense, the attention is centered in the study of storage systems, data structures and computational efficiency to work operations in relation to the sparse matrixes. The software MATLAB and DEV-C++ are used as a help to do numerical tests that allow us to obtain numerical results with the computational time, the use of memory and in order to compare the real and approximated solutions. The sparse matrixes used are related to varied real applications and they have been acquired from the electronic resources as Matrix Market and the Florida´s University data base.

Índice general

Índice de tablas

Índice de figuras

Abreviaciones

Matrices y vectores

- $\boldsymbol{det}()$: Determinante.
- $<,>$: Producto punto.
- $\boldsymbol{span}\{\cdot\}$: Espacio Generado.
- **LI**: Linealmente Independiente.
- **LD**: Linealmente Dependiente.
- **dim**: Dimensión.
- **S.D.P**: Simétrica definida positiva.
- $\boldsymbol{D_g}\{\cdot\}$: Matriz diagonal.
- $\boldsymbol{k}()$: Condicionamiento.

Norma Matriciales y Vectoriales

- $\|\cdot\|$: Norma vectorial.
- $\|\cdot\|_{\mathbf{1}}$: Norma vectorial 1.
- $\|\cdot\|_{\mathbf{2}}$: Norma vectorial 2.
- $\|\cdot\|_{\boldsymbol{\infty}}$: Norma vectorial infinito.
- $|||\cdot|||$: Norma matricial.
- $|||\cdot|||_{\mathbf{1}}$: Norma matricial 1.
- $|||\cdot|||_{\mathbf{2}}$: Norma matricial 2.
- $|||\cdot|||_{\boldsymbol{\infty}}$: Norma matricial infinito.
- $|||\cdot|||_{\boldsymbol{F}}$: Norma de Frobenius.

Esquemas de Almacenamiento

- **AIJ**: Esquema coordinado.
- **CSR**: Compressed sparse rows.
- **CSC**: Compressed sparse columns.
- **CSR mod**: Compressed sparse rows modified.
- **MSR**: Modified sparse rows.
- **MSC**: Modified sparse columns.
- **CSV**: Compressed sparse vector.

Métodos del Gradiente

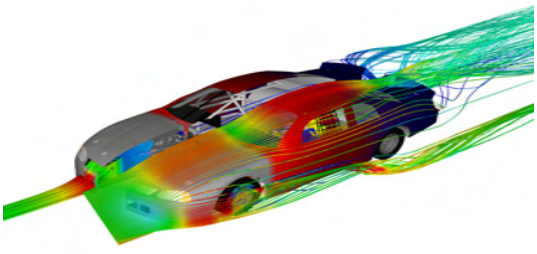
- **CG**: Gradiente conjugado.
- **CG pre**: Gradiente conjugado precondicionado.
- **BCG**: Gradiente biconjugado.
- **BCG pre**: Gradiente biconjugado precondicionado.
- **CGS**: Gradiente conjugado cuadrado.
- **CGS pre**: Gradiente conjugado cuadrado. precondicionado.
- **BICGSTAB**: Gradiente biconjugado estabilizado.
- **BICGSTAB pre**: Gradiente biconjugado estabilizado. precondicionado.

Introducción

Al abordar diversos problemas científicos, surge la necesidad de recurrir a las matemáticas para diversos fines, como realizar el modelamiento, entender el comportamiento o para poder solucionarlos. En disciplinas como la ingeniería estructural, la electrónica, la dinámica de fluidos computacional, los problemas de optimización, la simulación de procesos químicos y contaminantes, etc., la modelación matemática se sustenta frecuentemente en la solución de ecuaciones diferenciales; sin embargo, dado que en muchos casos no se cuenta con las soluciones analíticas se requieren de aproximaciones numéricas.

Entre las discretizaciones numéricas para ecuaciones diferenciales ordinarias o parciales, se destacan los métodos de paso simple o múltiple, los métodos de diferencias finitas, volúmenes finitos y elementos finitos, entre otros. Una característica común de estas discretizaciones es que con frecuencia aparece la necesidad de recurrir en su proceso a realizar productos matriz por vector o resolver sistemas lineales. Las matrices que intervienen en estos procesos suelen ser de grandes dimensiones y cuentan con una gran cantidad de elementos no nulos al compararse con el número de elementos que tienen, siendo así conocidas como matrices esparzas. Debido a que el trabajar con la matriz densa, sin tener en cuenta sus características, conlleva al uso de una gran cantidad de memoria y tiempo computacional, en algunos casos es imposible o toma demasiado tiempo realizar las operaciones necesarias con las matrices esparzas y por tanto se recurre al uso de ciertas estructuras de datos o esquemas de almacenamiento que tengan en cuenta únicamente la información de sus entradas diferentes de cero, con el fin de aprovechar la propiedad que estas poseen y aplicarlas de forma eficiente.

(a) Ingeniería estructural [30].



(b) Dinámica de fluidos computacional [1].

Figura 1: Aplicaciones relacionadas con matrices esparzas.

El estudio de las matrices esparzas inicia cerca de 1960 debido a la investigación realizada por ingenieros electrónicos al resolver un problema real de redes eléctricas a través de la resolución de sistemas lineales, donde se encontraron que la matriz del sistema era esparza y sus elementos tenían un patrón irregular. Dado que se quería solucionar este problema en computadores que en dicha época eran de baja gama, se vio la necesidad de explotar la esparcidad de la matriz. Fue entonces cuando se empieza a considerar para el almacenamiento de estas matrices estructuras de datos eficientes que realizaran operaciones de alto costo con la meta de disminuir tiempo y memoria computacional. Lo anterior hizo que se pudiese trabajar con problemas y aplicaciones que involucraban grandes sistemas lineales esparzas, mientras que si no se aplicaba la esparcidad y se usaban métodos con la matriz completa o "densa", algunos problemas no podían ser solucionados por que llegaban al límite de la memoria o demoraban tiempos absurdos que los tornaban imposibles. Con el paso del tiempo las matrices esparzas llegaron a ser de gran relevancia en varios campos, prueba de ello se da en 1968 con la primera conferencia en los centros de investigación de IBM, dedicada a estas matrices. Luego, se crearon otras seis conferencias con memorias enfocadas únicamente a estas mismas matrices. Fue tal su importancia, que cerca de 1975 ya había alrededor de 600 publicaciones de este tópico, además libros de textos de análisis numérico y de estructuras de datos contaban con capítulos enfocados a este tema [26].

Actualmente debido al avance tecnológico, con el uso de computadores de alta gama y a las diversas investigaciones alrededor de las matrices esparzas, se ha podido mejorar y realizar operaciones de alto costo computacional con matrices esparzas de grandes dimensiones que años atrás eran imposibles ser almacenadas y/o operadas con las herramientas computacionales que se tenían. Este logro se debe también a los avances en estructuras de datos y la implementación de métodos como la programación en paralelo y en GPU [6, 2]. Además de esto existen muchos paquetes que permiten trabajar con matrices esparzas, como es el caso de LAPACK [23, 24]. Otro hecho relevante es que debido a recursos electrónicos, como "Matrix Market" y el disponible por la Universidad de Florida, es posible descargar una gran variedad de matrices esparzas obtenidas de diversas aplicaciones reales, promoviendo así la investigación en este campo [4, 33, 13].

Por la importancia de las matrices esparzas, el objetivo principal de este trabajo es realizar por un lado, una recopilación teórica alrededor de las matrices esparzas, centrándose en su definición formal, en ciertos esquemas de almacenamiento que se encuentran en la literatura, en ondear en operaciones donde interviene frecuentemente y que son más investigadas como es el producto matriz por vector y la solución de sistemas lineales, mediante la elección de métodos directos como la factorización LU, eliminación Gaussiana y la factorización de Cholesky y métodos iterativos como los métodos del gradiente. Por otro lado, además de la teoría es presentar las pruebas numéricas obtenidas al aplicar los esquemas de almacenamiento estudiados destacando el funcionamiento en algunas operaciones matriciales y en la solución de sistemas lineales a matrices esparzas adquiridas de aplicaciones reales disponibles de los recursos electrónicos mencionados y de la discretización explícita de una ecuación de transporte de onda unidireccional. Para la obtención de los resultados numéricos se creó tanto algoritmos en MATALB como en DEV-C++ aplicando la programación orientada a objetos POO [22, 14, 27].

Para cumplir los objetivos propuestos, el trabajo se distribuye de la siguiente manera: en el capítulo 1, se muestran los preliminares alrededor del estudio de las matrices y vectores, necesarios para abordar temas posteriores. En el capítulo 2, se introduce el tema de las matrices esparzas, con su definición formal, los esquemas de almacenamiento estudiados, operaciones matriciales y se incluyen algunos algoritmos, con el estudio del orden de complejidad. Seguidamente, en el capítulo 3 de resolución de sistemas lineales, se incluyen teoremas importantes en álgebra lineal numérica sobre el error en las soluciones al presentarse perturbaciones sobre el sistema. Además, se estudian el condicionamiento de la matriz, algunos métodos directos e iterativos para determinar la solución de sistemas lineales. En el capítulo 4, se presentan los métodos del gradiente que son métodos iterativos que aproximan la solución de sistemas, entre estos están: el gradiente conjugado (**CG**), el gradiente biconjugado (**BCG**), el gradiente conjugado cuadrado (**CGS**), el gradiente biconjugado estabilizado (**BICGSTAB**), e incluso se presenta el uso de un precondicionador básico para alguno de estos métodos. Finalmente, en el capítulo 5, se dan a conocer algunos resultados numéricos relacionados con las matrices esparzas, al implementar los esquemas de almacenamiento tratados en el capítulo 3, al estudiar su comportamiento en las operaciones más relevantes y en la solución de sistemas lineales con los métodos que se estudiaron. De lo anterior, en el capítulo 6 se encuentran las conclusiones y trabajos futuros. Adicionalmente se anexa junto a este trabajo un DVD que contiene las implementaciones realizadas en MATLAB, en DEV-C++ y el manual de usuario o documentación, para cada uno de estos.

Capítulo 1

Preliminares

En este capítulo se presentan algunos conceptos básicos sobre el estudio de matrices. Particularmente se centra en temas tales como las normas de vectores, las normas de matrices y algunas propiedades matriciales, que afectan tanto resultados teóricos como numéricos.

Los teoremas que se van a presentar a continuación se pueden encontrar en algunos textos de álgebra lineal como [12, 19, 16] y [21]. Dada la importancia, será incluida o no la demostración.

1.1. Matrices y vectores

Definición 1.1. (vector). Un vector $\boldsymbol{x}$ de longitud n, es un conjunto de n elementos ordenados, $x_1, x_2, \ldots, x_n$, que se lo representa como

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \text{ó} \quad \boldsymbol{x} = \begin{bmatrix} x_1 & \ldots & x_n \end{bmatrix}.$$

Los valores x_i, $i = 1, \ldots, n$, representan la posición i-ésima del vector $\boldsymbol{x}$ y se pueden denotar también en algunos contextos como $\boldsymbol{x}(i)$. Además, si se representa el vector de forma horizontal se denomina *vector fila* y de forma vertical se denomina *vector columna*.

El conjunto de vectores de longitud n, con elementos reales se denota con $\mathbb{R}^n$, y cuando los elementos son complejos se denota por $\mathbb{C}^n$.

El vector $\boldsymbol{x}$ que tenga todos sus elementos nulos se lo denota por $\mathbf{0}$.

A continuación se definen algunas operaciones básicas relacionadas con el conjunto de vectores reales $\mathbb{R}^n$.

Definición 1.2. (Suma de vectores). Sean $\boldsymbol{x}$ y $\boldsymbol{y} \in \mathbb{R}^n$. Se define la *suma de vectores* como

$$\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y} \in \mathbb{R}^n,$$

donde $z_i = x_i + yi$, $i = 1, 2, \ldots, n$.

Definición 1.3. (Vector por escalar).. Sea $\boldsymbol{x} \in \mathbb{R}^n$ y $a \in \mathbb{R}$. Se define el producto *vector por escalar* como

$$\boldsymbol{z} = \alpha\boldsymbol{x} \in \mathbb{R}^n,$$

donde $z_i = \alpha x_i$, $i = 1, 2, \ldots, n$.

Definición 1.4. (Producto punto). Sean $\boldsymbol{x}$ y $\boldsymbol{y} \in \mathbb{R}^n$. Se define el *producto punto* $< x, y >$, como

$$\alpha =< \boldsymbol{x}, \boldsymbol{y} > \in \mathbb{R},$$

donde $\alpha = \sum_{i=1}^{n} x_i y_i$.

Definición 1.5. (Matriz). Una *matriz* A , es un arreglo rectangular de $m \times n$ números conformados por m filas y n columnas representado de la siguiente forma

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

Si la matriz A tiene m filas y n columnas, se dice que la matriz es de orden $m \times n$. En el caso en que $m = n$, A se denomina *matriz cuadrada* de orden n.

Otra manera de representar la matriz A es

$$A = (a_{ij}),\ i = 1, 2, ..., m \text{ y } j = 1, 2, ..., n.$$

Los valores a_{ij} que componen la matriz A, se conocen como *entradas* o *elementos* de la matriz denotados también como $ent_{ij}A$ o $A(i, j)$.

El conjunto de matrices de orden $m \times n$ con entradas reales se los va a denotar por $\mathbb{R}^{m\times n}$ y para entradas complejas como $\mathbb{C}^{m\times n}$.

La columna j-ésima $\boldsymbol{a(ij)}$, $i = 1, 2, \ldots, n$ se denota como $Col_i A$. De igual forma $Row_i A$ denota los elementos de la fila i-ésima.

Se define A como la *matriz nula* denotada como $\mathbf{0}$, si sus entradas son ceros, es decir, $a_{ij} = 0$ $\forall\ i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n$. También se define la matriz $A \in \mathbb{R}^{n\times n}$ como la *matriz identidad* I_n, si $a_{ii} = 1$ y $a_{ij} = 0$, $i \neq j$, para $i = 1, 2, \ldots, n$, y $j = 1, 2, \ldots, n$.

Las matrices pertenecientes a $\mathbb{R}^{n\times 1}$ y $\mathbb{R}^{1\times n}$, que representan respectivamente los vectores columnas y vectores filas de n elementos, se denotan como $\mathbb{R}^n$.

A continuación se definen algunas de las operaciones matriciales básicas para el conjunto de matrices $\mathbb{R}^{m\times n}$.

Definición 1.6. (Suma de matrices). Sean A y $B \in \mathbb{R}^{m\times n}$. Se define la suma de matrices como

$$C = A + B \in \mathbb{R}^{m\times n},$$

donde $c_{ij} = a_{ij} + b_{ij}$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$.

Definición 1.7. (Multiplicación de Matrices). Sean $A \in \mathbb{R}^{m\times n}$ y $B \in \mathbb{R}^{n\times s}$. Se define la *multiplicación de matrices* $A \times B$ como

$$C = A \times B \ \in\ \mathbb{R}^{m\times s},$$

donde $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{ki}$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, s$.

Definición 1.8. (Matriz por vector). Sean la matriz $A \in \mathbb{R}^{m\times n}$ y el vector $\boldsymbol{x} \in \mathbb{R}^n$. Se define el producto *matriz por vector* $A\boldsymbol{x}$ como

$$\boldsymbol{y} = A\boldsymbol{x} \ \in\ \mathbb{R}^m,$$

donde $y_{ij} = \sum_{j=1}^{n} a_{ij} x_j$.

Definición 1.9. (Matriz por escalar). Sea $A \in \mathbb{R}^{m\times n}$ y $\alpha \in \mathbb{R}$. Se define el *producto matriz por escalar* αA como

$$C = \alpha A \in \mathbb{R}^{m\times n},$$

donde $c_{ij} = \alpha a_{ij}$, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$.

Definición 1.10. (Transpuesta). Sea $A \in \mathbb{R}^{m\times n}$. La matriz transpuesta de A, denotada por A^T, es una matriz de orden $n \times m$, en la cual sus entradas $a_{ij}^T = a_{ji}$, $i = 1, 2, ..., m$, $j = 1, 2, ..., n$.

Observación 1.1. En el caso particular de la operación transpuesta para los vectores columnas $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, se observa que al realizar el producto $\boldsymbol{x}^T\boldsymbol{y}$, se obtiene

$$\boldsymbol{x}^T\boldsymbol{y} = \sum_{i=1}^{n} x_i y_i =< \boldsymbol{x}, \boldsymbol{y} > .$$

Definición 1.11. (Determinante). Sea $A \in \mathbb{R}^{n\times n}$. El *determinante* de una matriz A denotado por $\det(A)$ o $|A|$ esta definido de forma inductiva como sigue: Si $n = 1$ entonces el $det(A) = a_{11}$. De otro modo,

$$det(A) = \sum_{j=1}^{n} a_{ij}(-1)^{i+j} det(Min_{ij}(A)) \quad (i = 1, 2, ..., n)$$

o también,

$$det(A) = \sum_{i=1}^{n} a_{ij}(-1)^{i+j} det(Min_{ij}(A)) \quad (j = 1, 2, ..., n),$$

donde $Min_{ij}(A)$ es la submatrix $n-1 \times n-1$ obtenida de borrar la fila i y la columna j de A.

Definición 1.12. (Operaciones elementales por fila). Sea $A \in \mathbb{R}^{m\times n}$. Se define una operación elemental por fila de una matriz A, a las operaciones que se realizan en las filas de la matriz, obtenidas de multiplicar a izquierda de A por una cierta matriz. Estas operaciones son:

1. Multiplicar la fila i-ésima de A por un escalar c no nulo

$$R_i \leftarrow CR_i.$$

2. Sumar un múltiplo de la fila i-ésima a la fila j-ésima de A

$$R_j \leftarrow cR_i + R_j,$$

3. Permutar o intercambiar las filas i-ésima y j-ésima de A

$$R_i \leftrightarrows R_j.$$

Definición 1.13. (Matrices elementales) Una matriz $E \in \mathbb{R}^{n\times n}$, es una *matriz elemental*, si se obtiene al aplicarle la matriz identidad I_n por medio de una de las operaciones elementales por fila.

Se denotará por E_i una matriz elemental obtenida de la aplicación de la operación elemental 1 aplicada a la fila i-ésima. La matriz elemental $E_{i,j}$, representa la matriz resultante al aplicarse la operación elemental 2 en la fila i-ésima, a través de la fila j-ésima. Por último se denota por P_{ij} a la matriz elemental obtenida de intercambiar la fila i-ésima con la fila j-ésima. De forma análoga se define las operaciones elementales por columnas.

Ejemplo 1.1. De la matriz identidad I_3 se puede definir las siguientes matrices elementales

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} \xrightarrow{R1 \leftarrow 4R_1} \begin{bmatrix}4&0&0\\0&1&0\\0&0&1\end{bmatrix} = E_1,$$

$$\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix} \xrightarrow{R3 \leftarrow 3R_1+R_3} \begin{bmatrix}1&0&0\\0&1&0\\3&0&1\end{bmatrix} = E_{13},$$

$$\begin{bmatrix}0&0&1\\0&1&0\\1&0&0\end{bmatrix} \xrightarrow{R1 \leftrightarrows R_3} \begin{bmatrix}1&0&0\\0&1&0\\3&0&1\end{bmatrix} = P_{13}.$$

Teorema 1.1. *Realizar una operación elemental en una matriz A, es equivalente a multiplicar a izquierda por una matriz elemental.*

Ejemplo 1.2. Sea

$$A = \begin{bmatrix}1&0&2\\3&-1&1\\1&1&5\end{bmatrix},$$

al multiplicar a izquierda por la matriz E_1 del ejemplo 1.1, se obtiene

$$E_1A = \begin{bmatrix}4&0&0\\0&1&0\\0&0&1\end{bmatrix}\begin{bmatrix}1&0&2\\3&-1&1\\1&1&5\end{bmatrix},$$

que es equivalente al realizar la operación elemental por fila $R_1 \leftarrow 4R_1$.

Corolario 1.1. *Multiplica A por derecha por una matriz elemental implica realizar una operación elemental por columnas.*

Definición 1.14. (Matriz particionada). Una matriz particionada es una descomposición exhaustiva de la matriz hacia una submatriz mutuamente exclusivas tal que cada entrada de la matriz original cae en una y solo una submatriz de la partición.

Ejemplo 1.3. De la matriz A del ejemplo 1.2, se puede particionar como

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right],$$

donde $A_{11} = \begin{bmatrix}1&3\end{bmatrix}$ $A_{12} = 1$, $A_{21} = \begin{bmatrix}0&2\\-1&1\end{bmatrix}$ y $A_{22} = \begin{bmatrix}1\\5\end{bmatrix}$.

El uso de las matrices particionadas es muy importante dado que se utiliza en la demostración y obtención de diversos resultados con matrices, en particular en los métodos directos como la factorización LU y la factorización de Cholesky, que se abordarán en el capítulo 3.

Con las matrices particionadas también es posible hacer operaciones tales como la suma y multiplicación de matrices en términos de la submatrices que conforman la partición, siempre y cuando se cumplan algunas condiciones las cuales son:

1. Dos matrices particionadas A y B pueden sumarse en términos de sus submatrices si y sólo si A y B son de el mismo orden de partición de la misma manera.

   Ejemplo 1.4. Sea las matrices

   $$A = \left[\begin{array}{cc|c} 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ 7 & 8 & 9 \end{array}\right] \quad \text{y} \quad B = \left[\begin{array}{cc|c} 3 & 4 & 5 \\ \hline 2 & 1 & 8 \\ 6 & 7 & 9 \end{array}\right]$$

   Sumando las matrices A y B se tiene que

   $$A + B = \left[\begin{array}{c|c} \begin{bmatrix} 1 & 2 \end{bmatrix} + \begin{bmatrix} 3 & 4 \end{bmatrix} & 3 + 5 \\ \hline \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} + \begin{bmatrix} 2 & 1 \\ 6 & 7 \end{bmatrix} & \begin{bmatrix} 6 \\ 9 \end{bmatrix} + \begin{bmatrix} 8 \\ 9 \end{bmatrix} \end{array}\right]$$
   $$= \left[\begin{array}{c|c} \begin{bmatrix} 4 & 6 \end{bmatrix} & 8 \\ \hline \begin{bmatrix} 6 & 6 \\ 13 & 15 \end{bmatrix} & \begin{bmatrix} 14 \\ 18 \end{bmatrix} \end{array}\right].$$

2. Dos matrices particionadas, A de orden $m \times n$ y B de orden $n \times s$ se pueden multiplicar en términos de sus submatrices si y sólo si las submatrices de A y las submatrices correspondientes de B obedecen la ley de la multiplicación de matrices; es decir, si A_α es una submatriz de orden $r \times s$ de A y B_β es una submatriz de orden $p \times q$ B entonces la multiplicación es solo posible si $p = s$.

   Ejemplo 1.5. Sean las matrices

   $$A = \left[\begin{array}{cc|c} 1 & 2 & 3 \\ 4 & 5 & 6 \\ \hline 7 & 8 & 9 \end{array}\right] \quad \text{y} \quad B = \left[\begin{array}{c|c} 3 & 5 \\ 1 & 3 \\ \hline 4 & 2 \end{array}\right].$$

Luego la multiplicación de $A \times B$ es

$$A \times B = \left[\begin{array}{c|c} \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} & \begin{bmatrix} 3 \\ 6 \end{bmatrix} \\ \hline \begin{bmatrix} 7 & 8 \end{bmatrix} & 9 \end{array}\right] \left[\begin{array}{c|c} \begin{bmatrix} 3 \\ 1 \end{bmatrix} & \begin{bmatrix} 5 \\ 3 \end{bmatrix} \\ \hline 4 & 2 \end{array}\right]$$

$$= \left[\begin{array}{c|c} \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \end{bmatrix} + 4 \begin{bmatrix} 3 \\ 6 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \end{bmatrix} + 2 \begin{bmatrix} 3 \\ 6 \end{bmatrix} \\ \hline \begin{bmatrix} 7 & 8 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \end{bmatrix} + 9 \times 4 & \begin{bmatrix} 7 & 8 \end{bmatrix} \times \begin{bmatrix} 5 \\ 3 \end{bmatrix} + 9 \times 2 \end{array}\right]$$

$$= \begin{bmatrix} 17 & 17 \\ 41 & 47 \\ 65 & 77 \end{bmatrix}.$$

1.2. Espacios vectoriales

En este apartado se dará a conocer algunas definiciones y teoremas básicos en el estudio de álgebra lineal, en específico algunos resultados para el conjunto de vectores de $\mathbb{R}^n$ y el conjunto de matrices $\mathbb{R}^{m \times n}$, que servirán como sustento teórico para abordar temas posteriores. Para el lector interesado, la demostración de los teoremas se pueden encontrar en [21].

Definición 1.15. (Espacio vectorial) Un *espacio vectorial* $\mathbb{V}$ es un conjunto de objetos, llamados *vectores*, junto con dos operaciones llamadas suma y multiplicación por un escalar que satisfacen los siguientes axiomas:

1. Si $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$, entonces $\boldsymbol{x} + \boldsymbol{y} \in \mathbb{V}$ (cerrado bajo la suma).
2. $\forall\ \boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$, $(x + y) + z = x + (y + z)$ (Ley asociativa).
3. Existe un vector $0 \in \mathbb{V}$, tal que para cada $\boldsymbol{x} \in \mathbb{V}$, $\boldsymbol{x} + 0 = 0 + \boldsymbol{x} = \boldsymbol{x}$ (el 0 es el vector nulo).
4. Para cada $\boldsymbol{x} \in \mathbb{V}$, existe un vector $-\boldsymbol{x}$ en $\mathbb{V}$ tal que $\boldsymbol{x} + (-\boldsymbol{x}) = 0$. (Inverso aditivo de la suma).
5. Si $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$ entonces $\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{y} + \boldsymbol{x}$ (Ley conmutativa).
6. Si $\boldsymbol{x} \in \mathbb{V}$ y α un escalar, entonces $\alpha\boldsymbol{x} \in \mathbb{V}$.
   (Cerradura bajo la multiplicación por un escalar).
7. Si $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{V}$ y *alpha* un escalar entonces $\alpha(\boldsymbol{x} + \boldsymbol{y}) = \alpha\boldsymbol{x} + \alpha\boldsymbol{y}$.
   (Ley distributiva).

8. Si $\boldsymbol{x} \in \mathbb{V}$ y α y β son escalares, entonces $(\alpha + \beta)\boldsymbol{x} = \alpha\boldsymbol{x} + \beta\boldsymbol{x}$.
   (ley distributiva).

9. Si $\boldsymbol{x} \in \mathbb{V}$ y α y β son escalares, entonces $\alpha(\beta\boldsymbol{x}) = (\alpha\beta)\boldsymbol{x}$.
   (Ley asociativa).

10. Para cualquier vector $\boldsymbol{x} \in \mathbb{V}$, existe un escalar 1, tal que $1\boldsymbol{x} = \boldsymbol{x}$. Se denota como 1 la identidad multiplicativa del escalar.

Ejemplo 1.6. El conjunto de vectores $\mathbb{R}^n$ y el conjunto de matrices $\mathbb{R}^{m\times}$ con la suma y la multiplicación por un escalar definidas anteriormente es un espacio vectorial.

Definición 1.16. (Subespacio Vectorial) Sea $\mathbb{H}$ un subconjunto no vació de $\mathbb{V}$ y suponga que $\mathbb{H}$ es un espacio vectorial bajo las operaciones de la suma y multiplicación por un escalar definidas en $\mathbb{V}$, entonces se dice que $\mathbb{H}$ es un *subespacio vectorial* de $\mathbb{V}$.

Definición 1.17. (combinación lineal). Sea el conjunto de vectores $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n\}$ en $\mathbb{V}$. Se dice que un elemento $\boldsymbol{x} \in \mathbb{V}$ es una combinación lineal de $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n$ si

$$\boldsymbol{x} = \alpha_1\boldsymbol{v}_1 + \alpha_2\boldsymbol{v}_2 + \alpha_3 \ldots + \alpha_n\boldsymbol{v}_n,$$

donde $\alpha_1, \alpha_2, \ldots, \alpha_n$ son escalares.

Definición 1.18. (Espacio Generado) Se dice que el conjunto formado por los vectores $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_n\}$ en $\mathbb{V}$ *genera el espacio* $\mathbb{V}$ si todo elemento de $\mathbb{V}$ se puede escribir como una combinación lineal de ellos. Esto nos quiere decir, que para cualquier $\boldsymbol{v} \in \mathbb{V}$, existe $\alpha_1, \alpha_2, \ldots \alpha_n$ escalares tales que

$$\boldsymbol{v} = \alpha_1\boldsymbol{v}_1 + \alpha_2\boldsymbol{v}_2 + \ldots + \alpha_n\boldsymbol{v}_n.$$

El conjunto de todas las combinaciones lineales a partir del conjunto generador se denota como

$$span\{v_1, v_2, \ldots, v_k\} = \{v : v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n\}$$

Definición 1.19. (Linealmente dependiente e independiente). Un conjunto de vectores $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ en el espacio vectorial $\mathbb{V}^n$ es *linealmente independiente* (LI) si la combinación lineal $\sum_{j=1}^{n} \alpha_j \boldsymbol{x}_j = 0$, con $\alpha_i, \alpha_2, \ldots, \alpha_n \in \mathbb{R}$, implica que $\alpha_i = 0$, $\forall\ i = 1, 2, \ldots, n$. De lo contrario, $\{x_1, \ldots, x_n\}$ es *linealmente dependiente* (LD).

Ejemplo 1.7. El conjunto de vectores $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$, en $\mathbb{R}^n$, donde $e_i(i) = 1$ y $e_i(j) = 0\ \forall\ j \neq i$, para $i = 1, 2, \ldots, n$, es un conjunto LD en $\mathbb{R}^n$.

Teorema 1.2. *El conjunto generado*

$$span\{v_1, v_2, \ldots, v_k\} = \{v : v = \alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n\}$$

es un subespacio vectorial de $\mathbb{V}$.

Teorema 1.3. *Un conjunto de n vectores en* $\mathbb{R}^m$ *siempre es LD si* $n > m$

Corolario 1.2. *Cualquier conjunto de n vectores LI en* $\mathbb{R}^n$ *generan a* $\mathbb{R}^n$.

Definición 1.20. (Base) Un conjunto finito de vectores $\{v_1, v_2, \ldots, v_k\}$ es una base para un espacio vectorial $\mathbb{V}$ si es una LI y genera a $\mathbb{V}$.

Observación 1.2. Por la definición de base y el teorema 1.2 cualquier conjunto de vectores LI en $\mathbb{R}^n$ es una base para $\mathbb{R}^n$. Por lo tanto el conjunto $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n\}$ del ejemplo 1.7 es una base de $\mathbb{R}^{`n}$, que se la conoce como *base canónica.*

Teorema 1.4. *Si los conjuntos de vectores* $\{\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_m\}$ *y* $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n\}$ *son bases del espacio vectorial* $\mathbb{V}$, *entonces* $m = n$. *Es decir que cualquier base tiene el mismo número de vectores.*

Definición 1.21. (Dimensión)Si el espacio vectorial $\mathbb{V}$ tiene una base finita, entonces se define la *dimensión* de $\mathbb{V}$ como el número de vectores en todas las bases, que se denota por dim $\mathbb{V}$.

Ejemplo 1.8. Como n vectores LI constituyen una base $\mathbb{R}^n$, se ve que

$$\dim \mathbb{R}^n = n.$$

Definición 1.22. (Ortogonalidad) Un conjunto de vectores $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k\}$ en $\mathbb{R}^n$ es *ortogonal* si $\boldsymbol{x}_i^T \boldsymbol{x}_j = 0$ sólo si $i \neq j$, $1 \leq i, j \leq n$. Si además, $x_i^T x_i = 1$, $i = 1, 2, \ldots, n$ entonces el conjunto de vectores es *ortonormal.* Así un conjunto de vectores que forman una base y es un ortonormal, es una *base ortonormal.*

1.3. Propiedades matriciales

A continuación se muestran algunos propiedades matriciales en $\mathbb{R}^{m \times n}$, que se usarán a lo largo del trabajo.

Definición 1.23. (Simétrica) . Sea $A \in \mathbb{R}^{n \times n}$. Una matriz A se denomina *simétrica* si

$$A^T = A.$$

Definición 1.24. (Matriz diagonal). Sea $A \in \mathbb{R}^{m\times n}$. Una matriz A se denomina **diagonal** si

$$a_{ij} = 0, \ \forall \ i \neq j.$$

Una matriz diagonal A también se la puede representar únicamente con los elementos de su diagonal, como $A = D_g\{a_{11}, a_{22}, \ldots, a_{kk},\}$, donde $k = \text{mín}(m, n)$.

Definición 1.25. (Triangular superior). Sea $A \in \mathbb{R}^{m\times n}$. Una matriz A se dice que es *triangular superior* si

$$a_{ij} = 0, \ \forall \ i > j.$$

Definición 1.26. (Triangular inferior). Sea $A \in \mathbb{R}^{m\times n}$. Una matriz A se dice que es *triangular superior* si

$$a_{ij} = 0, \ \forall \ i < j.$$

Definición 1.27. (Matriz de banda) . Sea $A \in \mathbb{R}^{m\times n}$. Una matriz A se llama *matriz de banda* si $a_{ij} \neq 0$ cuándo $l_1 < j - i < l_2$, $l_1, l_2 \in \mathbb{Z}^+$. La suma $l_1 + l_2 + 1$ es llamada la *longitud de banda* de A.

Definición 1.28. (Matriz no singular). Sea $A \in \mathbb{R}^{n\times n}$. Una matriz A cuadrada es no singular o invertible, si existe una matriz cuadrada $A^{-1} \in \mathbb{R}^{n\times n}$ tal que

$$AA^{-1} = A^{-1}A = I_n.$$

Definición 1.29. (Matriz ortogonal). Sea $A \in \mathbb{R}^{n\times n}$. Una matriz A es *ortogonal* si y sólo si

$$A^{-1} = A^T.$$

Definición 1.30. (Definida positiva). Sea $A \in \mathbb{R}^{n\times n}$. Una matriz A es *definida positiva* si se cumple que para cualquier $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$

$$\boldsymbol{x}^T A\boldsymbol{x} > 0.$$

En el caso de que A sea además simétrica se denomina simétrica definida positiva (S.D.P).

Definición 1.31. (Diagonalmente dominante por filas). Sea $A \in \mathbb{R}^{m\times n}$. Una matriz A es *diagonalmente dominante por filas* si

$$a_{ii} \geq \sum_{k=1}^{n} |a_{ik}| \ \ \forall \ i = 1, 2, \ldots, m.$$

En el caso en que se cumpla estrictamente la desigualdad anterior se dice que A es *estrictamente diagonalmente dominante por filas.* Análogamente una matriz es *diagonalmente dominante por columnas* si y sólo si

$$a_{ii} \geq \sum_{k=1}^{m} |a_{ki}| \ \ \forall \ i = 1, 2, \ldots, n$$

y *estrictamente diagonalmente dominante por columnas* si se cumple la desigualdad anterior.

El siguiente teorema reúne varias relaciones alrededor de matrices cuadradas reales. La demostración del teorema se puede encontrar en [21].

Teorema 1.5. *Para cualquier matriz $A \in \mathbb{R}^{n\times n}$ las siguientes proposiciones son equivalentes:*

1. *A^{-1} existe.*
2. *A tiene inverso a izquierdo ($A^{-1}A = I_n$) o inverso a la derecha ($AA^{-1} = I_n$).*
3. *$A\boldsymbol{x} = 0$ implica que el vector $\boldsymbol{x} = 0$.*
4. *A es una fila equivalente a I_n.*
5. *A es un producto de matrices elementales*
6. *$det(A) \neq 0$.*
7. *El número de filas (columnas) linealmente independientes de A es n.*
8. *Las filas de A son linealmente independientes.*
9. *Las columnas de A son linealmente independientes.*
10. *$AX = B$ tiene una única matriz solución X para cualquier matriz $B \in \mathbb{R}^{n\times s}$.*

El siguiente resultado relaciona una matriz simétrica con una matriz definida positiva.

Teorema 1.6. *Si A es una matriz simétrica, estrictamente diagonalmente dominante (columnas) y tiene entradas positivas en la diagonal, entonces A es definida positiva.*

La demostración se abordará en el capítulo 3, en la sección de métodos directos.

Ejemplo 1.9. Sea la matriz

$$B = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & -1 & 4 \end{bmatrix}.$$

Se puede observar que B es simétrica y estrictamente diagonal dominante por filas y columnas, entonces por el **teorema 1.6** se puede afirmar que B es S.D.P.

Definición 1.32. (Matrices similares) . Sean A y $B \in \mathbb{R}^{n\times n}$. Las matrices A y B son *similares* si sólo si existe una matriz no singular P tal que $B = P^{-1}AP$. Además si P es ortogonal ($P^TP = I_n$), las matrices son *ortogonalmente similares.*

Teorema 1.7. *Toda matriz real simétrica es ortogonalmente similar a una matriz diagonal $P^{-1}AP = D$, donde D es una matriz diagonal y P es una matriz ortogonal.*

La demostración del teorema se puede encontrar en [12].

Definición 1.33. (valores propios). Sea $A \in \mathbb{R}^{n\times n}$. Un número complejo λ, para el cual el sistema lineal

$$A\boldsymbol{x} = \lambda\boldsymbol{x}$$

tiene una solución $\boldsymbol{x} \neq 0 \in \mathbb{C}^n$ es llamado *valor propio* de A y el vector x de a se llama *vector propio* de A asociado al valor propio λ. Al mayor valor propio de A en valor absoluto se lo llama radio espectral y se lo denota como $\rho(A)$.

Teorema 1.8. *Si A es definida positiva entonces sus valores propios son positivos.*

Demostración. Sean A S.D.P y λ su valor propio de A, entonces existe un $\boldsymbol{x} \neq 0$ tal que,

$$A\boldsymbol{x} = \lambda\boldsymbol{x}.$$

Multiplicando por $\boldsymbol{x}^T$ a ambos lados se obtiene que

$$\boldsymbol{x}^T A\boldsymbol{x} = \lambda\boldsymbol{x}^T\boldsymbol{x} = \lambda\sum_{i=1}^{n} x_i^2.$$

Luego como A es definida positiva, se debe cumplir $\boldsymbol{x}^T A\boldsymbol{x} > 0$, entonces el valor $\lambda\sum_{i=1}^{n} x_i^2 > 0$ y por lo tanto λ debe ser positivo.

■

Teorema 1.9. *Sea $A \in \mathbb{R}^{n\times n}$. La matriz A es similar a una matriz diagonal de los valores propios de A, $P^{-1}AP = D = Dg\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, si sólo si las columnas de P son conjuntos linealmente independientes de vectores propios de A.*

Demostración. Si A es similar a una matriz diagonal de sus valores propios, entonces $P^{-1}AP = D = Dg\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Multiplicando a ambos lados de la igualdad a izquierda por P se deduce que $AP = PD$. Esto quiere decir que

$$ACol_1P = \lambda_1 Col_1P,\ Acol_2P = \lambda_2 Col_2P,\ \ldots,\ Acol_nP = \lambda_n Col_nP.$$

Como P es no singular, implica que las columnas de P son un conjunto vectores propios de A los cuales son ser linealmente independientes (teorema 1.5).

Ahora, para demostrar el recíproco, si P es una matriz en la cual sus columnas son un conjunto linealmente independientes de valores propios de A, entonces

$$ACol_1\ P = \lambda_1 Col_1\ P,\ ACol_2\ P = \lambda_2 Col_2\ P\ \ldots\ ACol_n\ P = \lambda_n Col_n\ P$$

lo que quiere decir que, $AP = PD$ y como las columnas de P son un forman un conjunto linealmente independiente, P es no singular (teorema 1.5), y se puede multiplicar a ambos lados a izquierda por P^{-1}, cumpliéndose que A es similar a $D = Dg\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$, la matriz diagonal de sus valores propios.

■

1.4. Normas de matrices y vectores

En conjuntos tales como las matrices y los vectores, en muchos casos se ve la necesidad de comparar cuan "cercanos" están dos matrices o dos vectores. Esto se observa, por ejemplo, en el vector error asociado en los métodos numéricos para resolver sistemas lineales, que ayuda a definir que tan cercanos son el vector aproximado del método y el vector solución. Una forma de realizar esta medición es por medio de normas vectoriales, dado que estas funciones permiten pasar del conjunto de vectores $\mathbb{R}^n$ al conjunto $\mathbb{R}$, permitiendo comparar vectores con valores reales.

De igual manera, el uso en normas matriciales nos permite medir distancias entre matrices y calcular una propiedad importante en los sistemas lineales, llamada condicionamiento, que se abordará en el capitulo 3.

El concepto de norma que se tratará en este apartado será para matrices cuadradas y vectores reales, dado que es donde se va a enfocar a lo largo del trabajo, sin embargo el concepto se define para mucho mas campos.

Definición 1.34. (Norma vectorial). Una norma vectorial $\|\cdot\|$ para vectores reales es una función

$$\|\cdot\|\colon \mathbb{R}^n \to \mathbb{R},$$

que cumple las siguientes condiciones:

1. *Definida positiva.* Para cualquier $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$

$$\|\boldsymbol{x}\| > 0,$$

y para $\boldsymbol{x} = 0$.

$$\|\boldsymbol{x}\| = 0.$$

2. *Homogeneidad.* Para cualquier $\boldsymbol{x} \in \mathbb{R}^n$ y $\alpha \in \mathbb{R}$

$$\|\alpha \boldsymbol{x}\| = \mid \alpha \mid \|\boldsymbol{x}\|.$$

3. *Desigualdad triangular* Para cualquier $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$

$$\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|.$$

En la literatura se pueden encontrar diferentes normas vectoriales en $\mathbb{R}^n$, algunas son las siguientes

1. La norma uno $\|.\|_1$, se define como $\|\boldsymbol{x}\|_1 = \sum_{k=1}^{n} \mid x_i \mid$.

2. La norma infinito $\|.\|_\infty$, se define como $\|\boldsymbol{x}\|_\infty = \max\limits_{i=1,2,\ldots,n} \mid x_i \mid$ (máximo elemento del vector $\boldsymbol{x}$ en valor absoluto).

3. La norma dos $\|.\|_2$, se define como $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{k=1}^{n} x_i^2}$.

A continuación se prueba que efectivamente cumplen la definición de norma vectorial.

Demostración.

1. Norma uno $\|\boldsymbol{x}\|_1 = \sum_{k=1}^{n} \mid x_i \mid$.

   - Definida positiva. Se puede notar que para cualquier $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$ existe al menos un $x_i \neq 0$, para algún $i = 1, 2, \ldots, n$, con lo cuál $|x_i| > 0$ y así

   $$|x_1| + |x_2| + \ldots + |x_i| + \ldots + |x_n| = \|\boldsymbol{x}\|_1 > 0.$$

   En el caso en el que todos los elementos del vector $\boldsymbol{x}$ son nulos, es decir $\boldsymbol{x} = 0$, entonces

   $$\|\boldsymbol{x}\|_1 = \sum_{i=1}^{n} |0| = 0.$$

   - Homogeneidad. Sean $\alpha \in \mathbb{R}^n$ un escalar real y $\boldsymbol{x} \in \mathbb{R}^n$ un vector cualesquiera. Por las propiedades de valor absoluto en los reales se tiene que

   $$|ab| = |a||b| \ \forall \ a, b \in \ \mathbb{R},$$

   entonces,

   $$\|\alpha \boldsymbol{x}\|_1 = \sum_{i=1}^{n} |\alpha x_i| = \alpha \sum_{i=1}^{n} |x_i| = \alpha \|\boldsymbol{x}\|_1.$$

- Desigualdad triangular. Para probar esta tercera propiedad, se va utilizar la desigualdad triangular en $\mathbb{R}$. Entonces, para $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$

$$\|x+y\|_1 = \sum_{i=1}^{n} |x_i + y_i| \leq \sum_{i=1}^{n} |x_i| + |y_i| = \|x\|_1 + \|y\|_1.$$

2. Norma infinito $\|\boldsymbol{x}\|_\infty = \text{máx}_{i=1,2,\ldots,n}(|\ x_i\ |)$

   Igualmente que el anterior, por la propiedad y definición de valor absoluto en los reales se cumplen las dos primeras condiciones. Ahora se va a determinar la ultima propiedad.

   - Desigualdad triangular. Sean $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, entonces

$$\|\boldsymbol{x}+\boldsymbol{y}\|_\infty = \max_{i=1,2,\ldots,n} |x_i + y_i|.$$

     Por la desigualdad triangular en $\mathbb{R}$

$$\max_{i=1,2,\ldots,n} |x_i + y_i| \leq \max_{i=1,2,\ldots,n} |x_i| + \max_{i=1,2,\ldots,n} |y_i|,$$

     es decir,

$$\|\boldsymbol{x}+\boldsymbol{y}\|_\infty \leq \|\boldsymbol{x}\|_\infty + \|\boldsymbol{y}\|_\infty.$$

3. Norma dos $\|\boldsymbol{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i}$.

   Como esta definida $\|.\|_2$ por medio de la raíz cuadrada, y por la propiedades que tiene esta función, de forma análoga se cumplen las dos primeras propiedades como se hizo para la norma uno.

   - Desigualdad triangular. Sean $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ dos vectores cualesquiera. Para la tercera propiedad se va a utilizar la desigualdad de *Cauchy-Schwarz*

$$\sum_{i=1}^{n} x_i y_i \leq \sqrt{\sum_{i=1}^{n} x_i^2} \sqrt{\sum_{i=1}^{n} y_i^2}.$$

     A partir de ello se obtiene que

$$\sum_{i=1}^{n} x_i^2 + 2\sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} y_i^2 \leq \sum_{i=1}^{n} x_i^2 + 2\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} y_i^2} + \sum_{i=1}^{n} y_i^2,$$

     lo que equivale a

$$\|\boldsymbol{x}+\boldsymbol{y}\|_2^2 \leq (\|\boldsymbol{x}\|_2 + \|\boldsymbol{y}\|_2)^2.$$

     Extrayendo la raíz cuadrada a ambos lados, se demuestra la propiedad.

■

Existen relaciones entre las normas uno dos e infinito que nos permiten comparar sus valores sabiendo el valor de una sola de estas normas. Estas relaciones se presentan en el siguiente teorema.

Teorema 1.10. *Para cualquier vector $\boldsymbol{x} \in \mathbb{R}^n$, se cumplen las siguientes relaciones*

a. $\|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_1 \leq n\|\boldsymbol{x}\|_\infty$.

b. $\frac{1}{n}\|\boldsymbol{x}\|_1 \leq \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_1$.

c. $\|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty$.

d. $\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2$.

e. $\frac{1}{\sqrt{n}}\|\boldsymbol{x}\|_1 \leq \|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1$.

Demostración. Para la demostración se considera un vector $\boldsymbol{x} \in \mathbb{R}^n$ y además $\|x\|_\infty = \text{máx}_{i=1,2,\ldots,n}\,|x_i| = |x_k|$ para algún $k = 1, 2, \ldots, n$.

a. $\|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_1 \leq n\|\boldsymbol{x}\|_\infty$.

Para la primera inecuación se observa que

$$|\boldsymbol{x}_k| = \|x\|_\infty \leq \boldsymbol{x}_1 + \boldsymbol{x}_2 + \ldots + \boldsymbol{x}_k + \ldots + \boldsymbol{x}_n = \|x\|_1$$

y ademas,

$|x_1| \leq |x_k|,\ |x_2| \leq |x_k|,\ \ldots,\ |x_n| \leq |x_k|$, entonces

$$\sum_{i=1}^{n} |x_i| = \|\boldsymbol{x}\|_1 \leq n\|\boldsymbol{x}\|_\infty.$$

Cómo $\|x\|_\infty \leq \|\boldsymbol{x}\|_1$ y $\|\boldsymbol{x}\|_1 \leq n\|x\|_\infty$ se obtiene la primera relación.

b. $\frac{1}{n}\|\boldsymbol{x}\|_1 \leq \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_1$.

Se observa claramente que esta relación se obtiene a partir de la relación anterior.

c. $\|\boldsymbol{x}\|_\infty \leq \boldsymbol{x}_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty$.

Para este caso se utiliza el hecho de que

$$\sqrt{a} \leq \sqrt{a+b}, \quad \forall\ a, b \geq 0.$$

A partir de esto,

$$|x_k| = \sqrt{x_k^2} = \|x\|_\infty \leq \sqrt{x_1 + x_2 + \ldots + x_k^2 + \ldots + x_n^2} = \|x\|_2$$

y por definición de máximo

$$x_i^2 \leq x_k^2,\ i = 1, 2, \ldots, n$$

de donde,

$$\sum_{i=1}^{n} x_i^2 = \|\boldsymbol{x}\|_2^2 \leq n x_k^2 = n\|\boldsymbol{x}\|_\infty^2$$

y extrayendo la raíz cuadrada a ambos lados, se obtiene que

$$\|\boldsymbol{x}\|_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty.$$

Cómo $\|\boldsymbol{x}\|_\infty \leq \|x\|_2$ y $\|\boldsymbol{x}\|_2 \leq \sqrt{n}\|\boldsymbol{x}\|_\infty$ se demuestra la desigualdad.

d. $\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2$

Para demostrar esta relación, se va a probar primero que $\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1$ por medio del principio de inducción sobre n, que es la longitud del vector $\boldsymbol{x}$.
Si $n = 1$, $\boldsymbol{x} = (x_1)$, entonces

$$\|\boldsymbol{x}\|_2 = \sqrt{x_1^2} = |x_1| = \|x\|_1,$$

es decir,

$$\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1.$$

Suponiendo que se cumple la anterior desigualdad para $n = 1, 2, \ldots, k$ se va a determinar que se cumple para $n = k + 1$.

Si $n = k + 1$, $\boldsymbol{x} = (\boldsymbol{x}^*, x_{k+1})$, donde $\boldsymbol{x}^* = (x_1, x_2, \ldots, x_n)$ y así,

$$\|\boldsymbol{x}\|_2^2 = \sum_{i=1}^{k} x_i^2 + x_{k+1}^2 = \|\boldsymbol{x}^*\|_2^2 + x_{k+1}^2,$$

$$\begin{aligned}\|\boldsymbol{x}\|_1^2 &= \left(\sum_{i=1}^{k} |x_i| + |x_{k+1}|\right)^2 = \left(\|\boldsymbol{x}^*\|_1 + |x_{k+1}|\right)^2 \\ &= \|\boldsymbol{x}^*\|_1^2 + 2|x_{k+1}|\|\boldsymbol{x}^*\|_1 + x_{k+1}^2.\end{aligned}$$

Pero por la hipótesis de inducción se tiene que

$$\|\boldsymbol{x}^*\|^2 \leq \|\boldsymbol{x}^*\|_1^2$$

y además

$$x_{k+1}^2 \leq +2|x_{k+1}|\|\boldsymbol{x}^*\|_1 + x_{k+1}^2.$$

Entonces,

$$\|\boldsymbol{x}^*\|^2 + x_{k+1}^2 = \|\boldsymbol{x}\|_2^2 \leq \boldsymbol{x}^{*2}_1 + 2|x_{k+1}|\|\boldsymbol{x}^*\|_1 + x_{k+1}^2 = \|\boldsymbol{x}\|_1^2$$

y extrayendo la raíz cuadrada de la desigualdad se cumple la relación. Por el principio de inducción

$$\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1, \ \forall \ \boldsymbol{x} \in \mathbb{R}^n.$$

Ahora se va a probar que $\|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|^2$. Aplicando nuevamente la desigualdad

$$\sqrt{a} \leq \sqrt{a+b}, \quad \forall \ a, b \geq 0,$$

si tiene que

$$\sqrt{x_1^2} = |x_1| \leq \sqrt{\sum_{i=1}^{n} x_i^2}, \ \sqrt{x_2^2} = |x_1| \leq \sqrt{\sum_{i=1}^{n} x_i^2}, \ \ldots \ , \sqrt{x_n^2} = |x_n| \leq \sqrt{\sum_{i=1}^{n} x_i^2}$$

lo que implica que

$$|x_1| + |x_2| + \ldots + |x_n| = \|\boldsymbol{x}\|_1^2 \leq n\|\boldsymbol{x}\|_2^2$$

y extrayendo la raíz cuadrada se obtiene la desigualdad.

Cómo $\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1$ y $\|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|^2$ se verifica la relación.

e. $\frac{1}{\sqrt{n}}\|\boldsymbol{x}\|_1 \leq \|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_1$.

Se obtiene esta relación aplicando la relación probada anteriormente.

■

A continuación se muestra el concepto de norma de matrices, que sigue la misma idea que el concepto visto para la normas de vectores.

Definición 1.35. (Norma matricial). Una norma matricial $|||\ .\ |||$ para matrices reales cuadradas es una función

$$|||\ .\ |||\colon \mathbb{R}^{n\times n} \to \mathbb{R},$$

que cumple las siguientes propiedades:

1. *Definida positiva.* Para todo $A \neq 0 \in \mathbb{R}^{n\times n}$

$$|||\ A\ ||| > 0,$$

y para $A = 0$

$$|||\ A\ ||| = 0.$$

2. *Homogeneidad.* Para cualquier matriz $A \in \mathbb{R}^{n\times n}$ y un escalar $\alpha \in \mathbb{R}^n$

$$|||\ \alpha A\ ||| = |\ \alpha\ |\,|||\ A\ |||\ .$$

3. *Desigualdad triangular.* Para cualquier matriz A y $B \in \mathbb{R}^{n\times n}$

$$|||\ A+B\ |||\leq |||\ A\ ||| + |||\ B\ |||\ .$$

4. *Consistencia.* Para cualquier matriz A y $B \in \mathbb{R}^{n\times n}$

$$|||\ AB\ |||\leq |||\ A\ ||||||\ B\ |||\ .$$

A partir de esta definición una de las normas matriciales frecuentemente usadas en el área de álgebra lineal numérica es la norma de Frobenius, que se define como

$$|||\ A\ |||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2}$$

y la norma inducida

$$|||\ A\ ||| = \max_{\boldsymbol{x}\neq 0} \frac{\|\ A\boldsymbol{x}\ \|}{\|\ \boldsymbol{x}\ \|},$$

donde $\|.\|$ es una norma vectorial.

Se va a demostrar que estas funciones son efectivamente normas matriciales.

Demostración.

1. Norma de Frobenius $|||\ A\ |||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2}$.

   Para las propiedades definida positiva, homogénea y la desigualdad triangular la demostración se realiza de forma análoga a la realizado en las demostraciones para las normas matriciales uno y dos.

   Para la propiedad de consistencia, por medio de la desigualdad de Cauchy-Schwarz se tiene que para cualquier matriz A y $B \in \mathbb{R}^{n\times n}$

$$\begin{aligned}\|AB\|_F^2 &= \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\sum_{k=1}^{n} a_{kj}b_{kj}\right)^2 \leq \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} a_{ik}^2 \sum_{k=1}^{n} b_{kj}^2 \\ &\leq \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik}^2 \sum_{j=1}^{n}\sum_{k=1}^{n} b_{kj}^2 = \|A\|_F^2\|B\|_F^2\end{aligned}$$

   y extrayendo la raíz cuadrada a ambos lados de la desigualdad se obtiene que

$$\|AB\|_F \leq \|A\|_F\|B\|_F$$

2. Norma inducida

$$||| A |||= \max_{\boldsymbol{x}\neq 0} \frac{\| A\boldsymbol{x} \|}{\| \boldsymbol{x} \|},$$

donde $\|.\|$ es una norma vectorial.

