Universidad de Valladolid
Facultad de Ciencias

Trabajo Fin de Máster

Master en Investigación en Matemáticas

La familia de métodos IDR (Induced Dimension Reduction) para resolver grandes sistemas lineales.

Autor: Iván Corredor Mazo

Tutor: Luis María Abia Llera

.

Autorización del tutor

Luis M. Abia Llera, Catedrático de Matemática Aplicada de la Universidad de Valladolid,

CERTIFICA: que la presente Memoria *La familia de métodos IDR (Induced Dimension Reduction) para resolver grandes sistemas lineales* ha sido realizada por D. Iván Corredor Mazo en la Universidad de Valladolid y constituye la memoria de su trabajo fin de Máster, preceptiva para la finalización de sus estudios del Máster en Investigación en Matemáticas de dicha Universidad.

Valladolid, 30 de junio de 2017

Fdo.:

Luis M. Abia Llera

Introducción

Los métodos basados en los subespacios de Krylov son extensamente usados como métodos iterativos para hallar la solución de sistemas lineales de ecuaciones

$$Ax = b.$$

El método más conocido dentro de éstos no es otro que el gradiente conjugado o GC y aunque su importancia está justificada solo es válido en el caso en que la matriz A es simétrica y definida positiva. Esto nos motiva a buscar nuevos métodos que generalicen el comportamiento del GC y sea válido para una matriz general.

La búsqueda de un método eficiente de Krylov se afronta de dos maneras distintas. La primera de ellas se trata de elaborar un método que en cada iteración minimice la norma del residuo del sistema sobre un subespacio. Dentro de éstos el método que más destaca es el método de mínimo residuo, GMRES, el cual minimiza los residuos del sistema sobre los espacios de Krylov. La segunda corriente es aquella en la que pedimos al método recurrencias cortas, es decir, que exijan almacenar pocos vectores para obtener una solución factible. Dentro de este grupo destaca el bigradiente conjugado, BiGC, que es equivalente al GC en el caso de que A sea simétrica definida positiva. Sin embargo, este método no es óptimo para todas las matrices en general y nos vemos en la necesidad de buscar métodos más rápidos. Los métodos BiGCSTAB son una variante de éstos que mejoran el comportamiento de los sucesivos iterantes.

La búsqueda de métodos más rápidos se focaliza en los métodos del tipo BiGCSTAB, es decir, que generalicen la idea que éste presenta. El primer método de la clase de los métodos de reducción inducida de la dimensión, IDR por 'induced dimension reduction', se deduce directamente del BiGCSTAB multiplicando el polinomio del método por otro polinomio que mejore las propiedades del primero. A lo largo de los años este método ha sido eclipsado por otros del mismo tipo aunque no ha sido olvidado completamente, quizás por su estrecha relación con el BiGCSTAB.

En este trabajo vamos a estudiar la familia de métodos IDR(s), utiliza la idea del método IDR original pero con mejoras sustanciales para acelerar la convergencia.

A la vista de lo anterior hemos dividido esta memoria en tres capítulos teóricos.

El capítulo 1 habla de los métodos de proyección en general y los métodos de Krylov describiendo cómo funcionan éstos de una manera general. En el capítulo 2 vamos a estudiar los métodos del GC, BiGC, BiGCSTAB y GMRES, debido a su importancia y relación con el IDR. En un tercer capítulo vamos a describir la familia de métodos IDR(s) y a estudiar algunas de sus características. Por último en el capítulo 4 vamos a aplicar los métodos estudiados a lo largo del trabajo a diversos problemas comparando los resultados entre ellos.

Índice general

Capítulo 1

Métodos de Proyección

Para poder entender con mayor claridad la familia de métodos IDR antes debemos conocer algunas propiedades generales de los métodos de proyección. Dentro de los métodos de proyección prestaremos especial atención a los denominados métodos basados en subespacios de Krylov.

1.1. Métodos de proyección

Queremos llegar a resolver un sistema lineal de la forma

$$Ax = b, \tag{1.1}$$

donde $A \in \mathbb{R}^{N\times N}$, $b \in \mathbb{R}^N$ y $N \in \mathbb{N}$ grande. Para ello consideramos $\mathcal{K} \subseteq \mathbb{R}^N$, un subespacio de dimensión $n \ll N$. Nuestro objetivo es proyectar la matriz A de gran dimensión sobre el subespacio de dimensión n con la esperanza de que en este nuevo subespacio seamos capaces de obtener una buena aproximación a la solución de (1.1).

En general, para hacer ésto tenemos que imponer n condiciones que típicamente tienen que ver con la ortogonalidad del residuo del sistema a n vectores linealmente independientes: es decir, si $r = b - Ax$, entonces,

$$r \perp v_i, \quad i = 1, \ldots, n.$$

Aquí $v_i,\ i = 1, \ldots, n$ son vectores linealmente independientes. Esto nos da un nuevo subespacio, $\mathcal{L} = \langle v_1, \ldots, v_n \rangle \subseteq \mathbb{R}^N$ al que llamamos subespacio de condiciones.

En estas condiciones pueden ocurrir dos cosas,

1. $\mathcal{L} = \mathcal{K}$, en cuyo caso se denomina **proyección ortogonal**.

2. $\mathcal{L} \neq \mathcal{K}$, en cuyo caso se denomina **proyección oblicua**.

Nota:
El subespacio $\mathcal{L}$ es el que nos marca en que dirección proyectamos, así, en el caso **1** se trata de una proyección ortogonal al uso, mientras que en el caso **2**, la dirección de proyección no es ortogonal a $\mathcal{K}$. En principio, $\mathcal{L}$ puede ser cualquier subespacio de $\mathbb{R}^N$, sin embargo, más adelante veremos que debe de cumplir cierta relación con $\mathcal{K}$.

Podemos describir los métodos de proyección de la siguiente manera. Dada una aproximación inicial, x_0, encontrar $\hat{x}$ tal que

$$\begin{aligned} &\hat{x} \in x_0 + \mathcal{K}, \\ &b - A\hat{x} \perp \mathcal{L}. \end{aligned}$$

Si ponemos $\hat{x} = x_0 + \delta$, el primer residuo, r_0, esta definido por $r_0 = b - Ax_0$ y la ecuación anterior se transforma en

$$b - A(x_0 + \delta) \perp \mathcal{L} \;\; \text{ó} \;\; r_0 - A\delta \perp \mathcal{L},$$

con $\delta \in \mathcal{K}$. En otras palabras, la aproximación es solución del problema

$$\hat{x} = x_0 + \delta, \quad \delta \in \mathcal{K} \tag{1.2}$$

$$(r_0 - A\delta, w) = 0, \quad \forall w \in \mathcal{L} \tag{1.3}$$

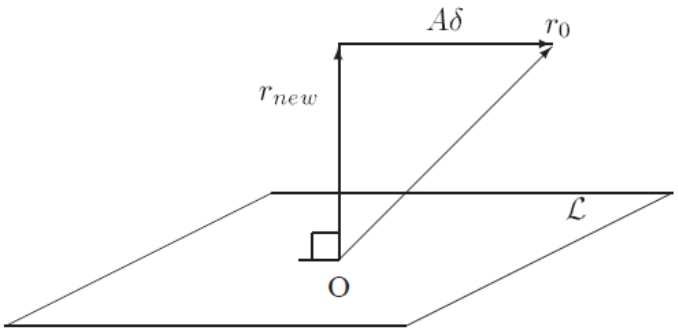

Figura 1.1: Interpretación de la condición (1.3), donde $r_{new} = r_0 - A\delta$

Lo anterior es un paso de proyección básico en su forma mas abstracta. Los métodos suelen usar sucesivamente estos pasos para encontrar aproximantes cada vez mas cercanos a la solución del sistema.

Nota:
Un ejemplo sencillo y probablemente el primero que se nos debería de ocurrir es cuando tomamos $\mathcal{K} = \mathcal{L} = \langle e_1, \ldots, e_n \rangle$. El método que resulta de computar todo lo anterior no es otro que el conocido método de Gauss-Seidel.

Para poder estudiar todo lo anterior de una manera más clara veamos como podemos interpretarlo en función de las bases de los subespacios $\mathcal{K}$ y $\mathcal{L}$.

Si tomamos $V = [v_1|\ldots|v_n], W = [w_1|\ldots|w_n] \in \mathbb{R}^{N\times n}$ donde $\mathcal{K} = \langle v_1, \ldots, v_n \rangle$ y $\mathcal{L} = \langle w_1, \ldots, w_n \rangle$, entonces (1.2) y (1.3) se reescriben como,

$$\hat{x} = x_0 + Vy,\ y \in \mathbb{R}^n \tag{1.4}$$

$$W^T(r_0 - AVy) = 0, \tag{1.5}$$

respectivamente. Podemos desarrollar lo anterior para ver que condiciones tienen que cumplir las matrices V y W para que el problema tenga solución. Reescribiendo (1.5) como $W^T r_0 - W^T AVy = 0$, resulta el sistema n-dimensional

$$W^T AVy = W^T r_0. \tag{1.6}$$

Si la matriz $W^T AV$ es no singular podemos calcular su inversa y obtener en (1.6),

$$y = (W^T AV)^{-1} W^T r_0,$$

y finalmente

$$\hat{x} = x_0 + Vy = x_0 + V(W^T AV)^{-1} W^T r_0.$$

Esta expresión nos da una forma de calcular un aproximante a la solución de $Ax = b$. Este proceso se puede iterar en la forma descrita por el algoritmo 1.

La existencia de la solución del sistema (1.6) depende de la inversibilidad de $W^T AV$. El siguiente resultado nos dice cuando esto puede ocurrir.

Proposición 1.1 *Si se tiene una de las siguientes condiciones, la matriz $W^T AV$ es invertible para cualquier W y V.*
1) *A es definida positiva y $\mathcal{L} = \mathcal{K}$.*
2) *A es no singular y $\mathcal{L} = A\mathcal{K}$.*

Algorithm 1 Algoritmo método de proyección.

while No halla convergencia **do**
  Seleccionamos $\mathcal{K}$ y $\mathcal{L}$ de dimensión n.
  Elegimos bases adecuadas $V = [v_1|\dots|v_n]$ y $W = [w_1|\dots|w_n]$, respectivamente.
  Partimos de x_n
  $r_n = b - Ax_n$
  $y = (W^T AV)^{-1} W^T r_n$
  $x_{n+1} = x_n + Vy$
  $n = n + 1$
end while

La demostración de esta proposición 1.1 es sencilla y hace uso de la existencia de una matriz, G, no singular que nos lleva V a W y puede encontrarse en su totalidad en [1].

A continuación vamos a dar dos resultados que entre otras cosas también nos permiten deducir la existencia de solución del problema.

Proposición 1.2 *Supongamos que $A \in \mathbb{R}^{N\times N}$ es simétrica definida positiva y $\mathcal{K} = \mathcal{L}$. El vector $\hat{x}$ es solución del problema de proyección sobre $\mathcal{K}$ con aproximación inicial x_0 si y solo si minimiza la A-norma (norma de la energía asociada a la matriz A) sobre $x_0 + \mathcal{K}$, es decir,*

$$E(\hat{x}) = \min_{x\in x_0+\mathcal{K}} E(x),$$

donde,

$$E(x) = (A(y-x), y-x)^{1/2}$$

e y es la solución exacta del sistema lineal $Ax = b$.

Demostración:

En las condiciones de la proposición como consecuencia de que la matriz A sea simétrica definida positiva, claramente la expresión $E(x)$ define una norma en $\mathbb{R}^N$, que se suele llamar norma de la energía asociada a A, además esta norma proviene de un producto escalar que viene definido por

$$\forall x, y \in \mathbb{R}^N,\ (x, y)_A = (Ax, y),$$

por lo que tenemos también el concepto de ortogonalidad con respecto a la matriz A. Denotaremos esta norma por $\|\cdot\|_A$. Por tanto, desde este punto de vista, minimizar el funcional $E(x)$ es equivalente al problema

$$\min_{x \in x_0 + \mathcal{K}} \|y - x\|_A.$$

La solución a este problema, $\hat{x}$, existe y cumple que

$$y - \hat{x} \perp_A \mathcal{K},$$

y está caracterizada por las siguientes condiciones equivalentes

$$\begin{aligned}
(y - \hat{x}, w)_A &= 0, \quad \forall w \in \mathcal{K}, \\
(A(y - \hat{x}), w) &= 0, \quad \forall w \in \mathcal{K}, \\
(b - A(\hat{x}), w) &= 0, \quad \forall w \in \mathcal{K}, \\
(b - A(x_0 + \delta), w) &= 0, \quad \forall w \in \mathcal{K}, \\
(r_0 + A\delta, w) &= 0, \quad \forall w \in \mathcal{K},
\end{aligned}$$

y llegamos a lo que se quería demostrar, $\hat{x}$ es solución de (1.2) y (1.3). El argumento es válido en ambas direcciones y por tanto, la proposición queda demostrada. □

Proposición 1.3 *Sea $A \in \mathbb{R}^{N \times N}$ una matriz arbitraria y supongamos que $\mathcal{L} = A\mathcal{K}$. El vector $\hat{x}$ es solución del problema de proyección oblicua sobre $\mathcal{K}$ con aproximación inicial x_0 si y solo si minimiza la norma 2 del residuo $b - Ax$ sobre $x_0 + \mathcal{K}$, es decir,*

$$R(\hat{x}) = \min_{x \in x_0 + \mathcal{K}} R(x),$$

donde,

$$R(x) = \|b - Ax\|_2.$$

Demostración:

La demostración de esta proposición es exactamente igual que la de la proposición anterior salvo que en este caso ya sabemos que $R(x)$ es una norma (norma euclídea) que proviene de un producto escalar (producto escalar habitual) y por tanto solo tenemos que repetir el razonamiento. □

Observaciones:
1) Los dos resultados anteriores nos dan una manera alternativa de buscar la solución exacta. En las condiciones que hemos impuesto los dos problemas de aproximación óptima tienen solución.
2) La proposición anterior nos dice que si A es singular entonces lo que estamos resolviendo es el problema de aproximación mínimos cuadrados. En este caso el método iterativo nos da la solución del sistema lineal siempre y cuando b esté en la imagen de A.

1.2. Métodos basados en los subespacios de Krylov

Resolver un sistema lineal de la forma $Ax = b$ mediante un método de proyección general requiere iterar el siguiente procedimiento: conocida la aproximación inicial x_0 a la solución, encontrar una aproximación x_n en el subespacio afín $x_0 + \mathcal{K}$, de dimensión finita n, requiriendo la condición de ortogonalidad para el residuo

$$b - Ax_n \perp \mathcal{L},$$

donde, de nuevo $\mathcal{L}$ es un subespacio de dimension n. Un método basado en subespacios de Krylov toma como subespacio $\mathcal{K}$ un subespacio relacionado con el n-ésimo subespacio de Krylov.

Definición 1.4 *Sean $A \in \mathbb{R}^{N\times N}$, $v \in \mathbb{R}^N$ no nulo y $n \in \mathbb{N}$. Definimos el n-ésimo subespacio de Krylov como el subespacio vectorial de $\mathbb{R}^N$ generado por los vectores $v, Av, \ldots, A^{n-1}v$, y lo denotamos por*

$$\mathcal{K}_n(A, v) = \left\langle v, Av, A^2v, \ldots, A^{n-1}v \right\rangle . \tag{1.7}$$

Claramente estos subespacios forman una sucesión encajada, es decir,

$$\mathcal{K}_n(A, v) \subseteq \mathcal{K}_{n+1}(A, v), \quad \forall n.$$

En virtud del Teorema de Cayley-Hamilton, A es raiz de su polinomio característico, es decir, si $p(x) = x^N + a_{N-1}x^{N-1} + \cdots + a_1x + a_0$ es el polinomio característico de A, entonces $A^N = -a_{N-1}A^{N-1} - \cdots - a_1A - a_0I$. En consecuencia $A^Nv \in \mathcal{K}_N(A, v)$ y para $m > N$, $A^mv \in \mathcal{K}_N(A, v)$. No tiene sentido, por tanto, considerar en (1.7) valores $n > N$.

Si escogemos dichos subespacios como $\mathcal{K}$ en nuestro problema de proyección vamos a obtener una sucesión de iterantes, $x_n \in x_0+\mathcal{K}_n(A, v)$,

$n = 1, \cdots, N$, que van a converger a la solución del problema, dándonos un método directo que acaba en a lo sumo N pasos.

En la práctica puede ocurrir que exista un entero $m < N$, que depende de v, tal que

$$\mathcal{K}_n = \mathcal{K}_m, \quad \forall n > m. \tag{1.8}$$

Ésto, en principio, nos dice que en algunos casos tendríamos convergencia a la solución en un subespacio de dimensión menor que la del espacio ambiente y por tanto una reducción del coste de los métodos. Esto cobra especial importancia en nuestro caso que trataremos con matrices dispersas y de gran tamaño.

Nota:
En general, este entero no lo conoceremos pero sí está caracterizado, para más detalles ver [10]. Aunque importante teóricamente, en la práctica no esperamos alcanzar dicho m y finalizar en un número de pasos mucho menor.

En adelante vamos a considerar $v = r_0$ y pondremos $\mathcal{K}_n = \mathcal{K}_n(A, r_0)$ siempre que lo necesitemos y no haya lugar a confusión.

Recapitulando, si partimos de un iterante $x_{n-1} \in x_0 + \mathcal{K}_{n-1}(A, r_0)$, un paso puede describirse de manera general como, encontrar $x_n \in x_{n-1} + \mathcal{K}_n(A, r_0)$ tal que

$$(r_0 - A\delta, w) = 0, \quad \forall w \in \mathcal{L}_n \ \text{ y } \ \delta \in \mathcal{K}_n.$$

Si tenemos en cuenta los generadores de $\mathcal{K}_n$, entonces encontrar $x_n = x_{n-1} + \sum_{j=0}^{n-1} \alpha_j A^j r_0$ tal que

$$(r_0 - \sum_{j=0}^{n-1} \alpha_j A^{j+1} r_0, w) = 0, \quad \forall w \in \mathcal{L}_n.$$

En consecuencia para todo n tenemos la siguiente aproximación,

$$x = A^{-1}b \approx x_n = x_0 + P_{n-1}(A)r_0, \tag{1.9}$$

donde $P_{n-1}(A)$ es cierto polinomio de grado $n-1$ sobre la matriz A. En el caso más simple, $x_0 = 0$, la aproximación viene dada por $P_{n-1}(A)r_0$. Esto también nos da una expresión polinómica en A para los residuos,

$$r_n = (I + AP_{n-1}(A))r_0. \tag{1.10}$$

Estos métodos pueden ser vistos como un proceso de generación de los polinomios en A que caracterizan los diversos iterantes (desde este

punto de vista se abordará la obtención del BiGCSTAB y el propio IDR original).

Si nos fijamos en el algoritmo general de un método de proyección que estudiamos en la sección anterior nos damos cuenta que un paso importante es la elección de las bases de los subespacios $\mathcal{K}$ y $\mathcal{L}$ que vamos a manejar, por lo tanto es necesario estudiar el llamado *Proceso de Arnoldi*.

