Notas sobre el desarrollo del método numérico de Newton-Krylov y su relación con FreeGSNKE

Andrés Camacho López[1]

[1]Instituto de Ciencias Nucleares, Universidad Nacional Autónoma de México
[1]andres.camacho@correo.nucleares.unam.mx

26 de Octubre de 2025

Resumen

La presentes notas pretenden dar un panorama más amplio a como funciona el método de Newton-Krylov, y tener un mejor entendimiento de como este se integra en el solucionador de equilibrio de flujo magnético FreeGNSKE. Además de dar un panorama de las dificultades que puede traer esta implementación y como mejorar los resultados para futuros tokamaks.

1 Método Newton-Krylov

Una forma de resolver sistemas de ecuaciones no lineales de gran dimensión –particularmente aquellos que surgen de discretizaciones de ecuaciones diferenciales– es mediante un solucionador numérico que combina dos métodos: el **método de Newton-Raphson**, que reduce el sistema no lineal a una sucesión de problemas lineales usando la matriz jacobiana, y el **método del residual mínimo generalizado (GMRES)**, que utiliza el **subespacio de Krylov** construyendo una base mediante el proceso de Arnoldi para resolver la parte lineal.

Este enfoque, conocido como **método Newton-Krylov**, se emplea frecuentemente en problemas computacionales de alta complejidad, ya que resulta más eficiente y escalable que los métodos tradicionales para este tipo de sistemas.

En lo que sigue, se desarrollará una explicación detallada del método. Se comenzará por la parte lineal del problema, dado que su comprensión es fundamental antes de aplicarla en el contexto de Newton-Raphson, y constituye el núcleo de la solución de sistemas no lineales, como se mostrará más adelante.

PARTE I - Problema Lineal: GMRES, Subespacio de Krylov y El Proceso de Arnoldi

Para ilustrar los conceptos de esta sección, se considera un sistema lineal de ecuaciones:

$$Ax = b, \tag{1}$$

donde:

- $A \in \mathbb{R}^{n\times n}$ es una matriz grande y dispersa,
- $b \in \mathbb{R}^n$ es el vector de términos independientes,
- $x \in \mathbb{R}^n$ es el vector solución del sistema.

Nótese que una matriz dispersa es aquella de gran dimensión en la que la mayoría de sus elementos son cero.

Desde el punto de vista computacional, este sistema podría resolverse mediante métodos directos como la eliminación gaussiana o la factorización LU; sin embargo, estos métodos presentan limitaciones en términos de eficiencia para matrices de gran tamaño:

- **Eliminación gaussiana:** Requiere un número excesivo de operaciones para escalonar y reducir la matriz.
- **Factorización LU:** La descomposición $A = L \cdot U$ depende críticamente de los pasos anteriores y también

demanda un costo computacional elevado para calcular la solución.

Por estas razones, resulta preferible emplear métodos iterativos más eficientes, como el **GMRES** (Generalized Minimal Residual). No obstante, antes de describir este método en detalle, es necesario introducir el concepto de **subespacio de Krylov**.

1.1 Subespacio de Krylov

Definición 1.1 *Sea $A \in \mathbb{R}^{n\times n}$ una matriz cuadrada y $r_0 \in \mathbb{R}^n$ un vector. El subespacio de Krylov de orden m se define como:*

$$K_m(A, r_0) = \text{Span}\{r_0, Ar_0, A^2r_0, \ldots, A^{m-1}r_0\}, \quad (2)$$

es decir, el conjunto de todas las combinaciones lineales de los vectores $\{r_0, Ar_0, \ldots, A^{m-1}r_0\}$:

$$\left\{ \sum_{i=0}^{m-1} c_i A^i r_0 \;\middle|\; c_0, c_1, \ldots, c_{m-1} \in \mathbb{R} \right\}. \quad (3)$$

1.1.1 Dimensión del Subespacio de Krylov

En el contexto del subespacio de Krylov, resulta fundamental el concepto de polinomio mínimo:

Definición 1.2 (Polinomio mínimo) *El polinomio mínimo de un vector r_0 respecto a una matriz A es el polinomio mónico (único) $m(x)$ de menor grado, distinto de cero, tal que:*

$$m(A)r_0 = 0, \quad (4)$$

es decir, hasta que se produzca una dependencia lineal en la secuencia de vectores.

El grado d del polinomio mínimo representa el número máximo de vectores linealmente independientes (l.i.) que pueden generarse con $\{r_0, Ar_0, A^2r_0, \ldots\}$.

Definición 1.3 (Punto de saturación) *El punto de saturación ocurre cuando los vectores generados por $\{r_0, Ar_0, A^2r_0, \ldots\}$ ya no aportan nuevas direcciones al subespacio de Krylov.*

Para el subespacio de Krylov $K_m(A, r_0)$ con $\dim(K_m) = m$, existe un polinomio $m(x)$ de grado d tal que:

$$m(A)r_0 = a_0r_0 + a_1Ar_0 + a_2A^2r_0 + \cdots + a_dA^dr_0 = 0, \quad (5)$$

y por lo tanto A^dr_0 es una combinación lineal de los vectores anteriores.

De lo anterior se deduce que:

- Cuando $m < d$, entonces $\dim(K_m) = m$,
- Cuando $m \geq d$, entonces $\dim(K_m) = d$,

de donde se concluye que:

$$\dim(K_m(A, r_0)) = \text{mín}(m, d). \quad (6)$$

Para ilustrar estos conceptos, considérense los siguientes ejemplos.

Ejemplo 1:

Sea $A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$ y $r_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.
Se obtiene:

- $r_0 = (1, 1)$
- $Ar_0 = (3, 3) \Rightarrow Ar_0 = 3r_0$
- $A^2r_0 = (9, 9) \Rightarrow A^2r_0 = 9r_0 = 3^2r_0$

Considérese el polinomio $A^2r_0 - 5Ar_0 + 6r_0$:

- $5Ar_0 = (15, 15)$
- $6r_0 = (6, 6)$

Entonces $A^2r_0 - 5Ar_0 + 6r_0 = (9 - 15 + 6, 9 - 15 + 6) = (0, 0)$.
Por lo tanto, $m(A)r_0 = A^2r_0 - 5Ar_0 + 6r_0 = 0$.
El polinomio mínimo es:

$$m(x) = x^2 - 5x + 6.$$

Ejemplo 2:

Sea $A = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$ y $r_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.
Se obtiene:

- $r_0 = (1, 0, 0)$
- $Ar_0 = (2, 1, 0)$
- $A^2r_0 = A(Ar_0) = (5, 4, 0) = -3(1, 0, 0) + 4(2, 1, 0)$

Puede verificarse que A^2r_0 es combinación lineal de r_0 y Ar_0. Analizando los subespacios generados:

- $K_1 = \text{Span}\{(1, 0, 0)\}$ con $\dim(K_1) = 1$

- $K_2 = \text{Span}\{(1,0,0),(2,1,0)\}$ con $\dim(K_2) = 2$

- $K_3 = \text{Span}\{(1,0,0),(2,1,0),(5,4,0)\}$ con $\dim(K_3) = 2$ (pues $(5,4,0)$ es combinación lineal de los anteriores)

La saturación se produce a partir de K_3, y la dimensión del subespacio se estabiliza en 2.

Definición 1.4 (Estabilización) *Se dice que la dimensión de un subespacio de Krylov se estabiliza si existe $k \leq n$ tal que $K_k(A, r_0) = K_{k+1}(A, r_0) = \cdots$*

Como conclusión adicional, cualquier vector $S \in K_m(A, r_0)$ puede escribirse como $S = P_{m-1}(A)r_0$, donde P_{m-1} es un polinomio de grado $d \leq m-1$, es decir:

$$S = \alpha_0 r_0 + \alpha_1 A r_0 + \alpha_2 A^2 r_0 + \cdots + \alpha_{m-1} A^{m-1} r_0. \quad (7)$$

1.1.2 Propiedades algebraicas

Una parte importante del subespacio de Krylov, al igual que en el álgebra lineal en general, son los eigenvalores y eigenvectores. Entender cómo funcionan en este subespacio es fundamental para comprender cómo influyen en la generación de soluciones a nuestros problemas iniciales.

Aunque en el subespacio de Krylov no es necesario operar con matrices diagonalizables A, para desarrollar las siguientes propiedades algebraicas se considerarán de este modo, pues así es más fácil visualizar dónde se encuentran los eigenvectores y eigenvalores.