- Definida positiva. Se puede notar que para cualquier $A \neq 0 \in \mathbb{R}^{n\times n}$ existe al menos un $a_{ij} \neq 0$ para algún $i = 1, 2, \ldots, n$ y $j = 1, 2, \ldots, n$. Entonces, si se toma un vector $\boldsymbol{x} \in \mathbb{R}^n$ tal que en la posición $x_j \neq 0$ y para las demás posición $x_k = 0$ con $k \neq j$ nos garantiza que en la posición j-ésima del vector $A\boldsymbol{x}$ es distinto cero y por definición de norma vectorial se cumple que

$$\frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} > 0.$$

A partir de esto y por la definición de máximo

$$||| A |||= \max_{y\neq 0} \frac{\|A\boldsymbol{y}\|}{\|\boldsymbol{y}\|} > \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} > 0.$$

En el caso de que la matriz $A = 0$

$$||| A |||=||| 0 |||= \max_{x\neq 0} \frac{\|0\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \max_{x\neq 0} \frac{\|0\|}{\|\boldsymbol{x}\|} = 0.$$

- Homogeneidad. Sean $A \in \mathbb{R}^{n\times n}$ y $\alpha \in \mathbb{R}$. Por la propiedad de homogeneidad en las normas vectoriales

$$\forall\ \boldsymbol{x} \in \mathbb{R}^n \|\alpha\boldsymbol{x}\| = |\alpha|\, \|\boldsymbol{x}\|,$$

entonces se tiene que,

$$\begin{aligned} ||| \alpha A ||| &= \max_{\boldsymbol{x}\neq 0} \frac{\|\alpha A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = \max_{\boldsymbol{x}\neq 0} |\alpha| \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \\ &= |\alpha| \max_{\boldsymbol{x}\neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} = |\alpha|\ ||| A |||\ . \end{aligned}$$

- Desigualdad triangular. Sean $A, B \in \mathbb{R}^{n\times n}$ dos matrices cualesquiera. Por la desigualdad triangular en las normas vectoriales se tiene que

$$\forall\ \boldsymbol{x} \in \mathbb{R}^n, \quad \|(A + B)\boldsymbol{x}\| = \|A\boldsymbol{x} + B\boldsymbol{x}\| \leq \|A\boldsymbol{x}\| + \|B\boldsymbol{x}\|.$$

Con este resultado

$$\begin{aligned} ||| A + B ||| &= \max_{\boldsymbol{x}\neq 0} \frac{\|(A + B)\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \max_{\boldsymbol{x}\neq 0} \frac{\|A\boldsymbol{x}\| + \|B\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \\ &\leq \max_{\boldsymbol{x}\neq 0} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} + \max_{\boldsymbol{x}\neq 0} \frac{\|B\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \\ &\leq ||| A ||| + ||| B |||\ . \end{aligned}$$

- Consistencia. Por la definición de norma inducida, para cualquier vector $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$ y cualquier matriz $C \in \mathbb{R}^{n\times n}$

$$||| \ C \ |||= \max_{\boldsymbol{y}\neq 0} \frac{\|C\boldsymbol{y}\|}{\|\boldsymbol{y}\|} > \frac{\|C\boldsymbol{x}\|}{\|x\|},$$

es decir,

$$||| \ C\boldsymbol{x} \ ||| \ \| \leq \| \ ||| \ C \ ||| \ \|C\boldsymbol{x}\|.$$

De este modo, para dos matrices A y $B \in \mathbb{R}^{n\times n}$

$$||| \ A(B\boldsymbol{x}) \ ||| \ \| \leq \| \ ||| \ A \ ||| \ \|A(B\boldsymbol{x})\| \leq ||| \ A \ ||| ||| \ B \ ||| \ \|\boldsymbol{x}\|$$

y en consecuencia

$$\frac{\|AB\boldsymbol{x}\|}{\|x\|} \leq ||| \ A \ ||| ||| \ B \ ||| \ .$$

Dado que se cumple la anterior desigualdad para cualquier $\boldsymbol{x} \neq 0$ se concluye que

$$\max_{\boldsymbol{x}\neq 0} \frac{||| \ AB\boldsymbol{x} \ |||}{\|x\|} = ||| \ AB \ ||| \leq ||| \ A \ ||| ||| \ B \ ||| \ .$$

■

Observación 1.3. Sobre la definición de norma inducida, se puede observar que para cualquier $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$ se cumple que, $\left|\left|\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}\right|\right| = 1$, entonces se puede definir la norma inducida como

$$||| \ A \ ||| = \max_{\boldsymbol{x}\neq 0} \frac{\| \ A\boldsymbol{x} \ \|}{\| \ \boldsymbol{x} \ \|} = \max_{\boldsymbol{x}\neq 0} \left|\left| A(\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}) \right|\right| = \max_{\|\boldsymbol{y}\|=1} \|A\boldsymbol{y}\|.$$

También para el caso de la norma de frobenius se puede definir para cualquier matriz $A \in \mathbb{R}^{n\times n}$ como

$$||| \ A \ |||_F = \sqrt{\sum_{i=1}^{} \sum_{j=1}^{} a_{ij}^2} = \sqrt{\sum_{i=1}^{n} \|Col_i A\|_2^2} = \sqrt{\sum_{i=1}^{n} \|Row_i A\|_2^2}.$$

Una de las características del las normas vectoriales $p = 1, 2, \infty$, aplicadas en las normas inducidas p, es que satisfacen ciertas igualdades, que nos permiten calcular su norma de forma mas sencilla y directa. El teorema que muestra estas igualdades es el siguiente.

Teorema 1.11. *Para cualquier matriz $A \in \mathbb{R}^{n\times n}$ se cumplen las siguientes siguientes igualdades*

1. *Norma inducida uno* $||| \ A \ |||_1 = \max_{i=1,\ldots,n} \| \ Row_i A \ \|$.
2. *Norma inducida infinito* $||| \ A \ |||_\infty = \max_{i=1,\ldots,n} \| \ Col_i A \ \|$.
3. *Norma inducida dos* $||| \ A \ |||_2 = \sqrt{\rho(A^T A)}$.

Demostración.

1. Norma inducida uno $\|\!| A |\!\|_1 = \max\limits_{i=1,\ldots,n} \| Col_i A \|_1$.

   Sean $A \in \mathbb{R}^{n\times n}$ y $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$. Realizando el producto matriz por vector

   $$A\boldsymbol{x} = \begin{bmatrix} a_{11}x_1 + a_{12}x_2 + \ldots a_{1n}x_n \\ a_{21}x_1 + a_{22}x_2 + \ldots a_{2n}x_n \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots a_{nn}x_n \end{bmatrix} = x_1 Col_1 A + x_2 Col_2 A + \ldots + x_n Col_n A.$$

   A partir de esto se tiene que

   $$\begin{aligned} \frac{\|A\boldsymbol{x}\|_1}{\|\boldsymbol{x}\|_1} &= \frac{\|x_1 Col_1 A + x_2 Col_2 A + \ldots + x_n Col_n\|_1}{\|\boldsymbol{x}\|_1} \\ &\leq \frac{|x_1|\|Col_1 A\|_1 + |x_2|\|Col_2 A\|_1 + \ldots + |x_n|\|Col_n A\|_1}{\|\boldsymbol{x}\|_1} \quad \text{(D. triangular.)} \\ &\leq \frac{(|x_1| + |x_2| + \ldots + |x_n|)\,\text{máx}_{1\leq i\leq n}\,\|Col_i A\|_1}{\|\boldsymbol{x}\|_1} \quad \text{(Def. máximo)} \\ &\leq \frac{\|\boldsymbol{x}\|_1\,\text{máx}_{1\leq i\leq n}\,\|Col_i A\|_1}{\|\boldsymbol{x}\|_1} \quad \text{(Def. norma 1)} \\ &\leq \max_{1\leq i\leq n} \|Col_i A\|_1 = M. \end{aligned}$$

   De lo anterior M es cota superior de $\|\!| A |\!\|$. Si se encuentra un $\boldsymbol{x}$ para el cual la desigualdad anterior coincida con esta cota, se habrá demostrado la igualdad de esta norma. Para ello se toma un vector $\boldsymbol{x} \in \mathbb{R}^n$, de forma tal que,

   $$x_i = 0, \forall\ i \neq k \text{ y } x_k = 1,$$

   para un k entre 1 y n tal que k es la k-ésima componente en la cual $\|Col_k A\|_1 = M$. Por lo tanto $\|\boldsymbol{x}\|_1 = 1$ y

   $$\frac{\|Ax\|_1}{\|x\|_1} = \|\sum_{i\neq k} 0 Col_i A + Col_k A\|_1 = \|Col_k A\|_1 = \max_{1\leq i\leq n} \|Col_i A\|_1.$$

2. Norma inducida infinito $\|\!| A |\!\|_\infty = \max\limits_{i=1,\ldots,n} \| Row_i A \|$.

Sean $A \in \mathbb{R}^{n\times n}$ y $x \neq 0 \in \mathbb{R}^n$.

$$\begin{aligned}\frac{\|A\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} &= \frac{1}{\|\boldsymbol{x}\|_\infty} \max_{1\leq i\leq n} \left|\sum_{j=1}^{n} a_{ij}x_j\right| \\ &\leq \frac{1}{\|\boldsymbol{x}\|_\infty} \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}x_j| \quad (\text{ D. triangular.}) \\ &\leq \frac{1}{\|\boldsymbol{x}\|_\infty} \max_{1\leq i\leq n} \sum_{j=1}^{n} \|\boldsymbol{x}\|_\infty |a_{ij}| \quad (\text{Def. norma infinito. }) \\ &\leq \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}| = \max_{1\leq i\leq n} \|Row_i A\|_1 = M.\end{aligned}$$

Como la desigualdad anterior se cumple para cualquier vector $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$, entonces

$$\max_{1\leq i\leq n} \frac{\|A\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} \leq M.$$

Ahora, sea el vector $\boldsymbol{x}^* \in \mathbb{R}^n$ tal que $x_i^* = 1 \ \forall \ i = 1, 2, \ldots n$ así, $\|\boldsymbol{x}^*\|_\infty = 1$ y

$$\begin{aligned}\frac{\|A\boldsymbol{x}^*\|_\infty}{\|\boldsymbol{x}^*\|_\infty} &= \max_{1\leq i\leq n} \sum_{i=1}^{n} |a_{ij}x_i| \\ &= \max_{1\leq i\leq n} \sum_{i=1}^{n} |a_{ij}| \\ &= \max_{1\leq i\leq n} \|Row_i A\|_\infty = M,\end{aligned}$$

esto quiere decir que,

$$\max_{1\leq i\leq n} \frac{\|Ax\|_\infty}{\|x\|_\infty} \geq M.$$

Como $\|A\| \leq M$ y $\|A\| \geq M$, se concluye que

$$|||\ A\ |||_\infty = \max_{1\leq i\leq n} \frac{\|A\boldsymbol{x}\|_\infty}{\|\boldsymbol{x}\|_\infty} = M.$$

3. Norma inducida dos $|||\ A\ |||_2 = \sqrt{\rho(A^TA)}$.

   En primer lugar se puede observar que A^TA es simétrica y además definida positiva, dado que para cualquier $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$

   $$\boldsymbol{x}^TA^TA\boldsymbol{x} = (A\boldsymbol{x})^TA\boldsymbol{x} = \|Ax\|_2^2 > 0.$$

   Esto implica que los valores propios de A^TA son todos positivos (Teorema 1.8) y se pueden ordenar como

   $$0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n = \rho(A^TA).$$

Además de este resultado, por el Teorema 1.7 la matriz A^TA es similar a una matriz diagonal y por el Teorema 1.9 los vectores propios de A^TA forman una matriz no singular y son un conjunto de vectores linealmente independientes, implicando que formen una base en R^n. Esto quiere decir que para cualquier $\boldsymbol{x} \in \mathbb{R}^n$

$$\boldsymbol{x} = \sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i,$$

donde $\boldsymbol{v}_i$, para $i = 1, \ldots, n$, son los vectores propios ortogonales a A^TA asociados a los valores propios λ_i, $i = 1, 2 \ldots, n$. De lo anterior,

$$\begin{aligned}
\frac{\|A\boldsymbol{x}\|_2^2}{\|\boldsymbol{x}\|_2^2} = \frac{\boldsymbol{x}^T A^T A \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}} &= \frac{\left(\sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i\right)^T \left(A^TA \left(\sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i\right)\right)}{\left(\sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i\right)^T \left(\sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i\right)} \\
&= \frac{\left(\sum_{i=1}^{n} \alpha_i \boldsymbol{v}_i\right)^T \left(\sum_{i=1}^{n} \lambda_i \alpha_i \boldsymbol{v}_i\right)}{\sum_{i=1}^{n} (\alpha_i \boldsymbol{v}_i)^2} = \frac{\sum_{i=1}^{n} \lambda_i (\alpha_i \boldsymbol{v}_i)^2}{\sum_{i=1}^{n} (\alpha_i \boldsymbol{v}_i)^2} \\
&\leq \rho(A^TA) \frac{\sum_{i=1}^{n} (\alpha_i \boldsymbol{v}_i)^2}{\sum_{i=1}^{n} (\alpha_i \boldsymbol{v}_i)^2} \\
&\leq \rho(A^TA).
\end{aligned}$$

Como resultado de la inecuación anterior, $\sqrt{\rho(A^TA)}$ es una cota superior de $\max\limits_{x \neq 0} \dfrac{\|Ax\|_2^2}{\|x\|_2^2}$. Si se encuentra un vector $\boldsymbol{x}$ que coincida con esta cota, se habrá demostrado la igualdad de la norma. Para ello se toma $\boldsymbol{x} = \boldsymbol{v}_n$, donde $\boldsymbol{v}_n$ es el vector propio de A^TA asociado al máximo valor propio de A^TA, es decir $\lambda_n = \rho(A^TA)$. Con esto se tiene que

$$\begin{aligned}
\frac{\|A\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} = \frac{\|A\boldsymbol{v}_n\|_2}{\|\boldsymbol{v}_n\|_2} &= \frac{\|\rho(A^TA)\boldsymbol{v}_n\|_2}{\|\boldsymbol{v}_n\|_2} \\
&= \sqrt{\rho(A^TA)} \frac{\|\boldsymbol{v}_n\|_2}{\|\boldsymbol{v}_n\|_2} \\
&= \sqrt{\rho(A^TA)}.
\end{aligned}$$

■

Ejemplo 1.10. Para la matriz

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 2 & 3 & 2 \\ 1 & 1 & 2 \end{bmatrix}$$

se tiene $|||\ A\ |||_1 = 5$, $|||\ A\ |||_\infty = 7$ y $|||\ A\ |||_{\mathbb{F}} = 3\sqrt{3} \approx 5.19615$.
Para hallar $|||\ A\ |||_2$ se va a encontrar los valores propios asociados a A^TA, es decir, que debe existir un vector $\boldsymbol{x} \neq 0$ tal que

$$A^TA\boldsymbol{x} = \lambda\boldsymbol{x},$$

lo que implica que

$$(A^TA - I_3\lambda)\boldsymbol{x} = 0.$$

Por el teorema 1.5, el $\det(A^TA - I_3\lambda) = 0$, entonces,

$$\begin{aligned} \det \begin{bmatrix} 9-\lambda & 7 & 6 \\ 7 & 10-\lambda & 8 \\ 6 & 8 & 8-\lambda \end{bmatrix} &= (9-\lambda)\det \begin{bmatrix} 10-\lambda & 8 \\ 8 & 8-\lambda \end{bmatrix} - 7 \begin{bmatrix} 7 & 8 \\ 6 & 8-\lambda \end{bmatrix} \\ &\quad + 6 \begin{bmatrix} 7 & 10-\lambda \\ 6 & 8 \end{bmatrix} \\ &= -\lambda^3 + 27\lambda^2 - 92\lambda + 64. \end{aligned}$$

Resolviendo la ecuación

$$-\lambda^3 + 27\lambda^2 - 92\lambda + 64 = 0$$

se encuentra que los valores propios de A son $\lambda_1 \approx 23.14444$, $\lambda_2 \approx 2.90302$ y $\lambda_3 \approx 0.95254$. Por la definición de norma inducida dos $|||\ A\ |||_2 = \sqrt{\lambda_1} \approx 4.81087$.

Igual que en las normas vectoriales, se pueden relacionar normas de matrices; en este caso las normas inducidas uno dos e infinito y la norma de frobenius. Estas relaciones son muy utilizadas en el análisis computacional de matrices [19].

Teorema 1.12. *Para una matriz $A \in \mathbb{R}^{n\times n}$ se cumplen las siguientes relaciones*

1. $|||\ A\ |||_2 \leq \|A\|_F \leq \sqrt{n}\ |||\ A\ |||_2$.
2. $\max\limits_{1\leq i,j\leq n} |a_{ij}| \leq |||\ A\ |||_2 \leq n \max\limits_{1\leq i,j\leq n} |a_{ij}|$.
3. $\frac{1}{\sqrt{n}}\ |||\ A\ |||_\infty \leq |||\ A\ |||_2 \leq \sqrt{n}\ |||\ A\ |||_1$.
4. $\frac{1}{\sqrt{n}}\ |||\ A\ |||_1 \leq |||\ A\ |||_2 \leq \sqrt{n}\ |||\ A\ |||_1$.

Demostración.
Sea $A \in \mathbb{R}^{n\times n}$ y $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$ tal que, $\|\boldsymbol{x}\|_p = 1$, para $p = 1, 2, \infty$. Se Denota por

$$|a_{cd}| = \max_{1\leq i,j\leq n} |a_{ij}|,$$

para algún $1 \leq c, d \leq n$, luego

$$\|Col_k A\| = \max_{1\leq i\leq n} \|Col_i A\| \text{ y } \|Col_k A\| \max_{1\leq i\leq n} \|Row_k A\|$$

para algún $1 \leq k \leq n$.

1. $|||\ A\ |||_2 \leq \|A\|_F \leq \sqrt{n}\ |||\ A\ |||_2$.

   El producto $A\boldsymbol{x}$ se lo puede expresar como

   $$A\boldsymbol{x} = \begin{bmatrix} Row_1 A\boldsymbol{x} \\ Row_2 A\boldsymbol{x} \\ \vdots \\ Row_n A\boldsymbol{x} \end{bmatrix}.$$

   Ahora, dado que para cualquier vector $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ se cumple que $< \boldsymbol{x}, \boldsymbol{y} > \leq \|\boldsymbol{x}\|\|\boldsymbol{y}\|$ entonces,

   $$\sum_{i=1}^{n} (Row_i A\boldsymbol{x})^2 = |||\ A\ |||_2^2 \leq \sum_{i=1}^{n} \|Row_i A\|^2 \|\boldsymbol{x}\|^2 = |||\ A\ |||_F^2,$$

   es decir,

   $$|||\ A\ |||_2 \leq |||\ A\ |||_F\ .$$

   Para demostrar que $\|A\|_F \leq \sqrt{n}\ |||\ A\ |||_2$, si se toma $\boldsymbol{x} = \boldsymbol{e}_k$ (Ejemplo 1.7), se tiene

   $$\|A\boldsymbol{e}_k\|_2 = \|\sum_{i=1}^{n} Col_i A e_k(i)\|_2^2 = \|Col_k A\|_2^2,$$

   por lo tanto,

   $$|||\ A\ |||_F^2 = \sum_{i=1}^{n} \|Col_i A\|_2^2 \leq n\|Col_k A\|_2^2 \leq n\ |||\ A\ |||_2^2,$$

   con lo cual

   $$\|A\|_F \leq \sqrt{n}\ |||\ A\ |||_2\ .$$

2. $\max\limits_{1\leq i,j\leq n}|a_{ij}| \leq ||| A |||_2 \leq n \max\limits_{1\leq i,j\leq n}|a_{ij}|$.

   Para probar que $\max\limits_{1\leq i,j\leq n}|a_{ij}| \leq ||| A |||_2$, se puede observa que

   $$\|A\boldsymbol{e}_i\|_2 = \|Col_i A\|_2,$$

   para cada $i = 1, 2, \ldots, n$, entonces

   $$|a_{cd}| = \sqrt{a_{cd}^2} \leq \sqrt{\sum_{i=1}^{n} a_{ci}^2} = \|Col_c A\|_2 = ||| Ae_e |||\leq ||| A |||_2 .$$

   Para probar el caso $|a_{ij}| \leq ||| A |||_2 \leq n \max\limits_{1\leq i,j\leq n}|a_{ij}|$, se va aplicar la desigualdad triangular en las normas vectoriales, así,

   $$\|A\boldsymbol{x}\|_2 = \|\sum_{i=1}^{n} Col_i A x_i\| \leq \sum_{i=1}^{n} \|Col_i A\|_2 |x_i| \leq \|Col_k A\|_2 \|\boldsymbol{x}\|_1.$$

   Ahora por el teorema 1.10 en la desigualdad anterior se obtiene que,

   $$\begin{aligned}\|A\boldsymbol{x}\|_2 &\leq \|Col_k A\|_2 \|\boldsymbol{x}\|_1 \leq \sqrt{n}\|\boldsymbol{x}\|_2 \|Col_k A\|_1 \\ &\leq n\sqrt{n}|a_{cd}| \\ &\leq n^2|a_{cd}|.\end{aligned}$$

   Como se cumple la anterior desigualdad para cualquier $\boldsymbol{x} \neq 0$ entonces,

   $$||| A |||_2 \leq n|a_{cd}| = n \max_{1\leq i,j\leq n}|a_{ij}|.$$

3. $\frac{1}{\sqrt{n}} ||| A |||_\infty \leq ||| A |||_2 \leq \sqrt{n} ||| A |||_1$.

   Por las propiedades de normas vectoriales y matriciales y por el teorema 1.11 se obtiene

   $$\begin{aligned}\frac{||| A |||_\infty}{\sqrt{n}} = \frac{\|Row_k A\|_1}{\sqrt{n}} \leq \frac{\sqrt{n}\|Row_k A\|_2}{\sqrt{n}} &= \|A^T \boldsymbol{e}_k\|_2 \\ \leq ||| A^T |||_2 \, \|\boldsymbol{e}_k\|_2 \leq ||| A^T |||_2 &= ||| A |||_2 .\end{aligned}$$

   Se cumple que $||| A^T |||_2 = ||| A |||_2$, ya que se puede probar usando resultados anteriores que

   $$||| A |||_2 = \max\{|\boldsymbol{x}^T A \boldsymbol{y}| : \|\boldsymbol{x}\|_2 = \|\boldsymbol{y}\|_2 = 1\} = ||| A^T |||_2 .$$

   Para el caso $||| A |||_2 \leq \sqrt{n} ||| A |||_1$ se puede ver que,

   $$\begin{aligned}\|A\boldsymbol{x}\|_2 = \|\sum_{i=1}^{n} Col_i A x_i\|_2 &\leq \sum_{i=1}^{n} \|Col_i A\|_2 |x_i| \\ &\leq \max_{i=1,\ldots,n} \|Col_i A\|_2 \|\boldsymbol{x}\|_1 \leq \sqrt{n}\, \|\boldsymbol{x}\|_2 \, ||| A |||_1 \\ &\leq \sqrt{n} \, ||| A |||_2 .\end{aligned}$$

Dado que se cumple la desigualdad anterior para cualquier $\boldsymbol{x}$, se obtiene que

$$||| A |||_2 \leq \sqrt{n} \, ||| A |||_1 .$$

4. $\frac{1}{\sqrt{n}} \, ||| A |||_\infty \leq ||| A |||_2 \leq \sqrt{n} \, ||| A |||_1$.

   Como ya se probó anteriormente que $||| A |||_2 \leq \sqrt{n} \, ||| A |||_1$, falta probar que

$$\frac{1}{\sqrt{n}} \, ||| A |||_\infty \leq ||| A |||_2 .$$

   Como se cumple para cualquier vector $\boldsymbol{x} \in \mathbb{R}^n$ que

$$\begin{aligned} \|A\boldsymbol{x}\|_1 = \| \sum_{i=1}^{n} Col_i A x_i \|_1 &\leq \sqrt{n} \| \sum_{i=1}^{n} Col_i A x_i \|_2 \\ &\leq \sqrt{n} \, ||| A |||_2, \end{aligned}$$

   implica que $||| A |||_1 \leq \sqrt{n} \, ||| A |||_2$, es decir, $\frac{1}{\sqrt{n}} \, ||| A |||_1 \leq ||| A |||_2$.

■

Ejemplo 1.11. Retomando la matriz A del ejemplo anterior, se pudo observar lo dispendioso y el costo computacional que tiene al calcular $||| A |||_2$. Sin embargo con el teorema anterior se puede saber en que intervalo se encuentra esta norma. En este caso

$$\begin{gathered} ||| A |||_2 \leq ||| A |||_F = 5,19615, \\ \max_{1 \leq i,j \leq 3} |a_{ij}| = 3 \leq ||| A |||_2 \leq 3 \max_{1 \leq i,j \leq 3} |a_{ij}| = 9. \\ \frac{1}{\sqrt{3}} \, ||| A |||_\infty = 4.04145 \leq ||| A |||_2 \leq \sqrt{n} \, ||| A |||_\infty = 12.12435, \\ \frac{1}{\sqrt{3}} \, ||| A |||_1 = 2.88675 \leq ||| A |||_2 \leq \sqrt{n} \, ||| A |||_1 = 8.660254. \end{gathered}$$

El intervalo mas pequeño que se encuentra $||| A |||_2$ con respecto a las anteriores inecuaciones es $(4.04145, 5, 19615)$ y el valor que se había obtenido era de $||| A |||_2 = 4, 81087$, que claramente se encuentra en el intervalo.

Capítulo 2

Matrices Esparzas

En este capítulo se darán a conocer algunos aspectos generales acerca del uso de estructuras de datos eficientes para trabajar computacionalmente con matrices esparzas. Se destaca en las estructuras de datos propuestas el funcionamiento, las ventajas y desventajas, su implementación y por la necesidad de aplicaciones la eficiencia en operaciones matriciales de alto costo computacional como son el producto matriz por vector, el producto matriz transpuesta por vector y el cálculo de la matriz transpuesta, resaltando que se han propuesto algoritmos usando dichas estructuras para las operaciones mencionadas. Cabe aclarar que para ejemplificar cada tema, en el transcurso del capítulo se ha requerido del uso de de matrices de pequeñas dimensiones, aunque en la práctica las matrices son de gran tamaño.

Definición 2.1. (Matriz esparza). Una matriz $A \in \mathbb{R}^{m\times n}$ se puede definir como *matriz esparza*, si tiene una cantidad considerable de elementos nulos respecto al número de entradas de la matriz. El número de elementos no nulos de una matriz esparza se denota por nnz (no number zero). El porcentaje de elementos nulos de una matriz esparza se le conoce como la *esparcidad de la matriz* y el porcentaje de elementos no nulos o diferentes de cero de la matriz se denomina la *densidad de la matriz*.

Ejemplo 2.1. Sean las matrices

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ -2 & 0 & 5 & 0 & 0 \\ 3 & 0 & 7 & 8 & 0 \\ 0 & 0 & 6 & 5 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{y} \quad B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 \end{bmatrix}.$$

Se tiene que son matrices esparzas, dado que la matriz A tiene 16 elementos nulos y 9 elementos no ceros, con respecto al número de entradas de la matriz que es 25; mientras que la matriz B tiene 30 elementos, de los cuales 25 son nulos y solo 5 son diferentes de cero. Lo anterior nos índica que

la matriz A tiene una esparcidad del 64 % y una densidad del 36 %, mientras que la matriz B tiene una esparcidad del 83.33 % y una densidad del 16.67 %.

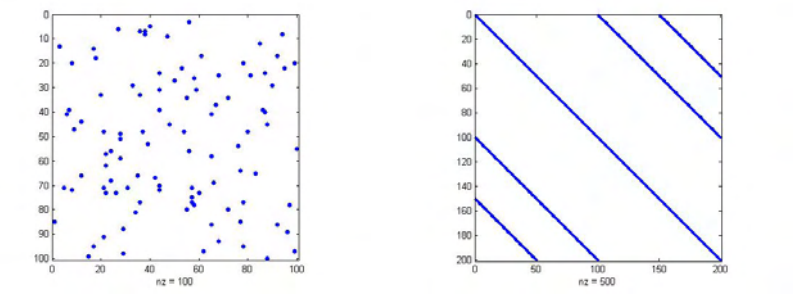
Definición 2.2. (Matriz estructurada y no estructurada). Dependiendo del comportamiento de los elementos no nulos de una matriz esparza, puede ser definida como *matriz esparza estructurada*, si existe un patrón de distribución de sus elementos no nulos, o en caso contrario, como *matriz esparza no estructurada*.

Ejemplo 2.2. Además de las matrices A y B del ejemplo anterior, también la matriz

$$C = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 & 0 \\ 0 & 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & 0 & -1 & 4 \end{bmatrix}$$

es una matriz esparza. Del ejemplo 2.1, se puede observar que la matriz A no posee un patrón de distribución de sus elementos no nulos, por lo tanto es una matriz no estructurada. En cambio, las matrices B y C son matrices estructuradas, puesto que los elementos diferentes de cero están distribuidos en forma diagonal. La estructura de la matriz B se conoce como matriz diagonal y la presentada por la matriz C es conocida como matriz de matriz de banda.

Ejemplo 2.3. En la figura 2.1, se tiene los gráficos de la distribución de los elementos no nulos de dos matrices, donde los elementos diferentes de cero de la matriz se presentan en color y los nulos son blancos. En ellos se puede observar que la matriz (a) según el patrón de distribución de sus elementos, es una matriz esparza no estructurada, mientras que la matriz (b) dado que tiene únicamente elementos no nulos distribuidos en forma diagonal, es una matriz esparza estructurada.



(a) Matriz estructurada. (b) Matriz no estructurada.

Figura 2.1: Gráficos de esparcidad. Matriz estructurada y no estructurada.

Debido a la aplicabilidad que poseen las matrices esparzas, se ve la necesidad de crear algoritmos, que facilitan el trabajo con dichas matrices, usualmente para la realización de operaciones matriciales. En este sentido es necesario conocer el número de operaciones que se efectúan para poder buscar otras alternativas o comparar la implementación de otros algoritmos sin necesidad de ejecutarlos para obtener mejores resultados.

Una de las alternativas para observar el número de operaciones de un algoritmo es por medio del "orden de complejidad", que representa de forma significativa y más sencilla dichas operaciones.

Definición 2.3. (Orden de complejidad). Sea $g(n)$ una función que determina el número de operaciones de un algoritmo. Se dice que un algoritmo tiene orden de complejidad denotado por $\mathrm{O}(f(n))$, si existen enteros n_0 y c, con $c > 0$, tal que para todo $n \geq n_0$,

$$g(n) \leq cf(n).$$

Ejemplo 2.4. Para obtener el resultado al evaluar un número en el polinomio de segundo grado

$$p_2(n) = 1 + n + 2n^2,$$

se deben realizar dos sumas y tres multiplicaciones. Además, es posible probar que para $n > 2$ se cumple que $1 + n + n^2 \leq 4n^2$, es decir que $p_2(n)$ es de orden $\mathrm{O}(n^2)$.

Así mismo, se puede verificar que el orden para un polinomio de grado k

$$p_k(n) = \alpha_0 + \alpha_1 n + \ldots + \alpha_k n^k$$

es $\mathrm{O}(n^k)$. En este caso para obtener el resultado de evaluar algún valor, es necesario realizar k sumas y $\sum_{i=0}^{k+1}(k-i) = \frac{k(k+1)}{2} = \frac{k^2}{2} + \frac{k}{2}$ multiplicaciones. Igual que para $p_2(n)$, se puede encontrar un $c > \alpha_k$ para el cual $p_k(n) \leq cg(n) = cn^k$ para todo $n > n_0$, dado que cuando c es demasiado grande cn^k tiende a crecer mas rápido que $p_k(n)$ y por ende $p_k(n)$ es de orden $\mathrm{O}(n^k)$.

Entre los ordenes de complejidad $\mathrm{O}(f(n))$ a partir de una función $f(n)$ más utilizadas se tiene los siguientes:

- $\mathrm{O}(1)$ orden constante.
- $\mathrm{O}(\log_n)$ orden logarítmico.
- $\mathrm{O}(n)$ orden lineal.
- $\mathrm{O}(n^2)$ orden cuadrático.

2.1. Esquemas de almacenamiento para matrices esparzas

Un esquema de almacenamiento para matrices esparzas se puede definir como una estructura de datos que almacena los elementos no nulos de una matriz esparza, con el fin de aprovechar su esparcidad, para poder tanto ahorrar memoria computacional en el almacenamiento, como realizar de forma eficiente operaciones matriciales de alto costo computacional donde se utilicen dichas matrices, como es la operación matriz por vector, la transpuesta, entre otras.

Entre algunos esquemas de almacenamiento que se encuentran en la literatura se tiene los siguientes

2.1.1. Almacenamiento AIJ

Conocido también como el esquema coordinado. En este almacenamiento se consideran tres vectores: $\boldsymbol{AA}$ que es la que almacena los elementos no nulos, $\boldsymbol{JA}$ y $\boldsymbol{IA}$ la fila que almacena el índice de la columna y fila de los elementos no nulos de A respectivamente.

De la matriz A del ejemplo 2.1 su almacenamiento **AIJ** es

$\boldsymbol{AA} =$

1	-2	5	3	7	8	6	5	1

$\boldsymbol{IA} =$

1	2	2	3	3	3	4	4	5

$\boldsymbol{JA} =$

4	1	3	1	3	4	3	4	1

En este almacenamiento no es necesario ubicar los elementos no nulos en orden, sin embargo el orden por fila ayuda a realizar una búsqueda de los elementos con mayor rapidez.

Las ventajas que posee este esquema son su sencillez y su flexibilidad. Es por eso que los paquetes de software lo suelen utilizar como formato para el ingreso de matrices esparzas [26].

2.1.2. Almacenamiento CSR

Por sus siglas significa “Compressed Sparse Row”, en este almacenamiento se evita almacenar directamente el índice de las filas. Para este almacenamiento se utilizan 3 vectores:

- El vector $\boldsymbol{AA}$ almacena el número de elementos no nulos de A
- El vector $\boldsymbol{JA}$ guarda el índice de columna de cada elemento no nulo de $\boldsymbol{AA}$
- El vector $\boldsymbol{nnz_i}$ almacena la frecuencia acumulada de elementos no nulos por cada fila, iniciando con $nnz_i(1)$=0 y $nnz_i(i+1)$ el número de elementos no nulos desde la fila 1 hasta la fila i-ésima para $i = 2, 3, \ldots, n+1$. Al vector $\boldsymbol{nnz_i}$ también se lo conoce como el *puntero de filas*, puesto

que nos permite ubicar en que posiciones de $\boldsymbol{JA}$ y $\boldsymbol{AA}$ está la información de los índices de columna y el valor de las entradas respectivamente de una fila determinada.

El almacenamiento **CSR** para la matriz A es el siguiente

$$\boldsymbol{AA} = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 1 & -2 & 5 & 3 & 7 & 8 & 6 & 5 & 1 \\ \hline \end{array}$$

$$\boldsymbol{JA} = \begin{array}{|c|c|c|c|c|c|c|c|c|} \hline 4 & 1 & 3 & 1 & 3 & 4 & 3 & 4 & 1 \\ \hline \end{array}$$

$$\boldsymbol{nnz_i} = \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 3 & 6 & 8 & 9 \\ \hline \end{array}$$

Debido a la forma como se almacena este formato es necesario ir almacenando los elementos por orden de fila, aunque no implica seguir el orden de elementos no nulos por columna, pero si se realiza el orden tanto de fila y columna permite mayor eficiencia en la búsqueda de elementos en el almacenamiento. También este esquema es más usado que el esquema coordinado, puesto que a menudo es más útil para realizar operaciones matriciales usuales [26].

Se conoce a este esquema como **CSC** "Compressed Sparse Column" si se almacena la matriz por orden de columnas siguiendo este procedimiento.

2.1.3. Almacenamiento CSR mod

Por sus siglas significa "Compressed Sparse Row Modified". Para este almacenamiento se utilizan dos listas de vectores AA, JA y una fila nnz_i. El funcionamiento de cada uno es el siguiente.

- los vectores de AA almacena los elementos no nulos por cada fila. En el caso de que una fila no tenga elementos nulos, el vector asociado a dicha fila, queda vacío.
- Los vectores de JA almacenan el índice de columna de los elementos no nulos por cada filas. Si no hay elementos en una fila, el vector asociado a esta, queda vacío.
- El vector $\boldsymbol{nnz_i}$ almacena el número de elementos no nulos por cada fila.

El almacenamiento de la matriz A es

$$AA = \begin{array}{|c|} \hline 1 \\ \hline \end{array} \begin{array}{|c|c|} \hline -2 & 5 \\ \hline \end{array} \begin{array}{|c|c|c|} \hline 3 & 7 & 8 \\ \hline \end{array} \begin{array}{|c|c|} \hline 6 & 5 \\ \hline \end{array} \begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

$$JA = \begin{array}{|c|} \hline 4 \\ \hline \end{array} \begin{array}{|c|c|} \hline 1 & 3 \\ \hline \end{array} \begin{array}{|c|c|c|} \hline 1 & 3 & 4 \\ \hline \end{array} \begin{array}{|c|c|} \hline 3 & 4 \\ \hline \end{array} \begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

$$\boldsymbol{nnz_i} = \begin{array}{|c|c|c|c|c|} \hline 1 & 2 & 3 & 2 & 1 \\ \hline \end{array}$$

Igual que en el **CSR**, almacenar los elementos no nulos por orden de fila y por columna permite acceder de mejor forma los almacenamientos, aunque no es obligatorio seguir el orden de columnas.

Una de las ventajas del **CSR mod** es que facilita en tiempo constante la búsqueda de cada fila de la matriz y así se realiza en forma eficiente el producto de matriz por vector [9].

Se conoce a este esquema como **CSC mod** "Compressed Sparse Column modified" si se almacena la matriz por orden de columnas siguiendo este procedimiento.

2.1.4. Formato CSV

El formato Compressed Sparse Vector (**CSV**) está compuesto por dos vectores $\boldsymbol{AA}$ y $\boldsymbol{JA}$, donde en el primera se almacenan los elementos no nulos por orden de fila y en el segundo se almacena, teniendo en cuenta los siguientes casos:

1. Si $a_{ij} = 0$, se le asigna el índice 1 y partir de este, se cuenta el número de posiciones hasta encontrar un elemento no nulo y se almacena dicha cuenta. A partir de este elemento nulo, nuevamente se asigna el índice 1 y se cuenta el número de posiciones para encontrar el siguiente elemento no nulo. Este proceso continua hasta almacenar todos los valores asociados a los elementos no nulos.
2. Si $a_{ij} \neq 0$, se almacena en la primera posición el número 1 y para las demás posiciones se sigue la idea del caso uno.

El almacenamiento **CSV** asociado a la matriz A es

$\boldsymbol{AA} =$

1	-2	5	3	7	8	6	5	1

$\boldsymbol{JA} =$

4	2	2	3	2	1	4	1	2

El orden de almacenamientos en el CSV debe realizarse por orden filas y de columnas, dado que se necesita saber las distancias entre elementos no ceros siguiendo dicho orden.

Las ventajas que posee este formato de almacenamiento, es que permite un menor volumen de almacenado como también el cálculo de forma rápida de la matriz transpuesta con un menor costo computacional [15].

2.1.5. Almacenamiento MSR

El formato Modified Sparse Row (**MSR**) cuenta con dos vectores:

- El vector $\boldsymbol{AA}$ almacena en las primeras n posiciones las entradas de la diagonal principal; la posición $n+1$ no es usada, y a partir de la posición $n+2$ se almacenan por orden de fila las demás entradas sin contar los elementos de la diagonal principal.
- El vector $\boldsymbol{JA}$ en su primera entrada ($JA(1)$) tiene un valor de $n+2$, para ubicar en la posición en que inician los elementos fuera de la diagonal. Desde esta posición hasta la posición $n+1$ se almacena la frecuencia acumulada de elementos no nulos por fila sin contar los elementos de la diagonal principal. De esta manera los $n+1$ primeros elementos de $\boldsymbol{JA}$ actúan como el puntero fila $\boldsymbol{nnz_i}$ del esquema **CSR**, pero con la diferencia de ubicar los elementos no ceros fuera de la diagonal.

Por como esta definido este formato, solo se puede utilizar cuando todos los elementos de la diagonal principal son diferentes de cero. Por tal motivo no se puede utilizar la matriz A de los ejemplos anteriores y para ilustrar el funcionamiento se considera la matriz

$$B = \begin{bmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 5 & 0 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{bmatrix}.$$

El almacenamiento **MSR** de B es

$\boldsymbol{BB} =$

1	4	7	11	12	$*$	2	3	5	6	8	9	10

$\boldsymbol{JB} =$

7	8	10	13	14	14	4	1	4	1	4	5	3

El orden del esquema **MSR** debe realizarse estrictamente almacenando por orden los elementos de la diagonal y posteriormente los elementos no nulos fuera de la diagonal por filas aunque no se exige orden por columnas.

Se puede aprovechar la potencialidad del **MSR** al trabajar con matrices esparzas estructuradas en las que los elementos sean cercanos a la diagonal principal como son por ejemplo las matrices diagonales y las matrices de banda.

Se conoce a este esquema como **MSC** "Modified Sparse Column" si se almacena la matriz por orden de columnas siguiendo este procedimiento.

2.2. Operaciones básicas con almacenamientos esparzas

Al utilizar esquemas de almacenamiento, se obtiene la información de la matriz esparza, por lo que es posible determinar la ubicación de sus elementos y permite realizar operaciones matriciales, sin necesidad de recurrir a la matriz densa. Existen diversas operaciones que se pueden aplicar, sin embargo se mencionan algunas que serán de gran relevancia en el trabajo.

Debido a que en operaciones, como la inserción y búsqueda de elementos de la matriz por medio de los almacenamientos, se necesita realizar una búsqueda de sus elementos, la búsqueda usual punto por punto o secuencial es ineficiente, puesto que podría suceder que se recorra todos los elementos una o varias veces. Uno de los métodos de búsqueda que se utilizan pueden ser secuenciales o como en nuestro caso se propone la búsqueda binaria.

La búsqueda binaría se utiliza para determinar si un elemento existe o no en una estructura de datos (como por ejemplo un vector o una lista de elementos), el cual esta previamente ordenado. La característica principal de este método es que reduce el tiempo de búsqueda, puesto que no recorre completamente, sino que la recorre por partes sustentándose en la estrategia de dividir y conquistar.

Para realizar la búsqueda binaria se compara un elemento cualesquiera de la estructura (en general con el elemento central) con el valor que se desea buscar. Si al comparar este valor con el elemento buscado se concluye que son iguales finaliza el proceso, pero si el valor es menor al deseado se repite el proceso con la lista de datos a la izquierda o si es mayor con los datos a la derecha. Realizando este procedimiento de búsqueda para encontrar el elemento se obtendrán intervalos cada vez más pequeños hasta llegar a un solo elemento. Si no se encuentra en este ultimo, el elemento buscado no esta en la estructura. En la figura 2.2, que se muestra a continuación se esquematiza este proceso, en un vector $\boldsymbol{x}$.

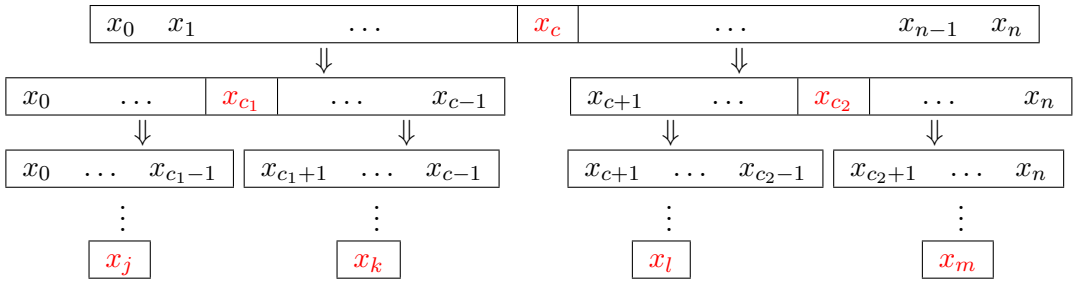


Figura 2.2: Esquema de Búsqueda Binaria.

Ejemplo 2.5. Dado el vector

$$\boldsymbol{x} = [1\ 6\ 7\ 8\ 9\ 12\ 15\ 20\ 22\ 30\ 32\ 34\ 37\ 40],$$

determinar si existe un elemento del vector $\boldsymbol{x}$ con valor 16 y otro con el valor 9.

Mediante las figuras 2.3 y 2.4 se ilustran, respectivamente, el procedimiento para determinar si el 16 y el 9 están en $\boldsymbol{x}$.

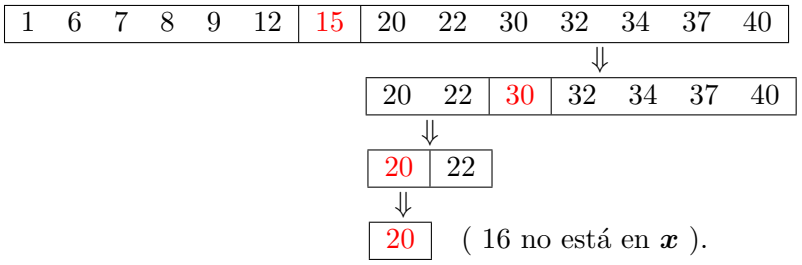


Figura 2.3: Búsqueda Binaria para el valor 16.

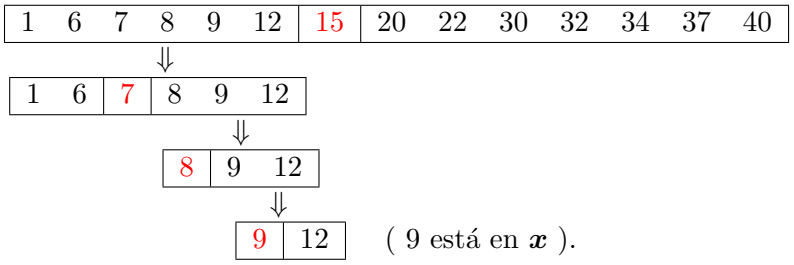


Figura 2.4: Búsqueda binaria para el valor 9.

En el caso que se realice la comparación del elemento en la mitad de un vector de longitud n (como se suele realizar), el número máximo de k intervalos posibles donde está el intervalo es $\frac{n}{2^k} = 1$. Es decir, en el peor de los casos se obtenga al realizar el proceso un intervalo de longitud uno. Despejando k, se encuentra que $k = \log_2 n$. Así, del ejemplo 2.5, como la longitud del vector $\boldsymbol{x}$ es 14, el número de intervalos máximos posibles está dado por

$$log_2(14) \approx 3.80 < 4.$$

Esto quiere decir que se puede dividir máximo en cuatro subintervalos, para realizar la búsqueda binaria. Del mismo modo si un vector es de longitud 100,000,000 el número máximo de pasos para realizar la búsqueda es cerca de 27.

2.2.1. Búsqueda de elementos

Dada la posición (i, j) de un elemento de una matriz esparza, se desea encontrar el valor correspondiente a dicha posición, es decir a_{ij} utilizando un esquema de almacenamiento asociado a la matriz esparza. Si al recorrer el esquema, no se lo puede relacionar con ningún elemento, entonces $a_{ij} = 0$, debido a que se almacenan únicamente los elementos no nulos de la matriz.

Ejemplo 2.6. De la matriz A del ejemplo 2.1, si se desea encontrar el elemento $a_{1,3}$ y $a_{3,4}$ utilizando el esquema **AIJ**, la búsqueda se inicia con el vector fila $\boldsymbol{IA}$, ya que el orden está dado por filas. Así, realizando la búsqueda para a_{13} se encuentra que solo hay un índice 1, en la primera posición y si se ubica en $\boldsymbol{JA}$ en la misma posición se encuentra que el valor es 4, lo que quiere decir que el valor $a_{1,3} = 0$.

Para el valor $a_{3,4}$ se encuentra que en $\boldsymbol{IA}$ hay tres veces el índice de la fila 3 y si se ubica en $\boldsymbol{JA}$ en la misma posición de estos tres índices se encuentra que el índice de columna 4 en la posición 6 y por lo tanto, $a_{34} = AA(6) = 8$.

Hay que recordar, que la eficiencia en la búsqueda de elementos en los esquemas, depende de la organización de los elementos, el funcionamiento del esquema para relacionar los índices de fila y de columna de la entrada de la matriz y el método de búsqueda que se realice.

2.2.2. Inserción y eliminación de elementos

Al trabajar con esquemas de almacenamiento una de las funciones más importantes es la inserción de elementos no nulos de la matriz a dichos esquemas. Esta acción se realiza para almacenar la matriz o para realizar cambios en el almacenamiento, debido a procesos que sufre la matriz como el reordenamiento de sus elementos. Esto se observa por ejemplo, en la eliminación Gaussiana, donde hay un cambio de elementos, por lo cual se requiere insertar o reemplazar elementos del esquema.

Ejemplo 2.7. Para la matriz A se va a insertar el elemento $a_{5,4} = 6$, $a_{3,4}$ =-0.4 y $a_{1,4} = 0$ en el esquema **CSR**, el cual se obtuvo al introducir dicho esquema.

En primer lugar se va a insertar el elemento $a_{5,4} = 6$. Para ello se calcula $nnz_i(5) + 1$ y $nnz_i(6)$ para conocer en que posición inician los elementos no nulos de la fila 5 y en que posición terminan los elementos de dicha fila respectivamente, y así poder ubicar el índice de columna 4 en $\boldsymbol{JA}$ y el valor de la entrada 6 en $\boldsymbol{AA}$. Como $nnz_i(5) + 1 = 9$ y $nnz_6 + 1 = 9$, por tanto solo hay un elemento no nulo en la fila 5 y al ubicarse en la novena posición de JA, que es $JA(9)$=1, entonces se ubica

después y el $a_{5,4} = 6$ luego de $AA(9)$=1.

Ahora, de forma análoga para insertar el elemento $a_{3,3} = -0.4$ se ubica en la posición $nnz_i(3) + 1 = 4$ y $nnz_i(4) = 6$. Entonces, se puede ver que $JA(4)$=1, $JA(5)$=3 y $JA(6)$=4 son los índices de columna de la fila 4, y se reemplaza el valor $AA(5) = 7$ por -0,3.

Finalmente, para insertar el elemento $a_{1,4} = 0$, de la misma manera que en los casos anteriores, se encuentra por el apuntador de nnz_i que hay un único elemento no nulo en la fila en la posición $(1, JA(1)) = (1, 4)$ y así se quita los elementos $JA(1)$ y $AA(1)$.

El almacenamiento con los cambios efectuados es

$\boldsymbol{AA} =$

-2	5	3	-0.3	8	6	5	1	6

$\boldsymbol{JA} =$

1	3	1	3	4	3	4	1	4

$\boldsymbol{nnz_i} =$

0	1	3	6	8	9

Para otros esquemas como por ejemplo el **AIJ** y **CSV** a diferencia del **CSR** no existe un apuntador para llegar exactamente a la posición donde se almacenen los índices de columna de una fila respectiva y en consecuencia se requiere recorrer las filas del almacenamiento.

2.2.3. Producto matriz por vector

Una de las operaciones matriciales matriciales que debe realizarse de forma eficiente es el producto matriz por vector. El algoritmo que se sigue frecuentemente para realizar esta operación utilizando la matriz densa es el siguiente.

Algoritmo 1 Matriz por vector

Entrada: : Matriz A, vector $\boldsymbol{x}$

Salida: : vector $\boldsymbol{y} = A\boldsymbol{x}$

1: Inicializar vector $\boldsymbol{y} = 0$
2: **para** $i = 1 : m$ **hacer**
3: **para** $j = 1 : n$ **hacer**
4: $y(i) = y(i) + A(i, j)x(i)$
5: **fin para**
6: **fin para**
7: **devolver** vector y

A partir del algoritmo 1, se observa que hay un ciclo externo de tamaño m y un ciclo interno de tamaño n, en las lineas 2 y 3 respectivamente. Dentro de estos ciclos se realiza una suma, una multiplicación y una asignación. Luego el número de operaciones está dado por

$$\sum_{i=1}^{m}\sum_{i}^{n} 3 = 3mn$$

y el orden del algoritmo es $\mathrm{O}(mn)$, siendo m el número de filas y n el número de columnas de la matriz.

Ejemplo 2.8. Sean

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ -2 & 0 & 5 & 0 & 0 \\ 3 & 0 & 7 & 8 & 0 \\ 0 & 0 & 6 & 5 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{y} \quad \boldsymbol{x} = \begin{bmatrix} 9 \\ 10 \\ 1 \\ 10 \\ 7 \end{bmatrix}.$$

Realizando el producto se tiene que

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ -2 & 0 & 5 & 0 & 0 \\ 3 & 0 & 7 & 8 & 0 \\ 0 & 0 & 6 & 5 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 9 \\ 10 \\ 1 \\ 10 \\ 7 \end{bmatrix} = \begin{bmatrix} 0\times 9 + 0\times 10 + 0\times 1 + 1\times 10 + 0\times 7 \\ -2\times 9 + 0\times 10 + 5\times 1 + 0\times 10 + 0\times 7 \\ 3\times 9 + 0\times 10 + 7\times 1 + 8\times 10 + 0\times 7 \\ 0\times 9 + 0\times 10 + 6\times 1 + 5\times 10 + 0\times 7 \\ 1\times 9 + 0\times 10 + 0\times 1 + 0\times 10 + 0\times 7 \end{bmatrix} = \begin{bmatrix} 10 \\ -13 \\ 114 \\ 56 \\ 9 \end{bmatrix}.$$

En este ejemplo, se observa la cantidad de productos por ceros, en comparación con los productos que no se anulan es mayor. En el caso de las matrices esparzas se realizará el producto únicamente con los elementos no nulos de la matriz, minimizando los costos de tiempo y el orden computacional.

El orden de operaciones con les esquemas de almacenamientos para realizar el producto matriz por vector será $\mathrm{O}(nnz)$, siendo nnz el número de elementos no nulos de la matriz esparza que es menor comparado con $\mathrm{O}(nm)$.

A continuación se muestran los algoritmos para realizar el producto matriz esparza por vector usando los esquemas de almacenamientos mencionados en la sección 2.1, que se han propuesto a partir del funcionamiento de cada esquema.

Producto matriz por vector usando AIJ

Para realizar el producto matriz por vector usando el esquema **AIJ** asociado a una matriz esparza A, se utilizarán los vectores $\boldsymbol{AA}$, $\boldsymbol{IA}$ y $\boldsymbol{JA}$ que componen este esquema. El algoritmo para realizar el producto es el siguiente.

Algoritmo 2 Matriz por vector usando **AIJ**.

Entrada: Almacenamiento **AIJ** de A: $\boldsymbol{AA}$, $\boldsymbol{JA}$ y $\boldsymbol{IA}$, vector $\boldsymbol{x}$

Salida: vector $\boldsymbol{y} = A\boldsymbol{x}$

1: inicializar vector $\boldsymbol{y} = 0$
2: **para** $i = 1 : nnz$ **hacer**
3: $\quad y(IA(i)) = y(IA(i)) + AA(i)x(JA(i))$
4: **fin para**
5: **devolver** vector $\boldsymbol{y}$.

Como se observa, en la linea 3 del algoritmo 2, realizan una suma, una y una asignación. En la linea 2 hay un ciclo de nnz veces. Por tanto el número de operaciones que se realizan está dada por

$$\sum_{i=1}^{nnz} 3 = 3 * nnz,$$

lo que quiere decir que el algoritmo tiene un orden de O(nnz).

Producto matriz por vector usando CSR

Para realizar el producto matriz por vector usando el esquema **CSR** asociado a una matriz esparza, se utilizarán los vectores $\boldsymbol{AA}$, $\boldsymbol{JA}$ y $\boldsymbol{nnz_i}$ que componen este esquema. El algoritmo para realizar el producto es el siguiente.

Algoritmo 3 Matriz por vector usando **CSR**.

Entrada: Almacenamiento **CSR** de A: $\boldsymbol{AA}$, $\boldsymbol{JA}$ y $\boldsymbol{nnz_i}$, vector $\boldsymbol{x}$

Salida: vector $\boldsymbol{y} = A\boldsymbol{x}$

1: inicializar vector $\boldsymbol{y} = 0$
2: **para** $i = 1 : n$ **hacer**
3: $\quad$ **para** $j = nnz_i(i) + 1 : nnz_i(i + 1)$ **hacer**
4: $\quad\quad y(i) = y(i) + AA(j)x(JA(j))$
5: $\quad$ **fin para**
6: **fin para**
7: **devolver** vector $\boldsymbol{y}$

En este algoritmo se realizan tres operaciones en la linea 4, que se encuentra dentro de un ciclo interno de longitud $nnz_i(i + 1) - nnz_i(i)$, para $i = 1, 2, \ldots, n$ en la linea 3 y un ciclo externo de longitud n en la linea 2. El orden es O(nnz), dado que el número de operaciones es

$$\begin{aligned} 3\left(\sum_{i=1}^{n} nnz_i(i+1) - nnz_i(1)\right) &= 3(nnz_i(n+1) - nnz_i(i)) \\ &= 3(nnz - 0) = 3(nnz). \end{aligned}$$

Producto matriz por vector usando CSR mod

Para realizar el producto matriz por vector usando el esquema **CSR mod** asociado a una matriz esparza A , se utilizarán el conjunto vectores AA, JA y el vector $\boldsymbol{nnz_i}$ que componen este esquema. El algoritmo para realizar el producto es el siguiente.

Algoritmo 4 Matriz por vector usando **CSR mod**.

Entrada: Almacenamiento **CSR mod** de A: AA, JA y $\boldsymbol{nnz_i}$, vector $\boldsymbol{x}$

Salida: vector $\boldsymbol{y} = A\boldsymbol{x}$

1: inicializar vector $\boldsymbol{y} = 0$
2: **para** $i = 1 : n$ **hacer**
3: **para** $j = 1 : nnz_i(i)$ **hacer**
4: $y(i) = y(i) + AA(nnz_i(i), j)x(JA(nnz_i(i), j))$
5: **fin para**
6: **fin para**
7: **devolver** vector $\boldsymbol{y}$

En este algoritmo se realizan tres operaciones en la linea 4, dentro de un ciclo interno de longitud $nnz_i(i)$, para $i = 1, 2, \ldots, n$ en la linea 3 y dentro del ciclo externo de longitud n en la linea. Nuevamente el orden es O(nnz) y el número de operaciones está dado por

$$3\left(\sum_{i=1}^{n} nnz_i(i)\right) = 3(nnz).$$

Producto matriz por vector usando MSR

Para realizar el producto matriz por vector usando el esquema **MSR** asociado a una matriz esparza A , se utilizarán los vectores $\boldsymbol{AA}$ y $\boldsymbol{JA}$ que componen este esquema. El algoritmo para realizar el producto es el siguiente.