Es conocido que dada $A \in \mathbb{R}^{N\times N}$ siempre se puede encontrar una matriz $Q \in \mathbb{R}^{N\times N}$ ortogonal de forma que

$$Q^T AQ = H, \tag{1.11}$$

donde H tiene forma de Hessenberg superior. Se dice que A se ha reducido a forma de Hessenberg superior mediante transformaciones ortogonales. Dada $A \in \mathbb{R}^{N\times N}$ el proceso de Arnoldi calcula su reducción a forma de Hessenberg.

Para describir el método vamos a reescribir la relación (1.11) en el formato equivalente $AQ = QH$, por ser Q ortogonal. Si denotamos por $q_1, \ldots, q_N$ los vectores columna de Q y reescribimos en función de éstos la relación anterior, obtenemos

$$Aq_n = \sum_{i=1}^{n+1} h_{in} q_i \ , \ 1 \leq n \leq N-1. \tag{1.12}$$

Supongamos ahora que $h_{n+1,n} \neq 0$, $1 \leq n \leq N-1$. De la relación (1.12) se concluye el siguiente resultado.

Teorema 1.5 *En las condiciones anteriores, para* $1 \leq n \leq N-1$ *se tiene*

$$\langle q_1, \ldots, q_n \rangle = \left\langle q_1, Aq_1, A^2 q_1, \ldots, A^{n-1} q_1 \right\rangle .$$

Demostración:

Razonemos por inducción sobre n.
Para $n = 1$ claramente se tiene la relación,

$$\langle q_1 \rangle = \langle q_1 \rangle .$$

Supongamos cierta la identidad para $n = s$ y veamos que se cumple para $n = s + 1$.

Como, por hipótesis de inducción,

$$\langle q_1, \ldots, q_s \rangle = \left\langle q_1, Aq_1, A^2q_1, \ldots, A^{s-1}q_1 \right\rangle,$$

basta probar que $q_{s+1} \in \langle q_1, Aq_1, A^2q_1, \ldots, A^sq_1 \rangle$ y $A^sq_1 \in \langle q_1, \ldots, q_{s+1} \rangle$. De la relación (1.12) se tiene

$$h_{s+1,s}q_{s+1} = Aq_s - \sum_{i=1}^{s} h_{is}q_s.$$

El sumatorio claramente está en $\langle q_1, \ldots, q_s \rangle = \langle q_1, Aq_1, A^2q_1, \ldots, A^{s-1}q_1 \rangle$ y $Aq_s \in \langle Aq_1, A^2q_1, \ldots, A^sq_1 \rangle \subseteq \langle q_1, Aq_1, A^2q_1, \ldots, A^sq_1 \rangle$, luego al ser $h_{s+1,s}$ no nulo,

$$q_{s+1} \in \left\langle q_1, Aq_1, A^2q_1, \ldots, A^sq_1 \right\rangle.$$

Por otro lado, como $A^{s-1}q_1 \in \langle q_1, \ldots, q_s \rangle$ entonces

$$A^sq_1 = A(A^{s-1}q_1) \in \langle Aq_1, \ldots, Aq_s \rangle.$$

Utilizando de nuevo (1.12) concluimos que

$$Aq_l \in \langle q_1, \ldots, q_{l+1} \rangle \text{ para } 1 \leq l \leq s,$$

lo cual implica que $A^sq_1 \in \langle q_1, \ldots, q_{s+1} \rangle$. □

A los vectores $q_1, \ldots, q_n$ se los conoce como *vectores de Arnoldi*, que por ser Q ortogonal forman una base ortonormal de $\mathcal{K}_n(A, q_1)$.
Volviendo de nuevo a (1.12), si despejamos el último término de la suma obtenemos

$$h_{n+1,n}q_{n+1} = Aq_n - \sum_{i=1}^{n} h_{in}q_i \ , \ 1 \leq n \leq N-1. \tag{1.13}$$

Para que q_{n+1} sea ortogonal a $q_1, \ldots, q_n$ tienen que ser $h_{in} = (Aq_n)^Tq_i$, para todo i.

Definiendo

$$r_n = Aq_n - \sum_{i=1}^{n} h_{in}q_i$$

entonces, si $r_n = 0$, $Aq_n \in \langle q_1, \ldots, q_n \rangle$, $\mathcal{K}_{n+1}(A, q_1) = \mathcal{K}_n(A, q_1)$ y $\{q_1, \ldots, q_n\}$ constituyen una base ortonormal de $\mathcal{K}_{n+1}(A, q_1)$, mientras que en el caso en que $r_n \neq 0$, $h_{n+1,n} = \|r_n\|_2$ y $q_{n+1} = \dfrac{r_n}{h_{n+1,n}}$.

De este modo se calculan los n primeros vectores de Arnoldi que por (1.13) producen la factorización

$$AQ_n = Q_nH_n + h_{n+1,n}q_{n+1}e_n^T, \tag{1.14}$$

donde $Q_n = [q_1, \ldots, q_n]$ y H_n es Hessenberg superior de orden $n \times n$ y está formada por los elementos de las n primeras filas y columnas de H.

Alternativamente, siempre que nos convenga, podemos utilizar equivalentemente

$$AQ_n = Q_{n+1}\overline{H}_{n+1,n}, \tag{1.15}$$

donde $\overline{H}_{n+1,n}$ es una matriz $(n+1) \times n$ donde todos sus elementos no nulos son los elementos no nulos de la matriz H_n, resultante del proceso de Arnoldi.

Multiplicando por la izquierda por Q_n^T y teniendo en cuenta que q_{n+1} es ortogonal a las columnas de Q_n se tiene que

$$Q_n^T A Q_n = H_n + h_{n+1,n} Q_n^T q_{n+1} e_n^T = H_n.$$

Si el proceso anterior se repite hasta $n = N$ llegamos a la matriz H de (1.11). Sin embargo, si ocurre de (1.8) el proceso de Arnoldi con $q_1 = \frac{r_0}{\|r_0\|_2}$ termina para $n = m$, donde obtendríamos que $r_m = 0$ y a lo sumo deberíamos calcular m vectores.

Observaciones:
1) Para cada n, a la vista de la igualdad $Q_n^T A Q_n = H_n$, decimos que la matriz H_n es la proyección de A sobre el subespacio de Krylov $\mathcal{K}_n(A, q_1)$.
2) En general, no es cierto que $Q_n H_n Q_n^T = A$ para cada n. En efecto, tomemos (1.14) y multipliquemos por Q_n^T por la derecha, entonces

$$A = Q_n H_n Q_n^T + h_{n+1,n} q_{n+1} e_n^T Q_n^T = Q_n H_n Q_n^T + h_{n+1,n} q_{n+1} q_n^T.$$

El segundo sumando de la expresión anterior sólo es 0 cuando $h_{n+1,n} = 0$. El primer n para el que ocurre esto define el m de (1.8) y el proceso termina. En este caso $A = Q_m H_m Q_m^T$.
3) Si tenemos en cuenta (1.4), como hemos construido una nueva base del subespacio de Krylov, el problema será encontrar $y \in \mathbb{R}^n$ tal que

$$x_n = x_0 + Q_n y. \tag{1.16}$$

En resumen, podemos dar el siguiente algoritmo:

Dada $A \in \mathbb{R}^{N \times N}$ y $q_1 \in \mathbb{R}^N$ unitario, el siguiente algoritmo usa el proceso de Arnoldi para calcular la factorización $AQ_m = Q_m H_m$, donde $Q_m \in \mathbb{R}^{N \times m}$ (m es el mínimo entero tal que $\mathcal{K}_n = \mathcal{K}_m$, si $n > m$) tiene columnas ortonormales y $H_m \in \mathbb{R}^{m \times m}$ es Hessenberg superior con todos los elementos subdiagonales distintos de 0.

Algorithm 2 Algoritmo del Proceso de Arnoldi.

for $n = 1 : N$ **do**
  $z = Aq_n$
  for $i = 1 : n$ **do**
    $h_{in} = q_i^t z$
    $z = z - h_{in} q_i$
  end for
  $h_{n+1,n} = \|z\|_2$
  if $h_{n+1,n} = 0$ **then**
    $m = n$, **break**
  end if
  $q_{n+1} = \frac{z}{h_{n+1,n}}$
end for

Observaciones:
1) El algoritmo anterior no es otro que el algoritmo de Gram-Schmidt modificado para ortogonalizar Aq_n frente a los vectores $q_1, \ldots, q_n$. Esta forma del algoritmo es computacionalmente preferible al algoritmo clásico Gram-Schmidt. Basta con disponer de un algoritmo que calcule los productos Aq_n. Esto tiene especial relevancia cuando se trabaja con matrices dispersas.
3) Si la matriz A es simétrica no hace falta calcular muchos de los productos por lo que el costo operativo se reduce sustancialmente y obtenemos el llamado **Proceso de Lanczos** (que será estudiado en el siguiente capítulo). En este caso obtenemos que H_m es tridiagonal simétrica.

El siguiente resultado, que no vamos a demostrar y puede verse en [10] , permite computar $P_{n-1}(A)r_0$ en (1.9) utilizando las bases de Arnoldi.

Proposición 1.6 *Sea $A \in \mathbb{R}^{N\times N}$ y sean Q_n y H_n las matrices resultantes del proceso de Arnoldi en n pasos. Entonces, para todo polinomio $p(x)$ de grado $\leq n-1$ tenemos una aproximación exacta, es decir, se cumple que*

$$p(A)q_1 = Q_n p(H_n)e_1.$$

Las diferentes versiones de los métodos basados en los subespacios de Krylov vienen dadas por las diferentes opciones a la hora de escoger los subespacios $\mathcal{K}$ y $\mathcal{L}$ y por el apreacondicionamiento del sistema a resolver. Esta última cuestión no va a ser tratada en este trabajo.

Sin embargo, pueden encontrarse estudios al respecto en algunos de los libros de la bibliografía tales como [1] o [3] entre otros. Nosotros nos vamos a centrar en la elección de los subespacios, que será suficiente para nuestros propósitos. Por tanto, podemos agrupar todos los métodos en cuatro grandes grupos tal y como se hace en [3].

1) Aproximación de Ritz-Galerkin:
En estos métodos se toma $\mathcal{K}_n = \mathcal{L}_n = \mathcal{K}_n(A, r_0)$. Dentro de estos métodos podemos encontrar como método más significativo el método del gradiente conjugado GC por su importancia y su extendido uso. El nombre de aproximación de Ritz-Galerkin proviene del hecho de, como ya se ha visto, se trata de una proyección ortogonal.

2) Mínima norma del residuo:
En estos métodos se toma $\mathcal{K}_n = \mathcal{L}_n(A, r_0)$ y $\mathcal{K}_n = A\mathcal{K}_n(A, r_0)$. Este tipo de métodos, en esencia, busca minimizar la norma del residuo sobre el espacio $\mathcal{K}_n(A, r_0)$. El método arquetipo de estas familias es el GMRES.

3) Aproximación de Petrov-Galerkin:
Estos métodos también toman $\mathcal{L}_n \neq \mathcal{K}_n$. Se trata de proyecciones oblicuas generales. En general los problemas de Petrov-Galerkin son problemas de proyección en un subespacio distinto al subespacio de condiciones. Como veremos en el capítulo 3 la familia de métodos IDR(s) son de este tipo.

4) Mínima norma del error:
Buscan el aproximante en el espacio $x_0 + A^T\mathcal{K}_n(A^T, r_0)$ que haga que la norma euclidea del error sea mínima. A esta familia pertenece el método del gradiente conjugado.

Hacer un estudio profundo sobre todos estos tipos de métodos se escapan del objetivo de este trabajo, por lo que nosotros, interesados en la família de métodos IDR(s), únicamente vamos a estudiar con detalle los métodos del gradiente conjugado, mínimo residuo y el bigradiente conjugado, que pasamos a describir en el siguiente capítulo.

Capítulo 2

EL gradiente conjugado, el mínimo residuo y el bigradiente conjugado

Los métodos basados en los subespacios de Krylov buscan adaptarse, en principio, a dos requerimientos básicos que son:

a) Minimizar una cierta norma del vector residuo sobre un subespacio de Krylov generado por la matriz del sistema.

b) Ofrecer un bajo coste computacional por iteración y no exigir alta disponibilidad de almacenaje.

Sin embargo, esto no es siempre posible. Sólo en sistemas de ecuaciones lineales cuya matriz de coeficientes es simétrica y definida positiva, el algoritmo del Gradiente Conjugado, propuesto por Hestenes y Stiefel en 1952 y desarrollado en la práctica a partir de 1970, alcanza, salvo errores de redondeo, teóricamente la solución exacta en un número de iteraciones a lo sumo igual a la dimensión del sistema y cumple los requisitos esenciales anteriores de minimalidad y optimalidad.

Cuando la matriz del sistema no cumple las condiciones de simetría y de ser definida positiva, el método del Gradiente Conjugado no es aplicable y en general, no existen métodos que cumplan los dos requisitos anteriores simultáneamente sin añadir inconvenientes y/o desventajas. Por todo ello, los métodos utilizados para estos sistemas, optan por una solución de compromiso, bien requiriendo la minimización de la norma del residuo, o bien buscando que ofrezcan un bajo coste computacional y de almacenamiento. Los métodos del GMRES y BiGC se encuentran entre éstos respectivamente.

Comenzamos el capítulo explicando el método del gradiente conjugado.

2.1. Gradiente conjugado (GC)

El método del gradiente conjugado es con toda seguridad el método más conocido de todos los que vamos a explicar en esta memoria. De hecho puede ser estudiado y deducido sin necesidad de hablar de métodos iterativos basados en proyección o en subespacios de Krylov.

En muchos libros de texto se parte del problema de minimización expuesto en la Proposición 1.2, sin embargo, nosotros vamos a aprovechar lo estudiado en el capítulo anterior.

Para empezar a estudiar el gradiente conjugado debemos empezar diciendo que se trata de un método para calcular la solución del sistema lineal $Ax = b$, donde la matriz A es cuadrada $N \times N$, simétrica y definida positiva. El método es del tipo Ritz-Galerkin en el cual se toma $\mathcal{K} = \mathcal{L} = \mathcal{K}_n(A, r_0)$ y por tanto el proceso de Arnoldi es válido y será nuestro punto de partida.

La primera observación que debemos hacer es que en lo que sigue podemos considerar por simplicidad $x_0 = 0$. Si fuera otro el caso una simple traslación permite llegar a los mismos resultados.

Al ser la matriz A simétrica, el proceso de Arnoldi nos produce la factorización $Q_n^T A Q_n = H_n$, donde la matriz H_n resulta ser tridiagonal. Por tanto, si denotamos por $\alpha_j = h_{jj}$, $1 \leq j \leq n$ y $\beta_j = h_{j-1,j}$, $2 \leq j \leq n$, la matriz H_n se reduce a la matriz tridiagonal T_n

$$T_n = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \beta_2 & \alpha_2 & \beta_3 & \cdots & 0 \\ 0 & \beta_3 & \alpha_3 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \beta_n \\ 0 & \cdots & 0 & \beta_n & \alpha_n \end{bmatrix}.$$

Esto, nos da, en este caso, una reducción importante del proceso de Arnoldi, dando como resultado el conocido *Proceso de Lanczos*.

Algorithm 3 Proceso de Lanczos

1: Elegir un vector q_1 unitario
2: $\beta_1 = 0$, $q_0 = 0$
3: **for** $n = 1 : N$ **do**
4: $z = Aq_n$
5: $z = z - \beta_n q_{n-1}$
6: $\alpha_n = (z, q_n)$
7: $z = z - \alpha_n q_n$
8: $\beta_{n+1} = \|z\|_2$
9: **if** $\beta_{n+1} = 0$ **then**
10: $m = n$, **break**
11: **end if**
12: $q_{n+1} = \frac{z}{\beta_{n+1}}$
13: **end for**

El m que aparece en el algoritmo es el entero en el que sabemos que el proceso de Arnoldi tiene que parar, aunque como anticipamos en la descripción del proceso de Arnoldi esperamos terminar el procedimiento en un n muy inferior a dicho m, dando como resultado la factorización (1.14) con la matriz T_n en lugar de H_n.

Una vez calculada la base del subespacio de Krylov correspondiente, la condición de Ritz-Galerkin impone que

$$r_n \perp \mathcal{K}_n(A, r_0) = \langle q_1, \ldots, q_n \rangle ,$$

por lo que por construcción de la base, el vector r_n debe estar en la dirección del siguiente vector q_{n+1} del proceso de Arnoldi. Esto nos permite sustituir en la factorización resultante la matriz Q_n por $R_n = (r_0 | \ldots | r_{n-1})$, que además sabemos que es ortogonal por construcción de los vectores de Arnoldi. En este caso, buscar iterantes en el n-ésimo subespacio de Krylov equivale a buscar $y \in \mathbb{R}^n$, tal que $x_n = R_n y$. Entonces,

$$\begin{aligned} R_n^T(Ax_n - b) &= 0, \\ R_n^T A x_n - R_n^T b &= 0, \\ R_n^T A R_n y - R_n^T b &= 0, \\ R_n^T R_n T_n y &= \|r_0\|_2^2 e_1, \end{aligned}$$

Como $R_n^T R_n = diag(\|r_0\|_2^2, \ldots, \|r_{n-1}\|_2^2)$, podemos calcular el iterante x_n como

$$x_n = R_n y = R_n T_n^{-1} e_1,$$

equivalente a resolver $T_n y = e_1$ y luego poner $x_n = R_n y$.

Nota:
Si usáramos directamente la matriz Q_n recaeríamos en el método conocido como *FOM* (Full Orthogonalitation Method. Ver [3]).

El conocido algoritmo del gradiente conjugado no es otra cosa que una reinterpretación de lo anterior para realizar los cálculos de la forma más económica posible tanto en memoria como en tiempo de CPU.

Como consecuencia de la factorización resultante del método de Lanczos podemos escribir

$$R_n^T A R_n = R_n^T R_n T_n,$$

de donde concluimos que T_n es también definida positiva. Podemos aplicarle la factorización LU para obtener dos matrices L_n y U_n tales que

$$T_n = L_n U_n,$$

donde L_n es bidiagonal inferior y U_n bidiagonal superior con 1's en la diagonal. Por lo tanto podemos computar el n-ésimo iterante como

$$x_n = R_n T_n^{-1} e_1 = (R_n U_n^{-1})(L_n^{-1} e_1). \tag{2.1}$$

Vamos a centrarnos en cada uno de los factores por separado.

Si escribimos,

$$L_n = \begin{bmatrix} \delta_0 & 0 & 0 & \cdots & 0 \\ \phi_0 & \delta_1 & 0 & \cdots & 0 \\ 0 & \phi_1 & \delta_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \phi_{n-2} & \delta_{n-1} \end{bmatrix},$$

Si llamamos $s = L_n^{-1} e_1$ (que podemos resolver del sistema $L_n s = e_1$), entonces las entradas del vector s pueden calcularse como

$$\begin{aligned} s_0 &= \frac{1}{\delta_0} \\ s_1 &= \frac{-\phi_0 s_0}{\delta_1} \\ &\vdots \\ s_{n-1} &= \frac{-\phi_{n-2} s_{n-2}}{\delta_{n-1}}, \end{aligned}$$

por lo que se pueden calcular de manera recursiva.

