USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

Resolución de sistemas de ecuaciones lineales con matrices complejas no hermitianas

Antón Soto Martínez

2024-2025

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRADO DE MATEMÁTICAS

Traballo Fin de Grado

Resolución de sistemas de ecuaciones lineales con matrices complejas no hermitianas

Antón Soto Martínez

Junio, 2025

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Conocimiento: Matemática Aplicada
Título: Resolución de sistemas de ecuaciones lineales con matrices complejas no hermitianas
Breve descripción del contenido
El método de Gradiente Conjugado es un método muy eficiente para resolver sistemas de ecuaciones lineales Ax = b en los cuales la matriz A es simétrica y definida positiva. En la práctica, surgen numerosos sistemas lineales donde la matriz A es compleja y no hermitiana; esto ocurre, por ejemplo, en la discretización de sistemas de ecuaciones en derivadas parciales que modelan procesos electromagnéticos. En la literatura se pueden encontran distintas variantes del método de Gradiente Conjugado para abordar estos problemas. El objetivo del trabajo será estudiar e implementar algunas de estas metodologías, y en particular, el método denominado Conjugate Orthogonal Conjugate Gradient (COCG). Para el desarollo del trabajo, será preciso profundizar primero en el método clásico de Gradiente Conjugado.
Recomendaciones
Otras observaciones

Índice

Resumen

El objetivo de este trabajo es dar a conocer un método iterativo denominado «método de Gradiente Conjugado Ortogonal» para resolver sistemas de ecuaciones lineales con matrices complejas, simétricas y no hermitianas. Dado que el método pertenece a los llamados métodos de Krylov, en el trabajo se introducirán estos métodos en el caso de matrices reales para luego extender alguno de ellos al caso de matrices complejas. Se introducirán primero los métodos de Gradiente Conjugado y Gradiente Biconjugado para, por último, presentar el método de Gradiente Conjugado Ortogonal. Se describirán algoritmos para implementar los distintos métodos y se presentarán algunos ejemplos prácticos con matrices obtenidas de Matrix Market.

Abstract

The aim of this dissertation is to present an iterative technique, the «Conjugate Orthogonal Conjugate Gradient» (COCG) method, designed to solve linear systems $Ax = b$ whose coefficient matrix A is complex, symmetric in the real sense, and non-Hermitian. Because COCG belongs to the family of Krylov subspace methods, we first introduce these methods for real matrices and then show how several of them can be extended to the complex case. The exposition begins with the Conjugate Gradient (CG) and Biconjugate Gradient (BCG) algorithms, and culminates with a detailed presentation of the COCG scheme. Algorithms for implementing the different methods will be described, and some practical examples using matrices obtained from Matrix Market will be presented.

Introducción

La resolución eficiente de sistemas lineales de gran tamaño mediante métodos iterativos es un problema fundamental en las matemáticas. En la actualidad, existen numerosas investigaciones sobre métodos iterativos para resolver sistemas lineales. Algunos libros de referencia en esta temática son [5] y [3]. En este trabajo se introducen y desarrollan algunos métodos para la resolución de dichos problemas, y en particular se desarrollará el método de Gradiente Conjugado Ortgonal para la resolución de sistemas lineales de la forma $Ax = b$ cuando A es una matriz compleja no hermitiana ($A \neq A^H$) y simétrica ($A = A^T$). Este tipo de sistemas surgen por ejemplo en la discretización de EDPs como la ecuación de Helmholzt compleja [4] o en problemas de corrientes de Focault [8]. El método de Gradiente Conjugado Ortgonal, COCG, por sus siglas en inglés (Conjugate Orthogonal Conjugate Gradient) fue propuesto en la década de los 90 por van der Vorst y Melissen en [8] para resolver problemas que surgen al discretizar ecuaciones de corrientes de Focault. Otros métodos para resolver sistemas con matrices complejas simétricas y no hermitianas pueden verse en el capítulo 3 de [7] y en el artículo [4].

El objetivo de este trabajo es abordar de manera didáctica la teoría sobre los métodos de Krylov, hasta tener herramientas adecuadas para describir y formular el método COCG. Para ello se siguen los libros [5] y [7] así como en el artículo [8]. Se ha implementado en Matlab el método COCG y se ha validado resolviendo ejemplos extraídos del repositorio Matrix Market [2].

A lo largo del trabajo se describirán algoritmos usando pseudocódigo, dicho pseudocódigo será formulado en inglés ya que de este modo se facilita el uso del entorno algorithm en Latex. Este trabajo consta de tres capítulos que siguen un orden coherente para facilitar la comprensión de los contenidos. El primer capítulo, introduce fundamentos del álgebra lineal y conceptos básicos que se necesitan para el desarrollo del trabajo. El segundo capítulo, introduce los métodos iterativos llamados métodos de Krylov que se basan en los métodos de proyección. Se consideran sistemas lineales con matrices reales para adentrarnos en dichos métodos de manera más didáctica. Además se abordan los métodos derivados o relacionados con la ortogonalización de Arnoldi y los relacionados con la biortogonalización de Lanczos. El tercer capítulo, desarrolla los métodos

de Krylov pero para sistemas lineales complejos siguiendo [7] y finaliza con la descripción del método COCG. El capítulo finaliza con la presentación de los ejemplos utilizados para validar el código que implementa el método Gradiente Conjugado Ortogonal. El código de Matlab utilizado se describe en el Anexo I.

Capítulo 1

Fundamentos de álgebra lineal

En este capítulo, se introducen los conceptos básicos relacionados con el álgebra lineal que serán necesarios en este trabajo siguiendo [5].

1.1. Matrices

Se supone que todos los espacios vectoriales considerados en este capítulo son complejos, a menos que se indique lo contrario. Una matriz compleja A de tamaño $n \times m$ es una matriz de números complejos

$$a_{ij}, \quad i = 1, \ldots, n, \quad j = 1, \ldots, m.$$

El conjunto de todas las matrices $n \times m$ es un espacio vectorial complejo denotado por $\mathbb{C}^{n \times m}$. Las operaciones principales con matrices son las siguientes:

- Suma: $C = A + B$, donde A, B y C son matrices de tamaño $n \times m$ y

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

- Multiplicación por un escalar: $C = \alpha A$, donde

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

- Multiplicación por otra matriz: $C = AB$, donde

$$A \in \mathbb{C}^{n \times m}, B \in \mathbb{C}^{m \times p}, C \in \mathbb{C}^{n \times p}$$

y

$$c_{ij} = \sum_{k=1}^{m} a_{ik} b_{kj}.$$

En ocasiones, se utiliza una notación con vectores columna y vectores fila. El vector columna a_{*j} es el vector que consiste en la j-ésima columna de A,

$$a_{*j} = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{nj} \end{pmatrix}.$$

De forma similar, la notación a_{i*} denota la i-ésima fila de la matriz A:

$$a_{i*} = (a_{i1}, a_{i2}, \ldots, a_{im}).$$

Por ejemplo, una matriz A puede escribirse como

$$A = (a_{*1}, a_{*2}, \ldots, a_{*m}) = \begin{pmatrix} a_{1*} \\ a_{2*} \\ \vdots \\ a_{n*} \end{pmatrix}.$$

La transpuesta de una matriz A en $\mathbb{C}^{n\times m}$ es una matriz C en $\mathbb{C}^{m\times n}$ cuyos elementos están definidos por $c_{ij} = a_{ji}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Se denota por A^T.

A veces es más relevante usar la matriz conjugada transpuesta denotada por A^H y definida por

$$A^H = \overline{A}^T = \overline{A^T},$$

donde la barra indica la conjugación compleja (elemento a elemento).

Se dice que una matriz es cuadrada si el número de filas es igual al número de columnas y se denota por $\mathbb{C}^{n\times n}$ al conjunto de las matrices cuadradas $n \times n$.
La inversa de una matriz cuadrada A se denota por A^{-1}. Y se tiene que $AA^{-1} = I$ siendo I la matriz identidad. Para una matriz cuadrada, A se dice que A es singular cuando no exista A^{-1}. Del mismo modo, si existe A^{-1} se dirá que la matriz A es no singular.

1.2. Tipos de matrices

A continuación se introducen los tipos de matrices cuadradas que se consideran importantes para el desarrollo del trabajo. Las más importantes son las siguientes:

- Matrices simétricas: $A^T = A$.
- Matrices hermitianas: $A^H = A$.
- Matrices antisimétricas: $A^T = -A$.
- Matrices antihermitianas: $A^H = -A$.
- Matrices normales: $A^H A = AA^H$.
- Matrices no negativas: $a_{ij} \geq 0, \quad i, j = 1, \ldots, n$
  (definición similar para matrices no positivas, positivas y negativas).
- Matrices unitarias: $Q^H Q = I$.

A continuación se da una lista de matrices que tienen estructuras interesantes computacionalmente. Los tipos de matrices que se consideran a continuación hacen énfasis en la estructura, es decir, en las posiciones de los elementos no nulos con respecto a los ceros.

- Matrices diagonales: $a_{ij} = 0$ para $j \neq i$. Notación:

$$A = \text{diag}(a_{11}, a_{22}, \ldots, a_{nn}).$$

- Matrices triangulares superiores: $a_{ij} = 0$ para $i > j$.
- Matrices triangulares inferiores: $a_{ij} = 0$ para $i < j$.
- Matrices bidiagonales superiores: $a_{ij} = 0$ para $j \neq i$ o $j \neq i + 1$.
- Matrices bidiagonales inferiores: $a_{ij} = 0$ para $j \neq i$ o $j \neq i - 1$.
- Matrices tridiagonales: $a_{ij} = 0$ para todo par i, j tal que $|j - i| > 1$. Notación:

$$A = \text{tridiag}(a_{i,i-1}, a_{ii}, a_{i,i+1}).$$

- Matrices de Hessenberg superiores: $a_{ij} = 0$ para todo par i, j tal que $i > j+1$. Las matrices de Hessenberg inferiores se definen de forma análoga.

Se dice que una matriz es dispersa cuando la mayoría de sus elementos son nulos. Por el contrario a una matriz que no es dispersa se le llamará matriz densa o llena.

1.3. Producto interno y normas vectoriales

Un producto interno en un espacio vectorial $\mathbb{X}$ es una aplicación

$$x \in \mathbb{X},\ y \in \mathbb{X} \quad \longrightarrow \quad s(x,y) \in \mathbb{C},$$

que satisface las siguientes condiciones:

1. $s(x,y)$ es lineal respecto de x, es decir,

$$s(\lambda_1 x_1 + \lambda_2 x_2,\ y) = \lambda_1 s(x_1,y) + \lambda_2 s(x_2,y), \quad \forall x_1, x_2 \in \mathbb{X},\ \forall \lambda_1, \lambda_2 \in \mathbb{C}.$$

2. $s(x,y)$ es hermitiano, es decir,

$$s(y,x) = \overline{s(x,y)}, \qquad \forall x, y \in \mathbb{X}.$$

3. $s(x,y)$ es una aplicación definida positiva, es decir,

$$s(x,x) > 0 \quad \forall x \neq 0.$$

Nótese que (2) implica que $s(x,x)$ es real, y por ello, (3) añade la condición de que $s(x,x)$ también debe ser positivo para todo $x \neq 0$. Para cualquier x e y, se cumple:

$$s(x,0) = s(x,0y) = 0s(x,y) = 0.$$

De modo análogo, $s(0,y) = 0$ para todo y.

En el caso particular del espacio vectorial $\mathbb{X} = \mathbb{C}^n$ se define el producto interno euclídeo de dos vectores $x = (x_i)_{i=1}^n$ e $y = (y_i)_{i=1}^n$ como:

$$(x,y) = \sum_{i=1}^{n} x_i \overline{y_i}, \tag{1.1}$$

que suele reescribirse en notación matricial como:

$$(x,y) = y^H x. \tag{1.2}$$

Una propiedad fundamental del producto interno euclídeo en notación matricial es:

$$(Ax,y) = (x, A^H y), \qquad \forall x, y \in \mathbb{C}^n. \tag{1.3}$$

Se dice que una matriz real A es definida positiva si $(Ax,x) > 0 \quad \forall x \neq 0$. Y semidefinida positiva cuando $(Ax,x) \geq 0$. Análogamente, diremos que A es definida negativa cuando $(Ax,x) < 0$. Y semidefinida negativa cuando $(Ax,x) \leq 0$.

Se dice que una matriz compleja y hermitiana A es definida positiva si $(Ax, x) > 0 \quad \forall x \neq 0$. Y semidefinida positiva cuando $(Ax, x) \geq 0$. Análogamente, diremos que A es definida negativa cuando $(Ax, x) < 0$. Y semidefinida negativa cuando $(Ax, x) \leq 0$.

Proposición 1.1. *Las matrices unitarias preservan el producto interno euclídeo, es decir:*

$$(Qx, Qy) = (x, y),$$

para cualquier matriz unitaria Q y cualesquiera vectores x e y.

Demostración.

$$(Qx, Qy) = (x, Q^H Qy) = (x, y).$$

□

Una norma vectorial sobre un espacio vectorial $\mathbb{X}$ es una aplicación real $\|\cdot\|$ definida en $\mathbb{X}$ que satisface las siguientes condiciones:

1. $\|x\| \geq 0, \quad \forall x \in \mathbb{X}$, y $\|x\| = 0 \Leftrightarrow x = 0$.
2. $\|\alpha x\| = |\alpha| \, \|x\|, \quad \forall x \in \mathbb{X}, \ \forall \alpha \in \mathbb{C}$.
3. $\|x + y\| \leq \|x\| + \|y\|, \quad \forall x, y \in \mathbb{X}$.

Si $\mathbb{X} = \mathbb{C}^n$, se puede asociar con el producto interno (1.3) la norma euclídea de un vector complejo, definida por:

$$\|x\|_2 = (x, x)^{1/2}.$$

Las normas vectoriales más usadas en álgebra lineal numérica son casos particulares de las normas de Hölder.

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}. \tag{1.4}$$

Obsérvese que el límite de $\|x\|_p$ cuando $p \to \infty$ existe y es igual al valor absoluto máximo de los x_i. Esto define una norma denotada por $\|x\|_\infty$. Los casos más importantes en la práctica son:

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|, \quad \|x\|_2 = \left(|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2 \right)^{1/2}, \quad \|x\|_\infty = \max_{i=1,\ldots,n} |x_i|.$$

1.4. Normas matriciales

Para una matriz $A \in \mathbb{C}^{n \times m}$, se define el siguiente conjunto de normas:

$$\|A\|_{pq} = \max_{x \in \mathbb{C}^m, \ x \neq 0} \frac{\|Ax\|_p}{\|x\|_q}. \tag{1.5}$$

La norma $\|A\|_{pq}$ está inducida por las dos normas vectoriales $\|\cdot\|_p$ y $\|\cdot\|_q$. Estas normas satisfacen las propiedades habituales de una norma:

$$\|A\| \geq 0, \quad \forall A \in \mathbb{C}^{n\times m}, \tag{1.6}$$

$$\|A\| = 0 \;\Leftrightarrow\; A = 0. \tag{1.7}$$

$$\|\alpha A\| = |\alpha|\,\|A\|, \quad \forall A \in \mathbb{C}^{n\times m},\ \forall \alpha \in \mathbb{C} \tag{1.8}$$

$$\|A + B\| \leq \|A\| + \|B\|, \quad \forall A, B \in \mathbb{C}^{n\times m}. \tag{1.9}$$

Una norma que satisface estas tres propiedades es simplemente una norma vectorial aplicada a la matriz vista como un vector de tamaño nm.

Los casos más importantes vuelven a estar asociados con $p, q = 1, 2, \infty$. El caso particular $q = p$ es de especial interés, y la norma correspondiente $\|A\|_{pp}$ se denota simplemente por $\|A\|_p$, y se llama la norma-p. Una propiedad fundamental de cualquier norma-p es:

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

que es una consecuencia inmediata de la definición (1.5). Una norma matricial que cumple con la condición de arriba se dice que es consistente. Una consecuencia de esta consistencia es que para toda matriz cuadrada A:

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

En particular, la matriz A^k converge a cero si alguna de sus p-normas es menor que 1.
Las siguientes igualdades para normas matriciales permiten expresar de modo equivalente las nomas definidas previamente y conducen a un cálculo más sencillo:

$$\|A\|_1 = \max_{j=1,\ldots,m} \sum_{i=1}^{n} |a_{ij}|, \tag{1.10}$$

$$\|A\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{m} |a_{ij}|, \tag{1.11}$$

$$\|A\|_2 = \left(\rho(A^H A)\right)^{1/2} = \left(\rho(AA^H)\right)^{1/2}, \tag{1.12}$$

donde $\rho(A)$ denota el máximo de los autovalores de la matriz A en valor absoluto.