Algoritmo 5 Matriz por vector usando **MSR**

Entrada: -Almacenamiento **MSR** de A: $\boldsymbol{AA}$, $\boldsymbol{JA}$, vector $\boldsymbol{x}$

Salida: vector $\boldsymbol{y} = A\boldsymbol{x}$

1: **para** $i = 1 : n$ **hacer**
2: $y(i) = AA(i)x(i)$
3: **para** $j = JA(i) : JA(i + 1) - 1$ **hacer**
4: $y(i) = y(i) + AA(j)x(JA(j))$
5: **fin para**
6: **fin para**
7: **devolver** vector $\boldsymbol{y}$

De forma análoga que en los anteriores algoritmos, se encuentra que el orden de este algoritmo es O(nnz).

Producto matriz por vector usando CSV

Para realizar el producto matriz por vector usando el esquema **CSV** asociado a una matriz esparza A , se utilizarán los vectores $\boldsymbol{AA}$ y $\boldsymbol{JA}$ que componen este esquema. El algoritmo para realizar el producto es el siguiente.

Algoritmo 6 Matriz por vector usando **CSV**

Entrada: -Almacenamiento **CSV** de A: $\boldsymbol{AA}$, $\boldsymbol{JA}$
-vector $\boldsymbol{x}$

Salida: vector $\boldsymbol{y} = A\boldsymbol{x}$

1: variables a utilizar:
frecuencia acumulada de los elementos de JA: k
fila i-ésima de la matriz : i
2: inicializar $k = 0$ y $i = 1$
3: **para** $j = 1 : nnz$ **hacer**
4: $k = JA(j) + k$
5: **mientras** $k > n * i$ **hacer**
6: $i = i + 1$
7: **fin mientras**
8: $y(i) = y(i) + AA(j)\boldsymbol{x}(k - n * (i - 1))$
9: **fin para**
10: **devolver** vector $\boldsymbol{y}$

De Igual forma, se encuentra que el orden de este algoritmo es O(nnz).

Utilizando algunos de los esquemas de almacenamiento que se presentaron, se pueden mostrar que otras operaciones que tienen el orden de complejidad O(nnz) son el producto matriz transpuesta por vector y el cálculo de la matriz transpuesta.

Capítulo 3

Sistemas lineales

Los sistemas lineales son de gran importancia, dada su utilidad en la formulación matemática de muchas aplicaciones. En particular es de interés para los casos donde la matriz del sistema se caracteriza por ser esparza y de grandes dimensiones, como los resultantes de discretizaciones numéricas de ecuaciones diferenciales.

En este capítulo se abordan temas relacionados con propiedades matriciales y sistemas lineales como lo es el número condicionamiento. Además se introducen métodos directos como la eliminación Gaussiana, la factorización LU y la factorización de Cholesky, también se presentarán métodos iterativos como el método de Jacobi, de Gauss-Seidel y el de SOR. Aunque los métodos del gradiente son iterativos, se abordarán en el capítulo 4. La teoría y los teoremas presentados en este capítulo, han sido tomados de [12, 26] y [8].

Definición 3.1. (Sistema lineal) Un sistema lineal es un conjunto de n ecuaciones lineales con m incógnitas, que se expresa de forma genérica como

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_2 + a_{22}x_2 + a_{32}x_3 + \ldots + a_{2n}x_n &= b_2 \\ a_{31}x_2 + a_{32}x_2 + a_{33}x_3 + \ldots + a_{3n}x_n &= b_3 \\ \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots \quad \vdots & \\ a_{m1}x_2 + a_{m2}x_2 + a_{m3}x_3 + \ldots + a_{mn}x_n &= b_m, \end{cases}$$

donde x_i, $i = 1, 2, \ldots, n$, son las incógnitas y a_{ij} y b_i, $i = 1, 2, \ldots, m$ y $j = 1, 2, \ldots, n$, son valores conocidos. Además de expresar un sistema de forma genérica, se puede expresar de forma matricial como

$$A\boldsymbol{x} = \boldsymbol{b},$$

donde

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{32} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \quad y \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix}.$$

La matriz A se denomina *matriz del sistema*, $\boldsymbol{b}$ es el *vector independiente* y $\boldsymbol{x}$ es el *vector solución* del sistema. Dependiendo del sistema, puede existir un único vector o infinitos vectores $\boldsymbol{x}$ que resuelvan el sistema, o que simplemente no tenga solución. La solución de un sistema lineal se puede relacionar con el determinante de la matriz del sistema como se observa en el teorema 1.5.

3.1. Número de condicionamiento

Dadas las grandes dimensiones que puede alcanzar un sistema lineal, se requiere del uso de herramientas computacionales para poder resolverlo, sin embargo, el computador maneja un cierto número de cifras de aproximación lo cual puede afectar al proceso de obtención de la solución.

Si se quiere determinar que tan cercana o que tan lejana está la solución exacta de la obtenida de forma aproximada para un sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, se define el vector error $\boldsymbol{e} = \boldsymbol{x}_t - \boldsymbol{x}_c$, siendo x_t, la solución real del sistema y x_c la solución aproximada. Así, si $\boldsymbol{e} = 0$ implicaría que $x_t = x_c$. Dado que en general no se puede calcular $\boldsymbol{e}$, debido a que se no se conoce en general a $\boldsymbol{x}_t$, se debe recurrir a calcular el vector residual $\boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}_c$. De la misma manera que en el vector error, si $\boldsymbol{r} = 0$, x_c, es la solución exacta del sistema. Aunque si el residual es bastante cercano al vector nulo no implica que $\boldsymbol{x}_c$ sea bastante cercano a $\boldsymbol{x}_t$.

Para analizar el comportamiento del error y el residuo, se hace mediante el valor de su norma, esto es, $\|\boldsymbol{e}\| = \|\boldsymbol{x}_t - \boldsymbol{x}_c\|$ y $\|\boldsymbol{r}\| = \|\boldsymbol{x}_c - \boldsymbol{x}_t\|$. También se puede medir el error relativo de la solución del sistema

$$\frac{\|\boldsymbol{x}_c - \boldsymbol{x}_t\|}{\boldsymbol{x}_t} = \frac{\|\boldsymbol{e}\|}{\|\boldsymbol{x}_t\|}.$$

Ejemplo 3.1. Considerando el sistema lineal

$$\begin{bmatrix} 0.757 & 0.546 \\ 0.913 & 0.659 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.211 \\ 0.254 \end{bmatrix}.$$

Si se soluciona con tres cifras de aproximación mediante uno de los método más conocidos, como es el de eliminación, se obtiene que $x_c = 0.279$ y $y_c = 0$. Así,

$$r = \begin{bmatrix} 0.211 \\ 0.254 \end{bmatrix} - \begin{bmatrix} 0.757 & 0.546 \\ 0.913 & 0.659 \end{bmatrix} \begin{bmatrix} -0.819 \\ 0.500 \end{bmatrix} = \begin{bmatrix} 2.3 \times 10^{-4} \\ 7.27 \times 10^{-4} \end{bmatrix},$$

por lo que $\|r\|_2 = 7.625 \times 10^{-4}$. De este sistema es fácil ver que $x_t = -1$ y $y_t = 1$ es la solución exacta del sistema, pero

$$\boldsymbol{e} = \boldsymbol{x}_t - \boldsymbol{x}_c = \begin{bmatrix} -1 \\ 1 \end{bmatrix} - \begin{bmatrix} -0.0819 \\ 0.500 \end{bmatrix} = \begin{bmatrix} -0.181 \\ -1.500 \end{bmatrix},$$

y así $\|e\|_2 = 1.511$, que al compararse con el valor del residuo no es tan pequeño.

Al igual que analizar el vector error y el vector residuo, se analiza la solución del sistema original $A\boldsymbol{x} = \boldsymbol{b}$, con los sistemas perturbados $A(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b}$, $(A + \Delta A)(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b}$, o $A\boldsymbol{x} = \boldsymbol{b} + \Delta\boldsymbol{b}$, siendo $\Delta A, \Delta b$ y Δb, una pequeña perturbación del sistema original.

Ejemplo 3.2. Considerar el sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$ y $A\boldsymbol{x} = (\boldsymbol{b} + \Delta\boldsymbol{b})$, donde

$$A = \begin{bmatrix} -1 & 4 & 4 & 0 \\ 4 & -2 & 4 & 4 \\ 4 & 4 & -3 & 4 \\ 0 & 4 & 4 & -4 \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} 0.5 \\ 2.9 \\ 6.775 \\ 1.4 \end{bmatrix} \quad y \quad \Delta\boldsymbol{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.1 \\ 0 \end{bmatrix}.$$

La solución exacta del sistema $A\boldsymbol{x} = \boldsymbol{b}$ es

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ 0.5 \\ -0.125 \\ 0.1 \end{bmatrix},$$

mientras que la solución aproximada de $A\boldsymbol{x} = (\boldsymbol{b} + \Delta\boldsymbol{b})$ calculada con la ayuda del software MATLAB con el comando $A \setminus b$ es

$$\boldsymbol{x}_1 = \begin{bmatrix} 1.0056338028169 \\ 0.508450704225352 \\ -0.132042253521127 \\ 0.105633802816901 \end{bmatrix}.$$

En este caso los cambios pequeños en las entradas del vector $\boldsymbol{b}$ no han generado un pequeño cambio $\Delta\boldsymbol{x}$ en la solución. Si se mide el error relativo usando la norma 2 se obtiene que

$$\frac{|||\, \boldsymbol{x}_2 - \boldsymbol{x} \,|||_2}{|||\, \boldsymbol{x} \,|||_2} = \frac{|||\, \Delta\boldsymbol{x} \,|||_2}{|||\, \boldsymbol{x} \,|||_2} =$$

Ahora realizando los mismos cálculos pero con la matriz del sistema

$$H = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix} \text{ y } \boldsymbol{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix},$$

se obtiene que la solución del sistema $H\boldsymbol{x} = \boldsymbol{b}$ es

$$\boldsymbol{x} = \begin{bmatrix} -64 \\ 900 \\ -2520 \\ 1820 \end{bmatrix},$$

mientras que la solución del sistema perturbado $H\boldsymbol{x} = \boldsymbol{b} + \Delta b$ es

$$\boldsymbol{x}_1 = \begin{bmatrix} -40 \\ 630 \\ -1872 \\ 1400 \end{bmatrix}.$$

Midiendo el error relativo de la solución usando la norma 2, se obtiene que

$$\frac{|||\ \boldsymbol{x} - \boldsymbol{x}_1\ |||_2}{|||\ \boldsymbol{x}\ |||_2} = \frac{|||\ \Delta \boldsymbol{x}_2\ |||_2}{|||\ \boldsymbol{x}\ |||_2} = 0.252.$$

En este caso hubo un aumento considerable en el error relativo con respecto al sistema anterior, por lo cual las pequeñas perturbaciones en el vector independiente del sistema pueden afectar a la solución del sistema original.

Las perturbaciones de un sistemas suelen suceder, por el manejo de cifras de aproximación del computador y por tanto las entradas del vector $\boldsymbol{b}$ o de la matriz A tienen perturbaciones e incluso los métodos que se usan para determinar la solución del sistema también generan perturbaciones en el vector solución $\boldsymbol{x}$.

Por tanto si se considera el sistema $A\boldsymbol{x} = \boldsymbol{b}$, el efecto en la solución del vector $\boldsymbol{x}$ de un cambio $\Delta\boldsymbol{b}$ en la entrada del vector $\boldsymbol{b}$, podría venir de errores de redondeo asociados con la entradas de datos o por una actual perturbación del vector $\boldsymbol{b}$. El resultado del sistema $A\boldsymbol{x} = \boldsymbol{b} + \Delta\boldsymbol{b}$, tendrá una solución $\boldsymbol{x}_t + \Delta\boldsymbol{x}$ para el cual $A\Delta\boldsymbol{x} = \Delta\boldsymbol{b}$. Ahora $\Delta x = A^{-1}\Delta\boldsymbol{b}$ y usando las propiedades de normas matriciales se tiene que

$$\|A\boldsymbol{x}\| \leq \|A^{-1}\Delta\boldsymbol{b}\| \leq \|A^{-1}\|\|\Delta\boldsymbol{b}\|,$$

$$\|\boldsymbol{b}\| = \|A\boldsymbol{x}\| \leq ||| A |||\ \|\boldsymbol{x}\| \ \rightarrow\ \frac{1}{\|\boldsymbol{x}\|} \leq \frac{||| A |||}{|||\ \boldsymbol{b}\ |||}.$$

Multiplicando estas dos inecuaciones se obtiene

$$\frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq ||| A^{-1} ||| ||| A\boldsymbol{b} ||| \ \frac{||| A |||}{||| b} = ||| A ||| ||| A^{-1} ||| \ \frac{\|\Delta\boldsymbol{b}\|}{\|\boldsymbol{b}\|}.$$

Así, el intervalo donde se encuentra el error relativo depende en gran medida del factor $||| A ||| ||| A |||^{-1}$, dado que el error relativo no es muy grande. Esto quiere decir que si $||| A ||| ||| A |||^{-1}$, es muy pequeño el error relativo también lo es y en caso contrario podría ser muy grande este valor. A continuación se define el factor mencionado.

Definición 3.2. (Número de condicionamiento). El número de condicionamiento de una matriz A, con respecto a la norma inducida $|||\cdot|||$, es definido como

$$k(A) = ||| A ||| ||| A^{-1} ||| .$$

Se puede observar que es posible obtener una cota para el número de condicionamiento, independientemente de cualquier norma inducida. Por la propiedad de consistencia de una norma matricial

$$1 = ||| I ||| = ||| AA^{-1} ||| \leq ||| A ||| ||| A^{-1} ||| = k(A).$$

Por tanto para un sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$ se dice que es *bien condicionado* si el número de condicionado es cercano a uno y *mal condicionado* en caso contrario, es decir, si esta muy alejado de uno.

Ejemplo 3.3. Tomando la matriz A y la matriz H del ejemplo anterior, si se calcula su condicionamiento con respecto a la norma 2 se tiene

$$k_2(A) = 3.808 \quad \text{y} \quad k_2(H) = 15513.738,$$

por lo cual el sistema $A\boldsymbol{x} = \boldsymbol{b}$, es mejor condicionada que el sistema $H\boldsymbol{x} = \boldsymbol{b}$ y de hay porque los valores tan grandes en el error relativo de la solución.

A continuación se presenta un teorema que relaciona el error relativo de la solución del sistema con el número de condicionamiento.

Teorema 3.1. *Si $\boldsymbol{x}^*$ es solución real del sistema $A\boldsymbol{x} = \boldsymbol{b}$, entonces la solución del sistema perturbado $A\boldsymbol{x} = \boldsymbol{b} + \Delta\boldsymbol{b}$ es $\boldsymbol{x}^* + \Delta\boldsymbol{x}^*$, que satisface la siguiente desigualdad*

$$\frac{\| \Delta\boldsymbol{x} \|}{\| \boldsymbol{x}^* \|} \leqslant ||| A ||| ||| A^{-1} ||| \frac{||| \Delta\boldsymbol{b} |||}{||| \boldsymbol{b} \|} = k(A) \frac{||| \Delta\boldsymbol{b} |||}{||| \boldsymbol{b} \|}.$$

Demostración. Sea $\boldsymbol{x}^*$ la solución del sistema $A\boldsymbol{x} = \boldsymbol{b}$ y $\boldsymbol{x}^* + \Delta\boldsymbol{x}$ la solución del sistema perturbado $A\boldsymbol{x} = b + \Delta\boldsymbol{b}$ entonces,

$$A(\boldsymbol{x}^* + \Delta\boldsymbol{x}) = \boldsymbol{b} + \Delta\boldsymbol{b} \;\Rightarrow Ax^* + A\Delta\boldsymbol{x} = \boldsymbol{b} + \Delta\boldsymbol{b}.$$

Puesto que de $A\boldsymbol{x}^* = \boldsymbol{b}$ se obtiene que $A\Delta\boldsymbol{x} = \Delta\boldsymbol{b}$, se deduce que $\Delta\boldsymbol{x} = A^{-1}\Delta\boldsymbol{b}$. Usando los resultados anteriores, se tiene

$$\|\boldsymbol{b}\| \leq ||| A ||| \, \|\boldsymbol{x}\| \;\rightarrow\; \frac{1}{\|\boldsymbol{x}^*\|} \leq \frac{||| A |||}{,} \|\boldsymbol{b}\| \tag{3.1.1}$$

$$\|\Delta\boldsymbol{x}\| \leq ||| A^{-1} ||| \, \|\Delta\boldsymbol{b}\|. \tag{3.1.2}$$

Multiplicando las desigualdades (3.1.2) y (3.1.2) se deduce la desigualdad.

$$\frac{\| \Delta\boldsymbol{x} \|}{\| \boldsymbol{x}^* \|} \leqslant ||| A ||| ||| A^{-1} ||| \frac{||| \Delta\boldsymbol{b} |||}{||| \boldsymbol{b} \|} = k(A) \frac{||| \Delta\boldsymbol{b} |||}{||| \boldsymbol{b} \|}.$$

■

El teorema anterior, nos muestra como una perturbación en el vector $\boldsymbol{b}$ puede cambiar la solución del sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, el cual depende del valor de condicionamiento que la matriz del sistema posea. Cabe aclarar que las pequeñas perturbación en $\boldsymbol{b}$ pueden ser por el redondeo de cifras que se aplican cuando se ingresan los datos.

El siguiente teorema describe como un cambio en las entradas de la matriz A, es decir, la matriz $A + \Delta A$ pueden afectar la solución.

Teorema 3.2. *Sean A y $A+\Delta A \in \mathbb{R}^{n\times n}$ matrices cuadradas no singulares. Si $A\boldsymbol{x} = \boldsymbol{b}$ y $(A+\Delta A)$ $(\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b}$. Entonces*

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x} + \Delta\boldsymbol{x}\|} \leq |||A|||||A^{-1}||| \frac{|||\Delta A|||}{|||A|||} = k(A)\frac{|||\Delta A|||}{|||A|||}.$$

Demostración. Partiendo del hecho de que $(A + \Delta A)\ (\boldsymbol{x} + \Delta\boldsymbol{x}) = \boldsymbol{b}$ y $A\boldsymbol{x} = \boldsymbol{b}$ se cumple que

$$A\boldsymbol{x} + A\Delta\boldsymbol{x} + \Delta A(\boldsymbol{x} + \Delta\boldsymbol{x}) = A\boldsymbol{x},$$

es decir,

$$\Delta\boldsymbol{x} = -A^{-1}\Delta A(\boldsymbol{x} + \Delta\boldsymbol{x}).$$

A partir de esta igualdad y por las propiedades de las normas matricial, se tiene

$$\|\Delta\boldsymbol{x}\| \leq ||| A^{-1} ||| ||| \Delta A ||| \ \|(\boldsymbol{x} + \Delta\boldsymbol{x})\| = ||| A^{-1} ||| \frac{||| A |||}{||| A |||} ||| \Delta A ||| \ \|(\boldsymbol{x} + \Delta\boldsymbol{x})\|.$$

Multiplicando esta inecuación entre $\dfrac{1}{\|(\boldsymbol{x} + \Delta\boldsymbol{x})\|}$, se obtiene la desigualdad

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x} + \Delta\boldsymbol{x}\|} \leq |||A|||||A^{-1}||| \frac{|||\Delta A|||}{|||A|||} = k(A)\frac{|||\Delta A|||}{|||A|||}.$$

■

Este resultado no nos muestra una cota superior del error relativo del sistema, pero si nos índica la del error relativo del sistema perturbado, el cual de forma análoga de lo expuesto en el teorema 3.1, depende del condicionamiento de la matriz del sistema. Este resultado no nos provee información del sistema original, sin embargo servirá para el siguiente teorema, que es un mejor resultado que este.

Teorema 3.3. *Sea $A \in \mathbb{R}^{n\times n}$ una matriz cuadrada de orden n no singular con $A\boldsymbol{x} = \boldsymbol{b}$ y sea ΔA una perturbación en A con $(A + \Delta A)(\boldsymbol{x} + \Delta\boldsymbol{x}) = b$. Si $|||A^{-1}|||||\Delta A||| < 1$, entonces*

$$\frac{|||\Delta\boldsymbol{x}|||}{\|\boldsymbol{x}\|} \leq \frac{k(A)\frac{|||\Delta A|||}{|||A|||}}{1 - k(A)\frac{|||\Delta A|||}{|||A|||}}.$$

Demostración. Por la prueba realizada en el teorema 3.2, se deduce que

$$\|\Delta\boldsymbol{x}\| \leq ||| A^{-1} ||| ||| \Delta A ||| \, \|(\boldsymbol{x} + \Delta\boldsymbol{x})\|.$$

Aplicando la desigualdad triangular a $\|(\boldsymbol{x} + \Delta\boldsymbol{x})\|$ se tiene

$$\|\Delta\boldsymbol{x}\| \leq ||| A^{-1} ||| ||| \Delta A ||| \, \|(\boldsymbol{x} + \Delta\boldsymbol{x})\| \leq ||| A^{-1} ||| ||| \Delta A ||| \, \|\boldsymbol{x}\| + \|\Delta\boldsymbol{x}\|,$$

en consecuencia

$$\|\Delta\boldsymbol{x}\|(1- ||| A^{-1} ||| ||| \Delta A |||) \leq ||| A^{-1} ||| ||| \Delta A ||| \, \|\boldsymbol{x}\| = ||| A^{-1} ||| ||| A ||| \left(\frac{||| \Delta A |||}{||| A |||}\right) \|\boldsymbol{x}\|.$$

Por la hipótesis del problema, $||| A^{-1} ||| ||| \Delta A |||< 1$, lo que implica,

$$1- ||| A^{-1} ||| ||| \Delta A |||> 0.$$

A partir de esta desigualdad se concluye que

$$\frac{\|\Delta\boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{||| A^{-1} ||| ||| A ||| \frac{||| \Delta A |||}{||| A |||}}{1- ||| A^{-1} ||| ||| \Delta A |||} = \frac{||| A^{-1} ||| ||| A ||| \frac{||| \Delta A |||}{||| A |||}}{1- ||| A^{-1} ||| ||| A ||| \frac{||| \Delta A |||}{||| A |||}}$$

$$= \frac{k(A)\frac{||| \Delta A |||}{||| A |||}}{1 - k(A)\frac{||| \Delta A |||}{||| A |||}}$$

■

Este resultado nos indica que si el error relativo en A es pequeño y el $k(A)$ no es demasiada grande, el denominador de es cercano a 1. En conclusión el error relativo respecto a la la solución del vector $\boldsymbol{x}$, al error relativo en A es acotado por $\gamma k(A)$, donde γ es una constante del orden de la unidad.

3.2. Métodos directos

Los métodos directos se utilizan para resolver de forma exacta un sistema lineal y se conoce el número de pasos necesarios para ello. El número de operaciones que realizan depende del tamaño

de la matriz del sistema. Una de las características de estos métodos es que en general utilizan operaciones elementales por fila para convertir el sistema lineal a un sistema triangular, es decir, un sistema donde la matriz del sistema es triangular superior o inferior, con el fin de que se puede despejar las variables de las ecuaciones de los sistemas y encontrar su respectiva solución.

Entre los métodos directos que se van a tratar en esta sección, son: eliminación Gaussiana, factorización LU y la factorización de Cholesky. Los temas que se van a centrar de estos temas son en sus algoritmos, teoremas que garantizan su aplicabilidad y en el costo computacional que estos conllevan. Entre las operaciones elementales, se resalta que la permutación, se estudiará de forma superficial.

3.2.1. Eliminación Gaussiana

La eliminación Gaussiana es uno de los métodos más conocidos e introductorios para abordar métodos directos. El método se basa en que a partir de una matriz $A \in \mathbb{R}^{m\times n}$, se realizan operaciones elementales por filas para obtener una matriz triangular superior U. El algoritmo de este proceso es muy conocido y se puede encontrarse en [12]. Cuando la matriz A es cuadrada, se puede deducir que el número de multiplicaciones que se realizan en el algoritmo es cerca de $\frac{n^3}{3}$.

Ejemplo 3.4. Dado el sistema

$$\begin{cases} x_1 - 5x_2 = 1 \\ 2x_1 + x_2 + x_3 = 1 \\ 3x_1 - 2x_2 + 3x_2 = 1. \end{cases}$$

Aplicando eliminación Gaussiana a la matriz aumentada $[A|b]$ asociada al sistema se tiene

$$\left[\begin{array}{ccc|c} 1 & -5 & 0 & 1 \\ 2 & 1 & 1 & 1 \\ 3 & -2 & 3 & 1 \end{array}\right] \xrightarrow[R_2 \leftarrow -2R_1+R_2]{R3 \leftarrow -3R_1+R_3} \left[\begin{array}{ccc|c} 1 & -5 & 0 & 1 \\ 0 & 11 & 1 & -1 \\ 0 & 13 & 3 & 1 \\ 0 & 13 & 3 & -2 \end{array}\right] \xrightarrow[R_3 \leftarrow -\frac{13}{11}R_2+13]{} \left[\begin{array}{ccc|c} 1 & -5 & 0 & 1 \\ 0 & 11 & 1 & -1 \\ 0 & 0 & \frac{20}{11} & -\frac{9}{11} \end{array}\right].$$

De lo anterior, el sistema dado, se convierte en el sistema

$$\begin{cases} x_1 - 5x_2 = 1 \\ 11x_2 + x_3 = -1 \\ \frac{20}{11}x_3 = -\frac{9}{11} \end{cases},$$

y como la matriz del sistema es triangular se despeja la última variable x_3 y posteriormente x_2

y x_1. De esta manera, la solución del sistema es

$$\boldsymbol{x} = \begin{bmatrix} \frac{3}{4} \\ -\frac{1}{20} \\ -\frac{9}{20} \end{bmatrix}.$$

En este ejemplo, al realizar el proceso de eliminación Gaussiana el sistema original se transforma en un sistema triangular superior. El proceso de resolver dicho sistema, se basa en despejar las variables, empezando desde la última ecuación del sistema hasta la primera y se denomina back substitution. Ahora se dará a conocer el concepto de pivotes que es fundamental para poder aplicar la eliminación Gaussiana.

Observación 3.1. Los elementos en la diagonal utilizados en el proceso de la eliminación Gaussiana, se conocen como pivotes*pivotes*. Si los pivotes son cero el método de eliminación Gaussiana falla y por ello es necesario realizar una operación por filas de tipo 3, es decir un intercambio de filas para continuar dicho proceso. A este proceso se le conoce como Eliminación Gaussiana con pivoteo.

3.2.2. Factorización LU

Retomando los preliminares del capítulo 1, para realizar una operación elemental a las filas de una matriz $A \in \mathbb{R}^{m\times n}$, se multiplica a izquierda por una matriz elemental adecuada E, esto es

$$E(A) = E(I)A.$$

Si E es una matriz elemental de tipo 2, es decir, una operación de fila elemental $R_j \leftarrow kR_i + R_j$, con $i < j$, entonces la matriz elemental $E_{i,j}$ es una matriz triangular inferior, en el cual la diagonal principal son unos, y los elementos debajo de la diagonal esta formado por los pivotes.

Supongase que no se necesita intercambio de filas en la eliminación Gaussiana de A para llegar a una matriz triangular superior U. En este caso se tendría

$$E_{l-1,m}E_{l-1,m-1}\ldots, E_{2,3}E_{2,2}E_{1,2}A = U,$$

donde $l = \text{máx}(m, n)$.

Ahora se debe tener en cuenta, que:

- El producto de dos matrices triangulares es una matriz triangular inferior.
- La inversa de una matriz triangular inferior no singular es una matriz triangular inferior.

A partir de lo anterior

$$L_1 = E_r E_{r-1} \dots E_3 E_2 E_1,$$

es una matriz triangular inferior. De igual forma forma la inversa de una matriz no singular triangular inferior es también triangular inferior, entonces se sigue que

$$A = L_1^{-1} U = LU.$$

Por tanto A es un producto de una matriz de una matriz triangular unitaria inferior L y una matriz triangular superior U. Se asegura que la matriz L es unitaria dado que la inversa de una matriz triangular no singular se puede realizar mediante eliminación Gaussiana, sin intercambio de filas y se obtiene una matriz diagonal D, pero como los elementos de la diagonal de L_1 son unitarios entonces $D = I$ y por tanto L_1^{-1} es triangular inferior unitaria.

A esta factorización se la conoce como *factorización LU* de la matriz A. Sin embargo, si en el proceso de eliminación Gaussiana aparece $a_{ii} = 0$, para alguno $i = 1, \dots, n-1$, entonces no es posible realizar dicha factorización sin intercambio de filas.

Recíprocamente si $A = LU$, donde L es triangular inferior unitaria, entonces $L^{-1}A = U$, donde la matriz triangular inferior L^{-1} es un producto de matrices elementales de tipo 2 y A puede ser reducido a U sin intercambio de filas.

A continuación se presenta un teorema para garantizar algunas condiciones sobre la factorización LU de una matriz A y sobre la unicidad de dicha factorización.

Teorema 3.4. *Una matriz $A \in \mathbb{R}^{n\times n}$ tiene una factorización LU si y sólo si no requiere intercambio de filas en la eliminación Gaussiana de A, para llegar a una matriz triangular superior. Si A es no singular la factorización es única.*

Demostración. La primera parte ya fue probada. Falta probar la unicidad de la factorización. Sea $A \in \mathbb{R}^{n\times n}$ no singular tal que $A = L_1U_1 = L_2U_2$, donde L_1 y L_2 son dos matrices triangular unitarias inferiores y U_1 y U_2 son matrices triangulares superiores respectivamente. Dado que A es no singular, L_2^{-1} y U_1^{-1} existen y multiplicando a izquierda por L_2^{-1} y derecha por U_1^{-1} en la igualdad anterior, se obtiene que

$$L_2^{-1}L_1 = U_2U_1^{-1}.$$

Además $L_2^{-1}L_1$ es una matriz triangular inferior y $U_2U_1^{-1}$ es una matriz triangular superior, en consecuencia la única manera de que sean iguales es que $L_2^{-1}L_1$ y $U_2U_1^{-1}$ sean matrices diagonales. Pero L_2^{-1} y L_1 son matrices unitarias entonces $L_2^{-1}L_1 = I$. A partir de eso $L_2 = L_1$ y de igual forma $U_1 = U_2$.

■

Antes de poder realizar el algoritmo de la factorización LU de una matriz $A \in \mathbb{R}^{n\times n}$, es necesario encontrar la matriz $L = L_1^{-1}$ de dicha factorización. Para ello se puede notar que

$$L = L_{n-1}^{-1} L_{n-2}^{2} \dots L_2^{-1} L_1^{-1}$$

siendo $L_i = E_{i,i+1}E_{i,i+2}E_{i,i+2}$ para $i = 1, 2, \dots, n-1$ un producto de matrices elementales resultantes de realizar operaciones elementales en la filas de A. Es decir que L_i, es de la forma

$$L_i = \begin{bmatrix} & & 0 & & \\ & I_{i-1} & \vdots & & 0 \\ & & 0 & & \\ 0 & \dots & 1 & 0 & \dots & 0 \\ & & -l_{i+1,i} & & \\ & 0 & \vdots & & I_{n-i} \\ & & -l_{n,i} & & \end{bmatrix},$$

donde $l_{k,i}$, $k = i+1, \dots, n$ son los pivotes para anular los elementos debajo de la diagonal de la columna i-ésima. Ahora, si se realiza en L_i una operación elemental de fila de tipo 2, $R_i \to -k_{ij}R_j + R_i$, para transformar L_i a la identidad I, se comprueba que

$$L_i^{-1} = \begin{bmatrix} & & 0 & & \\ & I_{i-1} & \vdots & & 0 \\ & & 0 & & \\ 0 & \dots & 1 & 0 & \dots & 0 \\ & & l_{i+1,i} & & \\ & 0 & \vdots & & I_{n-i} \\ & & l_{n,i} & & \end{bmatrix}.$$

De esta manera, para el producto $L = L_{n-1}^{-1} L_{n-2}^{-1} \dots L_1^{-1}$ se observa que es de la forma

$$L = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -l_{12} & 1 & 0 & \dots & 0 \\ -l_{13} & -l_{23} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -l_{1n} & -l_{2n} & -l_{3n} & \dots & 1 \end{bmatrix}.$$

El algoritmo de la factorización LU, igual que con la eliminación Gaussiana es muy conocido y se puede encontrar en [12].

Solución de un sistema usando la factorización LU

Sea el sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$. Si se tiene la factorización LU de A, entonces el sistema $A\boldsymbol{x} = L(U\boldsymbol{x}) = \boldsymbol{b}$ se puede transforma a la solución de dos sistemas triangulares

$$L\boldsymbol{y} = \boldsymbol{b} \qquad \text{y} \qquad U\boldsymbol{x} = \boldsymbol{y}.$$

El sistema triangular $L\boldsymbol{y} = \boldsymbol{b}$ se resuelve despejando las variables, cuyo proceso se lo denomina forward substitution y es similar al proceso de back substitution, pero con la diferencia de que se inicia despejando las variables a partir de la primera ecuación hasta la última del sistema.

Luego de resolver el sistema triangular $L\boldsymbol{y} = \boldsymbol{b}$ se prosigue a resolver el sistema $U\boldsymbol{x} = \boldsymbol{y}$ mediante el proceso de back substitution, donde se encuentra la solución del sistema original $A\boldsymbol{x} = \boldsymbol{b}$.

Obsérvese que el número de multiplicaciones necesarias para resolver un sistema mediante la factorización LU sigue siendo cerca $n^3/3$ multiplicaciones, dado que el proceso de back substitution mas el proceso forward substitution nos da cerca n^2 multiplicaciones.

Ejemplo 3.5. Retomando el sistema lineal del ejemplo 3.4, al aplicar la factorización a la matriz del sistema, se obtiene que

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & \frac{13}{11} & 1 \end{bmatrix} \quad \text{y} \quad U = \begin{bmatrix} 1 & -5 & 0 \\ 0 & 11 & 1 \\ 0 & 0 & \frac{20}{11} \end{bmatrix}.$$

A partir de esto se sigue con resolver los sistemas triangulares

$$\begin{cases} y_1 = 1 \\ 2y_2 + y_3 = 1 \\ 3y_1 + \frac{13}{11}y_2 + y_3 = 1 \end{cases} \qquad \text{y} \qquad \begin{cases} x_1 - 5x_2 = y_1 \\ 11x_2 + x_3 = y_2 \\ \frac{20}{11}x_3 = y_3. \end{cases}$$

Resolviendo el sistema asociado a L, mediante forward substitution, se obtiene

$$\boldsymbol{y} = \begin{bmatrix} 1 \\ -1 \\ -\frac{9}{11} \end{bmatrix}.$$

Con este vector se puede resolver el otro sistema triangular mediante back substitution para encontrar la solución del sistema original, que es

$$\boldsymbol{x} = \begin{bmatrix} \frac{3}{4} \\ -\frac{1}{20} \\ -\frac{9}{20} \end{bmatrix}.$$

Observación 3.2. Como se puede apreciar en el ejemplo 3.5, se ve la independencia de la matriz del sistema con el vector independiente, ya que únicamente se lo aplica cuando se obtiene los sistemas triangulares. Esto quiere decir que se puede resolver varios sistemas lineales que tengan la misma matriz del sistema, dado que solo se realizaría una vez el proceso de la factorización LU y el proceso de solución se realizaría varias veces, pero este es de menor costo.

A continuación se muestra un teorema que garantiza la eliminación Gaussiana sin necesidad de realizar pivoteo, ya sea por aminorar los problemas de redondeo de cifras o por la aparición de elementos nulos en la diagonal.

Teorema 3.5. *Si una matriz A es S.D.P, entonces la eliminación Gaussiana puede realizarse sin pivoteo. Después de que las primeras k columnas han sido reducidas, entonces la matriz de orden $n-k \times n-k$ que se forma en la esquina inferior derecha es siempre S.D.P. Además , A es nosingular con $det(A) > 0$ y el más grande elemento de A, en valor absoluto, aparece en la diagonal.*

Demostración. En primer lugar se puede observar que si P es no singular, entonces PAP^T es definida positiva, ya que para cualquier $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$

$$\boldsymbol{x}^T PAP^T \boldsymbol{x} = (P^T \boldsymbol{x}) AP^T \boldsymbol{x} > 0.$$

También como A es definida positiva, los elementos en la diagonal son positivos. Esto se comprueba si se toma un vector $x = \boldsymbol{e}_i = (0\ 0\ 0 \ldots 1 \ldots\ 0\ 0\ 0)^T$, entonces

$$e_i^T A e_i = e_i^T \begin{bmatrix} a_{1i} \\ a_{2i} \\ \vdots \\ a_{ii} \\ \vdots \\ a_{in} \end{bmatrix} = a_{ii} > 0.$$

Por tanto como $a_{ii} \neq 0$, $i = 1, 2, \ldots, n$ se puede realizar la reducción de la primera fila de A sin intercambio de filas. Sea P la matriz que describe la operación de fila usada para reducir la primera fila de A. Entonces P es una matriz triangular inferior unitaria de la forma

$$P = \left[\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \boldsymbol{c} & I_{n-1} \end{array}\right],$$

por lo cual,

$$PA = \left[\begin{array}{c|c} a_{11} & \boldsymbol{r} \\ \hline \mathbf{0} & A_{22}^{(2)} \end{array}\right],$$

donde $A_{22}^{(2)} \in \mathbb{R}^{n-1\times n-1}$ es obtenida al realizar la reducción de filas de la primer fila de A. Ahora realizando el producto P^TAP se tiene

$$\begin{aligned} PAP^T &= \left[\begin{array}{c|c} a_{11} & \boldsymbol{r} \\ \hline \mathbf{0} & A_{22}^{(2)} \end{array}\right]\left[\begin{array}{c|c} \mathbf{1} & \boldsymbol{c}^T \\ \hline \mathbf{0} & I_{n-1} \end{array}\right] = \left[\begin{array}{c|c} a_{11} & a_{11}\boldsymbol{c}^T + \boldsymbol{r} \\ \hline \mathbf{0} & A_{22}^{(2)}, \end{array}\right] \\ &= \left[\begin{array}{c|c} a_{11} & \mathbf{0} \\ \hline \mathbf{0} & A_{22}^{(2)} \end{array}\right]. \end{aligned}$$

El vector $a_{11}\boldsymbol{c}^T + \boldsymbol{r} = \mathbf{0}$, ya que A es simétrica, $Row_1\ A = (Col_1\ A)^T$ y el hecho de que $\boldsymbol{c}$ es el vector que contiene a los pivotes para reducir la primera fila de A, es decir, $\boldsymbol{c}_i = -\dfrac{a_{i1}}{a_{11}}$, para $i = 2, 3, \ldots, n$.

Cómo la matriz $\left[\begin{array}{c|c} a_{11} & \mathbf{0} \\ \hline \mathbf{0} & A_{22}^{(2)}. \end{array}\right]$ es definida positiva, la submatriz $A_{22}^{(2)}$ es definida positiva también. Esto se prueba utilizando un vector $\boldsymbol{y} = (0\ \boldsymbol{x})^T$, con $\boldsymbol{x} \neq 0 \in \mathbb{R}^{n-1}$ y al realizar la operación

$$\boldsymbol{y}^T A\boldsymbol{y} = \boldsymbol{x}^T A_{22}^{(2)}\boldsymbol{x} > 0.$$

Además de ser $A_{22}^{(22)}$, definida positiva también se verifica que es simétrica, puesto que para cualquier $i, j = 1, 2, \ldots, n$ se tiene que

$$\begin{aligned} ent_{ij}A_{22}^{(2)} &= ent_{ij}A - \frac{a_{1j}a_{i1}}{a_{11}} = ent_{ji}A - \frac{a_{j1}a_{1i}}{a_{11}} \\ &= ent_{ji}A_{22}^{(2)}. \end{aligned}$$

De lo anterior, mediante inducción sobre el tamaño de la matriz A, se prueba que A puede realizarse sin pivoteo y se obtiene la factorización LU de A. También las entradas de la matriz U obtenidas de la eliminación Gaussiana sin pivoteo son positivas, por el hecho de que se forman submatrices en la esquina inferior derecha que son S.D.P y por tanto los elementos de la diagonal son positivos. De este hecho

$$det(A) = det(L)det(U) = u_{11}u_{22}\ldots u_{nn} > 0.$$

Para probar que la matriz A es invertible osea, no singular, se utiliza el teorema 1.5, que nos garantiza de que si el sistema $A\boldsymbol{x} = \mathbf{0}$ tiene como solución única $\boldsymbol{x} = \mathbf{0}$, la matriz A es invertible. Entonces si se supone lo contrario, es decir, que A es singular implicaría que existe un $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$, tal que $A\boldsymbol{x} = 0$ y si se multiplica a izquierda por $\boldsymbol{x}^T$ a ambos lados de la igualdad, se obtendría que $\boldsymbol{x}^T A\boldsymbol{x} = 0$, lo cual contradice el hecho de que A sea S.D.P.

Por último para determinar que el máximo valor de las entradas de A en valor absoluto se encuentra en la diagonal, se supondrá lo contrario. Así, supongase que existe un $a_{ij} = a_{ij} > 0$, para algún $i \neq j$, $1 \leq i, n \leq n$, es la entrada de A con el más grande valor absoluto. Entonces si se toma

$\boldsymbol{x} = \boldsymbol{e}_j - \boldsymbol{e}_i$ se tiene

$$\boldsymbol{x}^T A\boldsymbol{x} = (\boldsymbol{e}_j - \boldsymbol{e}_i)^T \begin{bmatrix} a_{1j} - a_{1i} \\ a_{2j} - a_{2i} \\ \vdots \\ a_{ij} - a_{ii} \\ \vdots \\ a_{jj} - a_{ji} \\ \vdots \\ a_{jn} - a_{in} \end{bmatrix} = a_{jj} - aji - a_{ij} + a_{ii}$$

$$= a_{jj} + a_{ii} - 2a_{ij} < 0, \qquad (a_{ij} = a_{ji})$$

lo cual es una contradicción, ya que A es S.D.P. Por otro lado si se supone que $a_{ji} = a_{ij} < 0$ es el elemento de A con el valor más grande en valor absoluto, entonces de igual forma si se toma $\boldsymbol{x} = \boldsymbol{e}_i + \boldsymbol{e}_j$, se obtiene que

$$\boldsymbol{x}^T A\boldsymbol{x} = a_{ii} + a_{ij} + a_{ji} + a_{jj} < 0,$$

lo cual es de nuevo una contradicción. Por ello el más grande elemento de A debe aparecer en la diagonal.

■

Factorización de Cholesky

Al aplicar la factorización LU de una matriz A se necesita trabajar con la matriz L y la matriz U o la forma compacta de la factorización, esto es almacenando todo en un sola matriz. Sin embargo con la *factorización de Cholesky*, que es una variante de esta factorización, la cual la matriz A se la puede expresar como $A = LL^T$, siendo L una matriz triangular inferior con elementos en la diagonal positivos, nos permite trabajar únicamente con una sola matriz.

Un caso particular donde se asegura la factorización de Cholesky es cuando la matriz es S.D.P. En efecto, por el teorema 3.4, si A es S.D.P a tiene factorización LU sin realizar pivoteo $A = L_1U_1$. Debido a la simetría de A se cumple que

$$L_1U_1 = (L_1U_1)^T = U_1^T L_1^T.$$

Multiplicando por U_1^{-T} a izquierdo por L_1^{-1} y a derecha por L_1^{-T} se tiene

$$U_1L_1^{-T} = L_1^{-T}U_1^T.$$

Pero $U_1L_1^T$ es triangular superior y $L_1^{-1}U^T$ es triangular inferior. Entonces la única forma para que sean iguales es que $U_1L_1 = D$, donde D es una matriz diagonal, cuyos elementos en la diagonal son

positivos, ya que los elementos de la diagonal de U_1 y L_1^{-1} son positivos (Teorema 3.4). Despejando U_1 se obtiene que $U_1 = DL_1^T$. Por lo tanto, $A = L_1U_1 = L_1DL_1^T$.

Como D tiene elementos en la diagonal positivos entonces $D = D^{\frac{1}{2}}D^{\frac{1}{2}}$,
donde $D^{\frac{1}{2}} = D_g\{\sqrt{d_{11}}, \sqrt{d_{22}}, \ldots, \sqrt{d_{nn}}\}$. Así, $A = L_1D^{\frac{1}{2}}D^{\frac{1}{2}}L_1^T = LL^T$, donde $L = L_1D^{\frac{1}{2}}$, es una matriz triangular inferior y tiene elementos en la diagonal positivos, dado que L_1 y $D^{\frac{1}{2}}$ también tienen elementos positivos en la diagonal. Además dicha factorización es única, ya que por el teorema 3.4, se garantiza la unicidad de la factorización LU y por tanto L_1 es única por lo cual L es único.

A continuación se muestra un teorema que garantiza la factorización de Cholesky, el cual se puede encontrar en [12].

Teorema 3.6. *Una matriz $A \in \mathbb{R}^{n\times n}$ es S.D.P, si y sólo si hay una única matriz triangular inferior L, con elementos en la diagonal positivos tal que $A = LL^T$. A esta factorización se la conoce como factorización de Cholesky.*

Demostración. Anteriormente se probó que si A es S.D.P, implica que A tiene factorización de Cholesky. Ahora falta probar el recíproco. Entonces si se supone que A tiene factorización de Cholesky, esto quiere decir que $A = L^TL$ con L una matriz triangular inferior con elementos en la diagonal positivos. A partir de esto si se toma un $\boldsymbol{x} \neq 0 \in \mathbb{R}^n$ se tiene que

$$\boldsymbol{x}^TA\boldsymbol{x} = \boldsymbol{x}^TA\boldsymbol{x} = \boldsymbol{x}^TL^TL\boldsymbol{x} = (L\boldsymbol{x})^T(L\boldsymbol{x}) = \|L\boldsymbol{x}\|_2 > 0,$$

lo que nos índica que A es definida positiva y además dado que $(LL^T)^T = LL^T$, se concluye que A es S.D.P.

Ya se ha probado que si A es S.D.P tiene factorización de Cholesky, pero daremos una prueba alternativa con el fin determinar las entradas la matriz L y así obtener el algoritmo de la factorización de Cholesky. Para ello, se va a realizar la prueba mediante inducción sobre el orden n de las matrices cuadradas S.D.P.

Para $n = 1$, se observa que $A = \sqrt{a_{11}}\sqrt{a_{11}} = L^TL$. Suponiendo que el teorema se cumple para matrices S.D.P cuadradas de orden $k = 1, 2, \ldots, n-1$, se va a determinar que se cumple para matrices cuadradas S.D.P de orden n.

Sea $A = \left[\begin{array}{c|c} A_{11} & \boldsymbol{A}_{12} \\ \hline \boldsymbol{A}_{21} & A_{nn} \end{array}\right] \in \mathbb{R}^{n\times n}$ S.D.P, donde $A_{11} \in \mathbb{R}^{n-1\times n-1}$ y es también S.D.P, esto se demuestra tomando el vector $\boldsymbol{y} = (\boldsymbol{x}\ 0)^T$, para cualquier $\boldsymbol{x} \neq 0 \in \mathbb{R}^{n-1}$ se comprueba que

$$\boldsymbol{y}^TA\boldsymbol{y}^T = \boldsymbol{x}^TA_{11}\boldsymbol{x} > 0.$$

Así, por la hipótesis de inducción $A_{11} = L_{11}L_{11}^T$, donde L_{11} es una matriz triangular inferior con entradas positivas en la diagonal. Si se asume que $L = \left[\begin{array}{c|c} L_{11} & \mathbf{0} \\ \hline \boldsymbol{L}_{21} & x \end{array}\right]$, entonces $A = LL^T$ es equivalente a

$$\begin{aligned} LL^T &= \left[\begin{array}{c|c} L_{11} & \mathbf{0} \\ \hline \boldsymbol{L}_{21} & x \end{array}\right] \left[\begin{array}{c|c} L_{11^T} & \boldsymbol{L}_{21}^T \\ \hline \mathbf{0} & x \end{array}\right] = \left[\begin{array}{c|c} L_{11}L_{11}^T & L_{11}\boldsymbol{L}_{21}^T \\ \hline \boldsymbol{L}_{21}L_{11}^T & \boldsymbol{L}_{21}\boldsymbol{L}_{21}^T + x^2 \end{array}\right] \\ &= \left[\begin{array}{c|c} A_{11} & \boldsymbol{A}_{12} \\ \hline \boldsymbol{A}_{21} & a_{nn} \end{array}\right]. \end{aligned}$$

De esta última ecuación se puede apreciar que $\boldsymbol{L}_{21}$ está determinada únicamente con la solución del sistema triangular $L_{11}L_{21}^T = \boldsymbol{A}_{12}$ y que x esta determinado por la ecuación $\boldsymbol{L}_{21}\boldsymbol{L}_{21}^T + x^2 = a_{nn}$, es decir, que $x = +\sqrt{a_{nn} - \boldsymbol{L}_{21}\boldsymbol{L}_{21}^T}$. Se obvia el signo negativo ya que las entradas de la diagonal son positivas. Para ver que x es bien definida se usa el hecho de que $det(A) > 0$ (teorema 3.5.) Entonces

$$\begin{aligned} det(A) &= det(L)det(L^T) = (xdet(L_{11}))(xdet(L_{11}^T)) \\ &= x^2 det(L_{11})^2 > 0. \end{aligned}$$

Como el $det(L_{11})^2 > 0$, implica que $x^2 > 0$ y así x esta bien definida. En conclusión, la factorización de Cholesky está únicamente determinada por las ecuaciones anteriores y la unicidad de la factorización de Cholesky de A_{11} dada por la inducción de hipótesis.

■

A continuación se muestra el algoritmo 7, que retorna la matriz L del método de la factorización de Cholesky.

Algoritmo 7 Factorización de Cholesky

Entrada: $A \in \mathbb{R}^{n\times n}$ S.D.P

Salida: $L \in \mathbb{R}^{n\times n}$ triangular inferior con elementos en la diagonal positivos

1: **para** $i = 1, 2, \ldots, n$ **hacer**
2: **para** $i = 1, 2, \ldots, k-1$ **hacer**
3: $a_{ki} = l_{ki} = l_{ii}^{-1}\left(a_{ki} - \sum_{j=1}^{i-1} l_{ij}l_{kj}\right)$
4: **fin para**
5: $a_{kk} = l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}}$
6: **fin para**
7: **devolver** $L = A$

El algoritmo 7, se aprecia que al contabilizar el número de multiplicaciones, se realizan cerca de $n^3/6$ multiplicaciones, con lo cual se reduce la mitad que lo realizado con los algoritmos de la factorización LU con y sin pivoteo. Por lo tanto es importante determinar si la matriz es S.D.P. Ya encontrada la factorización se recurre a resolver los sistemas triangulares

$$L\boldsymbol{y} = \boldsymbol{b}, \qquad \text{Ł}^T\boldsymbol{x} = \boldsymbol{y}.$$

Debido a la importancia de conocer si una matriz es S.D.P a continuación un teorema que garantiza que la matriz es S.D.P si es diagonalmente dominante.

Teorema 3.7. *Si una matriz A simétrica, es estrictamente diagonalmente dominante y tiene entradas positivas, entonces A es S.D.P*

Demostración. Para realizar la demostración se probará que se puede proceder la eliminación Gaussiana en A sin intercambio de filas y por tanto la factorización LU es única, (teorema 3.4) implicando la factorización de Cholesky. Para ello se utiliza el principio de inducción sobre n.

Si $k = 1$, entonces con $L = \sqrt{a_{11}}$, $A = (a_{11}) = LL^T$ y esta factorización es única. Suponiendo que se cumple el anterior resultado para $k = 1, 2, \ldots, n-1$, se va probar que se cumple para $k = n$. Se Considera una matriz $A \in \mathbb{R}^{n\times n}$ que cumple con las hipótesis del teorema. Dado que A es simétrica y diagonalmente dominante, a_{11} es el más grande elemento que cualquier otro fuera de la diagonal respecto a la primera fila y primera columna de A. Así, aplicando eliminación Gaussiana en la primera columna de A se tiene que

$$A \to A' = \left[\begin{array}{c|c} a_{11} & \boldsymbol{a}_{1,2:n} \\ \hline \mathbf{0} & A_1 \end{array}\right],$$

donde $A_1 \in \mathbb{R}^{n-1\times n-1}$ y $\boldsymbol{a}_{1,2:n} = (a_{12}\ a_{13} \ldots\ a_{1n})$. Con esto se mostrará que A_1 es también simétrica, estrictamente diagonalmente dominantes y tiene además entradas positivas en la diagonal, esto con el fin de aplicar la hipótesis de inducción.

Iniciando con la simetría desde que A es simétrica, para $i, j \neq 1$ se tiene que

$$\begin{aligned} ent_{ij}(A') &= a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j} \\ &= a_{ji} - \frac{a_{ji}}{a_{11}}a_{1i} = ent_{ji}(A'). \end{aligned}$$

Para ver que la matriz A_1 es diagonalmente dominante se considera la suma de los elementos fuera de la diagonal en la columna $k-1$ de A_1. (están en la columna k de A') Entonces se tiene

$$\sum_{i=2,\neq k} \left|ent_{ik}(A')\right| = \left|a_{2k} - \frac{a_{21}}{a_{11}}a_{1k}\right| + \left|a_{3k} - \frac{a_{31}}{a_{11}}a_{1k}\right| + \ldots + \left|a_{nk} - \frac{a_{n1}}{a_{11}}a_{1k}\right|,$$

lo cual, por la desigualdad en $\mathbb{R}$ ($\forall\ a, b \in \mathbb{R}\ \ |a-b| \leq |a| + |b|$) y la propiedad de A de ser estrictamente diagonalmente dominante para $i = 1, 2, \ldots, n$,

$$|a_{ii}| = a_{ii} > \sum_{k=1,\leq i}^{n} |a_{ki}| \ \rightarrow\ (a_{ii} - |a_{1i}|) > \sum_{k=1,\leq i}^{n} |a_{ki}|,$$

se obtiene

$$\begin{aligned}\sum_{i=2,\neq k} \left|ent_{ik}(A')\right| &< (a_{kk} - |a_{1k}| + \frac{|a_{1k}|}{a_{11}}(a_{11} - |a_{k1}|) \\ &= a_{kk} - \frac{|a_{1k}||a_{k1}|}{a_{11}} \\ &= a_{kk} - \frac{a_{k1}^2}{a_{11}} = ent_{kk}(A') = ent_{k-1k-1}(A_1).\end{aligned}$$

Por lo tanto, A_1 es diagonalmente dominante con elementos positivos en la diagonal. Con este resultado A_1 tiene factorización LU y se supone que es $A_1 = L_1U_1$. Con esto, la matriz A se la puede expresar como

$$\begin{aligned}A &= \left[\begin{array}{c|c} a_{11} & \boldsymbol{a}_{1,2:n} \\ \hline \mathbf{0} & A_1 \end{array}\right] = \left[\begin{array}{c|c} a_{11} & \boldsymbol{a}_{1,2:n} \\ \hline \mathbf{0} & LU \end{array}\right] \\ &= \left[\begin{array}{c|c} 1 & \mathbf{0} \\ \hline \mathbf{0} & L_1 \end{array}\right] \left[\begin{array}{c|c} a_{11} & \boldsymbol{a}_{1,2:n} \\ \hline \mathbf{0} & U_1 \end{array}\right].\end{aligned}$$

Por lo tanto A tiene factorización LU sin pivoteo.

■

3.3. Métodos iterativos

Se denomina métodos iterativos, aquellos métodos que generalmente mediante una serie no exacta de pasos aproximan la solución de un problema específico, en nuestro caso aproximan la solución de un sistema lineal. Por los problemas de redondeo que se suelen presentarse a la hora de aplicar los métodos iterativos, puede que el método no converja y se requiera utilizar otro método o aplicar un precondicionamiento a la matriz del sistema.

A continuación se mostrarán algunos métodos iterativos derivados a partir del método iterativo genérico y se introducirán algunos resultados que garantizan la convergencia de estos métodos.

3.3.1. Método Genérico

Considérese una matriz $A \in \mathbb{R}^{n\times n}$, la cual se la puede descomponer como

$$A = M - N,$$

donde M es una matriz arbitraria no singular y la matriz N se deduce de la diferencia entre $M - A$. Esta descomposición permite resolver un sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, mediante un problema de punto fijo $\boldsymbol{x} = \phi(\boldsymbol{x})$, donde $\phi(\cdot)$ está definido como

$$\phi(\boldsymbol{x}) = M^{-1}N\boldsymbol{x} + M^{-1}\boldsymbol{b}. \tag{3.3.1}$$

Esto se comprueba ya que

$$\begin{aligned}\phi(\boldsymbol{x}) = \boldsymbol{x} &= M^{-1}N\boldsymbol{x} + M^{-1}\boldsymbol{b}\\ &= M^{-1}(M - A)\boldsymbol{x} + M^{-1}\boldsymbol{b}\\ &= \boldsymbol{x} - M^{-1}A\boldsymbol{x} + M^{-1}\boldsymbol{b}\\ &= \boldsymbol{x}M^{-1}(\boldsymbol{b} - A\boldsymbol{x}),\end{aligned}$$

lo que implica que $\mathbf{0} = M^{-1}(\boldsymbol{b} - A\boldsymbol{x})$ y por tanto se obtiene el sistema $A\boldsymbol{x} = \boldsymbol{b}$.