Por otra parte, si definimos $P_n = R_n U_n^{-1}$, tenemos,

$$R_n = P_n U_n = \begin{bmatrix} p_0 \cdots p_{n-1} \end{bmatrix} \cdot \begin{bmatrix} 1 & \epsilon_0 & 0 & \cdots & 0 \\ 0 & 1 & \epsilon_1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & \epsilon_{n-2} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix}, \tag{2.2}$$

por lo tanto

$$\begin{aligned} r_0 &= p_0 \\ r_{j-1} &= \epsilon_{j-2} p_{j-2} + p_{j-1} \quad 1 \leq j \leq n, \end{aligned}$$

donde el vector p_{j-1} puede ser calculado de manera recursiva como

$$p_{j-1} = r_{j-1} - \epsilon_{j-2} p_{j-2} \quad 1 \leq j \leq n. \tag{2.3}$$

En principio solo hay que computar la matriz T_n y resolver el sistema resultante mediante la factorizacion LU, pero como vamos a ver, conocer ésta explícitamente no es necesario. Si ponemos $\alpha_j = s_j$ y $\beta_j = \epsilon_j$, las recurrencias quedan de la siguiente forma,

$$p_{j-1} = r_{j-1} + \beta_{j-2} p_{j-2} \quad 1 \leq j \leq n, \tag{2.4}$$
$$x_j = x_{j-1} + \alpha_{j-1} p_{j-1} \quad 1 \leq j \leq n, \tag{2.5}$$
$$r_j = r_{j-1} - \alpha_{j-1} A p_{j-1} \quad 1 \leq j \leq n. \tag{2.6}$$

Usemos lo que conocemos acerca de los vectores que intentamos calcular. Primero, los residuos son ortogonales, luego de $r_j^T r_{j-1} = 0$, sustituyendo en (2.6), podemos despejar

$$\alpha_{j-1} = \frac{r_{j-1}^T r_{j-1}}{r_{j-1}^T A p_{j-1}}.$$

Usando (2.4) llegamos a la fórmula equivalente para α_{j-1}

$$\alpha_{j-1} = \frac{r_{j-1}^T r_{j-1}}{p_{j-1}^T A p_{j-1}} = \frac{\|r_{j-1}\|_2}{(p_{j-1}, A p_{j-1})}.$$

Para los β_{j-1} podemos deducir una expresión similar multiplicando (2.4) por A.

$$A p_{j-1} = A r_{j-1} + \beta_{j-2} A p_{j-2},$$

y sustituyendo en (2.6),

$$r_j = r_{j-1} - \alpha_{j-1} A r_{j-1} - \frac{\alpha_{j-1}\beta_{j-2}}{\alpha_{j-2}}(r_{j-2} - r_{j-1}).$$

Como los residuos son ortogonales,

$$\beta_{j-2} = \frac{r_{j-1}^T r_{j-1}}{r_{j-2}^T r_{j-2}}.$$

En consecuencia obtenemos el conocido algoritmo del gradiente conjugado. Aunque sabemos que es un método directo que acaba en N pasos en aritmética exacta, pararemos el método cuando el tamaño del residuo sea suficientemente pequeño.

Algorithm 4 Algoritmo del método del gradiente conjugado.

1: Aproximación inicial x_0
2: $r_0 = b - Ax_0$
3: $p_0 = r_0$
4: $n = 1$
5: Fijamos precisión y $maxit$
6: **while** error>precisión y $n < maxit$ **do**
7: $z = r_{n-1}^T r_{n-1} (= \|r_{n-1}\|^2)$
8: $w = p_{n-1}^T A p_{n-1}$
9: $\alpha_n = \frac{z}{w}$
10: $x_n = x_{n-1} + \alpha_n p_{n-1}$
11: $r_n = r_{n-1} - \alpha_n A p_{n-1}$
12: $w = r_n^T r_n (= \|r_n\|^2)$
13: $\beta_n = \frac{w}{z}$
14: $p_n = r_n + \beta_n p_{n-1}$
15: computamos el error
16: $n = n + 1$
17: **end while**

Observaciones:
1) Por supuesto, de los coeficientes α_j y β_j se puede recuperar la matriz T_n, aunque esto solo tiene interés si buscamos la expresión explícita de la matriz. Ver [3].
2) Como ya indicamos, los métodos iterativos buscan minimizar la norma del error para obtener convergencia y hacer ésto mediante recurrencias cortas para no obtener costos operativos excesivamente caros.

El método del gradiente conjugado reúne esas dos condiciones, pero además, es el único método con estas características. Esto se conoce como el *Teorema de Faber y Manteuffel*, que podemos encontrar en [5]. Esto hace que si la matriz del sistema es definida positiva este método sea la mejor opción para resolverlo.
3) El método del GMRES está pensado para obtener mejores resultados a la hora de minimizar la norma del error, mientras que el BiGC esta pensado para minimizar la longitud de las recurrencias.

Se ha implementado dicho método en MATLAB y se han llevado a cabo diversos experimentos que veremos en el capítulo 4.

2.2. Método bigradiente conjugado (BiGC)

El método del bigradiente conjugado busca aproximantes en el espacio $\mathcal{K}_n(A, r_0)$ con residuos ortogonales a $\mathcal{K}_n(A^T, r_0)$. Para poder describir el algoritmo del método debemos estudiar cómo generar bases de estos subespacios que formen un sistema biortogonal. El algoritmo de **biortogonalización de Lanczos** construye estas bases recursivamente y el método BIGC y sus propiedades se deduce de este proceso de la misma forma que dedujimos el método gradiente conjugado a partir del método de ortogonalización de Lanczos. Si estamos interesados en resolver un sistema dual de la forma $A^T x^* = b^*$, el método del bigradiente conjugado también nos da la solución.

Los métodos que biortogonalizan bases tienen la ventaja de que usan formulas de recurrencia reducidas y, por tanto, el almacenamiento no aumenta con el número de iteraciones. Por contra, presentan un comportamiento irregular con convergencia con oscilaciones y abundantes ”picos”que pueden estropear las buenas propiedades de convergencia características de estos métodos de Krylov.

Describamos el método de biortogonalización de Lanczos.

El algoritmo nos da como resultado dos matrices $V_n = [v_1|\cdots|v_n]$ y $W_n = [w_1|\cdots|w_n]$ tales que los vectores v_i son base de $\mathcal{K}_n(A, r_0)$ y los vectores w_i son base de $\mathcal{K}_n(A^T, r_0)$ tales

$$W_n^T V_n = I_n \tag{2.7}$$

y

$$W_n^T A V_n = T_n, \tag{2.8}$$

Algorithm 5 Método de biortogonalización de Lanczos.

Si consideramos v_1 y w_1 tales que $(v_1, w_1) = 1$.
Escogemos $\beta_1 = \delta_1 = 0$
for $k = 1, \ldots, n$ **do**
  $\alpha_k = (Av_k, w_k)$
  $\hat{v}_{k+1} = Av_k - \alpha_k v_k - \beta_k v_{k-1}$
  $\hat{w}_{k+1} = A^T w_k - \alpha_k w_k - \delta_k w_{k-1}$
  $\delta_{k+1} = |(\hat{v}_{k+1}, \hat{w}_{k+1})|^2$
  if $\delta_{k+1} = 0$ **then**
    Paramos.
  end if
  $\beta_{k+1} = \frac{(\hat{v}_{k+1}, \hat{w}_{k+1})}{\delta_{k+1}}$
  $w_{k+1} = \frac{\hat{w}_{k+1}}{\beta_{k+1}}$
  $v_{k+1} = \frac{\hat{v}_{k+1}}{\delta_{k+1}}$
end for

donde I_n es la matriz identidad de tamaño n y

$$T_n = \begin{bmatrix} \alpha_1 & \beta_2 & 0 & \cdots & 0 \\ \delta_2 & \alpha_2 & \beta_3 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \delta_{n-1} & \alpha_{n-1} & \beta_n \\ 0 & \cdots & 0 & \delta_n & \alpha_n \end{bmatrix}$$

Nota: Es importante darse cuenta que tratamos con un conjunto biortogonal por lo que se cumple (2.7) y no debemos caer en la confusión de que, por tanto, se cumple que los conjuntos por separado son ortogonales.

Al igual que hicimos en el método de Arnoldi, de la relación (2.8) podemos deducir, con el mismo razonamiento que seguimos allí, que

$$\begin{aligned} AV_n &= V_n T_n + \delta_{n+1} v_{n+1} e_n^T \\ AW_n^T &= W_n T_n^T + \beta_{n+1} w_{n+1} e_n^T \end{aligned}$$

El Bi-GC aprovecha todo lo anterior de la misma forma que el GC aprovecha el proceso de Arnoldi, por lo tanto solo nos limitaremos a explicar cualitativamente el proceso de deducción del método. El método del bigradiente conjugado arranca el proceso de biortogonalización de Lanczos con los vectores $v_1 = \frac{r_0}{\|r_0\|_2}$ y w_1 arbitrario tal que $(v_1, w_1) \neq 0$.

En general se suele escoger $w_1 = v_1$. Si tuvieramos interés en la solución del sistema dual $Ax^* = b^*$, entonces tomaríamos $w_1 = \frac{r_0^*}{\|r_0^*\|_2}$, donde r_0^* denota el residuo del sistema dual.

Algorithm 6 Algoritmo del método del bigradiente conjugado.

Aproximación inicial x_0
$r_0 = b - Ax_0$
Elegir r_0^* adecuadamente.
$p_0 = r_0$, $p_0^* = r_0^*$
$n = 1$
Fijamos precisión y $maxit$
while error>precisión y $n < maxit$ **do**
    $z = r_{n-1}^T r_{n-1}^*$
    $w = p_{n-1}^T A(p_{n-1}^*)^T$
    $\alpha_n = \frac{z}{w}$
    $x_n = x_{n-1} + \alpha_n p_{n-1}$
    $x_n^* = x_{n-1}^* + \alpha_n p_{n-1}^*$
    $r_n = r_{n-1} - \alpha_n A p_{n-1}$
    $r_n^* = r_{n-1}^* - \alpha_n A^T p_{n-1}^*$
    $w = r_n^T r_n^*$
    $\beta_n = \frac{w}{z}$
    $p_n = r_n + \beta_n p_{n-1}$
    $p_n^* = r_n^* + \beta_n p_{n-1}^*$
    computamos el error
    $n = n + 1$
end while

Observaciones:
1) En caso de que no nos interese o no haya sistema dual la elección de r_0^* se hace de manera arbitraria siempre que se cumpla $(r_0, r_0^*) \neq 0$.
3) Es inmediato que si aplicamos el método a una matriz simétrica definida positiva y escogiéramos $r_0^* = r_0$ recaeríamos en el GC original.
3) El método del BiGC se puede modificar de diferentes maneras para conseguir mejores resultados, ver [1]. Una de dichas modificaciones, el método del BIGCSTAB, produce resultados similares al método IDR(1), el más simple de la familia que queremos estudiar. Por ello explicaremos dicha modificación.

El método del bigradiente conjugado estabilizado busca evitar problemas derivados de la aritmética flotante que van deteriorando las

condiciones de biortogonalidad entre los vectores v_n y w_n. El BiGCSTAB no usa una relación del tipo (1.10) para el residuo, en este caso se busca de la siguiente forma

$$r_n = \psi_n(A)\phi_n(A)r_0,$$

donde $\phi_n(t)$ es el polinomio de recurrencia para los residuos, r_n, asociados al BiGC y $\psi_n(t)$ es un nuevo polinomio definido recursivamente con el objetivo de suavizar el comportamiento de la convergencia del algoritmo original. Concretamente

$$\psi_{n+1}(t) = (1 - \omega_n t)\psi_n,$$

donde ω_n es un escalar que se selecciona de forma apropiada. En el BiGCSTAB w_n se toma de forma que minimiza la norma euclídea de r_n.

Considerando que:

$$\begin{aligned} r_n &= \phi_n(A)r_0, \\ p_n &= \pi_n(A)r_0, \end{aligned}$$

y

$$\begin{aligned} \phi_{n+1} &= \phi_n - \alpha_n t\pi_n(t), \\ \pi_{n+1} &= \phi_{n+1} - \beta_n t\pi_n(t), \end{aligned}$$

entonces

$$\begin{aligned} \psi_{n+1}(t)\phi_{n+1}(t) &= (1 - \omega_j t)\psi_n(t)\phi_{n+1}(t) \\ &= (1 - \omega_j t)(\psi_n\phi_n - \alpha_n t\psi_n\pi_n) \end{aligned}$$

y teniendo en cuenta que

$$\psi_n(t)\pi_n(t) = \psi_n(t)\phi_n(t) + \beta_{n-1}(1 - \omega_{n-1}t)\psi_{n-1}(t)\pi_{n-1}(t),$$

podemos definir los nuevos vectores del método como $r_n = \psi_n(A)\phi_n(A)r_0$ y $p_n = \psi_n(A)\pi_n(A)r_0$. Por último se deducen las constantes para las nuevas recurrencias de la misma forma que para el BiGC, obteniendo el siguiente algoritmo.

Algorithm 7 Algoritmo del método BiGCSTAB.

Aproximación inicial x_0
$r_0 = b - Ax_0$
Elegir r_0^* adecuadamente.
$p_0 = r_0$, $p_0^* = r_0^*$
$n = 1$
Fijamos precisión y $maxit$
while error>precisión y $n < maxit$ **do**
    $z = r_n^T r_0^*$
    $w = Ap_n^T r_0^*$
    $\alpha_n = \frac{z}{w}$
    $s_n = r_n - \alpha_n A p_n$
    $z = As_n^T s_n$
    $w = As_n^T As_n$
    $\omega_n = \frac{z}{w}$
    $x_{n+1} = x_n + \alpha_n p_n + \omega_n s_n$
    $r_{n+1} = s_n - \omega_n A s_n$
    $\beta_n = \frac{r_{n+1}^T r_0^*}{r_n^T r_0^*} \times \frac{\alpha_n}{\omega_n}$
    $p_{n+1} = r_{n+1} + \beta_n (p_n - \omega_n A p_n)$
    computamos el error
    $n = n + 1$
end while

Observaciones:
1) El método del bigradiente conjugado estabilizado como ya se ha dicho anteriormente produce resultados similares al método IDR(1) debido a la relación entre ambos métodos. Aunque en esta memoria no vamos a tratar este punto cabe destacar que la estrecha relación se debe al parecido de los polinomios de la expresión (1.10) de cada método. De hecho el polinomio $\psi_n(t)$ aparece en ambos casos. Este análisis se puede encontrar en [6].
2) El método del BiGCSTAB está implementado en MATLAB y pueden ser llamado mediante el comando `bicgstab`.
3) La deducción de las constantes y el resto de detalles del método puede encontrase en su totalidad en [1].

2.3. Mínimo residuo (GMRES)

Para acabar el capítulo vamos a explicar el GMRES, el método más actual de todos los explicados en la memoria para tratar el sistema $Ax = b$ con A no simétrica. Se trata de un método de proyección en el cual, $\mathcal{K} = \mathcal{K}_n(A, r_0)$ y $\mathcal{L} = A\mathcal{K}_n(A, r_0)$, lo que equivale a minimizar en cada iteración la norma euclidea del residuo. Así, el desarrollo del algoritmo consiste en encontrar un vector $x \in x_0 + \mathcal{K}_n(A, r_0)$ tal que

$$x = x_0 + Q_n y$$

de forma que minimice el funcional

$$J(y) = \|b - Ax\|_2.$$

Sin embargo, si tenemos en cuenta todo lo que hemos estudiado del proceso de Arnoldi entonces,

$$\begin{aligned} & b - Ax \\ & = b - A(x_0 + Q_n y) \\ & = r_0 - AQ_n y \end{aligned}$$

y usando (1.15) y $q_1 = \frac{r_0}{\|r_0\|_2}$,

$$\begin{aligned} & = \|r_0\|_2 q_1 - Q_{n+1}\overline{H}_{n+1,n} y \\ & = Q_{n+1}(\|r_0\|_2 e_1 - \overline{H}_{n+1,n} y), \end{aligned}$$

en consecuencia,

$$J(y) = \|Q_{n+1}(\|r_0\|_2 e_1 - \overline{H}_{n+1,n} y)\|_2$$

y usando la ortogonalidad de las columnas de Q_n,

$$J(y) = \|\|r_0\|_2 e_1 - \overline{H}_{n+1,n} y\|_2.$$

La solución mínimos cuadrados de $\overline{H}_{n+1,n} y = \|r_0\|_2 e_1$ determinan el nuevo iterante. Podemos mejorar la eficiencia computacional para resolver este problema de mínimos cuadrados calculando la factorización ***QR*** mediante los reflectores de Householder o rotaciones de Givens de la matriz $\overline{H}_{n+1,n}$. Es decir, calculamos una matriz ortogonal, $Q_{n+1,n+1}$ tal que

$$\overline{H}_{n+1,n} = Q^T_{n+1,n+1} R_{n+1,n}. \tag{2.9}$$

Por ejemplo como resultado de la multiplicación sucesiva de rotaciones de Givens para cancelar los términos $h_{j+1,j}$ $j = 1 \ldots n$ de $\overline{H}_{n+1,n}$, entonces

$$\begin{aligned} J(y) &= \|\|r_0\|_2 e_1 - \overline{H}_{n+1,n} y\|_2 \\ &= \|\overline{H}_{n+1,n} y - \|r_0\|_2 e_1\|_2 \\ &= \|Q^T_{n+1,n+1} R_{n+1,n} y - \|r_0\|_2 e_1\|_2 \\ &= \|R_{n+1,n} y - Q_{n+1,n+1} \|r_0\|_2 e_1\|_2. \end{aligned}$$

La solución del problema de mínimos cuadrados puede escribirse de forma explícita como

$$y = R^{-1}_{n,n} Q_{n+1,n+1} \|r_0\|_2 e_1,$$

y el nuevo iterante se puede calcular como $x_n = x_0 + Q_n y$.

Observaciones:
1) La matriz $R_{n,n}$ siempre es invertible a menos que el aproximante coincida con la solución exacta del sistema.
2) Si la matriz A es simétrica pero no necesariamente definida positiva, la matriz $H_{n+1,n}$ es tridiagonal y el método puede simplificarse, de donde obtendríamos el método conocido como MINRES (mínimo residuo), ver [3].
3) Implementar todo lo que hemos descrito conlleva un gasto grande en memoria. Para suavizar este problema cada m pasos se reinicia el algoritmo, dando como resultado la familia de métodos GMRES(m). El método GMRES es el método que requiere mayor almacenamiento de todos los métodos que vamos a estudiar.
4) Al igual que los métodos anteriores este método esta implementado en MATLAB y puede ser llamado por `gmres`. Aún así, en el apéndice de programas hemos implementado tanto el GMRES como el GMRES(m).

5) El método está pensado para reducir la norma del residuo, lo que dará como resultado un método de gran velocidad a costa de mayor gasto de almacenamiento ya que carece de recurrencias cortas.

Como resultado de todo lo anterior obtenemos el siguiente algoritmo general para el GMRES(m). El algoritmo para el GMRES básico (full GMRES, en algunos textos) puede ser deducido fácilmente eliminando el segundo 'for'.