Recordemos la definición de matriz diagonalizable:

Definición 1.5 (Matriz Diagonalizable) *Una matriz A es diagonalizable si existe una matriz V de eigenvectores y una matriz diagonal de eigenvalores Λ tales que:*

$$A = V\Lambda V^{-1}, \quad (8)$$

donde $V \in \mathbb{R}^{n\times n}$, $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n)$ y V^{-1} es la inversa de V.

En el subespacio de Krylov podemos realizar el siguiente procedimiento:

Si $K_m(A, r_0) = \text{span}\{r_0, Ar_0, A^2 r_0, \ldots, A^{m-1} r_0\}$, podemos tomar las siguientes relaciones de la definición (1.5):

$$\begin{aligned}
&r_0 = VV^{-1}r_0 = Vw \\
&Ar_0 = (V\Lambda V^{-1})r_0 = V\Lambda(V^{-1}r_0) = V\Lambda w \\
&A^2 r_0 = A(Ar_0) = (V\Lambda V^{-1})(V\Lambda w) = V\Lambda(V^{-1}V)\Lambda w = V\Lambda^2 w
\end{aligned}$$

y en general se sigue que

$$A^k r_0 = (V\Lambda V^{-1})^k r_0 = V\Lambda^k V^{-1} r_0 = V\Lambda^k w,$$

donde $w = V^{-1}r_0$.

Por lo que podemos reescribir el subespacio de Krylov como:

$$\begin{aligned}
&K_m(A, r_0) = \text{span}\{Vw, V\Lambda w, V\Lambda^2 w, \ldots, V\Lambda^{m-1} w\} \\
&\Rightarrow K_m(A, r_0) = V \cdot \text{span}\{w, \Lambda w, \Lambda^2 w, \ldots, \Lambda^{m-1} w\} \\
&\therefore K_m(A, r_0) = VK_m(\Lambda, w) = VK_m(\Lambda, V^{-1}r_0).
\end{aligned} \quad (9)$$

Ahora podemos relacionar esto con el polinomio característico:

$$p(\lambda) = \det(A - \lambda I), \quad (10)$$

y recordar el teorema de Cayley-Hamilton:

Teorema 1.1 (Teorema de Cayley-Hamilton) *Sea*

$$p(\lambda) = (-1)^n\lambda^n + c_{n-1}\lambda^{n-1} + c_{n-2}\lambda^{n-2} + \cdots + c_2\lambda^2 + c_1\lambda + c_0$$

el polinomio característico de una matriz A de orden n. Entonces:

$$p(A) = (-1)^n A^n + c_{n-1}A^{n-1} + c_{n-2}A^{n-2} + \cdots + c_1 A + c_0 I$$

es la matriz nula. Es decir, cada matriz cuadrada A satisface su ecuación característica $p(A) = 0$.

Esto en el subespacio de Krylov se traduce en que si $p(A) = 0 \Rightarrow p(A)r_0 = 0$. Entonces, para el grado d del polinomio mínimo de r_0 se satisface que $d \leq n$, donde n es el grado del polinomio característico. De esto se puede ver que este es un criterio de estabilización para la dimensión del subespacio, pues si $d \geq n$ en estos polinomios se obtendrán combinaciones lineales de los términos anteriores para generar el subespacio, como se vio anteriormente.

Otra discusión importante por señalar es que si para un eigenvalor λ_i, $|\lambda_i|$ es grande, entonces la dirección del eigenvector V_i "aparecerá" rápidamente en K_m; es decir, los eigenvectores dominantes capturan la acción de A.

Si $|\lambda_1| \gg |\lambda_2|$, el subespacio se alineará más rápido con

el eigenvector V_1.

En conclusión, los eigenvalores pequeños contribuirán poco a K_m y A actuará más fuertemente en las direcciones donde $\|AV\|$ es grande y donde hay eigenvalores y eigenvectores más grandes.

1.1.3 Propiedad de aproximación óptima en el subespacio de Krylov

En relación con el problema inicial planteado y la forma en que se aplicará el método, a continuación se describe brevemente cómo se construye una corrección en el subespacio de Krylov.

Para el sistema $Ax = b$, si x_0 es una aproximación inicial y $r_0 = b - Ax_0$ es el residual correspondiente, entonces puede considerarse una corrección en el subespacio de Krylov de la forma:

$$x_m = x_0 + p_m(A)r_0 \in x_0 + K_m(A, r_0), \tag{11}$$

donde p_m es un polinomio de grado a lo más $m - 1$.

1.2 Método del Residual Mínimo Generalizado (GMRES)

Regresando al problema inicial planteado en (1), para evitar las limitaciones de los métodos directos que construyen una solución exacta, se propone utilizar en su lugar la construcción de una secuencia de aproximaciones:

$$x_0, x_1, x_2, \ldots, x_m, \ldots,$$

de tal forma que se espera que estas aproximaciones converjan a la solución exacta x^*.

Para describir el método de GMRES puede usarse la siguiente definición:

Definición 1.6 *Para cada $m \geq 1$, el método GMRES produce una aproximación $x_m \in \mathbb{R}^n$ mediante una proyección sobre $K = K_m(A, r_0)$ ortogonal a $L = AK$, donde $K_m(A, r_0)$ es el m-ésimo subespacio de Krylov generado por A y r_0.*

Esto puede explicarse en términos generales como un método iterativo que busca obtener en la m-ésima iteración una corrección de la forma vista anteriormente:

$$x_m \in x_0 + K_m(A, r_0),$$

tal que minimice la norma del residual $\|b - Ax_m\|$, es decir:

$$\text{mín } \|b - Ax_m\|, \tag{12}$$

donde, como se vio anteriormente, $r_0 = b - Ax_0$.

Para ello, se restringe la búsqueda de soluciones en el subespacio de Krylov a vectores de la forma:

$$x_m = x_0 + V_m y \tag{13}$$

donde $V_m \in \mathbb{R}^{n \times m}$ es una **base ortonormal** de $K_m(A, r_0)$ y $y \in \mathbb{R}^m$ es un vector de coeficientes por determinar. Así, el problema de minimización se reduce a:

$$\begin{aligned} \|b - Ax_m\| &= \|b - A(x_0 + V_m y)\| \\ &= \|b - Ax_0 - AV_m y\| \\ &= \|r_0 - AV_m y\|. \end{aligned} \tag{14}$$

Si el método de GMRES no se ha entendido completamente en este punto, no hay problema, pues el algoritmo completo se detallará más adelante. No obstante, en esta sección se ha mencionado un aspecto crucial para la resolución del problema: para construir las soluciones x_m es necesario obtener una **base ortonormal** del subespacio de Krylov. Dado que esto conecta el subespacio de Krylov con el método de GMRES, la pregunta más importante en este momento es: ¿cómo construir esta base? Además, la construcción debe ser eficiente para no incrementar el costo computacional del método.

Es por ello que a continuación se discutirá el proceso mediante el cual puede realizarse esta construcción de manera eficiente: el proceso de Arnoldi.

1.3 Proceso de Arnoldi

El proceso de Arnoldi es un algoritmo para construir una base ortonormal del subespacio de Krylov $K_m(A, r_0)$, el cual debe satisfacer tres criterios importantes:

- $\text{Span}\{v_1, v_2, \ldots, v_m\} = K_m(A, r_0)$ (subespacio generado),
- $v_i \cdot v_j = \delta_{ij}$, donde δ_{ij} es la delta de Kronecker (ortonormalidad),
- $AV_m = V_m H_m + h_{m+1,m} v_{m+1} e_m^T$ (relación fundamental).

A continuación se desarrollará paso a paso la forma en

que opera el proceso de Arnoldi, y posteriormente se deducirá la relación fundamental, la cual será importante al combinar este proceso con GMRES.