La matriz $M^{-1}N$ que aparece en (3.3.1) recibe el nombre de matriz de iteración. Con el problema de punto fijo $\phi(\boldsymbol{x})$ se puede construir un método iterativo el cual se base en generar una sucesión de vectores $\{\boldsymbol{x}_k\}$ a partir de un vector inicial $\boldsymbol{x}_0$, como

$$\boldsymbol{x}_{k+1} = M^{-1}(N\boldsymbol{x}_k + \boldsymbol{b}).$$

Si se reemplaza $N = M - A$ en $\boldsymbol{x}_{k+1}$, se obtiene

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + M^{-1}(\boldsymbol{b} - A\boldsymbol{x}_k). \tag{3.3.2}$$

Esta última iteración es más utilizada en la práctica ya que involucra la matriz A del problema original. El vector $\boldsymbol{r}_k = \boldsymbol{b} - A\boldsymbol{x}_k$ es el vector residual en la k-ésima iteración. Cuando $\|\boldsymbol{r}_k\| \to 0$ implica que $\boldsymbol{x}_k \to \boldsymbol{x}$, por lo que nos servirá como criterio para determinar cuando terminar el método iterativo.

Observación 3.3. El método iterativo genérico realiza un producto matriz por vector y además requiere calcular la matriz M^{-1}, que son procesos de gran costo computacional. Es por ello que es importante la escogencia de un M para el cual obtener su matriz inversa no sea tan costosa de calcular.

Dado que el método iterativo genérico está determinada por la matriz M, si se denota como $\boldsymbol{x}^*$ la solución exacta y por $\boldsymbol{e}_k = \boldsymbol{x}^* - \boldsymbol{x}_k$ el error de la k-ésima iteración, se tiene

$$\begin{aligned}
\boldsymbol{e}_{k+1} &= \boldsymbol{x}^* - \boldsymbol{x}_{k+1} \\
&= \boldsymbol{x}^* - M^{-1}(N\boldsymbol{x}_k + \boldsymbol{b}) \\
&= M^{-1}(N\boldsymbol{x}^* + \boldsymbol{b}) - M^{-1}(N\boldsymbol{x}_k + b) \\
&= M^{-1}N(\boldsymbol{x}^* - \boldsymbol{x}_{k+1}) \\
&= M^{-1}N\boldsymbol{e}_k \\
&= M^{-1}N(M^{-1}N)^{k-1}\boldsymbol{e}_0 \\
&= (M^{-1}N)^k\boldsymbol{e}_0.
\end{aligned}$$

Por tal motivo la convergencia depende de la matriz

$$M^{-1}N = M^{-1}(M - A) = I - M^{-1}A.$$

Ahora se presenta un teorema que garantiza la convergencia del método genérico dada por la iteración (3.3.2).

Teorema 3.8. *(Convergencia del método genérico) Sea $A \in \mathbb{R}^{n\times n}$ que tiene la descomposición $A = M - N$, con $M \in \mathbb{R}^{n\times n}$ no singular, tal que*
$\rho(M^{-1}N) < 1$, entonces el método iterativo (3.3.2) converge para cualquier valor inicial $\boldsymbol{x}_0$.

Demostración. Por las anteriores relaciones, se pudo deducir que para cualquier valor incial $\boldsymbol{x}_0$ el error $\boldsymbol{e}_k$ satisface que

$$\begin{aligned}
\boldsymbol{e}_{k+1} &= M^{-1}N\boldsymbol{e}_{k+1} \\
&= (M^{-1}N)^k\boldsymbol{e}_0,
\end{aligned}$$

por lo tanto se prueba el teorema si $\rho(M^{-1}N) < 1$, implica que

$$\lim_{k\to\infty}(M^{-1}N)^k = 0.$$

Sea λ un valor propio de $M^{-1}N$, entonces existe un $\boldsymbol{x} \neq 0 \in \mathbb{C}^n$ tal que

$$(M^{-1}N)\boldsymbol{x} = \lambda\boldsymbol{x}.$$

Si se multiplica a ambos lados a izquierda por $(MN^{-1})^{k-1}$ la igualdad anterior, se obtiene que

$$\begin{aligned}
(MN^{-1})^{k-1}(MN^{-1})\boldsymbol{x} = (MN^{-1})^k\boldsymbol{x} &= (MN^{-1})^{k-1}\lambda\boldsymbol{x} \\
&= \lambda\lambda^{k-1}\boldsymbol{x} \\
&= \lambda^k\boldsymbol{x}.
\end{aligned}$$

Sea $\|\cdot\|$ una norma vectorial y $|||\cdot|||$ una norma matricial inducida asociada a $\|\cdot\|$. Aplciando ambos lados $\|\cdot\|$ en la igualdad anterior, se deduce que

$$\|\lambda^k \boldsymbol{x}\| = |\lambda^k| \|x\| \leq ||| A^k ||| \, \|x\|,$$

de donde,

$$|\lambda^k| \leq ||| A^k ||| .$$

Por la hipótesis de que $\rho(M^{-1}N) < 1$ y por la desigualdad anterior

$$\lim_{k\to\infty} |\lambda^k| = 0 \quad \rightarrow \quad \lim_{k\to\infty} ||| A^k ||| = 0.$$

■

Observación 3.4. Del teorema expuesto anteriormente, se puede ver la importancia del radio espectral de la matriz de iteraciones $M^{-1}N$, ya que entre mas cercana sea a cero el método convergerá más rápido.

A continuación se muestran algunos métodos iterativos que se derivan a partir del método genérico.

3.3.2. Método de Richardson

Como en le método genérico se debe calcular la inversa de la matriz M, lo que se desea es tomar una matriz que sea sencilla de calcular este valor. El método de Richardson se basa en la descomposición $M = \dfrac{1}{\alpha} I$, donde $\alpha \in \mathbb{R} - \{0\}$. Así, el método se define como

$$\begin{aligned} x_{k+1} &= \boldsymbol{x}_k + M^{-1}\boldsymbol{r}_k \\ &= \boldsymbol{x}_k + \alpha(\boldsymbol{b} - A\boldsymbol{x}). \end{aligned}$$

El siguiente teorema garantiza la convergencia del método con los valores de α y la propiedad de ser estrictamente diagonalmente dominante.

Teorema 3.9. *Sea A una matriz diagonalmente dominante cuyos valores propios son todos positivos, entonces la iteración del método de Richardson converge para cualquier valor inicial $\boldsymbol{x}_0$ si y sólo si $0 < \alpha < 2$.*

Demostración. Sea $\lambda_i = 1, 2, \ldots, n$ los valores propios de la matriz A, los cuales son positivos. Si se Supone además que estos valores están ordenados del tal forma que

$$0 < \lambda_{min} = \lambda_1 < \lambda_2 < \ldots < \lambda_n = \lambda_{max} = \rho(A).$$

Como $M^{-1}N = I - M^{-1}A = I - \alpha A$, entonces

$$(I - \alpha A)\boldsymbol{x} = \boldsymbol{x} - \alpha A\boldsymbol{x}$$
$$\rho(M^{-1}N) = \rho(I - \alpha A) = 1 - \alpha\rho(A).$$

Sea $\boldsymbol{r}_i$, con $i = 1, 2, \ldots, n$ los valores propios de la matriz $M^{-1}N$ luego se tiene que

$$1 - \alpha\lambda_{max} \leq \boldsymbol{r}_i \leq 1 - \alpha\lambda_{min}, \quad \forall\, i = 1, 2, \ldots, n.$$

Ahora para que se cumpla la convergencia, $\rho(M^{-1}N) < 1$, entonces se debe cumplir que

$$-1 < 1 - \alpha\lambda_{max} \quad \text{y} \quad 1 - \alpha\lambda_{min} < 1,$$

con lo que se deduce que $\alpha < \dfrac{2}{\lambda_{max}}$ y $\alpha > 0$.

■

Del teorema mostrado, se puede apreciar que la idea para mejorar la rapidez del método de Richardson, es buscar un $\alpha_{opt} \neq 0$, tal que $\rho(M^{-1}N)$ sea el más pequeño posible. Por la demostración anterior,

$$\rho(M^{-1}N) = \rho(I - \alpha A) = \text{máx}\{|1 - \alpha_{\text{máx}}|, |1 - \alpha_{\text{mín}}|\}.$$

De esto, α_{opt} debe satisfacer

$$\alpha_{opt}\lambda_{\text{máx}} - 1 = 1 - \alpha_{opt}\lambda_{\text{mín}},$$
$$\alpha_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}}.$$

Para este α_{opt} el radio espectral de MN^{-1} es

$$\begin{aligned}
\rho_{\alpha_{opt}}(M^{-1}N) &= |1 - \alpha_{opt}\lambda_{min}| \\
&= \left|1 - \frac{2}{\lambda_{max} + \lambda_{min}}\lambda_{min}\right| && \left(\alpha_{opt} = \frac{2}{\lambda_{min} + \lambda_{max}}\right) \\
&= 1 - \frac{2\lambda_{min}}{\lambda_{max} + \lambda_{min}} && \left(\frac{\lambda_{min} + \lambda_{min}}{\lambda_{max} + \lambda_{min}} < 1\right) \\
&= \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}.
\end{aligned}$$

A partir de este resultado se puede notar que:

1. Si λ_{max} es muy grande con relación a 1, $\rho(M^{-1}N)$ es muy cercana a 1 y por tanto la convergencia no es rápida.

2. Si A es S.D.P el condicionamiento de A está dado por,

$$k = k_2(A) = ||| A |||_2 ||| A^{-1} |||_2 = \frac{\lambda_{max}}{\lambda_{min}},$$

ya que por un lado,

$$||| A |||_2 = \sqrt{\rho(A^T A} = \sqrt{\rho_{max}(A^2)} = \sqrt{\lambda_{max}^2} = \lambda_{max}$$

y por otro lado, debido a que para cada valor propio λ' de la matriz A^{-1} existe un vector ortogonal, $\boldsymbol{x} \in \mathbb{C}^n$ ($\boldsymbol{x}^T\boldsymbol{x} = 1$) tal que $A^{-1}\boldsymbol{x} = \lambda'\boldsymbol{x}$, es decir $\boldsymbol{x} = \lambda' A\boldsymbol{x}$, y por tanto $\boldsymbol{x} = \lambda'\lambda\boldsymbol{x}$,. Multiplicando a la anterior igualdad por $\boldsymbol{x}^T$ y despejando λ, se obtiene

$$\frac{1}{\lambda} = \lambda'.$$

Con este resultado y de forma análoga como se calculó $||| A |||_2$, se calcula $||| A^{-1} |||_2 = \dfrac{1}{\lambda_{min}}$.

A partir del valor $k = k_2(A)$ el radio espectral de MN^{-1} es

$$\begin{aligned} \rho(M^{-1}N) &= \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} - \lambda_{min}} \\ &= \frac{\dfrac{\lambda_{max}}{\lambda_{min}} - 1}{\dfrac{\lambda_{max}}{\lambda_{min}} + 1} = \frac{k-1}{k+1} \\ &= 1 - \frac{2}{k+1} \end{aligned} \tag{3.3.3}$$

3.3.3. Método de Jacobí

El método de Jacobi generaliza el método de Richardson, debido a que utiliza una matriz diagonal, para intentar reducir un poco el radio espectral de la matriz iterativa $M^{-1}N$. La idea de este método es descomponer la matriz A de la forma

$$A = L + D + U,$$

donde L es estrictamente triangular inferior ($l_{ij} \neq 0,\ \forall\ i > j$), D una matriz diagonal y U una matriz estrictamente triangular superior ($u_{ij} \neq 0,\ \forall\ i < j$).

En este método se toma a M como la diagonal principal de A, es decir,
$M = D = diag\{a_{11}, a_{22}, \ldots, a_{nn}\}$. De esta manera el método de Jacobi se define como

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + M^{-1}(\boldsymbol{r}_k) \\ &= \boldsymbol{x}_k + D^{-1}(\boldsymbol{r}_k), \end{aligned}$$

de donde

$$D^{-1}\boldsymbol{r}_k = \begin{bmatrix} \frac{r_1}{a_{11}} & 0 & \dots & 0 \\ 0 & \frac{r_2}{a_{22}} & \dots & 0 \\ 0 & \dots & \ddots & 0 \\ 0 & 0 & \dots & \frac{r_n}{a_{nn}} \end{bmatrix} \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} = \begin{bmatrix} \frac{r_1}{a_{11}} \\ \frac{r_2}{a_{22}} \\ \vdots \\ \frac{r_n}{a_{nn}} \end{bmatrix}.$$

De lo anterior, se puede notar, que no se usa ningún método para calcular la inversa de M y además, $a_{ii} \neq 0$, $i = 1, 2, \dots, n$ y $-N = (L + U)$, que es equivalente a $N = -(L + U)$.

A continuación se presenta un resultado que garantiza la convergencia del método de Jacobi.

Teorema 3.10. *Si A es una matriz estrictamente diagonalmente dominante, entonces la iteración asociada al método de Jacobi converge para cualquier valor inicial.*

Demostración. Por el método de Jacobí, la matriz de iteración $MN^{-1} = D^{-1}N^{-1} = I - D^{-1}A$. Entonces la matriz MN^{-1} es de la forma

$$\begin{aligned} MN^{-1} &= \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 1 \end{bmatrix} - \begin{bmatrix} 1 & \frac{a_{12}}{a_{11}} & \dots & \frac{a_{1n}}{a_{11}} \\ \frac{a_{21}}{a_{22}} & 1 & \dots & \frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{a_{n1}}{a_{nn}} & \frac{a_{n2}}{a_{nn}} & \dots & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{bmatrix}. \end{aligned}$$

Calculando la norma infinito de $|||\ M^{-1}N\ |||_\infty$ se tiene que

$$\begin{aligned} |||\ M^{-1}N\ |||_\infty &= \max_{i=1,\dots,n} \left(\left|\frac{a_{i1}}{a_{ii}}\right| + \left|\frac{a_{i2}}{a_{ii}}\right| + \dots + \left|\frac{a_{ii-1}}{a_{ii}}\right| + \left|\frac{a_{ii+1}}{a_{ii}}\right| + \dots + \left|\frac{a_{in}}{a_{ii}}\right| \right) \\ &= \max_{i=1,\dots,n} \sum_{j \neq i}^{n} \left|\frac{a_{ij}}{a_{ii}}\right| < 1, \end{aligned}$$

dado que por hipótesis A es estrictamente diagonalmente dominante. De este resultado, si λ es un valor propio de $M^{-1}N$ y $\boldsymbol{x} \in \mathbb{C}^n$ un vector propio de $M^{-1}N$, se obtiene

$$\begin{aligned} &M^{-1}N\boldsymbol{x} = \lambda\boldsymbol{x}, \\ &\|M^{-1}N\boldsymbol{x}\|_\infty = \|\lambda\boldsymbol{x}\|_\infty = |\lambda|\|\boldsymbol{x}\|_\infty, \\ &|\lambda|\|\boldsymbol{x}\|_\infty \leq |||\ M^{-1}N\ |||_\infty\ \|\boldsymbol{x}\|_\infty, \\ &|\lambda| \leq |||\ M^{-1}N\ |||_\infty < 1. \end{aligned}$$

Por lo tanto $\rho(M^{-1}N) < 1$ y el método de Jacobi converge.

■

Observación 3.5. El método de Jacobi se lo puede escribir como

$$\begin{aligned}\boldsymbol{x}_{k+1}(i) &= \boldsymbol{x}_k(i) + \frac{1}{a_{ii}}\left(\boldsymbol{b}_i - \sum_{j=1}^{n} a_{ij}\boldsymbol{x}_k(j)\right) \\ &= \boldsymbol{x}_k(i) - \frac{\boldsymbol{x}_k(i)a_{ii}}{a_{ii}} + \left(\boldsymbol{b}_i - \sum_{j=i\neq i}^{n} a_{ij}\boldsymbol{x}_k(i)\right),\end{aligned}$$

que es equivalente a despejar la variable $\boldsymbol{x}_i$ de la ecuación i-ésima del sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, similar al proceso de back substitution realizado en los métodos directos.

3.3.4. Método de Gauss-Seidel

El método de Gauss-Seidel tiene dos versiones

1. (Forward Gauss-Seidel). Este queda determinado haciendo $M = L+A$ y la iteración nos queda

$$\boldsymbol{x}_{k+1} = (L + D)^{-1}(\boldsymbol{b} - U\boldsymbol{x}_k) \tag{3.3.4}$$

2. (Backward Gauss-Seidel). Este queda determinado haciendo $M = D + U$ y la iteración nos queda

$$\boldsymbol{x}_{k+1} = (D + U)^{-1}(\boldsymbol{b} - L\boldsymbol{x}_k) \tag{3.3.5}$$

Cada iteración se debe resolver un sistema lineal triangular y por lo tanto es más costosa que la de Jacobí.

A continuación se presentan dos teoremas que garantizan la convergencia del método de Gauss-Seidel (dos versiones) converge para cualquier valor inicial. Sin perdida de generalidad se considera en las demostraciones el método de "Forward Gauss-Seidel".

Teorema 3.11. *Sea A una matriz estrictamente diagonalmente dominante, entonces la iteración de Gauss-Seidel converge para todo valor inicial*

Teorema 3.12. *Si A es una matriz simétrica definida positiva (s.d.p) entonces la iteración de Gauss Seidel converge para cualquier valor inicial.*

La demostración a los teoremas presentados puede encontrarse en [8].

3.3.5. Método de SOR "Successive Over Relaxation"

Igual que el método de Jacobi que mejora la convergencia del método de Richardson, el método de SOR, acelera la convergencia de los métodos de Gauss-Seidel.

La idea del método de SOR es utilizar un parámetro w en cual debe ser escogida de forma adecuada para garantizar la convergencia del método. Este método inicia multiplicando el sistema original $A\boldsymbol{x}$, por el parámetro w, es decir, $wA\boldsymbol{x} = \boldsymbol{b}$. Luego se realiza la descomposición $A = L+D+U$ es

$$\begin{aligned} wA &= w(L + D + U) \\ &= wL + D - D + wD + wU \\ &= (D + wL) + ((w - 1)D + wU). \end{aligned} \tag{3.3.6}$$

Con esto la descomposición $A = M - N$, siendo

$$M = \frac{1}{w}(D + wL) = \left(\frac{D}{w} + L\right) \quad \text{y} \quad N = -\frac{1}{w}((w - 1)D + wU) = \left(\frac{1}{w} - 1\right) D - U.$$

El método iterativo de SOR nos queda de la siguiente manera

$$\boldsymbol{x}_{k+1} = M^{-1}N\boldsymbol{x}_k + M^{-1}\boldsymbol{b},$$

$$\boldsymbol{x}_{k+1} = \left(L + \frac{1}{w}D\right)^{-1}\left(\left(\frac{1}{w} - 1\right) D - U\right)\boldsymbol{x}_k + \boldsymbol{b},$$

$$\left(L + \frac{1}{w}D\right)\boldsymbol{x}_{k+1} = \left(\left(\frac{1}{w} - 1\right) D - U\right)\boldsymbol{x}_k + \boldsymbol{b}.$$

Observación 3.6. Si $w = 1$, la iteración de SOR nos queda

$$(L + D)\boldsymbol{x}_{k+1} = -U\boldsymbol{x}_k + \boldsymbol{b} = \boldsymbol{b} - U\boldsymbol{x}_k,$$

que es el método de Forward Gauss-Seidel y de forma análoga se obtiene el método de Gauss-Seidel back substitution, ya que de (3.3.6), wA se lo puede expresar también como

$$wA = (D + wU) + (wL + (w - 1)D)$$

y se toma $M = U + \frac{1}{w}D$ y $N = \left(\left(\frac{1}{w} - 1\right) D - L\right)$.

Ahora se muestran algunos resultados que garantizan la convergencia del método de SOR.

Lema 3.1. *(Kahan). El radio espectral de la matriz de iteración, $M^{-1}N$, del método SOR satisface la siguiente igualdad*

$$\rho(M^{-1}N) \geq |1 - w|.$$

En consecuencia el método diverge si $w \in (-\infty, 0] \cup [2, +\infty)$.

La demostración al lema puede encontrarse en [8].

3.4. Precondicionamiento

Aunque los métodos vistos para resolver sistemas lineales teóricamente garantizan la convergencia a la solución, todos son propensos a que por problemas de redondeo el método diverja o sufra una convergencia lenta para problemas que surgen de las aplicaciones en los que el sistema está muy mal condicionado o las entradas son muy grandes. Con la ayuda del precondicionamiento se puede mejorar o resolver estos problemas.

Un precondicionamiento significa transformar el sistema lineal original hacia uno que tenga la misma solución pero el cual sea mejor condicionado y así aumente la posibilidad para determinar una solución con un método iterativo o uno directo.

Para realizar un precondicionamiento es necesario encontrar un precondicionador que es una matriz M, la cual puede ser definida de diferentes maneras pero debe satisfacer unos requerimientos mínimos que son

- Que M no deba ser tan costosa para resolver el sistema $M\boldsymbol{x} = b$, esto es debido a que los algoritmos precondicionados requieren en algunos casos un sistema solución con la matriz M en cada paso.
- M debe ser cercana a A en algún sentido y esta deberá ser claramente no singular.

Cuando se tiene el precondicionador M hay 3 maneras de aplicarlo a un sistema $A\boldsymbol{x} = \boldsymbol{b}$. El precondicionamiento puede ser aplicado a la izquierda, que nos da el sistema precondicionado

$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}.$$

Alternativamente, puede ser aplicado a la derecha

$$AM^{-1}\boldsymbol{u} = \boldsymbol{b}, \qquad \boldsymbol{x} = M^{-1}\boldsymbol{u}.$$

Lo anterior requiere hacer el cambio de variable $\boldsymbol{u} = M\boldsymbol{x}$ y resolver el sistema con respecto a $\boldsymbol{u}$. Finalmente, otro caso es cuando el precondicionador esta factorizado en la forma

$$M = M_L M_R,$$

donde, generalmente M_L y M_R son matrices triangulares. En esta situación, el precondicionamiento puede ser dividido

$$M_L^{-1} A M_R^{-1} \boldsymbol{u} = M_L^{-1} \boldsymbol{b} \qquad \boldsymbol{x} = M_R^{-1} \boldsymbol{u}.$$

Es importante preservar la simetría cuando la matriz original es simétrica, entonces para este caso, el precondicionamiento dividido es obligatorio. Sin embargo hay otras maneras de preservar la simetría, o más bien tomar ventaja de la simetría, incluso si M no es posible de factorizar. Esto se observa al aplicarlo al método del gradiente conjugado (**CG**) y en el método gradiente biconjugado **BCG** que se abordará en el capítulo 4.

3.4.1. Precondicionadores del método genérico

Como se ha visto, la forma genérica de los métodos iterativos para resolver un sistema $A\boldsymbol{x} = b$ es de la forma $\boldsymbol{x}_{k+1} = M^{-1}N\boldsymbol{x}_k + M^{-1}\boldsymbol{b}$, donde $A = M - N$. Esto también se lo puede expresar de la forma

$$\boldsymbol{x}_{k+1} = G\boldsymbol{x}_k + f, \tag{3.4.1}$$

donde $f = M^{-1}\boldsymbol{b}$ y

$$G = M^{-1}N = M^{-1}(M - A) = I - M^{-1}A.$$

Así, para los métodos visto como Jacobi, Gauss Seidel y SOR, se tiene que

$$G_{JA}(A) = I - D^{-1}A,$$

$$G_{GS}(A) = I - (L + D)^{-1}A,$$

$$G_{SOR}(A) = I - \left(L + \frac{1}{w}D\right)A.$$

Si se observa la expresión (3.4.1), cuando $\boldsymbol{x} = \boldsymbol{x}_k$ converge a la solución es decir, se presenta una iteración de punto fijo, entonces

$$\boldsymbol{x} = G\boldsymbol{x} + f,$$

$$(I - G)\boldsymbol{x} = f,$$

pero

$$G = I - M^{-1}A \quad \text{y} \quad f = M^{-1}b,$$

por lo que se obtiene

$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}.$$

La última expresión nos quiere decir que el $M = D$, $M = L + D$, y $M = L + \frac{1}{w}D$ de los métodos de Jacobi, Gauss Seidel y SOR respectivamente actúan también como precondicionadores al sistema $A\boldsymbol{x} = \boldsymbol{b}$.

Capítulo 4

Métodos del Gradiente

Dada la importancia que tiene la resolución de sistemas lineales, un trabajo constante ha sido la búsqueda de métodos tanto directos como iterativos que permitan obtener su solución o una aproximación de estas de forma mas rápida y eficiente. En el año de 1950 se empezó a tratar con un método que se conoce hoy en día como el *método del gradiente conjugado*, cuyos pioneros fueron Hestenes y Stiefel [18], sin embargo, el interés actual se enfoca en el carácter iterativo de este método, el cual fue planteado por Ried (1971). Este método se aplica específicamente a sistemas lineales de la forma $A\boldsymbol{x} = \boldsymbol{b}$, donde A es simétrica definida positiva (S.D.P). Para abordar más casos donde A no es simétrica se ha creado una versión más general de este método conocido como *gradiente biconjugado*. Además de otras versiones mas eficientes respecto a la rapidez de la convergencia que es el método *gradiente biconjugado cuadrado* [28] y el *gradiente biconjugado estabilizado* [31].

La característica de estos métodos es la presencia en gran medida de producto matriz por vector y además en el caso de que la matriz del sistema sea S.D.P se garantiza la convergencia a lo mas en n número de pasos o iteraciones, donde n es el orden de la matriz A. Sin embargo, puesto que como se maneja aproximación de cifras, en algunas ocasiones, el método no funciona de forma exacta a como se esperaría que lo haga. Para mejorar estos métodos de gradientes se utiliza en muchos casos precondicionadores que permiten mejorar el error y la convergencia de estos.

En este capítulo se abordarán los métodos de gradiente, que se mencionan a continuación: el gradiente conjugado (**CG**), el gradiente biconjugado (**BCG**), el gradiente conjugado cuadrado (**CGS**) y el biconjugado estabilizado (**BICGSTAB**). Además como se explicó en el capítulo 3, para acelerar la convergencia de los métodos se utilizarán precondicionamientos de forma implícita en el algoritmo del **CG** y en el **BCG**, utilizando el precondicionador de Jacobí. La teoría, teoremas y algoritmos presentados han sido tomados de [25, 26, 16] y [8].

4.1. Método del Gradiente

Es una técnica iterativa que se basa en resolver el problema de minimización de una función de una forma cuadrática. Este método se aplica a sistemas lineales, en particular a problemas donde A es S.D.P.

Inicialmente se va considera la siguiente función:

$$f(\boldsymbol{x}) = \frac{1}{2} < A\boldsymbol{x}, \boldsymbol{x} > - < \boldsymbol{b}, \boldsymbol{x} > . \tag{4.1.1}$$

A partir de (4.1.1), se puede ver que la solución del sistema $A\boldsymbol{x} = \boldsymbol{b}$ es un mínimo para la función $f(\boldsymbol{x})$. Esto se comprueba de la siguiente manera.

Sea $\boldsymbol{x}^*$ la solución del sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, donde A es (S.D.P). Entonces, para cada $\boldsymbol{h} \in \mathbb{R}^n$ se tiene que

$$\begin{aligned} f(\boldsymbol{x}^* + \boldsymbol{h}) &= \frac{1}{2} < A(\boldsymbol{x}^* + \boldsymbol{h}), \boldsymbol{x}^* + \boldsymbol{h} > - < \boldsymbol{b}, \boldsymbol{x}^* + \boldsymbol{h} > \\ &= \frac{1}{2} (< A\boldsymbol{x}^*, \boldsymbol{x}^* > + < A\boldsymbol{x}^*, \boldsymbol{h} > + < A\boldsymbol{h}, \boldsymbol{x}^* > + < A\boldsymbol{h}, \boldsymbol{h} >) \\ &\quad - < \boldsymbol{b}, \boldsymbol{x}^* > - < \boldsymbol{b}, \boldsymbol{h} > \\ &= f(\boldsymbol{x}^*) + \frac{1}{2}(2 < A\boldsymbol{x}^*, \boldsymbol{h} >) + \frac{1}{2} < A\boldsymbol{h}, \boldsymbol{h} > - < \boldsymbol{b}, \boldsymbol{h} > \\ &= f(\boldsymbol{x}^*) + \frac{1}{2} < A\boldsymbol{h}, \boldsymbol{h} > + < A\boldsymbol{x}^* - \boldsymbol{b}, \boldsymbol{h} > \\ &= f(\boldsymbol{x}^*) + \frac{1}{2} < A\boldsymbol{h}, \boldsymbol{h} > . \end{aligned}$$

Luego, como A es S.D.P $\boldsymbol{h}^T A\boldsymbol{h} =< A\boldsymbol{h}, \boldsymbol{h} >\geq 0$ entonces, $f(\boldsymbol{x}^* + \boldsymbol{h}) - f(\boldsymbol{x}^*) = \frac{1}{2} < A\boldsymbol{h}, \boldsymbol{h} >\geq 0$, por lo tanto $\boldsymbol{x}^*$ minimiza la función.

Recíprocamente es cierto también. Si se supone que la función $f(\boldsymbol{x})$ se minimiza en $\boldsymbol{x}^*$, luego existe un $\delta > 0$ tal que para que para todo $\boldsymbol{h} \in B(0, \delta)$, el conjunto de vectores reales tales que $\|\boldsymbol{h}\| < \delta$, se tiene que

$$f(\boldsymbol{x}^* + \boldsymbol{h}) \geq f(\boldsymbol{x}^*). \tag{4.1.2}$$

Para un $\boldsymbol{h}$ fijo, se considera la función $\phi_{\boldsymbol{h}} : (-1, 1) \rightarrow \mathbb{R}$ definida por

$$\phi_{\boldsymbol{h}}(\alpha) = f(\boldsymbol{x}^* + \alpha\boldsymbol{h}) - f(x^*).$$

Por la manera de la que esta definida $\phi_{\boldsymbol{h}}$, mediante la desigualdad (4.1.2) se cumple que para cualquier $\alpha \in (-1, 1)$, entonces $\phi_h(0) \geq 0$. En efecto, como $\alpha \in (-1, 1)$, entonces $|\alpha| < 1$ y

$$|\alpha\|\boldsymbol{h}\| = \|\alpha\boldsymbol{h}\| < \|\boldsymbol{h}\| < \delta.$$

Por lo tanto $\alpha \boldsymbol{h} \in B(0,\delta)$ con lo cual se cumple la desigualdad 4.1.2), en dicho intervalo.

De lo anterior $\phi_{\boldsymbol{h}}$ tiene un mínimo en $\alpha = 0$, dado que $\phi_h(\alpha) = 0$. Entonces,

$$\begin{aligned} 0 = \phi'_{\boldsymbol{h}}(0) &= \left(\frac{1}{2}\alpha^2 \boldsymbol{h}^T A\boldsymbol{h} + \alpha < A\boldsymbol{x}^* - \boldsymbol{b}, \boldsymbol{h} >\right)'_0 \\ &= (\alpha < A\boldsymbol{h}, \boldsymbol{h} > + < A\boldsymbol{x}^* - \boldsymbol{b}, \boldsymbol{h} >)_0 \\ &=< A\boldsymbol{x}^* - \boldsymbol{b}, \boldsymbol{h} >= \boldsymbol{h}^T(A\boldsymbol{x}^* - \boldsymbol{b}). \end{aligned}$$

Como lo anterior se cumple para cualquier $\boldsymbol{h} \in B(0,\delta)$, si se toma $\boldsymbol{h} = \beta \dfrac{\boldsymbol{e}_i}{\| \boldsymbol{e}_i \|}$, para algún $i = 1, 2, ..., n$ y $\beta < \delta$, se obtiene que

$$0 = \boldsymbol{h}^T(A\boldsymbol{x}^* - \boldsymbol{b}) = (A\boldsymbol{x}^* - \boldsymbol{b})_i,$$

lo que implica que $\boldsymbol{b}_i = (A\boldsymbol{x}^*)_i$. Como se debe cumplir para cualquier $i = 1, 2, ..., n$, se deduce que $A\boldsymbol{x}^* = \boldsymbol{b}$ y se concluye que $\boldsymbol{x}^*$ es la solución del sistema $A\boldsymbol{x} = \boldsymbol{b}$.

Ahora lo que se busca es encontrar un vector $\boldsymbol{x}^*$ el cual se aproxime al mínimo de la función $f(\boldsymbol{x})$, que es equivalente aproximar la solución del sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$. Para ello se toma a $\boldsymbol{x}_0$ como la aproximación inicial para crear una sucesión de vectores $(\boldsymbol{x}_k)$ que aproximen al vector que minimiza a $f(\boldsymbol{x})$. La sucesión es la siguiente:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_x, \tag{4.1.3}$$

donde $\boldsymbol{d}_k$ es un vector de dirección de búsqueda y α_k es un escalar. El valor de $\boldsymbol{d}_k$ va estar dado por la dirección opuesta del gradiente, es decir, $\boldsymbol{d}_k = - \bigtriangledown f(\boldsymbol{x}_k) = \boldsymbol{b} - A\boldsymbol{x}_k$. Para la derivada parcial de f con respecto a la variable i-ésima, se tiene que

$$\begin{aligned} \frac{\partial f}{\partial x_i} &= \frac{\partial \left(\frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{x}^T\boldsymbol{b}\right)}{\partial x_i} \\ &= \frac{1}{2}\left(a_{1i}x_1 + a_{2i}x_2 + ... + a_{i-1}x_{i-1} + (A\boldsymbol{x})_i + a_{ii}x_i + ... + a_{ni}x_n\right) - \boldsymbol{b}(i) \\ &= \frac{1}{2}(2(A\boldsymbol{x})_i) - \boldsymbol{b}(i) \qquad (A = A^T) \\ &= (A\boldsymbol{x} - \boldsymbol{b})_i. \end{aligned}$$

Por lo tanto $\boldsymbol{d}_k = - \bigtriangledown f(\boldsymbol{x}_k) = -(A\boldsymbol{x}_k - \boldsymbol{b})$. Ahora para determinar α_k se considera la función

cuadrática definida por:

$$\begin{aligned}
\phi(\alpha) &= f(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k) \\
&= \frac{1}{2} < A(\boldsymbol{x}_k + \alpha \boldsymbol{d}_k), \boldsymbol{x}_k + \alpha \boldsymbol{d}_k > - < \boldsymbol{b}, \boldsymbol{x}_k + \alpha \boldsymbol{d}_k > \\
&= \frac{1}{2} < A\boldsymbol{x}_k, \boldsymbol{x}_k + \alpha + \alpha \boldsymbol{d}_k > + \frac{1}{2} < A\alpha \boldsymbol{d}_k, \boldsymbol{x}_k + \alpha \boldsymbol{d}_k > - < \boldsymbol{b}, \boldsymbol{x}_k > - < \boldsymbol{b}, \alpha \boldsymbol{d}_k > \\
&= f(\boldsymbol{x}_k) + \frac{1}{2} < A\boldsymbol{x}_k, \alpha \boldsymbol{d}_k > + \frac{1}{2} < A\alpha \boldsymbol{d}_k, \alpha \boldsymbol{d}_k > + \frac{1}{2} < A\alpha \boldsymbol{d}_k, \boldsymbol{x}_k > - \alpha < \boldsymbol{b}, \boldsymbol{d}_k > \\
&= \phi(0)\alpha < \frac{1}{2} A\boldsymbol{x}_k - \boldsymbol{b}, \boldsymbol{d}_k > + \frac{\alpha}{2} \boldsymbol{d}_k^T A\boldsymbol{x}_k + \frac{\alpha^2}{2} < A\boldsymbol{d}_k, \boldsymbol{d}_k > \\
&= \phi(0) + \alpha < \frac{1}{2} A\boldsymbol{x}_k - \boldsymbol{b}, \boldsymbol{d}_k > + \frac{\alpha}{2} < A\boldsymbol{x}_k, \boldsymbol{d}_k > + \frac{\alpha^2}{2} < A\boldsymbol{d}_k, \boldsymbol{d}_k > \\
&= \phi(0) - \alpha < \boldsymbol{b} - A\boldsymbol{x}_k, \boldsymbol{d}_k > + \frac{\alpha^2}{2} < A\boldsymbol{d}_k, \boldsymbol{d}_k > \\
&= \phi(0) - \alpha < \boldsymbol{d}_k, \boldsymbol{d}_k > + \frac{1}{2}\alpha^2 < A\boldsymbol{d}_k, \boldsymbol{d}_k > \qquad\qquad (\boldsymbol{d}_k = \boldsymbol{b} - A\boldsymbol{x}_k).
\end{aligned}$$

Como $\phi(\alpha)$ es un polinomio de grado 2 y el coeficiente de α^2 es positivo, entonces el mínimo de esta función esta dado por

$$\alpha_k = -\frac{- < \boldsymbol{d}_k, \boldsymbol{d}_k >}{2(\frac{1}{2} < A\boldsymbol{d}_k, \boldsymbol{d}_k >)} = \frac{< \boldsymbol{d}_k, \boldsymbol{d}_k >}{< A\boldsymbol{d}_k, \boldsymbol{d}_k >}.$$

De lo anterior se observa que:

1. $\boldsymbol{d}_k = r_k$ (residuo de la k-ésima iteración).

2. $f(\boldsymbol{x}_k) = \phi(0) \;\rightarrow\; f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k)$.

   Esto se justifica ya que, de (4.1.3) se tiene que:

$$\begin{aligned}
f(\boldsymbol{x}_{k+1}) &= f(\boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k) \\
&= f(\boldsymbol{x}_k) - \alpha_k < \boldsymbol{d}_k, \boldsymbol{d}_k > + \frac{1}{2}\alpha_k^2 < A\boldsymbol{d}_k, \boldsymbol{d}_k > \\
&= f(\boldsymbol{x}_k) - \frac{< \boldsymbol{d}_k, \boldsymbol{d}_k >^2}{< A\boldsymbol{d}_k, \boldsymbol{d}_k >} + \frac{1}{2}\frac{< \boldsymbol{d}_k, \boldsymbol{d}_k >^2}{< A\boldsymbol{d}_k, \boldsymbol{d}_k >} \qquad \text{(Reemplazando } \alpha_k\text{)}. \\
&= f(\boldsymbol{x}_k) - \frac{1}{2}\frac{< \boldsymbol{d}_k, \boldsymbol{d}_k >^2}{< A\boldsymbol{d}_x, \boldsymbol{d}_k >}.
\end{aligned}$$

A partir de esta iteración se considera el siguiente algoritmo para este método:

Algoritmo 8 Método del gradiente

Entrada: : Matriz A, vector $\boldsymbol{b}$ y vector inicial $\boldsymbol{x}_0$

Salida: : vector $\boldsymbol{x}_k$, solución del método

1: $\boldsymbol{x} = \boldsymbol{x}_0$
2: r $= \boldsymbol{b} - A\boldsymbol{x}$
3: **mientras** $\boldsymbol{r} \neq 0$ **hacer**
4: $\quad \alpha = \dfrac{< \boldsymbol{r}, \boldsymbol{r} >}{< A\boldsymbol{r}, \boldsymbol{r} >}$.
5: $\quad \boldsymbol{x} = \boldsymbol{x} + \alpha \boldsymbol{r}$.
6: $\quad \boldsymbol{r} = \boldsymbol{b} - A\boldsymbol{x}$
7: **fin mientras**
8: **devolver** vector $\boldsymbol{x}_k$

Ejemplo 4.1.

Considere el sistema $A\boldsymbol{x} = \boldsymbol{b}$, con $A = \begin{bmatrix} 2 & 1 \\ 1 & 5 \end{bmatrix}$, $\boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ y $\boldsymbol{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

Solucionando el sistema $A\boldsymbol{x} = \boldsymbol{b}$ con el algoritmo 8, una tolerancia de 1E-12 y $\boldsymbol{x}_0 = 0$, se encuentra que el método converge en la iteración número catorce

$$x_{14} = \begin{pmatrix} 0.444444444444444 \\ 0.111111111111111 \end{pmatrix}.$$

En la siguiente gráfica se muestra el comportamiento de las iteraciones, tratando de encontrar el mínimo de la función

$$f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^2 + \boldsymbol{y}\boldsymbol{x} + \frac{5}{2}\boldsymbol{y}^2 - \boldsymbol{x} - \boldsymbol{y}$$

que es equivalente a resolver el sistema $A\boldsymbol{x} = \boldsymbol{b}$.

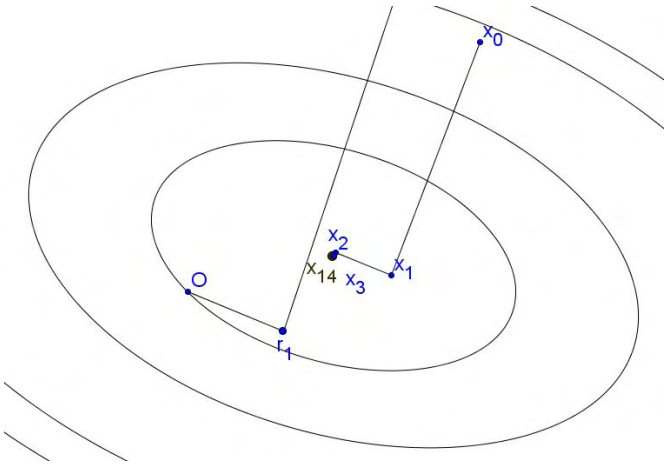


Figura 4.1: Minización de $f(x, y)$ con el Met. Gradiente, ejemplo 1.

4.2. Gradiente Conjugado

El método del Gradiente Conjugado es un método eficiente para resolver un sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, donde la matriz del sistema $A \in \mathbb{R}^{n\times n}$ es S.D.P. Para esté método se necesita de la teoría vista del método del gradiente y algunos conceptos como son los vectores conjugados respecto a una matriz.

Definición 4.1. Sea $A \in \mathbb{R}^{n\times n}$ una matriz S.D.P. Los vectores $\boldsymbol{x}$ y $\boldsymbol{y} \in \mathbb{R}^n$, son $A-$*conjugados* si y solo si,

$$< A\boldsymbol{x}, \boldsymbol{y} >=< \boldsymbol{x}, A\boldsymbol{y} >= 0$$

Observación 4.1. Sea $f = \{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1}\}$ es una familia de vectores no nulos y A-conjugados entonces:

1. La familia $f = \{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1}\}$ es linealmente independiente. Sean $\alpha_0, \alpha_2, \ldots, \alpha_{n-1} \in \mathbb{R}$ tal que,

   $$\alpha_0\boldsymbol{p}_0 + \alpha_1\boldsymbol{p}_1 + \ldots + \alpha_i\boldsymbol{p}_i + \ldots + \alpha_{n-1}\boldsymbol{p}_{n-1} = 0.$$

   Multiplicando por $\boldsymbol{p}_i^T A$ para algún $i = 0, 2, ..., n-1$, se tiene que

   $$\boldsymbol{p}_i^T A\alpha_0\boldsymbol{p}_0 + \boldsymbol{p}_i^T A\alpha_1\boldsymbol{p}_i^T A\boldsymbol{p}_1 + \ldots + \boldsymbol{p}_i^T A\alpha_i\boldsymbol{p}_i + \ldots + \boldsymbol{p}_i^T A\alpha_{n-1}\boldsymbol{p}_{n-1} = 0.$$

   Pero por ser f una familia A-conjugado entonces finalmente se obtiene que

   $$\alpha_i\boldsymbol{p}_i^T A\boldsymbol{p}_i = 0 \ \rightarrow \ \alpha_i = 0 \quad (A \text{ es S.D.P}).$$

   Como se cumple para todo $i = 0, 1, 2, \ldots, n-1$ entonces,

   $$\alpha_0 = \alpha_1 = \alpha_2 = \ldots = \alpha_{n-1} = 0.$$

   De esta manera se concluye que la familia f es una base para $\mathbb{R}^n$.

2. La solución $\boldsymbol{x}^*$ del sistema lineal $A\boldsymbol{x} = 0$ satisface la ecuación

   $$\boldsymbol{x}^* = \sum_{i=0}^{n-1} \alpha_i\boldsymbol{p}_i, \quad \text{donde } \alpha_i = \frac{< A\boldsymbol{x}^*, \boldsymbol{p}_i >}{< A\boldsymbol{p}_i, \boldsymbol{p}_i >} = \frac{< \boldsymbol{b}, \boldsymbol{p}_i >}{< A\boldsymbol{p}_i, \boldsymbol{p}_i >}$$

   Para llegar a esta ecuación se debe tener en cuenta que por la observación 1 la familia f es una base para R^n entonces $\boldsymbol{x}^*$ es de la forma

   $$\boldsymbol{x}^* = \sum_{i=0}^{n-1} \alpha_i\boldsymbol{p}_i.$$

   Si se multiplica por $\boldsymbol{p}_k^T A$ para algún $k = 1, 2, 3, \ldots, n-1$ se deduce que

   $$\boldsymbol{p}_k^T\boldsymbol{b} = \boldsymbol{p}_k^T A\boldsymbol{x}^* = \boldsymbol{p}_k^T A\sum_{i=0}^{n-1} \alpha_i\boldsymbol{p}_i = \alpha_k\boldsymbol{p}_k^T A\boldsymbol{p}_k \quad (f\text{es una familia } A-\text{conjugados}).$$

   Despejando α_k se obtiene el resultado, para todo $k = 0, 1, 2, \ldots, n-1$.

Como se puede ver α_i requiere de datos conocidos, por lo que la solución del sistema se puede obtener conociendo una base de vectores A-conjugados. A continuación se va presentar un lema con el fin de crear un algoritmo que permita obtener la solución del sistema lineal a partir de una base de vectores A-conjugados y un vector inicial $\boldsymbol{x}_0$ arbitrario. Este algoritmo tiene la propiedad que converge en un número finito de pasos.

Lema 4.1. *Sea $f = \{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{n-1}\}$ una base formada por direcciones A-conjugadas y $\boldsymbol{x}_0$ un vector arbitrario de $\mathbb{R}^n$. La sucesión generada por:*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \quad \alpha_k = -\frac{< \boldsymbol{p}_k, \boldsymbol{r}_k >}{< A\boldsymbol{p}_k, \boldsymbol{p}_k >} \tag{4.2.1}$$

calcula la solución $\boldsymbol{x}^*$ del sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, con A S.D.P, a lo más en n iteraciones.

Demostración. Como $f = \{\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{n-1}\}$ es una base de $\mathbb{R}^n$ (observación 4.1) entonces existen escalares β_i, $i = 0, 1, 2, \ldots, n-1$, tales que

$$\boldsymbol{x}^* - \boldsymbol{x}_0 = \sum_{n=0}^{n-1} \beta_i \boldsymbol{p}_i. \tag{4.2.2}$$

Se va encontrar los escalares β_i, $i = 0, 1, ..., n-1$. Para ello, se observa que por la construcción de iteración 4.2.1, $\boldsymbol{x}_k = \boldsymbol{x}_0 + \sum_{i=0}^{k-1} \alpha_i \boldsymbol{p}_i$, por lo cual cada vector $\boldsymbol{x}_k - \boldsymbol{x}_0 \in span\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{k-1}\}$. A partir de esto

$$\begin{aligned} 0 =& < A\boldsymbol{p}_k, \sum_{i=0}^{k-1} \alpha_i \boldsymbol{p}_i > = < A\boldsymbol{p}_k, \boldsymbol{x}_k - \boldsymbol{x}_0 > \\ & = < \boldsymbol{p}_k, A(\boldsymbol{x}_k - \boldsymbol{x}_0) > . \quad (A = A^T) \end{aligned}$$

Luego,

$$\begin{aligned} 0 =& < A\boldsymbol{p}_k, \boldsymbol{x}_k - \boldsymbol{x}_0 > \\ =& < A\boldsymbol{p}_k, \boldsymbol{x}_k - \boldsymbol{x}^* > + < A\boldsymbol{p}_k, \boldsymbol{x}^* - \boldsymbol{x}_0 > \\ =& < A\boldsymbol{p}_k, \boldsymbol{x}_k - \boldsymbol{x}^* > + < A\boldsymbol{p}_k, \sum_{i=0}^{n-1} \beta_i \boldsymbol{p}_i > \\ =& < A\boldsymbol{p}_k, \boldsymbol{x}_k - \boldsymbol{x}^* > + \beta_k < A\boldsymbol{p}_k, \boldsymbol{p}_k > . \qquad (\text{f es una familia } A - conjugados) \end{aligned}$$

Despejando β_k, se tiene

$$\begin{aligned} \beta_k &= -\frac{< \boldsymbol{A}p_k, \boldsymbol{x}_k - \boldsymbol{x}^* >}{< A\boldsymbol{p}_k, \boldsymbol{p}_k >} = -\frac{< \boldsymbol{p}_k, A(\boldsymbol{x}_k - \boldsymbol{x}^*) >}{< A\boldsymbol{p}_k, \boldsymbol{p}_k >} \\ &= -\frac{< \boldsymbol{p}_k, \boldsymbol{r}_k >}{< A\boldsymbol{p}_k, \boldsymbol{p}_k >}. \quad (\boldsymbol{r}_k = A(\boldsymbol{x}_k - \boldsymbol{x}^*)) \end{aligned}$$

Por lo tanto de la igualdad (4.2.2) se obtiene

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \quad \text{donde} \quad \alpha_k = \beta_k = -\frac{< \boldsymbol{p}_k, \boldsymbol{r}_k >}{< A\boldsymbol{p}_k, \boldsymbol{p}_k >}.$$

Por ser f una base para $x^* - x_0$, implica que la iteración 4.1.3 converge a la solución del sistema $A\boldsymbol{x} = \boldsymbol{b}$ a lo más en n pasos.

■

Por este lema, el método del gradiente conjugando necesita de una familia de vectores conjugados, que se van construyendo a medida que se avanza en las iteraciones, partiendo vector inicial $\boldsymbol{x}_0$. La característica de este método es que para cada vector $\boldsymbol{p}_k$, se lo construye usando solo el vector previo $\boldsymbol{p}_{k-1}$. Esto nos índica que no se necesita saber los elementos previos $\boldsymbol{p}_0, \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{k-2}$ de la base conjugada y por tanto $\boldsymbol{p}_k$ es automáticamente conjugado a estos vectores.

Para la elección de cada vector $\boldsymbol{p}_k$ se escogen de tal manera que sean una combinación del vector de dirección de descenso ($\bigtriangledown\phi(\boldsymbol{x}_k) = Ax - b)) = r(x)$) y la dirección previa $\boldsymbol{p}_{k-1}$. Se escribe

$$\boldsymbol{p}_k = -\boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1}, \tag{4.2.3}$$

donde β_k será determinado por el requerimiento de que $\boldsymbol{p}_{k-1}$ y $\boldsymbol{p}_k$ deben ser conjugados con respecto a A. Multiplicando (4.2.3) por $\boldsymbol{p}_{k-1}^T A$ y aplicando la condición de ser A-conjugado ($\boldsymbol{p}_{k-1}^T A\boldsymbol{p}_k = 0$), se tiene que

$$\boldsymbol{p}_{k-1}^T A\boldsymbol{p}_k = -\boldsymbol{p}_{k-1}^T A\boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1}^T A\boldsymbol{p}_{k-1} = 0 \quad \rightarrow \quad \beta_k = \frac{\boldsymbol{p}_{k-1}^T A\boldsymbol{r}_k}{\boldsymbol{p}_{k-1}^T A\boldsymbol{p}_{k-1}}.$$

Para iniciar se considera $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b} = \boldsymbol{p}_0$ como el vector inicial para construir la familia de vectores A-conjugados. A continuación se presenta el algoritmo 14 que el algoritmo preliminar del método del gradiente conjugado.

Algoritmo 9 G.C versión preliminar

Entrada: : Matriz A, vector $\boldsymbol{b}$ y vector inicial $\boldsymbol{x}_0$

Salida: : vector x_k, solución del método

1: $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$
2: $\boldsymbol{p}_0 = -\boldsymbol{r}_0$
3: $k = 0$
4: **mientras** $\boldsymbol{r}_k \neq 0$ **hacer**
5: $\quad \alpha_k = \dfrac{-\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$
6: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
7: $\quad \boldsymbol{r}_{k+1} = A\boldsymbol{x}_{k+1} - \boldsymbol{b}$
8: $\quad \beta_{k+1} = \dfrac{\boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$
9: $\quad \boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
10: $\quad k = k + 1$
11: **fin mientras**
12: **devolver** vector $\boldsymbol{x}_k$

Mas adelante se prueba que las direcciones $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{n-1}$ son efectivamente conjugadas y entonces el algoritmo 14 termina a lo más en n pasos. Mediante el teorema siguiente dr establece esta propiedad y otros dos importantes propiedades que son: los residuos $\boldsymbol{r}_i$ son mutuamente ortogonales; cada búsqueda de dirección $\boldsymbol{p}_k$ y el residual $\boldsymbol{r}_k$ están contenidos en el subespacio de krylov de grado k por $\boldsymbol{r}_0$ definido como

$$K(\boldsymbol{r}_0; k) = span\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^k \boldsymbol{r}_0\}.$$

Teorema 4.1. *Sea $\boldsymbol{x}_0 \in \mathbb{R}^n$ cualquier valor inicial y supóngase que la sucesión de vectores $\{\boldsymbol{x}_k\}$ es generada por la secuencia del lema 4.1. Entonces*

$$\boldsymbol{r}_k^T \boldsymbol{p}_i = 0, \qquad para\ i = 0, \ldots, k-1$$

y $\boldsymbol{x}_k$ es el el mínimo de $\phi(\boldsymbol{x}) = \dfrac{1}{2} < A\boldsymbol{x}, \boldsymbol{x} > - < \boldsymbol{b}, \boldsymbol{x} >$ respecto al conjunto

$$\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \in \boldsymbol{x}_0 + span\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{k-1}\}\}$$

.

Demostración. Se iniciará mostrando que un vector $\boldsymbol{x}$ minimiza ϕ sobre el conjunto

$$\{\boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{x} \in \boldsymbol{x}_0 + span\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_{k-1}\},$$

si y sólo si $\boldsymbol{r}_{\boldsymbol{x}^\sim}^T \boldsymbol{p}_i = 0$, para cada $i = 0, 1, \ldots, k-1$. Se Define $h(\boldsymbol{\sigma}) = \phi(x_0 + \sigma_0 \boldsymbol{p}_0 + \ldots + \sigma_{k-1}\boldsymbol{p}_{k-1})$, donde $\boldsymbol{\sigma} = (\sigma_0, \sigma_1, \ldots, \sigma_{k-1})^T$. Desde que $h(\boldsymbol{\sigma})$ es un polinomio cuadrático convexo y por ende tiene un único minimizador σ^* que satisface

$$\frac{\partial \boldsymbol{\sigma}^*}{\partial \sigma_i} = 0, \qquad i = 0, 1, \ldots, k-1.$$

Por regla de la cadena, implica que

$$\bigtriangledown h(\boldsymbol{\sigma}^*) = \bigtriangledown \phi(\boldsymbol{x}_0 + \sigma_0^* \boldsymbol{p}_0 + \ldots + \sigma_{k-1}^* \boldsymbol{p}_{k-1})^T \boldsymbol{p}_i = 0, \quad i = 0, 1, \ldots, k-1.$$

Como $\bigtriangledown \phi(\boldsymbol{x}^\sim) = A(\boldsymbol{x}^\sim) - b = r_{\boldsymbol{x}^\sim}$ entonces, se obtiene el resultado requerido.

Ahora se usará inducción para mostrar que $\boldsymbol{x}_k$ satisface $\boldsymbol{r}_k^T \boldsymbol{p}_i = 0$, para $i = 0, 1, \ldots, k-1$. Para $k = 1$ se cumple que, $r_1^T p_0$, esto se debe dado que, por la iteración (4.2.1)

$$\begin{aligned} \boldsymbol{r}_1 &= A\boldsymbol{x}_1 - \boldsymbol{b} = A(\boldsymbol{x}_0 + \alpha_0 \boldsymbol{p}_0) - \boldsymbol{b} \\ &= A(\boldsymbol{x}_0) - \boldsymbol{b} + \alpha_0 A\boldsymbol{p}_0. \end{aligned}$$

Despejando α_0 se tiene que $\alpha_0 = \dfrac{\boldsymbol{p}_0^T(\boldsymbol{r}_1 - \boldsymbol{r}_0)}{\boldsymbol{p}_0^T A \boldsymbol{p}_0}$. Pero por el lema 4.1, $\alpha_0 = -\dfrac{\boldsymbol{p}_0^T \boldsymbol{r}_0}{\boldsymbol{p}_0^T A \boldsymbol{p}_0}$. De los dos valores de α_0, se obtiene $0 = \boldsymbol{p}_0^T \boldsymbol{r}_1 = \boldsymbol{r}_1^T \boldsymbol{p}_0$. Ahora suponiendo que $\boldsymbol{r}_{k-1}^T \boldsymbol{p}_i = 0$, para todo $i = 0, 1, 2, \ldots, k-2$, se va a probar que se cumple también para k.