Algorithm 8 Algoritmo del GMRES(m).

Aproximación inicial x_0
$r_0 = b - Ax_0$
$x = x_0$
for $j = 1, 2, \ldots$ **do**
  $\beta = \|r\|_2$, $q_1 = \frac{r}{\beta}$, $\hat{b} = \beta e_1$
  for $i = 1, \ldots, m$ **do**
    $w = Aq_i$
    for $k = 1, \ldots, i$ **do**
      $h_{k,i} = q_k^T w$, $w = w - h_{k,i} q_k$
    end for
    $h_{i+1,i} = \|w\|_2$, $q_{i+1} = \frac{w}{h_{i+1,i}}$
    $r_{1,i} = h_{1,i}$
    for $k = 2, \ldots, i$ **do**
      $\gamma = c_{k-1} r_{k-1,i} + s_{k-1} h_{k,i}$
      $r_{k,i} = -s_{k-1} r_{k-1,i} + c_{k-1} h_{k,i}$
      $r_{k-1,i} = \gamma$
    end for
    $\delta = (r_{i,i}^2 + h - i + 1, i^2)^{1/2}$, $c_i = \frac{r_{i,i}}{\delta}$, $s_i = \frac{h_{i+1,i}}{\delta}$
    $r_{i,i} = c_i r_{i,i} + s_i h_{i+1,i}$
    $\hat{b}_{i+1} = -s_i \hat{b}_i$, $\hat{b}_i = c_i \hat{b}_i$
    $\rho = \|\hat{b}_{i+1}\|$
    if ρ es suficientemente pequeños **then**
      break
    end if
  end for
  $n_r = m$, $y_{n_r} = \frac{\hat{b}_{n_r}}{r_{n_r,n_r}}$
  for $k = n_r - 1, \ldots, 1$ **do**
    $y_k = \frac{\hat{b}_k - \sum_{i=1}^{n_r} r_{k,i} y_i}{r_{k,k}}$
  end for
  $x = x + \sum_{i=1}^{n_r} y_i q_i$
  if ρ es suficientemente pequeños **then**
    break
  end if
  $r = b - Ax$
end for

Capítulo 3

Los métodos IDR(s)

La familia de métodos IDR(s) se trata de métodos de proyección que, a diferencia de los métodos explicados anteriormente en la memoria, no usan como espacios de proyección espacios de Krylov. Los espacios de Krylov seguirán estando involucrados ya que, al igual que los métodos BiGC y GMRES, los métodos IDR(s) intentan generalizar las propiedades del método del GC.

En la primera sección del capítulo vamos a describir de manera general los métodos IDR(s). Como veremos, la familia de métodos IDR(s) dan problemas para valores grandes de s y es por ello que nos vemos obligados a modificar el método. En la segunda sección del capítulo nos dedicaremos a estudiar este punto. Por último, para cerrar el capítulo demostraremos que efectivamente se trata de un método de proyección Petrov-Galerkin.

3.1. Los métodos IDR(s) genéricos.

Para describir los métodos IDR(s) se antoja necesario estudiar el teorema IDR, resultado en el que están basados. Llamaremos teorema IDR al siguiente resultado que, en realidad, es una generalización del resultado que se puede encontrar en [4]

Teorema 3.1 Teorema IDR
Sean $A \in \mathbb{R}^{N\times N}$, $v_0 \in \mathbb{R}^N$ no nulo, $\mathcal{G}_0$ el subespacio de Krylov $\mathcal{K}_N(A, v_0)$ y S cualquier subespacio propio de $\mathbb{R}^N$ tal que S y $\mathcal{G}_0$ no comparten ningún subespacio invariante por A. Definimos la secuencia $\mathcal{G}_j,\ j = 1, 2, \ldots,$ como

$$\mathcal{G}_j = (I - \omega_j A)(\mathcal{G}_{j-1} \cap S),$$

donde ω_j son escalares no nulos. Entonces podemos afirmar que
1) $\mathcal{G}_{j+1} \subseteq \mathcal{G}_j \ \ \forall j > 0$.
2) $\mathcal{G}_j = 0$ *para algún* $j \leq N$.

Demostración:

Primero vamos a demostrar **1)** por inducción.

Primero para $j = 1$ tenemos

$$\begin{aligned} \mathcal{G}_1 &= (I - \omega_1 A)(\mathcal{G}_0 \cap S) \\ &= (I - \omega_1 A)(\mathcal{K}_N(A, v_0) \cap S) \\ &\subseteq (I - \omega_1 A)(\mathcal{K}_N(A, v_0)), \end{aligned}$$

donde

$$\mathcal{G}_1 = (I - \omega_1 A)(\mathcal{K}_N(A, v_0)) = \left\langle (I - \omega_1 A)v_0, \ldots, (A^{N-1} - \omega_1 A^N)v_0 \right\rangle \subseteq \mathcal{G}_0.$$

Supongamos ahora que $\mathcal{G}_j, \subseteq \mathcal{G}_{j-1} \ \ j > 0$. Veamos entonces $\mathcal{G}_{j+1} \subseteq \mathcal{G}_j$.

Sea $x \in \mathcal{G}_{j+1} = (I - \omega_{j+1}A)(\mathcal{G}_j \cap S)$, entonces $x = (I - \omega_{j+1}A)y$ con $y \in (\mathcal{G}_j \cap S)$. Como $\mathcal{G}_j \subset \mathcal{G}_{j-1}$, $y \in (\mathcal{G}_{j-1} \cap S)$. Entonces $(I - \omega_j A)y \in \mathcal{G}_j$ y en consecuencia

$$y - (y - \omega_j Ay) = \omega_j Ay \in \mathcal{G}_j \Rightarrow Ay \in \mathcal{G}_j.$$

Por lo tanto

$$x = (I - \omega_{j+1}A)y = y - \omega_{j+1}Ay \in \mathcal{G}_j,$$

como queríamos demostrar.

Para acabar probemos **2)**.

Por **1)** sabemos que $\mathcal{G}_{j+1} \subseteq \mathcal{G}_j, \ \ j > 0$, luego tenemos dos posibilidades, o bien que $\mathcal{G}_{j+1}$ es un subespacio propio de $\mathcal{G}_j$ o que sean iguales.

En el primer caso $\mathcal{G}_{j+1} \subset \mathcal{G}_j$ y sabemos $dim(\mathcal{G}_{j+1}) < dim(\mathcal{G}_j)$ mientras que el segundo caso, $\mathcal{G}_{j+1} = \mathcal{G}_j$, sólo puede darse si $\mathcal{G}_j \cap S = \mathcal{G}_j$, pues si no tendríamos que $dim(\mathcal{G}_j \cap S) < dim(\mathcal{G}_j)$ y por tanto, $dim(\mathcal{G}_{j+1}) < dim(\mathcal{G}_j)$. En consecuencia $\mathcal{G}_j \cap S = \mathcal{G}_j$ y tiene que ocurrir que $\mathcal{G}_j \subset S$. Además $\mathcal{G}_{j+1} = (I - \omega_{j+1}A)(\mathcal{G}_j \cap S) = (I - \omega_{j+1}A)\mathcal{G}_j$, lo que implica que $\mathcal{G}_j$ es subespacio invariante de A.

Por lo tanto sabemos que $\mathcal{G}_j \subset S$ y $\mathcal{G}_j \subset \mathcal{G}_0$. Por hipótesis S y $\mathcal{G}_0$ no comparten ningún subespacio invariante no nulo de A, luego $\mathcal{G}_j = \{0\}$.

En consecuencia la dimension de $\mathcal{G}_j$ se reduce en cada paso o $\mathcal{G}_j = \{0\}$. Por último, como $dim(\mathcal{G}_0) \leq N$, la reducción de las dimensiones tiene que producirse en menos de N pasos, por lo tanto $j \leq N$ para el j tal que $\mathcal{G}_j = \{0\}$ y el teorema queda probado.

□

Observaciones:
1) La hipótesis de que S y $\mathcal{G}_0$ no compartan un subespacio invariante no nulo de A no es una restricción fuerte debido a que $\mathcal{G}_0$ es el subespacio de Krylov completo de dimensión N. Todos los subespacios invariantes de A en $\mathcal{G}_0$ son unidimensionales. Por tanto, si S es escogido de forma aleatoria el que uno de esos subespacios esté en S tiene una probabilidad nula.
2) El teorema anterior nos dice que los espacios $\mathcal{G}_j$ son buenos candidatos sobre los que proyectar los residuos de nuestros iterantes con la seguridad de que en uno de ellos llegaremos a cero.
3) En general, estamos pensando en sistemas de gran dimensión y dispersos por lo que el hecho de que pueda existir un $j \leq N$ para el cual $\mathcal{G}_j = \{0\}$, el método que vamos a describir podría acabar mucho antes de llegar a la dimensión del problema.

Dado $Ax = b$, sabemos por el capítulo anterior que un método basado en subespacios de Krylov produce iterantes $x_n \in x_0 + \mathcal{K}_n$ tales que $x_n = x_0 + P_{n-1}(A)r_0$, donde x_0 es una aproximación inicial a la solución del sistema. Por lo tanto

$$\begin{aligned} r_n &= b - Ax_n \\ &= b - A(x_0 + P_{n-1}(A)r_0) \\ &= (I - AP_{n-1}(A))r_0 \\ &= \hat{P}_n(A)r_0. \end{aligned}$$

Luego r_n puede ser escrito como $\hat{P}_n(A)r_0$, donde $\hat{P}_n$ es un polinomio de grado n con $\hat{P}_n(0) = 1$. Como resultado de lo anterior dada una recursión de los residuos r_n podemos ser capaces de conocer la correspondiente recursión para los iterantes x_n.

Si asumimos que lo anterior es posible para los residuos hasta r_n, podemos conocer x_{n+1} de la siguiente ecuación,

$$A(\Delta x_n) = -\Delta r_n = (\hat{P}_n(A) - \hat{P}_{n+1}(A))r_0, \tag{3.1}$$

donde hemos usado el operador de diferencias $\Delta x_n = x_{n+1} - x_n$.

Lo anterior es posible siempre que la diferencia de polinomios $\hat{P}_n(\tau) - \hat{P}_{n+1}(\tau)$ sea divisible por τ. Esto quiere decir que existe un polinomio $R_n(\tau)$ estrictamente de grado n tal que $\hat{P}_{n+1}(\tau) = \hat{P}_n(\tau) + \tau R_n(\tau)$. Afortunadamente, todos los métodos basados en los subespacios de Krylov cumplen esta propiedad, sencilla de comprobar partiendo de la expresión (1.9).

Las recurrencias de los métodos basados en subespacios de Krylov pueden escribirse de forma general de la siguiente forma:

$$\begin{aligned} r_{n+1} &= r_n - \alpha A v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \\ x_{n+1} &= x_n + \alpha v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta x_{n-l}, \end{aligned}$$

donde v_n es un vector de $\mathcal{K}_n(A, r_0) - \mathcal{K}_{n-1}(A, r_0)$ y $\hat{l}$ es la longitud de la recursión.

Nota:
Si $\hat{l} = n$ tenemos una recurrencia larga, lo que implica que el costo operativo y de memoria crece con n. Si en cambio fijamos $\hat{l}$ pequeño comparado con N, tenemos una recurrencia corta, que desde el punto de vista computacional es lo que deseamos.

Si generamos una sucesión de residuos r_n que forzamos a pertenecer a $\mathcal{G}_j$, donde j no decrece cuando crece n, entonces en virtud del Teorema 3.1, el sistema del que proviene la sucesión de residuos se resuelve a lo sumo en N pasos.

El residuo, r_{n+1} esta en $\mathcal{G}_{j+1}$ si

$$r_{n+1} = (I - \omega_{j+1} A) v_n,$$

donde $v_n \in \mathcal{G}_j \cap S$. Ahora, si escogemos

$$v_n = r_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}, \tag{3.2}$$

entonces la expresión para r_{n+1} se escribe como

$$r_{n+1} = r_n - \omega_{j+1} A v_n - \sum_{l=1}^{\hat{l}} \gamma_l \Delta r_{n-l}$$

que, como ya se ha visto, se corresponde con la expresión general de un método de Krylov.

Sin pérdida de generalidad, podemos asumir que el subespacio S se corresponde con el subespacio nulo por la izquierda de una matriz P de tamaño $N \times s$, es decir,

$$P = [p_1|\cdots|p_s], \quad S = Ker(P^T).$$

Como v_n está también en S, satisface que

$$P^T v_n = 0,$$

que combinado con (3.2), origina un sistema de tamaño $s \times \hat{l}$ para los coeficientes γ_l.
Bajo circunstancias normales este sistema solo tiene solución única si $\hat{l} = s$. Como consecuencia inmediata, computar el primer vector en $\mathcal{G}_{j+1}$ requiere $s+1$ vectores en $\mathcal{G}_j$ y sólo podemos esperar que r_n esté en $\mathcal{G}_{j+1}$ si $n \geq (j+1)(s+1)$. En la práctica, para generar estos s residuos intermedios haremos uso de la propiedad (3.1).

Definimos las siguientes matrices:

$$\Delta R_n = [\Delta r_{n-1}|\cdots|\Delta r_{n-s}], \tag{3.3}$$
$$\Delta X_n = [\Delta x_{n-1}|\cdots|\Delta x_{n-s}]. \tag{3.4}$$

La computación de $r_{n+1} \in \mathcal{G}_{j+1}$ puede ser implementada según el siguiente algoritmo.

Algorithm 9 Computación de residuos $r_{n+1} \in \mathcal{G}_{j+1}$.

1: Calcular $c \in \mathbb{R}^s$ de $(P^T \Delta R_n)c = P^T r_n$,
2: $v = r_n - \Delta R_n c$,
3: $r_{n+1} = v - \omega_{j+1} A v$.

Como $\mathcal{G}_{j+1} \subseteq \mathcal{G}_j$, repitiendo el proceso nos da como resultado una sucesión de residuos hasta llegar a uno que sea nulo, aunque, en la práctica sacrificaremos precisión y pararemos el programa cuando la norma del residuo sea suficientemente pequeña. Se presenta en forma de pseudocódigo en el algoritmo 10 el IDR(s) genérico.

Observaciones:
1) La primera observación que debemos hacer hace referencia a la generación de los residuos iniciales. Como ya hemos dicho hacemos uso

del método del máximo descenso. Sin embargo, esto no es necesario aunque ello favorece la generación de los vectores r_n rápidamente sin producir grandes problemas en el almacenamiento.
2) La elección de ω_{j+1}, en el cálculo del primer residuo en $\mathcal{G}_{j+1}$, puede ser arbitraria, pero debemos mantener ese valor al calcular la subsecuencia de residuos dentro de dicho subespacio. En general se suele escoger el valor que minimiza la norma de r_{n+1}, de forma similar a como lo hicimos en el BiGCSTAB.
3)El algoritmo puede interrumpirse por dos causas:
Primero puede ocurrir que el sistema lineal $s \times s$ en los coeficientes $\gamma_l, l = 1, 2, \ldots, s$, resulte inconsistente. Este tipo de situación se conoce como **interrupción de tipo I**.
La segunda causa se produce cuando el parámetro ω_j toma valores próximos a cero, lo que implica un estancamiento en la convergencia a la solución. A esta clase de interrupción se conoce como **interrupción de tipo II**. Más adelante indicaremos algunas de las mejoras propuestas para atender a estas anomalías.

El algoritmo, en principio, es finito aunque existen un par de resultados muy interesantes que nos darán información parcial de como se comporta el algoritmo en este sentido y también de su rapidez. Este último punto cobra especial interés sobre todo cuando la matriz es dispersa y de gran dimensión.

Teorema 3.2 Teorema IDR extendido
Sea A una matriz real de tamaño $N \times N$, sean $p_1, \ldots, p_s \in \mathbb{R}^N$ vectores linealmente independientes, P la matriz cuyas columnas son los vectores $p_i,\ i = 1, \ldots, s$, $\mathcal{G}_0 = \mathcal{K}_N(A, r_0)$ y la sucesión de espacios $\{\mathcal{G}_j,\ j = 1, 2, \ldots\}$ definida por

$$\mathcal{G}_j = (I - \omega_j A)(\mathcal{G}_{j-1} \cap Ker(P^T)),$$

donde ω_j son números no nulos tales que $I - \omega_j A$ es no singular.
Si $d_j = dim(\mathcal{G}_j)$, entonces la sucesión

$$\{d_j,\ j = 1, 2, \ldots\}$$

es monótona no creciente y satisface

$$0 \leq d_j - d_{j+1} \leq d_{j-1} - d_j \leq s$$

Demostración:

Algorithm 10 Algoritmo genérico de la familia de métodos IDR(s).

Seleccionamos $x_0 \in \mathbb{R}^N$, $P \in \mathbb{R}^{N \times s}$, $Tol \in (0,1)$, $maxit > 0$.
$r_0 = b - Ax_0$
Para poder arrancar el método necesitamos $s+1$ *vectores en* $\mathcal{G}_0$*, para eso usamos pasos de mínima norma.*
for $n = 0$ hasta $s-1$ **do**
    $v = Ar_n$, $\omega = (v^T r_n)/(v^T v)$
    $\Delta x_n = \omega r_n$, $\Delta r_n = -\omega v$
    $r_{n+1} = r_n + \Delta r_n$, $x_{n+1} = x_n + \Delta x_n$
end for
$\Delta R_{n+1} = (\Delta r_n, \cdots, \Delta r_0)$
$\Delta X_{n+1} = (\Delta x_n, \cdots, \Delta x_0)$
Ahora construimos los espacios $\mathcal{G}_j$, $j = 1, 2, 3, \ldots$
$n = s$
while $\|r_n\|_2 >$Tol y $n <$maxit **do**
    Bucle dentro de cada $\mathcal{G}_j$
    for $i = 0$ hasta s **do**
        Resolver c de $P^T \Delta R_n c = P^T r_n$
        $v = r_n - \Delta R_n c$
        if $i = 0$ **then**
            Entrando en el subespacio.
            $t = Av$
            $\omega = \frac{t^T v}{t^T t}$
            $\Delta r_n = -\Delta R_n c - \omega t$
            $\Delta x_n = -\Delta X_n c + \omega v$
        else
            Resto de los residuos en el subespacio.
            $\Delta x_n = -\Delta X_n c + \omega v$
            $\Delta r_n = -A\Delta x_n$
        end if
        $r_{n+1} = r_n + \Delta r_n$
        $x_{n+1} = x_n + \Delta x_n$
        $n = n + 1$
        $\Delta X_n = (\Delta x_n, \cdots, \Delta x_{n-s})$
        $\Delta R_n = (\Delta r_n, \cdots, \Delta r_{n-s})$
    end for
end while

Sea $U = \mathcal{G}_{j-1} \cap Ker(P^T)$, y sea G_{j-1} la matriz cuyas columnas forman una base de $\mathcal{G}_{j-1}$. Entonces cada $x \in \mathcal{G}_{j-1}$ puede ser escrito como $x = G_{j-1}c$ para algun vector c. Además cada $x \in U$ puede ser representado como $x = G_{j-1}c$, con c satisfaciendo $P^T G_{j-1} c = 0$. Entonces claramente $U = G_{j-1}(Ker(P^T G_{j-1}))$ y en consecuencia

$$\mathcal{G}_j = (I - \omega_j A) G_{j-1} Ker(P^T G_{j-1}).$$

Si asumimos que $(I - \omega_j A)$ es no singular, por lo que

$$d_j = dim(\mathcal{G}_j) = dim(U),$$

entonces $P^T G_{j-1}$ es una matriz de tamaño $s \times d_{j-1}$ y además

$$d_j = dim(Ker(P^T G_{j-1})) = d_{j-1} - rango(P^T G_{j-1}).$$

Por otro lado, $rango(P^T G_{j-1}) = s - dim(Ker(G_{j-1}^T P))$, luego

$$d_j = d_{j-1} - s + l \tag{3.5}$$

con $l = dim(Ker(G_{j-1}^T P)) \in [0, s]$. En consecuencia queda probado que

$$0 \leq d_{j-1} - d_j \leq s.$$

Ahora supongamos que $v \in Ker(G_{j-1}^T P)$ no nulo, entonces

$$Pv \in Ker(G_{j-1}^T),$$

y $Pv \perp \mathcal{G}_{j-1}$ por definición de $Ker(G_{j-1}^T)$. Como $\mathcal{G}_j \subset \mathcal{G}_{j-1}$, esto implica que $Pv \perp \mathcal{G}_j$ y entonces $v \in Ker(G_j^T P)$. Así $Ker(G_{j-1}^T P) \subset Ker(G_j^T P)$ y entonces

$$dim(Ker(G_{j-1}^T P)) \leq dim(Ker(G_j^T P)).$$

En consecuencia, si tenemos en cuenta (3.5) para los índices $j+1$ y j, deducimos que

$$d_{j+1} = d_j - s + l'$$

con $l' = dim(Ker(G_j^T P)) \geq l$. Entonces $d_j - d_{j+1} \leq d_{j-1} - d_j$ que prueba el teorema.