1.5. Subespacios

Un subespacio de $\mathbb{C}^n$ es un subconjunto de $\mathbb{C}^n$ que es un espacio vectorial complejo. El conjunto de todas las combinaciones lineales de un conjunto de vectores $G = \{a_1, a_2, \ldots, a_q\}$ de

$\mathbb{C}^n$ es un subespacio vectorial llamado el subespacio generado linealmente por G:

$$\text{span}\{G\} = \text{span}\{a_1, a_2, \ldots, a_q\} = \left\{ z \in \mathbb{C}^n \;\middle|\; z = \sum_{i=1}^{q} \alpha_i a_i, \quad \alpha_i \in \mathbb{C} \right\}.$$

Si los vectores a_i son linealmente independientes, entonces cada vector de span$\{G\}$ admite una expresión única como combinación lineal de los a_i. En este caso, el conjunto G se llama una base del subespacio span$\{G\}$.

Dados dos subespacios vectoriales S_1 y S_2, su suma S es el subespacio definido como el conjunto de todos los vectores que son suma de un vector de S_1 y un vector de S_2. La intersección de dos subespacios también es un subespacio.

Si la intersección de S_1 y S_2 es trivial ($\{0\}$), entonces la suma $S_1 + S_2$ se llama suma directa y se denota por:

$$S = S_1 \oplus S_2.$$

Cuando $S = \mathbb{C}^n$, todo vector $x \in \mathbb{C}^n$ se puede escribir de manera única como:

$$x = x_1 + x_2, \quad x_1 \in S_1, \quad x_2 \in S_2.$$

La transformación que asigna x a su componente x_1 es una transformación lineal idempotente, es decir, tal que $P^2 = P$. Se llama un proyector sobre S_1 a lo largo de S_2.

Dos subespacios importantes asociados a una matriz $A \in \mathbb{C}^{n\times m}$ son:

- Su imagen (rango):
$$\text{Ran}(A) = \{Ax \mid x \in \mathbb{C}^m\}.$$
- Su núcleo o espacio nulo:
$$\text{Null}(A) = \{x \in \mathbb{C}^m \mid Ax = 0\}.$$

La imagen de A es igual al subespacio generado linealmente por sus columnas. El rango de una matriz es la dimensión de su imagen, es decir, el número de columnas linealmente independientes.

Este número también coincide con el rango por filas, el número de filas linealmente independientes. Una matriz en $\mathbb{C}^{n\times m}$ es de rango completo si su rango es igual al menor de n y m.

Un resultado fundamental del álgebra lineal establece la siguiente relación:

$$\mathbb{C}^n = \text{Ran}(A) \oplus \text{Null}(A^T). \tag{1.13}$$

Este mismo resultado se aplica a la transpuesta de A:

$$\mathbb{C}^m = \text{Ran}(A^T) \oplus \text{Null}(A).$$

1.6. Vectores y subespacios ortogonales

Un conjunto de vectores $G = \{a_1, a_2, \ldots, a_r\}$ de $\mathbb{C}^n$ se dice que es ortogonal si:

$$(a_i, a_j) = 0 \quad \text{si} \quad i \neq j.$$

Se dice que es ortonormal si, además, cada vector de G tiene norma 2 (norma euclídea) unitaria. Un vector se dice ortogonal a un subespacio S si es ortogonal a todos los vectores de S. El conjunto de todos los vectores ortogonales a S es un subespacio vectorial, llamado el complemento ortogonal de S y denotado por $S^{\perp}$.

Así, el espacio $\mathbb{C}^n$ puede escribirse como la suma directa de S y su complemento ortogonal:

$$\mathbb{C}^n = S \oplus S^{\perp}.$$

Cualquier vector x de $\mathbb{C}^n$ puede escribirse de manera única como:

$$x = x_1 + x_2, \quad x_1 \in S, \quad x_2 \in S^{\perp}.$$

El operador que asigna x a su componente en el subespacio S se llama el proyector ortogonal sobre S.

Todo subespacio admite una base ortonormal que se puede obtener a partir de cualquier base mediante un proceso de ortonormalización. Una forma de hacerlo es el proceso de Gram–Schmidt.

Dado un conjunto de vectores linealmente independientes $\{x_1, x_2, \ldots, x_r\}$, el proceso de Gram–Schmidt consiste en normalizar x_1 dividiéndolo por su norma-2 para obtener q_1. Después se ortogonaliza x_2 respecto a q_1 restándole a x_2 un múltiplo de q_1 para hacer que el vector resultante sea ortgonal a q_1, es decir :

$$x_2 \leftarrow x_2 - (x_2, q_1)q_1.$$

Finalmente, se normaliza para obtener q_2. En general, el paso i consiste en ortogonalizar x_i respecto a todos los vectores anteriores q_j.

Algoritmo 1.2. *Gram–Schmidt*

1: *Compute* $r_{11} = \|x_1\|_2$.
2: ***if*** $r_{11} = 0$ ***then***
3: *Stop;*
4: ***else*** $q_1 = x_1/r_{11}$.
5: ***end if***
6: ***for*** $j = 2, \ldots, r$ ***do***

Compute $r_{ij} = (x_j, q_i)$, *para* $i = 1, \ldots, j-1$.
$\widehat{q} = x_j - \sum_{i=1}^{j-1} r_{ij} q_i$.
Compute $r_{jj} = \|\widehat{q}\|_2$.
if $r_{jj} = 0$ ***then***
    Stop;
else $q_j = \widehat{q}/r_{jj}$.
end if
end for

Es fácil ver que este algoritmo no fallará (completará r pasos) si y solo si el conjunto $\{x_1, x_2, \ldots, x_r\}$ es linealmente independiente. Además, se satisface que:

$$x_j = \sum_{i=1}^{j} r_{ij} q_i.$$

Si se definen:

$$X = [x_1, x_2, \ldots, x_r], \quad Q = [q_1, q_2, \ldots, q_r],$$

y R denota la matriz triangular superior $r \times r$ cuyos elementos no nulos son los r_{ij} definidos en el algoritmo, entonces:

$$X = QR. \tag{1.14}$$

Esta es la llamada descomposición QR de la matriz X. Existe una versión modificada del algoritmo estándar, conocida como Gram–Schmidt modificado (MGS), que tiene mejores propiedades numéricas. Al igual que el Algoritmo 1.2, MGS transforma un conjunto de vectores linealmente independiente en una base ortonormal; para ver en detalle resultados sobre los procesos de ortogonalización se recomienda la lectura de [6].

Algoritmo 1.3. *Gram–Schmidt modificado*

Define $r_{11} = \|x_1\|_2$.
if $r_{11} = 0$ ***then***
    Stop;
else $q_1 = x_1/r_{11}$.
end if
for $j = 2, \ldots, r$ ***do***
    Define $\hat{q} = x_j$.
    for $i = 1, \ldots, j-1$ ***do***
        $r_{ij} = (\hat{q}, q_i)$.
        $\hat{q} = \hat{q} - r_{ij} q_i$.
    end for
    Compute $r_{jj} = \|\hat{q}\|_2$.

13: **if** $r_{jj} = 0$ **then**
14: *Stop;*
15: **else** $q_j = \hat{q}/r_{jj}$.
16: **end if**
17: **end for**

1.7. Operadores de proyección

Los operadores de proyección o proyectores juegan un papel importante en el álgebra lineal numérica, en particular en los métodos iterativos para resolver problemas matriciales. Esta sección introduce estos operadores desde un punto de vista puramente algebraico y presenta algunas de sus propiedades más importantes.

1.7.1. Espacio imagen y espacio nulo de un proyector

Un proyector P es cualquier aplicación lineal de $\mathbb{C}^n$ en $\mathbb{C}^n$ que es idempotente, es decir, P una aplicación tal que:

$$P^2 = P.$$

De esta definición se deducen algunas propiedades sencillas. Primero, si P es un proyector, entonces también lo es $I - P$

$$(I-P)^2 = (I-P)(I-P) = I^2 - P - P + P^2 = I - 2P + P = I - P \tag{1.15}$$

y se cumple la relación:

$$\text{Null}(P) = \text{Ran}(I-P). \tag{1.16}$$

Para ver $\subseteq$. Sea $v \in Null(P)$ entonces Pv=0. Para ver que $v \in Ran(I-P)$ se substituye, $(I-P)v = v - P(v) = v - 0 = v$ entonces $v \in Ran(I-P)$.
Para ver $\supseteq$. Sea $v \in Ran(I-P)$, entonces existe un $x \in \mathbb{C}^n$ tal que $v = (I-P)x$. Veamos a que equivale Pv:

$$Pv = P(I-P)x = (P - P^2)x = (P-P)x = 0$$

entonces $v \in Null(P)$

Además, los dos subespacios Null(P) y Ran(P) intersecan solo en el elemento cero. De hecho, si un vector x pertenece a Ran(P), entonces $Px = x$ por idempotencia. Si además x pertenece a Null(P), entonces $Px = 0$. Por tanto, $x = 0$.

Por otro lado, cada elemento de $\mathbb{C}^n$ se puede escribir como:

$$x = Px + (I - P)x.$$

Por tanto, $\mathbb{C}^n$ se descompone como suma directa:

$$\mathbb{C}^n = \mathrm{Null}(P) \oplus \mathrm{Ran}(P).$$

Recíprocamente, cualquier par de subespacios M y S que forman una suma directa de $\mathbb{C}^n$ definen un único proyector tal que:

$$\mathrm{Ran}(P) = M, \qquad \mathrm{Null}(P) = S.$$

Este proyector asociado P asigna a un elemento x de $\mathbb{C}^n$ su componente x_1 en M, en la descomposición: $x = x_1 + x_2, \quad x_1 \in M, \quad x_2 \in S$.

Así, un proyector P queda determinado por:

$$Px \in M, \quad x - Px \in S.$$

El operador lineal P se dice que proyecta x sobre M y a lo largo de S.

Si P es de rango m, entonces $\dim(S) = n - m$. Resulta natural definir S a través de su complemento ortogonal $L = S^{\perp}$, de dimensión m.

Así, las condiciones que definen $u = Px$ se expresan como:

$$u \in M, \tag{1.17}$$

$$x - u \perp L. \tag{1.18}$$

Estas ecuaciones definen un proyector P sobre M y ortogonal a L. La primera establece m grados de libertad y la segunda ecuación impone m restriciones que permiten definir Px a partir de los grados de libertad.

Surge ahora la cuestión: dado un par de subespacios M y L de dimensión m, ¿es siempre posible definir un proyector sobre M y ortogonal a L? El siguiente lema responde a esta pregunta:

Lema 1.4. *Dados dos subespacios M y L de la misma dimensión m, las siguientes condiciones son equivalentes:*

(i) Ningún vector no nulo de M es ortogonal a L.

(ii) Para cada $x \in \mathbb{C}^n$ existe un único vector u que satisface las condiciones (1.17) *y* (1.18).

Demostración. La condición (i) establece que $M \cap L^\perp = \{0\}$. Dado que $\dim(L^\perp) = n - m$, se deduce que:

$$\mathbb{C}^n = M \oplus L^\perp.$$

Esto implica que cualquier x se puede escribir de forma única como:

$$x = u + w, \quad u \in M, \quad w \in L^\perp,$$

lo que prueba (ii). □

En resumen, dados M y L tales que $M \cap L^\perp = \{0\}$, existe un único proyector P sobre M y ortogonal a L, que cumple:

$$\mathrm{Ran}(P) = M, \qquad \mathrm{Null}(P) = L^\perp.$$

En particular, la condición $Px = 0$ equivale a $x \in L^\perp$, es decir:

$$Px = 0 \quad \Longleftrightarrow \quad x \perp L. \tag{1.19}$$

1.7.2. Representaciones matriciales de un proyector

Sean $M = \mathrm{Ran}(P)$ y $L = \mathrm{Null}(P)$ los subespacios rango y núcleo, respectivamente, de un proyector $P : C^{n\times n} \to C^{n\times n}$. Para obtener una representación matricial se necesitan dos bases de M y L

$$V = \big[v_1, \ldots, v_m\big], \quad W = \big[w_1, \ldots, w_m\big]$$

Se dice que estas bases son biortogonales si

$$(v_i, w_j) = \delta_{ij}, \qquad i, j = 1, \ldots, m. \tag{1.20}$$

donde δ_{ij} es la delta de Kronecker.

Sea $x \in \mathbb{C}^n$ y supóngase que su proyección se expresa como $Px = Vy$ en la base V. La condición de ortogonalidad $x - Px \perp L$ equivale, para cada $j = 1, \ldots, m$, a

$$\big(x - Vy,\, w_j\big) = 0.$$

En forma matricial:

$$W^H\big(x - Vy\big) = 0. \tag{1.21}$$

Si se cumple (1.20) (esto es, $W^H V = I$), de (1.21) se deduce $y = W^H x$; por tanto,

$$P \;=\; V\,W^H. \tag{1.22}$$

Cuando V y W no son biortogonales, la ecuación (1.21) conduce a

$$P \;=\; V\big(W^H V\big)^{-1} W^H, \tag{1.23}$$

Bajo el supuesto de que ningún vector de M sea ortogonal a L, puede demostrarse que $W^H V$ es invertible, garantizando así la validez de (1.23).

1.7.3. Proyectores ortogonales y oblicuos

Sea $P : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ un proyector ($P^2 = P$). En el marco de los métodos de proyección se distinguen dos casos según los subespacios

$$M := \mathrm{Ran}(P), \qquad L := \mathrm{Null}(P)^{\perp}.$$

Se dice que P es un proyector ortogonal sobre M cuando el subespacio de restricciones coincide con el de búsqueda, es decir

$$\mathrm{Null}(P) = \mathrm{Ran}(P)^{\perp}.$$

Equivalentemente, la condición que caracteriza un proyector ortogonal es: para todo $x \in \mathbb{C}^n$ se verifica

$$Px \in M, \qquad (I - P)x \;\perp\; M. \tag{1.24}$$

Si (1.24) no se cumple, P se llama proyector oblicuo.

Sea P^H el adjunto respecto al producto hermitiano,

$$(P^H x, y) = (x, Py), \qquad \forall\, x, y \in \mathbb{C}^n. \tag{1.25}$$

Entonces P^H es también un proyector y

$$\mathrm{Null}(P^H) = \mathrm{Ran}(P)^{\perp}, \tag{1.26}$$

$$\mathrm{Null}(P) = \mathrm{Ran}(P^H)^{\perp}. \tag{1.27}$$

Proposición 1.5. *Un proyector P es ortogonal si y sólo si es Hermitiano, esto es $P = P^H$.*

Demostración. Si P es ortogonal entonces $\text{Null}(P) = \text{Ran}(P)^\perp$. Las igualdades (1.26) y (1.27) implican $\text{Null}(P) = \text{Null}(P^H)$ y $\text{Ran}(P) = \text{Ran}(P^H)$. Como los proyectores quedan totalmente determinados por rango y nulo, se obtiene $P = P^H$.

Recíprocamente, si $P = P^H$, de (1.27) se deduce $\text{Null}(P) = \text{Ran}(P)^\perp$; por lo tanto P es ortogonal. □

1.7.4. Propiedades de los proyectores ortogonales

Sea P un proyector ortogonal, es decir, $P^2 = P$ y $P = P^H$. Para todo $x \in \mathbb{C}^n$ puede descomponerse $x = Px + (I - P)x$ en dos componentes ortogonales; de ahí se deduce

$$\|x\|_2^2 = \|Px\|_2^2 + \|(I - P)x\|_2^2. \tag{1.28}$$

Una consecuencia inmediata de (1.28) es la cota

$$\|Px\|_2 \;\leq\; \|x\|_2, \qquad \forall\, x \in \mathbb{C}^n, \tag{1.29}$$

de donde se obtiene $\|P\|_2 = 1$. Además, la matriz P sólo tiene dos autovalores: 1 (con subespacio propio $\text{Ran}(P)$) y 0 (con subespacio propio $\text{Null}(P)$).

A continuación, se da un resultado importante sobre la optimalidad de los proyectores ortogonales.