De resultados anteriores, $\boldsymbol{r}_k = \boldsymbol{r}_{k-1} + \alpha_{k-1} A \boldsymbol{p}_{k-1}$, por lo cual

$$\boldsymbol{p}_{k-1}^T \boldsymbol{r}_k = \boldsymbol{p}_{k-1}^T \boldsymbol{r}_{k-1} + \alpha_{k-1} \boldsymbol{p}_{k-1}^T A \boldsymbol{p}_{k-1} = 0 \quad (\alpha_{k-1} = -\frac{\boldsymbol{p}_{k-1}^T \boldsymbol{r}_{k-1}}{\boldsymbol{p}_{k-1}^T A \boldsymbol{p}_{k-1}}).$$

Mientras tanto, para los otros vectores $\boldsymbol{p}_i$, $i = 0, 1, 2, \ldots, k-2$ por ser A-conjugados y por la hipótesis de inducción, se cumple que

$$\boldsymbol{p}_i^T \boldsymbol{r}_k = \boldsymbol{p}_i^T \boldsymbol{r}_{k-1} + \alpha_{k-1} \boldsymbol{p}_i^T A \boldsymbol{p}_{k-1} = 0.$$

De esto se concluye que $\boldsymbol{r}_k \boldsymbol{p}_i = 0$, para $i = 0, 1, \ldots, k-1$, entonces la prueba esta completada.

■

A continuación se presenta un teorema que nos garantiza que el algoritmo 14 converge a lo más en n pasos.

Teorema 4.2. *Suponga que la k-ésima iteración generada por el método del gradiente conjugado (Algoritmo 14) no es la solución exacta de $A\boldsymbol{x} = \boldsymbol{b}$, las siguientes cuatro propiedades se mantienen:*

1. $\boldsymbol{r}_k^T \boldsymbol{r}_i = 0$ *para* $i = 0, \ldots, k-1$.
2. $span\{\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_k\} = span\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^k\boldsymbol{r}_0\}$.
3. $span\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\} = span\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^k\boldsymbol{r}_0\}$.
4. $\boldsymbol{p}_k^T A\boldsymbol{p}_i = 0$ *para* $i = 0, 1, \ldots, k-1$.

Por lo tanto la secuencia $\{\boldsymbol{x}_k\}$ *converge a* $\boldsymbol{x}_*$ *en a lo sumo* n *pasos.*

Demostración. La demostración se va a realizar por inducción. Para $n = 0$ Las expresiones 2 y 3 se cumplen claramente; mientras que la expresión 4 se cumple por construcción para $n = 1$ (Algoritmo 14.) ($\boldsymbol{p}_1^T A\boldsymbol{p}_0 = 0$).

Si se asume ahora que estas expresiones son verdaderas para $n = 1, 2, \ldots, k$, se va a mostrar que se cumple para $n = k + 1$.

Para probar 2, se determinará primero que el conjunto del lado izquierdo esta contenido en el conjunto de el lado derecho. Debido a la inducción de hipótesis, de 3 y 4 se tiene que

$$\boldsymbol{r}_k,\, \boldsymbol{p}_k \in span\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^k\boldsymbol{r}_0\} \;\rightarrow\; A\boldsymbol{p}_k \in span\{A\boldsymbol{r}_0, \ldots, A^{k+1}\boldsymbol{r}_0\}$$

y como

$$\begin{aligned}
\boldsymbol{r}_{k+1} &= A\boldsymbol{x}_{k+1} - A\boldsymbol{x}^* = A(\boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k) - A\boldsymbol{x}^* \\
&= A\boldsymbol{x}_k - A\boldsymbol{x}^* + A\alpha_k\boldsymbol{p}_k \\
&= \boldsymbol{r}_k + A\alpha_k\boldsymbol{p}_k \\
&= \boldsymbol{r}_0 + A\sum_{i=0}^{k}\alpha_i\boldsymbol{p}_i,
\end{aligned}$$

entonces, $\boldsymbol{r}_{k+1} \in span\{A\boldsymbol{r}_0, \ldots, A^{k+1}\boldsymbol{r}_0\}$. Por la inducción de hipótesis de 2, $(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_k) \in span\{A\boldsymbol{r}_0, \ldots, A^k\boldsymbol{r}_0\}$, de esto se concluye que

$$\boldsymbol{r}_{k+1} \in span\{A\boldsymbol{r}_0, \ldots, A^{k+1}\boldsymbol{r}_0\} \subset span\{A\boldsymbol{r}_0, \ldots, A^k\boldsymbol{r}_0\}.$$

Para probar que la inclusión del lado contrario se mantiene también, se usa la inducción de hipótesis de 3 para deducir que

$$A^{k+1}\boldsymbol{r}_0 = A(A^k\boldsymbol{r}_0) \in span\{A\boldsymbol{p}_0, A\boldsymbol{p}_1, \ldots, A\boldsymbol{p}_k\}.$$

De $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i + A\boldsymbol{p}_i\alpha_i$, se obtiene que $A\boldsymbol{p}_k = \dfrac{\boldsymbol{r}_{i+1} - \boldsymbol{r}_i}{\alpha_i}$ para $i = 0, 1, \ldots, k$. De esto se sigue que

$$A^{k+1}\boldsymbol{r}_0 \in span\{\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_{k+1}\}.$$

Combinando esta expresión con la inducción de hipótesis de 2 se encuentra que

$$span\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{k+1}\boldsymbol{r}_0\} \subset span\{\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_k, \boldsymbol{r}_{k+1}\}.$$

Por lo tanto, la relación 2 se cumple cuando k es reemplazado por $k+1$. Ahora se va a mostrar que 3 se cumple cuando k es reemplazado por $k+1$ por el siguiente argumento

$$\begin{aligned} span\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k, \boldsymbol{p}_{k+1}\} &= span\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k, \boldsymbol{r}_{k+1}\} \quad (\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k) \\ &= span\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^k\boldsymbol{r}_0, \boldsymbol{r}_{k+1}\} \quad \text{(hipótesis de inducción de 3)} \\ &= span\{\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_k, \boldsymbol{r}_{k+1}\} \quad \text{(por 2, demostrado)} \\ &= span\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^{k+1}\boldsymbol{r}_0\}. \quad \text{(por 2)} \end{aligned}$$

Ahora se prueba la condición 4 con k reemplazado por $k+1$. Como

$$\boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k \rightarrow \boldsymbol{p}_{k+1}^T = -\boldsymbol{r}_{k+1}^T + \beta\boldsymbol{p}_k^T,$$

entonces,

$$\boldsymbol{p}_{k+1}^T = -\boldsymbol{r}_{k+1}^T + \beta_{k+1}\boldsymbol{p}_k^T \;\rightarrow\; \boldsymbol{p}_{k+1}^T A\boldsymbol{p}_i = -\boldsymbol{r}_{k+1}^T A\boldsymbol{p}_i + \beta_{k+1}\boldsymbol{p}_k^T A\boldsymbol{p}_i.$$

Si $i = k$ al lado derecho de la ultima expresión es igual a cero. Para $i \leq k-1$, se puede notar primero que la inducción de hipótesis para 4 implica que las direcciones $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k$ son conjugadas, entonces se puede aplicar el teorema 4.1 para deducir que $\boldsymbol{r}_{k+1}^T\boldsymbol{p}_i = 0$, para $i = 0, 1, \ldots, k$.

Segundo, por 3 se encuentra que para $i = 0, 1, \ldots, k-1$

$$A\boldsymbol{p}_i \in Aspan\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^i\boldsymbol{r}_0\} = span\{A\boldsymbol{r}_0, A^2\boldsymbol{r}_0, \ldots, A^{i+1}\boldsymbol{r}_0\} \subset span\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots \boldsymbol{p}_{i+1}\}.$$

De lo anterior y de $\boldsymbol{r}_{k+1}^T\boldsymbol{p}_o = 0$ para $i = 0, 1, \ldots, k-1$ se deduce que $\boldsymbol{r}_{k+1}^T A\boldsymbol{p}_i = 0$ para $i = 0, 1, \ldots, k-1$. $(A\boldsymbol{p}_k = a_0\boldsymbol{p}_0 + a_1\boldsymbol{p}_1 + \ldots + a_{k+1}\boldsymbol{p}_{k+1} \rightarrow \boldsymbol{r}_{k+1}^T A\boldsymbol{p}_k = \boldsymbol{r}_{k+1}^T(a\boldsymbol{p}_0 + \ldots + a_{k+1}\boldsymbol{p}_{k+1}) = 0,$ donde $k = 0, 1, \ldots, k-1)$

Así, $\boldsymbol{p}_{k+1}^T A\boldsymbol{p}_i = 0$ y $\beta_{k+1}^T\boldsymbol{p}_k^T A\boldsymbol{p}_i$ para $i = 0, 1, \ldots, k-1$, y por tanto, $\boldsymbol{p}_{k+1}^T A\boldsymbol{p}_i = 0$, $i = 0, 1, \ldots, k$. Entonces se cumple 4, lo que implica que $\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k$ es un conjunto A-conjugado y por el lema 4.1 se dice que el algoritmo termina en a lo más en n iteraciones.

Finalmente se prueba 1 por un argumento no inductivo. Dado que este conjunto dirección es conjugado se tiene que del teorema 4.1 $\boldsymbol{r}_k^T \boldsymbol{p}_i = 0$, para todo $i = 0, \ldots, k-1$ y cualquier $k = 1, 2, \ldots, n-1$. Como $\boldsymbol{p}_i = -\boldsymbol{r}_i + \beta_i \boldsymbol{p}_{i-1}$, para $\boldsymbol{r}_i \in span\{\boldsymbol{p}_i, \boldsymbol{p}_{i-1}\}$, para $i = 1, \ldots, k-1$ se concluye que,

$$0 = \boldsymbol{r}_k^T \boldsymbol{p}_i = -\boldsymbol{r}_k^T \boldsymbol{r}_i + \beta_i \boldsymbol{r}_k^T \boldsymbol{p}_{i-1} = -\boldsymbol{r}_k^T \boldsymbol{r}_i.$$

Entonces $\boldsymbol{r}_k^T \boldsymbol{r}_i = 0$ para cada $i = 1, \ldots, k-1$.

■

Observación 4.2. Si $\boldsymbol{p}_0 \neq -\boldsymbol{r}_0$ no se mantiene el método del gradiente conjugado, dado que, los r_k son mutuamente ortogonales y dependen de $\boldsymbol{p}$, como se observa en la demostración de la condición 1.

4.2.1. Versión final del método del Gradiente Conjugado

De los anteriores resultados se puede encontrar, que el algoritmo 14, presenta varios productos matriz por vector. Para realizar en menor medida esta operación y ser mas eficiente se puede obtener un algoritmo mas eficiente, usando los teoremas 4.1 y 4.2. Para ello, en primer lugar se puede notar que

$$\alpha_k = -\frac{\boldsymbol{r}_k^T \boldsymbol{p}_k}{\boldsymbol{p}_k A \boldsymbol{p}_k^T} \quad \text{y} \quad \boldsymbol{p}_k = -\boldsymbol{r}_k + \beta_k \boldsymbol{p}_{k-1} \quad \text{entonces,}$$

$$\begin{aligned} \alpha_k &= -\frac{\boldsymbol{r}_k^T(-r_k + \beta_k \boldsymbol{p}_{k-1})}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \\ &= \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k - \beta_k \boldsymbol{r}_k^T \boldsymbol{p}_{k-1}}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \\ &= \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \qquad \text{(Teorema 4.1).} \end{aligned}$$

En segundo lugar se tiene que

$$\beta_k = \frac{\boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k} \quad \text{y} \quad \boldsymbol{r}_{k+1} = A\boldsymbol{x}_{k+1} - \boldsymbol{b} = A(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) - \boldsymbol{b} = \boldsymbol{r}_k + A\boldsymbol{p}_k \alpha_k.$$

A partir de esto, $\frac{\boldsymbol{r}_{k+1} - \boldsymbol{r}_k}{\alpha_k} = A\boldsymbol{p}_k$ y $\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$. Reemplazando en $A\boldsymbol{p}_k$ se obtiene que

$$A\boldsymbol{p}_k = \frac{\boldsymbol{r}_{k+1} - \boldsymbol{r}_k}{\boldsymbol{r}_k^T \boldsymbol{r}_k} \boldsymbol{p}_k^T A \boldsymbol{p}_k.$$

Nuevamente reemplazando ahora en β_k se deduce que

$$\begin{aligned}\beta_k &= \frac{\boldsymbol{r}_{k+1}^T(\boldsymbol{r}_{k+1}-\boldsymbol{r}_k)}{(\boldsymbol{r}_k^T\boldsymbol{r}_k)\boldsymbol{p}_k^TA\boldsymbol{p}_k}\boldsymbol{p}_k^TA\boldsymbol{p}_k \\ &= \frac{\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}-\boldsymbol{r}_{k+1}^T\boldsymbol{r}_k}{\boldsymbol{r}_k^T\boldsymbol{r}_k} \\ &= \frac{\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}}{\boldsymbol{r}_k\boldsymbol{r}_k} \qquad (\text{ Teorema 4.2, condicion 4}).\end{aligned}$$

El algoritmo 10, que se presenta a continuación es el algoritmo de la versión final del **CG**.

Algoritmo 10 G.C versión final

Entrada: : Matriz A, vector $\boldsymbol{b}$ y vector inicial $\boldsymbol{x}_0$

Salida: : vector x_k, solución del método

1: $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$
2: $\boldsymbol{p}_0 = -\boldsymbol{r}_0$
3: $k = 0$
4: **mientras** $\boldsymbol{r}_k \neq 0$ **hacer**
5: $\quad \boldsymbol{w}_k = A\boldsymbol{p}_k$
6: $\quad \alpha_k = \dfrac{\boldsymbol{r}_k^T\boldsymbol{r}_k}{\boldsymbol{p}_k^T\boldsymbol{w}_k}$
7: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k$
8: $\quad \boldsymbol{r}_{k+1} = A\boldsymbol{x}_{k+1} - \boldsymbol{b}$
9: $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \alpha_k\boldsymbol{w}_k$
10: $\quad \beta_{k+1} = \dfrac{\boldsymbol{r}_{k+1}^T\boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T\boldsymbol{r}_k}$
11: $\quad \boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
12: $\quad k = k + 1$
13: **fin mientras**
14: **devolver** vector $\boldsymbol{x}_k$

Ejemplo 4.2. Retomando el sistema $A\boldsymbol{x} = \boldsymbol{b}$ del ejemplo 4.1, si se resuelve con el algoritmo 10, con una tolerancia de 1E-12, se encuentra que el método converge en la iteración número dos,

$$x_2 = \begin{bmatrix} 0.444444444444444 \\ 0.111111111111111 \end{bmatrix}.$$

En la siguiente gráfica se muestra el comportamiento de las iteraciones, al encontrar el mínimo de la función

$$f(x,y) = x^2 + yx + \frac{5}{2}y^2 - x - y$$

que es equivalente a resolver el sistema $A\boldsymbol{x} = b$.

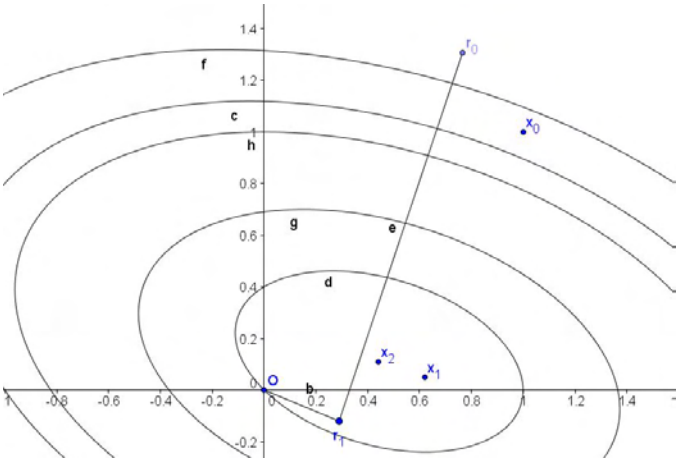


Figura 4.2: Minización usando el **CG**, ejemplo 2

Si se compara con el ejemplo 4.1 se puede ver que converge en número menor de iteraciones y cumple con la teoría tratada.

Ejemplo 4.3. Considere un sistema no simétrico $A\boldsymbol{x} = \boldsymbol{b}$ con

$$A = \begin{bmatrix} 2 & -1 \\ 2 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \text{y} \quad x_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Solucionando el sistema $A\boldsymbol{x} = b$ con el **CG** con una tolerancia de 1E-12, se encuentra que el método no converge como se observa en la figura 4.3, la cual no converge como se espera en la teoría a pesar de acerca al punto mínimo pero no va a coincidir debido a que la matriz no es simétrica, y por tanto el residuo no coincide con el gradiente de la función

$$f(x, y) = x^2 + \frac{yx}{2} + y^2 - x - y.$$

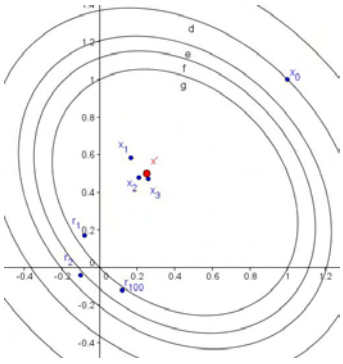


Figura 4.3: Minización usando el **CG**, ejemplo 3.

4.3. Gradiente Biconjugado (BCG)

Como se observo en la anterior sección, el método del gradiente conjugado se aplica específicamente para sistemas lineales S.D.P. Sin embargo para el caso de un sistema $Ax = b$ donde A no es simétrica, se podría también aplicar el método del gradiente conjugado resolviendo el sistema $A^TAx = A^Tb$ la cual cumple con las condiciones pedidas, pero la operación A^TA tiene un gran costo computacional. Siguiendo esta idea se recurre a un nuevo método, extendido a estos sistemas no simétricos llamado método del gradiente biconjugado **(BCG)**.

El **BCG** es similar **CG**, pero debido a la no simetría utiliza además de los vectores residuales $\boldsymbol{r}_k$ asociados al sistema $A\boldsymbol{x} = \boldsymbol{b}$, los vectores residuales $\tilde{\boldsymbol{r}_k}$ asociada al sistema dual $A^T\boldsymbol{x} = \boldsymbol{b}^*$, respectivamente. De igual manera que en el **CG** estos cumplen que

$$\tilde{\boldsymbol{r}_n} = \tilde{\boldsymbol{p}_n}(A)\tilde{\boldsymbol{r}_0} \ \in span\{\tilde{\boldsymbol{r}_0}, A^T\tilde{\boldsymbol{r}_0}, \ldots, (A^T)^n\tilde{\boldsymbol{r}_0}\},$$

donde $\boldsymbol{p}_n(\tilde{A^T})$ denota un polinomio de grado n. Ahora el conjunto $\{\boldsymbol{r}_n\}_{n=0}^{m=0}$ y $\{\tilde{\boldsymbol{r}_n}\}_{n=0}^{m}$ forman bases biortogonales, que se definen como

$$< \tilde{\boldsymbol{r}_m}, \boldsymbol{r}_n >= \begin{cases} 0 & si \quad m \neq n, \\ \delta_n \neq 0 & si \quad m = n. \end{cases}$$

De igual forma al utilizar dos conjuntos residuales, se construyen dos familias de vectores conjugados $\{\boldsymbol{p}_n\}_{n=0}^{m}$ y $\{\tilde{\boldsymbol{p}_n}\}_{n=0}^{m}$ asociados al sistema $A\boldsymbol{x} = b$ y al sistema $A^T\boldsymbol{x}^* = \boldsymbol{b}$ respectivamente, las cuales forman bases biconjugadas, es decir,

$$< \tilde{\boldsymbol{p}_m}, A\boldsymbol{p}_n >= \begin{cases} 0 & si \quad m \neq n, \\ \delta_n \neq 0 & si \quad m = n. \end{cases}$$

A partir de lo anterior, se requiere un vector inicial $\boldsymbol{x}_0$, para construir el residuo inicial $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$ y el vector $\tilde{\boldsymbol{r}_0}$, los cuales deben cumplir que $< \boldsymbol{r_0}, \tilde{\boldsymbol{r_0}} >\neq 0$, por lo que una manera conveniente de hacer la escogencia es $\tilde{\boldsymbol{r}}_0 = \boldsymbol{r}_0$.

A continuación se presenta el algoritmo 11, del método **BCG**. Si se observa en su algoritmo las lineas 1 2, 7 9 utilizan producto matriz por vector y en la linea 10 el producto matriz transpuesta por vector. De esta manera si A es simétrica el método coincide con el **CG**, pero con la diferencia de que se realiza más operaciones las cuales serían innecesarias.

Algoritmo 11 BCG

Entrada: : Matriz A, vector $\boldsymbol{b}$ vector inicial $\boldsymbol{x}_0$ y vector $\tilde{\boldsymbol{r}}_0$ ($< \tilde{\boldsymbol{r}}_0, \boldsymbol{r}_0 > \neq 0$)

Salida: : vector $\boldsymbol{x}_k$, solución del método

1: $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$
2: $\boldsymbol{p}_0 = -\boldsymbol{r}_0$
3: $\tilde{\boldsymbol{p}}_0 = -\tilde{\boldsymbol{r}}_0$
4: $k = 0$
5: **mientras** $\boldsymbol{r}_k \neq 0$ **hacer**
6: $\quad \alpha_k = \dfrac{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_k >}{< A\boldsymbol{p}_k, \tilde{\boldsymbol{p}}_k >}$
7: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
8: $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k$
9: $\quad \tilde{\boldsymbol{r}}_{k+1} = \tilde{\boldsymbol{r}}_k + \alpha_k A^T \tilde{\boldsymbol{p}}_k$
10: $\quad \beta_k = \dfrac{< \boldsymbol{r}_{k+1}, \tilde{\boldsymbol{r}}_{k+1} >}{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_k >}$
11: $\quad \boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$
12: $\quad \tilde{\boldsymbol{p}}_{k+1} = -\tilde{\boldsymbol{r}}_{k+1} + \beta_k \tilde{\boldsymbol{p}}_k$
13: $\quad k = k + 1$
14: **fin mientras**
15: **devolver** vector $\boldsymbol{x}_k$

Ejemplo 4.4. Al aplicar el algoritmo 11 del **BCG**, al sistema $A\boldsymbol{x} = \boldsymbol{b}$ del ejemplo 4.3, se obtuvo la solución y mediante la forma cuadrática asociada a la matriz $A^T A$ y al vector $A^T \boldsymbol{b}$ que es

$$f(x, y) = x^2 + \frac{yx}{2} + y^2 - x - y,$$

coincide la solución con el mínimo que es $\boldsymbol{x} = \begin{bmatrix} 0.75 \\ -0.25 \end{bmatrix}$. En la figura 4.4, se observa como aplicando **BCG**, se llega al mínimo de la función $f(x, y)$ en 2 iteraciones, donde $\boldsymbol{x}_2$ es el mínimo. Además se observa el comportamiento biortogonal de los vectores residuales.

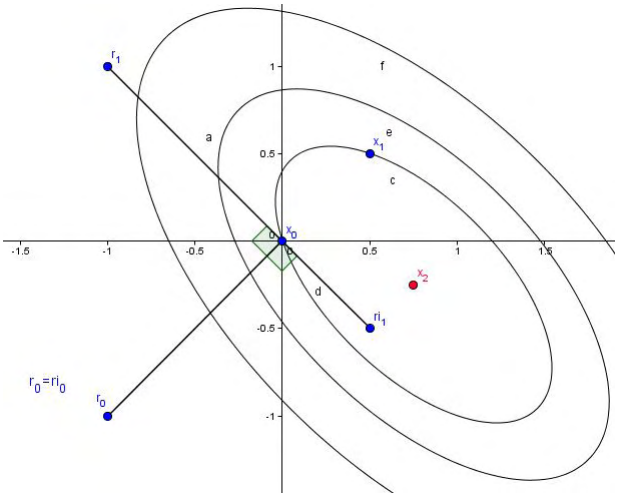


Figura 4.4: Minización de $f(x, y)$ usando **BCG**, ejemplo 4.

Observación 4.3. Si un sistema lineal es **S.D.P** el método **CG**, coincide con el método **BCG** por tanto se cumple el lema 4.1; en caso contrario no se garantiza que se cumpla las consecuencias de dicho lema.

4.4. Gradiente Conjugado Cuadrado (CGS)

El algoritmo del Gradiente Conjugado Cuadrado (**CGS**) fue desarrollado por Sonneveld en 1984 [28], principalmente para evitar el uso de la transpuesta del **BCG** y ganar una mejor convergencia con aproximadamente el mismo costo computacional. La idea principal se basa en la siguiente observación. En el algoritmo **BCG**, el vector residual que se observar en el paso 9 del algoritmo 11, k puede ser expresado como

$$\boldsymbol{r}_k = \phi_k(A)\boldsymbol{r}_0, \qquad \text{(Teorema 4.2)}$$

donde ϕ es un cierto polinomio de grado k, con indeterminada A, que satisface la restricción $\phi_k(0) = \mathbf{1}$ (Dado que $\boldsymbol{r}_k = \boldsymbol{r}_0 + A\sum_{i=0}^{j-1} \alpha_i p_i$). Además se define la dirección conjugada polinomial $\pi_k(t)$ que esta dada por

$$\boldsymbol{p}_k = \pi_k(A)\boldsymbol{r}_0, \qquad \text{(Teorema 4.2)}$$

donde π_k es un polinomio de grado k. Del mismo algoritmo 11, observe que las direcciones $\tilde{\boldsymbol{r}}_k$ y $\tilde{\boldsymbol{p}}_k$ son definidas a través de las mismas recurrencias de $\boldsymbol{r}_k$ y $\boldsymbol{p}_k$ con la diferencia de que A es reemplazado por A^T, y como resultado

$$\tilde{\boldsymbol{r}}_k = \phi_k(A^T)\tilde{\boldsymbol{r}}_0, \qquad \tilde{\boldsymbol{p}}_k = \pi_k(A^T)\tilde{\boldsymbol{r}}_0.$$

Por tanto, el escalar α_k en el **BCG** se puede expresar como

$$\begin{aligned}\alpha_k &= \frac{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_k >}{< A\boldsymbol{p}_k, \tilde{\boldsymbol{p}}_k >} \\ &= \frac{< \phi_k(A)\boldsymbol{r}_0, \phi_k(A^T)\tilde{\boldsymbol{r}}_0}{< A\phi_{\boldsymbol{k}}(A)r_0, \pi_k(A^T)\tilde{\boldsymbol{r}}_0 >} \\ &= \frac{< \phi_k^2(A)\boldsymbol{r}_0, \tilde{\boldsymbol{r}}_0 >}{< A\pi_k^2(A)\boldsymbol{r}_0, \tilde{\boldsymbol{r}}_0 >}.\end{aligned}$$

La idea del método es buscar un algoritmo en el cual se pueda expresar la secuencia de iteraciones residuales $\boldsymbol{r}'_k$ que satisfagan

$$\boldsymbol{r}'_k = \phi_k^2(A)\boldsymbol{r}_0.$$

La derivación del **CGS** se basa realizando cálculos algebraicos. Para establecer las recurrencias requeridas para los polinomios cuadrados, se inicia con la recurrencia que definen ϕ_k y π_k que son

$$\phi_{k+1}(t) = \phi_k + \alpha_k t\pi_k(t) \quad \text{y} \quad \pi_{k+1}(t) = -\phi_{k+1}(t) + \beta_k\pi_k(t).$$

Estos resultados se obtienen ya que para $\phi_{k+1}(t)$

$$\begin{aligned}\boldsymbol{r}_{k+1} = \phi_{k+1}(A)\boldsymbol{r}_0 &= A(\boldsymbol{x}_{k+1}) - \boldsymbol{b}) = A(\boldsymbol{x}_k + \alpha_k\boldsymbol{p}_k) \\ &= A(\boldsymbol{x}_k) - \boldsymbol{b} + \alpha A\boldsymbol{p}_k \\ &= \phi_k(A)\boldsymbol{r}_0 + \alpha_k A\boldsymbol{\pi}_k(A)r_0 \\ &= (\phi_k(A) + \alpha_k A\pi_k(A))\boldsymbol{r}_0\end{aligned}$$

y para $\pi_{k+1}(A)$

$$\begin{aligned}p_{k+1} = \pi_{k+1}(A)r_0 &= -\boldsymbol{r}_{k+1} + \boldsymbol{\beta}_k p_k \\ &= -\phi_{k+1}(A)\boldsymbol{r}_0 + \beta_k\pi_k(A)\boldsymbol{r}_0 \\ &= (-\phi_{k+1}(A) + \beta_k\pi_k(A))\boldsymbol{r}_0.\end{aligned}$$

Por igualación de los coeficientes de r_0 en ambos casos se obtiene dichos resultados. Si las relaciones son cuadradas se obtiene

$$\phi_{k+1}^2(t) = \phi_k^2 + 2\alpha_k t\pi_k(t)\phi_k(t) + \alpha_k^2 t^2\pi_k^2(t),$$

$$\pi_{k+1}^2 = \phi_{k+1}^2(t) - 2\beta_k\phi_{k+1}(t)\phi_k(t) + \beta_k^2\pi_k(t)^2.$$

Si no hubiese el cruce de términos $\pi_k(t)\phi_k(t)$ y $\phi_{k+1}(t)\pi_k(t)$ en el lado derecho de estas ecuaciones no formarían un sistema de recurrencia redefinible. La solución es introducir uno de estos dos términos cruzados, este es, $\phi_{k+1}(t)\pi_k(t)$, como un tercer miembro de la recurrencia. Para el otro termino $\pi_k(t)\phi_l(t)$ se puede explotar la relación

$$\phi_k(t)\pi_k(t) = \phi_k(t)(-\phi_k(t) + \beta_{k-1}\pi_{k-1}(t)) = -\phi_k(t)^2 + \beta_{k-1}\phi_k(t)\pi_{k-1}(t).$$

Para colocar estas relaciones juntas, las siguientes relaciones de recurrencia pueden ser derivadas, en la cual la variable t, se omite, donde no haya ambiguedades

$$\phi_{k+1}^2 = \phi_k + \alpha_k t(-2\phi_k^2 + 2\beta_{k-1}\phi\pi_{k-1} + \alpha_k t\pi_k^2),$$

$$\phi_k + \pi_k = -\phi_k^2 + \beta_{k-1}\phi_k\pi_{k-1} + \alpha_k t\pi_k^2,$$

$$\pi_{k+1}^2 = \phi_{k+1}^2 - 2\beta_k\phi_{k+1}\pi_k + \beta_k^2\pi_2^2.$$

Estas recurrencias son las bases del algoritmo. Si se define

$$\boldsymbol{r}_k = \phi_k^2(A)\boldsymbol{r}_0,$$

$$\boldsymbol{p}_k = \pi_k^2(A)\boldsymbol{r}_0,$$

$$\boldsymbol{q}_k = \phi_{k+1}(A)\pi_k(A)\boldsymbol{r}_0,$$

las relaciones para los polinomios se convierten en

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \alpha_k A(-2\boldsymbol{r}_k + 2\beta_{k-1}\boldsymbol{q}_{k-1} + \alpha_k A\boldsymbol{p}_k),$$

$$\boldsymbol{q}_k = -\boldsymbol{r}_k + \beta_{k-1}\boldsymbol{q}_{k-1} + \alpha_k A\boldsymbol{p}_k,$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} - 2\beta_k\boldsymbol{q}_k + \beta_k^2\boldsymbol{p}_k.$$

Se Define un vector auxiliar

$$\boldsymbol{d}_k = -2\boldsymbol{r}_k + 2\beta_{k-1}\boldsymbol{q}_{k-1} + \alpha_k A\boldsymbol{p}_k.$$

Con este vector se obtiene la siguiente secuencia de operaciones para calcular la solución aproximada, iniciando con $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$, $\boldsymbol{p}_0 = \boldsymbol{r}_0$, $\boldsymbol{q}_0 = 0$, y $\beta_0 = 0$.

- $\alpha_k = \dfrac{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_0 >}{< A\boldsymbol{p}_k, \tilde{\boldsymbol{r}}_0 >}$
- $\boldsymbol{d}_k = -2\boldsymbol{r}_k + 2\beta_{k-1}\boldsymbol{q}_{k-1} + \alpha_k A\boldsymbol{p}_k$
- $\boldsymbol{q}_k = -r_k + \beta_{k-1}\boldsymbol{q}_{k-1} + \alpha_k A\boldsymbol{p}_k$
- $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k\boldsymbol{d}_k$
- $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \alpha_k A\boldsymbol{d}_k$
- $\beta_k = \dfrac{< \boldsymbol{r}_{k+1}, \tilde{\boldsymbol{r}}_0 >}{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_0 >}$
- $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} - \beta_k(2\boldsymbol{q}_k - \beta_k\boldsymbol{p}_k)$.

Una ligera simplificación para el algoritmo puede ser hecha con un vector auxiliar $\boldsymbol{u}_k = -\boldsymbol{r}_k + \beta_{k-1}\boldsymbol{q}_{k-1}$. Esta definición nos da las relaciones

$$\boldsymbol{d}_k = \boldsymbol{u}_k + \boldsymbol{q}_k$$

$$\boldsymbol{q}_k = \boldsymbol{u}_k + \alpha_k A\boldsymbol{p}_k,$$

$$\boldsymbol{p}_{k+1} = -\boldsymbol{u}_{k+1} - \beta_k(\boldsymbol{q}_k - \beta_k\boldsymbol{p}_k).$$

Como resultado de la modificación $\boldsymbol{d}_k$ no es necesario. El algoritmo 12, representa el método **CGS**.

Algoritmo 12 CGS

Entrada: : Matriz A, vector $\boldsymbol{b}$, vector inicial $\boldsymbol{x}_0$ y vector arbitrario $\tilde{\boldsymbol{r}}_0$

Salida: : vector $\boldsymbol{x}_k$, solución del método

1: $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$
2: $\boldsymbol{u}_0 = -\boldsymbol{r}_0$
3: $\boldsymbol{p}_0 = -\boldsymbol{u}_0$
4: $k = 0$
5: **mientras** $\boldsymbol{r}_k \neq 0$ **hacer**
6: $\quad \alpha_k = \dfrac{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_0 >}{< A\boldsymbol{p}_k, \tilde{\boldsymbol{r}}_0 >}$
7: $\quad \boldsymbol{q}_k = \boldsymbol{u}_k + \alpha_k A\boldsymbol{p}_k$
8: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k(\boldsymbol{u}_k + \boldsymbol{q}_k)$
9: $\quad \boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \alpha_k A(\boldsymbol{u}_k + \boldsymbol{q}_k)$
10: $\quad \beta_k = \dfrac{< \boldsymbol{r}_{k+1}, \tilde{\boldsymbol{r}}_0 >}{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_0 >}$
11: $\quad \boldsymbol{u}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_k\boldsymbol{q}_k$
12: $\quad k = k + 1$
13: **fin mientras**
14: **devolver** vector $\boldsymbol{x}_k$

Observación 4.4. En el algoritmo 12 del **CGS** no hay productos matriz por vector con la transpuesta A. En vez de ello dos productos matriz por vector con la matriz A son ahora realizados en cada paso. En general, uno debería esperar que el resultado algoritmo converga dos veces tan rápido como **BCG**. Por lo tanto, lo que esencialmente se ha logrado es reemplazar el producto matriz transpuesta por vector.

El algoritmo del gradiente conjugado cuadrado trabajo bastante bien en muchos casos. Sin embargo, una dificultad es que como los polinomios son cuadrados, los errores de redondeo tiende a ser mas peligrosos que en el algoritmo estándar **BCG**. En particular, muy altas variaciones de los vectores residuales por lo que pueden ser muy inexactos.

4.5. Gradiente Biconjugado Estabilizado (BICGSTAB)

El método del Gradiente Biconjugado Estabilizado fue publicado por Van der Vorst en 1992 [31]. Como el algoritmo **CGS** es basado en expresar los residuos de forma cuadrática, lo cual en casos de irregular convergencia, esto podría conducir a una considerable acumulación de errores de redondeo que pueden afectar la convergencia del método. El (**BICGSTAB**) es una variación del **CGS** el cual fue desarrollado para remediar esta dificultad. En vez de buscar un método que proporcione un vector residual de la forma $\boldsymbol{r}'_k = \phi_k^2(A)\boldsymbol{r}_0$, donde

$$\phi_{k+1}(t) = \phi_k(t) + \alpha_k t\pi_k(t),$$

en el **BICGSTAB** produce iteraciones cuyos vectores residuales son de la forma

$$\boldsymbol{r}'_k = \psi_k(A)\phi_k(A)\boldsymbol{r}_0,$$

donde $\phi_k(t)$ es el polinomio residual asociado con el **BCG** algoritmo y $\psi_k(t)$ es un nuevo polinomio el cual es definido recursivamente en cada paso con la meta de "establizar" o "suavizar" el comportamiento de la convergencia del algoritmo original. Específicamente, $\psi_k(t)$ es definido por la simple recurrencia

$$\psi_{k+1}(t) = (1 - w_k t)\psi_k(t),$$

en el cual el escalar w_k va a ser determinado por la derivación de las relaciones de recurrencia que son similares a las del **CGS**. De $\psi_k(t)$ y de $\phi_k(t)$ se tiene una relación para el polinomio residual $\psi_{k+1}\phi_{k+1}$, la cual es

$$\begin{aligned}\psi_{k+1}\phi_{k+1} &= (1 - w_k t)\psi_k(t)\phi_{k+1} \\ &= (1 - w_k t)(\psi_k\phi_k + \alpha_k t\psi_k\pi_k),\end{aligned}$$

que proporciona una relación de recurrencia para encontrar los $\psi_k\pi_k$. Para esto se escribe

$$\begin{aligned}\psi_k\pi_k &= \psi_k(-\phi_k + \beta_{k-1}\pi_{k-1}) \\ &= -\psi_k\phi_k + \beta_{k-1}(1 - w_{k-1}t)\psi_{k-1}\pi_{k-1}.\end{aligned}$$

Se define

$$\boldsymbol{r}_k = \psi_k(A)\phi_k(A)\boldsymbol{r}_0,$$

$$\boldsymbol{p}_k = \psi_k(A)\pi_k(A)\boldsymbol{r}_0.$$

Con estas formulas, estos vectores pueden ser redefinidos desde una doble recurrencia, siempre que los escalares α_k y β_k fueran calculables. Esta recurrencia es

$$\boldsymbol{r}_{k+1} = (I - w_k A)(\boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k),$$

$$\boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_k(I - w_k A)\boldsymbol{p}_k.$$

Ahora falta calcular los escalares necesarios en la recurrencia. De acuerdo al algoritmo 11 del **BCG**, $\beta_k = \dfrac{\rho_{k+1}}{\rho_k}$ con

$$\begin{aligned}\rho_k &=< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_k > \\ &=< \phi_k(A)\boldsymbol{r}_0, \phi_k(A^T)\tilde{\boldsymbol{r}}_0 > .\end{aligned}$$

Infortunadamente, ρ_k no es calculable de estas formulas debido a que ninguno de los vectores $\phi_k(A)\boldsymbol{r}_0$, $\phi_k(A^T)\tilde{\boldsymbol{r}}_0$ o $\phi_k^2(A)\boldsymbol{r}_0$ no se ha encontrado ninguna relación. Sin embargo, ρ_k puede ser relacionado al escalar

$$\tilde{\rho}_k =< \phi_k(A)\boldsymbol{r}_0, \psi_k(A^T)\tilde{\boldsymbol{r}}_0 >$$

que es calculable con

$$\tilde{\rho}_k =< \phi_k(A)\boldsymbol{r}_0, \psi_k(A^T)\tilde{\boldsymbol{r}}_0 >=< \psi_k(A)\phi_k(A)\boldsymbol{r}_0, \tilde{\boldsymbol{r}}_0 >=< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_0 > .$$

Para relacionar los dos escalares ρ_k y $\tilde{\rho}_k$, se expande $\psi_j(A^T)\tilde{r}_0$ en bases de potencias para obtener

$$\tilde{\rho}_k =< \phi_k(A)\boldsymbol{r}_0, \eta_1^{(k)}(A^T)^k\tilde{\boldsymbol{r}}_0 + \eta_2^{(k-1)}(A^T)^{k-1}\tilde{\boldsymbol{r}}_0 + \ldots >$$

Por lo ortogonalidad de $\boldsymbol{r}_k$ con respecto a $\tilde{\boldsymbol{r}}_k$, entonces, sólo la primera potencia es importante en la expresión al lado derecho del producto interno anterior en particular si $\gamma_1^{(k)}$ es el coeficiente que acompaña al término de mayor grado del polinomio $\phi_k(t)$, entonces

$$\tilde{\rho}_k =< \phi_k(A)\boldsymbol{r}_0, \frac{\eta_1^{(k)}}{\gamma_1^{(k)}}\phi_k(A^T)\boldsymbol{r}_0 >= \frac{\eta_1^{(k)}}{\gamma_k^{(k)}}\psi_k.$$

Cuando se examina las relaciones de secuenciales para ϕ_{k+1} y ψ_{k+1}, los coeficientes lideres para estos polinomios satisfacen las relaciones

$$\eta_1^{k+1} = -w_k\eta_k^{(k)}, \qquad \gamma_1^{(k+1)} = -\alpha_k\gamma^{(k)}$$

y como resultado

$$\frac{\eta_1^{(k)}}{\gamma_1^{(k)}} = \frac{\tilde{\rho}_k}{\rho_k},$$

lo que implica que

$$\begin{aligned}\tilde{\rho}_{k+1} &= \frac{\eta_1^{(k+1)}}{\rho_1^{(k+1)}}\rho_{k+1} \\ &= \frac{-w_k\eta_1^{(k)}}{-\alpha_k\gamma_1^{(k)}}\eta_{k+1} \\ &= \frac{w_k}{\alpha_k}\frac{\tilde{\rho}_k}{\rho_k}\rho_{k+1}.\end{aligned}$$

Por lo tanto,

$$\frac{\tilde{\rho}_k}{\tilde{p}_k} = \frac{w_k}{\alpha_k}\frac{\rho_{k+1}}{\rho_k}.$$

Lo que nos lleva a la siguiente relación para β_k

$$\beta_k = \frac{\tilde{\rho}_{k+1}}{\tilde{\rho}_k} \times \frac{\alpha_k}{w_k}.$$

Similarmente una simple formula de recurrencia para α_k puede ser derivada. Por definición

$$\alpha_k = \frac{< \phi_k(A)\boldsymbol{r}_0, \phi_k(A^T)\tilde{\boldsymbol{r}}_0 >}{< A\pi_k(A)\boldsymbol{r}_0, \pi_k(A^T)\tilde{\boldsymbol{r}}_0 >}$$

y como en los previos casos, los polinomios en el lado derecho de los productos internos, en el numerador y denominador pueden ser reemplazados por su termino líder. Sin embargo en este caso los coeficientes lideres de $\phi_j(A^T)\tilde{\boldsymbol{r}}_0$ y $\pi_j(A^T)\tilde{\boldsymbol{r}}_0$ es el inverso aditivo del otro (dado que $\tilde{\boldsymbol{p}}_{j+1}$ depende de $-\tilde{r}_{j+1}$ y de $\tilde{\boldsymbol{p}}_j$) y por lo tanto

$$\begin{aligned}
\alpha_k &= -\frac{< \phi_k(A)\boldsymbol{r}_0, \phi_k(A^T)\tilde{\boldsymbol{r}}_0 >}{< A\pi_k(A)\boldsymbol{r}_0, \phi_k(A^T)\tilde{\boldsymbol{r}}_0 >} \\
&= -\frac{< \phi_k(A)\boldsymbol{r}_0, \psi_k(A^T)\tilde{\boldsymbol{r}}_0 >}{< A\pi_k(A)\boldsymbol{r}_0, \psi_k(A^T)\tilde{\boldsymbol{r}}_0} \\
&= -\frac{< \psi_k(A)\phi_k(A)\boldsymbol{r}_0, \tilde{\boldsymbol{r}}_0 >}{< A\psi_k(A)\pi_k(A)\boldsymbol{r}_0, \tilde{\boldsymbol{r}}_0 >}.
\end{aligned}$$

Desde que $\boldsymbol{p}_k = \psi_k(A)\pi_k(A)\boldsymbol{r}_0$, esto nos lleva a

$$\alpha_k = \frac{< \tilde{\boldsymbol{\rho}}_k, \tilde{\boldsymbol{r}}_0 >}{< A\boldsymbol{p}_k, \tilde{\boldsymbol{r}}_0 >}.$$

Lo siguiente, es definir un parámetro adicional que es w_k , para lograr un paso de descenso más empinado en la dirección obtenida antes, multiplicando el vector residual por $(I - w_k A)$ en

$$\boldsymbol{r}_{k+1} = (I - w_k A)(\boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k). \tag{4.5.1}$$

En otras palabras, w_k es escogido para minimizar la norma 2 del vector $(I - w_k A)\psi_k(A)\phi_{k+1}(A)\boldsymbol{r}_0$. La ecuación (4.5.1) puede ser reescrita como

$$\boldsymbol{r}_{k+1} = (I - w_k A)\boldsymbol{s}_k,$$

donde $\boldsymbol{s}_k = \boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k$. Entonces el valor optimo para w_k esta dado por

$$w_k = \frac{< A\boldsymbol{s}_k, \boldsymbol{s}_k >}{< A\boldsymbol{s}_k, A\boldsymbol{s}_k >}.$$

Lo anterior se logra igualando a cero $\boldsymbol{r}_{k+1}$, multiplicando por $\boldsymbol{s}_k^T A^T$ y despejando w_k. Finalmente se necesita una formula para actualizar la solución aproximada de x_{k+1} y x_k. La ecuación (4.5.1) puede ser reescrita como

$$\begin{aligned}\boldsymbol{r}_{k+1} &= (I - w_k A)\boldsymbol{s}_k \\ &= \boldsymbol{s}_k - w_k A\boldsymbol{s}_k \\ &= \boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k - w_k A\boldsymbol{s}_k,\end{aligned}$$

la cual implica que

$$A\boldsymbol{x}_{k+1} - \boldsymbol{b} = A\boldsymbol{x}_k - \boldsymbol{b} + \alpha_k A\boldsymbol{p}_k - w_k A\boldsymbol{s}_k$$

entonces,

$$A\boldsymbol{x}_{k+1} = A(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k - w_k \boldsymbol{s}_k)$$

y dado que A es invertible se concluye que

$$\boldsymbol{x}_{k+1} = x_k + \alpha_k \boldsymbol{p}_k - w_k \boldsymbol{s}_k$$

Luego de organizar todas estas relaciones, se obtiene la forma final del algoritmo, que se muestra a continuación.

Algoritmo 13 BICGSTAB

Entrada: : Matriz A, vector $\boldsymbol{b}$, vector inicial $\boldsymbol{x}_0$ y vector arbitrario $\tilde{\boldsymbol{r}}_0$

Salida: : vector $\boldsymbol{x}_k$, solución del método

1: $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$
2: $\boldsymbol{p}_0 = -\boldsymbol{r}_0$
3: **mientras** $\boldsymbol{r}_k \neq 0$ **hacer**
4: $\quad \alpha_k = -\dfrac{< r_k, \tilde{\boldsymbol{r}}_0 >}{< A\boldsymbol{p}_k, \tilde{\boldsymbol{r}}_0 >}$
5: $\quad \boldsymbol{s}_k = \boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k$
6: $\quad w_k = \dfrac{< A\boldsymbol{s}_k, \boldsymbol{s}_k >}{< A\boldsymbol{s}_k, A\boldsymbol{s}_k >}$
7: $\quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k - w_k \boldsymbol{s}_k$
8: $\quad \boldsymbol{r}_{k+1} = \boldsymbol{s}_k - w_j A\boldsymbol{s}_k$
9: $\quad \beta_k = \dfrac{< \boldsymbol{r}_{k+1}, \tilde{\boldsymbol{r}}_0 >}{< \boldsymbol{r}_k, \tilde{\boldsymbol{r}}_0 >} \times \dfrac{\alpha_k}{w_k}$
10: $\quad \boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_k(\boldsymbol{p}_k - w_j A\boldsymbol{p}_k)$
11: **fin mientras**
12: **devolver** vector $\boldsymbol{x}_k$

4.6. Precondicionamiento en los métodos del gradiente

Sea M un precondicionador del sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, donde A es S.D.P. El precondicionador M es una matriz la cual se aproxima a A en algún sentido sin definir; esto es, asumir que M es también S.D.P, entonces se puede precondicionar el sistema de las tres maneras mencionadas, como se explico en el capítulo 3. Sin embargo los dos primeros precondicionadores no se asegura que sea S.D.P, debido a que $M^{-1}A$ o AM^{-1} puede o no ser simétricas.

Como se supone que el precondicionador M es S.D.P, por el teorema 3.6, implica que tenga una factorización de Cholesky, es decir, M se lo puede factorizar como

$$M = LL^T,$$

donde L es una matriz no singular triangular inferior. Entonces al aplicar el precondicionador dividido, la matriz del sistema precondicionado $L^{-1}AL^{-T}$ es S.D.P. En efecto, para cualquier vector $\boldsymbol{x} \neq \in \mathbb{R}^n$

$$\boldsymbol{x}^T L^{-1} A L^{-T} \boldsymbol{x} = (L^{-T}\boldsymbol{x})^T A (L^{-T}\boldsymbol{x}) > 0,$$

dado que A es S.D.P. Como aplicar el precondicionamiento divido requiere un costo computacional, ya que se necesita encontrar la factorización de Cholesky de $M = LL^T$, encontrar la inversa de L y L^T, realizar el producto $L^{-1}AL^{-T}$ y resolver el sistema $\boldsymbol{u} = L^{-1}\boldsymbol{x}$.

A continuación se muestra una manera de implementar el **CG** precondicionado y el **BCG precondicionado**, sin realizar explicitamente dicho precondiciomaento, es decir se aplica el precondicionador en el algoritmo del método sin modificar el sistema original.

4.6.1. Gradiente Conjugado precondicionado (CG pre)

Si se aplica el método **CG** al sistema precondicionado $\tilde{A}\boldsymbol{u} = \tilde{\boldsymbol{b}}$, donde $\tilde{A} = L^{-1}AL^{-T}$, $\boldsymbol{u} = L^T\boldsymbol{x}$ y $\tilde{\boldsymbol{b}} = L^{-1}\boldsymbol{b}$, los pasos de dicho método, sin contar los pasos iniciales nos queda:

- $\alpha_k = \dfrac{\tilde{\boldsymbol{r}}_k^T \tilde{\boldsymbol{r}}_k}{\tilde{\boldsymbol{p}}_k^T A \tilde{\boldsymbol{p}}_k}$
- $\boldsymbol{u}_{k+1} = \boldsymbol{u}_k + \alpha_k \tilde{\boldsymbol{p}}_k$
- $\tilde{\boldsymbol{r}}_{k+1} = \tilde{\boldsymbol{r}}_k + \tilde{A}\alpha_k \tilde{\boldsymbol{p}}_k$
- $\beta_k = \dfrac{\tilde{\boldsymbol{r}}_{k+1}^T \tilde{\boldsymbol{r}}_{k+1}}{\tilde{\boldsymbol{r}}_k^T \tilde{\boldsymbol{r}}_k}$
- $\tilde{\boldsymbol{p}}_{k+1} = -\tilde{\boldsymbol{r}}_{k+1} + \beta_k \tilde{\boldsymbol{p}}_k.$

Como el residual

$$\tilde{\boldsymbol{r}} = \tilde{A}\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{b}} = L^{-1}(A\boldsymbol{x} - \boldsymbol{b}) = L^{-1}\boldsymbol{r},$$

se lo puede expresar los anteriores pasos en términos del sistema lineal original y entonces nos queda:

- $\alpha_k = \dfrac{\boldsymbol{r}_k^T L^{-T} L^{-1} \boldsymbol{r}_k}{(L^{-T}\tilde{\boldsymbol{p}}_k)^T A (L^{-T}\tilde{\boldsymbol{p}}_k)}$
- $L^T \boldsymbol{x}_{k+1} = L^T \boldsymbol{x}_k + \alpha_k \tilde{\boldsymbol{p}}_k$
- $L^{-1}\boldsymbol{r}_{k+1} = L^{-1}\boldsymbol{r}_k + L^{-1} A \alpha_k (L^{-T}\tilde{\boldsymbol{p}}_k)$
- $\beta_k = \dfrac{\boldsymbol{r}_{k+1}^T L^{-T} L^{-1} \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T L^{-T} L^{-1} \boldsymbol{r}_k}$
- $\tilde{\boldsymbol{p}}_{k+1} = -L^{-1}\boldsymbol{r}_{k+1} + \beta_k \tilde{\boldsymbol{p}}_k$

Se denota por $\boldsymbol{z_k} = L^{-T}L^{-1}\boldsymbol{r_k} = M^{-1}\boldsymbol{r_k}$ y $\boldsymbol{p_k} = L^{-T}\tilde{\boldsymbol{p}}_k$. Despejando $\boldsymbol{x_k}$ y multiplicando por la derecha a $\tilde{\boldsymbol{p}}_k$ y a $L^{-1}\boldsymbol{r_{k+1}}$, se puede aplicar el método **CG** para encontrar la solución del sistema original, cuyos pasos son los siguientes.

- $\alpha_k = \dfrac{\boldsymbol{r}_k^T \boldsymbol{z}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$
- $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
- $\boldsymbol{z}_{k+1} = M^{-1}(\boldsymbol{r}_k + A\alpha_k (L^{-T}\tilde{\boldsymbol{p}}_k)) = M^{-1}\boldsymbol{r}_{k+1}$
- $\beta_k = \dfrac{\boldsymbol{r}_{k+1}^T \boldsymbol{z}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{z}_k}$
- $\boldsymbol{p}_{k+1} = -\boldsymbol{z}_{k+1} + \beta_k \boldsymbol{p}_k$.

En el algoritmo 14, se muestra el método del gradiente conjugado precondicionado (**CG pre**), la cual mantiene una estructura similar al **CG**.

Algoritmo 14 CG pre

Entrada: : matriz A, precondicionador M, vector $\boldsymbol{b}$ y vector inicial $\boldsymbol{x}_0$

Salida: : vector $\boldsymbol{x}_k$, solución del método

1: $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$
2: $\boldsymbol{z}_0 = M^{-1}\boldsymbol{r}_0$
3: $\boldsymbol{p}_0 = -\boldsymbol{z}_0$
4: **mientras** $\boldsymbol{r}_k \neq 0$ **hacer**
5: $\alpha_k = \dfrac{< \boldsymbol{r}_k, \boldsymbol{z}_k >}{< A\boldsymbol{p}_k, \boldsymbol{p}_k >}$
6: $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
7: $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k$
8: $\boldsymbol{z}_{k+1} = M^{-1}\boldsymbol{r}_{k+1}$
9: $\beta_k = \dfrac{< \boldsymbol{r}_{k+1}, \boldsymbol{z}_{k+1} >}{< \boldsymbol{r}_k, \boldsymbol{z}_k >}$
10: $p_{k+1} = -\boldsymbol{z}_{k+1} + \beta_k \boldsymbol{p}_k$
11: **fin mientras**
12: **devolver** vector $\boldsymbol{x}_k$

4.6.2. Gradiente Biconjugado precondicionado (BCG pre)

De la misma forma que se siguió para encontrar el **CG pre**, se puede encontrar el Gradiente Biconjugado precondicionado (**BCG pre**), para un sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, donde A es S.D.P, sin aplicar explicitamente el precondicionamiento y se puede aplicar también si la matriz A no es simétrica. El algoritmo 15, muestra el método **BCG pre**.

Observación 4.5. Para el caso del **CGS** y el **BICGSTAB** no se puede implementar un precondicionamiento siguiendo los mismos pasos como el caso del **CG** y **CGS**, ya que se dan operaciones de alto costo como el producto matriz por matriz.

Algoritmo 15 BCG pre

Entrada: : matriz A, precondicionador M, vector $\boldsymbol{b}$,
vector inicial $\boldsymbol{x}_0$ y vector $\tilde{\boldsymbol{r}}_0$

Salida: : vector $\boldsymbol{x}_k$, solución del método $A\boldsymbol{x} = \boldsymbol{b}$

$\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b}$
$\boldsymbol{z}_0 = M^{-1} r_0$
$\tilde{\boldsymbol{z}}_0 = M^{-1}\tilde{\boldsymbol{r}}_0$
$\boldsymbol{p}_0 = -\boldsymbol{z}_0$
$\tilde{\boldsymbol{p}}_0 = -\tilde{\boldsymbol{z}}_0$
$k = 0$
mientras $\boldsymbol{r}_k \neq 0$ **hacer**
  $\alpha_k = \dfrac{< \boldsymbol{r}_k, \tilde{\boldsymbol{z}}_k >}{< A\boldsymbol{p}_k, \tilde{\boldsymbol{p}}_k >}$
  $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k + \alpha_k A\boldsymbol{p}_k$
  $\boldsymbol{z}_{k+1} = M^{-1}\boldsymbol{r}_{k+1}$
  $\tilde{\boldsymbol{r}}_{k+1} = \tilde{\boldsymbol{r}}_k + \alpha_k A^T \tilde{\boldsymbol{p}}_k$
  $\tilde{\boldsymbol{z}}_{k+1} = M^{-1}\tilde{\boldsymbol{r}}_{k+1}$
  $\beta_k = \dfrac{< \boldsymbol{r}_{k+1}, \tilde{\boldsymbol{z}}_{k+1} >}{< \boldsymbol{r}_k, \tilde{\boldsymbol{z}}_k >}$
  $\boldsymbol{p}_{k+1} = -\boldsymbol{z}_{k+1} + \beta_k \boldsymbol{p}_k$
  $\tilde{\boldsymbol{p}}_{k+1} = -\tilde{\boldsymbol{z}}_{k+1} + \beta_k \tilde{\boldsymbol{p}}_k$
  $k = k + 1$
fin mientras
devolver vector $\boldsymbol{x}_k$

Capítulo 5

Resultados Numéricos

En este capítulo se darán a conocer algunos resultados computacionales al utilizar matrices esparzas en operaciones de alto costo computacional, como el producto matriz por vector y la solución de sistemas lineales esparzas. Para ello se utilizarán diferentes estructuras de datos o esquemas de almacenamiento para matrices esparzas presentados en el cap 3 y la matriz densa, para comparar los tiempo de computo y la memoria utilizada entre estos y los resultados de la operación a realizar.