□

Nota:
El teorema anterior nos dice que la reducción de la dimensión de un subespacio de la secuencia a otro está entre 0 y s. Si la reducción es 0 significa que entonces $\mathcal{G}_j \subset Ker(P^T)$, lo que es muy poco probable ya

que la probabilidad de que $\mathcal{G}_0$ y S compartan un subespacio común es nula. En la práctica la reducción va a ser, en la mayoría de los casos, de s, la máxima posible.

Si nos fijamos en la demostración del teorema anterior podemos ver que la reducción equivale al rango de la matriz $P^T G_{j-1}$ de tamaño $s \times d_{j-1}$. Las columnas de G_{j-1} son linealmente independientes, por su condición de base. Las columnas de P son independientes por definición, por tanto, si el rango de la matriz $P^T G_{j-1}$ es menor estrictamente que s, debería existir un vector $p = Pc$, para un c no nulo, tal que $p^T G_{j-1} = 0^T$ y si $d_{j-1} > s$ esto implica que p tiene que satisfacer d_{j-1} relaciones lineales aún cuando p solo depende de s parámetros libres. Si la elección de P fuera aleatoria obviamente $d_j - d_{j-1} < s$ solo puede ocurrir con probabilidad cero. Desafortunadamente la matriz G_{j-1} no ha sido construida independientemente de la matriz P (el subespacio S esta en la construción de $\mathcal{G}_{j-1}$) y pudiera ocurrir que para una elección arbitraria de P se tuviera que $d_j - d_{j-1} < s$. Esta situación, como ya se ha indicado reiteradamente, es improbable, pero si se diera a la diferencia $s - d_{j-1} - d_j$ se la llama deficiencia en la reducción. El teorema anterior prueba que esta deficiencia es no decreciente.

Cuando la reducción de la dimesión después de un paso es exactamente s durante todo el proceso, se denomina **caso genérico**, mientras que, en caso contrario se denomina **caso no genérico**. En el caso genérico, el caso de mayor interés, tenemos el siguiente corolario que presentamos sin demostración y que goza de gran importancia.

Corolario 3.3 *En el caso de un método IDR(s) genérico, el proceso requiere de a lo sumo $N + \frac{N}{s}$ multiplicaciones de matriz por vector para calcular la solución en aritmética exacta.*

El corolario anterior nos habla del costo computacional del método ya que la multiplicación matriz por vector es la componente más costosa del algoritmo. Además el número de multiplicaciones matriz por vector coincide con el número de iteraciones del método, luego tenemos que el método acaba tras $N + \frac{N}{s}$ iteraciones, en el peor de los casos. Si $s = 1$, acabará en a lo sumo $2N$ iteraciones.

Por otra parte esto nos dice que cuanto mayor sea el parámetro de la reducción, s, a priori, mayor será la velocidad de convergencia del método, luego, cuanto mayor sea la dimensión en la que trabajemos, debemos escoger un s más grande. Por otro lado cuanto más grande sea s más vectores intermedios tendremos que calcular y mayor será el

almacenamiento. Se trata de encontrar un equilibrio para cada problema.

Nota:
En el caso de métodos no genéricos no tenemos un resultado similar y por tanto no tiene tanto interés auque ésto no debe provocarnos ningún trastorno, pues estos casos tienen probabilidades muy bajas de aparecer en la práctica. Aún así podemos encontrar alguna información útil en [6].

Hasta ahora nos hemos limitado a describir el método y alguna de sus características, el algoritmo que hemos dado teóricamente tiene su interés pero hay que puntualizar ciertas elecciones dentro del mismo para así obtener una programación satisfactoria.

La elección de la matriz P.

Sólo en el caso del GC para matrices simétricas y definidas positivas podemos encontrar un análisis de la convergencia basado en el comportamiento de los polinomios del método . Sin embargo, si la matriz no cumple dichas características, el análisis no se puede realizar. En el resto de métodos donde no necesariamente la matriz es definida positiva y simétrica parte del análisis se puede realizar si, entre otras hipótesis, el primer vector de residuos es el residuo r_0, tal y como ocurre en el BiGC. Motivados por esta elección, casi natural, del primer vector del método, tenemos varias opciones para elegir P en relación al problema.

En esta memoria se ha seguido [6], donde se asegura que, en contra de lo que uno cabría esperar, la elección de P de manera adecuada para intentar mejorar el método no da grandes resultados y aunque si que mejora el comportamiento, lo hace de manera leve. El texto asegura que después de una gran cantidad de experimentos la mejor elección para la matriz P es aquella cuyas columnas son una ortogonalización de un conjunto de vectores aleatorios. La justificación del porqué de esta elección es por robustez, ya que en este caso, el que la reducción de la dimensión sea menor que s tiene probabilidad 0. Sin entrar más en detalles sobre esta cuestión, la cual requeriría un análisis propio, en este trabajo asumiremos dicha elección para P.

En el capítulo anterior se dijo que en el caso más simple, el método IDR(1) y el BiGCSTAB tienen una estrecha relación. Efectivamente en aritmética exacta y si en la elección de P en el método IDR se toma $p_1 = r_0$, entonces recaemos en el método BiGCSTAB. Ver [6].

La elección de ω.

Ya hemos indicado que una buena elección de ω es aquella en la que la norma de r_{n+1} se hace mínima, es decir

$$\omega_{j+1} = \frac{t^T v_n}{t^T t}, \quad t = Av_n.$$

Los problemas en la computación aparecen cuando el coseno del angulo entre t y v_n es demasiado pequeño, porque entonces los ω_j son próximos a cero y la convergencia se estanca. Para mejorar este comportamiento Sleijpen y Van der Vorst propusieron (aunque no en este contexto) no utilizar exactamente el parámetro del mínimo residuo, si no incrementarlo cuando dicho coseno es más pequeño que una cantidad prefijada. Esto es,

Algorithm 11 Algoritmo de mejora de selección del parámetro ω.

1: Fijar κ
2: $t = Av_n$
3: $\omega_{j+1} = \frac{t^T v_n}{t^T t}$
4: $\rho = \frac{t^T v_n}{\|t\|\|v_n\|}$
5: **if** $|\rho| < \kappa$ **then**
6: $\quad \omega_{j+1} = \frac{\omega_{j+1}\kappa}{|\rho|}$
7: **end if**

En [8] se recomienda el valor de $\kappa = 0{,}7$.

Se ha implementado en MATLAB el algoritmo IDR(s) genérico tal y como lo describimos anteriormente. Se ha considerado el sistema lineal resultante de la discretización del probelma

$$\begin{aligned} -u'' &= 0, \qquad x \in [0,1] \\ u(0) &= u(1) = 1 \end{aligned}$$

sobre una malla con 100 nodos, es decir, consideramos

$$Ax = b, \tag{3.6}$$

donde

$$A = \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \in \mathbb{R}^{100\times 100},$$

y $b = [-1, 0, \cdots, 0, -1] \in \mathbb{R}^{100}$. Queremos resolver el problema y para ello utilizaremos el método IDR(s) para distintos valores de s.

Claramente la solución exacta es el vector cuyas entradas son todas iguales a 1 y de tamaño 100. Este problema, aunque simple, será significativo ya que incluso en un caso tan sencillo los resultados van a ser reveladores. Al computar la solución mediante el programa para distintas elecciones de s hemos obtenido los siguientes resultados.

s	$\|r_n\|_2$
$s = 1$	$1{,}343916036299533e - 11$
$s = 8$	$1{,}489410920495473e - 04$
$s = 15$	$1{,}518919415938477e + 03$
$s = 51$	$2{,}662005531516695e + 44$

Cuadro 3.1: Norma euclídea del residuo en la solución para distintas elecciones de s para una tolerancia de 10^{-16}.

En el cuadro 3.1 podemos ver que valores pequeños de s nos da residuos pequeños, sin embargo, conforme vamos dando valores más grandes los residuos se deterioran de forma importante. Esto se debe al mal acondicionamiento del sistema que tenemos que resolver en cada iteración, el cual nos produce para valores grandes de s interrupciones de tipo I del algoritmo. En la siguiente sección vamos a estudiar como podemos mejorar el método de forma que resolver el sistema no nos produzca estos errores.

3.2. Los métodos IDR(s) mejorados.

Los métodos IDR(s) son una familia de métodos muy versátiles. Disponemos, como ya hemos visto en la sección anterior, de mucha libertad a la hora de escoger los parámetros. Sin embargo, si en alguna parte tenemos más libertad para movernos, ésta es en la elección del cálculo de los residuos intermedios. En esta sección vamos a ver como podemos llevar a cabo esta elección y la manera de computarlo. Como resultado obtendremos una mejora de la familia IDR(s) original que solventará los problemas del algoritmo genérico. En [8] a esta variante del IDR(s) se la denomina IDR(s)-Ortho en referencia a las condiciones de ortogonalidad que impondremos. Sin embargo, en este trabajo hablaremos en términos del método genérico y método mejorado.

Cuando hemos llegado al subespacio $\mathcal{G}_j$ tenemos que calcular s vectores residuales con el objetivo de que el siguiente esté en $\mathcal{G}_{j+1}$. En el algoritmo genérico partimos de $r_n \in \mathcal{G}_j$ y calculamos $r_{n+1} \in \mathcal{G}_{j+1}$ como sigue:

Primero calculamos $c \in \mathbb{R}^s$ de $(P^T \Delta R_n)c = P^T r_n$,
ponemos $v = r_n - \Delta R_n c$,
y obtenemos $r_{n+1} = v - \omega_{j+1} A v$.

En este segmento del algoritmo tanto r_n como ΔR_n deben estar en $\mathcal{G}_j$ para que r_{n+1} esté en $\mathcal{G}_{j+1}$. Además la matriz $P^T \Delta R_n$ debe ser de rango s. Estos requisitos nos dan cierto margen para decidir sobre los residuos intermedios y sus diferencias.

Primero veamos qué podemos hacer con ΔR_n. En lo que sigue de sección la matriz ΔR_n va a ser sustituida por una matriz genérica $G_n = [g_{n-1}, \ldots, g_{n-s}]$ cuyas columnas también pertenecen a $\mathcal{G}_j$. De igual manera ΔX_n será sustituida por una matriz general $U_n = [u_{n-1}, \ldots, u_{n-s}]$.

El procedimiento descrito en la anterior sección puede ser usado directamente para calcular $r_{n+2} \in \mathcal{G}_{j+1}$. Como $g_i \in \mathcal{G}_j,\ i = n-1, \ldots, n-s$ y $r_{n+1} \in \mathcal{G}_{j+1} \subset \mathcal{G}_j$, entonces, calcular $c \in \mathbb{R}^s$ tal que $(P^T G_n)c = P^T r_{n+1}$, definir $v_{n+1} = r_{n+1} - G_n c$ y por último actualizar $r_{n+2} = v_{n+1} - \omega_{j+1} A v_{n+1}$ nos da como resultado $r_{n+2} \in \mathcal{G}_{j+1}$.

Además, podemos observar que el vector diferencias de residuos $(r_{n+2} - r_{n+1})$ esta en el espacio $\mathcal{G}_{j+1}$. Como $A^{-1}(r_{n+2} - r_{n+1}) = -(x_{n+2} - x_{n+1})$ hemos encontrado vectores g_{n+1} y u_{n+1}.

$$g_{n+1} = -(r_{n+2} - r_{n+1}),\ u_{n+1} = (x_{n+2} - x_{n+1}).$$

En la práctica, la computación de los vectores anteriores preceden a la computación de r_{n+2} y x_{n+2}.

La actualización de los vectores iteración puede ser computado como

$$u_{n+1} = \omega_{j+1} v_{n+1} + U_n c,$$

seguido de la implementación de g_{n+1} como

$$g_{n+1} = A u_{n+1}$$

para preservar lo máximo posible en aritmética finita la relación entre los vectores g_{n+1} y u_{n+1}. Entonces los nuevos iterante y residuo quedan

definidos por

$$x_{n+2} = x_{n+1} + u_{n+1}, \quad r_{n+2} = r_{n+1} - g_{n+1}. \tag{3.7}$$

El vector g_{n+1} esta en el espacio $\mathcal{G}_{j+1}$ y por tanto en $\mathcal{G}_j$. Esto quiere decir que podemos usar este vector para calcular los nuevos vectores en $\mathcal{G}_{j+1}$ y descartar el vector mas antiguo. Esto se puede hacer definiendo las matrices

$$G_{n+2} = [g_{n-1}, \ldots, g_{n-s+1}, g_{n+1}], \tag{3.8}$$
$$U_{n+2} = [u_{n-1}, \ldots, u_{n-s+1}, u_{n+1}], \tag{3.9}$$
$$\tag{3.10}$$

La ventaja de este procedimiento es que podemos operar en el espacio exactamente con $2s$ vectores. Repitiendo el proceso s veces obtenemos $r_{n+s+1}, g_{n+k} \in \mathcal{G}_{j+1}$ con $k = 1, \ldots, s$, y los correspondientes vectores x y u.

Hasta ahora la única diferencia con el método genérico es que en él también computamos el vector $g_n = -(r_{n+1} - r_n)$ que aparecía en las matrices G_{n+k}, $k = 1, \ldots, s$. Dejando este vector libre simplificaremos el algoritmo cuando biortogonalicemos más adelante.

En el algoritmo genérico, los vectores en el subespacio $\mathcal{G}_{j+1}$ son generados con la aplicación directa del teorema IDR. La computación del primer residuo del subespacio es la misma que la computación de los s vectores intermedios. Sin embargo, al computar estos s vectores tenemos mucha más libertad y podemos explotarla. En el algoritmo genérico los residuos son actualizados mediante la expresión

$$r_{n+k+1} = r_{n+k} - g_{n+k}\ (r_{n+k} + \Delta r_{n+k}),$$

donde r_{n+k+1},r_{n+k} y g_{n+k} están en $\mathcal{G}_{j+1}$. Pero con vistas a generar un nuevo residuo en dicho subespacio podemos reemplazar esta expresión por otra más general

$$r_{n+k+1} = r_{n+k} - \sum_{i=1}^{k} \beta_i g_{n+i} \in \mathcal{G}_{j+1}.$$

Podemos escoger los parámetros β_i para obtener las propiedades que deseamos en los residuos intermedios. En el algoritmo que vamos a describir se toman de manera que el residuo sea ortogonal a los vectores $p_1, \ldots, p_k$ del subespacio S.

La misma libertad la encontramos para computar g_{n+k}. Combinaciones lineales de vectores en $\mathcal{G}_j$ sigue estando en $\mathcal{G}_j$. Por lo tanto si denominamos

$$\hat{g} = -(r_{n+k+1} - r_{n+k})$$

entonces el vector

$$g_{n+k} = \hat{g} - \sum_{i=1}^{n-1} \alpha_i g_{n+i} \in \mathcal{G}_j$$

y puede ser usado para generar los residuos intermedios. De nuevo, los parámetros α_i pueden ser escogidos de la manera que más nos convenga. En el algoritmo que vamos a describir en esta sección, están escogidos de forma que g_{n+k} sea ortogonal a los vectores $p_1, \ldots, p_{k-1}$.

En lo anterior no hemos señalado nada acerca de los vectores necesarios para arrancar el algoritmo, es decir, los vectores de $\mathcal{G}_0$. Esto puede hacerse con cualquier otro método de Krylov. En el algoritmo genérico usamos el método de máximo descenso, y el algoritmo mejorado hace esto de manera implícita en los s primeros pasos.

Por ultimo, el escoger los parámetros α_i y β_i de manera que se cumpla

$$g_{n+k} \perp p_i, \quad i = 1, \ldots, k-1, \quad k = 2, \ldots, s \tag{3.11}$$

$$r_{n+k+1} \perp p_i, \quad i = 1, \ldots, k, \quad k = 1, \ldots, s \tag{3.12}$$

nos permite dar el algoritmo general 12.

Veamos que ventajas tiene en la implementación las relaciones (3.11) y (3.12). Al calcular el vector $v_{n+k} \in \mathcal{G}_j \cap S$, debemos resolver un sistema de la forma

$$(P^T G_{n+k})c = P^T r_{n+k}.$$

Al usar las condiciones (3.11) y (3.12) el sistema se simplificará.

Sea

$$\mu_{i,k} = p_i^T g_{n+k}, \quad i = 1, \ldots, s.$$

Como consecuencia de (3.11), $\mu_{i,k} = 0$ para $i < k$. De la misma forma si definimos

$$\phi_i = p_i^T r_{n+k}, \quad i = 1, \ldots, s.$$

Como consecuencia de (3.12), $\phi_i = 0$ para $i < k$.