1.3.1 Desarrollo del proceso de Arnoldi

Se pretende obtener una base ortonormal tal que:

$$\text{Span}\{v_1, v_2, \ldots, v_m\} = K_m(A, r_0). \tag{15}$$

Para lograr esto, se sigue el siguiente proceso:

Paso 0

Se toma el primer vector v_1 como el vector unitario en la dirección de r_0 (según la definición (2)), de modo que:

$$v_1 = \frac{r_0}{\|r_0\|}, \quad \text{donde} \quad \|r_0\| = \beta. \tag{16}$$

Ahora se define la j-ésima iteración como:

$$w = A \cdot v_j. \tag{17}$$

Esto se justifica porque:

$$\begin{aligned}
&\text{Si} \quad \text{Span}\{v_1, \ldots, v_j\} = K_j(A, r_0), \\
&\text{donde} \\
&v_j \in K_j = \text{Span}\{r_0, Ar_0, \ldots, A^{j-1}r_0\}, \\
&\Rightarrow v_j = \alpha_0 r_0 + \alpha_1 A r_0 + \cdots + \alpha_{j-1} A^{j-1} r_0, \\
&\Rightarrow A \cdot v_j = \alpha_0 A r_0 + \alpha_1 A^2 r_0 + \cdots + \alpha_{j-1} A^j r_0, \\
&\Rightarrow v_1, \ldots, v_j \in K_{j+1}, \\
&\Rightarrow A \cdot v_j \in AK_j \subseteq AK_{j+1}, \\
&\text{además} \quad A^j r_0 \quad \text{está en} \quad Av_j, \\
&\Rightarrow \text{Span}\{v_1, \ldots, v_j\} \subseteq K_{j+1}(A, r_0), \\
&\therefore \text{Span}\{v_1, \ldots, v_j\} = K_{j+1}(A, r_0).
\end{aligned}$$

Paso 1

A continuación se realiza una ortogonalización de Gram–Schmidt:

$$\begin{aligned}
&\text{Para } i = 1 \text{ hasta } j: \\
&\quad h_{ij} = w \cdot v_i = (A \cdot v_j) \cdot v_i \quad \text{(proyección de } Av_j \text{ sobre } v_i), \\
&\quad w = w - h_{ij} v_i.
\end{aligned} \tag{18}$$

En este paso se ha construido:

$$\begin{aligned}
w &= (\text{componente en } K) + (\text{componente ortogonal a } K) \\
&= (w \cdot v_1)v_1 + \cdots + (w \cdot v_j)v_j + w_\perp, \\
&\text{tal que} \quad w_\perp \perp \text{Span}\{v_1, \ldots, v_j\}.
\end{aligned} \tag{19}$$

Al definir h_{ij}, se obtiene un vector que contiene solo las componentes en el subespacio; así, al restar $h_{ij}v_i$, se eliminan las componentes en la dirección de v_i (que ya pertenece al subespacio actual), obteniendo un vector puramente nuevo para expandir el subespacio.

Una pregunta natural en este punto es si realmente $w = w_\parallel + w_\perp$. Para aclararlo, se presenta la siguiente demostración.

Demostración 1.1 ***Demostrar que*** $w = w_\parallel + w_\perp$.
Sea $w_\parallel = (w \cdot v_1)v_1 + (w \cdot v_2)v_2 + \cdots + (w \cdot v_j)v_j$,
y sea $w_\perp = w - w_\parallel$.
Basta demostrar que $w_\perp \perp \text{Span}\{v_1, \ldots, v_j\}$.
Consideremos $w_\perp \cdot v_k = (w - w_\parallel) \cdot v_k$, *para* $v_k \in \text{Span}\{v_1, \ldots, v_j\}$.
Si no se cumple la perpendicularidad, entonces $w_\perp \cdot v_k \neq 0$.
Observemos que:

$$w_\perp \cdot v_k = w \cdot v_k - \left[(w \cdot v_1)(v_1 \cdot v_k) + \cdots + (w \cdot v_j)(v_j \cdot v_k)\right].$$

Dado que $v_i \cdot v_k = \delta_{ik}$ *y* $v_i \parallel v_k$ *solo cuando* $i = k$, *se tiene:*

$$w_\perp \cdot v_k = w \cdot v_k - (w \cdot v_k) \cdot 1 = 0.$$

Por lo tanto, $w_\perp \perp v_k$ *para todo* k, *y en consecuencia* $w_\perp \perp \text{Span}\{v_1, \ldots, v_j\}$.
$\therefore w = w_\parallel + w_\perp$.

Paso 2

En este paso se normaliza w, ya que su norma puede ser muy grande ($\|w\| \gg 1$) o muy pequeña ($\|w\| \approx 0$). Además, dado que todo vector depende de A y de v_j, la normalización ayuda a evitar errores numéricos.

Para llevar a cabo la normalización, se requiere un vector unitario que apunte en la dirección de la componente ortogonal. El procedimiento es el siguiente:

- Se define la norma $h_{j+1,j} = \|w\|$ (nótese que esta notación es distinta de la h_{ij} del Paso 1).

- Se verifica si es posible normalizar:

  Si $h_{j+1,j} \neq 0$,

  $\Rightarrow v_{j+1} = \dfrac{w}{h_{j+1,j}}$ (se normaliza),

  $\Rightarrow w = v_{j+1} \cdot h_{j+1,j}$.

  Si $h_{j+1,j} = 0$,

  se detiene el proceso, pues el subespacio es invariante.

Para ilustrar este proceso, se presenta el siguiente ejemplo.

Ejemplo 3

Sean $v_1 = (1,0,0)$, $v_2 = (0,1,0)$ y $Av_2 = (2,3,4)$. Mediante ortogonalización de Gram–Schmidt:

- $h_{12} = (2,3,4) \cdot (1,0,0) = 2$,
- $w = (2,3,4) - 2 \cdot (1,0,0) = (0,3,4)$,
- $h_{22} = (0,3,4) \cdot (0,1,0) = 3$,
- $w = (0,3,4) - 3 \cdot (0,1,0) = (0,0,4)$.

Normalizando:

- $h_{32} = \|(0,0,4)\| = 4$,
- $v_3 = (0,0,4)/h_{32} = (0,0,1)$.

Finalmente, se obtiene la base expandida:

$$\{(1,0,0),(0,1,0),(0,0,1)\}.$$

1.3.2 Obtención de la relación fundamental

Para obtener la relación fundamental se parte de la ecuación (18), de donde se tiene que:

$$w = Av_j - (h_{1j}v_1 + h_{2j}v_2 + \cdots + h_{jj}v_j). \tag{20}$$

De la ortonormalización se obtiene que:

$$\begin{aligned} w &= h_{j+1,j}v_{j+1}, \\ \Rightarrow Av_j &= h_{1j}v_1 + h_{2j}v_2 + \cdots + h_{jj}v_j + h_{j+1,j}v_{j+1}. \end{aligned} \tag{21}$$

Revisando Av_j para $j = 1, 2, \ldots, m$:

$$\begin{aligned} Av_1 &= h_{11}v_1 + h_{21}v_2, \\ Av_2 &= h_{12}v_1 + h_{22}v_2 + h_{32}v_3, \\ &\vdots \\ Av_m &= h_{1m}v_1 + h_{2m}v_2 + h_{3m}v_3 + \cdots + h_{mm}v_m + h_{m+1,m}v_{m+1}, \end{aligned}$$

y expresando esto en forma matricial compacta:

$$[Av_1|Av_2|\cdots|Av_m] = [v_1|v_2|\cdots|v_m|v_{m+1}] \cdot H, \tag{22}$$

donde H es una matriz de Hessenberg superior de tamaño $(m+1) \times m$ con la estructura:

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} & \cdots & h_{1m} \\ h_{21} & h_{22} & h_{23} & \cdots & h_{2m} \\ 0 & h_{32} & h_{33} & \cdots & h_{3m} \\ 0 & 0 & h_{43} & \cdots & h_{4m} \\ \ddots & \ddots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & \cdots & h_{mm} \\ 0 & 0 & 0 & \cdots & h_{m+1,m} \end{bmatrix}. \tag{23}$$

Nótese que una matriz de Hessenberg es una matriz casi triangular.

De lo anterior se sigue que:

$$\begin{aligned} V_m &= [v_1|v_2|\cdots|v_m] \in \mathbb{R}^{n\times m}, \\ V_{m+1} &= [v_1|v_2|\cdots|v_m|v_{m+1}] \in \mathbb{R}^{n\times(m+1)}, \\ H_m &= \text{primeras } m \text{ filas de } H \in \mathbb{R}^{m\times m}, \\ h_{m+1,m}e_m^T &= \text{última fila de } H, \\ \Rightarrow AV_m &= V_{m+1}H = V_mH_m + h_{m+1,m}v_{m+1}e_m^T, \\ \therefore AV_m &= V_mH_m + h_{m+1,m}v_{m+1}e_m^T. \end{aligned} \tag{24}$$

Recordando la restricción en el subespacio de Krylov para GMRES, se utiliza $r_0 + AV_my$. Usando la relación fundamental recién obtenida, se tiene:

$$r_0 + AV_my = \beta v_1 - (V_mH_m + h_{m+1,m}v_{m+1}e_m^T)y,$$

y dado que V_m es ortonormal a v_{m+1}, entonces:

$$r_0 + AV_my = V_m(\beta e_1 - H_my) - h_{m+1,m}(e_m^Ty)v_{m+1}.$$

Por lo tanto, para la minimización se llega a:

$$\|r_0 + AV_my\|^2 = \|\beta e_1 - H_my\|^2 + \|h_{m+1,m}(e_m^Ty)v_{m+1}\|^2. \tag{25}$$

Para ilustrar completamente el proceso de Arnoldi, se propone el siguiente ejemplo:

Ejemplo 4

Sea $A = \begin{bmatrix} 4 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix}$ y $r_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.