Para la realización de los algoritmos se ha utilizado el software *MATLAB R2014a* y en lenguaje C++, mediante el software *DEV-C++ 64 bits, versión 5.6.2* con el compilador *TDM-GCC 4.8.1 64-bit Debug*. Ambos software se trabajaron bajo la plataforma WINDOWS. El uso de MATLAB fue necesario para adquirir conocimientos iniciales en programación. La programación orientada a objetos, se realizó con el uso de clases, objetos y template en lenguaje c++, aplicando el concepto de apuntadores y lectura de archivos o ficheros. Se anexa un DVD con las respectivas implementaciones y documentación, siendo que para C++ se usó el paquete DOXYGEN para generar la documentación en html y en pdf [32].

Debido a que se han utilizado dos computadores para realizar estos resultados se describen a seguir la arquitectura de cada uno de estos equipos.

1. **Resultados Computador 1**: Computador portátil personal con las siguientes características:
    - Procesador Intel(R) Pentium(R) CPU P6100 @ 2.0GHz 2.0GHz
    - Memoria RAM de 3 GB
    - Sistema operativo de 64 bits, procesador por x64.
    - Memoria de disco duro de 500 GB.
    - 2 núcleos.

2. **Resultados Computador 2**: Este computador se adquirió gracias al proyecto de investigación llamado "Análisis teórico y computacional de las propiedades de matrices para ciertas discretizaciones numéricas" que se esta realizando en la Universidad de Nariño. A diferencia del computador 1, este computador posee mejores característica y se pueden abordar problemas con matrices muchos más grandes. Las especificaciones de este computador son las siguientes:

   - Procesador intel(R) Core(TM) i7-4790 CPU @ 3.60GHz 3.60GHz
   - Memoria RAM de 8 GB.
   - Sistema operativo de 64 bits, procesador por x64.
   - Memoria de disco duro de 2 TB.
   - 4 núcleos.

Las matrices esparzas que se van a utilizar para los resultados, han sido tomadas del recurso electrónico "Matrix Market" que se denotará por (**MM**) y el recurso electrónico de la Universidad de Florida que se denotará por (**UF**) [4, 33].

Las abreviaturas que se van a utilizar a lo largo de este capítulo son las siguientes:

- **Mem. Alm.**: Memoria usada que se obtuvo al almacenar la matriz leída desde un archivo, ya sea usando los esquemas de almacenamiento o la matriz densa. Para contabilizar este valor se utilizó el comando *whos* en MATLAB y para DEV-C++ el comando *system("tasklist")*. Por tanto no se va a realizar una comparación de memoria entre un software y otro.
- **T. Alm.**: Tiempo que se tardó en almacenar la matriz leída desde un archivo.
- **T. prod.**: Tiempo que se tardó en realizar el producto matriz por vector $A\boldsymbol{x}$.
- **T. Dis.**: Tiempo utilizado para realizar la discretización.
- **T. total**: Tiempo total utilizado por una operación en específico.
- **Densa:** Se refiere a la matriz densa.
- **T. Densa**: Tiempo que se tarda en realizar en método directo o iterativo asociado, mediante la matriz **densa**.
- **T. CSR**: Tiempo que se tarda en realizar en método directo o iterativo asociado, mediante el formato **CSR**.
- **T. CSV**: Tiempo que se tarda en realizar en método directo o iterativo asociado, mediante el formato **CSV**.

- **MB**: Denota las megabytes utilizadas.
- **GB**: Denota las gibaytes utilizadas.
- **seg**: Denota los segundos.
- **min**: Denota los minutos.
- **hrs**: Denota las horas.
- **Productos:** Número de productos matriz por vector realizados.
- **Iter.**: Número de iteraciones que realiza el método iterativo asociado.
- **V. máx.** y **V. mín.**: Denotan el valor del elemento más grande y más pequeño de la matriz respectivamente.
- **Val. inicial.**: Valor utilizado para incializar vectores asociados a los esquemas **AIJ**, **CSR** y **CSR mod.**
- **nnz**: Número de elementos no cero.
- **Porcentaje nnz**: Porcentaje de elementos no cero.
- **Diag. Dom.**: Diagonalmente dominante.
- **Cond.**: Número de condicionamiento respecto a la norma 1 de la matriz, proporcionado por **UF** y **MM**.
- **Met. Grad.**: Son los métodos del gradiente.
- **Fact_LU**: Tiempo realizado en la factorización LU.
- **Elim. Gauss**: Tiempo realizado en la eliminación Gaussiana.
- **Fact_Cho**: Tiempo realizado en la factorización de Cholesky.
- **Sol** $A\boldsymbol{x}$: Tiempo de solución del sistema lineal.
- **Residuo** ($\|\cdot\|_2$): Residuo evaluado en la norma 2.
- **Error** ($\|\cdot\|_2$): Error, medido por la norma 2.
- **Error max** ($\|\cdot\|_2$): Error máximo evaluado por la norma 2, respecto a los errores de la soluciones de un método para resolver varios sistemas lineales.
- **Residuo max** ($\|\cdot\|_2$): Residuo máximo evaluado por la norma 2, respecto a los residuo de las soluciones de un método para resolver varios sistemas lineales.

La metodología usada para concluir los tiempos computo, ha sido el realizar a cada problema el promedio del tiempo de computo de tres ejecuciones en instantes distintos. Por último, se ha considerado una tolerancia de 1.0E-12 para almacenar los elementos no nulos, es decir que se almacena la entrada $a_{i,j}$ de una matriz esparza A en las estructuras de datos, si $|a_{i,j}| > 1.E-12$. También este mismo valor se tomará como criterio de parada para los métodos del gradiente.

5.1. Producto matriz por vector

En este apartado se muestran algunos resultados respecto al producto de una matriz por vector $A\boldsymbol{x}$, donde A es una matriz esparza y el vector $\boldsymbol{x}$, es obtenido de manera aleatoria.

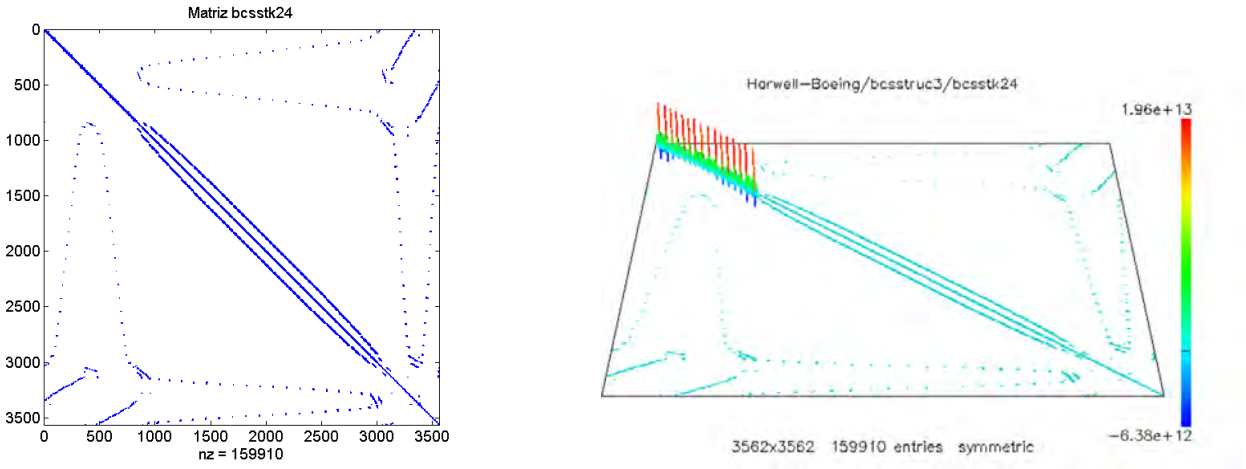
5.1.1. Resultados Computador 1

1. Matriz $A =$*bcsstk24* (**MM**)

   Para analizar el producto matriz por vector con la matriz *bcsstk24* del recurso "matrix market", se presenta en primer lugar el patrón de densidad y el intervalo de los valores de sus elementos diferentes de cero. En la figura 5.1 se observa en (a) el patrón de esparcidad y en (b) el intervalo donde se encuentran los elementos de la matriz, representados por puntos, lineas y por colores, con una barra que posee una línea negra que identifica el cero y la distribución de colores dependiendo del valor mínimo y máximo de las entradas de la matriz.

   Para ver datos más precisos de la matriz *bcsstk24*, con la tabla 5.1, se pude ver información, tales como: el orden de la matriz, el número de elementos no nulos (nnz), el porcentaje de elementos nulos, la simetría de la matriz (simetría) el condicionamiento de la matriz con respecto a la norma 1 (Cond.), la norma dos (norma 2), la norma de Frobenius (norma F), la disciplina y el año donde se ha obtenido esta matriz.

   Los resultados computacionales al realizar el producto matriz por vector con la matriz *bcsstk24*, se presentan en las tablas 5.2 y 5.3. En estas tablas se encuentra la información acerca del costo computacional de almacenamiento y tiempo de ejecución para los diferentes esquemas de almacenamiento implementados. En la tabla 5.2 se encuentran los resultados con el software MATLAB y en la tabla 5.3 con DEV-C++.



(a) Gráfico de esparcidad.

(b) Intervalo de las entradas [4].

Figura 5.1: Gráficos de la matriz *bcsstk24*.

Filas	3562
Columnas	3562
nnz	159910
Porcentaje nnz	1.26 %
Simetría	simétrica
Cond.	65
Norma 2	3.1E+13
Norma F	1.4E+14
Disciplina	Análisis dinámico en ingeniería estructural
Año	1983

Tabla 5.1: Datos de la matriz *bcsstk24* [4].

	Densa	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem. (MB)	101.503	1.962	1.336	2.134	1.308	1.308
T. Alm (seg)	2.865	2.916	2.736	3.505	2.949	2.882
T. prod (seg)	0.454	0.004	0.006	0.028	0.006	0.008

Tabla 5.2: Resultados producto matriz por vector *bcsstk24* MATLAB.

	Densa	AIJ	CSR	CSR MOD	MSR	CSV
Mem. (MB)	102.311	4.327	4.019	4.959	4.003	4.007
T. Alm (seg)	0.795	0.696	0.718	0.702	0.728	0.718
T. prod (seg)	0.089	0.001	0.001	0.001	0.001	0.002

Tabla 5.3: Resultados producto matriz por vector *bcsstk24* DEV-C++.

Observación 5.1. De los datos anteriores acerca del producto matriz por vector con la matriz *bcsstk24*, se pudo obtener lo siguiente:

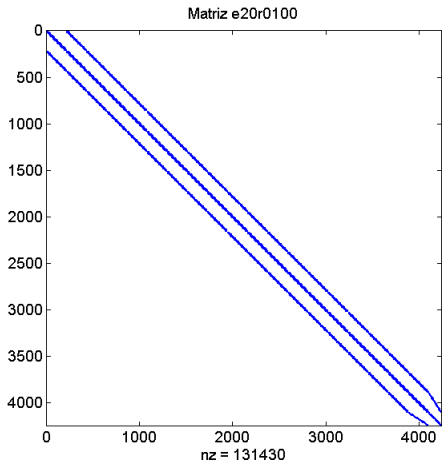
- De la figura 5.1, gráfica (a) se puede observar que existe una simetría de la distribución de los elementos no ceros, que se concentran en su mayoría cerca de la diagonal principal y en los extremos de las esquinas de la matriz. Además de la gráfica (b) se observa que el intervalo en el que se encuentran los valores de las entradas no nulas es muy grande, pero la mayoría de entradas se concentran en el color azul claro y por ende en un valor cercano a cero.
- De la tabla 5.1, se ve que la matriz es cuadra de orden 3562, el porcentaje de elementos es cerca del 1 %, la matriz es simétrica, el valor de la norma 2 es mas pequeño que la norma de frobenius, como se esperaba de la teoría y sus valores son grandes, pero el valor de condicionamiento es pequeño, frente a estos. Por la simetría los esquemas de almacenamiento sólo almacenan los elementos ubicados en la parte triangular superior de la matriz, por lo tanto almacenan menos o igual a la mitad los elementos nnz, es decir menos o igual que 79955.
- Al comparar los datos obtenidos de las tablas 5.2 y 5.3 se puede ver que en ambas tablas se gasta mucho más memoria en la matriz densa que en la demás ya que su valor es cerca de 102 MB, mientras que en las demás no superan las 5 MB. También, al comparar la memoria empleada por los diferentes esquemas de almacenamiento, se concluye que el esquema **CSR MOD** gasta más memoria que los demás y el que gasta menos está entre el **MSR** y el **CSV**.

  En el caso de los resultados del tiempo de almacenamiento en cada tabla, no hay diferencias notables, pero la eficiencia en el tiempo del producto si se empieza a detectar con los esquemas de almacenamiento en MATLAB frente a la matriz densa. Otro resultado es que al comparar los tiempo de almacenamiento entre los dos software, se puede ver que MATLAB se tarda más que con DEV-C++ . Por último el error para realizar el producto matriz por vector utilizando los esquemas con respecto al usar la matriz densa fue cero, por lo cual no se muestra en la tabla de resultados.

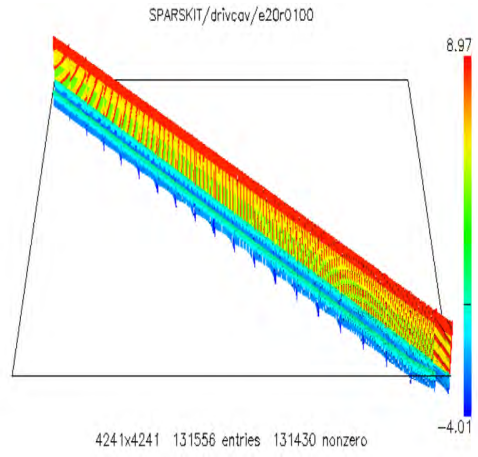
2. Matriz *A=e20r0100* **(MM)**

Igual que para la matriz anterior, en la matriz *e20r0100*, se va analizar la gráfica de la matriz para determinar el patrón de distribución y el intervalo de los elementos no nulos, como también los datos de la matriz y las tablas de resultados al realizar el producto matriz por vector con dicha matriz.

En la figura 5.2, la gráfica (a), se observa que las entradas no cero poseen un patrón estructurado, distribuidas en diagonales y cerca de la diagonal principal. De esta misma figura en la gráfica (b), se ve que las entradas son pequeñas debido a que se encuentran entre -4.01 y 8.97, por lo cual permite que no haya tantos errores de redondeo al realizar el producto.



(a) Gráfico de esparcidad.



(b) Intervalo de las entradas [4].

Figura 5.2: Gráficos de la matriz *e20r0100*.

De la tabla 5.4, se puede ver que la matriz tienen un mayor porcentaje de esparcidad y más grande que la anterior, dado que el porcentaje es del 0.73 % y es una matriz cuadrada de orden 4242. Para el caso de las normas, la norma de Frobenius y la norma 2 son pequeñas, pero el condicionamiento es muy grande, por lo cual la matriz es mal condicionada.

Filas	4241
Columnas	4241
nnz	131556
Porcentaje	0.73 %
Simetría	no simétrica
Cond.	2.15E+10
Norma 2	13
Norma F	390
Disciplina	Dinámica de fluidos
Año	1996

Tabla 5.4: Datos de la matriz *e20r0100* [4].

De las tablas 5.5 y 5.6, en primera instancia se puede ver que no se encuentra el esquema **MSR**, debido a que existen elementos nulos en la diagonal con la tolerancia propuesta. Respecto a la memoria, se utiliza más espacio la matriz densa respecto a los almacenamientos, puesto que tanto en ambas tablas el valor fue cercano a 143 MB, frente a un valor de 2 y 4 y 4 y 6, a los resultados de MATLAB y DEV-C++, respectivamente. Comparando este mismo valor entre los almacenamientos, el esquema que menos utilizo fue **CSV**, mientras el que más utilizo fue **CSR mod**. En caso del tiempo para el almacenamiento y para realizar el producto no hay diferencias significativas entre la matriz densa y los almacenamientos, pero si se observa que en el almacenamiento **CSR mod** en MATLAB supera los cinco segundos, mientras que los demás no superan esta cifra. También al comparar las tablas en el tiempo, en MATLAB se tarda más tiempo en almacenar que en DEV-C++. En los resultados del error, los esquemas presentan un error de 1.654E-14.

	Densa	**AIJ**	**CSR**	**CSR MOD**	**CSV**
Mem. (MB)	143.889	3.157	2.139	3.005	2.105
T. Alm (seg)	4.523	4.590	4.310	5.314	4.579
T. prod (seg)	0.593	0.007	0.005	0.129	0.007
Error $(\|\| \cdot \|\|)_2$		1.654E-14	1.654E-14	1.654E-14	1.654E-14

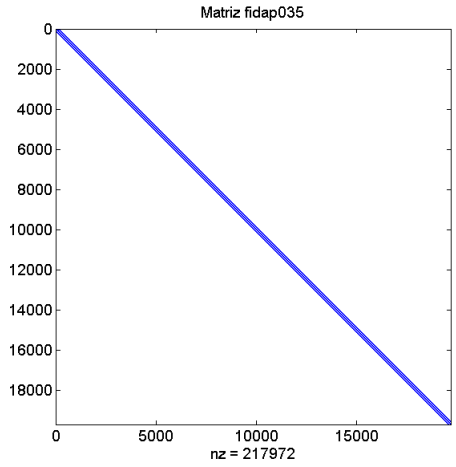
Tabla 5.5: Resultados producto matriz por vector *e20r0100* MATLAB.

	Densa	AIJ	CSR	CSR MOD	CSV
Mem. Alm (MB)	143.684	5.036	4.552	5.940	4.536
T. Alm (seg)	1.279	1.149	1.154	1.150	1.149
T. prod (seg)	0.125	0.002	0.001	0.001	0.001
Error ($\|\| \cdot \|\|_2$)		1.654E-014	1.654E-014	1.654E-014	1.654E-014

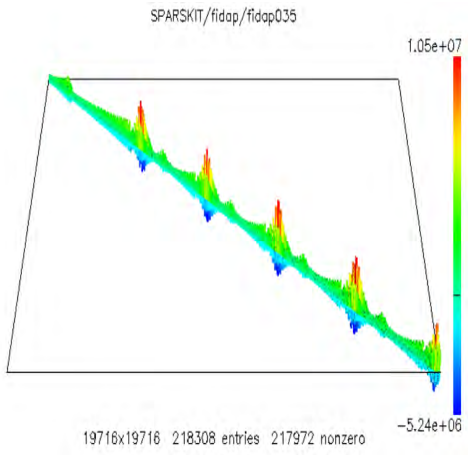
Tabla 5.6: Resultados producto matriz por vector *e20r0100* DEV-C++.

3. Matriz *A=fidap035* (**MM**)

De la figura 5.3 en la gráfica (a), se observa que sus elementos no cero tienen un patrón estructurado distribuidos en diagonal y se concentran en la diagonal principal. De la gráfica (b) se observa que los elementos oscilan entre -5.24 millones y 10.5 millones y que los mayores valores están fuera de la diagonal principal.



(a) Gráfico de esparcidad.



(b) Intervalo de las entradas [4].

Figura 5.3: Gráficos de la matriz *fidap035*.

De la tabla 5.7, se ve que la matriz es mucho más grande y más esparcida que las anteriores, ya que es una matriz cuadrada de orden 19716 y su porcentaje de densidad es de 0.056 %. También se ve que no es simétrica a pesar de tener patrón un simétrico y por tanto se ve la necesidad de almacenar todos sus elementos no ceros.

Filas	19716
Columnas	19716
nnz	217972
Porcentaje nnz	0.056 %
Simetría	no simétrica
Norma 2	3.5E+7
Norma F	3.2E+8
Diag. Dom.	no
Disciplina	modelado de elementos finitos

Tabla 5.7: Datos de la matriz *fidap035* [4].

Las tablas 5.8 y 5.9, nos provee información de la memoria utilizada para almacenamiento, la cual claramente es más eficiente que al usar la matriz densa. Si se compara este valor entre los almacenamientos **CSR MOD** utiliza mayor espacio y el de menor esta entre **MSR** y **CSV**, ya que su valor es cercano. sobre los resultados de tiempo la matriz densa se tardó mucho más en almacenar frente a los almacenamiento, puesto que se obtuvo un valor de 195.526 segundos frente a 7 a 8 segundos en MATLAB, mientras que 138.742 segundos frente a 2 a segundos en DEV-C++. Igualmente se presenta este comportamiento para realizar el producto.

De los tiempos de computo se puede destacar que se tarda mucho más tiempo en almacenar en MATLAB y para el caso del producto no hay diferencias notables notables con los almacenamientos, pero si con la matriz densa con un valor de 5295.107 seg con respecto a 218.099 seg. Sobre error del producto no presenta problemas, exceptuando **MSR**, que a diferencia de los demás es de valor 4.694E-7. La causa es debido a su estructura, ya que realiza el producto matriz por vector, empezando con los elementos de la diagonal y posteriormente los elementos que se encuentran fuera.

	Densa	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem. (MB)	3109.765	5.239	3.651	8.062	3.493	3.493
T. Alm. (seg)	195.526	7.526	7.125	8.982	7.761	7.558
T. prod (seg)	5295.107	0.009	0.005	0.204	0.009	0.034
Error $\|\cdot\|_2$		0	0	0	4.694E-7	0

Tabla 5.8: Resultados producto matriz por vector *fidap035* MATLAB.

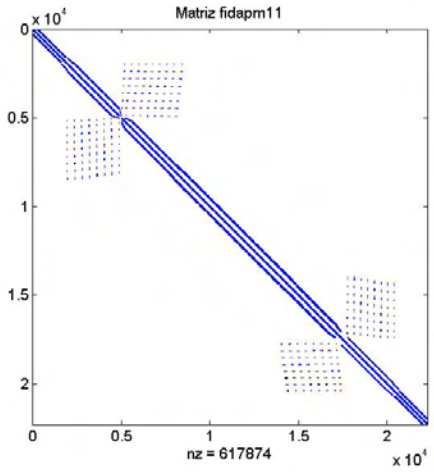
	Densa	AIJ	CSR	CSR MOD	MSR	CSV
Mem. (MB)	3041.756	6.464	5.684	8.452	5.604	5.608
T. Alm (seg)	138.742	2.740	1.950	2.745	1.903	1.877
T. prod (seg)	218.099	0.003	0.002	0.002	0.002	0.003
Error ($\|\| \cdot \|\|_2$)		0	0	0	4.694E-7	0

Tabla 5.9: Resultados producto matriz por vector *fidap035* DEV-C++.

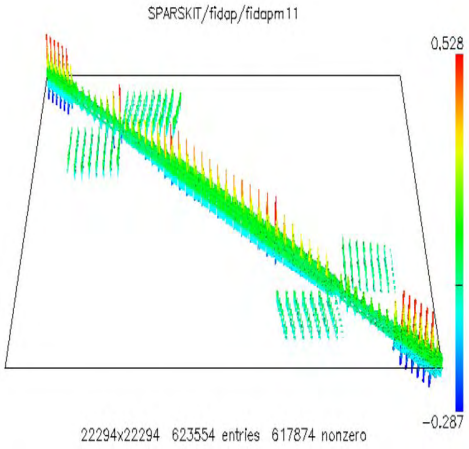
Debido a que se presentaron mejores resultados con las implementaciones realizadas en DEV-C++ que en MATLAB en tiempo de almacenamiento y el tiempo para realizar el producto matriz por vector en el caso de la matriz densa. Los siguientes pruebas se realizarán únicamente utilizando DEV-C++.

4. Matriz A=*fidapm11* (**MM**)

En la figura 5.4, gráfica (a), la matriz posee un patrón estructurado. En la gráfica (b), se puede determinar que los elementos están entre -0.287 y 0.528, por lo cual las elementos de la matriz son pequeños.



(a) Gráfico de esparcidad.



(b) Intervalo de las entradas [4].

Figura 5.4: Gráficos de la matriz *fidapm11*.

De la tabla 5.10, el orden de la matriz es cercano al de la matriz *fidap035*, pero el número de elementos cero es más grande que el anterior. Igualmente que en el anterior caso no se provee información del condicionamiento, pero si de las normas cuyos valores son considerables.

También se destaca el valor de las normas 2 y F, las cuales son bastante pequeñas.

Filas	22294
Columnas	22294
nnz	617874
Porcentaje nnz	0.12 %
Simetría	no simétrica
Cond.	no disponible
Norma 2	1.7
Norma F	26
Diag. Dom	no
Disciplina	modelación de elementos finitos

Tabla 5.10: Datos de la matriz *fidapm11* [4].

En la tabla 5.11, se observan resultados parecidos con los obtenidos en la matriz *fidap035*, referidos a la memoria y la eficiencia clara de realizar el producto con los esquemas, con la matriz densa, pero se destaca un aumento en los valores de memoria y tiempo de almacenamiento, de cada estructura con los anteriores y también de que no se puede utilizar **MSR**. Respecto al error del producto, fue cero y se puede relacionar esto con los valores pequeños de los elementos de la matriz.

-	**Densa**	**AIJ**	**CSR**	**CSR MOD**	**CSV**
Mem. (MB)	3887.796	12.712	10.380	17.124	10.296
T. Alm. (seg)	228.15	5.455	5.444	5.450	5.429
T. Prod (seg)	236.403	0.008	0.006	0.005	0.008

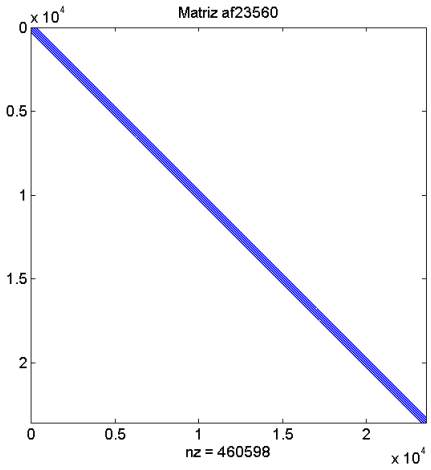
Tabla 5.11: Resultados matriz por vector *fidapm11* DEV-C++.

5. Matriz *A=af23560* (**MM**)

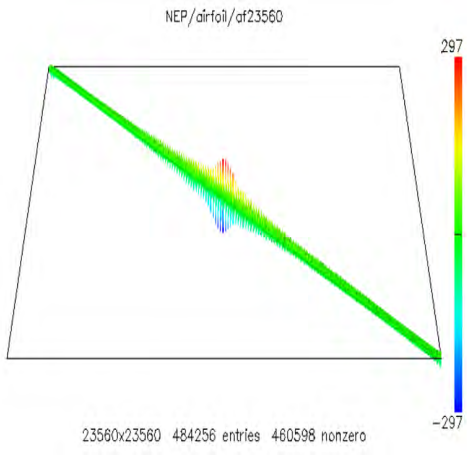
De la figura 5.5, gráfica (a), se puede ver un patrón estructurado , cuyos elementos se distribuyen diagonal, semejante a lo ocurrido con la matriz *fidap035*, sin embargo en este caso hay más elementos por el grosor de la gráfica. De esta misma figura en la gráfica (b), los valores se encuentran entre -297 y 297. Además se observa un predominio de color verde en esta gráfica que nos permite ver que hay una mayoría de elementos cercanos a cero y además que la matriz no es diagonalmente dominante debido al colo rojo o azul fuera de la diagonal.

De la tabla 5.12, se observa que la matriz tiene un tamaño poco más grande que las anteriores,

pero su número de elementos nulos es menor que en el anterior caso. A pesar de que el patrón sea simétrico la matriz no lo es. También no es diagonalmente dominante como se comprobó en la figura 5.5.



(a) Gráfico de esparcidad.



(b) Intervalo de las entradas [4].

Figura 5.5: Gráficos de la matriz *af23560*.

Filas	23560
Columnas	23560
nnz	460598
Porcentaje nnz	0.083 %
Simetría	no simétrica
Cond.	no disponible
Norma 2	6.5E+02
Norma F	1.0E+04
Diag. Dom.	no
Disciplina	Dinámica de Fluidos

Tabla 5.12: Datos de la matriz *af23560* [4].

De la tabla 5.13, se mantienen los resultados de memoria y tiempo anteriores, pero el tiempo de almacenamiento disminuyo debido a que es menos densa. Para el caso del error, comparado con los anteriores casos, se ve un aumento en el error aumento, con excepción de **MSR**.

-	Densa	AIJ	CSR	CSR MOD	MSR	CSV
Mem. Alm. (MB)	4341.544	10.252	8.536	13.420	8.448	8.452
T. Alm. (seg)	237.703	4.316	4.056	4.830	4.04	4.053
T. prod. (seg)	358.566	0.006	0.004	0.004	0.004	0.006
Error ($\|\| \cdot \|\|_2$)		4.357E-012	4.357E-012	4.357E-012	1.516E-011	4.357E-012

Tabla 5.13: Resultados matriz por vector *af23560* DEV-C++.

5.1.2. Resultados Computador 2

Debido a que se desea analizar matrices de mayor escala que las anteriores y poder ver mayores diferencias en la memoria y en los tiempos de operaciones entre los esquemas abordados, los siguientes resultados se realizaron en el computador 2, utilizando el software DEV-C++. Para mostrar las diferencia en resultados en este computador se retoma las pruebas para realizar el producto matriz por vector con la matriz *fidapm11* y la matriz *af23560* como se puede apreciar en las tablas 5.14 y 5.15 respectivamente.

-	Densa	AIJ	CSR	CSR MOD	CSV
Mem. Alm. (MB)	3887.796	12.712	10.380	17.124	10.296
T. Alm. (seg)	3.578	2.281	2.5	2.266	2.657
T. prod. (seg)	1.354	0.006	0.004	0.003	0.006

Tabla 5.14: Resultados producto matriz por vector *fidapm11*.

-	**Densa**	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem. Alm. (MB)	4341.544	10.252	8.536	13.420	8.448	8.452
T. Alm. (seg)	3.401	1.537	1.521	1.505	1.552	1.609
T. prod. (seg)	1.552	0.004	0.002	0.002	0.002	0.004
Error ($\|\cdot\|_2$)		4.357 E-12	4.357 E-12	4.357 E-12	1.516 E-11	4.357 E-12

Tabla 5.15: Resultados matriz por vector *af23560*.

Observación 5.2. De las tablas 5.14 y 5.15 referentes a las pruebas en el computador 2 de la matriz *fidapm11* y *af23560* respectivamente, si se comparan estos resultados con los del computador 1, el tiempo de almacenamiento y el tiempo para realizar el producto se redujo notablemente, sin poder observar diferencias notables entre la matriz esparza y la matriz densa. En el caso de la memoria y el error, estos valores se mantiene, independientemente del computador el cual que se esté usando.

Para los siguientes matrices han sido tomadas de la colección de matrices de la "Universidad de Florida" (**UF**). La información tal como el intervalo donde se encuentran los elementos de la matrices, el valor de las normas y el condicionamiento no la proporciona esta página, sin embargo se ha calculado el valor máximo y mínimo, cuyo valor se anexa en la tabla de datos de la matriz.

1. Matriz *pdb1HYS* (**UF**)

   Por medio de la figura 5.6, se puede detectar que la matriz *pdb1HYS*, presenta un patrón en su elementos, considerándola como estructurada y además están distribuidos en diferentes partes de la matriz, en particular se concentran más cerca de la diagonal principal.

   De la tabla 5.16, se ve que la matriz es cuadrada de orden 36417 y simétrica, que se verifica en parte por el patrón de los elementos y por tanto los esquemas almacenan un valor menor o igual a 2172382. Por el valor máximo y mínimo de los elementos de la matriz, el intervalo donde se encuentran no es tan amplio.

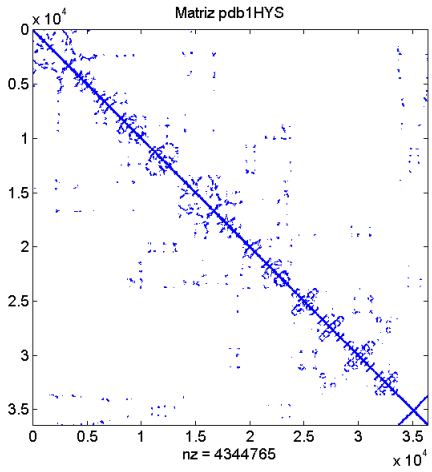


Figura 5.6: Gráfico de esparcidad de la matriz *pdb1HYS*.

Filas	36417
Columnas	36417
nnz	4344765
Porcentaje nnz	0.33 %
V. máx	257.076
V. mín	76.680
Simetría	simétrica
Disciplina	teoría de grafos
Año	2008

Tabla 5.16: Datos de la matriz *pdb1HYS* [33].

De la tabla 5.17, se puede notar que la memoria para la densa es muy grande en comparación de las esparzas, ya que su valor es de 6591 MB frente a un valor de 50 MB que fue el más grande usado por **CSR mod**. Para el tiempo de almacenamiento y el tiempo del producto no se observa diferencias de tiempo entre los almacenamientos, pero si con la densa, la cuál tarda más, pero a pesar de seguir aumentando la memoria el tiempo no se torna tan grande como sucedió con los resultados del computador 1.

-	Densa	AIJ	CSR	CSR MOD	MSR	CSV
Mem. Alm. (MB)	6591.284	37.288	28.864	50.308	28.724	28.728
T. Alm. (seg)	13.297	6.969	6.953	6.984	7.094	7.047
T. prod. (seg)	8.687	0.047	0.038	0.037	0.036	0.045

Tabla 5.17: Resultados producto matriz por vector *pdb1HYS*.

2. Matriz *c20stif* (**UF**).

En la figura 2, se determina que *c20stif* posee un patrón simétrico en sus elementos que es muy parecido al de la matriz *pdb1HYS*.

De la tabla 5.18, la matriz es más grande y más esparcida que la anterior y es simétrica, que concuerda con el patrón simétrico de los elementos no ceros. Por tanto los esquemas almacenan cerca de la mitad del valor *nnz*.

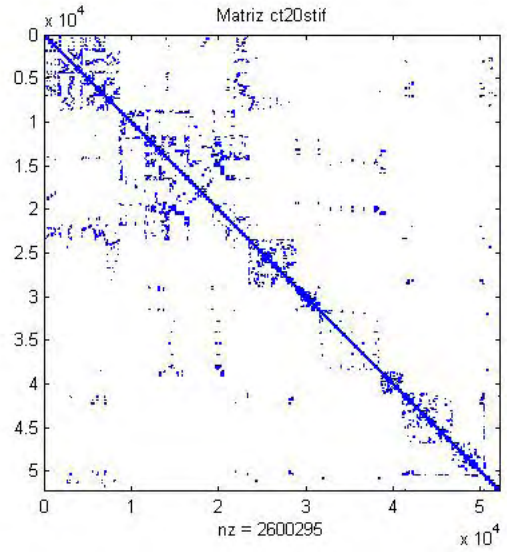


Figura 5.7: Gráfico de esparcidad de la matriz *ct20stif*.

De la tabla 5.19, claramente se observa la eficiencia de los esquemas frente a la densa, en los

datos obtenidos. En el caso de los resultados entre cada uno de los almacenamientos, **CSR mod**, gasta más memoria y quien tarda más en almacenar la matriz es *CSR*. Para el caso del error, el valor de este fue el mismo para todos los esquemas y sobrepasa a la tolerancia propuesta.

Filas	52329
Columnas	52329
nnz	2595415
Porcentaje nnz	0.09 %
V. máx	8.870E-10
V. mín	-1.762E+10
Simetría	simétrica
Disciplina	Problema estructural
Año	1995

Tabla 5.18: Datos de la matriz *ct20stif* [33].

-	**Densa**	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem. Alm. (MB)	6796.284	23.736	18.760	33.528	18.560	18.564
T. Alm. (seg)	79.526	4.958	11.846	6.798	4.533	4.526
T. prod. (seg)	619.248	0.033	0.007	0.024	0.006	0.009
Error ($\|\| \cdot \|\|_2$)		1.455 E-011	1.455 E-011	1.455 E-011	1.455 E-011	1.455 E-011

Tabla 5.19: Resultados producto matriz por vector *ct20stif*.

Para las siguientes pruebas, debido a que las matrices son de mayor escala que las anteriores y la mayor cantidad máxima posible para almacenar una matriz densa es de tamaño alrededor de orden 80000, los resultados que se presentan a continuación se analiza únicamente con los esquemas de almacenamiento tratados.

3. Matriz *A=gsm_106857*

Por la figura 3, los elementos están distribuidos en diferentes lugares de la matriz y a simple vista no es posible detectar si existe o no un patrón. A simple vista parece que la matriz no es esparza, sin embargo esto se debe a la vista global de la gráfica.

Por la tabla 5.20, la matriz es cuadrada de orden 589446, el número de elementos nulos es de 21758924, y el porcentaje de estos elementos que es de 6.2625E-3 %, lo que nos índica que es muy esparza y el cuidado que hay que tener con la gráfica de esparcidad. También nos índica que es simétrica lo cuál no era tan fácil de suponer de la gráfico de esparcidad.

La tabla 5.21, nos índica que la memoria de almacenamiento aumentó considerablemente en los esquemas, pasando a un valor superior a 100 MB. Para el caso del tiempo en el almacenamiento no hay diferencias notables e igualmente para realizar el producto.

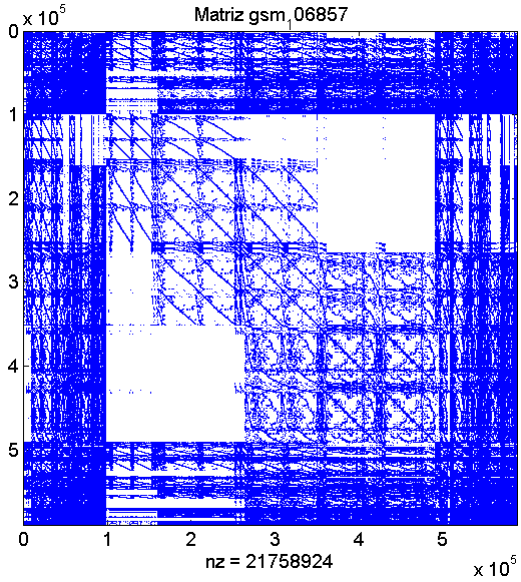


Figura 5.8: Gráfico de esparcidad de la matriz *gsm06857*.

Filas	589,446
Columnas	589,446
nnz	21,758,924
Porcentaje nnz	6.262E-3 %
V. máx.	13.342
V. mín.	-5.084
Simetría	simétrica
Disciplina	Problema de electromagnetismo
Año	2010
Definida positiva	no

Tabla 5.20: Datos de la matriz *gsm06857* [33].

-	AIJ	CSR	CSR MOD	MSR	CSV
Mem. Alm. (MB)	177.652	136.300	253.224	133.996	134.000
T. Alm. (seg)	36.568	36.527	36.625	36.751	36.578
T. prod. (seg)	0.12	0.104	0.125	0.094	0.120

Tabla 5.21: Resultados matriz por vector *gsm_106857*

4. Matriz A = *webbase-1M* (**(MM)**)

La figura 5.9, permite ver que no existe un patrón de distribución de sus elementos, por tanto se considera como matriz esparza no estructurada.

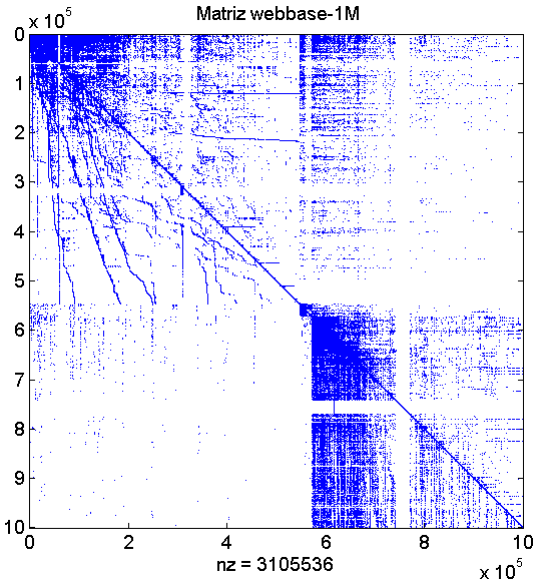


Figura 5.9: Gráfico de esparcidad de la matriz *webbase*.

De la tabla 5.22, el tamaño de la matriz es casi el doble que la matriz anterior, pero el número de elementos distintos de cero es menor y no es simétrica. También se puede notar que por los valores máximos y mínimos la matriz tiene entradas pequeñas.

De la tabla 5.23, se aprecia que a pesar de ser de gran tamaño los valores de tiempo y memoria son menores a los de la matriz a *gsm106857*, lo cual indica la relevancia de los elementos no

nulos frente a las dimensiones de la matriz.

Filas	1000005
Columnas	1000005
nnz	3105536
V. máx	1
V. mín	-3.486E-5
Densidad	0.05 %
Simetría	no simétrica
Disciplina	gráfico directo ponderado
Año	2008
Definida positiva	no

Tabla 5.22: Datos de la matriz *webbase-1M* [33].

-	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem. Alm. (MB)	51.576	43.348	112.804	39.440	39.444
T. Alm. (seg)	9.301	9.302	9.394	9.378	9.344
T. prod. (seg)	0.020	0.016	0.021	0.016	0.016

Tabla 5.23: Resultados matriz por vector *webbase-1Mes*.

5. Matriz *A=thermal2*

De la figura 5.10, se deduce que la matriz tiene un patrón en sus elementos concentrados mayormente en la diagonal principal, por lo cual es estructurada.

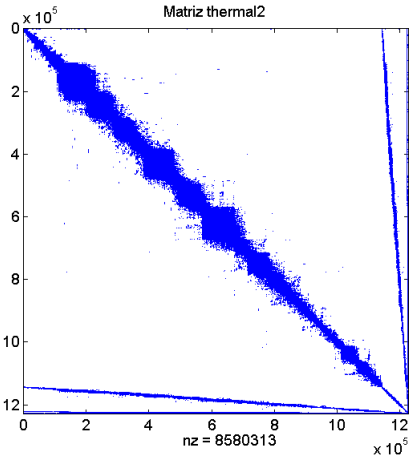


Figura 5.10: Gráfico de esparcida de la matriz *thermal2*.

De la tabla 5.24, se observa que la matriz es de orden superior a 10^6, la matriz es S.D.P, tiene un número de elementos $nnz = 8580313$ y para aprovechar la simetría los elementos almacenan cerca de la mitad de esto. Por otro lado, los elementos de la matriz son pequeños puesto que se encuentran en un intervalo entre 4.874 y -1.599.

Filas	1228045
Columnas	1228045
nnz	8580313
Porcentaje nnz	5.689E-4
Simetría	S.D.P
V. máx.	4.874
V. min.	-1.599
Año	2006
Aplicación	Problema térmico

Tabla 5.24: Datos matriz *thermal2* [33].

De la tabla 5.25, se observa el mismo patrón de comportamiento en la memoria y en los tiempos de almacenamiento. Para el producto con **CSV** es el esquema que tarda más.

-	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem (MB)	79.692	65.332	159.624	60.532	60.536
T. Alm. (seg)	16.167	16.189	16.273	16.189	16.204
T. prod (seg)	0.053	0.046	0.066	0.044	0.203

Tabla 5.25: Resultados matriz por vector *thermal2*.

6. Matriz $A = Hook\_1498$

Por la figura 5.11, se observa un patrón de esparcidad de los elementos, concentrados en la diagonal principal.

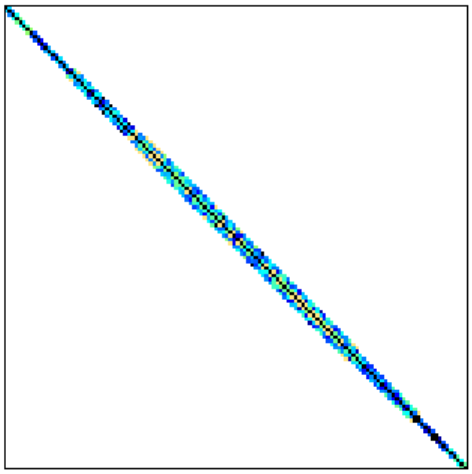


Figura 5.11: Gráfico esparcidad de la matriz *Hook_1498* [33].

De la tabla 5.26, la matriz tiene un tamaño 1498023 × 1498023, el número de elementos no nulos es de 59374451, con un porcentaje del 2.645E-3 %. También la matriz es simétrica por lo cual se almacena una parte de elementos. Por el intervalo de elementos máximos y mínimos se encuentra en un intervalo entre -102007.12 y 157592.67.

Filas	1498023
Columnas	1498023
nnz	59374451
Porcentaje nnz	2.645E-3 %
Simetría	Simétrica
V. máx.	157592.67
V. mín.	-102007.12
Disciplina	Problema estructural
Año	2011

Tabla 5.26: Datos de la matriz *Hook_1498* [33].

En la tabla 5.27, se observa un aumento considerable en el uso de la memoria de los almacenamiento y en particular con **CSR mod** seguida de **AIJ**, los cuales en el tiempo del producto tardan más en realizar el producto.

-	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem. (MB)	478.416	365.424	674.416	359.576	359.580
T. Alm. (seg)	99.272	99.062	97.981	98.34	98.341
T. Prod. (seg)	0.224	0.182	0.224	0.167	0.198

Tabla 5.27: Resultados matriz por vector *Hook_1498*.

Ahora para las siguientes matrices que sean no simétrica si analizará además del producto matriz por vector, el producto matriz transpuesta por vector, que nos servirá para analizar como se comporta al aplicar con los esquemas utilizados.

7. Matriz *A=circuit5M_dc*

Por la figura 5.12, la matriz tiene un patrón estructurado en sus elementos y esta distribuidos en diferentes lugares de la matriz y una cierta concentración de elementos, en la parte superior izquierda de la gráfica.

Por los datos de la tabla 5.28, la matriz es más grande que las anteriores, es simétrica, por lo que se debe almacenar todos los elementos *nnz* e implicaría ahorro de memoria y costo de tiempo.

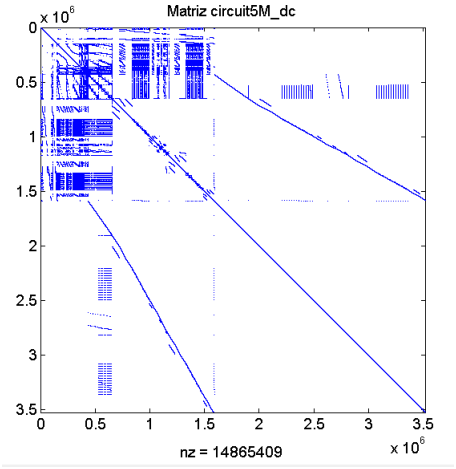


Figura 5.12: Gráfico de esparcidad de la matriz *circuit5M_dc* [33].

De la tabla 5.29, en relación al tiempo de almacenamiento por segundos se tardan más en almacenar **CSR mod** y **CSV**. Sobre el tiempo de los productos se verifica que se realiza más rápido el proceso con el producto matriz por vector (T. prod) que con el producto matriz transpuesta por vector (T. Prod. Trans.) que es mayor al comparar con cada esquema.

Filas	3523317
Columnas	3523317
nnz	14865409
Porcentaje nnz	1.197E-4 %
Simetría	Simétrica
V. máx.	19194192
V. mín.	-59.347
Disciplina	Simulación de circuitos.
Año	2010

Tabla 5.28: Datos matriz *circuit5M_dc* [33].

-	**AIJ**	**CSR**	**CSR MOD**	**MSR**	**CSV**
Mem.(MB)	216.188	176.688	439.096	162.896	162.924
T. Alm. (seg)	60.923	60.879	62.011	60.983	63.450
T. prod (seg)	0.078	0.062	0.094	0.062	0.078
T. Prod. Trans (seg)	0.094	0.0737	0.109	0.078	0.093

Tabla 5.29: Resultados matriz por vector *circuit5M_dc*.

8. Matriz *A=circuit5M*

De la figura 5.13, no es posible determinar si la matriz es estructura o no a simple vista, también se observa que hay concentración en diagonal de los elementos no cero.

De la tabla 5.30, se verifica que la matriz es no simétrica, de mayor tamaño que la anterior y es muy esparza. Por el valor máximo y mínimo de entradas la matriz no tiene entradas muy grandes.

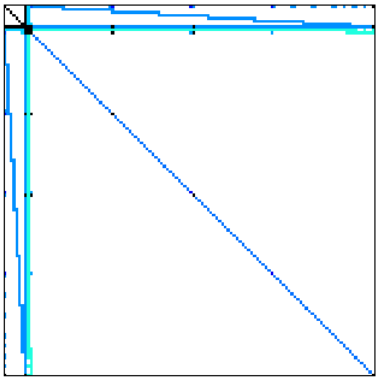


Figura 5.13: Gráfico de esparcidad de la matriz *circuit5M* [33].

Filas	5558326
Columnas	5558326
nnz	59524291
Porcentaje nnz	1.927E-4 %
Simetría	No Simétrica
V. máx.	1022
V. mín.	-133.333
Disciplina	Simulación de circuitos.
Año	2010

Tabla 5.30: Datos matriz *circuit5M* [33].

De la tabla 5.31, se consideró los esquemas de almacenamiento abordados pero siguiendo un orden de jerarquía por columna seguido por filas. Entre los resultados de memoria, se da un aumento en la memoria de los esquemas notablemente y en **CSR mod** es mayor a 1 GB. Sobre los tiempos al realizar el producto matriz por vector y al realizar el producto matriz transpuesta por vector, se tarda más al realizar el producto matriz por vector que al realizar el producto matriz transpuesta por vector, a diferencia de lo que ocurría con el resultado anterior.

-	**AIJ**	**CSC**	**CSC MOD**	**CSV**
Mem. (MB)	933.184	722.388	1398.128	700.664
T. Alm. (seg)	197.677	198.538	199.697	198.690
T. Prod. (seg)	0.354	0.260	0.265	0.302
T. Prod. Trans. (seg)	0.307	0.227	0.236	0.273

Tabla 5.31: Resultados matriz por vector *circuit5M*.

9. Matriz A=*cage15*

De la figura 5.14, la matriz posee un cierto patrón simétrico en sus elementos. También De la tabla 5.32, la matriz es de igual tamaño a la anterior, pero aumenta su número *nnz* y es no simétrica por lo cuál se requiere almacenar todos los elementos no ceros y de la tabla 5.33, se aprecia que todos los esquemas superan en espacio de memoria a 1 GB. Sobre el tiempo para realizar el producto se presentan mejores resultados al realizar el producto con la matriz transpuesta con un diferencia de 2 o 3 décimas de segundo que pueden afectar a la hora de realizar varios productos. El que mejor resultados se obtuvo en ambos producto fue **CSC** y **MSC** y también en el almacenamiento.

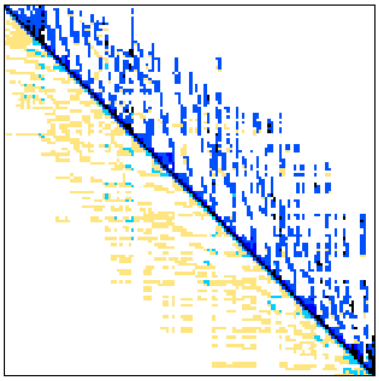


Figura 5.14: Gráfico de esparcidad de la matriz *cage15* [33].

Filas	5154859
Columnas	5154859
nnz	99199551
Porcentaje nnz	3.7331E-4 %
Simetría	No Simétrica
Disciplina	Grafo directo ponderado.
Año	2003

Tabla 5.32: Datos matriz *cage15* [33].

-	AIJ (COL)	CSC	CSC MOD	MSC	CSV (COL)
Mem. (GB)	1.553	1.185	2.177	1.166	1.171
T. Alm. (seg)	312.031	310.51	314.540	310.485	312.881
T. Prod. (seg)	0.617	0.421	0.492	0.406	0.562
T. Prod. trans. (seg)	0.453	0.313	0.421	0.297	0.422

Tabla 5.33: Resultados matriz por vector *cage15*.

10. Matriz *A=nlpkkt160*

De la figura 5.15, la matriz posee un patrón estructurado y tiene elementos nulos en la diagonal por lo que no es posible aplicar **MSR**.

Gracias a la tabla 5.34, se observa una matriz superior a 10^6, muy esparza con elementos entre -16 y 200, por lo que son entradas no muy grandes. Además debido a su simetría se almacena la mitad de sus elementos es decir cerca o menor de 112711056.

Filas	8345600
Columnas	8345600
nnz	225422112
Porcentaje nnz	3.236E-4 %
Simetría	Simétrica
V. máx.	-16
V. mín.	200
Disciplina	Problema de optimización.
Año	2008

Tabla 5.34: Datos matriz *nlpkkt160* [33].

Mediante la tabla 5.35, se puede apreciar que la memoria de los formatos sobrepasa a 1GB, donde **CSR mod** posee mayor este valor y **CSV** tiene menor valor. Sobre los tiempos **CSV** se tarda mucho más en almacenar la matriz que en los demás y en el tiempo para realizar un producto **CSR** realizar más rápido este proceso.

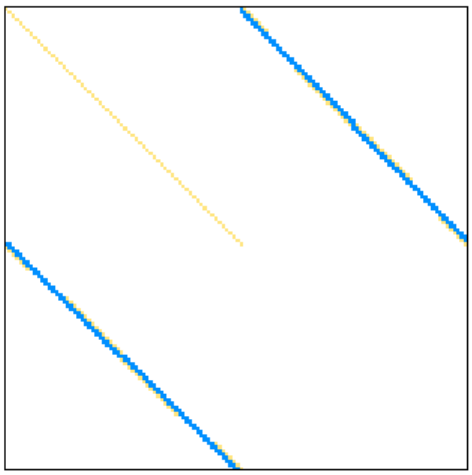


Figura 5.15: Gráfico de esparcidad de la matriz *nlpkkt160* [33].

-	**AIJ**	**CSR**	**CSR MOD**	**CSV**
Mem. (GB)	1.797	1.381	2.480	1.349
T. Alm. (seg)	319.473	318.952	318.260	339.950
T. Prod. (seg)	0.826	0.716	0.795	0.835

Tabla 5.35: Resultados matriz por vector *nlpkkt160*.

5.2. Discretización de la ecuación de onda unidimensional

En este caso, se va a realizar utilizando el computador 2 una discretización numérica de un PVI de una ecuación diferencial parcial de la ecuación de onda unidimensional

$$u_x + u_t = 0, \quad \text{en} \quad -2 \leq x \leq 3, \ t \geq 0$$

con el valor inicial

$$u(x,0) = \begin{cases} 1 - |x| & \text{si} |x| \leq 1 \\ 0 & \text{si} |x| \geq 1 \end{cases},$$

mediante el **esquema de Lax-Friedrichs** [29], con la formula

$$u_m^{n+1} = \frac{1}{2}\left(u_{m+1}^n + u_{m-1}^n\right) - \frac{1}{2}\lambda\left(u_{m+1}^n - u_{m-1}^n\right),$$

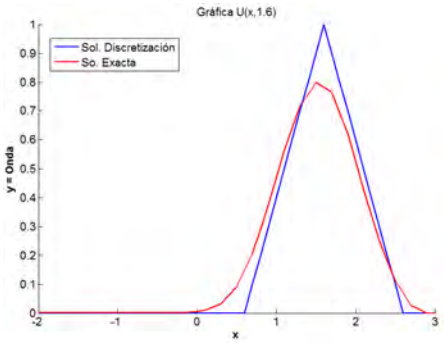
donde $\lambda = 0.8$, las condiciones de frontera a derecha son $u_M^{n+1} = u_{M-1}^{n+1}$, donde $x_M = 3$ y el valor inicial de la d $u_m = u_0(x_m)$.

La característica de esta discretización es que el proceso se realiza de forma iterativa productos matriz por vector, en la cual la matriz es esparza. A continuación mediante la figura 5.2 en el gráfico (a) el gráfico de esparcidad de la matriz de discretización de tamaño de 51×51 y en el gráfico (b) el gráfico de esparcidad de la matriz de discretización de un tamaño de 5001×5001.