Algorithm 12 Algoritmo general de la familia de métodos IDR(s) con las mejoras descritas en la sección

Necesitamos $A \in \mathbb{R}^{N\times N}; x, b \in \mathbb{R}^N; P \in \mathbb{R}^{N\times s}; TOL \in (0,1)$
Asegura x tal que $\|b - Ax\| \leq TOL \cdot \|b\|$
Calculamos $r = b - Ax$
$G = \mathbf{0} \in \mathbb{R}^{N\times s}$, $U = \mathbf{0} \in \mathbb{R}^{N\times s}$
$M = \mathbf{I} \in \mathbb{R}^{s\times s}$, $\omega = 1$

Bucle sobre los espacios $\mathcal{G}_j$
while $\|r\| > TOL$ **do**
  Computamos s vectores g_n independientes en $\mathcal{G}_j$
  for $n = 1$ hasta s **do**
    $f = P^T r$
    Resolver c de $Mc = f$
    $v = r - Gc$
    $u_n = Uc + \omega v$
    $g_n = Au_n$
    Fijamos los parámetros α_i y β_i
    $g_n = g_n - \sum_{i=1}^{n-1} \alpha_i g_i$ $u_n = u_n - \sum_{i=1}^{n-1} \alpha_i u_i$
    $r = r - \sum_{i=1}^{n} \beta_i g_i$ $x = x + \sum_{i=1}^{n} \beta_i u_i$
    $M(:, n) = P^T g_n$
    $G(:, n) = g_n$ $U(:, n) = u_n$
  end for
  $f = P^T r$
  Resolver c de $Mc = f$
  $v = r - Gc$
  $t = Av$
  seleccionamos ω
  $x = x + Uc + \omega v$
  $r = r - Gc - \omega t$
end while

Todo esto nos da que el sistema que tenemos que resolver toma la siguiente forma,

$$\begin{bmatrix} \mu_{1,1} & 0 & 0 & \cdots & 0 \\ \mu_{2,1} & \mu_{2,2} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & 0 \\ \mu_{s,1} & \mu_{s,2} & \cdots & \cdots & \mu_{s,s} \end{bmatrix} \cdot \begin{bmatrix} \gamma_1 \\ \vdots \\ \vdots \\ \vdots \\ \gamma_s \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \phi_k \\ \vdots \\ \phi_s \end{bmatrix}$$

Claramente $\gamma_1, \ldots, \gamma_{k-1}$ son nulos y el vector v_{n+k} se escribe como

$$v_{n+k} = r_{n+k} - \sum_{i=k}^{s} \gamma_i g_{n+i-s-1}.$$

Ahora, computamos los vectores u_{n+k} y g_{n+k} como

$$\begin{aligned} u_{n+k} &= \omega_{j+1} v_{n+k} + \sum_{i=k}^{s} \gamma_i u_{n+i-s-1}, \\ g_{n+k} &= A u_{n+k}. \end{aligned}$$

y hacemos el vector g_{n+k} ortogonal a los vectores $p_1, \ldots, p_{k-1}$ con el siguiente algoritmo.

Para $i = 1$ hasta $k-1$
$\alpha = \frac{p_i^T g_{n+k}}{\mu_{i,i}}$
$g_{n+k} = g_{n+k} - \alpha g_{n+i}$
$u_{n+k} = u_{n+k} - \alpha u_{n+i}$

Si nos fijamos atentamente en el algoritmo anterior podemos ver con facilidad que se trata del algoritmo de ortogonalización de Gram-Schmidt.

El siguiente residuo ortogonal a p_i, $i = 1, \ldots, k$, puede ser computado como

$$r_{n+k+1} = r_{n+k} - \frac{\phi_k}{\mu_{k,k}} g_{n+k},$$

que corresponde a una aproximación de la forma

$$x_{n+k+1} = x_{n+k} - \frac{\phi_k}{\mu_{k,k}} u_{n+k}.$$

Consideramos ahora el producto

$$\hat{\phi}_i = p_i^T r_{n+k+1}, \quad i = k, \ldots, s, \tag{3.13}$$

$\hat{\phi}_i$ será la componente i-ésima del término independiente del siguiente sistema que tendremos que resolver para el siguiente residuo. Si sustituimos en (3.13), r_{n+k+1} por la expresión que acabamos de deducir tendremos que,

$$p_i^T r_{n+k+1} = p_i^T r_{n+k} - \frac{\phi_k}{\mu_{k,k}} p_i^T g_{n+k}, \quad i = k+1, \ldots, s,$$

y por tanto,

$$\hat{\phi}_i = \phi_i - \frac{\phi_k \mu_{i,k}}{\mu_{k,k}}, \quad i = k+1, \ldots, s.$$

En consecuencia dados $\mu_{i,k}$, $i = k+1, \ldots, s$, el nuevo valor de $\hat{\phi}_i$, para el siguiente sistema puede ser calculado con una sencilla cuenta.

Todo lo anterior nos permite implementar un programa MATLAB para la familia de métodos IDR(s) con todas las mejoras incluidas y las facilidades en la implementación que acabamos de estudiar. Para ilustrar la mejora vamos a considerar el sistema (3.6) de la sección anterior y comprobamos el residuo para los mismos valores de s que tomamos allí.

s	$\|r_n\|_2$
1	$5{,}895622941186736e-12$
8	$2{,}383090339886361e-11$
15	$7{,}644590024936441e-12$
51	$6{,}325705887190952e-09$

Cuadro 3.2: Norma euclídea del residuo en la solución para distintas elecciones de s para una tolerancia de 10^{-16} con el método IDR(s) mejorado.

En el cuadro 3.2 se puede ver que a diferencia de lo que pasaba con el algoritmo genérico para $s = 15, 51$, se producen residuos que se siguen manteniendo en ordenes aceptables. En la figura 3.1 hemos representado en escala doblemente logarítmica los errores al resolver el sistema de Poisson tanto con el algoritmo genérico como con el mejorado para distintos valores de s entre 1 y 100. En ella se muestra de manera clara como hemos mejorado el comportamiento del error: para

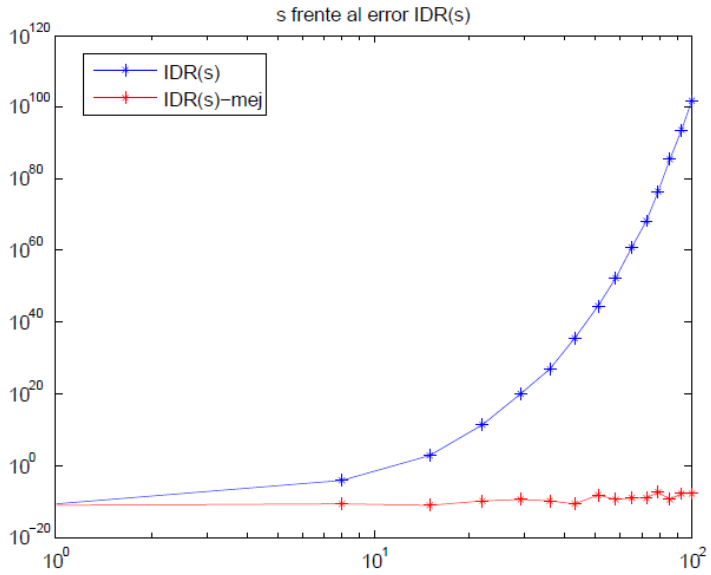

Figura 3.1: s frente al error en la resolución del sistema de Poisson con los métodos IDR(s) genérico y mejorado.

el método genérico rápidamente se dispara, mientras que el mejorado se mantiene estabilizado.

En la figura 3.1 no podemos ver como se comporta el error para diferentes s debido a la escala de la gráfica, para ello hemos generado la figura 3.2 donde mostramos únicamente el error para el método mejorado. Al igual que teníamos en el método genérico el error para este método crece aunque de forma irregular.

Como consecuencia de todo lo anterior hemos conseguido obtener mejores resultados para valores grandes de s. Ahora estudiaremos que efectivamente todos los métodos de esta familia son métodos de proyección.

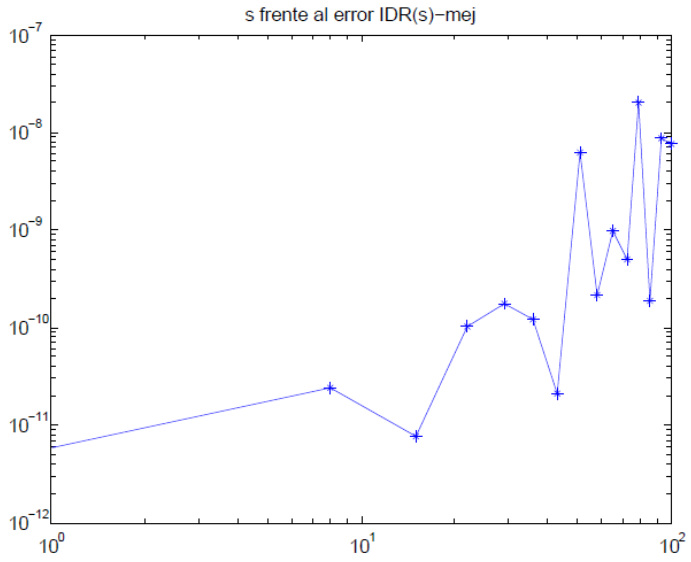

Figura 3.2: s frente al error en la resolución del sistema de Poisson con los métodos IDR(s) mejorado.

3.3. El método IDR(s) visto como método de proyección Petrov-Galerkin.

Debemos advertir que tal y como hemos introducido el método queda bastante claro que es un método basado en proyecciones sobre subespacios que tienen relación con los de Krylov a través del teorema IDR. Sin embargo, podemos estudiar más esta cuestión y dar una expresión cerrada para los espacios $\mathcal{K}$ y $\mathcal{L}$.

Por tanto, nuestro objetivo es llegar a ver que la familia de métodos IDR(s) esta constituida por métodos de proyección Petrov-Galerkin, es decir, aquellos métodos tales que $\mathcal{K} \neq \mathcal{L}$ y dar una expresión para los mismos.

Ya conocemos los espacios $\mathcal{K}$ donde tenemos que buscar los iterantes, que en este caso cambian en cada iteración. No conocemos los espacios $\mathcal{L}$, aunque podemos esperar que estos también cambien.

Definamos el polinomio

$$\Omega_j(t) = (1 - \omega_j t) \cdots (1 - \omega_1 t), \quad \omega_i \neq 0, \quad \forall i,$$

polinomio de grado j y tal que para $j = 0$ es el polinomio 1. Claramente

se trata de una función polinomial, por lo que si aplicamos el polinomio a una matriz A, la matriz resultante solo es invertible si los ceros del polinomio no pertenecen al espectro de la matriz y podemos definir los siguientes espacios.

Dados $p_1, \ldots, p_s \in \mathbb{R}^N$, los vectores que generan el subespacio S del método IDR(s), definimos la siguiente secuencia

$$W_n = (\Omega_j(A)^T)^{-1}(\mathcal{K}_j(A^T, P)) = (\Omega_n(A)^T)^{-1}(\sum_{i=1}^{s} \mathcal{K}_j(A^T, p_i)) \qquad (3.14)$$

Nota:
El método IDR(s) puede ser explicado en su totalidad partiendo de los subespacios que acabamos de definir tal y como se hace en [7].

Teorema 3.4 *En las condiciones anteriores*

$$\mathcal{G}_j \perp W_j,$$

es decir, $r \perp W_j$ *si y solo si* $r \in \mathcal{G}_j$ *y por tanto el subespacion* $\mathcal{L} = W_j$ *en cada paso.*

Demostración:

En primer lugar es inmediato que

$$\mathcal{G}_j = \left\{\Omega_j(A)v | v \perp \mathcal{K}_j(A^T, P)\right\},$$

tanto de la expresión de los $\mathcal{G}_j$ como de la definición de S como $Ker(P^T)$. Equivalentemente también podemos escribir

$$\mathcal{G}_j = \Omega_j(A)[\mathcal{K}_j(A^T, P)]^{\perp}.$$

Sea $B = \Omega_j(A)$ y $F = \mathcal{K}_j(A^T, P)^{\perp}$, entonces

$$\begin{aligned}
\mathcal{G}_j^{\perp} = (BF)^{\perp} &= \left\{w \mid w^T B v = 0, v \in F\right\} \\
&= \left\{w \mid (B^T w)^T v = 0, v \in F\right\} \\
&= \left\{B^{-T} y \mid y^T v = 0, v \in F\right\} \\
&= B^{-T}\left\{y \mid y^T v = 0, v \in F\right\} = B^{-T} F^{\perp},
\end{aligned}$$

que usando (3.14) acabamos, $\mathcal{G}_j^{\perp} = W_j$. □

En conclusión, la familia de métodos IDR(s), se tratan de métodos basados en subespacios de Krylov del tipo Petrov-Galerkin, estos métodos generalizan los basados en la idea del GC buscando recurrencias

cortas y una rápida convergencia hacia la solución deseada. Desde el punto de vista del almacenamiento es un método que requiere menos almacenamiento que el GMRES, que es el que más necesita de los estudiados en la memoria.

Por otra parte son métodos inestables que pueden producir interrupciones en su ejecución y por lo tanto hay que buscar otros métodos que puedan superar este obstáculo. No obstante, esto se produce con una baja probabilidad y hace que sean muy buena herramienta para la solución de sistemas de de gran dimensión y dispersos. En el siguiente capítulo vamos a estudiar algunas aplicaciones y a comparar los diversos métodos estudiados.

Capítulo 4

Aplicaciones

Ahora tenemos como objetivo estudiar el comportamiento de la familia de métodos IDR(s) frente al resto de métodos basados en subespacios de Krylov estudiados.

Para todas las aplicaciones se va a estudiar como varían el tiempo de CPU, el error y la velocidad de la convergencia para todos los métodos. Para los métodos del GMRES e IDR(s) se han usado los algoritmos que hemos implementado en MATLAB y que figuran en el apéndice de programas, mientras que para el BiGCSTAB se ha usado el comando de MATLAB, `bicgstab`.

4.1. Ecuación con coeficientes de convección, difusión y reacción.

La primera aplicación que vamos a considerar el siguiente problema de difusión-convección-reacción en el cubo unidad, $[0,1] \times [0,1] \times [0,1]$,

$$-\epsilon \Delta u + \vec{\beta} \nabla u - ru = F,$$

donde F esta definido de manera que la solución del problema sea la función $u(x,y,z) = x(1-x)y(1-y)z(1-z)$ y los parámetros están escogidos como $\epsilon = 0{,}02$ (constante de difusión), $\vec{\beta} = [0, \frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}}]^T$ (convección) y $r = 6$ (reacción). El ejemplo aparece en [11].

Hemos discretizado el dominio con una malla de diámetro $h = 0{,}1$ y se han utilizado las diferencias centradas para aproximar las diferentes derivadas que aparecen en la ecuación. Esto nos da un sistema

lineal $Ax = b$ donde la matriz A es cuadrada de tamaño 729 y con una estructura tal y como muestra la figura 4.1, formada por siete diagonales: las tres diagonales principales y dos diagonales adicionales en la parte triangular superior y parte triangular inferior. El número de elementos no nulos, nz, es de 4617.

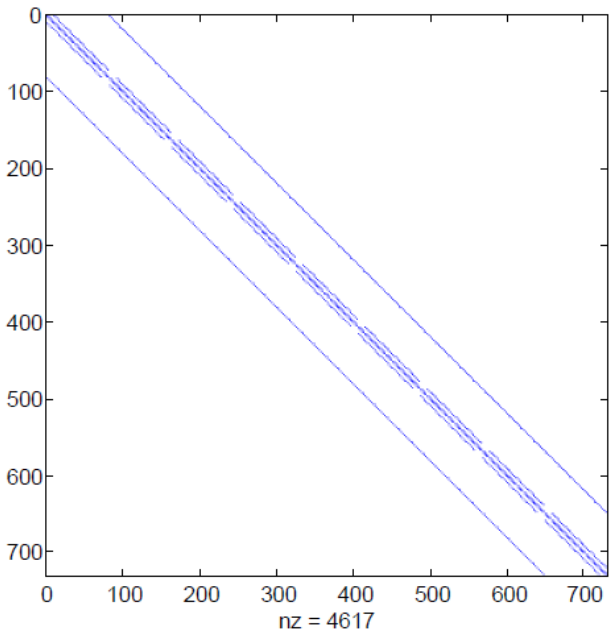

Figura 4.1: Estructura de la matriz del problema de convección-difusión-reacción.

Dicha matriz la hemos almacenado de manera dispersa con vistas a realizar las multiplicaciones matriz por vector de una manera más eficiente. Vamos a comparar los resultados que hemos obtenido al resolver este sistema mediante los métodos IDR(s), para varias elecciones de s, GMRES y BiGCSTAB.

En la figura 4.2 hemos representado el número de productos matriz por vector frente al error relativo en escala semilogarítmica en el eje y. Este tipo de representación nos permite estudiar la rapidez de la convergencia para cada uno de los métodos. En nuestro caso hemos tomado como tolerancia 10^{-7} y como número máximo de iteraciones el valor por defecto del programa, $min(2N, 1000)$, donde N es la dimensión de la matriz a resolver.

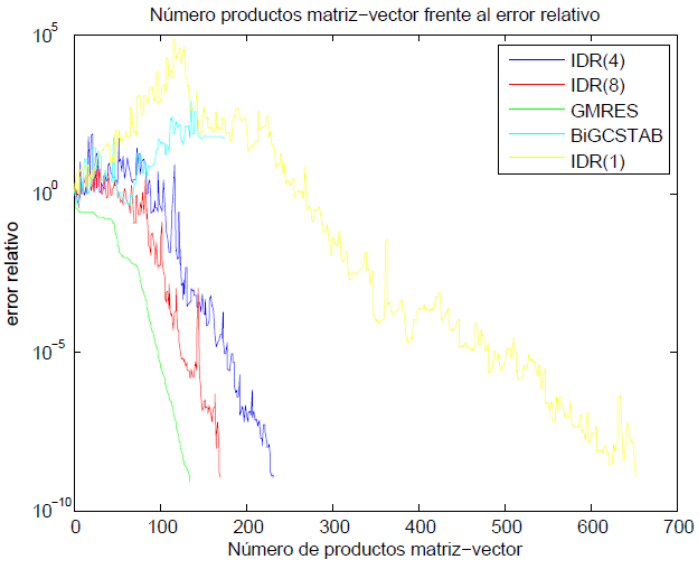

Figura 4.2: Número de productos matriz por vector frente a error relativo.

El corolario 3.3 nos dice que cuando trabajamos con la matriz de este ejemplo en aritmética exacta los métodos IDR(s) acaban tras 1458, 1094, 912, 821 multiplicaciones matriz por vector para $s = 1, 2, 4, 8$ respectivamente. En la figura se ve que para obtener soluciones cuyo error relativo es de orden -7 han hecho falta 650, 310, 215, 170 multiplicaciones matriz por vector respectivamente, valores que sugieren que en la práctica nos movemos lejos de la cota que hemos dado en teoría. El BiGCSTAB para en un número de pasos mucho más pequeño debido a que en el proceso uno de los parámetros del método se ha hecho muy pequeño y no ha podido continuar con el algoritmo. En este sentido vemos que con el IDR(1) si llegamos a la precisión prefijada.

Por otro lado podemos ver que las curvas correspondientes a los métodos IDR(s) están entre el BiGCSTAB y el GMRES que es el método más rápido y al que nos aproximamos para valores grandes de s, aunque con una convergencia más irregular.

Para los siguientes experimentos hemos modificado el problema. Hemos considerado mallas de diámetros $h = \frac{1}{k+1}, \quad k = 1, \ldots, 12$ y aplicado los métodos GRMES, BiGCSTAB y IDR(s), $s = 1, 4, 8, 15$ a las diferentes matrices de los sistemas lineales resultantes.