Se obtiene para la iteración $j = 1$:

$$
\begin{aligned}
&v_1 = (1, 0, 0),\\
&w = Av_1 = (4, 1, 0),\\
&h_{11} = w \cdot v_1 = 4,\\
&w = w - h_{11}v_1 = (0, 1, 0),\\
&h_{21} = \|w\| = 1,\\
&v_2 = w/h_{21} = (0, 1, 0),\\
&\text{Span}\{v_1, v_2\}.
\end{aligned}
$$

Para la iteración $j = 2$:

$$
\begin{aligned}
&w = Av_2 = (1, 3, 1),\\
&h_{12} = w \cdot v_1 = 1,\\
&w = w - h_{12}v_1 = (0, 3, 1),\\
&h_{22} = w \cdot v_2 = 3,\\
&w = w - h_{22}v_2 = (0, 0, 1),\\
&h_{32} = \|w\| = 1,\\
&v_3 = w/h_{32} = (0, 0, 1),\\
&\text{Span}\{v_1, v_2, v_3\} = \mathbb{R}^3 = K_3.
\end{aligned}
$$

Se obtiene así una base ortonormal para generar el subespacio de Krylov en este caso.

1.4 Método de GMRES completo

Una vez desarrollado el proceso de Arnoldi, es posible componer un algoritmo que describa el método de GMRES en su totalidad. Sin embargo, antes de ello, es importante revisar el criterio de minimización que se ha obtenido.

1.4.1 Criterio de minimización corregido

Partiendo de la norma final obtenida mediante el proceso de Arnoldi en la ecuación (25) ($\|r_0 + AV_m y\| = \|r_m\|$), se observa que ambos términos del lado derecho de la ecuación dependen de y, por lo que se busca:

$$\text{mín}\, \|r_m\| = \text{mín}\left(\|\beta e_1 - H_m y\|^2 + \|h_{m+1,m}(e_m^T y)v_{m+1}\|^2\right)^{1/2}. \tag{26}$$

Suponiendo que y^* es la solución de $\text{mín}\, \|\beta e_1 - H_m y\|$, se satisface entonces que:

$$H_m^T(\beta e_1 - H_m y^*) = 0, \tag{27}$$

y para la solución óptima y^* en general se cumple que $|e_m^T y^*|$ es pequeño, es decir, $|e_m^T y^*| \ll 1$. Por lo tanto, basta resolver:

$$\text{mín}\, \|r_0 + AV_m y\|^2 = \text{mín}\, \|\beta e_1 - H_m y\|^2. \tag{28}$$

1.4.2 Eigenvectores y eigenvalores

Una pregunta importante surge al usar la matriz de Hessenberg en lugar de A en GMRES: ¿dónde quedan los eigenvalores y eigenvectores en el subespacio de Krylov?

Recordando el resultado para AV_m obtenido en la ecuación (24), puede realizarse el siguiente procedimiento para visualizar los eigenvectores y eigenvalores:

$$V_m^T A V_m = V_m^T V_m H_m + h_{m+1,m} V_{m+1}(V_m^T V_{m+1}) e_m^T. \tag{29}$$

Por la ortonormalidad de los vectores de la base, se tiene $V_m^T V_{m+1} = 0$ y $V_m^T V_m = I$, de donde:

$$V_m^T A V_m = H_m. \tag{30}$$

Dado que H_m es la proyección de A sobre el subespacio K_m, puede usarse la descomposición espectral (1.5) para visualizar mejor los eigenvectores y eigenvalores. Así, se obtiene:

$$H_m = V_m^T V \Lambda V^{-1} V_m, \tag{31}$$

de lo cual es fácil ver que los eigenvalores y eigenvectores controlan cómo se construye el subespacio en cada iteración, hasta alcanzar la solución final.

1.4.3 Algoritmo para GMRES

Para ilustrar completamente la operación del método de GMRES, se presenta el siguiente algoritmo. No es posible incluir un ejemplo numérico concreto debido a la gran cantidad de operaciones involucradas y a la posible variación en los valores decimales obtenidos. En cualquier caso, se recomienda programar el método e imprimir cada paso para su mejor comprensión.

```
ENTRADA: A, b, x0, m, tol, max_iter
SALIDA: x (solución aproximada)
.

1. r0 = b - A*x0
2. beta = ||r0||
3. Si beta < tol: return x0  (ya convergió)
4. v1 = r0 / beta
5. Para iter = 1 hasta max_iter:
```

```
5.1. // Construir base de Krylov con Arnoldi
5.2. Para j = 1 hasta m:
      w = A * vj
      Para i = 1 hasta j:
          h_ij = producto_punto(w, vi)
          w = w - h_ij * vi
      h_j1j = ||w||
      Si h_j1j < epsilon_machine: break
      v_j1 = w / h_j1j

5.3. // Resolver minimización en subespacio
5.4. Resolver: min ||beta * e1 - Hm * y||
5.5. xm = x0 + Vm * y

5.6. // Verificar convergencia
5.7. rm = b - A * xm
5.8. Si ||rm|| < tol: return xm

5.9. // Preparar siguiente iteración (reinicio)
5.10. x0 = xm
5.11. r0 = rm
5.12. beta = ||r0||
5.13. v1 = r0 / beta
6. return xm
```

Un aspecto destacable de este método, y por tanto del algoritmo, es que los eigenvalores y eigenvectores no permanecen constantes en cada iteración de GMRES, sino que varían conforme se construye la base que acerca la solución al punto deseado.

PARTE II - Problema no lineal: Método de Newton-Raphson

El método de Newton-Raphson es un procedimiento para hallar las raíces de una ecuación no lineal o de un sistema de ecuaciones no lineales. Para el caso de una sola ecuación, las raíces son soluciones tales que:

$$f(x) = 0, \tag{32}$$

y se utiliza una expansión de Taylor con una corrección t para calcular la siguiente iteración:

$$f(x_k + t) = f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2 + \cdots. \tag{33}$$

Si bien esta descripción general puede no resultar completamente clara, en lugar de profundizar en este caso particular, a continuación se desarrollará el método completo para un sistema de ecuaciones no lineales.

1.5 Newton-Raphson para un sistema de ecuaciones no lineales

Para ilustrar el método, considérese el problema de resolver un sistema de ecuaciones no lineales con n incógnitas, hallando las raíces del sistema tales que:

$$F(x) = 0, \tag{34}$$

donde:

$$\begin{aligned} F_1(x_1, \cdots, x_n) &= 0, \\ &\vdots \\ F_n(x_1, \cdots, x_n) &= 0, \end{aligned} \tag{35}$$

y se cumplen las siguientes condiciones:

$F : \mathbb{R}^n \longrightarrow \mathbb{R}^n$,

- Dominio: vectores en $\mathbb{R}^n$ tales que $x \in \mathbb{R}^n$,
- Codominio: vectores en $\mathbb{R}^n$ tales que $F(x) \in \mathbb{R}^n$,
- No lineal: F no puede escribirse como $F(x) = Ax + b$.

Además, como característica importante, $F(x)$ debe ser una función suave; es decir, F es continuamente diferenciable. Esto garantiza que puedan realizarse aproximaciones lineales confiables.

1.5.1 Cómo resolver el problema

Para resolver este problema se utiliza una expansión de Taylor de primer orden. La expansión de Taylor multivariable se escribe como:

$$F_i(x_k + s) = F_i(x_k) + \nabla F_i(x_k)^T s + \frac{1}{2}s^T H_i(x_k)s + O(\|s\|^3), \tag{36}$$

donde:

- x_k es el estado actual,
- x_{k+1} es el nuevo estado,
- s es la corrección a calcular (Δx_k), tal que $x_{k+1} = x_k + s$ y entonces $s = x_{k+1} - x_k$; puede considerarse como el paso que se desea dar desde x_k,
- $H_i(x_k)$ es la matriz hessiana de F_i, de orden $n \times n$,
- $\nabla F_i(x_k)$ es el gradiente de F_i, es decir, un vector de tamaño $n \times 1$.