Teorema 1.6 (Propiedad de optimalidad)**.** *Sea P el proyector ortogonal sobre un subespacio $M \subset \mathbb{C}^n$. Entonces, para todo vector $x \in \mathbb{C}^n$,*

$$\min_{y \in M} \|x - y\|_2 \;=\; \|x - Px\|_2. \tag{1.30}$$

Demostración. Para $y \in M$ considérese $\|x - y\|_2^2$. Como $x - Px$ es ortogonal a M, en particular a $y - Px$, se cumple

$$\|x - y\|_2^2 \;=\; \|x - Px + (Px - y)\|_2^2 \;=\; \|x - Px\|_2^2 + \|Px - y\|_2^2 \;\geq\; \|x - Px\|_2^2.$$

La igualdad se alcanza en $y = Px$, con lo que se prueba (1.30). □

Corolario 1.7. *Sea $M \subset \mathbb{C}^n$ un subespacio y $x \in \mathbb{C}^n$. Se tiene*

$$\min_{y \in M} \|x - y\|_2 = \|x - y^*\|_2,$$

si y sólo si se verifican las condiciones

$$y^* \in M, \qquad x - y^* \;\perp\; M.$$

El corolario es una reformulación del teorema anterior en términos de condiciones necesarias y suficientes para que y^* sea la mejor aproximación de x en norma euclidiana sobre M.

1.8. Conceptos básicos en sistemas lineales

Los sistemas lineales son uno de los problemas más importantes y comunes en el cálculo científico. Desde el punto de vista teórico, se comprende bien cuándo existe una solución, cuándo no existe, y cuándo hay infinitas soluciones.

Se considera el sistema lineal:

$$Ax = b, \tag{1.31}$$

donde x es la incógnita y b es el término independiente.

Cuando se resuelve el sistema, se distinguen tres situaciones:

- Caso 1: La matriz A es no singular. Entonces existe una única solución dada por:

$$x = A^{-1}b.$$

- Caso 2: La matriz A es singular y $b \in \text{Ran}(A)$. Como b pertenece a la imagen de A, existe un x_0 tal que:

$$Ax_0 = b.$$

  Entonces, para cualquier $v \in \text{Null}(A)$, el vector $x_0 + v$ también satisface:

$$A(x_0 + v) = b.$$

  Dado que el núcleo de A es al menos de dimensión uno, existen infinitas soluciones.

- Caso 3: La matriz A es singular y $b \notin \text{Ran}(A)$. En este caso, no existe ninguna solución.

Para la resolución de sistemas de ecuaciones lineales se distinguen dos tipos de métodos. Por un lado, están los métodos directos que son aquellos que, en un número finito de operaciones, permiten obtener la solución exacta del sistema salvo errores de redondeo. Ejemplos de métodos directos son la eliminación de Gauss, la descomposición de Cholesky o la descomposición LU entre otros. Por otro lado, se tiene a los métodos iterativos que son técnicas que pretenden resolver sistemas de ecuaciones lineales a través de aproximaciones sucesivas de su solución. Son especialmente útiles para sistemas grandes o con matrices dispersas (con muchos ceros), donde los métodos directos pueden presentar problemas de almacenamiento de memoria. Dentro de los métodos iterativos existen dos grupos diferenciados. Los métodos iterativos estacionarios como Jacobi, Gauss-Seidel o SOR, que calculan la aproximación en cada iteración mediante una fórmula fija.La convergencia de estos métodos puede ser lenta y suele depender del uso de un precondicionador. Por otro lado, se distinguen los métodos iterativos no estacionarios como CG,

BCG, COCG o GMRES; estos métodos obtienen la aproximación en cada iteración a partir de información obtenida en iteraciones previas por lo que la fórmula para aproximar la solución va cambiando a lo largo del proceso iterativo. Suelen ser métodos con una convergencia mucho más rápida que la de los métodos estacionarios, a cambio de un mayor trabajo algebraico y la necesidad de multiplicaciones repetidas matriz-vector.

Capítulo 2

Métodos de Krylov. Sistemas con matrices reales

El capítulo siguiente explora los métodos de Krylov que se consideran actualmente entre las técnicas iterativas más importantes disponibles para la resolución de grandes sistemas lineales. Estas técnicas se basan en procesos de proyección, tanto ortogonales como oblicuos, sobre subespacios de Krylov, que serán descritos posteriormente. Este capítulo abarca los métodos de proyección de manera introductoria y los métodos derivados o relacionados con el método de ortogonalización de Arnoldi y con el método de biortogonalización de Lanczos.

2.1. Métodos de proyección

La mayoría de las técnicas iterativas existentes para resolver sistemas de ecuaciones lineales utilizan procesos de proyección. Esta sección describe estas técnicas y presenta los métodos de proyección para entender los métodos de Krylov.
Antes de empezar con los métodos, es necesario comentar lo siguiente: este trabajo está destinado a la resolución de ecuaciones lineales con matrices complejas no hermitianas, sin embargo para adentrarnos de forma más didáctica en los métodos se va a trabajar con sistemas lineales reales y después se extenderán al caso complejo.

Se considera el sistema lineal

$$A\,x \;=\; b, \tag{2.1}$$

donde A es una matriz real $n \times n$. En lo que sigue se utilizará el mismo símbolo A para referirnos tanto a la matriz como a la aplicación lineal $A:\ \mathbb{R}^n \to \mathbb{R}^n$.

La idea central de las técnicas de proyección consiste en extraer una solución aproximada de

(2.1) de un subespacio de $\mathbb{R}^n$. Sea $\mathcal{K}$ un subespacio de aproximaciones de la solución candidatas (también llamado subespacio de búsqueda) de dimensión m. Para poder determinar una aproximación dentro de $\mathcal{K}$ se imponen, en general, m condiciones de ortogonalidad independientes. Se exige que el vector residual $r = b - Ax$ sea ortogonal a m vectores linealmente independientes; dichos vectores generan otro subespacio $\mathcal{L}$ (de dimensión m) que se denominará subespacio de restricciones.

Existen dos grandes familias de métodos de proyección:

- Proyección ortogonal: $\mathcal{L} = \mathcal{K}$.
- Proyección oblicua: $\mathcal{L} \neq \mathcal{K}$.

2.1.1. Métodos de proyección generales

Sean $\mathcal{K}$ y $\mathcal{L}$ dos subespacios de $\mathbb{R}^n$ de dimensión m. Una técnica de proyección sobre el subespacio $\mathcal{K}$ y ortogonal a $\mathcal{L}$ es un proceso que busca una solución aproximada $\tilde{x}$ de (2.1) que satisfaga:

$$\tilde{x} \in \mathcal{K}, \qquad b - A\tilde{x} \ \perp \ \mathcal{L} \tag{2.2}$$

es decir, la solución aproximada tiene que pertencer al subespacio de búsqueda y el residuo de la solución aproximada tiene que ser ortogonal al subespacio de restricciones.

Si se parte de una aproximación inicial x_0 para la solución, entonces la aproximación debe buscarse en el espacio afín $x_0 + \mathcal{K}$ en lugar del espacio vectorial $\mathcal{K}$. Esto requiere una ligera modificación de la formulación anterior. El problema aproximado debe reformularse como

$$\tilde{x} \in x_0 + \mathcal{K}, \qquad b - A\tilde{x} \ \perp \ \mathcal{L} \ . \tag{2.3}$$

Escribiendo $\tilde{x} = x_0 + \delta$, $\delta \in \mathcal{K}$ y definiendo el residuo inicial

$$r_0 \ = \ b - Ax_0, \tag{2.4}$$

se tiene que $b - A\tilde{x} = b - Ax_0 - A\delta = r_0 - A\delta$ y entonces la condición (2.3) se convierte en

$$r_0 - A\,\delta \ \perp \ \mathcal{L}, \quad \delta \in \mathcal{K}.$$

En otras palabras, la solución aproximada $\tilde{x}$ está caracterizada por

$$\tilde{x} = x_0 + \delta, \quad \delta \in \mathcal{K}, \tag{2.5}$$

$$(\, r_0 - A\delta,\, w) = 0, \quad \forall\, w \in \mathcal{L}. \tag{2.6}$$

Lo anteriormente descrito es a lo que se le llama paso de proyección. La búsqueda de la solución aproximada de (2.1) es un proceso iterativo en el cual se van dando pasos de proyección, que se efectúan hasta llegar a una aproximación buena, según la tolerancia dada.

2.1.2. Representación matricial

Sean

$$V = [v_1, \ldots, v_m] \in \mathbb{R}^{n\times m}, \qquad W = [w_1, \ldots, w_m] \in \mathbb{R}^{n\times m},$$

matrices cuyas columnas constituyen bases de los subespacios $\mathcal{K}$ y $\mathcal{L}$, respectivamente. Si la solución aproximada se escribe como

$$\tilde{x} = x_0 + V\, y,$$

entonces la condición de ortogonalidad (2.6) conduce inmediatamente al sistema lineal para el vector de coordenadas y:

$$W^{\mathrm{T}} AV\, y \;=\; W^{\mathrm{T}} r_0.$$

Suponiendo que la matriz $B = W^{\mathrm{T}}AV \in \mathbb{R}^{m\times m}$ es no singular y que $y = B^{-1}W^T r_0$, la expresión de la solución aproximada resulta

$$\tilde{x} = x_0 + V\, B^{-1} W^{\mathrm{T}} r_0\ . \tag{2.7}$$

A continuación, se da un algoritmo prototipo de un método de proyección donde se busca iterativamente la solución del sistema lineal, a través de las bases de los subespacios $\mathcal{K}$ y $\mathcal{L}$.

Algoritmo 2.1. *Algoritmo prototipo de proyección*

1: *Until convergence, Do:*
2: *Select a pair of subspaces* $\mathcal{K}$ *and* $\mathcal{L}$
3: *Choose bases* $V = [v_1, \ldots, v_m]$ *and* $W = [w_1, \ldots, w_m]$ *for* $\mathcal{K}$ *and* $\mathcal{L}$
4: $r \leftarrow b - Ax$ ▷ *residuo actual*
5: $y \leftarrow (W^T AV)^{-1} W^T r$ ▷ B=$W^T AV$
6: $x \leftarrow x + Vy$
7: *EndDo*

Es necesario destacar que la solución aproximada existirá cuando la matriz B sea no singular, dicha propiedad no es cierta en general, ni siquiera suponiendo que A es no singular.

Proposición 2.2. *Sea* $A \in \mathbb{R}^{n\times n}$ *y sean* $\mathcal{K}, \mathcal{L} \subset \mathbb{R}^n$ *subespacios de dimensión* m. *Sea* V *(resp.* W*) cualquier base de* $\mathcal{K}$ *(resp.* $\mathcal{L}$*). Entonces la matriz* $B = W^{\mathrm{T}}AV$ *es no singular siempre que se cumpla al menos una de las siguientes condiciones:*

1. *A es definida positiva y* $\mathcal{L} = \mathcal{K}$ *(proyección ortogonal);*
2. *A es no singular y* $\mathcal{L} = A\mathcal{K}$.

Demostración. Caso (i). Puesto que $\mathcal{L} = \mathcal{K}$, existe una matriz no singular $G \in \mathbb{R}^{m\times m}$ tal que $W = VG$. Así,

$$B = W^{\mathrm{T}}AV = G^{\mathrm{T}}V^{\mathrm{T}}AV.$$

Como A es definida positiva, $V^{\mathrm{T}}AV$ lo es también y, por tanto, no singular; lo mismo ocurre con B.

Caso (ii). Ahora $W = AVG$ para cierta matriz no singular G. Se obtiene

$$B = W^{\mathrm{T}}AV = G^{\mathrm{T}}(AV)^{\mathrm{T}}AV.$$

El producto $AV \in \mathbb{R}^{n\times m}$ tiene rango completo porque A es no singular y las columnas de V son independientes; así $(AV)^{\mathrm{T}}AV$ es no singular, y por consiguiente también B. □

Obsérvese que, si además A es simétrica y se emplea una proyección ortogonal ($\mathcal{L} = \mathcal{K}$), la matriz proyectada $B = V^{\mathrm{T}}AV$ conserva la simetría; si A es simétrica definida positiva (SPD), entonces B lo es también.

2.2. Introducción a los métodos de subespacios de Krylov

Se recuerda, de la sección anterior, que un método de proyección general para resolver el sistema lineal dado por (2.1), extrae una solución aproximada x_m del subespacio afín $x_0 + \mathcal{K}_m$ de dimensión m imponiendo la condición de Petrov–Galerkin:

$$b - Ax_m \;\perp\; \mathcal{L}_m, \tag{2.8}$$

donde $\mathcal{L}_m$ es otro subespacio de dimensión m y x_0 representa una estimación inicial arbitraria de la solución.

Un método de subespacios de Krylov es aquel para el cual el espacio $\mathcal{K}_m$ es el subespacio de Krylov

$$\mathcal{K}_m(A, r_0) \;=\; \mathrm{span}\{\, r_0,\, Ar_0,\, A^2r_0, \ldots, A^{m-1}r_0\}, \tag{2.9}$$

siendo $r_0 = b - Ax_0$. Se denota $\mathcal{K}_m(A, r_0)$ simplemente por $\mathcal{K}_m$.

Desde el punto de vista de la teoría de la aproximación, las soluciones x_m obtenidas por un método de Krylov satisfacen

$$A^{-1}b \;\approx\; x_m = x_0 + q_{m-1}(A)\, r_0, \tag{2.10}$$

donde q_{m-1} es un polinomio de grado $m-1$. En el caso particular $x_0 = 0$ la relación se simplifica trivialmente a $A^{-1}b \approx q_{m-1}(A)b$.

Aunque todas las técnicas anteriores proporcionan el mismo tipo de aproximaciones polinómicas, la selección de $\mathcal{L}_m$, el subespacio de restricciones para construir la aproximación, tiene un efecto fundamental sobre el método iterativo resultante.

Existen dos elecciones clásicas que conducen a los métodos más conocidos: la condición $\mathcal{L}_m = \mathcal{K}_m$ y la condición asociada a A^T, es decir, $\mathcal{L}_m = \mathcal{K}_m(A^T, r_0)$.

2.3. Propiedades elementales de los subespacios de Krylov

En esta sección se estudian los métodos de proyección cuyos espacios de búsqueda son subespacios de Krylov, concretamente, subespacios de la forma

$$\mathcal{K}_m(A, v) \;=\; \text{span}\{\, v,\; Av,\; A^2v, \ldots, A^{m-1}v\}. \tag{2.11}$$

La dimensión de $\mathcal{K}_m$ crece en una unidad por paso del proceso iterativo.

Se pueden establecer algunas propiedades elementales de los subespacios de Krylov. Una primera propiedad es que $\mathcal{K}_m$ es el subespacio de todos los vectores en $\mathbb{R}^n$, que puede escribirse como $x = p(A)v$, donde p es un polinomio de grado no mayor que $m-1$. Se recuerda que el polinomio minimal de un vector v es el polinomio mónico distinto de cero, p, de menor grado tal que $p(A)v = 0$. El grado del polinomio minimal de v con respecto a A se suele denominar grado de v con respecto a A, o simplemente $grad(v)$. Una consecuencia del teorema de Cayley-Hamilton[1] es que el grado de v no es nunca mayor que n.

Un primer resultado relaciona $\mathcal{K}_m$ con el polinomio mínimo de un vector.

Proposición 2.3. *Sea μ el grado de v respecto a A, esto es, el grado del polinomio mínimo de v. Entonces*

1. *$\mathcal{K}_\mu$ es invariante por A, es decir, $Av \in \mathcal{K}_\mu \; \forall v \in \mathcal{K}_\mu$*
2. *$\mathcal{K}_m = \mathcal{K}_\mu \;\; \forall\, m \geq \mu$.*

Asimismo, la dimensión de $\mathcal{K}_m$ no decrece con m; de hecho la siguiente proposición determina la dimension de $\mathcal{K}_m$ en general.

[1]El teorema de Cayley-Hamilton asegura que todo endomorfismo de un espacio vectorial de dimensión finita sobre un cuerpo cualquiera anula su propio polinomio caracterísitco. En términos matriciales, significa que si A es una matriz cuadrada de orden n y $p(\lambda) = \lambda^n + p_{n-1}\lambda^{n-1} + \cdots + p_1\lambda + p_0$ es su polinomio característico entonces al substituir λ por A el resultado es la matriz nula, $p(A) = A^n + p_{n-1}A^{n-1} + \cdots + p_1A + p_0 = 0$.