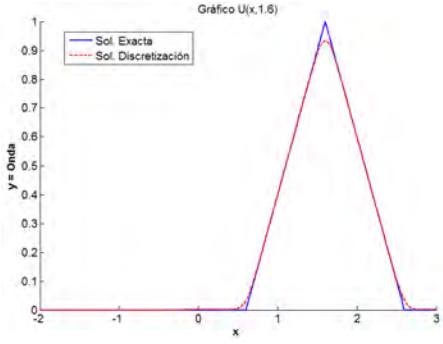
(a) matriz 51×51 (b) matriz 5001×50001

Figura 5.16: Gráficos de esparcidad, matrices de discretización ec. onda

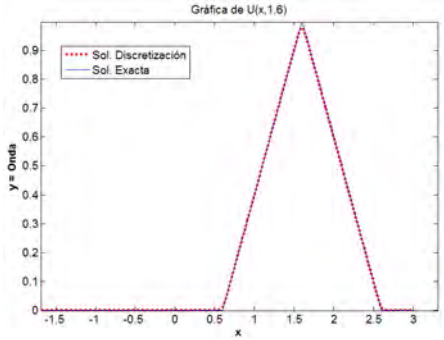
Ahora mediante la figura 5.2, se observa el comportamiento de las gráficas en $u(x, 1.6)$, respecto a la soluciones de las discretizaciones utilizadas representadas por el color rojo y la solución exacta representada por el color azul. De esto se puede ver que al ir aumentando la matriz de discretización se aproximan ambas gráficas. Sin embargo por los costos de memoria y el tiempo de computo, se requiere utilizar esquemas de almacenamiento. A continuación se muestra algunos resultados al aplicar esquemas de almacenamiento en el proceso de discretización con matrices de discretización de diferentes tamaños.



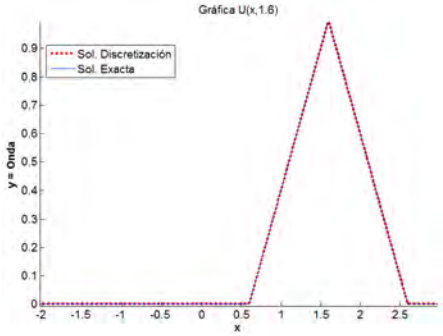
(a) matriz de 51 × 51



(b) matriz de 501 × 501



(c) matriz de 5001 × 5001



(d) matriz de 50001 × 50001

Figura 5.17: Gráficos de la solución de discretización ec. onda en $u(x, 1.6)$.

De la tabla 5.36, la matriz de discretización de tamaño 5001 × 5001 es una matriz bastante esparza con un porcentaje de 0.004 %, la cual no es simétrica, como se observa por la figura 5.2, gráfico (a), donde se puede apreciar que por el comportamiento de las entradas no posee un patrón simétrico. También se puede observar que para la discretización se necesita alrededor de 2000 productos (*Num. productos*) para realizar este proceso.

De la tabla 5.37, se requiere más espacio de memoria para la matriz densa y entre los esquemas, **CSV** es el que menos memoria utiliza, mientras que el **CSR mod** es el que más memoria utiliza; en el tiempo de almacenamiento no presenta diferencias notables con los esquemas, pero a la hora de realizar la discretización (*T. Dis.*), con la realización de 2000 productos, se puede notar que la matriz densa es muy ineficiente comparado con lo obtenido con los esquemas, los cuales no superan un segundo, al realizar todas estas operaciones. Sobre el error asociado al vector de la solución exacta (Error $\|\boldsymbol{y}_i - \boldsymbol{y}_i^*\|$), su valor tanto con la densa como las esparzas es de 0.113, lo que indica que independientemente de que estructura se utilice sigue siendo la misma solución.

Filas	5001
Columnas	5001
nnz	10002
Porcentaje nnz	0.04 %
Simetría	no simétrica
Num. productos	2000

Tabla 5.36: Datos matriz de discretización 5001 × 5001 ec. onda.

-	**Densa**	**AIJ**	**CSR**	**CSR MOD**	**CSV**
Mem. (MB)	199.196	3.404	3.380	3.736	3.360
T. Alm. (seg)	0.125	0.026	0.026	0.031	0.031
T. Dis. (seg)	114.955	0.104	0.088	0.093	0.109
T. total (seg)	115.08	0.13	0.114	0.124	0.14
Error $\|\boldsymbol{y_i} - \boldsymbol{y_i^*}\|_2$	0.113	0.113			

Tabla 5.37: Resultados, matriz de discretización 5001 × 5001 ec. onda.

De la tabla 5.38, la matriz de discretización de tamaño 50001 × 50001, es una matriz bastante esparza con un porcentaje de 0.004 %, no es simétrica y se necesita a diferencia de la anterior 20000 productos para la discretización. Respecto a los resultados al realizar la discretización no se tiene en cuenta la matriz densa, dado que por el número de productos que se necesitan realizar seria demasiado costoso. Así, analizando los resultados únicamente con los esquemas de almacenamientos se detecta que con **CSR MOD** se utiliza casi el doble de memoria que los demás; el tiempo de almacenamiento no hay diferencias notables; el tiempo de discretización, se observa que **AIJ** y **CSV** se tarda unos segundos más con respecto a **CSR** y **CSR mod**, y el que menos tarda es el **CSR mod**. El error disminuye cerca de 0.05, con respecto al utilizar la matriz de discretización anterior.

Filas	50001
Columnas	50001
nnz	100002
Porcentaje nnz	0.004 %
Simetría	no simétrica
Num. Productos,	20000

Tabla 5.38: Datos matriz de discretización 50001 × 50001 ec. onda.

-	**AIJ**	**CSR**	**CSR MOD**	**CSV**
Mem. (MB)	4.820	4.608	8.032	4.428
T. Alm. (seg)	0.302	0.297	0.302	0.302
T. Dis. (seg)	11.120	9.229	9.089	10.448
T. Total (seg)	11.152	9.526	9.391	10.75
Error $\|\boldsymbol{y_i} - \boldsymbol{y_i^*}\|_2$	0.0635			

Tabla 5.39: Resultados, matriz de discretización 50001 × 50001 ec. onda.

Los datos de la tabla 5.40 indican la información de la matriz de discretización de tamaño 500001 × 500001, la cual tiene un aumento proporcional en el tamaño, en el número nnz y en la cantidad de producto utilizados para la discretización que hace que este proceso se vuelva más costo, pero el error debe disminuir.

Filas	500001
Columnas	500001
nnz	1000002
Porcentaje nnz	4E-6 %
Simetría	no simétrica
Num. productos	200000

Tabla 5.40: Datos matriz de discretización 500001 × 500001 ec. onda.

De la tabla 5.2, se obtuvo que los tiempos de discretización a diferencia de la anterior matriz fueron más eficientes con **CSR** y **CSV**, que con **CSR mod**, por lo que varios productos y la memoria utilizada debido al aumento de nnz hacen el proceso más lento con este esquema. De la misma tabla el error asociado disminuyo con relación a resultados anteriores, con una disminución cerca del 0.03 %.

-	**CSR**	**CSR MOD**	**CSV**
Mem. (MB)	16.704	46.424	14.956
T. Alm. (seg)	3.014	3.034	3.043
T. Dis. (min)	23.589	34.723	22.861
Error $\|\boldsymbol{y_i} - \boldsymbol{y_i^*}\|_2$	0.0357		

Tabla 5.41: Resultados, matriz de discretización ec. onda 500001 × 500001.

A continuación, en la tabla 5.42, se presentan algunos resultados con **CSR** y **CSV** al realizar el proceso de discretización con diferentes espaciamientos, en donde se puede detectar que se dan mejores resultados con **CSV**, por décimas o centésimas de segundo y para el caso del

error se observa una disminución del error de cerca de 0.02 al ir disminuyendo el espaciamiento siguiendo una proporción.

	$h = 10^{-3}$	$h = 5 \times 10^{-4}$	$h = 2.5 \times 10^{-4}$	$h = 1.2 \times 10^{-4}$
T. CSR. (seg)	0.104	0.375	1.515	6.526
T. CSV (seg)	0.094	0.364	1.453	6.323
Error $\|y - y^*\|_2$	0.113	0.095	0.080	0.067

Tabla 5.42: Resultados discretización ec. onda diferentes espaciamientos.

5.3. Inserción de elementos

En esta parte se presenta los resultados en el proceso de inserción de entradas a la matriz, para el almacenamiento leída desde un archivo, la cuál tiene la forma similar al esquema coordinado pero con la diferencia que en este caso no se cuenta con la información de la simetría de la cantidad de elementos nnz y la información de los elementos no ceros esta distribuida de forma aleatoria.

Los esquemas que se van a utilizar en esta parte son **AIJ**, **CSR** y **CSR mod**. El esquema **MSR**, no se tomó en cuenta puesto que esta condicionado a que la matriz diagonal tenga todos sus elementos diferentes de cero en la diagonal y tampoco **CSV**, dado que no tiene un apuntador de filas o columnas necesaria para realizar de forma eficiente esta operación y se toma como representante para este tipo de formatos que no poseen apuntador a **AIJ**. También Debido a que no se cuenta con el número nnz, se requiere inicializar con un tamaño los vectores de los esquemas. Para ello se tomó como valor por defecto la cuarta parte de la cantidad de entradas de la matriz para **AIJ** y **CSR** y para el caso de **CSR mod**, se tomó la mitad del valor del tamaño de las columnas de la matriz, puesto que este esquema esta divido por vectores relacionados a cierta fila de una matriz y por ende en el peor de los casos la longitud máxima de los vectores es el valor de columnas de la matriz.

5.3.1. Resultados computador 1

1. **Matriz Aleatoria A.**

   En la figura 5.18, se tiene el gráfico de esparcidad de la matriz A, donde la gráfica (a) se observa dicho gráfico de forma global que nos permite ver que la matriz no es estructurada y por la gráfica (b), se observa que los elementos no nulos están distribuidos a lo largo de toda

la matriz y que debido a ello la gráfico global se ve muy densa.

(a) gráfica global

(b) gráfica con zoom

Figura 5.18: Gráfico de esparcidad matriz aleatoria A.

En la tabla 5.43, se observa que la matriz es pequeña de tamaño 500×500 y tiene una esparcidad del 45.10 %, por lo cual se verifica que es esparza.

filas	500
Columnas	500
nnz	112752
Porcentaje nnz	45.10 %

Tabla 5.43: datos de la matriz aleatoria A.

Con los resultados de la tabla 5.44 ,se observa en primer lugar que la memoria de almacenamiento utilizada en el proceso para almacenar la matriz A es mucho mayor en los esquemas que con la matriz densa, a diferencia de lo que sucedía para las matrices utilizadas para el producto matriz por vector. De esta manera el **AIJ** realiza más gasto de memoria y el que menos lo realiza es el **CSR mod**.

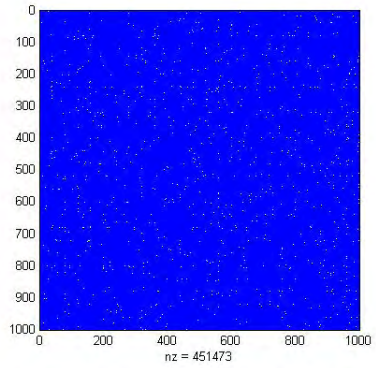
	Densa	AIJ	CSR	CSR MOD
Mem Alm (MB)	5.324	12.904	9.200	7.208
Tiempo Alm (seg)	1.155	44.898	33.529	1.241

Tabla 5.44: Resultados almacenamiento matriz aleatoria A.

2. **Matriz Aleatoria B.**

En la figura 5.45, el gráfico (a) y el gráfico (b) los elementos nulos están esparcidos en toda la matriz lo que no permite determinar si es esparza o no.

(a) gráfica global



(b) gráfica con zoom

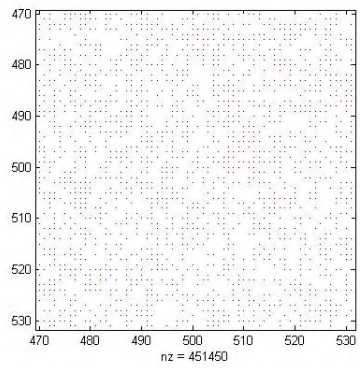


Tabla 5.45: Gráfico de esparcidad de la matriz

La tabla 5.46, índica que el tamaño de la matriz B es dos veces a la anterior y por el porcentaje de elementos nulos la matriz es esparza.

Filas	1000
Columnas	1000
nnz	451473
Porcentaje nnz	45.147 %

Tabla 5.46: Datos de la matriz aleatoria B.

En la tabla 5.47, igual que en el anterior caso se observa que los esquemas gastan más memoria que con la matriz densa, donde el que más consume es **AIJ** y el que menos utiliza es **CSR mod**. Sobre el tiempo de almacenamiento se observa que la matriz densa realiza este proceso en menor tiempo, siendo cercano el tiempo de **CSR mod** y el más lejano **AIJ**, por lo que se puede deducir de la ineficiencia de la búsqueda e inserción con este esquema, como se esperaba debido a que no tiene un apuntador de filas o columnas.

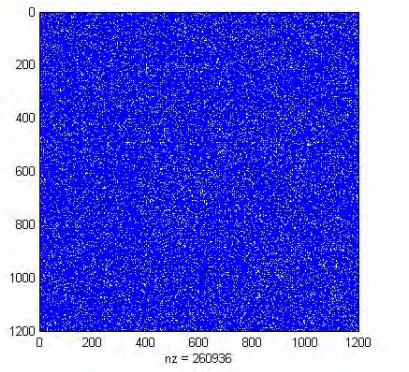
	Densa	**AIJ**	**CSR**	**CSR MOD**
Mem Alm (MB)	11.392	36.760	26.700	18.608
Tiempo Alm (seg)	4.546	703.231	533.775	5.138

Tabla 5.47: Resultados almacenamiento matriz aleatoria B.

3. **Matriz Aleatoria C.**

De igual manera que en la matrices A y C en la figura 5.48, la gráfica (a) y (b) no nos garantiza si la matriz es esparza o no.

(a) gráfica global



(b) gráfica con zoom

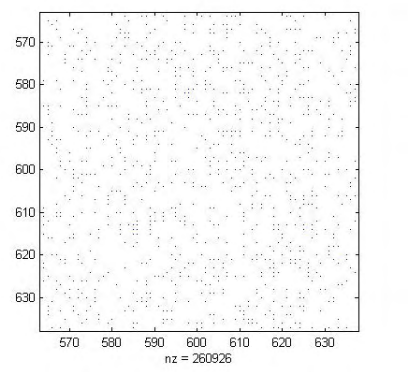


Tabla 5.48: Gráfico de esparcidad de la matriz C.

Por los datos de la tabla 5.49, la matriz C es un poco más grande que la anterior pero el porcentaje de elementos distintos de cero y el porcentaje de densidad es más pequeño.

filas	1200
Columnas	1200
nnz	260936
Porcentaje nnz	18.120 %

Tabla 5.49: Datos de la matriz aleatoria C.

A diferencia de los resultados con las matrices A y B, por la tabla 5.50, en el caso de la memoria, su valor es muy cercano, sin haber diferencias significativas. Para los tiempos de almacenamiento se sigue manteniendo el mismo comportamiento.

	Densa	**AIJ**	**CSR**	**CSR MOD**
Mem Alm (MB)	14.764	14.564	11.472	14.280
Tiempo Alm	2.210	232.107	177.806	2.366

Tabla 5.50: Resultados de almacenamiento matriz aleatoria C.

5.3.2. Resultados computador 2

4. **Matriz aleatoria D.**

   De la tabla 5.51, la matriz es de tamaño 30000×30000, con un porcentaje de nnz del $0.01\,\%$, es decir hay 90000 elementos no ceros.

   De la tabla 5.52, claramente se ve la diferencia en consumo memoria de la matriz densa que con los almacenamientos. El **CSR mod** es el que más consume memoria de estos esquemas. Para el caso del tiempo de almacenamiento la densa requiere más tiempo que los esquemas, pero las diferencias no son muy lejanas, ya que cerca de 4 segundos en relación con cerca de 1 segundo. Esto índica que si la matriz es muy esparcida se puede mejorar los resultados que con la matriz densa.

Filas	30000
Columnas	30000
nnz	90000
Porcentaje nnz	0.01 %

Tabla 5.51: Datos de matriz aleatoria D.

	Memoria (KB)			
Porcentaje nnz	**Densa**	**AIJ**	**CSR**	**CSR MOD**
Mem. Alm (MB)	6833.056	3.404	3.416	95.472
T. Alm (seg)	3.653	0.174	0.387	0.046

Tabla 5.52: Resultados almacenamiento matriz aleatoria D.

5. Matriz aleatoria E.

   Por los datos de la tabla 5.53, la matriz tiene igual tamaño de F pero el número de elementos nnz es mayor con un porcentaje de $0.07\,\%$.

Filas	30000
Columnas	30000
nnz	629885
Porcentaje nnz	0.07 %

Tabla 5.53: Datos matriz aleatoria E.

De la tabla 5.54, el comportamiento en la memoria es similar en la anterior, pero hay un aumento de este valor en **AIJ**, **CSR** y aun más en **CSR mod**. Para el caso de los tiempos, al aumentar el número nnz, se aumento este valor, ya que para el caso de la densa aumentó cerca de 1, para **CSR mod** el cambio fue alrededor de 2.5 segundos, mientras que con las estructuras **AIJ** y **CSR** el cambio es mayor, cuyo aumento fue de cerca de 380 y 299 segundos.

	Densa	**AIJ**	**CSR**	**CSR MOD**
Mem. Alm (MB)	6738.208	23.932	20.368	376.952
T. Alm (seg)	5.953	380.397	299.724	2.406

Tabla 5.54: Resultados almacenamiento matriz E

6. **Matrices aleatorias 1000 × 1000**

Se va a considerar matrices esparzas cuadradas de orden 1000, generadas aleatoriamente y con diferentes porcentajes de esparcidad. En las tablas 5.55 y 5.56, que se muestran a continuación se observa el tiempo y la memoria utilizada para almacenar estas matrices, para determinar la variación en estos items.

	Tiempo Alm. (seg)			
Porcentaje nnz	**Densa**	**AIJ**	**CSR**	**CSR MOD**
0.9936	0.042	0.125	0.11	0.036
4.8818	0.177	2.448	1.890	0.167
18.125	0.647	30.734	22.598	0.785
33.000	1.284	101.923	75.638	1.427
45.099	1.989	197.839	140.681	2.121
63	3.294	373.347	275.379	3.620

Tabla 5.55: Resultados de almacenamiento matrices aleatorias 1000 × 1000.

	Memoria Alm (MB)			
Porcentaje nnz	**Densa**	**AIJ**	**CSR**	**CSR MOD**
0.10	11.240	3.064	3.052	5.096
0.994	11.240	3.420	3.296	5.596
4.882	11.240	4.944	4.364	6.696
18.125	11.240	10.832	8.704	9.436
33.000	11.240	21.404	16.744	12.544
45.099	11.240	35.508	26.304	15.244
63	11.240	56.652	40.636	30.860

Tabla 5.56: Memoria de almacenamiento matrices aleatorias 1000×1000.

Observación 5.3. Como se aprecia en las tablas 5.55 y 5.56, al ir aumentando la densidad de los elementos nulos, los tiempos de almacenamiento y la memoria van aumentando considerablemente con **AIJ**, **CSR** y **CSR mod**. Pero en el caso de la matriz densa la memoria se mantiene constante y los tiempos de almacenamiento no aumentan tanto como en los demás. Por otro lado se observa que hasta el 4.8818 %, se puede ver que hay una pequeña eficiencia en los tiempos con el uso de algunos de los esquemas de almacenamiento, pero a partir del 18.125 % de porcentaje nnz, ya los tiempo en **AIJ** y **CSR** empiezan a ser muy ineficientes con respecto a la matriz densa y a **CSR mod**. Igualmente con la memoria sucede lo mismo, pero a partir del 33 % de porcentaje de elementos, donde se utiliza mayor uso de memoria con los esquemas.

Para la siguientes matrices aleatorias debido al tamaño de la matriz y a los resultados anteriores, se ha considerado únicamente los esquemas **CSR** y **CSR mod** con los que mejor resultados se han obtenido. También en esta parte se requiere inicializar con otro valor los vectores usados en los esquemas puesto que con la anterior estrategia se presenta saturación de memoria. Este valor en porcentaje se registra en la tabla de resultados (Val. inicial (%).) de las matrices a analizar.

7. **Matriz aleatoria F.**

   De la tabla 5.57, se ve que la matriz F es de mayor escala que las demás, con un valor $nnz =$ 699924, y un valor de densidad de 1.74 E-3 %.

	Matriz E
filas	200000
Columnas	2000000
nnz	699924
Porcentaje nnz	1.74E-3 %

Tabla 5.57: Datos de la matriz aleatoria F.

De la tabla 5.58, se observa un aumento de la memoria tanto en **CSR** como en **CSR mod**, pero sobre todo en **CSR mod** al tomar un valor inicial del 25 %, en los vectores de su esquema. Para el tiempo, se tarda más con **CSR** a pesar de utiliza menos espacio de memoria, mientras que con **CSR mod**, este valor sea menos de 3 segundos.

	CSR	CSR MOD
Mem. (MB)	29.216	432.132
T. Alm (seg)	597.989	2.656
Val. inicial. (%)	1E-3	25

Tabla 5.58: Resultados almacenamiento matriz aleatoria F.

8. **Matriz aleatoria** G

De la tabla 5.59, la matriz es de mayor tamaño que las anteriores y también en su número de elementos nnz, pero es muy esparza.

	Matriz G
filas	500000
Columnas	5000000
nnz	999860
Porcentaje nnz	2E-4 %

Tabla 5.59: Datos de la matriz aleatoria G.

De la tabla 5.60, no se observa un aumento tan drástico de la memoria con ambos esquemas pero si en el tiempo con **CSR**, debido a que tarda el triple de tiempo con respecto a la matriz anterior. En el caso de **CSR mod** el aumento en tiempo solo fue cerca de un segundo. De la misma tabla, los porcentajes utilizados para inicializar los vectores en los esquemas, se disminuyo el porcentaje en ambos casos. Cabe resaltar que para los esquemas mencionados se tomó varios porcentajes y en muchos casos se observaba saturación de memoria hasta encontrar los

valores con los cuales se pueda realizar este proceso.

	CSR	CSR MOD
Mem. (MB)	32.304	440.604
T. Alm (seg)	1664.515	3.729
Val. inicial. (%)	4E-4	10

Tabla 5.60: Resultados almacenamiento matriz G.

9. *Matrices aleatoria* $10^6 \times 10^6$.

   Ahora se considera matrices esparzas cuadradas de orden 1000000 generadas aleatoriamente y con diferentes porcentajes de esparcidad usando **CSR mod** con un valor inicial de 0.01 %, para cada caso. En las tablas 5.61 que se muestran a continuación se observa el tiempo y la memoria utilizada para almacenar estas matrices.

Almacenamiento con esquema CSR MOD			
nnz	**Porcentaje (%) nnz**	**Memoria (MB)**	**Tiempo (seg)**
3998765	3E-4	620.168	14.370
7994088	7E-4	851.676	25.245
15973612	1.5E-3	2255.123	64.834
35858605	3.5E-3	4555.032	169.550

Tabla 5.61: Resultados almacenamiento matrices aleatorias $10^6 \times 10^6$.

Observación 5.4. De la tabla 5.61, se puede ver que al ir aumentado la densidad, se aumenta la memoria y el tiempo de almacenamiento, sobre todo en el segundo. Además si se compara estas variables con algunas matrices de mayor tamaño y de mayor densidad analizadas en la primera sección del producto matriz por vector, resultan ser mucho más grandes.

10. *Productos matriz por vector con formato aleatorio.*

    Como se pudo apreciar, el tiempo y la memoria para el almacenamiento de una matriz esparza mediante una inserción de elementos aleatoria en la mayoría de los ejemplos que se mostraron resulta ser muy ineficiente con las implementaciones propuestas, pero se puede sacar provecho en situaciones de operaciones matriciales de alto costo, como es el producto matriz por vector como se observa en los siguientes ejemplos.

    Retomando la matriz D, cuyos datos era una matriz cuadrada de orden 30000 y un porcentaje nnz de 0.01 %. Al realizar 50 productos con esta matriz, mediante la tabla 5.62 se registran

los resultados de esta operación.

50 Productos	Densa	AIJ	CSR	CSR MOD
T. Alm (seg)	3.825	11.848	9.159	0.856
T. prods. (seg)	146.455	0.065	0.032	0.127
T. total (seg)	150.28	11.913	9.191	0.983

Tabla 5.62: Resultados 50 productos con la matriz aleatoria D.

Observación 5.5. Al leer la tabla 5.62, se puede notar que al realizar los 50 productos (*T. prods. (seg)*) con la matriz D, se tarda más la matriz densa frente a los esquemas y por medio del tiempo total, (T. total (seg)) se deduce que a pesar de los tiempos de almacenamiento se mantenga este resultado. Particularmente con **CSR mod** es donde se obtuvo los mejores resultados con un tiempo menor de 1 segundo en relación a **AIJ** y **CSR**, el cual fue de alrededor de 12 y 10 segundos aproximadamente.

De lo visto en los capítulos 3 y 4, se estudiaron algunos métodos para resolver sistemas lineales, como los métodos del gradiente que son métodos iterativos, la factorización LU y la factorización de Cholesky que son métodos directos.

En las siguientes dos secciones se muestran algunos resultados computacionales al resolver sistemas lineales. Para ello se han construido sistemas lineales $A\boldsymbol{x} = \boldsymbol{b}$, donde A es tomada de los recursos **MM** y **UF** y el vector independiente $\boldsymbol{b}$ es generado a partir de un vector aleatorio $\boldsymbol{c}$, tal que $A\boldsymbol{c} = \boldsymbol{b}$, esto con el fin de analizar el error de la solución del método con la solución real del sistema.

5.4. Métodos del Gradiente

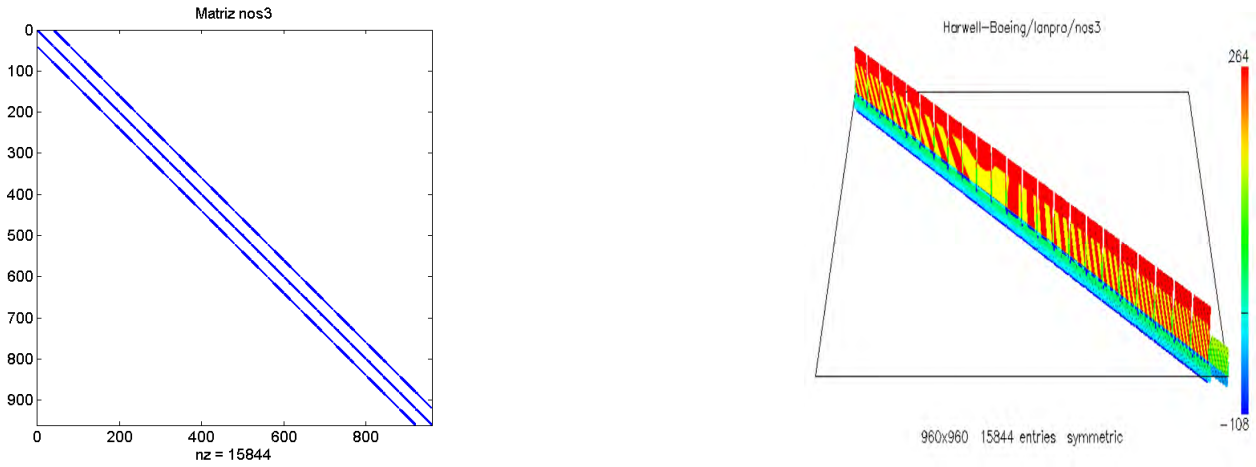
Los métodos del gradiente son uno de los métodos más usados para resolver sistemas lineales esparzas y su particularidad con los otros métodos iterativos que teóricamente converge a la solución exacta del sistema a lo más en n iteraciones, con n el orden de la matriz del sistema, cuando la matriz del sistema es S.D.P. Los métodos de gradiente a utilizar son: **CG**, **CG pre**, **BCG** ,**BCG pre**, **CGS** y **BICGSTAB**. También en algunos ejemplos se utilizan precondicionador de Jacobi explícito, para aplicarlos en **CGS** y **BICGSTAB**, para comparar las ventajas y desventajas de aplicar estos métodos con el sistema original y con el sistema precondicionado. A continuación se muestran algunos resultados. Se utilizarán los esquemas **CSR** y **CSV**, para aplicarlos en los métodos mencionados. Se escogieron estos dos métodos teniendo en cuenta el tiempo al realizar un producto matriz por vector y el espacio de memoria que utilizan.

5.4.1. Resultados computador 1

1. Matriz $A = nos3$ (**MM**)

   En la figura 5.19, se ve que en la gráfica (a) la matriz *nos3* parece ser una matriz de banda, esparza y estructurada. Respecto a la gráfica (b) las entradas de los elementos están entre -108 y 204 lo cual índica que son pequeños.

   En la tabla 5.63, nos índica que la matriz *nos3* es de tamaño de filas y columnas de 960, el número de elementos *nnz* es de 11800 y su porcentaje 1.29 %. Con respecto a la simetría es S.D.P, por lo que se puede aplicar teóricamente los métodos del gradiente. También de la misma tabla se encuentra el valor del condicionamiento de la matriz, que es cerca de 3723.59, la norma 2 de 680 y la norma de Frobenius de 1.4E+14.



(a) Gráfico de esparcidad

(b) Intervalo de las entradas de la matriz

Figura 5.19: Gráficos de la matriz *nos3*.

Filas	960
Columnas	960
nnz	11880
Porcentaje nnz	1.29 %
Simetría	S.D.P
Cond.	3723.5933
Norma 2	680
Norma de Frobenius	1.4E+14
Año	1983
Disciplina	Ing. Estructural

Tabla 5.63: Datos de la matriz *nos3* [4].

En las tablas 5.65 y 5.64, se encuentra información perteneciente de los resultados al resolver mediante los métodos del gradiente el sistema lineal con la matriz del sistema *nos3*, utilizando MATLAB y DEV-C++, respectivamente. Entre los resultados se observa que el número de iteraciones realizadas entre los métodos es menor que el orden de la matriz y que **CGS** converge en menos iteraciones, pero en el tiempo de convergencia es más rápido usar **CG** y **CG pre**. Además en el **BICGSTAB** se realiza más iteraciones al utilizar los esquemas que con la matriz densa.

Con respecto a los tiempos al utilizar esquemas y al utilizar la matriz densa, como se esperaba se tarda más tiempo aplicando la matriz densa. Debido a que los tiempos de **CSR** y **CSV** no superan ni siquiera 1 segundo, no hay diferencia notables. Otro caso a considerar es que en los resultados de MATLAB el tiempo para realizar los métodos con las diferentes estructuras es mayor que al realizarlo con DEV-C++.

Ahora al analizar el error de los métodos respecto al item de los residuos y el error, a pesar de que el error sobrepaso la tolerancia propuesta, con el residuo no sucede lo mismo y este resultado se lo puede relacionar con el valor del condicionamiento de la matriz del sistema el cual era de 3723.59.

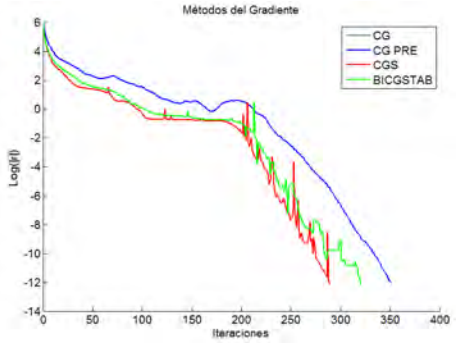
Met. Grad. (Esquemas)	Iter.	Residuo (|| ||$_2$)	Error (|| ||$_2$)	T. Densa (seg)	T. CSR (seg)	T. CSV (seg)
CG	351 (351)	8.51E-13 (8.557E-13)	2.59E-10 (5.355E-10)	2.47	0.06	0.073
CG pre	302 (302)	7.81E-13 (7.721E-13)	7.77E-11 (3.764E-10)	2.21	0.128	0.14
BCG	351 (351)	8.61E-13 (9.019E-13)	3.28E-10 (5.140E-10)	4.93	0.112	0.135
BCG pre	302 (302)	7.81E-13 (7.721E-13)	7.77E-11 (3.764E-10)	4.32	0.172	0.194
CGS	289 (289)	6.54E-13 (7.627E-13)	4.61E-10 (4.548E-10)	4.06	0.095	0.116
BICGSTAB	308 (320)	2.44E-13 (6.409E-13)	4.33E-10 (3.279E-10)	4.33	0.112	0.134

Tabla 5.64: Resultados DEV-C++ métodos del gradiente *nos3*.

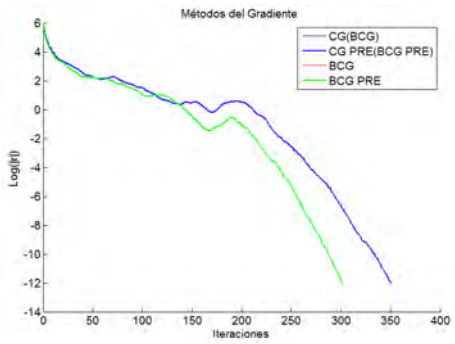
Met Grad (Esquemas)	Iter	Residuo (|| ||$_2$)	Error (|| ||$_2$)	T. Densa (seg)	T. CSR (seg)	T. CSV (seg)
CG	351 (351)	8.986E-13 (8.984-E13)	8.558E-10 (5.541E-11)	7.157	0.1872	0.203
CG pre	302 (302)	7.446E-13 (7.350E-13)	4.207E-11 (2.144E-10)	5.853	0.421	0.452
BCG	351 (350)	9.086E-13 (9.794-E13)	8.044E-11 (4.910E-11)	13.542	0.624	0.702
BCG pre	302 (302)	7.446E-13 (7.350E-13)	4.207E-11 (2.144E-10)	11.689	0.842	0.920
CGS	289 (292)	7.457E-13 (7.829E-13)	3.222E-10 (1.908E-10)	11.174	0.296	0.328
BICGSTAB	306 (317)	7.720-E13 (9.223E-13)	2.997E-10 (1.721E-10)	11.824	0.343	0.374

Tabla 5.65: Resultados MATLAB métodos del gradiente *nos3*.

En la figura 5.20, se muestran dos gráficas que nos permiten ver el comportamiento de la convergencia de los métodos del gradiente utilizados. En el eje de las abscisas, se encuentra el número de iteraciones y en el eje de las ordenadas el valor del residuo aplicado la norma 2 y posteriormente el logaritmo en base 10, esto con el fin de observar con mayor claridad el comportamiento de los residuos.



(a) Métodos sin precondicionar



(b) Métodos precondiconados

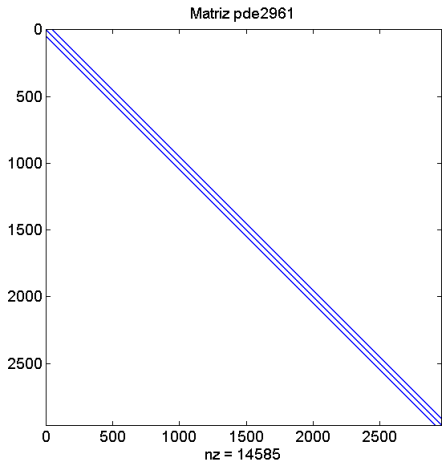
Figura 5.20: Gráficos de convergencia *nos3*.

De la figura 5.20, en la gráfica (a) se observa que la linea roja que representa el **CGS** converge en menos iteraciones, ya que termina la gráfica antes de 300 iteraciones en la tolerancia propuesta (-12 en las ordenadas) y la linea azul que es **CG** converge en mayor número de iteraciones y coincide con **BCG**, que es mayor a 350; mientras que en (b), se observa que la linea verde que representa el **BCG pre** coincide con el **CG pre** y converge en menor número de iteraciones, que es casi 300 iteraciones. Por tanto de las dos gráficas **CGS** convergen en menos iteraciones.

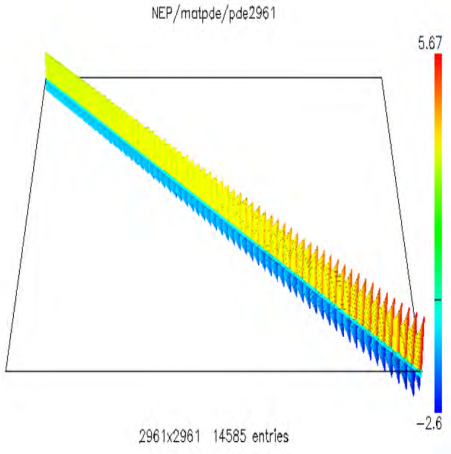
2. Matriz $A =$ *pde2961* (**MM**)

En la figura 5.21, gráfica (a) la matriz *pde2961* posee un patrón simétrico en sus elementos, muy cercanos a la diagonal principal. De la gráfica (b), el intervalo de los elementos nulos esta entre -2.6 y 5.87, por lo cual las entradas son pequeñas.

La tabla 5.66, índica que la matriz *pde2961* es de orden 2961, el número *nnz* es 14585, el porcentaje *nnz* es de 0.166 %. Para la simetría, nos índica que no es simétrica a pesar de que la matriz sea estructurada y por tanto no se garantiza la convergencia del método **CG** y **CG pre**. Para el caso del condicionamiento y el valor de las norma 2 y la norma de Frobenius son más pequeños relacionados con los de la matriz anterior.



(a) Gráfico de esparcidad



(b) Intervalo de las entradas de la matriz

Figura 5.21: Gráficos de la matriz *pde2961*.

Filas	2961
Columnas	2961
nnz	14585
Porcentaje nnz	0.166 %
Simetría	no simétrica
Cond.	642.493
Norma 2	10
Norma F	220
Disciplina	E.D.P

Tabla 5.66: Datos de la matriz *pde2961* [4].

En las tablas 5.67 y 5.68, se deduce que los métodos **CG** y **CG pre** no coinciden; el **BICGS-TAB** converge en menor número de iteraciones y más aún en menor tiempo que los demás; el error es más grande que el residuo y sobrepasa la tolerancia propuesta y en el caso de **CGS** el error es el más grande de todos y esto es debido al manejo de los residuos cuadráticos; para los tiempos de los métodos, se observa igualmente que el anterior mejores resultados con los esquemas de almacenamiento y lo realizado con DEV-C++.

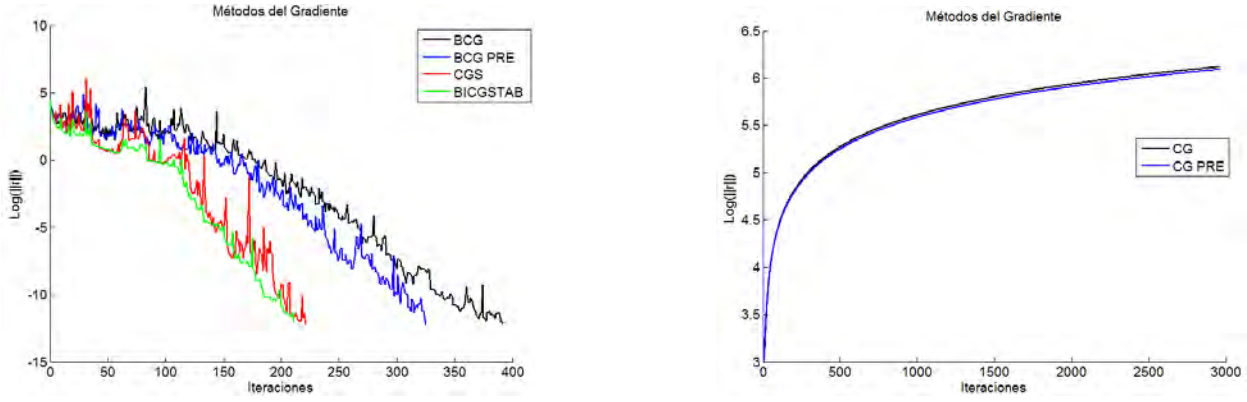
En la figura 5.22, en la gráfica(a) se observa que gráficamente la linea verde que representa el **BICGSTAB**, converge en menor número de iteraciones y la linea negra que representa el **BCG**, convergen en mayor número de iteraciones. Para el caso de **CG** y **CG pre** en la gráfica (b) se observa que divergen al irse alejando del eje de las abscisas.

Met. Grad.	Iter.	Residuo. ($\|\|\ \|\|_2$)	Error ($\|\|\ \|\|_2$)	T. Densa (seg)	T. CSR (seg)	T. CSV (seg)
CG	2961	1.314E+6	1.908E+6	190.333	0.931	1.14
CG pre	2961	1.234E+6	1.862E+6	190.667	1.177	1.393
BCG	392	6.714E-13	5.308E-11	50.767	0.218	0.400
BCG pre	325	5.407E-13	3.539E-11	42.2	0.437	0.494
CGS	221	5.933E-13	1.704E-9	28.567	0.130	0.166
BICGSTAB	211	9.468E-13	1.547E-11	27.333	0.124	0.135

Tabla 5.67: Resultados DEV-C++ métodos del gradiente *pde2961*.

Met. Grad.	Iter.	Residuo ($\|\|\ \|\|_2$)	Error ($\|\|\ \|\|_2$)	T. Densa (seg)	T. CSR (seg)	T. CSV (seg)
CG	2961	1.314E+6	1.908E+6	810.134	3.494	3.136
CG pre	2961	1.235E+6	1.862E+6	809.626	4.633	4.384
BCG	399	7.805E-13	1.301E-11	217.414	1.357	1.279
BCG pre	319	7.411E-13	4.411E-11	176.009	1.840	1.669
CGS	215	9.842E-13	1.290E-9	122.910	0.483	0.515
BICGSTAB	196	5.826E-13	2.015E-11	111.365	0.468	0.437

Tabla 5.68: Resultados MATLAB métodos del gradiente *pde2961*.



(a) Métodos gradiente conjugado

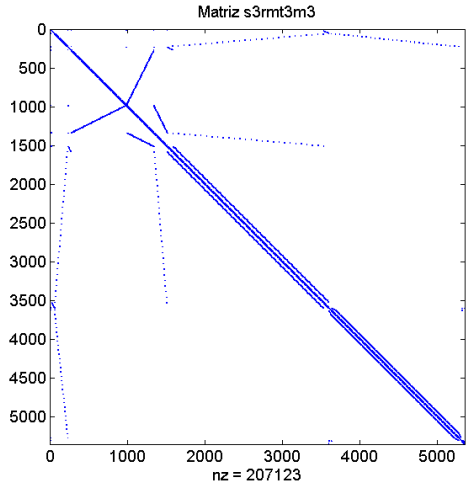
(b) CG y CG pre

Figura 5.22: Gráficos de convergencia *pde2961*.

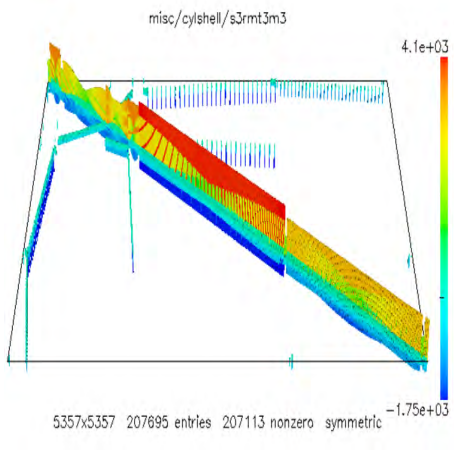
Para los siguientes sistemas, no se considera la matriz densa, debido a que los resultados no son eficiente y además se trabajará con sistemas mucho más grandes que los anteriores, lo que implica que el tiempo para realizar los tiempos sea muy costoso.

3. Matriz A = *s3rmt3m3* (**MM**)

En la figura 5.23, en la gráfica (a) se puede observar que las entradas, poseen un comportamiento simétrico y algunos elementos diferentes de cero están alejados de la diagonal principal. En la gráfica (b) se puede ver que el intervalo de los elementos no nulos es muy grande y se aprecia que hay entradas de color azul y color rojo, que son valores muy grandes.



(a) Gráfico de esparcidad



(b) Intervalo de las entradas de la matriz

Figura 5.23: Gráficos de la matriz *s3rmt3m3*.

En la tabla 3, se puede ver el sistema es es S.D.P, de orden 5357 y el condicionamiento es de tamaño 2.6E+10, por lo que el sistema es muy mal condicionado.

De la tablas 5.70 y 5.71 se observa que ningún método converge y se fijó una iteración máxima de 8000, lo cuál no cumple con la teoría que garantiza la convergencia a lo más en n pasos.

Respecto a los tiempos al realizar los métodos con los almacenamientos, igualmente que en los anteriores resultados en C++ y en **CSR** es donde se realiza más rápido estos métodos.

Filas	5357
Columnas	5357
nnz	203869
Porcentaje nnz	0.71 %
Simetría	S.D.P
Cond.	2.6E+10
Norma 2	9600
Norma de Frobenius	160000
Año	1997
Disciplina	Estructuras Mecánicas

Tabla 5.69: Datos de la matriz *s3rmt3m3* [4].

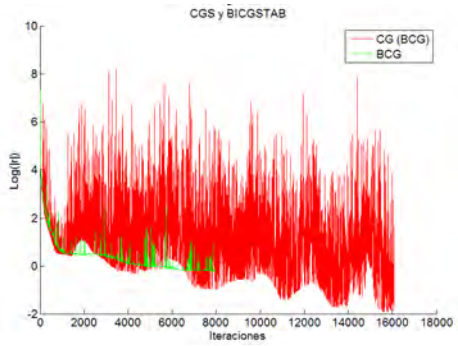
De la figura 5.24, se observa que ninguno de los métodos converge y que la linea roja, que representa el método **CGS** tiene un comportamiento muy inestable. Sin embargo se observa una tendencia de convergencia de los métodos **BICGSTAB**, **CG pre**, **BCG pre**, **CG** y **BCG**. Otra observación de estas gráficas es que **CG** y **BCG** no coinciden y por ende tampoco sucede con **CG pre** y **BCG pre**.

Met. Grad.	**Iter.**	**Residuo** ($\|\|\ \|\|_2$)	**Error** ($\|\|\ \|\|_2$)	**T. CSR (seg)**	**T. CSV (seg)**
CG	8000	4.2	806	17.067	23.1
CG pre	8000	0.32	619	18.667	24.133
BCG	8000	6.91	806	34.333	45
BCG pre	8000	0.32	619	36.033	46.6
CGS	8000	0.295	907	33.467	44.233
BICGSTAB	8000	0.612	1.89E+3	34.433	45.3

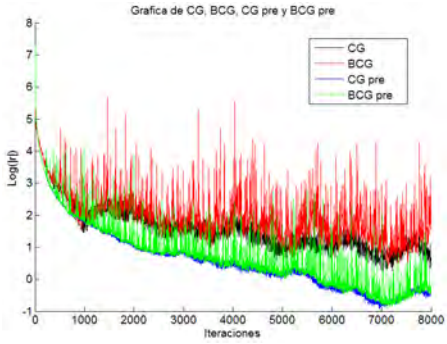
Tabla 5.70: Resultados métodos del gradiente C++ *s3rmt3m3*.

Met. Gradiente	**iter**	**Residuo** ($\|\|\ \|\|_2$)	**Error** ($\|\|\ \|\|_2$)	**Tiempo (CSR)**	**Tiempo (CSV)**
CG	8000	10.481	805.209	97.75	127.079
CG pre	8000	0.346	618.269	111.443	138.191
BCG	8000	13.732	915.437	203.050	254.885
BCG pre	8000	0.580	626.590	205.572	267.557
CGS	8000	389.371	806.510	188.571	251.380
BICGSTAB	8000	0.641	1.939E+3	182.661	255.623

Tabla 5.71: Resultados métodos del gradiente MATLAB *s3rmt3m3*.



(a) Gráfico CGS y BICGSTAB



(b) Gráfico CG, BCG, CG pre y BCG pre

Figura 5.24: Gráficos de convergencia *s3rmt3m3*

5.4.2. Resultados computador 2

Para los siguientes pruebas, se realizaron en el computador 2, con DEV-C++, para abordar sistemas más grandes que los anteriores y aplicar el sistema precondicionado con Jacobi en los métodos de **CGS** y de **BICGSTAB**, para matrices que están muy mal condicionadas, como se puede ver en los resultados anteriores.

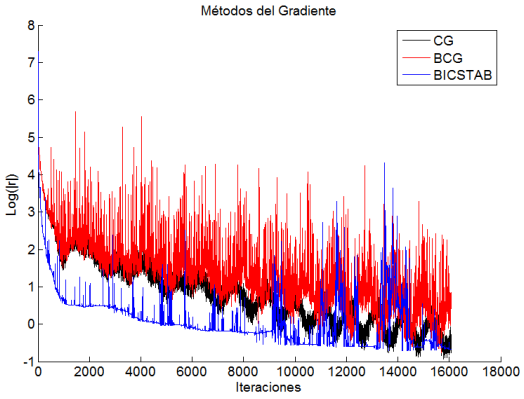
Como se pudo notar en el caso de la matriz *s3rmt3m3*, mediante la figura 5.24, puede existir la convergencia con los métodos **CG pre** y **BCG pre**, entonces realizando nuevamente las pruebas con esta matriz pero con iteraciones máximas más grandes (*iter max*) se obtiene lo siguiente.

En la tabla 5.72, se dan a conocer los resultados a realizar los métodos del gradiente fijando una iteración máxima de 16071, es decir tres veces el orden de la matriz. A pesar de esto, no convergen los métodos y además de haber precondicionado el sistema y aplicarlo en **CGS pre** y **BICGSTAB pre**. Sin embargo se observa que en **CGS pre** y **BCG pre** el residuo disminuye y el error también considerablemente.

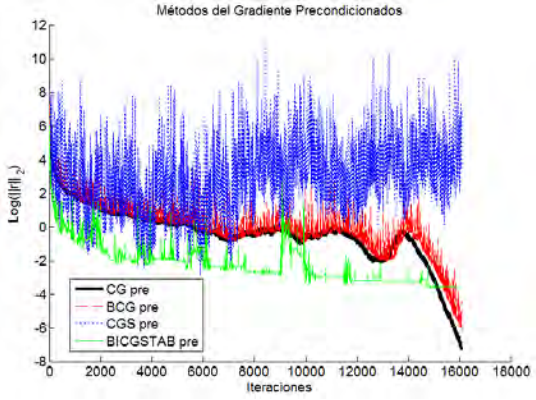
La figura 5.25, muestra que los métodos **CG**, **BCG** y **BICGSTAB** no han cambiado el comportamiento que se había presentado anteriormente. Para los métodos **CG Spre** y **BICGSTAB pre** se observa que no convergen; mientras que los métodos **CG pre** y **BCG pre** muestran un comportamiento de convergencia y se acercan a la tolerancia propuesta.

Met. Grad.	Iter	Residuo ($\|\| \cdot \|\|_2$)	Error ($\|\| \cdot \|\|_2$)	T. CSR (seg)	T. CSV (seg)
CG	16071	0.548	404.21	9.745	13.116
BCG	16071	4.598	421.02	19.595	25.903
CGS	16071	0.628	510.25	19.331	25.973
BICGSTAB	16071	0.217	1269.8	19.751	26.406
CG pre	16071	5.555E-8	1.514E-4	10.493	13.896
BCG pre	16071	4.033E-5	1.336E-4	20.335	26.685
CGS pre	16071	6191.6	24006	66.576	89.82
BICGSTAB pre	16071	2.573E-4	733.94	68.375	91.677

Tabla 5.72: Resultados métodos del gradiente *s3rmt3m3* con $itermax = 3n$.



(a) Gráfico métodos del gradiente



(b) Gráfico métodos precondicionados

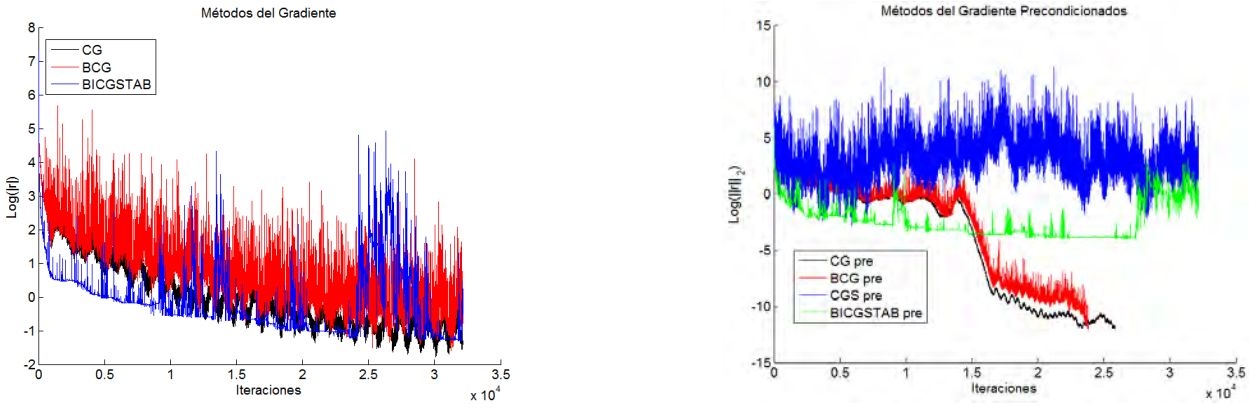
Figura 5.25: Gráficos de convergencia s3rmt3m3 con $itermax = 3n$.

En la tabla 5.73, se observa que al fijar un número de iteraciones máximas en los métodos de 6 veces al orden de la matriz del sistema, convergen los métodos **CG pre** y **BCG pre**, pero el error es muy grande, por lo cual se deduce del condicionamiento del sistema, de la cantidad de iteraciones que se han realizado, y de los que han traído por errores de redondeo, debido a las grandes entradas de la matriz. Respecto al tiempo de los métodos el **CG pre**, lo realiza más rápido usando el esquema **CSR**.

De la figura 5.23, gráfica (b) se confirma que los métodos **CG pre** y **BCG pre** convergen, dado que sobrepasan al valor -12 en el eje de la ordenadas, que nos representa la tolerancia del método.

Met. Grad.	Iter	Residuo ($\|\|\cdot\|\|_2$)	Error ($\|\|\cdot\|\|_2$)	T. CSR (seg)	T. CSV (seg)
CG	32142	0.0461	290.95	19.602	26.152
BCG	32142	0.25233	306.25	38.608	51.673
CGS	32142	1.082E+10	3.823E+10	38.960	52.788
BICGSTAB	32142	0.51933	2555.1	39.683	53.781
CG pre	25838	9.8262E-13	1.514E-4	16.761	22.052
BCG pre	23816	8.168E-13	1.331E-4	29.656	39.736
CGS pre	32142	916.43	420.12	131.533	177.95
BICGSTAB pre	32142	0.147	82112	135.29	181.95

Tabla 5.73: Resultados métodos del gradiente *s3rmt3m3* con $itermax = 6n$.



(a) Gráfico métodos del gradiente

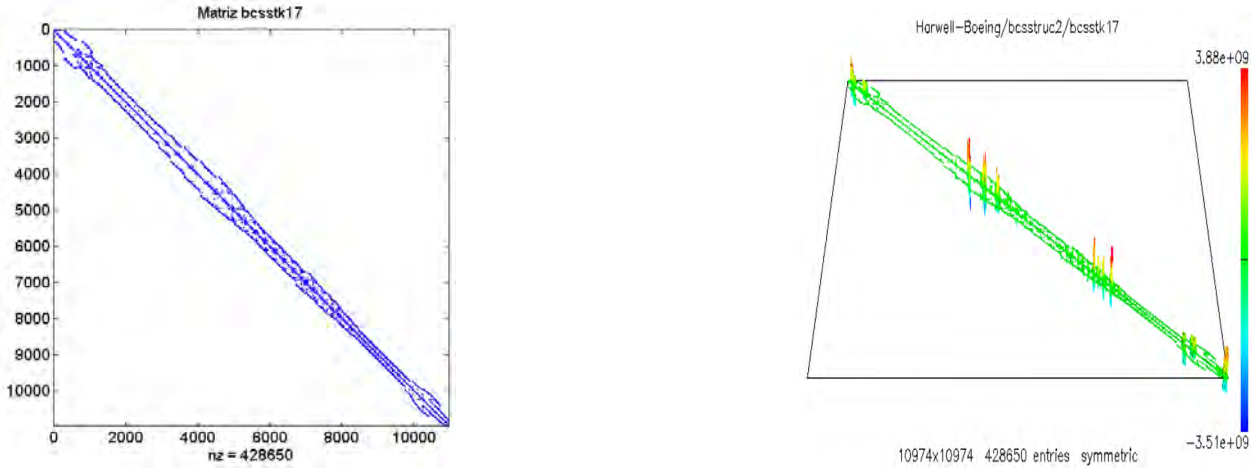
(b) Gráfico métodos precondicionados

Figura 5.26: Gráficos de convergencia *s3rmt3m3* con $itermax = 6n$.

4. Matriz A = *bcsstk17* (**MM**)

De la figura 5.27, en la gráfica (a) se ve que los elementos poseen un patrón simétrico y están muy cercano en la diagonal. Para el caso de (b) el intervalo de los elementos no nulos es muy grande, sobre todo los elementos en la diagonal que son cercanos a cero.

De la tabla 5.74, la matriz es de orden superior a 10^4, es S.D.P y el condicionamiento no es muy grande como en los otros resultados.



(a) Gráfico de esparcidad

(b) Intervalo de las entradas de la matriz

Figura 5.27: Gráficos de la matriz *bcsstk17*.