A partir del gradiente de F_i puede desarrollarse que:

$$\nabla F_i(x_k) = \left[\frac{\partial F_i}{\partial x_1}, \cdots, \frac{\partial F_i}{\partial x_n}\right]^T. \tag{37}$$

Este gradiente permite construir la matriz jacobiana J_k para el sistema de ecuaciones, definida como:

$$J_k = \begin{bmatrix} \nabla F_1(x_k)^T \\ \nabla F_2(x_k)^T \\ \vdots \\ \nabla F_n(x_k)^T \end{bmatrix} = J_F(x_k) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix}. \tag{38}$$

Con esto, podemos reducir la expansión de Taylor a:

$$F(x_k + s) = F(x_k) + J_k s + O(\|s\|^2), \tag{39}$$

donde $O(\|s\|^2)$ representa los términos de orden superior.

Para simplificar el problema $F(x_k + s) = 0$, puede utilizarse una aproximación lineal, considerando únicamente:

$$F(x_k) + J_k s \approx 0. \tag{40}$$

Un aspecto importante para la solución, que se discutirá más adelante, es acotar el error al aproximar la solución del sistema de ecuaciones. Para ello, es necesario mencionar el teorema de convergencia cuadrática.

Teorema 1.2 (Teorema de Convergencia Cuadrática) *Si F es una función dos veces continuamente diferenciable, x^* es una solución tal que $F(x^*) = 0$, y $J(x^*)$ es no singular, entonces para x_0 suficientemente cercano a x^* existe $C > 0$ tal que:*

$$\|x_{k+1} - x^*\| \leq C\|x_k - x^*\|^2. \tag{41}$$

Esto indica que el error en la iteración $k+1$, denotado ε_{k+1}, satisface $\varepsilon_{k+1} \leq C\varepsilon_k^2$.

Ejemplo 5

Supóngase que en la iteración 0 se tiene $C = 1$ y $x^* = 0$:

$$\begin{aligned} &\text{con} \quad \|x_0 - x^*\| = 0.1, \\ &\text{en la iteración 1:} \\ &\quad \Rightarrow \|x_1 - x^*\| \leq C(0.1)^2 = 0.01, \\ &\text{en la iteración 2:} \\ &\quad \Rightarrow \|x_2 - x^*\| \leq C(0.01)^2 = 0.0001. \end{aligned}$$

La obtención de esta cota de error puede explicarse mediante el siguiente desarrollo.
Partiendo de que $F(x^*) = F(x_k) + J_k(x^* - x_k) + O(\|x_k - x^*\|^2)$ y dado que $F(x^*) = 0$:

$$\begin{aligned} &\Rightarrow 0 = F(x_k) + J_k(x^* - x_k) + O(\|x_k - x^*\|^2), \\ &\Rightarrow J_k(x^* - x_k) = -F(x_k) + O(\|x_k - x^*\|^2). \end{aligned}$$

Recordando que $x_{k+1} = x_k - J_k^{-1}F(x_k)$:

$$\begin{aligned} x_{k+1} - x^* &= x_k - x^* - J_k^{-1}F(x_k) \\ &= J_k^{-1}[J_k(x_k - x^*) - F(x_k)] \\ &= J_k^{-1}[O(\|x_k - x^*\|^2)]. \end{aligned}$$

Por lo tanto:

$$\|x_{k+1} - x^*\| \leq \|J_k^{-1}\| \cdot \|O(\|x_k - x^*\|^2)\|.$$

Si $C = \|J_k^{-1}\| \cdot \|H(x)\|$ (donde H es la matriz hessiana), entonces:

$$\therefore \|x_{k+1} - x^*\| \leq C\|x_k - x^*\|^2.$$

Ahora que se han cubierto todos los aspectos referentes a este método, es posible explicar paso a paso cómo ejecutar el método de Newton-Raphson.

1.5.2 Cómo ejecutar Newton-Raphson

Paso 1

Se elige x_0 como una estimación inicial de la solución. Dado que se parte de una aproximación, la convergencia del método dependerá de una adecuada selección de x_0.

Paso 2

Se calcula $F_k = F(x_k)$ evaluando cada componente $F_i(x_k)$, lo cual permite medir qué tan lejos se está de la solución.

Paso 3

Se calcula $J_k = J_F(x_k)$ determinando las derivadas parciales $\frac{\partial F_i}{\partial x_j}$. No obstante, evaluar la jacobiana puede tener un costo computacional elevado para n grande, por lo que en tales casos suelen emplearse diferencias finitas.

Paso 4

Se resuelve el sistema lineal $J_k s_k = -F_k$. Este es un sistema de n ecuaciones con n incógnitas, y dado que se trata de un problema matricial como el planteado en la parte

de solución de problemas lineales, puede utilizarse cualquier método numérico adecuado para resolverlo.

Paso 5

Se actualiza $x_{k+1} = x_k + s_k$, tomando el paso s_k sin modificarlo. Cabe señalar que, si F es altamente no lineal, podría sobrepasarse la solución.

Paso 6

Para evitar lo anterior, se emplea el criterio de convergencia cuadrática discutido previamente, asignando una tolerancia $tol \sim \varepsilon$ y verificando que $\|F(x_{k+1})\| < tol$.

A continuación se presenta un algoritmo sintetizado del método de Newton-Raphson:

```
ENTRADA: x0, tol, max_iter
1. x = x0
2. Para k = 0 hasta max_iter-1:
   2.1. F_val = F(x)
   2.2. Si ||F_val|| < tol:
           return x (éxito)
   2.3. J = Jacobiana_F(x)
   2.4. Si |det(J)| < epsilon_machine:
           return error ("Jacobiana singular")
   2.5. Resolver J * s = -F_val para s
   2.6. x_nuevo = x + s
   2.7. Si ||x_nuevo - x|| < tol:
           return x_nuevo
   2.8. x = x_nuevo
3. return error ("No convergió en max_iter")
SALIDA: x (solución aproximada) o mensaje de error
```

PARTE III - Método de Newton-Krylov

Una vez estudiadas en profundidad las partes lineal y no lineal de la solución, solo resta integrar el método en su conjunto para construir la formulación completa de Newton-Krylov.

1.6 Cómo ejecutar Newton-Krylov

Se parte del problema:

$$F(x) = 0.$$

Paso 1

Se calcula la matriz jacobiana explícita $J(x_k)$.

Paso 2

Se formula el sistema lineal $J(x_k)s_k = -F(x_k)$.

Paso 3

Se resuelve el sistema lineal mediante GMRES, tomando $r_0 = -F_k$ y construyendo el subespacio de Krylov $K_m = \text{Span}\{r_0, J_k r_0, \cdots, J_k^{m-1} r_0\}$, para hallar una solución aproximada s_k en K_m.

Paso 4

Se verifica si la solución obtenida satisface $\|J_k s_k + F_k\| \leq \eta_k \|F_k\|$. Si se cumple, se devuelve la solución; en caso contrario, se expande el subespacio y se repite el proceso desde el inicio.

En este último paso, puede expresarse el criterio de convergencia como:

$$\frac{\|J_k s_k + F_k\|}{\|F_k\|} \leq \eta_k,$$

asignando simplemente un valor $\eta_k \sim \varepsilon$.

1.7 Newton-Krylov Libre de Jacobiano (Jacobian Free Newton-Krylov - JFNK)

Como se observó en el método usual de Newton-Krylov, se emplea una matriz jacobiana con derivadas parciales. Sin embargo, en este método numérico las derivadas parciales pueden reemplazarse por diferencias finitas.

Dado que en GMRES se utiliza $w = J_k v$, mediante diferencias finitas hacia adelante se obtiene la siguiente relación:

$$J_k v = \frac{F(x_k + \varepsilon v) - F(x_k)}{\varepsilon}. \tag{42}$$

Esto conduce a un sistema que evita el uso explícito de la jacobiana:

$$\frac{F(x_k + \varepsilon v) - F(x_k)}{\varepsilon} = \begin{bmatrix} \frac{F_1(x_1+\varepsilon v_1, \cdots, x_n+\varepsilon v_n) - F_1(x_k)}{\varepsilon} \\ \vdots \\ \frac{F_n(x_1+\varepsilon v_1, \cdots, x_n+\varepsilon v_n) - F_n(x_k)}{\varepsilon} \end{bmatrix}. \tag{43}$$

1.8 Eigenvectores y Eigenvalores en Newton-Krylov

Al operar con matrices de derivadas parciales, es posible perder de vista los eigenvalores y eigenvectores, cuya importancia en el método de GMRES ya se ha establecido.