Proposición 2.4. *El subespacio de Krylov $\mathcal{K}_m$ tiene dimensión m si, y sólo si, grad(v) $\geq m$. En general,*

$$\dim \mathcal{K}_m \;=\; \text{mín}\{\, m,\; grad(v)\}. \tag{2.12}$$

Demostración. Los vectores $\{v, Av, \ldots, A^{m-1}v\}$ son linealmente independientes si, y sólo si, la combinación $\sum_{i=0}^{m-1} \alpha_i A^i v$ no puede anularse salvo que $\alpha_0 = \cdots = \alpha_{m-1} = 0$; esto equivale a que no exista un polinomio p tal que $p \neq 0$ y $grad(p) \leq m-1$ que anule a v, lo cual sucede exactamente cuando grad(v) $\geq m$. Tomando la mínima de las dos cantidades se obtiene (2.12). □

2.4. Método de Arnoldi

El método de Arnoldi es una técnica de proyección ortogonal sobre el subespacio de Krylov $\mathcal{K}_m$ cuando la matriz A es, en general, no hermitiana. Presentado inicialmente como un procedimiento para reducir una matriz densa a forma de Hessenberg mediante transformaciones unitarias, Arnoldi observó que los autovalores de la matriz de Hessenberg H_m asociada (para valores moderados de m) aproximan correctamente a algunos autovalores de A. Posteriormente se descubrió que esta estrategia resulta muy eficiente para aproximar autovalores de matrices dispersas de gran tamaño y, más tarde, se extendió a la resolución de sistemas lineales de gran tamaño.

En esta sección se describe primero el método en aritmética exacta y se establecen sus propiedades básicas.

2.4.1. Método de Arnoldi para obtener una base ortonormal de $\mathcal{K}_m$

El procedimiento de Arnoldi tiene como objetivo generar una base ortonormal $\{v_1, \ldots, v_m\}$ de $\mathcal{K}_m$. Una variante clásica, usando Gram–Schmidt, se resume en el Algoritmo 2.5.

Algoritmo 2.5. *Arnoldi*

Choose a vector v_1 such that $\|v_1\|_2 = 1$
for $j = 1, 2, \ldots, m$ ***do***
    $h_{ij} \leftarrow (Av_j, v_i)$
    for $i = 1, \ldots, j$ ***do***
        $w_j \leftarrow Av_j - \sum_{i=1}^{j} h_{ij} v_i$
        $h_{j+1,j} \leftarrow \|w_j\|_2$
        if $h_{j+1,j} = 0$ ***then***
            stop

9: ***end if***
10: $v_{j+1} \leftarrow w_j/h_{j+1,j}$
11: ***end for***

En cada paso se multiplica el vector v_j por A y se ortogonaliza el resultado frente a los v_i ya construidos (método de Gram–Schmidt). Si en la línea 5 el vector w_j se anula, el algoritmo sufre *breakdown* y finaliza prematuramente.

Algunas propiedades esenciales del método de Arnoldi se recogen en la proposición siguiente.

Proposición 2.6. *Supóngase que el Algoritmo 2.5 no se detiene antes del paso m. Entonces los vectores* $\{v_1, \ldots, v_m\}$ *forman una base ortonormal de*

$$\mathcal{K}_m = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}.$$

Demostración. Los vectores v_j son ortonormales por construcción. Para demostrar que generan $\mathcal{K}_m$, nótese que cada v_j puede escribirse como $v_j = q_{j-1}(A)v_1$, donde q_{j-1} es un polinomio de grado $j-1$. Esto se puede probar mediante inducción en j: Para $j = 1$ es cierto, $v_1 = q_0(A)v_1$, con $q_0(t) \equiv 1$. Se asume cierto para todo j, y se considera v_{j+1}. Por tanto se tiene que:

$$h_{j+1,j}v_{j+1} = Av_j - \sum_{i=1}^{j} h_{i,j}v_i = Aq_{j-1}(A)v_1 - \sum_{i=1}^{j} h_{i,j}q_{i-1}(A)v_1 \tag{2.13}$$

v_{j+1} se puede expresar como $q_j(A)v_1$, siendo q_j polinomio de grado j. □

Proposición 2.7. *Sea* $V_m = [v_1, \ldots, v_m] \in \mathbb{C}^{n\times m}$ *la matriz cuyas columnas son los vectores de Arnoldi obtenidos tras m pasos, y sea* $\widetilde{H}_m \in \mathbb{C}^{(m+1)\times m}$ *la matriz de Hessenberg cuyos elementos no nulos* h_{ij} *están definidos en el Algoritmo 2.5. Denótese por* H_m *la matriz que se obtiene al eliminar la última fila de* $\widetilde{H}_m$*. Entonces se verifican las igualdades*

$$AV_m = V_m H_m + w_m e_m^\top, \tag{2.14}$$
$$= V_{m+1}\widetilde{H}_m, \tag{2.15}$$
$$V_m^\top A V_m = H_m. \tag{2.16}$$

Demostración. La igualdad (2.15) se deduce directamente de las líneas 5, 6 y 10 del Algoritmo 2.5, que proporcionan

$$Av_j = \sum_{i=1}^{j+1} h_{ij}\, v_i, \qquad j = 1, 2, \ldots, m. \tag{2.17}$$

La relación (2.14) no es más que una reformulación matricial de (2.17). Por último, (2.16) se obtiene al multiplicar (2.14) a izquierda por V_m^T y emplear la ortonormalidad de los vectores $\{v_1, \ldots, v_m\}$. □

Tal como se mencionó anteriormente, el algoritmo puede sufrir un *breakdown* si la norma de w_j se anula en alguna interación. En ese caso el vector v_{j+1} no puede calcularse y el proceso se detiene.

Proposición 2.8. *El algoritmo de Arnoldi se detiene en el paso j (es decir, $h_{j+1,j} = 0$ en la línea 6 del Algoritmo 2.5) si y sólo si el polinomio mínimo de v_1 tiene grado j. En tal caso, el subespacio $\mathcal{K}_j$ es invariante por A.*

Un resultado importante derivado de la proposición anterior es que el método de proyección en el subespacio $\mathcal{K}_j$ será exacto cuando en el paso j sufra un *breakdown*. Es por esto que dichos *breakdowns* se llamarán *lucky breakdowns*.

Usando el algoritmo de Gram-Schmidt modificado el algoritmo adopta la siguiente forma:

Algoritmo 2.9. *Arnoldi con Gram–Schmidt modificado*

1: *Choose a vector* v_1 *of norm* 1
2: ***for*** $j = 1, 2, \ldots, m$ ***do***
3: *Compute* $w_j := Av_j$
4: ***for*** $i = 1, \ldots, j$ ***do***
5: $h_{ij} = (w_j, v_i)$
6: $w_j := w_j - h_{ij}v_i$
7: ***end for***
8: $h_{j+1,j} = \|w_j\|_2$. ***If*** $h_{j+1,j} = 0$ ***Stop***
9: $v_{j+1} := w_j/h_{j+1,j}$
10: ***end for***=0

En aritmética exacta, el Algoritmo 2.9 y el Algoritmo 2.5 son matemáticamente equivalentes. Sin embargo, la formulación usando Gram-Schmidt modificado resulta mucho más estable numéricamente.

2.4.2. Método de Arnoldi para sistemas lineales (FOM)

Sea x_0 una aproximación inicial para el sistema lineal original (2.1) y considérese ahora un método de proyección ortogonal como se definió anteriormente, tomando $\mathcal{L} = \mathcal{K} = \mathcal{K}_m(A, r_0)$, con $r_0 = b - Ax_0$. El método busca una solución aproximada x_m en el subespacio $x_0 + \mathcal{K}_m$ de dimensión m imponiendo la condición de Galerkin (2.8).

Si en el proceso de Arnoldi se parte de $v_1 = r_0/\|r_0\|_2$ y se define $\beta = \|r_0\|_2$, se obtiene (usando (2.16))

$$V_m^{\mathrm{T}} A V_m = H_m, \qquad V_m^{\mathrm{T}} r_0 = V_m^{\mathrm{T}}(\beta v_1) = \beta e_1.$$

Como resultado, la solución aproximada definida por los subespacios de dimensión m anteriores viene dada por

$$x_m = x_0 + V_m y_m, \tag{2.18}$$

$$y_m = H_m^{-1}(\beta e_1). \tag{2.19}$$

Basado en esta idea se describe a continuación el Método de Ortogonalización Completa (Full Orthogonalization Method, FOM). En el procedimiento se utiliza el Algoritmo 2.9.

Algoritmo 2.10. *Método de ortogonalización completa para sistemas lineales (FOM)*

1: *Compute* $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$, *and* $v_1 := r_0/\beta$
2: *Define the* $m \times m$ *matrix* $H_m = \{h_{ij}\}_{i,j=1,\ldots,m}$; ***set*** $H_m = 0$
3: ***for*** $j = 1, 2, \ldots, m$ ***do***
4: *Compute* $w_j := Av_j$
5: ***for*** $i = 1, \ldots, j$ ***do***
6: $h_{ij} = (w_j, v_i)$
7: $w_j := w_j - h_{ij} v_i$
8: ***end for***
9: $h_{j+1,j} = \|w_j\|_2$. ***If*** $h_{j+1,j} = 0$ *set* $m := j$ *and Go to 12*
10: $v_{j+1} = w_j / h_{j+1,j}$
11: ***end for***
12: *Compute* $y_m = H_m^{-1}(\beta e_1)$ *and* $x_m = x_0 + V_m y_m$

El Algoritmo 2.10 depende del parametro m, que es la dimensión del subespacio de Krylov. En la práctica, es deseable seleccionar m de forma dinámica. Esto sería posible si la norma residual de la solución x_m estuviera disponible a bajo costo (sin tener que calcular explícitamente x_m). En ese caso, el algoritmo puede detenerse en el paso apropiado utilizando esta información. La siguiente proposición proporciona un resultado relacionado.

Proposición 2.11 (Residuo en FOM)**.** *Sea* x_m *la solución aproximada obtenida mediante el Algoritmo FOM. Entonces el vector residuo satisface*

$$b - Ax_m \;=\; -h_{m+1,m}\, e_m^{\mathrm{T}} y_m \, v_{m+1},$$

y, por consiguiente,

$$\big\|b - Ax_m\big\|_2 \;=\; h_{m+1,m}\, |e_m^{\mathrm{T}} y_m|. \tag{2.20}$$

Demostración. Se tienen las igualdades

$$\begin{aligned} b - Ax_m &= b - A\big(x_0 + V_m y_m\big) \\ &= r_0 - AV_m y_m \\ &= \beta v_1 \;-\; V_m H_m y_m \;-\; h_{m+1,m}\, e_m^{\mathrm{T}} y_m \, v_{m+1}. \end{aligned}$$

Por definición de y_m, se cumple $H_m y_m = \beta e_1$, luego $\beta v_1 - V_m H_m y_m = 0$, y el resultado se deduce inmediatamente. □

2.5. Método de Lanczos simétrico

El método de Lanczos simétrico puede interpretarse como una simplificación del método de Arnoldi para el caso particular en el que la matriz A es simétrica. Cuando A es simétrica, la matriz de Hessenberg H_m se vuelve tridiagonal simétrica.

Para presentar el algoritmo de Lanczos se parte del siguiente resultado dado en [5].

Teorema 2.12. *Supóngase que el método de Arnoldi se aplica a una matriz real simétrica A. Entonces los coeficientes h_{ij} generados por el algoritmo cumplen*

$$h_{ij} = 0, \qquad 1 \leq i < j - 1, \tag{2.21}$$

$$h_{j,j+1} = h_{j+1,j}, \qquad j = 1, 2, \ldots, m. \tag{2.22}$$

Dicho de otro modo, la matriz H_m obtenida en el proceso de Arnoldi es tridiagonal y simétrica.

Demostración. La demostración es consecuencia inmediata del hecho de que $H_m = V_m^T A V_m$ es una matriz simétrica (porque A lo es) y, por construcción, también es de Hessenberg. Por lo tanto, H_m debe ser tridiagonal simétrica. □

La notación estándar para describir el algoritmo de Lanczos se obtiene definiendo

$$\alpha_j \equiv h_{jj}, \qquad \beta_j \equiv h_{j-1,j},$$

y, si T_m denota la matriz H_m resultante, entonces es de la forma siguiente:

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}. \tag{2.23}$$

Todo esto conduce al siguiente algoritmo:

Algoritmo 2.13. *Método de Lanczos para obtener una base ortonormal*

1: ***Choose*** *an initial vector v_1 of 2-norm unity.* ***Set*** $\beta_1 \equiv 0$, $v_0 \equiv 0$
2: ***for*** $j = 1, 2, \ldots, m$ ***do***
3: $\quad w_j := A v_j - \beta_j v_{j-1}$

4: $\alpha_j := (w_j, v_j)$
5: $w_j := w_j - \alpha_j v_j$
6: $\beta_{j+1} := \|w_j\|_2$. ***If*** $\beta_{j+1} = 0$ ***then Stop***
7: $v_{j+1} := w_j/\beta_{j+1}$
8: ***end for***

Las principales diferencias prácticas con el método de Arnoldi son que la matriz H_m es tridiagonal y, sobre todo, que sólo es necesario almacenar tres vectores a la vez, a menos que se emplee algún tipo de re-ortogonalización.

En aritmética exacta, el núcleo del Algoritmo 2.13 viene dado por la relación

$$\beta_{j+1}\, v_{j+1} \;=\; Av_j \;-\; \alpha_j v_j \;-\; \beta_j v_{j-1}.$$

Esta recurrencia de tres términos recuerda a la relación estándar de tres términos que aparece en la teoría de los polinomios ortogonales, veáse en [5]. De hecho, existe un vínculo muy estrecho entre el algoritmo de Lanczos y dichos polinomios. Para verlo, baste recordar que, si el grado de v_1 es $\geq m$, entonces el subespacio $\mathcal{K}_m$ tiene dimensión m y está formado por todos los vectores de la forma $q(A)v_1$, donde q es un polinomio con $grado(q) \leq m-1$.

2.5.1. Método de Lanczos para sistemas lineales

Sea x_0 una estimación inicial de la solución de (2.1) y considérese el subespacio de Krylov $\mathcal{K}_m(r_0, A)$, con $r_0 = b - Ax_0$. Con los vectores de Lanczos v_i, $i = 1, \ldots, m$, y la matriz tridiagonal T_m procedentes del algoritmo de Lanczos, la proyección ortogonal sobre $\mathcal{K}_m$ proporciona la siguiente aproximación

$$x_m = x_0 + V_m\, y_m, \qquad y_m = T_m^{-1}(\beta e_1), \tag{2.24}$$

donde $\beta = \|r_0\|_2$.

Algoritmo 2.14. *Método de Lanczos para sistemas lineales*

1: *Compute* $r_0 = b - Ax_0$, $\beta := \|r_0\|_2$ *y* $v_1 := r_0/\beta$
2: ***for*** $j = 1, 2, \ldots, m$ ***do***
3: $w_j := Av_j - \beta_j v_{j-1}$ ▷ $\beta_1 \equiv 0$
4: $\alpha_j := (w_j, v_j)$
5: $w_j := w_j - \alpha_j v_j$
6: $\beta_{j+1} := \|w_j\|_2$. ***If*** $\beta_{j+1} = 0$ ***then Stop***
7: $v_{j+1} := w_j/\beta_{j+1}$
8: ***end for***
9: *Set* $T_m = \text{tridiag}(\beta_i, \alpha_i, \beta_{i+1})$ *y* $V_m = [v_1, \ldots, v_m]$

10: Compute $y_m = T_m^{-1}(\beta e_1)$ *y* $x_m = x_0 + V_m y_m$

Muchos de los resultados obtenidos para el método de Arnoldi siguen siendo válidos. Por ejemplo, el residuo de la solución aproximada x_m satisface lo que se vió en la Proposición 2.11

$$b - Ax_m = -\beta_{m+1} e_m^T y_m \, v_{m+1}. \tag{2.25}$$

2.6. Método de Gradiente Conjugado (CG)

El método de Gradiente Conjugado (Conjugate Gradient, CG) es una de las técnicas iterativas destacadas que se usan en este trabajo y se utiliza, generalmente, para resolver sistemas lineales donde A es una matriz simétrica y definida positiva. Dicho método puede verse como una proyección ortogonal sobre el subespacio de Krylov $\mathcal{K}_m(r_0, A)$, donde r_0 es el residuo inicial; desde un punto de vista matemático es equivalente al FOM, aunque la recurrencia de tres términos del proceso de Lanczos conduce a algoritmos más elegantes y económicos.