En la figura 4.3 podemos ver que el tiempo de CPU es comparable

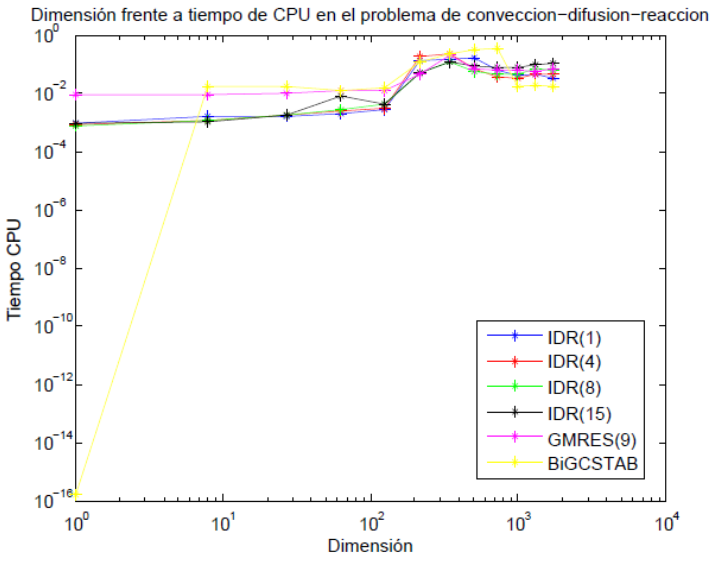

Figura 4.3: Dimensión frente al tiempo de CPU.

para todos los métodos cuando la dimensión crece, sin embargo, en la figura 4.4 podemos ver que si dejamos fija la dimensión, 729 en éste caso, el tiempo de CPU crece con s. Esto es debido a que al escoger el s el método debe construir $s+1$ vectores en el espacio $\mathcal{G}_j$ antes de pasar a $\mathcal{G}_j$ para obtener la siguiente aproximación. Si el s es grande la computación de dichos vectores eleva el costo computacional aunque aceleramos la convergencia ya que la reducción de la dimensión al cambiar de subespacio es de s.

En resumen, los resultados de los métodos IDR(s) se aproximan al GMRES para valores grandes de s mientras que IDR(1) nos da resultados similares al BiGCSTAB, como ya sabíamos pues en los programas hemos considerado $p_1 = r_0$ en la matriz P. De esto deducimos que la familia de métodos IDR(s) mejoran el BiGCSTAB y se aproximan a los resultados del GMRES, el método más rápido de los estudiados en la memoria. Sin embargo estos métodos no requieren la memoria exigida en el GMRES, que debe almacenar todos los residuos computados.

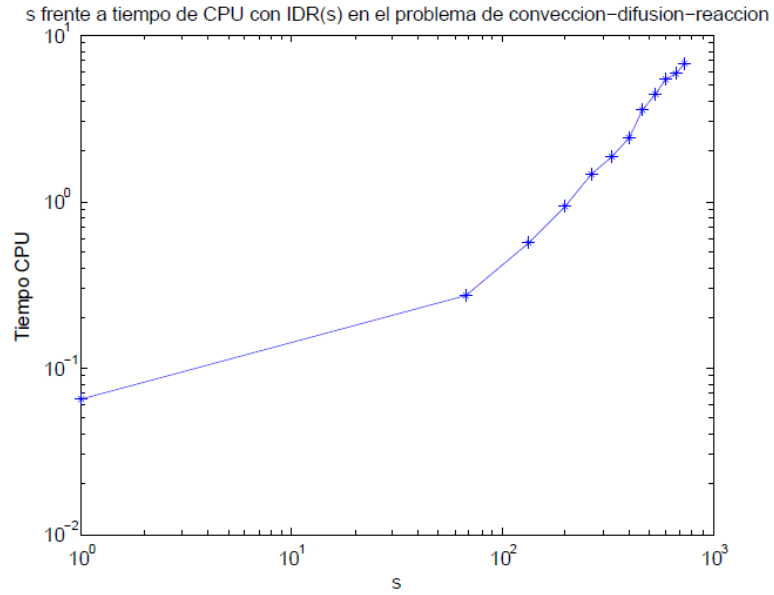

Figura 4.4: s frente al tiempo de CPU.

4.2. Sistema por bloques.

En esta sección vamos a estudiar el rendimiento de los métodos IDR(s) en comparación con los métodos del GMRES y BiGCSTAB en el caso de que la matriz del sistema esté definida por bloques de la siguiente manera:

$$M = \left[\begin{array}{c|c} A & B^T \\ \hline B & 0 \end{array}\right], \tag{4.1}$$

estructura típica de las matrices de los sistemas lineales resultantes resolver los problemas de Stokes tanto con diferencias finitas como con elementos finitos, entre otros.

Como ejemplo vamos a considerar el problema

$$\begin{aligned} -\nu\Delta u + \nabla p &= f, \text{ en } \Omega \\ \nabla u &= 0, \text{ en } \Omega \\ u &= 0, \text{ en } \partial\Omega \\ \int_\Omega p(x)dx &= 0, \end{aligned}$$

donde $\Omega = [0,1] \times [0,1]$, ν es la constante de viscosidad de un fluido, u representa la velocidad y p la presión del mismo. Si discretizamos

el dominio mediante diferencias finitas centradas de ordenes adecuados obtenemos un sistema lineal cuya matriz es de la forma (4.1). Las submatrices A y B vienen dadas por

$$A = \begin{bmatrix} I \otimes T + T \otimes I & 0 \\ 0 & I \otimes T + T \otimes I \end{bmatrix} \in \mathbb{R}^{2p^2 \times 2p^2}$$

y

$$B = \begin{bmatrix} I \otimes F \\ F \otimes I \end{bmatrix} \in \mathbb{R}^{2p^2 \times p^2},$$

donde

$$T = \frac{1}{h^2} \cdot \begin{bmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{bmatrix} \in \mathbb{R}^{p \times p},$$

$$F = \frac{1}{h} \cdot \begin{bmatrix} -1 & 1 & 0 & \cdots & 0 \\ 0 & -1 & 1 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 1 \\ 0 & \cdots & 0 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{p \times p},$$

y $h = \frac{1}{p+1}$. Para el lado derecho de la ecuación se ha tomado $f = 1$, de manera que $b = [1, \ldots, 1] \in \mathbb{R}^{3p^2}$.

Se ha resuelto el sistema con $p = 9$, lo que se traduce en un sistema con 243 ecuaciones con la estructura mostrada en la figura 4.5.

La matriz de dicho sistema ha sido almacenada de manera dispersa del mismo modo que la matriz de la sección anterior, aunque a diferencia de dicha matriz, la estructura de ésta no es diagonal. Se ha resuelto el problema con todos los métodos que queremos comparar, usando una tolerancia de $10e-8$ y un máximo de 1000 iteraciones en MATLAB. Tal y como hicimos con la ecuación anterior hemos representado en la figura 4.6 el número de productos matriz por vector frente al error relativo en cada iteración. Los resultados obtenidos claramente son similares, aunque para este sistema todos los métodos convergen de manera más rápida y el BiGCSTAB no para el algoritmo.

Como conclusión, la familia de métodos IDR(s) para resolver problemas de Stokes no presenta ninguna ventaja destacada que sea debida a la estructura por bloques de la matriz del sistema.

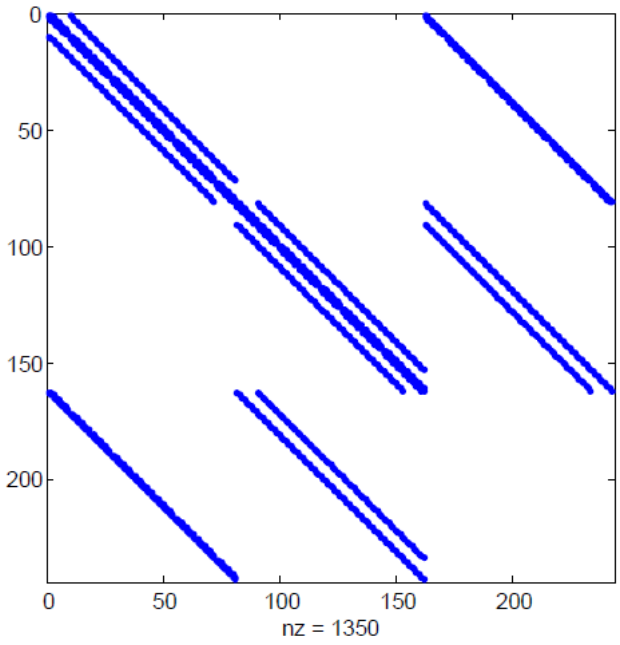

Figura 4.5: Estructura de la matriz por bloques del problema de Stokes.

4.3. Conclusiones.

A la vista de los dos problemas anteriores podemos sacar diversas conclusiones acerca del rendimiento de los métodos IDR(s).

La familia de métodos IDR(s) forman un abanico de métodos cuya velocidad de convergencia varía entre el BiGCSTAB (recurrencias cortas) y el GMRES (minimización del residuo), mejorando los resultados del primero y acercándose a los del segundo conforme avanzamos el s. Ya sabemos que el único método que minimiza el residuo con recurrencias cortas es el GC, la familia de métodos IDR(s) obtiene resultados que se acercan a ambos conceptos. En consecuencia, forman una herramienta alternativa al método del GMRES en el caso en que no queramos utilizar un alto almacenaje y al BiGCSTAB en el caso en el que queramos más velocidad en la convergencia sin sacrificar demasiada memoria. Además puede ocurrir que el BiGCTAB pare debido a problemas con alguno de sus parámetros mientras que el IDR(1) llega hasta el final.

El s que escogemos para obtener resultados similares al GMRES no

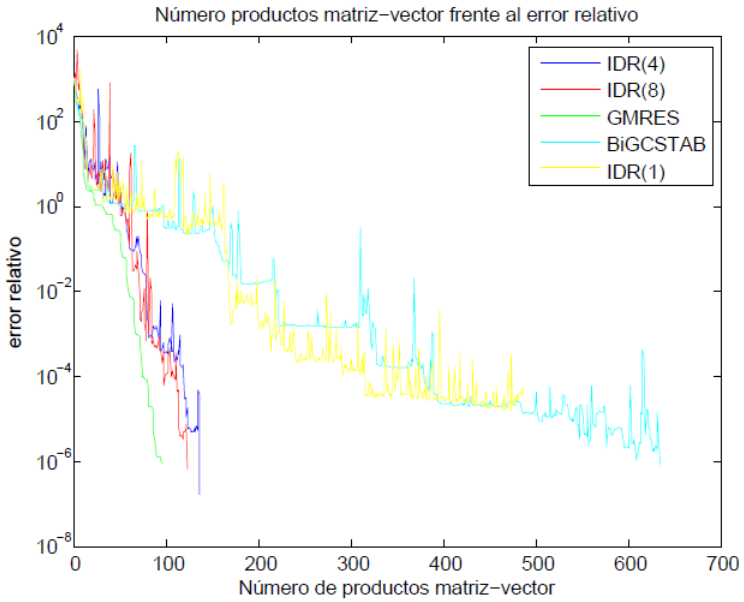

Figura 4.6: Número de productos matriz-vector frente a error relativo.

es excesivamente grande, hemos podido comprobar para los ejemplos anteriores que con $s = 8$ los resultados han sido satisfactorios. Es aquí donde se hace evidente la necesidad de explorar una implementación satisfactoria para valores grandes de s dando de nuevo sentido a la sección 2 del capítulo anterior. En los ejemplos considerados, el IDR(s) genérico también nos da resultados satisfactorios en este sentido para $s = 8$ pero cuanto mayor sea la dimensión de la matriz, mayor es el s que necesitamos.

El GMRES(m), como sabemos, es una variación del GMRES original que busca aliviar el número de vectores almacenados con éxito. Sin embargo, con el objetivo de llevar a cabo una comparación justa entre todos los métodos hemos usado en nuestros experimentos el GMRES original ya que al igual que construimos el GMRES(m) se puede hacer exactamente lo mismo para los IDR(s) dando como resultado una familia biparamétrica de métodos iterativos, IDR(s,m), que compite con el GMRES(m) en igualdad de condiciones. Podemos adelantar que la comparación entre estos dos últimos métodos es la misma que la estudiada entre los IDR(s) y el GMRES. Esto es debido a que el coste en memoria no afecta a los resultados obtenidos.

En conclusión, los métodos IDR(s) son una alternativa al GMRES

a tener en cuenta ya que no necesitan tanto almacenamiento. Sin embargo, la convergencia no es tan suave como lo es con el GMRES y en algunos casos pudiera ser que fallara, aunque como sabemos esto ocurre de manera aislada.

Por último, los métodos IDR(s) pueden ser mejorados en muchos sentidos dando lugar a una familia de métodos cada vez más pulidos basados en el Teorema IDR.

Bibliografía

[1] Saad Y., *Iterative methods for sparse linear systems*, (2nd ed., 2000).

[2] Corredor I.; Trabajo de fin de grado, Facultad de ciencias. Universidad de Valladolid. *La exponencial matricial: cálculo eficiente y aplicaciones*, (2016).

[3] Henk A. van der vorst , *Iterative Krylov methods for large linear systems*, Cambridge, (2003).

[4] Wesseling P., P.Sonneveld P. *Numerical experiments with a multiple grid and a conditioned Lanczos type method*, Springer-Verlag, (543-562) , (1980).

[5] Greenbaum A. *Iterative methods for solving linear systems*, Siam Frontiers, (1997)

[6] Sonneveld P., Van Gijzen M.B. *IDR(s): A Family of Simple and Fast Algorithms For Linear Equations.*, SIAM Journal, Vol. 31, No 2, pp 1035-1062, (2008)

[7] Simoncini V., Szyld D.B. *Interpreting IDR as a Petrov-Galerkin Method*, SIAM journal, vol. 32, No 4, pp 1898.1912, (2010)

[8] Sonneveld P., van Gijzen M.B. 2011. *Algorithm 913: An elegant IDR(s) variant that efficiently exploits biorthogonality properties*. ACM Trans. Math. Softw. 38, 1, Article 5 (November 2011), 19 pages.

[9] Guo-Feng Zhang, Qun-hua Lu, *On generalized symmetric SOR method for augmented systems*. School of mathematics and Statistics, Lanzhou 730000, PR China (February 2007).

[10] Saad Y., *Analysis of some Krylov subspace approximations to the Matrix Exponential Operator*, SIAM J. Numer. Anal. vol. 29, No 1, (1992), 209–228.

[11] van Gijzen M. B., *The Induced Dimension Reduction Method*, http://ta.twi.tudelft.nl/nw/users/gijzen/IDR.html

Apéndice A

Programas de MATLAB

En este apéndice recopilamos los programas que hemos realizado en MATLAB para los métodos IDR(s) y GMRES y sus respectivas modificaciones. Omitimos los algoritmos del resto de métodos estudiados en la memoria ya que pueden encontrarse implementados como comandos de MATLAB. En concreto, `cgs`, `bicg` y `bigcstab` para el GC, BiGC y BiGCSTAB respectivamente.

A.1. idrs.m

En este programa hemos implementado la aproximación de la solución del sistema $Ax = b$ mediante el método IDR(s) implementado con el algoritmo genérico.

```
function [x,iter,temp,resvec]=idrs(A,b,s,tol,maxit,x0)

%Esta función implementa el método IDR(s) para
resolver sistemas lineales del tipo Ax = b.
%Los parámetros de entrada del programa son
los siguientes:

%Parámetros de entrada:
%-- A matriz cuadrada, no necesariamente
definida positiva ni simétrica.
%-- b término independiente del sistema lineal.
%-- s dimensión del espacio S que interviene
en el teorema IDR. Si no se especifica
el valor de s se tomara como defecto s = 4.
%-- tol tolerancia que vamos a usar para el criterio de
```

```
parada del programa. Si no se especifica
el valor de tol se tomara como defecto tol=1e-8.
%-- maxit número máximo de iteraciones que
dejaremos correr el programa.
Si no se especifica el valor de maxit
se tomara como defecto maxit=min(2N,1000),
donde N es la dimensión del problema.
%-- x0 aproximación inicial, si no se
tiene ninguna se puede tomar nula.
Si no se especifica la aproximación
inicial se toma la identicamente nula.

%Parámetros de salida:
%-- x Solución.
%-- iter Número de iteraciones que da el programa.
%-- temp tiempo de CPU que tarda el programa
en alcanzar la solución.
%-- resvec vector de residuos.

%En primer lugar necesitamos una
aproximación inicial, la vamos a tomar
identicamente nula:

%Vamos a medir el tiempo de CPU
que tarda el programa en calcular la solución:
tic;

N=length(b);

if ( nargout == 0 )
    help idrs;
    return
end

%Comprobamos que hay argumentos suficientes:
if nargin <2
    error('Not enough input arguments.');
end

%Comprobando las dimensiones correctas de la
matriz A y el vector b:
[m,N] = size(A);
if (m  = N)
    error('Matrix must be square.');
end
```

```
if  isequal(size(b),[m,1])
    es = sprintf(['Right hand side must be a column vector
of' ...
    ' length %d to match the coefficient matrix.'],m);
    error(es);
end

%Asignamos valores por defecto:
if nargin <3 || isempty(s)
    s = 4;
end
if ( s >N )
    s = N;
end
if nargin <4 || isempty(tol)
    tol = 1e-8;
end
if nargin <5 || isempty(maxit)
    maxit = min(2*N,1000);
end

if nargin <6 || isempty(x0)
    x0 = zeros(N,1);
else
    if  isequal(size(x0),[N,1])
    es = sprintf(['Initial guess must be a column vector
of' ...
    ' length %d to match the problem size.'],n);
    error(es);
end

x=x0; %aprox. inic.
r=b-A*x; %residuo inicial.
normr=norm(r);
resvec=normr;

%vamos a cubrir la posibilidad de que la condición inicial
ya este lo suficientemente cerca.
if(normr<=tol)
    iter=0;
    return;
end

%En segundo lugar necesitamos general la matriz
```

```
P del subespacio S:
randn('state',0);
P=rand(N,s);
P(:,1)=r;
P=orth(P);
%La elección de P es una colección de vectores
aleatorios donde el primero es el primer
residuo(para comparar con el BiGCSTAB) y
posteriormente ortogonalizados.

%Por último antes de comenzar con el bucle
inicial del programa debemos generar los
vectores suficientes en el espacio inicial:
deltaR=zeros(N,s);
deltaX=zeros(N,s);
PtdeltaR=zeros(s,s);
%Los vecores δ se van a ir actualizando
durante todo el programa para ganar memoria.
for k=1:s
    %Cálculo del ω:
    v=A*r;
    aux1=dot(v,r);
    aux2=dot(r,r);
    omega=aux1/aux2;
    %Actualizamos los vectores δ:
    deltaX(:,k)=omega*r;
    deltaR(:,k)=-omega*v;
    %Actualizamos la solución siguiendo
el método del mínimo residuo:
    x=x+deltaX(:,k);
    r=r+deltaR(:,k);
    normr=norm(r);
    %Por último como vamos a tener que resolver
un sistema lineal, vamos construyendo una
a una las columnas:
    resvec=[resvec,normr];
    PtdeltaR(:,k)=P'*deltaR(:,k);
end
%Ahora estamos listos para comenzar el bucle:
iter=s;
cont=1;%Contador para saber cuando
```

```
pasamos de espacio G.
space=1;
Ptr=P'*r; %Término independiente del sistema
while normr>tol && iter<maxit
    for k=0:s
      %Resolvemos el sistema, esto solo lo vamos a
hacer una vez por espacio G.
      c=PtdeltaR\ Ptr;
      q=-deltaR*c;
      v=r+q;
      if k==0  %Solo vamos a calcular el ω
una vez por espacio G.
        t=A*v;
        aux1=dot(t,v);
        aux2=dot(t,t);
        omega=aux1/aux2;
        deltaR(:,cont)=q-omega*t;
        deltaX(:,cont)=-deltaX*c+omega*v;
      else
        %Matrices que nos dan los vectores intermedios,
nos aprovechamos de la identidad AdeltaX = -deltaR:
        deltaX(:,cont)=-deltaX*c+omega*v;
        deltaR(:,cont)=-A*deltaX(:,cont);
      end
      %Actualizamos el residuo de la iteración:
      r=r+deltaR(:,cont);
      x=x+deltaX(:,cont);
      iter=iter+1;

      normr=norm(r); %Actualizamos el residuo.
      resvec=[resvec,normr];

      %Actualizamos el sistema para la siguiente
iteración:
      PdRnew=P'*deltaR(:,cont);
      PtdeltaR(:,cont)=PdRnew;
      Ptr = Ptr + PdRnew;

      cont=cont+1;
      %Por último tenemos que asegurarnos de
cambiar de subespacio si tenemos s iteraciones
en el mismo subespacio:
```

```
      if cont>s
        cont=1;
        space=space+1;
      end
    end
end

temp=toc;
%Cuando salgamos del bucle obtendremos ya el vector
x y el número de iteraciones.
end
```