Filas	10974
Columnas	10974
nnz	428650
Porcentaje nnz	0.356 %
Simetría	S.D.P
Cond.	65
Norma 2	1.3e+10
Norma de Frobenius	4.8e+10
Año	1985
Disciplina	Análisis estático en Ing. estructural.

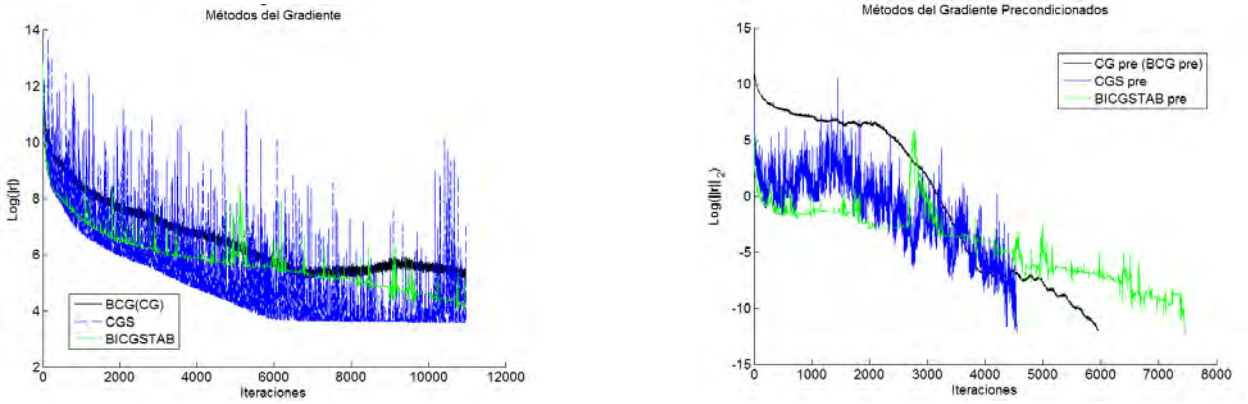
Tabla 5.74: Datos de la matriz *bcsstk17* [4].

En la tabla 5.75, se observa que únicamente converge los métodos precondicionandos, en donde **CGS pre** lo realiza en menos iteraciones y **BICGSTAB** realiza más iteraciones, pero **CG pre** lo realiza en menor tiempo con **CSR**. Una de las particularidades de aplicar **CGS pre** y **BICGSTAB pre** es que el error obtenido es muy grande comparado con el residuo obtenido.

De la figura 5.28 gráficamente se observa que la linea azul que representa el método **BCG pre** y **CG pre** converge, mientras que los otros no y en particular la linea roja, que representa el **CGS**, presenta un comportamiento muy inestable.

Met. Gradiente	Iter	Residuo ($\|\| \cdot \|\|_2$)	Error ($\|\| \cdot \|\|_2$)	T. CSR (seg)	T. CSV (seg)
CG	10974	2.5034E+5	2850.1	14.115	19.391
BCG	10974	2.334E+5	2850	28.401	37.881
CGS	10974	32460	2875	27.923	38.360
BICGSTAB	10974	15347	2881.9	28.115	39.610
CG pre	5959	9.751E-13	2.166E-8	8.637	11.444
BCG pre	5959	9.751E-13	2.166E-8	16.757	21.865
CGS pre	4544	5.874E-13	7.817E-3	13.469	17.229
BICGSTAB pre	7450	4.265E-13	6.5E-2	22.068	28.236

Tabla 5.75: Resultados métodos del gradiente *bcsstk17*.



(a) Gráfico métodos del gradiente

(b) Gráfico métodos precondicionados

Figura 5.28: Gráficos de convergencia *bcsstk17*.

5. Matriz *thermomech_dM*

En la figura 5.29, se observa un comportamiento muy particular de los elementos de la matriz, por lo que podría considerarse estructurada.

En la tabla 5.76, se puede ver que el orden es mayor que 10^4., es S.D.P lo que implica poder utilizar todos los métodos del gradiente abordados. Además por los valores mínimos y máximos se observa que las entradas de la matriz son pequeñas.

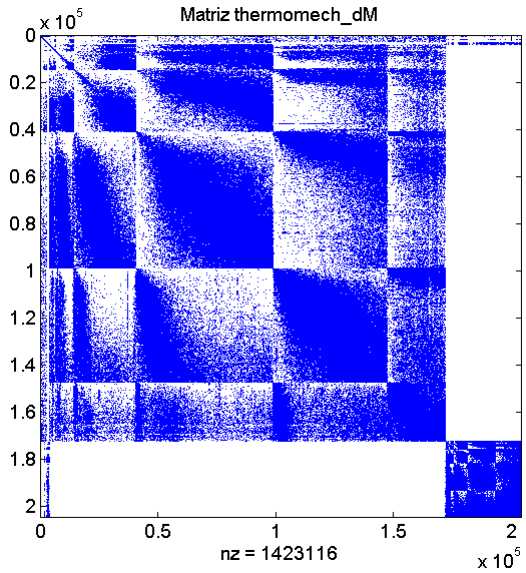


Figura 5.29: Gráfico de esparcidad *thermodech_dM* .

Filas	204316
Columnas	204316
nnz	813716
Porcentaje nnz	1.9E-3 %
V. máx.	4.251E-5
V. mín.	2.316E-7
Simetría	S.D.P
Año	2009
Disciplina	Problema térmico

Tabla 5.76: Datos matriz *thermomech_dM* [33].

La tabla 5.77, índica que los métodos convergen rápidamente a pesar de que se haya aumentado el tamaño de la matriz, como también el número de elementos no nulos, lo cual muestra que a mayor tamaño no implica necesariamente que los métodos del gradiente sean mas propensos a problemas de convergencia. Entre los resultados destacados de esta tabla está que todos los métodos convergen y en particular en menos iteraciones **CG pre**, sin embargo el que menos se tarda fue el **CG** con **CSR** a pesar de realizar mas iteraciones. Además de estos resultados los métodos **CGS pre** y **BICGSTAB pre**.

En la figura 5.30, gráfica (a), se detecta que gráficamente con **CGS** realza menos iteraciones y los que más realiza son **CG** y **BCG**, los cuales coinciden. Respecto a la gráfica (b) el que

menos iteraciones realiza es **CGS pre** y los que realizan más son **CG pre** y **BCG pre** los cuales coinciden.

Met. Gradiente	Iter	Residuo ($\|\|\cdot\|\|_2$)	Error ($\|\|\cdot\|\|_2$)	T. CSR (seg)	T. CSV (seg)
CG	101	7.950E-13	2.165E-7	1.052	1.084
CG pre	25	7.547E-13	7.095E-8	2.953	2.953
BCG	100	9.989E-13	2.765E-7	4.870	5.005
BCG pre	25	7.547E-13	7.095E-8	5.964	5.995
CGS	55	7.188E-13	7.764E-7	1.104	1.146
BICGSTAB	67	4.219E-13	3.554E-7	1.521	1.547
CGS pre	19	2.165E-13	1.718E-11	4.84	4.886
BICGSTAB pre	24	2.014E-13	2.319E-11	4.980	5.038

Tabla 5.77: Resultados métodos del gradiente *thermomech_dM*.

(a) Gráfico métodos del gradiente

(b) Gráfico métodos precondicionados

Figura 5.30: Gráficos de convergencia *thermomech_dM*.

6. Matriz *A=trans5*

De la figura 5.31, la matriz posee un cierto patrón en sus elementos.

De la tabla 5.78, el tamaño del sistema no es mayor en relación al anterior, ni tampoco la densidad, la diferencia es que el sistema no es simétrico y por los valores máximos y mínimos nos permite ver que las entradas no son demasiado grandes.

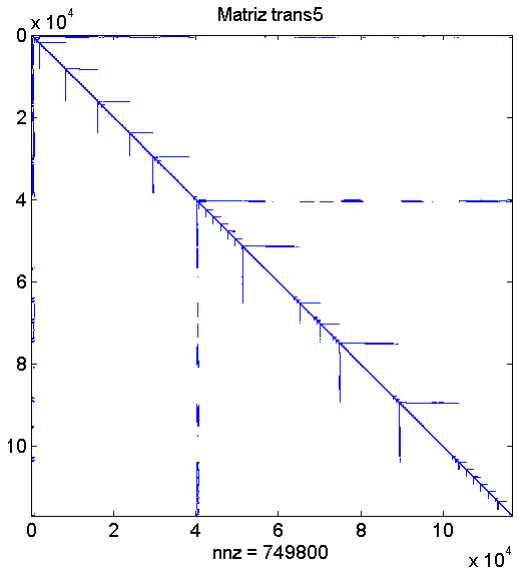


Figura 5.31: Gráfico de esparcidad *trans5*.

Filas	116835
Columnas	116835
nnz	749800
Porcentaje nnz	5.492E-3 %
V. máx.	11287.71
V. mín.	-98.361
Simetría	No simétrica
Año	2005
Disciplina	Problema simulación de circuitos

Tabla 5.78: Datos matriz *trans5* [33].

De la tabla 5.79, se ve la convergencia de manera muy rápida al aplicar los métodos precondicionados **BCG pre**, **CGS pre** y **BICGSTAB pre**, donde se tardan en menor tiempo, al aplicar los dos últimos con **CSR**. En términos de iteraciones se realiza menos usando **CGS pre** pero su valor de error es más grande de los otros.

De la tabla 5.32, se verifica lo que la tabla de resultados arrojaba, en donde **CGS pre** converge más rápido, pero su comportamiento es más inestable que los otros de hay el error más grande que se obtuvo respecto a los otros métodos.

Met. Grad.	Iter.	Residuo ($\|\|\cdot\|\|_2$)	Error ($\|\|\cdot\|\|_2$)	T. CSR (seg)	T. CSV (seg)
BCG pre	681	7.55E-13	2.544E-7	8.188	8.86
CGS pre	414	6.244E-13	2.467E-5	5.943	6.339
BICGSTAB pre	484	7.339E-13	1.701E-7	7.049	7.589

Tabla 5.79: Resultados métodos del gradiente precondicionados *trans5*.

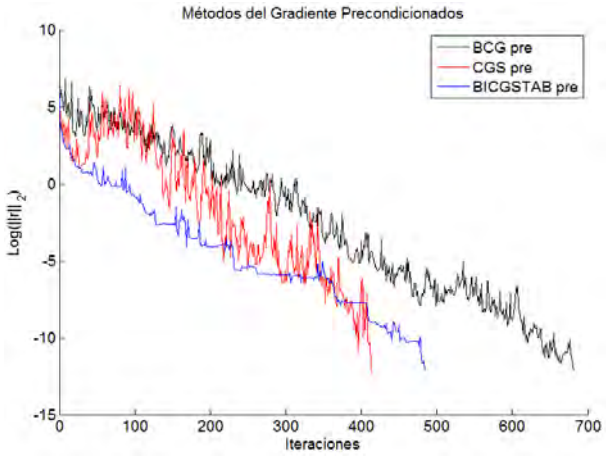


Figura 5.32: Gráfico de convergencia *trans5*.

7. Matriz $A = ASIC\_320ks$

De la figura 5.33, se observa un cierto patrón simétrico de los elementos, distribuidos alrededor de la matriz.

De la tabla 5.80, la matriz es cerca del triple del tamaño de la anterior y su densidad es mayor. La matriz del sistema es no simétrica y por los valores máximos y mínimos, se puede concluir que pueden existir entradas grandes.

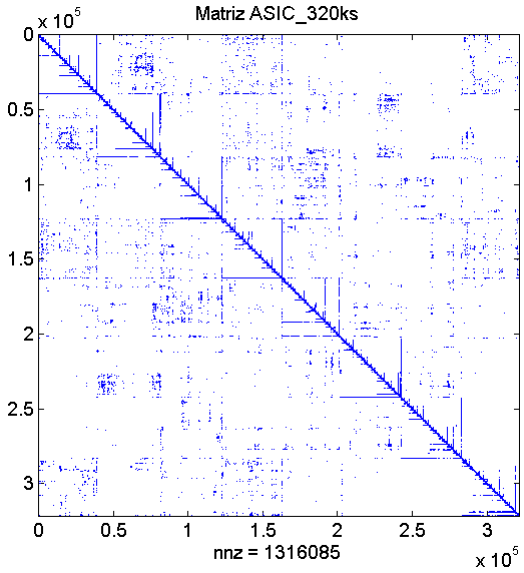


Figura 5.33: Gráfico de esparcidad *ASIC_320ks*.

Filas	321671
Columnas	321671
nnz	1316085
Porcentaje nnz	1.272E-3 %
V. máx.	1000000.7604
V. mín.	-1000000
Simetría	No simétrica
Año	2006
Disciplina	Problema simulación de circuitos

Tabla 5.80: Datos de la matriz *ASIC_320ks* [33].

De la tabla 5.81, se observa que se da una rápida convergencia al aplicar **BCG** y **BICGSTAB** con un menor número de iteraciones y tiempo con el primero. Sin embargo es muy grande y se aleja de la solución real del sistema.

De la tabla 5.81, se verifica la rápida convergencia de **BCG**, contra **BICGSTAB**, que a pesar que se haya creado para mejorar el **BCG** no se haya obtenido mejore resultados.

Met. Grad.	Iter.	Residuo ($\|\| \cdot \|\|_2$)	Error ($\|\| \cdot \|\|_2$)	T. CSR (seg)	T. CSV (seg)
BCG	469	9.551E-13	4457.9	12.126	11.834
BICGSTAB	6974	5.942E-13	4457.9	122.72	117.023

Tabla 5.81: Resultados métodos del gradiente *ASIC_320ks*

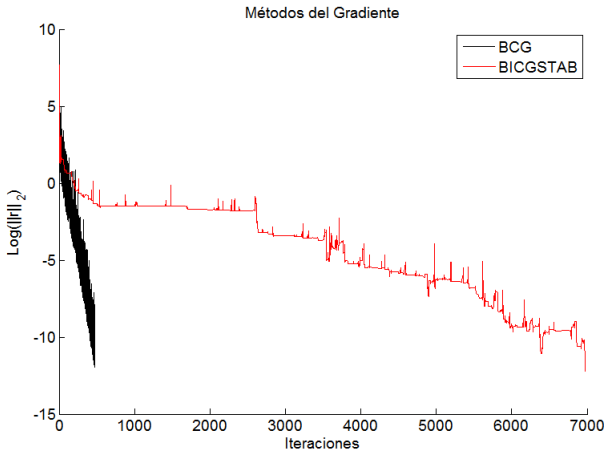


Figura 5.34: Gráfico de convergencia *ASIC_320ks*

8. Matriz $A =$ *raefsky4*

De la figura 5.35, la matriz es estructurada y se concentran los elementos no cero en su mayoría en la diagonal.

De la tabla 5.82, la matriz es más pequeña con la anterior, pero con la diferencia de que es más densa. Sobre la simetría es S.D.P y el valor de condicionamiento que es de 3.130E+13, que nos índica que es muy mal condicionada y posiblemente haya problemas con la convergencia.

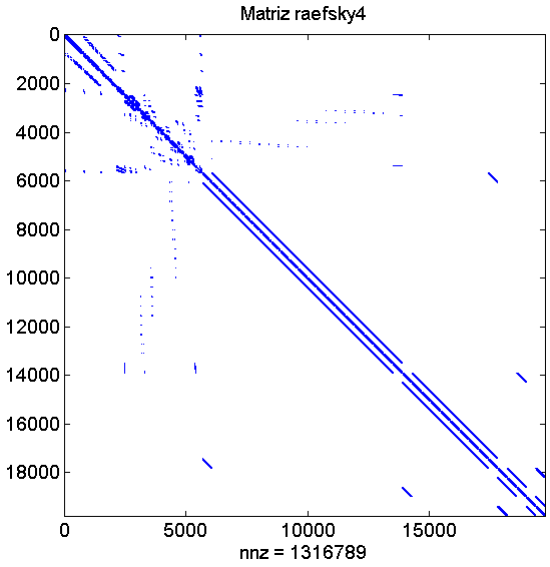


Figura 5.35: Gráfico de esparcidad *raefsky4*.

Filas	19,779
Columnas	19,779
nnz	1,316,789
Porcentaje nnz	0.336 %
V. máx.	1.569E+11
V. mín.	-3.458E+10
Simetría	S.D.P
Cond.	3.130E+13
Año	1993
Disciplina	Problema estructural

Tabla 5.82: Datos matriz *raefsky4* [33].

De la tabla 5.93, Se muestran los resultados al aplicar los métodos precondicionados y se concluye que no hay convergencia en el número de iteraciones que asegura la teoría. Al fijar una iteración máxima de $8n$, con **CGS pre** y **BICSTAB pre**, el error y el residuo son muy grandes mientras que con **CG pre**. Además en **CG pre** se muestra que al doblar el número de iteraciones fijas, no se muestra un cambio considerable en la convergencia.

De la tabla 5.36, se observa un inestable comportamiento con **CGS pre** y **BICGSTAB pre**. Mientras que con **CG pre**, se observa que va convergiendo de forma muy lenta, pero también inestable.

Met. Grad.	Iter.	Residuo ($\|\| \cdot \|\|_2$)	Error ($\|\| \cdot \|\|_2$)	T. CSR (seg)	T. CSV (seg)
CGS pre	158232	235.85	9758.4	923.048	1219.739
BICGSTAB pre	158232	8.661E+10	1.812E+16	945.237	1240.501
CG pre	158232	3.460E-3	8.533E-4	642.323	839.933
CG pre	316464	1.605E-4	8.524E-4	1288.211	1675.018

Tabla 5.83: Resultados métodos del gradiente precondicionados *raefsky4*.

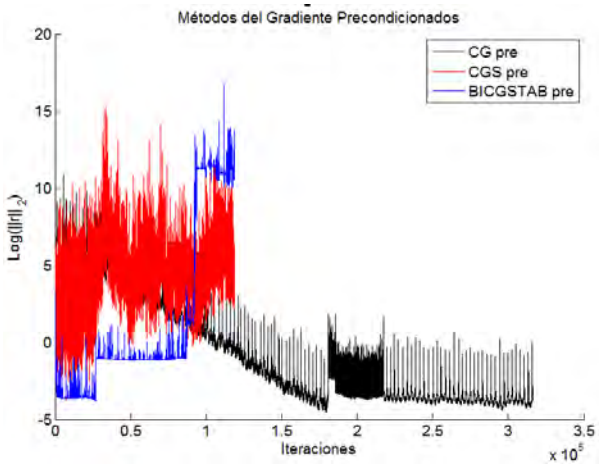


Figura 5.36: Gráfico de convergencia *raefsky4*

5.5. Métodos directos

Como se estudió en el capítulo 3, los métodos directos, obtienen teóricamente la solución exacta de un sistema lineal en un número exacto de pasos. En este apartado se pretende mostrar algunos resultados, al aplicar la factorización LU (Fact_LU.), la eliminación Gaussiana (Elim. Gauss.) y la factorización de Cholesky (Fact_Cho.). En estos métodos se realizar usando tanto la matriz densa como los esquemas de almacenamiento **AIJ**, **CSR** y **CSR mod**.

En los siguientes resultados se analizan las matrices utilizadas para los métodos del gradiente, para comparar ambos métodos, en tiempo de computo, el error y el residuo de los sistemas lineales.

5.5.1. Resultados computador 1

1. Matriz *A=nos3* (**MM**)

Retomando datos anteriores, la matriz *nos3* era una matriz cuadrada de orden 960, S.D.P, el número de elementos no ceros es de 11880 con un porcentaje de 1.289 % de entradas de la matriz y un valor de condicionamiento de 3723.59.

Dado que *nos3* es S.D.P se garantiza que es posible aplicar la Factorización de Cholesky y por tanto se adiciona este método a la tabla de resultados.

En la tabla 5.84, se muestran los resultados al utilizar la factorización LU.

	Densa	**AIJ**	**CSR**	**CSR mod**
Fact_LU (seg)	2.622	19.797	6.595	2.123
Sol Ax (seg)	0.011	0.341	0.047	0.047
T. total (seg)	2.633	20.138	6.642	2.17
Residuo $\|\cdot\|_2$	2.659E-12	5.633E-11	5.633E-11	5.633E-11
Error $\|\cdot\|_2$	8.749E-11	3.155E-10	3.155E-10	3.155E-10

Tabla 5.84: Resultados factorización LU *nos3*.

En la tabla 5.85, se muestran los resultados al utilizar el método de eliminación Gaussiana.

	T. Densa (seg)	**T. AIJ (seg)**	**T. CSR (seg)**	**T. CSR MOD (seg)**
Elim. Gauss. (seg)	2.491	19.547	6.833	2.121
Sol Ax (seg)	0.005	0.167	0.016	0.016
T. total (seg)	2.496	19.714	6.849	2.137
Residuo $\|\cdot\|_2$	3.088E-11	3.522E-11	3.522E-11	3.522E-11
Error $\|\cdot\|_2$	5.226E-11	6.246E-11	6.246E-11	6.246E-11

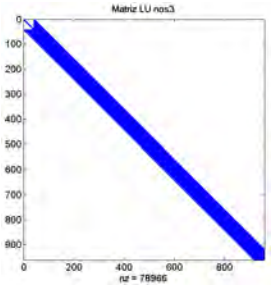
Tabla 5.85: Resultados eliminación Gaussiana *nos3*.

En la tabla 5.85, se muestran los resultados al utilizar la factorización de Cholesky.

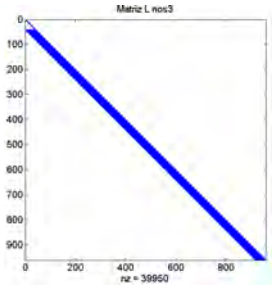
	Densa	AIJ	CSR	CSR MOD
Fact_Cho. (seg)	1.029	1.208	1.052	1.048
Sol Ax. (seg)	0.010	0.009	0.010	0.01
T. total. (seg)	1.039	1.217	1.062	1.058
Residuo $\|\cdot\|_2$	1.707E-11	1.632E-11	1.632E-11	1.632E-11
Error $\|\cdot\|_2$	1.278E-11	3.104E-11	3.104E-11	3.104E-11

Tabla 5.86: Resultados factorización de Cholesky nos3.

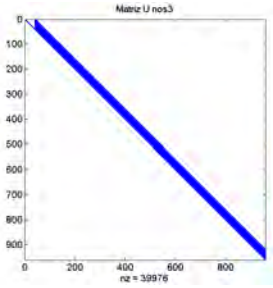
En la figura 5.37, se muestran los gráficos de esparcidad de la matriz L, la matriz U y la matriz LU, esta última es la matriz compacta de la factorización LU, es decir se guarda en una sola matriz la información de matriz L y la información de la matriz U.



(a) LU, nnz = 78,966.



(b) U, nnz = 39,250.



(c) L, nnz = 39,976.

Figura 5.37: Gráficos de esparcidad, matrices LU, L y U *nos3*.

Observación 5.6. Al analizar las tablas 5.84, 5.85 y 5.86 y la figura 5.37, se pudo observar lo siguiente:

- De la tabla 5.84, se puede notar que la matriz densa tarda menor tiempo tanto en la factorización LU de la matriz, como la solución de los sistemas triangulares y por ende en el tiempo total. Respecto a los tiempos entre los esquemas, se obtiene que **AIJ** se tarda más en los dos procesos y **CSR mod** se tarda menos en la factorización LU pero en el tiempo de la solución de los sistemas triangulares no hay diferencias con **CSR**.
- De la tabla 5.85 se obtiene los mismos resultados que con los de la factorización LU, pero con la diferencia que se tarda menos en los dos procesos para encontrar la solución. Además en el error y los residuos los valores son más pequeños que con los de la factorización LU.
- De la tabla 5.86 se observa ampliamente las diferencias con los dos métodos anteriores, en el tiempo para encontrar la matriz para solucionar el sistema triangular, puesto que

tarde menos de 2 segundos con la densa y los esquemas de almacenamiento. En este caso no hay diferencias notables entre los métodos en ambos procesos. En el caso del error y el residuo es más pequeño que los demás y se acerca mejor a la tolerancia propuesta.

- De la figura 5.37, se puede ver que las gráficas (a), (b) y (c) asociadas a las matrices L, U y LU respectivamente, siguen siendo esparzas y esto se comprueba también los valores nnz de estas matrices. El porcentaje de densidad de la matriz LU, la matriz L y la matriz U es de 8,56 %, 4.25 % y 4.34, respectivamente. Estos nos indican el aumento de elementos no ceros.

2. Matriz A=*pde2961*

Retomando datos anteriores, de la tabla 5.66, es una matriz no simétrica de tamaño 2961× 2961, con un número de elementos nulos de 14585 y su porcentaje del 0.166 %. Dado que no es simétrica no es posible aplicar la factorización de Cholesky.

En la tabla 5.87, se encuentra los resultados al aplicar la factorización LU.

	Densa	**AIJ**	**CSR**	**CSR mod**
Fact_LU (seg)	76.679	166.318	37.123	17.233
Sol Ax (seg)	0.068	3.552	0.411	0.379
T. total (seg)	76.747	169.87	37.534	17.612
Residuo $\|\cdot\|_2$	7.486E-12	2.387E-8	2.387E-8	2.387E-8
Error $\|\cdot\|_2$	9.660E-12	5.750E-8	5.750E-8	5.750E-8

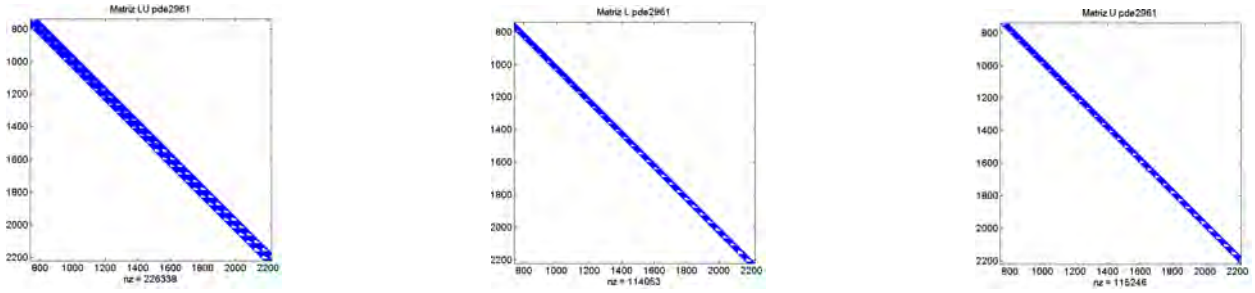
Tabla 5.87: Resultados factorización LU *pde2961*.

En la tabla 5.88, se da a conocer los resultados al aplicar el método de eliminación Gaussiana.

	T. Densa (seg)	**T. AIJ (seg)**	**T. CSR (seg)**	**T. CSR MOD (seg)**
Elim. Gauss. (seg)	73.502	165.235	36.483	17.102
Sol Ax (seg)	0.031	1.737	0.218	0.208
T. total (seg)	73.553	166.972	36.701	17.310
Residuo $\|\cdot\|_2$	9.407E-12	2.304E-8	2.304E-8	2.304E-8
Error $\|\cdot\|_2$	1.892E-11	5.750E-8	5.750E-8	5.750E-8

Tabla 5.88: Resultados eliminación Gaussiana *pde2961*.

En la figura 5.88, se da a conocer los gráficos de esparcidad de las matrices obtenidas de los métodos directos utilizados.



(a) LU, nnz=226,338. (b) L, nnz=114,053. (c) U, nnz=115,248.

Figura 5.38: Gráficos de esparcidad, matrices LU, L y U *pde2961*.

Observación 5.7. Los resultados al aplicar los métodos de la factorización y la eliminación Gaussiana, al sistema lineal asociado a *pde2961*, se encontró lo siguiente:

- De las tablas 5.87 y 5.88, a diferencia de los anteriores resultados se observa la eficiencia al realizar el proceso de la transformación de la matriz del sistema a uno triangular con los esquemas **CSR** y **CSR mod** y por ende el tiempo total del método, puesto que la búsqueda tarda menos de un segundo en estos métodos. Para el caso del error y el residuo, el valor es más grande que en los anteriores resultados, aplicando los esquemas de almacenamiento con un valor cercano a 1E-8 y se aleja más de la tolerancia propuesta. Al comparar los resultados con los del métodos del gradiente estos se tardan más mientras, dado que mientras que **BICGSTAB** se tarda menos de 1 segundo, mientras que el menor tiempo de los métodos directos fue de cerca de 18 segundos.
- De la figura 5.38, en las gráficas (a), (b) y (c) se observan que los elementos no cero, están cerca de la diagonal, el porcentaje de elementos no ceros es del 2.581 % para el caso de la matriz LU, del 1.31 % para la matriz L y del 1.31 % para la matriz U, por lo cual el aumento no están grande en sus elementos como el anterior y el problema del aumento de nnz no es tan considerable.

3. Matriz A=*s3rmt3m3*

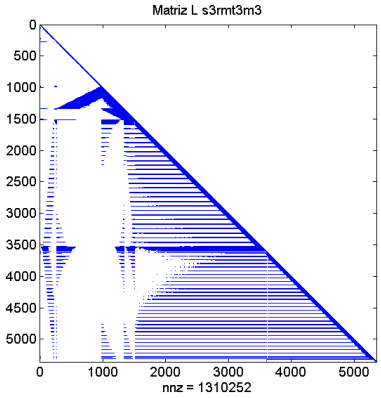
Retomando datos anteriores, de la tabla 3, es una matriz es de tamaño 5357×5357, con un porcentaje nnz del 0.71 %. La matriz es S.D.P y el valor de su condicionamiento es 2.6E+10, por lo cual es muy mal condicionada.

En la tabla 5.89 se muestran los resultados al aplicar la factorización de Cholesky para resolver el sistema de la matriz *s3rmt3m3*.

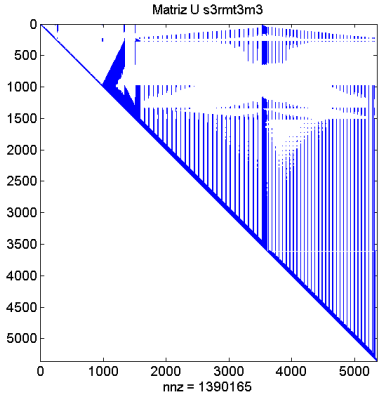
	Densa	AIJ	CSR	CSR MOD
Fact_Cho	184.699	178.218	180.424	180.247
Sol Ax (seg)	0.462	0.439	0.437	0.433
T. total (seg)	185.161	178.657	180.861	180.680
Residuo $\|\cdot\|_2$	1.138E-8	1.220E-8	1.220E-8	1.220E-8
Error $\|\cdot\|_2$	1.316E-5	9.849E-6	9.849E-6	9.849E-6

Tabla 5.89: Resultados Factorización de Cholesky *s3rmt3m3*.

En la figura 5.39, se encuentra las gráficas de esparcidad de la matriz L y L^T obtenidas de la factorización de Cholesky.



(a) L, nnz=1310252.



(b) L^T, nnz=1390165.

Figura 5.39: Gráficos de esparcidad, matrices L y L^T, *s3rmt3m3*.

Observación 5.8. A partir de la tabla 5.89, los tiempos para realizar la factorización con la matriz densa y los esquemas de almacenamiento no hay diferencias notables, sin embargo se observa que el **AIJ** registra menor tiempo para este proceso y el que más se tarda es usando la matriz densa. Respecto al tiempo para resolver los sistemas triangulares los tiempos son muy cercanos y esto se debe cumplir ya que este proceso se realiza con la matriz densa L. Sobre el error, este es más grande que el residuo y estos valores superan la tolerancia propuesta y se debe al mal condicionamiento de la matriz. Al comparar los resultados con los del gradiente se puede ver que en estos métodos se puede aplicar la matriz densa y además tardan más en encontrar la solución.

De la figura 5.39, se observa gráficamente, un aumento en la densidad y su distribución en diferentes partes de la matriz, por lo que la factorización de Cholesky, era más conveniente.

5.5.2. Resultados computador 2

Ahora utilizando el computador 2, se realizan las pruebas numérica con matrices esparzas de mayor tamaño y con mayor porcentaje de densidad.

4. Matriz $A = bcsstk17$

Retomando datos anteriores, de la tabla 5.74, es una matriz es de tamaño 10974×10974, con un porcentaje nnz del 0.356 %. La matriz es S.D.P y el valor de su condicionamiento es 65.

En las tablas 5.90, 5.91 y 5.92 se muestran los resultados al aplicar los métodos de la factorización de Cholesky, de la eliminación Gaussiana y de la factorización LU. Con la figura 5.92 se observa el gráfico de esparcidad de las matrices obtenidas de los métodos directos mencionados.

	T. Densa (seg)	**T. CSR MOD (seg)**
Fact_Cho	506.265	510.917
Sol Ax (seg)	0.870	0.878
Error $\|\cdot\|_2$	1.254E-8	1.254E-8
Residuo $\|\cdot\|_2$	0.0034	0.0034

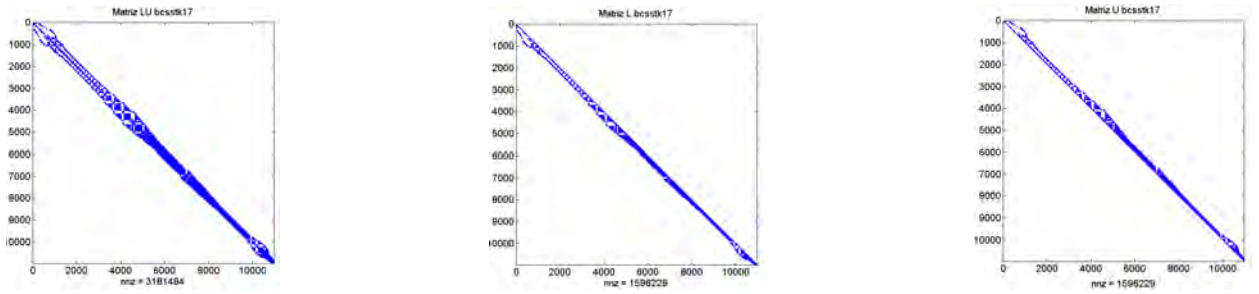
Tabla 5.90: Resultados factorización de Cholesky *bcsstk17*.

	T. Densa (seg)	**T. CSR MOD (seg)**
Elim. Gauss.	1034.005	312.004
Sol Ax (seg)	0.153	1.051
Error $\|\cdot\|_2$	3.998E-8	4.194E-8
Residuo $\|\cdot\|_2$	0.005	0.004

Tabla 5.91: Resultados eliminación Gaussiana *bcsstk17*.

	T. Densa (seg)	**T. CSR mod (seg)**
Fact_LU	1047.64	336.828
Sol Ax (seg)	0.313	1.900
Error $\|\cdot\|_2$	2.800E-8	6.159E-6
Residuo $\|\cdot\|_2$	0.004	0.157

Tabla 5.92: Resultados Factorización *LU* *bcsstk17*.



(a) LU, nnz=3181484. (b) L, nnz=1596229. (c) U, nnz=1596229.

Figura 5.40: Gráficos de esparcidad, matrices LU, L y U, *bcsstk17*.

Observación 5.9. Al aplicar los métodos directos para resolver el sistema asociado a la matriz *bcsstk17*, se pudo notar lo siguiente:

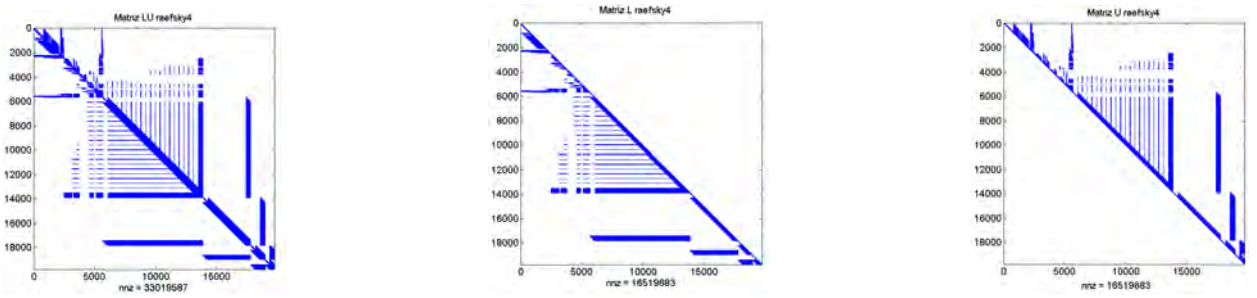
- Al comparar los resultados de las tablas 5.90, 5.91 y 5.92 el tiempo total utilizado para encontrar la solución se dio mejores resultados con el método de eliminación Gaussiana utilizando el esquema **CSR mod**. El método que más se tardo fue la factorización de Cholesky, que a diferencia de la anterior, esta era la que mejor resultados arrojaba. Sobre los resultados del error y el residuo, se puede ver que el residuo es muy grande pero el error es más pequeño. Ahora de la tabla 5.75 se ve que tardan mucho menos los métodos del gradiente precondicionados.
- De la figura 5.40, se aumento al densidad en un porcentaje superior al 1 %, en las tres matrices, pero todavía sigue siendo esparza.

5. Matriz A = *raefsky4*

Retomando datos anteriores, de la tabla 5.82, es una matriz es de tamaño 19779×19779, con un porcentaje nnz del 0.336 %. La matriz es S.D.P y el valor de su condicionamiento es 3.130E+13, por lo cual es muy mal condicionada. A continuación se presenta mediante la tabla 5.93 los resultados al utilizar el método de eliminación Gaussiana con **CSR mod** y de la figura 5.41 las matrices que se derivan de aplicar el método mencionado.

CSR MOD	
Elim. Gauss. (hrs)	5.0952
Sol. Ax (seg)	8.31
Error $\|\cdot\|_2$	1.271E-5
Residuo $\|\cdot\|_2$	6.034E-8

Tabla 5.93: Resultados eliminación Gaussiana *raefsky4*.



(a) LU, nnz=33019587. (b) L, nnz=16519683. (c) U, nnz=16519683.

Figura 5.41: Gráficos de esparcidad, matrices LU, L y U, *raefsky4*.

Observación 5.10. Al aplicar el método de eliminación Gaussiana, asociado a la matriz *raefsky4* se pudo notar que:

- En la tabla 5.83, se observa resultados muy ineficientes en tiempo para aplicar la eliminación Gaussiana a la matriz utilizando el esquema **CSR mod**, si se comparan con los métodos del gradiente se tardo dicho método mucho menos a pesar de no realizar más iteraciones para encontrar su convergencia.
- De la figura 5.41, se observa un aumento muy grande en la densidad de la matriz del 8.44 % para la matriz compacta LU y del 4.22 % para las matrices L y U.

Resolución de varios sistemas lineales

La resolución de sistemas lineales, se utiliza de forma iterativa en algunas discretizaciones numéricas, en donde los sistemas poseen la misma matriz y el vector independiente es el que va variando. Debido a esto es necesario utilizar métodos que sean eficientes para resolver varios sistemas de este tipo. Una de las cualidades de los métodos directos como la factorización LU y la factorización de Cholesky, es que para un sistema lineal $A\boldsymbol{x} = \boldsymbol{b}$, se observa que sus procesos se dividen en: la factorización de la matriz del sistema A y posteriormente, mediante el proceso de *forward substitution* y *backward substitution* se encuentra la solución del sistema. Esto quiere decir que se puede sacar provecho para resolver varios sistemas lineales, donde la matriz del sistema A se mantenga y el vector independiente $\boldsymbol{b}$ va variando, puesto que únicamente se va repetir varias veces el proceso de solución del sistema y como se observó en los resultados este resultado no es tan costoso.

A continuación se muestran algunos resultados al resolver varios sistemas lineales, con la matriz del sistema fija y el vector independiente variando de forma aleatoria. Para ello se toman matrices anteriores y los métodos directos e iterativos con los que se lograron mejores resultados.

1. Matriz *A=nos3*. Resolución de 200 sistemas lineales.

   En las tablas 5.94 y 5.95, se provee información al aplicar factorización *LU* y la factorización de Cholesky respectivamente, al resolver varios sistemas lineales con la matriz del sistema *nos3*. Para analizar el error y el residuo se ha tomado el valor máximo de estos valores medido en norma 2 (*Residuo max* $\|\cdot\|_2$ y *Error max* $\|\cdot\|_2$)) de los sistemas lineales.

   En la tabla 5.96, se registran los resultados al aplicar el **CG** para resolver los sistemas lineales. En esta misma tabla se registra el residuo y el error máximo medido con la norma 2 (*Residuo max* $\|\cdot\|_2$ y *Error max* $\|\cdot\|_2$) de los sistemas lineales.

	Densa	**CSR**	**CSR mod**
Fact_LU (seg)	2.626	6.77	2.174
sol Ax (seg)	1.358	9.693	9.001
T. total (seg)	3.984	16.463	11.175
Error max $\|\cdot\|_2$	1.887E-9	3.123E-9	3.123E-9
Residuo max $\|\cdot\|_2$	3.398E-10	1.758E-8	1.758E-8

Tabla 5.94: Solución de 200 sistemas *nos3*, factorización LU.

	Densa	**CSR**	**CSR mod**
Fact_Cho (seg)	1.019	1.03	1.03
sol Ax (seg)	1.597	1.793	1.784
T. total (seg)	2.616	2.823	2.814
Residuo max $\|\cdot\|_2$	2.459E-10	2.536E-10	2.536E-10
Error max $\|\cdot\|_2$	1.071E-9	1.145E-9	1.145E-9

Tabla 5.95: Solución de 200 sistemas *nos3*, factorización de Cholesky.

Método CG	**Densa**	**CSR**	**CSV**
sol Ax (seg)	479.293	11.851	14.321
Residuo max $\|\cdot\|_2$	9.987E-13	9.996E-13	9.996E-13
Error max $\|\cdot\|_2$	2.01E-9	1.937E-9	1.937E-9

Tabla 5.96: Solución de 200 sistemas *nos3*, **CG**.

Observación 5.11. Al analizar las tablas 5.94, 5.95 y 5.96 se deduce que con la factorización de Cholesky es donde se realizar de manera más eficiente en tiempo estos métodos. También se observa que el procedimiento al aplicar la matriz densa con los métodos directos utilizados, es más eficiente que al aplicarla para los métodos del gradiente abordados. Respecto al valor del residuo y error máximos los métodos mantiene su valor semejante que al resolver un solo sistema.

2. Matriz A=*pde2961*. Resolución de 50 sistemas lineales.

En las tablas 5.97 y 5.98, se provee la información al aplicar factorización LU y el método **BICGSTAB** respectivamente al resolver 50 sistemas lineales con la matriz del sistema *pde2961*.

	Densa	**CSR**	**CSR mod**
Fact_LU (seg)	77.969	39.209	17.585
sol Ax (seg)	3.185	21.045	19.662
T. total (seg)	81.154	60.254	37.247
Error max $\|\cdot\|_2$	2.536E-11	1.531E-7	1.531E-7
Residuo max $\|\cdot\|_2$	7.701E-12	2.665E-8	2.665E-8

Tabla 5.97: Resolución de 50 sistemas *pde2961* factorización LU.

BICGSTAB	**Densa**	**CSR**	**CSV**
sol Ax (seg)	1246.5	6.521	7.920
Residuo max $\|\cdot\|_2$	9.778E-13	9.778E-13	9.3871E-13
Error max $\|\cdot\|_2$	6.848E-11	6.848E-11	6.848E-11

Tabla 5.98: Resolución de 50 sistemas *pde2961* **BICGSTAB**.

Observación 5.12. En las tablas 5.97y 5.98 anteriores, se obtuvo mejores resultados en tiempo al resolver los sistemas con **BICGSTAB**, con excepción de la matriz densa la cual tardo menos tiempo con la factorización LU. Para el caso del error y el residuo máximo los valores más grandes se dieron en la factorización LU.

3. Matriz A=*s3rmt3m3*. Resolviendo 10 sistemas lineales.

En las tablas 5.99 y 5.100, se provee la información al aplicar la factorización de Cholesky y el método **CG pre** respectivamente al resolver 50 sistemas lineales con la matriz del sistema *s3rmt3m3*.

	Densa	**CSR**	**CSR mod**
Fact_Cho (seg)	175.532	176.374	176.717
sol Ax (seg)	4.242	4.18	4.211
T. total (min)	2.996	3.009	3.015
Error max $\|\cdot\|_2$	1.752E-4	1.643E-4	1.643E-4
Residuo max $\|\cdot\|_2$	1.250E-8	1.238E-8	1.238E-8

Tabla 5.99: Resolución de 10 sistemas lineales *s3rmt3m3*, factorización de Cholesky.

CGpre	**CSR**	**CSV**
sol Ax (min)	8.867	11.619
Residuo max $\|\cdot\|_2$	9.968E-13	9.968E-13
Error max $\|\cdot\|_2$	1.623E-4	1.623E-4

Tabla 5.100: Resolución de 10 sistemas lineales *s3rmt3m3*, **CG pre**.

Observación 5.13. De las tablas 5.99 y 5.100, se encontró que efectivamente al resolver estos sistemas con la factorización de Cholesky se realizo más rápido este procedimiento. Sin embargo el residuo máximo es más pequeño con el **CG pre** y supera la tolerancia esperada. Para le caso del error máximo, en ambos casos el error es cerca de 1E-4.

Capítulo 6

Conclusiones

6.1. Conclusiones

A continuación se dan a conocer las conclusiones a partir de la teoría recopilada y de los resultados numéricos obtenidos.

- En el presente trabajo se desarrolló, un enfoque tanto teórico como computacional sobre el estudio de las matrices esparzas, que resulta de interés para el lector al profundizar ciertos temas que son a menudo investigados sobre dichas matrices desde dos puntos de vista. La teoría tratada se muestra de forma sistemática y ordenada, mediante ejemplos y demostraciones realizadas y reescritas de los textos utilizados de la manera más clara posible. Además se realizaron implementaciones propias en lenguaje C++ y MATLAB para entender, justificar y obtener los resultados computacionales, y así ubicarse en el papel de programador e investigador, de una temática de actual interés.

- Los resultados numéricos presentados a través de los algoritmos creados en el software DEV-C++, usando la programación orientada a objetos POO y creados en el software MATLAB, permitieron observar la eficiencia e ineficiencia de ciertas operaciones matriciales aquí estudiadas, y poder a partir de este análisis realizar las posibles mejoras, particularmente en el manejo de la inserción de elementos y en la aplicación de métodos directos, vistos en el capítulo 3.

- La discretización numérica realizada de la ecuación de onda Unidireccional en el capítulo 5 a pesar de ser muy básica, evidencia la importancia de la escogencia de un esquema de almacenamiento para matrices esparzas adecuado, con el fin de almacenar utilizando la menor cantidad de memoria posible, y realizar de forma eficiente una discretización que se resume en varios producto matriz por vector y que sea lo más cercano posible a la solución exacta.

- Si bien en la teoría no se presentaban cambios en el orden de complejidad al utilizar los esquemas de almacenamiento para matrices esparzas en operaciones tales como el producto matriz

por vector, en las pruebas numéricas, los tiempos de cómputo y la memoria ocupada varían dependiendo del esquema a utilizar. Entre los resultados más representativos se encuentran el mayor gasto de memoria al usar **CSR mod** y el menor gasto de memoria con **CSV** o **MSR**. Para el caso del tiempo, estuvo ligado a la memoria usada por el almacenamiento, por lo cual en términos generales el **CSR**, **MSR** y **CSV** fueron los que mejor se comportaron en las operaciones mencionadas, al realizar en gran cantidad estas operaciones, que además implicaban utilizarlos en los métodos del gradiente.

- Los formatos de lectura para leer una matriz esparza como el formato coordinado de matrix market [5], contienen información esencial y un orden específico (por filas o columnas) de la matriz esparza asociada, permitiendo con el uso de los esquemas de almacenamiento para dichas matrices, almacenarlas dependiendo de las propiedades (simétrica o no simétrica) y realizándolo de forma más eficiente en tiempo y en costos de memoria. En cambio, sin estos formatos el proceso de almacenamiento se vuelve costoso en tiempo al realizar un reordenamiento de las posiciones de los elementos y requiere un manejo cuidadoso en la memoria asignada, como se observó en la sección de inserción de elementos del capítulo 5.

- Los métodos del gradiente, fueron los métodos iterativos utilizados para resolver los sistemas lineales propuestos, donde se obtuvieron resultados que en algunos casos cumplían con la teoría y en otros no. Entre las consecuencias que se corroboró, está la relación valor de las entradas y el condicionamiento de la matriz del sistema, la cual entre más pequeña sean las entradas y mejor condicionada sea la matriz, convergía a la solución. Una de las potencialidades notorias fue que en algunas matrices de grandes dimensiones que presentaban estas características o que se conocía al menos una de estas, realizaban estos métodos en muy pocas iteraciones y en poco tiempo. Sobre la relación número de iteraciones y el tiempo que se tarda los métodos, se determinó que puede suceder que en menor iteraciones un método se puede tardar más que otro o viceversa. Uno de los aspectos relevantes para convergencia de los métodos del gradiente, fue el uso del precondicionador de Jacobi y mejor aún si se lo implementa de forma implícita que explícita, sin embargo en muchos casos ni siquiera con este precondicionador se lograba tal convergencia.

- Los métodos directos utilizados como la eliminación Gaussiana, la factorización LU y la factorización de Cholesky, tuvieron los mayores problemas en las implementaciones. El problema del aumento de los elementos no nulos en el proceso de transformación de la matriz a una triangular (fill-in), hace que no sea conveniente por el aumento de memoria considerable al utilizar los esquemas de almacenamiento abordados, y por ende es ineficiente para solucionar los sistemas. En algunos el aumento de estos elementos no presentó inconvenientes pero si tardaron considerablemente. El esquema **CSR mod**, es el que menos tarda en este proceso debido a la forma de su almacenamiento, pero igualmente presenta problemas en la saturación

de su memoria más que en los demás, debido a que por su estructura utiliza mayor espacio de memoria que los otros.

6.2. Direcciones futuras de investigación.

A continuación se presentan las direcciones futuras de investigación que se pueden seguir a partir del trabajo realizado.

- Realización de un estudio tanto teórico y computacional con matrices esparzas complejas, siguiendo el mismo enfoque del presente estudio.
- Implementación de otros tipos de esquemas de almacenamiento para matrices esparzas que sean iguales o más eficientes a los ya estudiados en el trabajo, con relación a operaciones matriciales de alto costo.
- Estudio de otros métodos para resolución de sistemas lineales en métodos iterativos como GMRES y MULTIGRID, y en otros métodos directos como la factorización LU con pivoteo parcial y completo y la factorización LU incompleta.
- Aplicación de otros tipos precondicionadores además del de Jacobi, para la convergencia de métodos iterativos, particularmente de los métodos del gradiente.
- Implementaciones de los métodos utilizando la programación en Paralelo y en GPU, para mejorar o los resultados computacionales obtenidos.
- Estudiar la matriz de discretización y su proceso de obtención de un problema científico en particular.

Apéndice

A.1. Formato Coordiando

Debido a que la mayoría de las matrices esparzas son de grandes dimensiones y por el costo de memoria que esto conlleva, se han creado formatos de lectura que tengan en cuenta información sobre sus elementos no nulos e información adicional tales como: el número de filas y columnas, tipo de matriz, el tipo de entradas que posee, el número de elementos no nulos en total y por cada fila, entre otros mas.

Un ejemplo de formatos de lectura para matrices esparzas tenemos el "coordinate format" del recurso electrónico de "Matrix Market" [3, 5], como se observa en la figura 6.1, el cual tiene

- Un **encabezado** que nos da la información del número de filas y columnas, el tipo de entradas de la matriz (e.g real, integer, real, complex) y el tipo de simetría (e.g symmetric, skew-symmetric, general). En algunos casos contienen otro segundo encabezado acerca de otra información adicional como su aplicación, año, autores, etc, el cual lo hemos suprimido a la hora de leer los archivos que tenga dicho encabezado.
- Los **elementos no nulos de la matriz** que van después del encabezado y vienen organizados por sus indices de filas, columnas y su valor respectivo. El orden que van organizados los elementos es por orden de columna y luego por orden de fila.

```
%%MatrixMarket matrix coordinate real symmetric
48 48 224
1 1  2.8322685185200e+06
5 1  1.0000000000000e+06
6 1  2.0833333333300e+06
7 1 -3.3333333333300e+03
11 1  1.0000000000000e+06
19 1 -2.8000000000000e+06
25 1 -2.8935185185200e+04
30 1  2.0833333333300e+06
2 2  1.6354475308600e+06
4 2 -2.0000000000000e+06
6 2  5.5555555555500e+06
```

Figura 6.1: Ejemplo de un archivo con el formato coordinado.

Bibliografía

[1] Autoevolution. http://autoevolution.com/news/, 30 de Octubre, 2016.

[2] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on cuda. Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.

[3] Ronald F Boisvert, Roldan Pozo, Karin Remington, Richard F Barrett, and Jack J Dongarra. Matrix market: a web resource for test matrix collections. In *Quality of Numerical Software*, pages 125–137. Springer, 1997.

[4] Ronald F Boisvert, Roldan Pozo, Karin Remington, Richard F Barrett, and Jack J Dongarra. The matrix market, http://www.math.nist.gov, 20 de Octubre, 2016.

[5] Ronald F Boisvert, Roldan Pozo, and Karin A Remington. The matrix market exchange formats: Initial design. *National Institute of Standards and Technology Internal Report, NISTIR*, 5935, 1996.

[6] Aydin Buluç, Jeremy T Fineman, Matteo Frigo, John R Gilbert, and Charles E Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, pages 233–244. ACM, 2009.

[7] Richard L Burden, J Douglas Faires, and Oscar Palmas. *Análisis numérico*. Number QA297. B87 1985. Thomson Learning México, 2002.

[8] Paul C. *Algebra Lineal Numérica*. Notas de clase, Universidad de Puerto Rico, 2005.

[9] Rúa C. *Implementación de una libreria orientada a objetos para matrices esparcidas en paralelo*. Maestria, Universidad de Puerto Rico, 2007.

[10] Rúa C. *Adaptive computaional simulation of two-phase viscoelastic flows*. Doctorado, Universidad de Puerto Rico, 2013.

[11] Osvaldo Cairó, Silvia Guardati, and Silvia Guardati Osvaldo Cairó. *Estructuras de datos*. Mc Graw Hill, third edition, 2006.

[12] Charles G. Cullen. *An introduction to numerical linear algebra*. PWS Boston, 1964.

[13] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[14] DEV-C++. *Versión 4.9.9.2*. Free Software Foundation Inc. http://www.bloodshed.net, Cambridge, Massachusetts, USA, 2014.

[15] Aiyoub Farzaneh, Hossein Kheiri, and Mehdi Abbaspour Shahmersi. An efficient storage format for large sparse matrices. *Communications Series A1 Mathematics & Statistics*, 582:1–10, 2009.

[16] Golub Gener H., Charles F., and Van Loan. *matrix computations*. JHU Press., tercera edition, 1996.

[17] Martin B Gijzen, Gerard LG Sleijpen, and Jens-Peter M Zemke. Flexible and multi-shift induced dimension reduction algorithms for solving large sparse linear systems. *Numerical Linear Algebra with Applications*, 22(1):1–25, 2015.

[18] Magnus Rudolph Hestenes and Eduard Stiefel. *Methods of conjugate gradients for solving linear systems*, volume 49. NBS, 1952.

[19] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[20] Lei Jiang, Jianlong Tan, and Qiu Tang. An efficient sparse matrix format for accelerating regular expression matching on field-programmable gate arrays. *Security and Communication Networks*, 8(1):13–24, 2015.

[21] Stanley l Grossman. *Álgebra lineal*. Mc Graw Hill, 2004.

[22] MATLAB. *Versión 8.3 (R2014a)*. The MathWorks Inc. https://www.mathworks.com, Natick, Massachusetts, 2014.

[23] Carlos Enrique Mejía, Tomás Restrepo, and Christian Trefftz. Lapack una colección de rutinas para resolver problemas de algebra lineal numérica. *Revista Universidad EAFIT*, 37(123):73–80, 2012.

[24] Carlos Enrique Mejía, Tomás Restrepo, and Christian Trefftz. Una introducción al uso de lapack. *Cuadernos de Investigación*, (3), 2012.

[25] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[26] Yousef Saad. *Iterative methods for sparse linear systems*. Siam, second edition, 2003.

[27] F Javier Ceballos Sierra. *Programación orientada a objetos con C++*. Ra-ma, 1993.

[28] Peter Sonneveld. Cgs, a fast lanczos-type solver for nonsymmetric linear systems. *SIAM journal on scientific and statistical computing*, 10(1):36–52, 1989.

[29] John C Strikwerda. *Finite difference schemes and partial differential equations*. Siam, 2004.

[30] San Diego University of California. Jacobs school of engineering, http://www.jacobsschool.ucsd.edu/, 30 de Octubre, 2016.

[31] Henk A Van der Vorst. Bi-cgstab: A fast and smoothly converging variant of bi-cg for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing*, 13(2):631–644, 1992.

[32] Dimitri van Heesch. Doxygen: Source code documentation generator tool. URL: http://www.doxygen.org. 2008.

[33] Hu Yiang. A gallery of large graphs, http://yifanhu.net/GALLERY/GRAPHS, 20 de Octubre, 2016.

Índice alfabético