El algoritmo CG puede deducirse del procedimiento anterior factorizando $T_m = L_m U_m$, donde L_m es bidiagonal inferior unitaria y U_m es bidiagonal superior:

$$T_m = \begin{pmatrix} 1 & & & \\ \lambda_2 & 1 & & \\ & \lambda_3 & \ddots & \\ & & \ddots & 1 \end{pmatrix} \begin{pmatrix} \eta_1 & \beta_2 & & \\ & \eta_2 & \beta_3 & \\ & & \ddots & \beta_m \\ & & & \eta_m \end{pmatrix}.$$

La solución aproximada se reescribe como

$$x_m = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1),$$

y, definiendo

$$P_m := V_m U_m^{-1}, \qquad z_m := L_m^{-1} \beta e_1,$$

se tiene

$$x_m = x_0 + P_m z_m. \tag{2.26}$$

Debido a la estructura de U_m, el bloque P_m puede actualizarse explícitamente con facilidad. En efecto, al igualar las últimas columnas de la relación matricial $P_m U_m = V_m$ se obtiene lo que permite calcular el vector p_m a partir de los vectores p_i calculados a partir de $P_m := V_m U_m^{-1}$ y del vector v_m. Por consiguiente la última columna p_m de P_m puede actualizarse con

$$p_m = \eta_m^{-1}[v_m - \beta_m p_{m-1}],$$

donde β_m proviene de Lanczos y η_m es el último elemento de la diagonal de U_m que se calcula en el m-ésimo paso de eliminación de Gauss sobre la matriz tridiagonal tridiagonal, T_m:

$$\lambda_m = \frac{\beta_m}{\eta_{m-1}}, \tag{2.27}$$

$$\eta_m = \alpha_m - \lambda_m \beta_m. \tag{2.28}$$

Además, debido a la estructura de L_m, se verifica la relación

$$z_m \;=\; \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix}, \qquad \text{con} \quad \zeta_m = -\lambda_m \zeta_{m-1}.$$

A partir de la ecuación (2.26) se obtiene

$$x_m \;=\; x_0 \;+\; \begin{bmatrix} P_{m-1},\, p_m \end{bmatrix} \begin{bmatrix} z_{m-1} \\ \zeta_m \end{bmatrix} \;=\; x_0 \;+\; P_{m-1} z_{m-1} \;+\; \zeta_m\, p_m.$$

Puesto que $x_0 + P_{m-1} z_{m-1} = x_{m-1}$, se deduce que la aproximación x_m puede actualizarse en cada paso mediante la relación

$$x_m \;=\; x_{m-1} + \zeta_m\, p_m,$$

donde p_m se ha definido anteriormente. Esto conduce al siguiente algoritmo, denominado versión directa del algoritmo de Lanczos para sistemas lineales, que será equivalente al método Gradiente Conjugado cuando A es simétrica definida positiva. Este algoritmo recibe el nombre de D-Lanczos, y se detalla a continuación:

Algoritmo 2.15. *D-Lanczos*

1: *Compute* $r_0 = b - Ax_0$, $\zeta_1 := \beta := \|r_0\|_2$ *and* $v_1 := r_0/\beta$
2: *Set* $\lambda_1 = \beta_1 = 0$, $v_0 = 0$
3: ***for*** $m = 1, 2, \ldots$ *until convergence* ***do***
4: *Compute* $w := Av_m - \beta_m v_{m-1}$ *and* $\alpha_m = (w, v_m)$
5: ***if*** $m > 1$ ***then*** *compute* $\lambda_m = \dfrac{\beta_m}{\eta_{m-1}}$ *and* $\zeta_m = -\lambda_m \zeta_{m-1}$
6: ***end if***
7: $\eta_m = \alpha_m - \lambda_m \beta_m$
8: $p_m = \eta_m^{-1}\,(v_m - \beta_m p_{m-1})$
9: $x_m = x_{m-1} + \zeta_m p_m$
10: ***if*** r_m *has converged* ***then Stop***
11: ***end if***
12: $w := w - \alpha_m v_m$
13: $\beta_{m+1} = \|w\|_2$, $v_{m+1} = w/\beta_{m+1}$
14: ***end for***

El algoritmo anterior calcula la solución del sistema tridiagonal $T_m y_m = \beta e_1$ progresivamente, mediante eliminación de Gauss sin pivote.

Los Algoritmos 2.14 y 2.15 son matemáticamente equivalentes: si ambos se ejecutan sin incidencias, producen la misma solución aproximada. No obstante, al resolver explícitamente el sistema tridiagonal mediante eliminación gaussiana sin pivote, la versión directa puede ser más propensa a *breakdowns*.

Obsérvese que el vector residual de la aproximación x_m es proporcional a v_{m+1} (véase (2.25)); por tanto, los residuos resultan ortogonales entre sí y los vectores p_i forman un conjunto A-conjugado[2]. Estos hechos se recogen en la siguiente proposición.

Proposición 2.16. *Sea $r_m = b - Ax_m$, $m = 0, 1, \ldots,$ la sucesión de residuos generada por los Algoritmos 2.14 y 2.15 y sean p_m los vectores auxiliares del Algoritmo 2.15 Entonces:*

1. *Existe un escalar σ_m tal que $r_m = \sigma_m v_{m+1}$; en consecuencia, el conjunto de los residuos es ortogonal.*

2. *Los vectores $\{p_i\}$ son A-conjugados, esto es, $(Ap_i, p_j) = 0$ para $i \neq j$.*

Demostración. La primera parte es consecuencia inmediata de la relación (2.25) . Para la segunda, basta demostrar que $P_m^T A P_m$ es diagonal, donde $P_m = V_m U_m^{-1}$. En efecto,

$$P_m^T A P_m = U_m^{-T} V_m^T A V_m U_m^{-1} = U_m^{-T} T_m U_m^{-1} = U_m^{-T} L_m,$$

donde $T_m = V_m^T A V_m$ es tridiagonal simétrica y L_m es triangular inferior. Puesto que $U_m^{-T} L_m$ es simétrica y triangular inferior, necesariamente ha de ser diagonal, lo que prueba la ortogonalidad A-conjugada de los $\{p_i\}$. □

Imponiendo las condiciones de ortogonalidad y conjugación, se llega al algoritmo de Gradiente Conjugado (CG). En concreto, el nuevo vector x_{j+1} se escribe como:

$$x_{j+1} = x_j + \alpha_j p_j, \tag{2.29}$$

mientras que los residuos satisfacen la recurrencia

$$r_{j+1} = r_j - \alpha_j A p_j. \tag{2.30}$$

[2]Se dice que el conjunto $\{p_1, p_2, ..., p_n\}$ es un conjunto A-conjugado si los vectores son A-ortogonales, es decir,

$$p_i^H A p_j = 0 \quad \forall i \neq j$$

También se podrá denotar como $\langle p_i, p_j \rangle_A = (Ay, x) = 0$ siguiendo la notación de [7].

Imponiendo que $(r_{j+1}, r_j) = 0$, se obtiene

$$\alpha_j = \frac{(r_j, r_j)}{(Ap_j, r_j)}. \tag{2.31}$$

El vector de búsqueda p_{j+1} es una combinación lineal entre r_{j+1} y p_j, se define como

$$p_{j+1} = r_{j+1} + \beta_j p_j. \tag{2.32}$$

Por consiguiente, una primera consecuencia de la relación anterior es:

$$(Ap_j, r_j) = (Ap_j, p_j - \beta_{j-1} p_{j-1}) = (Ap_j, p_j)$$

esta igualdad se cumple debido a la A-conjugación de p_j y p_{j-1}. Por tanto, la igualdad (2.31) nos queda $\alpha_j = (r_j, r_j)/(Ap_j, p_j)$. Además escribiendo p_{j+1} como en (2.32) se tiene (gracias a la A-conjugación de p_{j+1} y p_j) que:

$$0 = (p_{j+1}, Ap_j) = (r_{j+1} + \beta_j p_j, Ap_j) = (r_{j+1}, Ap_j) + (\beta_j p_j, Ap_j).$$

Despejando β_j se llega a la siguiente igualdad:

$$\beta_j = -\frac{(r_{j+1}, Ap_j)}{(p_j, Ap_j)}.$$

Usando (2.30):

$$Ap_j = -\frac{1}{\alpha_j}(r_{j+1} - r_j) \tag{2.33}$$

y entonces se llega a la expresión simplificada de β_j

$$\beta_j = \frac{1}{\alpha_j}\frac{(r_{j+1}, r_{j+1} - r_j)}{(Ap_j, p_j)} = \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)}. \tag{2.34}$$

Sustituyendo estas relaciones se llega al algoritmo clásico de Gradiente Conjugado.

Algoritmo 2.17. *Método de Gradiente Conjugado*

1: *Compute* $r_0 \leftarrow b - Ax_0, \quad p_0 \leftarrow r_0$
2: ***for*** $j = 0, 1, \ldots$ *until convergence* ***do***
3: $\quad \alpha_j := (r_j, r_j)/(Ap_j, p_j)$
4: $\quad x_{j+1} := x_j + \alpha_j p_j$
5: $\quad r_{j+1} := r_j - \alpha_j Ap_j$
6: $\quad \beta_j := (r_{j+1}, r_{j+1})/(r_j, r_j)$
7: $\quad p_{j+1} := r_{j+1} + \beta_j p_j$
8: ***end for***

Los escalares α_j, β_j definidos en este algoritmo difieren de los que aparecen en el algoritmo de Lanczos. Los vectores p_j son múltiplos de los p_j en el Algoritmo 2.15. Desde el punto de vista del almacenamiento, además de la matriz A, basta guardar cuatro vectores (x, p, Ap, r) frente a los cinco vectores (v_m, v_{m-1}, w, p, x) que se requieren en el Algoritmo 2.15.

Por último, el método de Gradiente Conjugado combina a la vez características de los métodos directos e iterativos: genera una sucesión $\{x_j\}_{j\geq 0}$ que aproxima la solución exacta, y en ausencia de errores de redondeo se obtiene dicha solución tras, como máximo, n pasos; es decir, en aritmética exacta se comporta como un método directo. En la práctica, los residuos r_j que el algoritmo va calculando dejan de ser ortogonales debido al redondeo, de modo que la solución exacta no se obtiene al cabo de n pasos. Con el tiempo, esto ha hecho que el aspecto iterativo del método cobre mayor relevancia.

2.6.1. Convergencia del método CG

Para un vector x se define la A-norma de la forma

$$\|x\|_A \;=\; \big(Ax, x\big)^{1/2}.$$

El siguiente lema extraido de [5] caracteriza la aproximación que se obtiene tras m pasos del método de Gradiente Conjugado.

Lema 2.18. *Sea x_m la solución aproximada tras la m-ésima iteración del algoritmo CG y sea $d_m = x_* - x_m$ el error, donde x_* es la solución exacta. Entonces x_m puede escribirse como*

$$x_m \;=\; x_0 + q_m(A)r_0,$$

donde q_m es un polinomio de grado $m-1$ que verifica

$$\big\|(I - Aq_m(A))d_0\big\|_A \;=\; \min_{q\in\mathcal{P}_{m-1}} \big\|(I - Aq(A))d_0\big\|_A,$$

siendo $\mathcal{P}_{m-1}$ el conjunto de polinomios de grado $\leq m-1$.

A partir del lema anterior se obtiene el resultado clásico de cota de error cuando la matriz del sistema lineal (2.1) es simétrica y definida positiva:

Teorema 2.19. *Sea $A \in \mathbb{R}^{n\times n}$ simétrica y definida positiva con autovalores $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n > 0$. Sea x^* la solución exacta del sistema lineal $Ax = b$. Entonces, para los iterantes x_m generados por el método de Gradientes Conjugado, se cumple:*

$$\|x_m - x^*\|_A \;\leq\; \left(\tfrac{\lambda_{m+1}-\lambda_n}{\lambda_{m+1}+\lambda_n}\right)\|x_0 - x^*\|_A,$$

y, en particular,

$$\|x_m - x^*\|_A \;\leq\; 2\left(\frac{\sqrt{\kappa_2(A)}-1}{\sqrt{\kappa_2(A)}+1}\right)^m \|x_0 - x^*\|_A,$$

donde $\kappa_2(A) = \lambda_1/\lambda_n$ *es el número de condición de* A *en la norma euclidiana.*

2.7. Método de Biortogonalización de Lanczos

El algoritmo de biortogonalización de Lanczos extiende el algoritmo de Lanczos simétrico (visto en la sección (2.5)) al caso de matrices no simétricas.

Para una matriz $A \in \mathbb{R}^{n\times n}$ (no necesariamente simétrica) se construyen dos bases biortogonales asociadas a los subespacios:

$$K_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}, \qquad K_m(A^\top, w_1) = \text{span}\{w_1, A^\top w_1, \ldots, (A^\top)^{m-1}w_1\},$$

donde se exige la condición de biortogonalidad inicial $(v_1, w_1) = 1$.

La construcción de dichas bases viene dada por el siguiente procedimiento:

Algoritmo 2.20. *Método de biortogonalización de Lanczos*

1: *Choose two vectors* v_1, w_1 *such that* $(v_1, w_1) = 1$
2: *Set* $\beta_1 \equiv \delta_1 \equiv 0$, $w_0 = v_0 = 0$
3: ***for*** $j = 1, 2, \ldots, m$ ***do***
4: $\quad \alpha_j = (Av_j, w_j)$
5: $\quad \tilde{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$
6: $\quad \tilde{w}_{j+1} = A^\top w_j - \alpha_j w_j - \delta_j w_{j-1}$
7: $\quad \delta_{j+1} = |(\tilde{v}_{j+1}, \tilde{w}_{j+1})|^{1/2}$. ***If*** $\delta_{j+1} = 0$ ***Stop***
8: $\quad \beta_{j+1} = (\tilde{v}_{j+1}, \tilde{w}_{j+1})/\delta_{j+1}$
9: $\quad w_{j+1} = \tilde{w}_{j+1}/\beta_{j+1}$
10: $\quad v_{j+1} = \tilde{v}_{j+1}/\delta_{j+1}$
11: ***end for***

Las constantes de escala $\delta_{j+1}, \beta_{j+1}$ pueden elegirse de diversas maneras. Estos parametros son factores de escala de los dos vectores v_{j+1} y w_{j+1} y pueden seleccionarse de cualquier manera siempre que $(v_{j+1}, w_{j+1}) = 1$. De las líneas 9 y 10 del algoritmo se deduce que la forma de escoger los factores de escala $\delta_{j+1}, \beta_{j+1}$ es satisfaciendo la siguiente igualdad:

$$\delta_{j+1}\beta_{j+1} = (\tilde{v}_{j+1}, \tilde{w}_{j+1}). \tag{2.35}$$

Si $\beta_{j+1}, \delta_{j+1}$ cumplen (2.35), la matriz tridiagonal resultante

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix} \tag{2.36}$$

es, en general, no simétrica y se cumple que los δ_j son positivos y los $\beta_j = \pm\delta_j$. El algoritmo genera simultáneamente los vectores $v_i \in K_m(A, v_1)$ y $w_j \in K_m(A^\top, w_1)$.

Proposición 2.21. *Si el algoritmo no sufre breakdown antes del paso m, los vectores v_i, $i = 1, \ldots, m$ y w_j, $j = 1, \ldots, m$ forman un sistema bi-ortogonal, es decir*

$$(v_j, w_i) = \delta_{ij}, \qquad 1 \le i, j \le m.$$

Además, $\{v_i\}_{i=1}^m$ es una base de $K_m(A, v_1)$ y $\{w_i\}_{i=1}^m$ es una base de $K_m(A^\top, w_1)$, cumpliéndose las relaciones matriciales

$$AV_m = V_m T_m + \delta_{m+1} v_{m+1} e_m^\top, \tag{2.37}$$

$$A^\top W_m = W_m T_m^\top + \beta_{m+1} w_{m+1} e_m^\top, \tag{2.38}$$

$$W_m^\top A V_m = T_m. \tag{2.39}$$

Demostración. La bi-ortogonalidad de los vectores (v_i, w_i) se prueba por inducción. Por hipótesis $(v_1, w_1) = 1$. Se supone ahora que $v_1, \ldots, v_j$ y $w_1, \ldots, w_j$ son bi-ortogonales y se demuestra que $(v_{j+1}, w_i) = 0$ para $i \le j$.