A continuación se intentará visualizar cómo se manifiestan estos conceptos en el método de Newton-Krylov.

Sea $K_m = \text{Span}\{r_0, J_k r_0, \cdots, J_k^{m-1} r_0\}$. Si J_k tiene vectores propios $v_1, v_2, \ldots, v_n$ con valores propios $\lambda_1, \ldots, \lambda_n$, entonces cualquier vector como r_0 puede expresarse como $r_0 = \sum C_i v_i$. Por lo tanto:

$$J_k^k r_0 = \sum C_i \lambda_i^k v_i, \tag{44}$$

de modo que:

$$\begin{aligned} U_1 &= \frac{r_0}{\|r_0\|} = \sum \alpha_{1i} v_i, \\ U_2 &= J_k U_1 = \sum \alpha_{1i} \lambda_i v_i, \\ U_3 &= J_k U_2 = \sum \alpha_{1i} \lambda_i^2 v_i, \\ &\vdots \end{aligned} \tag{45}$$

Además, en GMRES se construye la relación $J_k U_m \approx U_{m+1} H_m$, donde los vectores propios pueden identificarse en H_m.

De lo anterior se deduce que en cada iteración de Newton-Krylov se expande la base del espacio generado, obteniéndose distintos eigenvectores y eigenvalores en cada paso. Por tanto, la generación de eigenvectores y eigenvalores depende de la solución inicial propuesta x_0.

Sin embargo, si λ_i es muy grande, el sistema tenderá a moverse preferentemente en esa dirección; si λ_i es pequeño, el movimiento en dicha dirección será menor. Así:

- Si $\text{Re}(\lambda_i) < 0$, la dirección hacia una solución concreta será más estable.
- Si $\text{Re}(\lambda_i) > 0$, la dirección hacia una solución concreta será más inestable.

En conclusión, los eigenvalores **NO** determinan la cantidad de soluciones existentes, sino que caracterizan cada solución individual. Además, estos eigenvalores variarán dependiendo de la solución inicial propuesta, es decir, de la iteración en la que se encuentre el método.

Como concepto final, a cada solución individual puede asociársele un espectro, por ejemplo:

- $J(x_1^*)$ con espectro $\sigma_1 = \{\lambda_{11}, \cdots, \lambda_{1n}\}$,
- $J(x_2^*)$ con espectro $\sigma_2 = \{\lambda_{21}, \cdots, \lambda_{2n}\}$.

1.8.1 Cuencas de atracción

Otro concepto importante para la discusión subsiguiente es el de **cuencas de atracción**. A continuación se presentan cinco definiciones relevantes:

Definición 1.7 (Raíces) *Se definen las raíces como los puntos del sistema de ecuaciones en los que se cumple:*

$$F(x) = 0.$$

Definición 1.8 (Valles) *Se definen los valles como las regiones alrededor de las raíces donde el sistema presenta un comportamiento de minimización local.*

Definición 1.9 (Cuenca) *Se define una cuenca como la región del espacio de entradas desde la cual el método de Newton converge a una raíz particular.*

Definición 1.10 (División) *Se define una división como la frontera que separa dos cuencas de atracción diferentes.*

Definición 1.11 (Cuenca de Atracción) *Para cada solución x_i^* de $F(x) = 0$, existe una cuenca de atracción denotada por $B(x_i^*)$, definida como:*

$$B(x_i^*) = \{x_0 \in \mathbb{R}^n \text{ tal que Newton con } x_0 \longrightarrow x_i^*\}, \tag{46}$$

es decir, el conjunto de todos los puntos iniciales desde los cuales el método de Newton converge a x_i^*.

1.8.2 Existencia de soluciones múltiples

Al resolver un sistema de ecuaciones no lineales, es común encontrar múltiples soluciones; es decir, diferentes raíces que satisfacen $F(x) = 0$. En el caso del método de Newton-Krylov, la solución encontrada depende del camino seguido, ya que al ser un método iterativo basado en aproximaciones, puede converger a diferentes raíces.

Como se observó en el desarrollo del método de Newton-Krylov, se emplean dos tolerancias diferentes: una asociada a la minimización en el subespacio de Krylov (GMRES) y otra asociada al error residual en Newton-Raphson. Una observación relevante es que, dependiendo de cómo se ajusten estas tolerancias, el método puede converger a una solución u otra. A continuación se exploran los casos posibles:

- **Tolerancia de GMRES más estricta**: Al resolver $J(x_k)s = -F(x_k)$ con mayor precisión, el paso s es más exacto y el descenso hacia la solución es más pronunciado. Esto generalmente conduce a una solución

más cercana a la inicial, capturando eigenvalores locales.

- **Tolerancia de GMRES más laxa**: La solución de $J(x_k)s = -F(x_k)$ es más aproximada, por lo que s puede ser más ruidoso o desviarse más. Esto puede provocar saltos entre cuencas de atracción, obteniendo eigenvalores diferentes y convergiendo a una solución más lejana.
- **Tolerancia de Newton más estricta**: Solo se aceptan soluciones muy precisas, lo que puede impedir que los pasos de GMRES alcancen la convergencia, incluso cuando se tiende a una solución con eigenvalores estables ($\mathrm{Re}(\lambda_i) < 0$).
- **Tolerancia de Newton más laxa**: Se permiten soluciones "suficientemente buenas", facilitando que los pasos de GMRES sean suficientes. Esto permite explorar más ramas de solución, hallando eigenvalores diferentes y posiblemente más inestables ($\mathrm{Re}(\lambda_i) > 0$).

En conclusión:

- **GMRES** controla la dirección del paso y la exploración de saltos entre cuencas, determinando qué eigenvectores dominan.
- **Newton** controla la precisión final aceptable, la calidad de las soluciones y el criterio de parada.
- Es importante limitar el número de iteraciones en la programación, ya que de lo contrario el método podría no converger (ni en GMRES ni en Newton), y el tiempo de espera para alcanzar las tolerancias podría ser excesivo. Esto debe evitarse para mantener un algoritmo eficiente.
- Se reitera que la convergencia a una solución también depende de la elección de la solución inicial x_0.

A continuación se presenta un ejemplo para visualizar la existencia de soluciones múltiples.

Ejemplo 6

Para este ejemplo se plantea el sistema de ecuaciones:

$$x^2 + y^2 = 1 \quad \text{(circunferencia de radio } r = 1\text{)},$$
$$y = x^3 \quad \text{(ecuación cúbica)}.$$

Las soluciones exactas se encuentran en los puntos $x_1^* = (0.82603,\ 0.56362)$ y $x_2^* = (-0.82603,\ -0.56362)$. Para observar cómo el método de Newton-Krylov produce múltiples soluciones (convergentes y no convergentes), se propone el siguiente código y sus salidas:

Código

```python
import numpy as np
from scipy.sparse.linalg import gmres, LinearOperator

def F(x):
    return np.array([x[0]**2 + x[1]**2 - 1, x[0]**3 - x[1]])

def Jv(v, x, epsilon=1e-8):
    return (F(x + epsilon*v) - F(x)) / epsilon

def newton_krylov(x0, tol=1e-3, max_iter=20,
                  gmres_tol=1e-6, gmres_maxiter=10):
    x = x0.copy()

    for k in range(max_iter):
        Fx = F(x)
        if np.linalg.norm(Fx) < tol:
            return x, k, True

        A = LinearOperator((2, 2), matvec=lambda v: Jv(v, x))
        s, info = gmres(A, -Fx, rtol=gmres_tol,
                        maxiter=gmres_maxiter, atol=0)

        if info != 0:
            return x, k, False

        x = x + s

    return x, max_iter, False

# CASO 1: Debería converger a Solución 1
print("CASO 1: x0 = [0.9, 0.5]")
x0_1 = np.array([0.9, 0.5])
sol1, iter1, conv1 = newton_krylov(x0_1)
print(f"Resultado: [{sol1[0]:.6f}, {sol1[1]:.6f}]")
print(f"Iteraciones: {iter1}, Convergió: {conv1}")
print()

# CASO 2: Debería converger a Solución 2
print("CASO 2: x0 = [-0.9, -0.5]")
x0_2 = np.array([-0.9, -0.5])
sol2, iter2, conv2 = newton_krylov(x0_2)
print(f"Resultado: [{sol2[0]:.6f}, {sol2[1]:.6f}]")
print(f"Iteraciones: {iter2}, Convergió: {conv2}")
print()

# CASO 3: Caso ambivalente
print("CASO 3: x0 = [-0.1, 0.1]")
x0_3 = np.array([-0.1, 0.1])
sol3, iter3, conv3 = newton_krylov(x0_3)
print(f"Resultado: [{sol3[0]:.6f}, {sol3[1]:.6f}]")
print(f"Iteraciones: {iter3}, Convergió: {conv3}")
print()

# CASO 4: Con diferentes parámetros de GMRES
print("CASO 4: x0 = [-0.1, 0.1] con más iteraciones GMRES")
x0_4 = np.array([-0.1, 0.1])
sol4, iter4, conv4 = newton_krylov(x0_4, gmres_maxiter=20,
                                   gmres_tol=1e-8)
print(f"Resultado: [{sol4[0]:.6f}, {sol4[1]:.6f}]")
print(f"Iteraciones: {iter4}, Convergió: {conv4}")
print()

# CASO 5: CAMBIANDO PARÁMETROS PARA OBTENER SOLUCIÓN 2
print("CASO 5: x0 = [-0.1, 0.1] con tolerancia GMRES muy laxa")
sol5, iter5, conv5 = newton_krylov(np.array([-0.1, 0.1]),
```

```
                                    gmres_tol=1e-1)
print(f"Resultado: [{sol5[0]:.6f}, {sol5[1]:.6f}]")
print(f"Iteraciones: {iter5}, Convergió: {conv5}")
print()

print("CASO 6: x0 = [-0.1, 0.1] con muy pocas iteraciones GMRES")
sol6, iter6, conv6 = newton_krylov(np.array([-0.1, 0.1]),
                                    gmres_maxiter=2)
print(f"Resultado: [{sol6[0]:.6f}, {sol6[1]:.6f}]")
print(f"Iteraciones: {iter6}, Convergió: {conv6}")
print()

print("CASO 7: x0 = [-0.1, 0.1] con tolerancia Newton más estricta")
sol7, iter7, conv7 = newton_krylov(np.array([-0.1, 0.1]),
                                    tol=1e-12)
print(f"Resultado: [{sol7[0]:.6f}, {sol7[1]:.6f}]")
print(f"Iteraciones: {iter7}, Convergió: {conv7}")
print()
```

Salida

```
CASO 1: x0 = [0.9, 0.5]
Resultado: [0.826062, 0.563608]
Iteraciones: 2, Convergió: True

CASO 2: x0 = [-0.9, -0.5]
Resultado: [-0.826062, -0.563608]
Iteraciones: 2, Convergió: True

CASO 3: x0 = [-0.1, 0.1]
Resultado: [0.826454, 0.563395]
Iteraciones: 9, Convergió: False

CASO 4: x0 = [-0.1, 0.1] con más iteraciones GMRES
Resultado: [-5.255669, -0.155670]
Iteraciones: 1, Convergió: False

CASO 5: x0 = [-0.1, 0.1] con tolerancia GMRES muy laxa
Resultado: [-0.826279, -0.563547]
Iteraciones: 8, Convergió: True

CASO 6: x0 = [-0.1, 0.1] con muy pocas iteraciones GMRES
Resultado: [0.826454, 0.563395]
Iteraciones: 9, Convergió: False

CASO 7: x0 = [-0.1, 0.1] con tolerancia Newton más estricta
Resultado: [0.826454, 0.563395]
Iteraciones: 9, Convergió: False
```

2 Implementación de Newton-Krylov a FreeGSNKE

Una vez estudiado el método de Newton-Krylov, es momento de ilustrar su implementación para la solución de equilibrio de flujo magnético en un tokamak, específicamente en el código FreeGSNKE.

2.1 Desarrollo teórico

FreeGSNKE es una evolución de FreeGS en la que se modifica el método numérico para resolver la ecuación de Grad-Shafranov y las ecuaciones asociadas para determinar el equilibrio. Dado que esta ecuación es la más importante, nos centraremos en su solución en el siguiente desarrollo.

Partiendo de:

$$\Delta^*\psi = -\mu_0 R J_\phi(\psi), \tag{47}$$

y buscando $F(\psi) = 0$, se define:

$$F(\psi) = \Delta^*\psi + \mu_0 R J_\phi(\psi) = 0, \tag{48}$$

y utilizando el método de Newton, se busca resolver:

$$\mathbf{J}(\psi_k)V = -F(\psi_k). \tag{49}$$

Nótese que el jacobiano se denota por $\mathbf{J}(\psi_k)$, el cual no debe confundirse con la densidad de corriente $J_\phi(\psi)$.

Ahora se define el residual como:

$$F(\psi) = L\psi + s(\psi) - B, \tag{50}$$

donde:

- L es la matriz de discretización de Δ^* mediante elementos finitos,
- $s(\psi)$ es el vector que discretiza $\mu_0 R J_{\phi,\mathrm{pl}}(\psi)$, considerando la dependencia no lineal de los términos de presión y campo toroidal,
- B es la contribución de las corrientes fijas de las bobinas, y dado que $\mu_0 R J_{\phi,\mathrm{cond}}$ no depende de ψ, se tiene $\frac{\partial B}{\partial \psi} = 0$.

Esto resulta en un residual de la forma:

$$F(\psi) = L\psi + \mu_0 M(R J_{\phi,\mathrm{pl}}(\psi)) = 0, \tag{51}$$

donde M es una matriz de proyección que mapea la densidad de corriente al espacio de elementos finitos.

La discretización de Δ^* está dada por:

$$\begin{aligned}\Delta^*\psi =& \frac{\psi_{j,l+1} - 2\psi_{j,l} + \psi_{j,l-1}}{\Delta R^2} - \frac{\psi_{j,l+1} - \psi_{j,l-1}}{2R_l \Delta R} \\ &+ \frac{\psi_{j+1,l} - 2\psi_{j,l} + \psi_{j-1,l}}{\Delta Z^2}.\end{aligned} \tag{52}$$

Además, la ecuación para la densidad de corriente es:

$$J_{\phi,\mathrm{pl}} = R p'(\psi) + \frac{F F'(\psi)}{R\mu_0}. \tag{53}$$

A partir de estos resultados, el jacobiano resulta:

$$\mathbf{J}(\psi) = L + \mu_0 M \left[R^2 \frac{d^2 p}{d\psi^2} + \frac{1}{\mu_0} \left(\frac{dF}{d\psi} \right)^2 + \frac{1}{\mu_0} F \frac{d^2 F}{d\psi^2} \right]. \tag{54}$$

Para implementar esto en FreeGSNKE de manera eficiente, se utiliza el enfoque **JFNK** (Jacobian-Free Newton-Krylov):

$$\mathbf{J}(\psi) V \approx \frac{F(\psi + \varepsilon V) - F(\psi)}{\varepsilon}, \tag{55}$$

y considerando que Δ^* ya está discretizado, se obtiene:

$$\mathbf{J}(\psi) V \approx LV + \frac{S(\psi + \varepsilon V) - S(\psi)}{\varepsilon}. \tag{56}$$

Sin embargo, esto aún no constituye un sistema completo de ecuaciones. Para la solución general, se incluyen ecuaciones para la corriente I_p, el parámetro β_p, el flujo magnético en la frontera ψ_b, y los puntos X. Así, el sistema de ecuaciones queda compuesto por:

$$\begin{aligned}
F_2(\psi, \lambda, \beta_0) &= \int J_{\phi,\mathrm{pl}}(\psi, \lambda, \beta_0)\, dA - I_p^{\mathrm{target}} = 0, \\
F_3(\psi, \beta_0) &= \beta_p(\psi, \beta_0) - \beta_p^{\mathrm{target}} = 0, \\
F_4(\psi, I_{\mathrm{coil}}) &= \psi(R_b, Z_b) - \psi_b^{\mathrm{target}} = 0 \quad \text{(puntos de frontera)}, \\
F_5(\psi, I_{\mathrm{coil}}) &= \nabla\psi(R_x, Z_x) = 0 \quad \text{(puntos-X)},
\end{aligned} \tag{57}$$

con el vector de variables:

$$X = \begin{bmatrix} \psi \\ \lambda \\ \beta_0 \\ I_{\mathrm{coil}} \end{bmatrix}. \tag{58}$$

Un último aspecto relevante de la implementación es la tolerancia utilizada en FreeGSNKE. El **Strong Stopping Criterion** (SSC) empleado para la convergencia difiere del criterio residual estándar en Newton-Krylov.