A.2. idrmej.m

En este programa hemos implementado la aproximación de la solución del sistema $Ax = b$ mediante el método IDR(s) implementado tal y como se describe en la sección 2 del capítulo 3.

```
function [x,iter,temp,resvec]=idrmej(A,b,s,tol,maxit,x0)


%Esta función implementa el método IDR(s) para
resolver sistemas lineales del tipo Ax = b.
%Los parámetros de entrada del programa son
los siguientes:

%Parámetros de entrada:
%-- A matriz cuadrada, no necesariamente
definida positiva ni simétrica.
%-- b término independiente del sistema lineal.
%-- s dimensión del espacio S que interviene
en el teorema IDR. Si no se especifica
el valor de s se tomara como defecto s = 4.
%-- tol tolerancia que vamos a usar para el criterio de
parada del programa. Si no se especifica
el valor de tol se tomara como defecto tol=1e - 8.
%-- maxit número máximo de iteraciones que
dejaremos correr el programa.
Si no se especifica el valor de maxit
se tomara como defecto maxit = min(2N, 1000),
```

```
donde N es la dimensión del problema.
%-- x0 aproximación inicial, si no se
tiene ninguna se puede tomar nula.
Si no se especifica la aproximación
inicial se toma la identicamente nula.

%Parámetros de salida:
%-- x Solución.
%-- iter Número de iteraciones que da el programa.
%-- temp tiempo de CPU que tarda el programa
en alcanzar la solución.
%-- resvec vector de residuos.

%En primer lugar necesitamos una
aproximación inicial, la vamos a tomar
identicamente nula:

%Vamos a medir el tiempo de CPU
que tarda el programa en calcular la solución:
tic;

N=length(b);

if ( nargout == 0 )
    help idrmej;
    return
end

%Comprobamos que hay argumentos suficientes:
if nargin <2
    error('Not enough input arguments.');
end

%Comprobando las dimensiones correctas
de la matriz A y el vector b:
[m,N] = size(A);
if (m  = N)
    error('Matrix must be square.');
end
if  isequal(size(b),[m,1])
    es = sprintf(['Right hand side must be a column vector
of' ...
    ' length %d to match the coefficient matrix.'],m);
    error(es);
end
```

```
%Asignamos valores por defecto:
if nargin <3 || isempty(s)
    s = 4;
end
if ( s >N )
    s = N;
end
if nargin <4 || isempty(tol)
    tol = 1e-8;
end
if nargin <5 || isempty(maxit)
    maxit = min(2*N,1000);
end

if nargin <6 || isempty(x0)
    x0 = zeros(N,1);
else
    if  isequal(size(x0),[N,1])
    es = sprintf(['Initial guess must be a column vector
of' ...
    ' length %d to match the problem size.'],n);
    error(es);
end

%Fin de las comprobaciones,
%En segundo lugar necesitamos general la matriz P
del subespacio S y parámetro κ:
kappa=0.7;
randn('state',0);
P=rand(N,s);
P(:,1)=r;
P=orth(P);
%La elección de P es una colección de
vectores aleatorios donde el primero es
el primer residuo (para comparar con el BiGCSTAB)
y posteriormente ortogonalizados.

%Comprobación de la solución nula:
%El vector nulo es la solución.
if (norm(b) == 0)
    x = zeros(N,1);
    iter = 0;
```

```
    return
end
x=x0; %aprox. inic.
normb=norm(b);
%Tolerancia relativa
tolb=tol*normb;
r=b-A*x; %Residuo inicial.
normr=norm(r);
resvec=normr;

%Vamos a cubrir la posibilidad de que
la condición inicial ya este lo suficientemente
cerca.
if(normr<=tol)
    iter=0;
    return;
end

%Por último antes de comenzar con el
bucle inicial del programa debemos generar
los vectores suficientes en el espacio inicial:
G = zeros(N,s);
U = zeros(N,s);
M = eye(s,s);
om = 1;
%Bucle principal:
iter=0;
while normr>tolb && iter<maxit

    %Término independiente del sistema a resolver:
    f = (r'*P)';
    for k = 1:s

      %Solución del sistema:
      c = M(k:s,k:s) \ f(k:s);
      v = r - G(:,k:s)*c;

      %Generación de nuevos vectores U(:,k) y G(:,k),
G(:,k) esta en el espacio G_j
      U(:,k) = U(:,k:s)*c + om*v;
      G(:,k) = A*U(:,k);

      %Ortogonalizar la nueva base:
      for i = 1:k-1
```

```
        alpha = ( P(:,i)'*G(:,k) )/M(i,i);
        G(:,k) = G(:,k) - alpha*G(:,i);
        U(:,k) = U(:,k) - alpha*U(:,i);
      end

      %Nueva columna de M = P'G (las primeras k-1
entradas son cero)
      M(k:s,k) = (G(:,k)'*P(:,k:s))';
      if ( M(k,k) == 0 )
        return;
      end

      %Hacemos r ortogonal a q_i, i = 1,...,k
      beta = f(k)/M(k,k);
      r = r - beta*G(:,k);
      x = x + beta*U(:,k);
      normr = norm(r);
      resvec = [resvec;normr];
      iter = iter + 1;
      if ( normr <tolb || iter == maxit )
        break
      end

      %Nueva f = P'*r (las primeras k entradas son cero)
      if ( k <s )
        f(k+1:s) = f(k+1:s) - beta*M(k+1:s,k);
      end
    end

    if ( normr <tolb || iter == maxit )
      break
    end

    %Ahora tenemos suficientes vectores en G_j
para computar el siguiente residuo en G_{j+1},
Nota: r es perpendicular a P, por lo tanto, v = r.
    v = r;
    t = A*v;

    %Nuevo omega:
    om = omega( t, r, kappa );
    if ( om == 0 )
      return;
    end
```

```
    r = r - om*t;
    x = x + om*v;
    normr = norm(r);
    resvec = [resvec;normr];
    iter = iter + 1;
end
temp=toc;
%Cuando salgamos del bucle obtendremos
ya el vector x y el número de iteraciones.
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function om = omega( t, s, angle )
ns = norm(s);
nt = norm(t);
ts = t'*s;
rho = abs(ts/(nt*ns));
om=ts/(nt*nt);
if ( rho <angle )
    om = om*angle/rho;
end
return
```

A.3. gmresm.m

En este programa hemos implementado la aproximación de la solución del sistema $Ax = b$ mediante el método GMRES(m).

```
function [x,iter,temp,error]=gmresm(A,b,m,tol,maxit,x0)

%Esta función implementa el método GMRES(m) para
resolver sistemas lineales del tipo Ax = b.
%Los parámetros de entrada del programa son
los siguientes:
%Parámetros de entrada:
%-- A matriz cuadrada, no necesariamente
definida positiva ni simétrica.
%-- b término independiente del sistema lineal.
```

```
%-- m parámetro para reiniciar las variables.
%-- s dimensión del espacio S que interviene
en el teorema IDR. Si no se especifica
el valor de s se tomara como defecto s = 4.
%-- tol tolerancia que vamos a usar para el criterio de
parada del programa. Si no se especifica
el valor de tol se tomara como defecto tol=1e − 8.
%-- maxit número máximo de iteraciones que
dejaremos correr el programa.
Si no se especifica el valor de maxit
se tomara como defecto maxit = min(2N, 1000),
donde N es la dimensión del problema.
%-- x0 aproximación inicial, si no se
tiene ninguna se puede tomar nula.
Si no se especifica la aproximación
inicial se toma la identicamente nula.

%Parámetros de salida:
%-- x Solución.
%-- iter Número de iteraciones que da el programa.
%-- temp tiempo de CPU que tarda el programa
en alcanzar la solución.
%-- resvec vector de residuos.

%En primer lugar necesitamos una
aproximación inicial, la vamos a tomar
identicamente nula:

%Vamos a medir el tiempo de CPU
que tarda el programa en calcular la solución:
tic;

N=length(b);

if ( nargout == 0 )
    help gmresm;
    return
end

%Comprobamos que hay argumentos suficientes:
if nargin <2
    error('Not enough input arguments.');
end

%Comprobando las dimensiones correctas de
```

```
la matriz A y el vector b:
[m,N] = size(A);
if (m  = N)
    error('Matrix must be square.');
end
if  isequal(size(b),[m,1])
    es = sprintf(['Right hand side must be a column vector
of' ...
    ' length %d to match the coefficient matrix.'],m);
    error(es);
end

%Asignamos valores por defecto:
if nargin <3 || isempty(s)
    s = 4;
end
if ( s >N )
    s = N;
end
if nargin <4 || isempty(tol)
    tol = 1e-8;
end
if nargin <5 || isempty(maxit)
    maxit = min(2*N,1000);
end

if nargin <6 || isempty(x0)
    x0 = zeros(N,1);
else
    if  isequal(size(x0),[N,1])
    es = sprintf(['Initial guess must be a column vector
of' ...
    ' length %d to match the problem size.'],n);
    error(es);
end

%Fin de las comprobaciones.
%Iniciamos:
%Comprobación de la solución nula:
%El vector nulo es la solución.
if (norm(b) == 0)
    x = zeros(N,1);
```

```
    iter = 0;
    return
end
x=x0; %aprox. inic.

h=zeros(maxit);
v=zeros(N,maxit);
c=zeros(maxit+1,1);
s=zeros(maxit+1,1);
if norm(x)  =0
    r = b-A*x;
else
    r = b;
end
rho=norm(r);
g=rho*eye(maxit+1,1);
tolb=tol*norm(b);
error=[];

error=[error,rho];
iter=0;
if(rho <tolb)
    return
end

iter=0;

%Bucle principal de GMRES:
while(rho >tolb & & iter <maxit)
    v(:,1)=r/rho;
    beta=rho;
    for k=1:m
      v(:,k+1)=A*v(:,k);
      normav=norm(v(:,k+1));
      %Ortogonalizamos, Gram-Schmidt modificado:
      for j=1:k
        h(j,k)=v(:,j)'*v(:,k+1);
        v(:,k+1)=v(:,k+1)-h(j,k)*v(:,j);
      end
      h(k+1,k)=norm(v(:,k+1));
      normav2=h(k+1,k);
      %Quiza debamos reortogonalizar:
      if (normav+.001*normav2 == normav)
```

```
        for j=1:k
          hr=v(:,j)'*v(:,k+1);
          h(j,k)=h(j,k)+hr;
          v(:,k+1)=v(:,k+1)-hr*v(:,j);
        end
        h(k+1,k)=norm(v(:,k+1));
      end
      if(h(k+1,k)  = 0)
        v(:,k+1)=v(:,k+1)/h(k+1,k);
      end
      %Rotaciones de Guivens:
      if k >1
        h(1:k,k)=givens(c(1:k-1),s(1:k-1),h(1:k,k),k-1);
      end
      nu=norm(h(k:k+1,k));
      if nu =0
        %c(k)=h(k,k)/nu;
        c(k)=conj(h(k,k)/nu);
        s(k)=-h(k+1,k)/nu;
        h(k,k)=c(k)*h(k,k)-s(k)*h(k+1,k);
        h(k+1,k)=0;
        g(k:k+1)=givens(c(k),s(k),g(k:k+1),1);
      end
      %Actualizamos el residuo:
      rho=abs(g(k+1));
      error=[error,rho];
    end
    iter=iter+m;
    y=h(1:k,1:k)\g(1:k);
    temp=toc;
    x = x0 + v(1:N,1:k)*y;
end
%Ya hemos acabado:
y=h(1:k,1:k)\g(1:k);
temp=toc;
iter=k;
x = x0 + v(1:N,1:k)*y;
return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function vrot=givens(c,s,vin,k)
%Función que realiza las rotaciones
de Guivens para el GMRES
vrot=vin;
for i=1:k
    w1=c(i)*vrot(i)-s(i)*vrot(i+1);
    w2=s(i)*vrot(i)+conj(c(i))*vrot(i+1);
    vrot(i:i+1)=[w1,w2];
end
return
```

A.4. gmres.m

```
function [x,iter,temp,resvec]=gmres(A,b,s,tol,maxit,x0)

%Esta función implementa el método GMRES(m) para
resolver sistemas lineales del tipo Ax = b.
%Los parámetros de entrada del programa son
los siguientes:

%Parámetros de entrada:
%-- A matriz cuadrada, no necesariamente
definida positiva ni simétrica.
%-- b término independiente del sistema lineal.
%-- m parámetro para reiniciar las variables.
%-- s dimensión del espacio S que interviene
en el teorema IDR. Si no se especifica
el valor de s se tomara como defecto s = 4.
%-- tol tolerancia que vamos a usar para el criterio de
parada del programa. Si no se especifica
el valor de tol se tomara como defecto tol=1e − 8.
%-- maxit número máximo de iteraciones que
dejaremos correr el programa.
Si no se especifica el valor de maxit
se tomara como defecto maxit = min(2N, 1000),
donde N es la dimensión del problema.
%-- x0 aproximación inicial, si no se
tiene ninguna se puede tomar nula.
```

```
Si no se especifica la aproximación
inicial se toma la identicamente nula.

%Parámetros de salida:
%-- x Solución.
%-- iter Número de iteraciones que da el programa.
%-- temp tiempo de CPU que tarda el programa
en alcanzar la solución.
%-- resvec vector de residuos.

%En primer lugar necesitamos una
aproximación inicial, la vamos a tomar
identicamente nula:

%Vamos a medir el tiempo de CPU
que tarda el programa en calcular la solución:
tic;

N=length(b);

if ( nargout == 0 )
    help gmres;
    return
end

%Comprobamos que hay argumentos suficientes:
if nargin <2
    error('Not enough input arguments.');
end

%Comprobando las dimensiones correctas de la
matriz A y el vector b:
[m,N] = size(A);
if (m  = N)
    error('Matrix must be square.');
end
if  isequal(size(b),[m,1])
    es = sprintf(['Right hand side must be a column vector
of' ...
    ' length %d to match the coefficient matrix.'],m);
    error(es);
end

%Asignamos valores por defecto:
if nargin <3 || isempty(s)
```

```
    s = 4;
end
if ( s >N )
    s = N;
end
if nargin <4 || isempty(tol)
    tol = 1e-8;
end
if nargin <5 || isempty(maxit)
    maxit = min(2*N,1000);
end

if nargin <6 || isempty(x0)
    x0 = zeros(N,1);
else
    if  isequal(size(x0),[N,1])
    es = sprintf(['Initial guess must be a column vector
of' ...
    ' length %d to match the problem size.'],n);
    error(es);
end

%Fin de las comprobaciones.
%Iniciamos:
%Comprobación de la solución nula:
%El vector nulo es la solución.
if (norm(b) == 0)
    x = zeros(N,1);
    iter = 0;
    return
end
x=x0; %aprox. inic.

h=zeros(maxit);
v=zeros(N,maxit);
c=zeros(maxit+1,1);
s=zeros(maxit+1,1);
if norm(x)  =0
    r = b-A*x;
else
    r = b;
end
```

```
rho=norm(r);
g=rho*eye(maxit+1,1);
tolb=tol*norm(b);
error=[];

error=[error,rho];
iter=0;
if(rho <tolb)
    return
end

v(:,1)=r/rho;
beta=rho;
k=0;

%Bucle principal de GMRES:
while(rho >tolb & & k <maxit)
    k=k+1;
    v(:,k+1)=A*v(:,k);
    normav=norm(v(:,k+1));
    %ortogonalizamos, Gram-Schmidt modificado:
    for j=1:k
      h(j,k)=v(:,j)'*v(:,k+1);
      v(:,k+1)=v(:,k+1)-h(j,k)*v(:,j);
    end
    h(k+1,k)=norm(v(:,k+1));
    normav2=h(k+1,k);
    %Quiza debamos reortogonalizar:
    if (normav+.001*normav2 == normav)
      for j=1:k
        hr=v(:,j)'*v(:,k+1);
        h(j,k)=h(j,k)+hr;
        v(:,k+1)=v(:,k+1)-hr*v(:,j);
      end
      h(k+1,k)=norm(v(:,k+1));
    end
    if(h(k+1,k)  = 0)
      v(:,k+1)=v(:,k+1)/h(k+1,k);
    end

    %Rotaciones de Guivens:
    if k >1
      h(1:k,k)=givens(c(1:k-1),s(1:k-1),h(1:k,k),k-1);
```

```
    end
    nu=norm(h(k:k+1,k));
    if nu =0
      %c(k)=h(k,k)/nu;
      c(k)=conj(h(k,k)/nu);
      s(k)=-h(k+1,k)/nu;
      h(k,k)=c(k)*h(k,k)-s(k)*h(k+1,k);
      h(k+1,k)=0;
      g(k:k+1)=givens(c(k),s(k),g(k:k+1),1);
    end
    %Actualizamos el residuo:
    rho=abs(g(k+1));
    error=[error,rho];
end

%Ya hemos acabado:
y=h(1:k,1:k)\g(1:k);
temp=toc;
iter=k;
x = x0 + v(1:N,1:k)*y;
return

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function vrot=givens(c,s,vin,k)
%Función que realiza las rotaciones
de Guivens apra el GMRES
vrot=vin;
for i=1:k
    w1=c(i)*vrot(i)-s(i)*vrot(i+1);
    w2=s(i)*vrot(i)+conj(c(i))*vrot(i+1);
    vrot(i:i+1)=[w1,w2];
end
return
```