Caso $i = j$. De la línea 4, 5 y 10 del algoritmo se tiene $v_{j+1} = \delta_{j+1}^{-1}(Av_j - \alpha_j v_j - \beta_j v_{j-1})$ y $\alpha_j = (Av_j, w_j)$, usando $(v_j, w_j) = 1$ y $(v_{j-1}, w_j) = 0$, resulta

$$(v_{j+1}, w_j) = \delta_{j+1}^{-1}\big[(Av_j, w_j) - \alpha_j - 0\big] = \delta_{j+1}^{-1}\left[\alpha_j - \alpha_j\right] = 0.$$

Caso $i < j$. Para $i < j$ se verifica igualmente, empleando la inducción, que

$$\begin{aligned} (v_{j+1}, w_i) &= \delta_{j+1}^{-1}\big[(Av_j, w_i) - \alpha_j(v_j, w_i) - \beta_j(v_{j-1}, w_i)\big] \\ &= \delta_{j+1}^{-1}\big[(v_j, A^\top w_i) - \beta_j(v_{j-1}, w_i)\big] \\ &= \delta_{j+1}^{-1}\big[(\beta_{i+1} w_{i+1} + \alpha_i w_i + \delta_i w_{i-1}, v_j) - 0\big] = 0. \end{aligned}$$

En la expresión de arriba se anulan todos los productos interiores para $i < j-1$ por la hipótesis de inducción. Para $i = j-1$, el producto interior es de la forma:

$$\begin{aligned}(v_{j+1}, w_{j-1}) &= \delta_{j+1}^{-1}[(v_j, \beta_j w_j + \alpha_{j-1} w_{j-1} + \delta_{j-1} w_{j-2}) - \beta_j (v_{j-1}, w_{j-1})] \\ &= \delta_{j+1}^{-1}[\beta_j (v_j, w_j) - \beta_j (v_{j-1}, w_{j-1})] = 0\end{aligned}$$

Un argumento análogo muestra que $(v_i, w_{j+1}) = 0$ para $i \leq j$. Finalmente, por construcción $(v_{j+1}, w_{j+1}) = 1$. Esto completa la inducción. Para la demostración de las relaciones (2.37), (2.38) y (2.39) se efectuan los mismos desarrollos que en la Proposición 2.7 de la sección del método de Arnoldi. □

Las relaciones (2.37), (2.38) y (2.39) permiten interpretar el algoritmo del método de Lanczos no simétrico. Nótese que la matriz T_m puede verse como la proyección de A obtenida mediante un proceso de proyección oblicua sobre el subespacio $\mathcal{K}_m(A, v_1)$ y los residuos son ortgonales al subespacio $\mathcal{K}_m\big(A^T, w_1\big)$. De modo análogo, la matriz T_m^T representa la proyección de A^T sobre $\mathcal{K}_m\big(A^T, w_1\big)$ y ortogonal a $\mathcal{K}_m(A, v_1)$. Una propiedad interesante es que los operadores A y A^T desempeñan papeles parecidos, puesto que se realizan operaciones similares con ambos. De hecho, el método resuelve de forma implícita dos sistemas lineales: uno con A y otro con A^T. Si el objetivo fuese resolver dos sistemas lineales, uno con A y otro con A^T, entonces este algoritmo sería idóneo; sin embargo, en la práctica las operaciones con A^T quedan sin utilizar.

Desde un punto de vista práctico, el algoritmo de Lanczos ofrece una ventaja importante frente al método de Arnoldi: requiere muy poco almacenamiento. En concreto, sólo se necesitan seis vectores de longitud n, además del almacenamiento para la matriz tridiagonal, independientemente del valor de m. Por otra parte, el método de Lanczos no simétrico es más propenso a fallar: se detiene siempre que δ_{j+1} (definido en la línea 7 del algoritmo) se anula.

2.7.1. Método de Biortogonalización de Lanczos para sistemas lineales

En esta sección se presenta una descripción concisa del método de Lanczos para la resolución de sistemas lineales no simétricos. Se considera el sistema lineal

$$Ax = b, \tag{2.40}$$

donde A es una matriz $n \times n$ no simétrica. Sea x_0 una aproximación inicial y se denota su vector residual por $r_0 = b - Ax_0$. El algoritmo de biortogonalización Lanczos para resolver (2.40) puede describirse de la forma siguiente.

Algoritmo 2.22. *Método de biortogonalización de Lanczos para sistemas lineales*

1: *Compute* $r_0 := b - Ax_0$ *and* $\beta := \|r_0\|_2$

2: Run m steps of the nonsymmetric Lanczos Algorithm, i.e.:
3: Start with $v_1 := r_0/\beta$ *and any* w_1 *such that* $(v_1, w_1) = 1$
4: Generate the Lanczos vectors $v_1, \ldots, v_m,\ w_1, \ldots, w_m$
5: and the tridiagonal matrix T_m *from Algorithm 2.20*
6: Compute $y_m := T_m^{-1}(\beta e_1)$ *and* $x_m := x_0 + V_m y_m$

Para este método es posible incorporar un test de convergencia al generar los vectores de Lanczos en el segundo paso sin calcular explícitamente la solución aproximada. Esto se debe a la siguiente fórmula, similar a la ecuación (2.21) para el caso simétrico:

$$||b - Ax_j||_2 = |\delta_{j+1} e_j^T y_j| ||v_{j+1}||_2. \tag{2.41}$$

Esta igualdad se demuestra siguiendo un procedimiento análogo al usado en la Proposición 2.11. Esta fórmula proporciona la norma residual de forma económica, sin calcular explícitamente la solución aproximada.

2.8. Método de Gradiente Biconjugado (BCG)

El algoritmo de Gradiente Biconjugado se deriva del Algoritmo 2.20 exactamente del mismo modo que el Algoritmo de Gradiente Conjugado 2.17 se deriva del Algoritmo 2.13. Implícitamente, el BCG resuelve dos sistemas lineales uno con la matriz A y otro con la matriz A^T. Normalmente, el sistema $A^T x^* = b^*$ se ignora en las formulaciones del algoritmo. El algoritmo puede interpretarse como un proceso de proyección sobre el subespacio de Krylov

$$\mathcal{K}_m = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}$$

y ortogonal al subespacio

$$\mathcal{L}_m = \text{span}\{w_1, A^T w_1, \ldots, (A^T)^{m-1} w_1\},$$

tomando, como es habitual, $v_1 = r_0/\|r_0\|_2$. El vector w_1 se elige arbitrario con la única condición de que $(v_1, w_1) \neq 0$; con frecuencia se toma $w_1 = v_1$. Si se desea resolver el sistema dual $A^T x^* = b^*$, puede obtenerse w_1 escalando el residuo inicial $b^* - A^T x_0^*$.

Siguiendo los mismos pasos utilizados para derivar el método de Gradiente Conjugado a partir del Lanczos simétrico, se escribe ahora la descomposición LU de la matriz tridiagonal

$$T_m = L_m U_m, \tag{2.42}$$

y se define

$$P_m \ = \ V_m U_m^{-1}. \tag{2.43}$$

Con estas notaciones la solución aproximada después de m pasos se expresa como

$$x_m = x_0 + V_m T_m^{-1}(\beta e_1) = x_0 + V_m U_m^{-1} L_m^{-1}(\beta e_1) = x_0 + P_m L_m^{-1}(\beta e_1).$$

Se define, de manera análoga,

$$P_m^* \;=\; W_m L_m^{-T}. \tag{2.44}$$

Es claro que las columnas p_i^* de P_m^* y las p_i de P_m son A-conjugadas, ya que

$$(P_m^*)^T A P_m = L_m^{-1} W_m^T A V_m U_m^{-1} = L_m^{-1} T_m U_m^{-1} = I.$$

Todo esto permite derivar, a partir del procedimiento de Lanczos, el siguiente algoritmo para matrices no simétricas.

Algoritmo 2.23. *Gradiente Biconjugado (BCG)*

1: *Compute* $r_0 := b - Ax_0$. *Choose* r_0^* *such that* $(r_0, r_0^*) \neq 0$.
2: *Set* $p_0 := r_0$, $p_0^* := r_0^*$
3: ***for*** $j = 0, 1, \ldots$ *until convergence* ***do***
4: $\quad \alpha_j := (r_j, r_j^*)/(Ap_j, p_j^*)$
5: $\quad x_{j+1} := x_j + \alpha_j p_j$
6: $\quad r_{j+1} := r_j - \alpha_j A p_j$
7: $\quad r_{j+1}^* := r_j^* - \alpha_j A^T p_j^*$
8: $\quad \beta_j := (r_{j+1}, r_{j+1}^*)/(r_j, r_j^*)$
9: $\quad p_{j+1} := r_{j+1} + \beta_j p_j$
10: $\quad p_{j+1}^* := r_{j+1}^* + \beta_j p_j^*$
11: ***end for***

Si además se desea resolver el sistema dual $A^T x^* = b^*$, en la línea 1 se define $r_0^* = b^* - A^T x_0^*$ y la actualización correspondiente para x_{j+1}^* es $x_{j+1}^* = x_j^* + \alpha_j p_j^*$ en la línea 5.

Como en el caso del Gradiente Conjugado, en este método los residuos r_j y r_j^* están en la misma dirección que v_{j+1} y w_{j+1} y por lo tanto forman secuencias biortogonales como se ve en la siguiente proposición:

Proposición 2.24. *Los vectores producidos por el Algoritmo 2.23 satisfacen las propiedades de ortogonalidad siguientes:*

$$(r_j, r_i^*) = 0, \qquad \text{para } i \neq j, \tag{2.45}$$

$$(Ap_j, p_i^*) = 0, \qquad \text{para } i \neq j. \tag{2.46}$$

Demostración. Sea $V_m = [v_1, \ldots, v_m]$, $W_m = [w_1, \ldots, w_m]$ el par de bases biortogonales generado por el procedimiento de Lanczos, de forma que $W_m^T V_m = I$. El algoritmo BCG produce

$$r_j = \sigma_j v_{j+1}, \qquad r_j^* = \tau_j w_{j+1},$$

para ciertos escalares $\sigma_j, \tau_j \neq 0$, y

$$p_j = \rho_j v_{j+1}, \qquad p_j^* = \eta_j w_{j+1},$$

para escalares $\rho_j, \eta_j \neq 0$. Dado que las bases $\{v_i\}$ y $\{w_i\}$ son biortogonales, se tiene $(v_j, w_i) = \delta_{ij}$. Así,

$$(r_j, r_i^*) = \sigma_j \tau_i (v_{j+1}, w_{i+1}) = \sigma_j \tau_i \delta_{ij},$$

que vale cero cuando $i \neq j$, demostrando (2.45). Por otra parte,

$$Ap_j = \rho_j A v_{j+1} = \rho_j \sum_{k=1}^{j+1} h_{k\,j+1} v_k,$$

donde los $h_{k\,j+1}$ provienen del procedimiento de Lanczos (no nulos sólo para $k \leq j+1$). Por biortogonalidad,

$$(Ap_j, p_i^*) = \rho_j \eta_i \sum_{k=1}^{j+1} h_{k\,j+1} (v_k, w_{i+1}) = \rho_j \eta_i \, h_{i+1\,j+1}.$$

Sin embargo, las propiedades de la matriz tridiagonal T_m implican que $h_{i\,j} = 0$ cuando $i \neq j$, lo que prueba (2.46). □

Capítulo 3

Métodos de Krylov. Sistemas con matrices complejas

En el capítulo anterior se considera siempre que la matriz A del sistema lineal es real y los algoritmos dados se pueden extender de manera sencilla al caso en el que A es una matriz compleja. En este capítulo se introducen algunos de ellos con el objetivo de introducir el llamado método de Gradiente Conjugado Ortogonal siguiendo [7]. Por tanto, de ahora en adelante se considera el sistema lineal:

$$Ax = b \tag{3.1}$$

donde $A \in \mathbb{C}^{n\times n}$, $b \in \mathbb{C}^n$.

3.1. Método de Biortogonalización de Lanczos con matrices complejas no hermitianas

En esta sección se adapta al caso complejo los Algoritmos 2.20 y 2.22. El procedimiento genera dos bases bi-ortogonales para los subespacios de Krylov:

$$\mathcal{K}_m(A, v_1) = \text{span}\{v_1, Av_1, \ldots, A^{m-1}v_1\}, \qquad \mathcal{K}_m(A^{\mathrm{H}}, w_1) = \text{span}\{w_1, A^{\mathrm{H}}w_1, \ldots, (A^{\mathrm{H}})^{m-1}w_1\},$$

donde $(v_1, w_1) = 1$. El algoritmo básico queda:

Algoritmo 3.1. *Algoritmo de biortogonalización de Lanczos para matrices no hermitianas*

1: *Choose two starting vectors* v_1 *and* w_1 *such that* $(v_1, w_1) = 1$*; set* $\beta_1 = \delta_1 \equiv 0$, $w_0 = v_0 \equiv 0$
2: ***for*** $j = 1, 2 \ldots, m$ ***do***
3: $\quad \alpha_j = (Av_j, w_j)$
4: $\quad \tilde{v}_{j+1} = Av_j - \alpha_j v_j - \beta_j v_{j-1}$

5: $\tilde{w}_{j+1} = A^{\mathrm{H}} w_j - \overline{\alpha_j} w_j - \overline{\delta_j} w_{j-1}$
6: $\delta_{j+1} = |(\tilde{v}_{j+1}, \tilde{w}_{j+1})|^{1/2}$;
7: ***if*** $\delta_{j+1} = 0$ ***stop***
8: $\beta_{j+1} = (\tilde{v}_{j+1}, \tilde{w}_{j+1})/\delta_{j+1}$
9: $w_{j+1} = \tilde{w}_{j+1}/\overline{\beta_{j+1}}$
10: $v_{j+1} = \tilde{v}_{j+1}/\delta_{j+1}$
11: ***end for***

Se puede observar que los únicos cambios con el Algoritmo 2.20 es que A^T se cambia por A^H y se conjugan los escalares. Los escalares $\delta_{j+1}, \beta_{j+1}$ pueden escogerse de forma distinta siempre que satisfagan $(v_{j+1}, w_{j+1}) = 1$.

Se obtiene, por tanto, la siguiente matriz tridiagonal

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \delta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \delta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \delta_m & \alpha_m \end{pmatrix},$$

cuya dimensión es $m \times m$. A medida que m crece, los autovalores extremos de $T_m \in \mathbb{C}^{m\times m}$ tienden a aproximar los autovalores extremos de la matriz A. Sea $V_m = [v_1, v_2, \ldots, v_m]$ y $W_m = [w_1, w_2, \ldots, w_m]$; con estas notaciones puede enunciarse el siguiente teorema:

Proposición 3.2. *Si el Algoritmo 3.1 no sufre breakdown hasta el paso m, los vectores $v_i, i = 1, 2, \ldots m$ y $w_j, j = 1, 2, \ldots m$ forman un sistema bi-ortogonal, esto es*

$$(v_i, w_j) = \delta_{ij}, \qquad 1 \le i, j \le m, \qquad W_m^{\mathrm{H}} V_m = I_m,$$

donde $\{v_1, v_2, \ldots, v_m\}$ es una base $\mathcal{K}_m(A, v_1)$ y $\{w_1, w_2, \ldots, w_m\}$ es una base de $\mathcal{K}_m(A^H, w_1)$. Además,

$$AV_m = V_m T_m + \tilde{v}_{m+1} e_m^{\mathrm{T}}, \tag{3.2}$$

$$A^{\mathrm{H}} W_m = W_m T_m^{\mathrm{H}} + \tilde{w}_{m+1} e_m^{\mathrm{T}}, \tag{3.3}$$

$$W_m^{\mathrm{H}} A V_m = T_m. \tag{3.4}$$

donde e_m^{T} es el m-ésimo vector de la base canónica.

Demostración. La demostración de esta proposición se hace de una manera análoga a la demostración de la Proposición 2.21. □

El siguiente algoritmo es análogo al Algoritmo 2.22

Algoritmo 3.3. *Algoritmo de Lanczos con matrices no hermitianas para sistemas lineales*

1: *Compute* $r_0 := b - Ax_0$ *and* $\beta := \|r_0\|_2$
2: *Run* m *steps of the Algorithm 3.1, i.e.:*
3: *Start with* $v_1 := r_0/\beta$ *and any* w_1 *such that* $(v_1, w_1) = 1$
4: *Generate the Lanczos vectors* $v_1, \ldots, v_m,\ w_1, \ldots, w_m$
5: *and the tridiagonal matrix* T_m *from Algorithm 3.1*
6: *Compute* $y_m := T_m^{-1}(\beta e_1)$ *and* $x_m := x_0 + V_m y_m$

siendo $V_m = [v_1, ..., v_m]$.

En el caso hermitiano, así como en el no hermitiano, puede escribirse la siguiente expresión para la norma del residuo [5]:

$$\|b - Ax_m\|_2 \ = \ \left|\delta_{m+1}\, e_m^{\mathrm{T}} y_m\right| \|v_{m+1}\|_2,$$

Por consiguiente, la norma del residuo puede evaluarse con muy poco esfuerzo y sin formar explícitamente la solución aproximada. En el paso 3 del algoritmo hay que resolver el sistema lineal

$$T_m\, y_m \ = \ \beta_1 e_1.$$

Esto puede resolverse, por ejemplo, mediante una descomposición LU de $T_m = L_m U_m$. Si se incorpora dicha descomposición dentro del proceso de Lanczos, pueden determinarse sucesivamente las aproximaciones x_m; este procedimiento se ha visto en el método de Gradiente Conjugado y se denomina algoritmo de Lanczos directo o D-Lanczos.

Por último, el Algoritmo 3.3 sólo requiere almacenar seis vectores y una matriz tridiagonal, independientemente del valor de m.

3.2. Método de Gradiente Biconjugado para matrices complejas

El método BCG para matrices complejas se obtiene de la misma manera que se obtuvo en 2.8 para matrices reales. El único cambio es que ahora, el BCG considera los dos sistemas lineales siguientes:

$$Ax = b \quad A^H x^* = b^*$$

Las iteraciones satisfacen la condición de Petrov–Galerkin (2.8):

$$b - Ax_m \perp \mathcal{K}_m(A, r_0), \qquad x_m \in x_0 + \mathcal{K}_m(A^H, \tilde{r_0}),$$

siendo $r_0 = b - Ax_0$ y $\tilde{r}_0$ un vector residuo adicional, que se le llamará pseudo-residuo relacionado con A^H. A continuación, se presenta la extensión del Algoritmo 2.23 al caso complejo.

Algoritmo 3.4. *Algoritmo de Gradiente Biconjugado para matrices complejas*

Choose x_0, $r_0 = b - Ax_0$ *and* $\tilde{r}_0$ *such that* $(r_0, \tilde{r}_0) \neq 0$*; set* $p_{-1} := \tilde{p}_{-1} := 0$, $(r_{-1}, \tilde{r}_{-1}) := 1$.
for $j = 0, 1, \ldots$ *until convergence* ***do***
  $\beta_j = \dfrac{(r_j, \tilde{r}_j)}{(r_{j-1}, \tilde{r}_{j-1})}$
  $p_j = r_j + \beta_j p_{j-1}$
  $\tilde{p}_j = \tilde{r}_j + \overline{\beta_j} \tilde{p}_{j-1}$
  if $(p_j, A\tilde{p}_j) = 0$ ***then*** *stop*
  end if
  $\alpha_j = \dfrac{(r_j, \tilde{r}_j)}{(p_j, A\tilde{p}_j)}$
  $x_{j+1} = x_j + \alpha_j p_j$
  $r_{j+1} = r_j - \alpha_j A p_j$
  $\tilde{r}_{j+1} = \tilde{r}_j - \overline{\alpha}_j A^H \tilde{p}_j$
end for

Los pseudo-residuos $\tilde{r}_j = b - A^H x_j$ y las pseudo-direcciones de búsqueda $\tilde{p}_j$ garantizan, desde el punto de vista teórico, la convergencia del proceso tras un número finito de pasos. Por construcción, los pseudo-residuos $\tilde{r}_j$ son ortogonales a los residuos r_j y las pseudo-direcciones $\tilde{p}_j$ son A-conjugadas a las direcciones de búsqueda p_j. En el caso real y simétrico, el algoritmo BCG se reduce al método CG, ya que en tal situación las pseudo-direcciones y los pseudo-residuos coinciden con las direcciones de búsqueda y los residuos originales.

3.3. Método de Gradiente Conjugado Ortgonal (COCG)

En [4] se desarrollan métodos del tipo Gradiente Conjugado que resuelven sistemas lineales cuando A es simétrica, como el método QMR (Quasi Minimal Residual). Asimismo, la sección 3.4.4 del libro [7] se centra en la descripción de distintos métodos para resolver este tipo de sistemas. Dichos sistemas lineales surgen por ejemplo en la discretización de ecuaciones en derivadas parciales como la ecuación compleja de Helmholtz descrita en [4] o en numerosos problemas de corrientes inducidas [7, 8]. En esta sección introduciremos uno de los métodos que resuelve sistemas lineales cuando la matriz es simétrica y no hermitiana. El método de Gradiente Conjugado Ortgonal , COCG por sus siglas en inglés (Conjugate Orthogonal Conjugate Gradient) formulado en [8] por van der Vorst y Melissen y también puede verse como una variante del algoritmo BCG para matrices complejas simétricas dado en [4]. En [8] la idea clave del método COCG consiste en reemplazar la ortogonalidad entre los residuos que se tendría en un método de biortogonalización por una relación de ortogonalidad conjugada:

$$(r_i, \overline{r}_j) \;=\; 0, \qquad i \neq j. \tag{3.5}$$

Será posible construir una base para $K_m(A; r_0)$ mediante una relación de recurrencia de tres términos; esta recurrencia ha sido definida en el método de Biortogonalización de Lanczos para matrices no hermitianas y se define del siguiente modo:

$$r_{j+1} = Ar_j - \alpha_j r_j - \beta_j r_{j-1},$$

donde los escalares α_j, β_j se obtendrán a partir de las condiciones

$$(r_j, \overline{r}_{j+1}) = 0 \quad y \quad (r_{j-1}, \overline{r}_{j+1}) = 0.$$

Se construye la secuencia $\{r_j\}$ tal que r_0 es el vector inicial $r_0 = b - Ax_0, \;\; \gamma_0 = 0.$

Por construcción, los vectores $\{r_1, \ldots, r_m\}$ forman una base de $K_m(A; r_0)$ y satisfacen la ortogonalidad conjugada (3.5). Para $1 \leq j \leq m$ se tiene:

$$(r_j, \overline{r}_{m+1}) = (Ar_m - \alpha_m r_m - \beta_m r_{m-1}, \overline{r}_j) = (Ar_m, \overline{r}_j) = (Ar_j, \overline{r}_m) = (r_{j+1} + \alpha_j r_j + \beta_j r_{j-1}, \overline{r}_m) = 0.$$

La relación de recurrencia indicada previamente puede expresarse en notación matricial como

$$AV_m \;=\; V_m T_m + r_{m+1} e_m^T,$$

con

$$T_m \;=\; \begin{pmatrix} \alpha_1 & \beta_1 & & & \\ 1 & \alpha_2 & \beta_2 & & \\ & 1 & \ddots & \ddots & \\ & & \ddots & & \beta_{m-1} \\ & & & 1 & \alpha_m \end{pmatrix},$$

donde $V_m = [r_1, r_2, \ldots r_m]$ y e_m es el m-ésimo vector canónico en $\mathbb{C}^n$. En resumen, los vectores r_j generados satisfacen la ortogonalidad conjugada y proporcionan la base sobre la que se construye el método Conjugate Orthogonal Conjugate Gradient.

Las siguientes ecuaciones se definen de la misma manera que se definieron para el método CG.

$$x_{j+1} = x_j + \alpha_j p_j,$$

$$r_{j+1} = r_j - \alpha_j A p_j,$$

$$p_{j+1} = r_{j+1} + \beta_j p_j,$$

donde $r_j = b - Ax_j$ es el residuo, p_j la dirección de búsqueda. Los vectores $\{p_i\}_{i \leq m}$ cumplen en este caso la siguiente relación:

$$(p_i, \overline{Ap}_j) = 0, \qquad i \neq j.$$

Usando la ortogonalidad conjugada de los residuos y procediendo de la misma forma que en Gradiente Conjugado llegamos a las siguientes expresiones para α_j y β_j:

$$\alpha_j = \frac{(r_j, \overline{r}_j)}{(p_j, \overline{Ap_j})} \quad \beta_j = \frac{(r_j, \overline{r}_j)}{(r_{j-1}, \overline{r}_{j-1})}.$$

A continuación, introducimos el algoritmo para el método de Gradiente Conjugado Ortgonal:

Algoritmo 3.5. *Algoritmo COCG*

Choose $x_0 \in \mathbb{C}^n$, *set* $p_0 = r_0 = b - Ax_0$, $p_{-1} = 0$, $(r_{-1}, \bar{r}_{-1}) = 1$
for $j = 0, 1, \ldots$ *until convergence* ***do***
  $\beta_j = (r_j, \bar{r}_j)/(r_{j-1}, \bar{r}_{j-1})$
  $p_j = r_j + \beta_j p_{j-1}$
  if $(p_j, \overline{Ap}_j) = 0$ ***then*** *stop*
  end if
  $\alpha_j = (r_j, \bar{r}_j)/(p_j, \overline{Ap}_j)$
  $x_{j+1} = x_j + \alpha_j p_j$
  $r_{j+1} = r_j - \alpha_j A p_j$
end for

El algoritmo COCG se ha definido siguiendo el artículo original [8]. Sin embargo, cabe señalar que se puede llegar a un algoritmo equivalente partiendo del método BCG con matriz compleja y simétrica como se hace en [4]. Para ello se define un producto interno para la biconjugación diferente al producto interno hermitiano. En particular, en dicho artículo se define el producto interno $[x, y] = x^T y$. De este modo puede verse que el Algoritmo 2.3 de [4] es equivalente al Algoritmo 3.5 presentado previamente. Teniendo en cuenta las siguientes igualdades:

$$(r_j, \bar{r}_j) = \bar{r}_j^H r_j = r_j^T r_j$$

$$(p_j, \overline{Ap}_j) = (\overline{Ap}_j)^H p_j = \bar{p}_j^H \overline{A}^H p_j = p_j^T A^T p_j = p_j^T A p_j.$$

3.4. Aplicaciones prácticas del método COCG

El propósito de esta sección es ilustrar, mediante ejemplos numéricos concretos, la eficacia y las limitaciones del método Conjugate Orthogonal Conjugate Gradient (COCG) cuando se aplica a la resolución de sistemas lineales con matrices complejas, simétricas y no hermitianas. En primer lugar se describe el entorno de programación utilizado (Matlab R2024a) y se comenta brevemente los datos que necesita el código así como los resultados que proporciona. Finalmente, se exponen resultados obtenidos aplicando el algoritmo en Matlab.

3.4.1. Implementación del método en Matlab

Matlab (abreviatura de Matrix Laboratory, laboratorio de matrices) es un sistema de cómputo numérico que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación

propio (lenguaje M). Entre sus prestaciones básicas se hallan la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. Las aplicaciones de Matlab se desarrollan en un lenguaje de programación propio. Este lenguaje es interpretado, y puede ejecutarse tanto en el entorno interactivo, como a través de un archivo de script (archivos *.m). Este lenguaje permite operaciones de vectores y matrices, funciones, y programación orientada a objetos. Las matrices que se usarán para la obtención mostrar el funcionamiento del método COCG se han obtenido del repositorio [2]. Se trata de un repositorio público mantenido por el NIST (National Institute of Standarts and Technology, EE.UU.) que recopila matrices reales y complejas utilizadas como casos de prueba en álgebra lineal numérica. Su objetivo es ofrecer conjuntos de datos estandarizados para que investigadores y desarrolladores puedan evaluar algoritmos (métodos directos, iterativos, de descomposición, precondicionadores, etc.), comparar rendimiento (tiempo, memoria, precisión) entre implementaciones de distintos laboratorios o lenguajes y reproducir resultados de artículos científicos.

Para la implementación del método COCG, se han creado dos programas:

- **prinCOCG.m** : Es el programa principal y se encarga de la lectura/definición de la matriz y del segundo miembro del sistema, así como la definición de distintos parametros que necesita el algoritmo (tolerancia, máximo número de iteraciones, etc) y la llamada al método COCG.
- **cocg.m** : Es el programa que se encarga de aplicar el método COCG siguiendo el Algoritmo 3.5.

El programa **prinCOCG.m** define las siguientes variables:

- A (la matriz cuadrada compleja simétrica y no hermitiana que define el sistema (3.1))
- b (el termino independiente del sistema (3.1))
- x_0 (aproximación inicial)
- tol (tolerancia del método)
- maxiter (número máximo de iteraciones)
- n (orden de la matriz A)
- solexac (solución exacta del sistema (3.1))

En la mayoría de casos el orden de la matriz A es grande, por ello, se utiliza la función de Matrix Market, **mmread.m**, que es capaz de leer archivos .mtx y proporcionar al programa

principal la matriz A. El código de la función **mmread.m** viene dado explícitamente en el Anexo I.1. Para algunos test con matriz pequeña, se ha introducido la matriz manualmente o, mediante la función **rnd_csPD**, ver (I.4), que genera una matriz aleatoria cumpliendo las condiciones necesarias para aplicar COCG. Utilizando el comando \ de matlab se puede obtener la solución exacta de un sistema lineal no singular mediante un método directo. La solución exacta del sistema (3.1) hace que se pueda comparar con la solución aproximada que ofrece el método COCG. Para calcular la solución aproximada se llama al programa **cocg.m**. Las variables de entrada son A,b, x_0, tol y maxiter, y las variables que devolverá el programa serán xSol (solución aproximada aplicando el método COCG) y numIter (número de iteraciones realizadas por el programa hasta alcanzar convergencia). El código del programa **cocg.m** viene dado explícitamente en el Anexo I.2. Si el programa converge, devolverá una solución aproximada. En el programa principal, **prinCOCG.m**, se compara la solución exacta con la solución aproximada calculando la norma de la diferencia. Si se llega que la diferencia es próxima a cero diremos que se ha encontrado una buena aproximación. El código de **prinCOCG.m** viene dado explícitamente en el Anexo I.3.

3.4.2. Resultados con matrices adquiridas en Matrix Market

En esta sección se proporcionan resultados numéricos aplicando el método de Gradiente Conjugado Ortogonal para la resolución de sistemas lineales. Las matrices que se usan fueron adquiridas en la web Matrix Market modelan fenómenos de dispersión acústica y son matrices cuadradas, sparse, complejas, simétricas y no hermitianas.

Las matrices que se usarán son las siguientes:

- **YOUNG1C** (841×841, 4089 entradas no nulas)
- **YOUNG3C** (841×841, 3988 entradas no nulas)
- **YOUNG4C** (841×841, 4089 entradas no nulas)

Todas las matrices provienen de la discretización de la ecuación de Helmholtz con distintas condiciones de contorno y/o distintos parámetros físicos. Las tres matrices corresponden a una discretización utilizando una malla de 29x29 puntos interiores.

Por tanto, el sistema lineal a resolver es:

$$Ax = b$$

donde A es una de las matrices definidas anteriormente y b es el vector columna tal que $b_i = 1 + i \quad \forall i \in \{1, \dots n\}$. La aproximación inicial, x_0, es el vector nulo. A continuación, se muestra

Matriz	**Tolerancia**	**Iteraciones**	**Orden** n	$\|\mathbf{x}_{\text{aprox}} - \mathbf{x}_{\text{exact}}\|_2$
YOUNG1C	$1{,}0\times10^{-8}$	368	841	$1{,}98\times10^{-6}$
YOUNG3C	$1{,}0\times10^{-8}$	1 093	841	$1{,}49\times10^{-7}$
YOUNG4C	$1{,}0\times10^{-8}$	724	841	$6{,}37\times10^{-6}$

Cuadro 3.1: Resultados del método COCG con las matrices Young1C,Young3C y Young4C.

una tabla en la que se describe la toleracia usada, el número de iteraciones hasta que se alcanza la convergencia, el orden de la matriz, y el error entre la solución exacta y aproximada.

Por tanto, se puede ver que la diferencia entre la solución aproximada y la solución exacta es pequeña y aceptable para poder afirmar, que el método COCG es capaz de resolver los sistemas lineales descritos anteriormente.

Además de los sitemas lineales descritos previamente, se estudió la posibilidad de resolver sistemas procedentes de la discretización de problemas de corrientes inducidas. En particular, se intentaron resolver los sistemas derivados del método numérico presentado en [1]. Sin embargo, se observó que el número de condición de estas matrices es muy elevado y el método requeriría el uso de precondicionadores adecuados.

3.4.3. Resultados con matrices aleatorias

En esta sección, se proporcionan resultados numéricos aplicando el método de Gradiente Conjugado Ortogonal para la resolución de sistemas lineales con matrices que han sido generadas de manera aleatoria por la función **rnd_csPD** y cumplen las condiciones para aplicar el método. El término independiente también se genera de forma aleatoria. Es necasario comentar que para que los resultados sean fácilmente reproducibles se utiliza la función de Matlab, rng(0), que incializa el generador de números aleatorios de Matlab usando la semilla y el algoritmo predeterminados.

A continuación, se muestra una tabla en la que se describe la tolerancia usada, el número de iteraciones hasta que se alcanza la convergencia, el orden de la matriz, y el error entre la solución exacta y aproximada.

Matriz	**Tolerancia**	**Iteraciones**	**Orden** n	$\|\mathbf{x}_{\text{aprox}} - \mathbf{x}_{\text{exact}}\|_{\mathbf{2}}$
Matriz1	$1{,}0\times10^{-8}$	16	71	$2,12\times10^{-8}$
Matriz2	$1{,}0\times10^{-8}$	19	1200	$2,28\times10^{-9}$
Matriz3	$1{,}0\times10^{-8}$	18	2100	$1{,}86\times10^{-9}$
Matriz4	$1{,}0\times10^{-8}$	19	5000	$3{,}51\times10^{-10}$
Matriz5	$1{,}0\times10^{-8}$	19	6988	$2,92\times10^{-10}$
Matriz6	$1{,}0\times10^{-8}$	21	8643	$7,20\times10^{-11}$
Matriz7	$1{,}0\times10^{-8}$	20	10000	$2,96\times10^{-10}$

Cuadro 3.2: Resultados del método COCG con las matrices generadas aleatoriamente.

Anexo I

Código en Matlab

I.1. mmread.m

A continuación se proporciona el código del programa **mmread.m** que hemos adquirido en la web Matrix Market [2]:

Listing I.1: mmread.m

```
function  [A,rows,cols,entries,rep,field,symm] = mmread(filename)
%
% function  [A] = mmread(filename)
%
% function  [A,rows,cols,entries,rep,field,symm] = mmread(filename)
%
%       Reads the contents of the Matrix Market file 'filename'
%       into the matrix 'A'.  'A' will be either sparse or full,
%       depending on the Matrix Market format indicated by
%       'coordinate' (coordinate sparse storage), or
%       'array' (dense array storage).  The data will be duplicated
%       as appropriate if symmetry is indicated in the header.
%
%       Optionally, size information about the matrix can be
%       obtained by using the return values rows, cols, and
%       entries, where entries is the number of nonzero entries
%       in the final matrix. Type information can also be retrieved
%       using the optional return values rep (representation), field,
%       and symm (symmetry).
%

mmfile = fopen(filename,'r');
```

```
23 if ( mmfile == -1 )
24  disp(filename);
25  error('File not found');
26 end;
27
28 header = fgets(mmfile);
29 if (header == -1 )
30   error('Empty file.')
31 end
32
33 % NOTE: If using a version of Matlab for which strtok is not
34 %       defined, substitute 'gettok' for 'strtok' in the
35 %       following lines, and download gettok.m from the
36 %       Matrix Market site.
37 [head0,header]    = strtok(header);  % see note above
38 [head1,header]    = strtok(header);
39 [rep,header]      = strtok(header);
40 [field,header]    = strtok(header);
41 [symm,header]     = strtok(header);
42 head1 = lower(head1);
43 rep   = lower(rep);
44 field = lower(field);
45 symm  = lower(symm);
46 if ( length(symm) == 0 )
47    disp(['Not enough words in header line of file ',filename])
48    disp('Recognized format: ')
49    disp('%%MatrixMarket matrix representation field symmetry')
50    error('Check header line.')
51 end
52 if ( ~ strcmp(head0,'%%MatrixMarket') )
53    error('Not a valid MatrixMarket header.')
54 end
55 if (  ~ strcmp(head1,'matrix') )
56    disp(['This seems to be a MatrixMarket ',head1,' file.']);
57    disp('This function only knows how to read MatrixMarket matrix files.');
58    disp('  ');
59    error('  ');
60 end
61
62 % Read through comments, ignoring them
63
64 commentline = fgets(mmfile);
65 while length(commentline) > 0 & commentline(1) == '%',
```

```
66    commentline = fgets(mmfile);
67  end
68
69  % Read size information, then branch according to
70  % sparse or dense format
71
72  if ( strcmp(rep,'coordinate')) %  read matrix given in sparse
73                                 %  coordinate matrix format
74
75    [sizeinfo,count] = sscanf(commentline,'%d%d%d');
76    while ( count == 0 )
77       commentline =  fgets(mmfile);
78       if (commentline == -1 )
79         error('End-of-file reached before size information was found.')
80       end
81       [sizeinfo,count] = sscanf(commentline,'%d%d%d');
82       if ( count > 0 & count ~= 3 )
83         error('Invalid size specification line.')
84       end
85    end
86    rows = sizeinfo(1);
87    cols = sizeinfo(2);
88    entries = sizeinfo(3);
89
90    if  ( strcmp(field,'real') )               % real valued entries:
91
92      [T,count] = fscanf(mmfile,'%f',3);
93      T = [T; fscanf(mmfile,'%f')];
94      if ( size(T) ~= 3*entries )
95         message = ...
96         str2mat('Data file does not contain expected amount of data.',...
97                 'Check that number of data lines matches nonzero count.');
98         disp(message);
99         error('Invalid data.');
100     end
101     T = reshape(T,3,entries)';
102     A = sparse(T(:,1), T(:,2), T(:,3), rows , cols);
103
104   elseif    ( strcmp(field,'complex'))          % complex valued entries:
105
106     T = fscanf(mmfile,'%f',4);
107     T = [T; fscanf(mmfile,'%f')];
108     if ( size(T) ~= 4*entries )
```

```
109         message = ...
110         str2mat('Data file does not contain expected amount of data.',...
111                 'Check that number of data lines matches nonzero count.');
112         disp(message);
113         error('Invalid data.');
114      end
115      T = reshape(T,4,entries)';
116      A = sparse(T(:,1), T(:,2), T(:,3) + T(:,4)*sqrt(-1), rows , cols);
117
118   elseif  ( strcmp(field,'pattern'))     % pattern matrix (no values given):
119
120      T = fscanf(mmfile,'%f',2);
121      T = [T; fscanf(mmfile,'%f')];
122      if ( size(T) ~= 2*entries )
123         message = ...
124         str2mat('Data file does not contain expected amount of data.',...
125                 'Check that number of data lines matches nonzero count.');
126         disp(message);
127         error('Invalid data.');
128      end
129      T = reshape(T,2,entries)';
130      A = sparse(T(:,1), T(:,2), ones(entries,1) , rows , cols);
131
132   end
133
134 elseif ( strcmp(rep,'array') ) %  read matrix given in dense
135                                %  array (column major) format
136
137   [sizeinfo,count] = sscanf(commentline,'%d%d');
138   while ( count == 0 )
139      commentline =  fgets(mmfile);
140      if (commentline == -1 )
141        error('End-of-file reached before size information was found.')
142      end
143      [sizeinfo,count] = sscanf(commentline,'%d%d');
144      if ( count > 0 & count ~= 2 )
145        error('Invalid size specification line.')
146      end
147   end
148   rows = sizeinfo(1);
149   cols = sizeinfo(2);
150   entries = rows*cols;
151   if  ( strcmp(field,'real') )               % real valued entries:
```

```
152     A = fscanf(mmfile,'%f',1);
153     A = [A; fscanf(mmfile,'%f')];
154     if ( strcmp(symm,'symmetric') | strcmp(symm,'hermitian') | strcmp(symm,'skew-sym
155       for j=1:cols-1,
156         currenti = j*rows;
157         A = [A(1:currenti); zeros(j,1);A(currenti+1:length(A))];
158       end
159     elseif ( ~ strcmp(symm,'general') )
160       disp('Unrecognized symmetry')
161       disp(symm)
162       disp('Recognized choices:')
163       disp('   symmetric')
164       disp('   hermitian')
165       disp('   skew-symmetric')
166       disp('   general')
167       error('Check symmetry specification in header.');
168     end
169       A = reshape(A,rows,cols);
170   elseif  ( strcmp(field,'complex'))          % complx valued entries:
171     tmpr = fscanf(mmfile,'%f',1);
172     tmpi = fscanf(mmfile,'%f',1);
173     A  = tmpr+tmpi*i;
174     for j=1:entries-1
175       tmpr = fscanf(mmfile,'%f',1);
176       tmpi = fscanf(mmfile,'%f',1);
177       A  = [A; tmpr + tmpi*i];
178     end
179     if ( strcmp(symm,'symmetric') | strcmp(symm,'hermitian') | strcmp(symm,'skew-sym
180       for j=1:cols-1,
181         currenti = j*rows;
182         A = [A(1:currenti); zeros(j,1);A(currenti+1:length(A))];
183       end
184     elseif ( ~ strcmp(symm,'general') )
185       disp('Unrecognized symmetry')
186       disp(symm)
187       disp('Recognized choices:')
188       disp('   symmetric')
189       disp('   hermitian')
190       disp('   skew-symmetric')
191       disp('   general')
192       error('Check symmetry specification in header.');
193     end
194     A = reshape(A,rows,cols);
```

```
  elseif  ( strcmp(field,'pattern'))     % pattern (makes no sense for dense)
   disp('Matrix type:',field)
   error('Pattern matrix type invalid for array storage format.');
  else                                   % Unknown matrix type
   disp('Matrix type:',field)
   error('Invalid matrix type specification. Check header against MM documentation.'
  end
end

%
% If symmetric, skew-symmetric or Hermitian, duplicate lower
% triangular part and modify entries as appropriate:
%

if ( strcmp(symm,'symmetric') )
   A = A + A.' - diag(diag(A));
   entries = nnz(A);
elseif ( strcmp(symm,'hermitian') )
   A = A + A' - diag(diag(A));
   entries = nnz(A);
elseif ( strcmp(symm,'skew-symmetric') )
   A = A - A';
   entries = nnz(A);
end

fclose(mmfile);
% Done.
```

I.2. cocg.m

A continuación se proporciona el código del programa del método COCG.

Listing I.2: COCG en MATLAB

```

function [x,iter,resvec] = cocg(A,b,x0,tol,maxIter)
%-----------------------------------------------------------------------

    x       = x0;
    r       = b - A*x;                  % r_0
    p       = r;                        % p_0
    rho     = conj(r)'*r;               % (r0,  r  0)
```

```
9       res0   = norm(b);  if res0==0, res0 = 1; end
10      resvec = zeros(maxIter+1,1);       % guarda residuales
11      resvec(1) = norm(r);
12
13      for k = 0:maxIter-1
14          q    = A*p;              % A p_k
15          denom = conj(q)'*p;                % (p_k , A p _k)
16          if abs(denom)<tol
17              warning('COCG:breakdown','p^T A p      0 (k = %d).',k); break
18          end
19          alpha = rho / denom;           %  _k
20
21          %----- x_{k+1} , r_{k+1}  ----------------------------------
22          x = x + alpha * p;             % x_{k+1}
23          r = r - alpha * q;            % r_{k+1}
24
25          res = norm(r);
26          resvec(k+2) = res;
27          if res <= tol*res0             % convergencia
28              iter   = k+1;
29              resvec = resvec(1:iter+1);
30              return
31          end
32
33         rho_new = conj(r)'*r;         % (r_{k+1},  r _{k+1})
34          if k==0
35              % para k = 0, el denominador usa r_{-1}; definimos  _0  = 0
36              beta = 0;
37          else
38              beta = rho_new / rho;      %  _k
39          end
40
41          %----- p_{k+1}  ---------------------------------------------
42          p    = r + beta * p;           % p_{k+1}
43          rho = rho_new;                 % listo para la siguiente vuelta
44      end
45
46      iter   = k;
47      resvec = resvec(1:iter+1);
48      warning('COCG:NoConverge','No converge tras %d iteraciones',iter);
49  end
```

I.3. princOCG.m

A continuación se proporciona el código del programa principal **princOCG.m**:

Listing I.3: princOCG en MATLAB

```
1
2  A = mmread('young3c.mtx'); %USANDO LA FUNCIoN mmread.m
3  %A = [  3,       1+2i,     0.5-1i,   -2i ;
4   %1+2i,    2,        4i,       1-1i;
5    %      0.5-1i, 4i,      -1,        3+2i;
6     %    -2i,      1-1i,     3+2i,     0     ];
7     %introduciendo la matriz manualmente
8
9  %%%%%%%%%%%%%%%%%%%%
10 n = 841; %Numero de columnas de la matriz
11 %A = rnd_csPD(n,0.4); %matriz generada aleatoriamente
12 b = randn(n,1)+1i*randn(n,1); %termino independiente generado aleatoriamente
13 x0 = zeros(n,1);
14 %%%%%%%%%%%%%%%%%%%%%%%%
15
16 %cond(A); % la funcion cond no funciona con matrices sparse
17 %b = ones(841,1) ; % Vector termino independiente
18 %b = [ 1; 2+ 1i;-1i;3-2i ]; %Vector termino independiente introducido
19 %manualmente
20 tol = 1e-8;        % Tolerancia deseada
21 maxIter = 10000;      % maximo numero de iteraciones
22
23 size(A)
24
25
26 solexac=A\b;
27
28 [xSol, numIter, resvec] = cocg(A, b, x0, tol, maxIter);
29
30 fprintf('Solucion encontrada en %d iteraciones.\n', numIter);
31 fprintf('  Xsol   -  s o l e x a c   = %.2e\n',norm(xSol - solexac));
```

I.4. rnd_csPD

Listing I.4: Función rnd_csPD en MATLAB

```
    %%%%%%%%%%%%%%%%%%%%%%
function A = rnd_csPD(n,imagScale)
% Devuelve A compleja simetrica, no hermitiana
% imagScale controla la     fuerza     de la parte imaginaria ( 0  .1 -- 0.5)

if nargin<2, imagScale = 0.3; end

% --- Parte real ---
C = randn(n);                  % aleatoria real
R = C.'*C + n*eye(n);

% --- Parte imaginaria (simetrica) ---
S0 = randn(n);  S = (S0+S0.')/2;       % simetrica real
S  = imagScale * S / norm(S,2);        % control de magnitud

A = R + 1i*S;                  % matriz resultante
end
```

Bibliografía

[1] Bermúdez, A., Rodríguez, R. y Salgado, P. (2005). Numerical solution of eddy current problems in bounded domains using realistic boundary conditions. *Computer Methods in Applied Mechanics and Engineering*, **194**, 411–425.

[2] Boisvert, R. F., Pozo, R., Remington, K., Dongarra, J. y Bischof, C. (1997). "The Matrix Market: A Web Resource for Test Matrices." Disponible en `https://math.nist.gov/MatrixMarket/`.

[3] Ciaramella, G. and Gander, M. J. (2022). *Iterative Methods and Preconditioners for Systems of Linear Equations*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia.

[4] Freund, R. W. (1992). Conjugate Gradient-Type Methods for Linear Systems with Complex Symmetric Coefficient Matrices. *SIAM Journal on Scientific and Statistical Computing*, **13**, 425–448.

[5] Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*, 2nd ed., Society for Industrial and Applied Mathematics (SIAM), Philadelphia.

[6] Trefethen, L. N. y Bau III, D. (1997). *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics (SIAM), Philadelphia.

[7] van Rienen, U. (2001). *Numerical Methods in Computational Electrodynamics: Linear Systems in Practical Applications*, Lecture Notes in Computational Science and Engineering, Vol. 12, Springer, Berlin.

[8] van der Vorst, H. A. y Melissen, J. B. M. (1990). A Petrov–Galerkin type method for solving $Az = b$, where A is symmetric complex. *IEEE Transactions on Magnetics*, **26**, 706–708.