Mientras que la convergencia en Newton se basa típicamente en el error residual:

$$\frac{\|F(\psi^*)\|}{\|F(\psi_0)\|} < \varepsilon_{\mathrm{res}}, \tag{59}$$

en FreeGSNKE se reemplaza por un error relativo, denominado SSC, definido como:

$$\frac{\max |\delta\psi|}{\psi_{\max} - \psi_{\min}} < \varepsilon_{\mathrm{rel}}. \tag{60}$$

2.2 Distintos problemas que se pueden encontrar en FreeGSNKE debidos al método de Newton-Krylov

Tras este desarrollo, aunque el método de Newton-Krylov aparenta ser más robusto que las iteraciones de Picard para hallar soluciones de equilibrio —y en efecto lo es—, su complejidad puede hacerlo más sensible al calcular un equilibrio. Esto deriva en varias consideraciones que deben tenerse en cuenta al utilizar este solucionador para obtener buenos resultados.

2.2.1 Problemas a considerar si el espacio de solución se expande a un dominio computacional más grande

Cuando se utilizan solucionadores basados en discretizaciones, un aspecto importante es la malla definida en el espacio de solución, como es usual en métodos de diferencias finitas.

Supóngase que se tiene una buena convergencia en un tokamak para una malla $(n_x \times n_y)$, y el dominio computacional se ha definido exclusivamente dentro de la cámara, sin incluir las bobinas. Si se desea extender la solución a todo el espacio del tokamak, podría pensarse en simplemente expandir el dominio computacional sin prestar mucha atención a la definición de la malla. Aunque esto parece inofensivo, es un error que podría acarrear problemas de no convergencia.

Esto se debe a que, al ampliar el dominio sin ajustar la malla, en el SSC (Strong Stopping Criterion) el valor $\psi_{\max} - \psi_{\min}$ podría incluir regiones de vacío lejano donde $\Delta\psi \approx 0$, es decir, donde ψ varía muy poco. En tal caso, el SSC sería artificialmente pequeño o no cumpliría el criterio de parada ε, y la solución inicial ψ_0 podría quedar fuera de la cuenca de atracción de una solución en este nuevo dominio computacional.

Otra posibilidad es que en ciertos puntos se tenga $\det(\mathbf{J}(\psi)) = 0$, lo que ocasionaría que no exista una dirección de Newton definida, o que GMRES encuentre una solución incorrecta o simplemente no encuentre solución.

2.2.2 Malla demasiado fina

Para resolver este tipo de problemas, podría considerarse incrementar el número de divisiones en la malla para obtener una malla más fina. Sin embargo, esta tampoco es una solución completamente adecuada.

Esto no solo ocasionaría un mayor costo computacional, ralentizando el cálculo, sino que también podría propagar más el error debido a la cantidad de operaciones realizadas.

2.2.3 Cómo mantener una proporción correcta entre el dominio y la fineza de la malla

Para lograr una buena relación entre el dominio y la malla, se propone la siguiente relación:

$$\Delta X = \frac{|L|}{CF_{\min}}, \tag{61}$$

donde $|L|$ es el tamaño del dominio computacional y $CF_{\min}$ es el tamaño de la característica física más pequeña. Por ejemplo, si una bobina está compuesta por pequeños rectángulos, estos se tomarán como $CF_{\min}$ para definir la malla.

Para entender mejor esto, considérese el siguiente ejemplo:

Supóngase que en el tokamak se tiene una bobina con secciones de aproximadamente 0.0127 y un dominio computacional de $L = 2$. Recordando que FreeGSNKE requiere que los puntos de la malla se definan como $2n + 1$, con $n \in \mathbb{N}$, entonces:

$$\Delta X = \frac{2}{0.0127} = 157.48 \approx 157 = 2n + 1.$$

Siguiendo el mismo procedimiento, se presenta la siguiente tabla como ejemplo, usando $\Delta R, \Delta Z \approx 0.0155$:

Dominio (R × Z)	Malla ($n_x \times n_y$)
1.5 × 1.5	97 × 97
2.0 × 2.0	129 × 129
2.5 × 2.5	161 × 161
3.0 × 3.0	193 × 193

Cuadro 1: Tabla de ejemplos para la relación dominio-malla

2.3 FreeGSNKE como método más realista que FreeGS

La siguiente discusión aborda si FreeGSNKE, con su método de Newton-Krylov, puede proporcionar resultados más ajustados a la realidad que FreeGS con iteraciones de Picard.

Para esta discusión, es importante destacar que FreeGS opera mediante iteraciones de Picard: en la iteración n se utiliza el flujo actual para calcular la corriente:

$$J_\phi^{(n)}(\psi) = J_{\phi,\text{pl}} = Rp'(\psi^n) + \frac{F(\psi^n)F'(\psi^n)}{R\mu_0}, \tag{62}$$

y luego se resuelve:

$$\Delta^* \psi^{n+1} = -\mu_0 R J_\phi^{(n)}, \tag{63}$$

obteniendo así el siguiente flujo. Esto implica que el campo magnético actualizado $\psi^{(n+1)}$ depende de la corriente del plasma del paso anterior.

Esto conlleva las siguientes consecuencias:

- Pueden aparecer corrientes negativas en regiones de alto vacío.
- Los puntos-X pueden presentar inconsistencias numéricas.
- Puede ocurrir una falsa convergencia, donde aparentemente la forma del plasma converge, pero al revisar los parámetros de salida se detecta una violación del equilibrio de fuerzas $J \times B = \nabla P$.

La ventaja de este método es que puede ser más estable al trabajar con datos ruidosos y sirve como una buena primera aproximación para encontrar equilibrios.

En cuanto al uso del método de Newton-Krylov, se logra una consistencia simultánea, ya que, como se vio, se resuelve (48), lo que genera un acoplamiento más exacto entre ψ y J_ϕ, convergiendo juntos hacia la solución. Por ello, el equilibrio final satisface la ecuación de Grad-Shafranov de manera más precisa.

Si no hay convergencia, esto puede deberse a un problema físico real, y la divergencia señalaría una inestabilidad física. Las múltiples soluciones numéricas serían entonces soluciones reales y no meramente artefactos numéricos.

Como ya se discutió, el método de Newton-Krylov es más sensible a inconsistencias en las mediciones, pero si la solución converge, se obtendrá un ajuste más preciso al equilibrio, satisfaciendo de manera más exacta el equilibrio de fuerzas.

Con esto, podría concluirse que FreeGSNKE es más confiable para diseños futuros.

2.4 Distintas soluciones con FreeGSNKE

Como ya se discutió, el método de Newton-Krylov puede converger a distintas soluciones dependiendo de la aproximación inicial y de cómo se ajusten las tolerancias en GMRES y Newton. Sin embargo, un problema común es que, al ejecutar el solucionador múltiples veces con los mismos parámetros iniciales, se obtienen soluciones diferentes, incluso cuando se borran las variables almacenadas y se reinicia el núcleo del intérprete de `Python`. Esto conduce a una discusión importante para intentar determinar por qué un solucionador teóricamente determinista, que debería seguir siempre el mismo camino, produce soluciones distintas.

2.4.1 Ruido numérico

Cuando una computadora recibe un número de entrada, debe convertirlo a binario bajo la representación de punto flotante `IEEE-754` para operar con él, y luego transformarlo nuevamente a un formato decimal para su visualización. Esta representación tiene la particularidad de generar diferencias microscópicas del orden de $\sim 1 \times 10^{-15}$.

Para ejemplificar esto, se muestra el siguiente código.

Ejemplo 7

Código

```python
import numpy as np

def demostracion_fpu():
    # Esto es más evidente con números que causan cancelación catastrófica
    a = 1.0000000000000001
    b = 1.0000000000000002
    c = -2.0000000000000003

    # Teóricamente: a + b + c = 0.0
    # En práctica:
    resultado = a + b + c

    print(f"a = {a:.20f}")
    print(f"b = {b:.20f}")
    print(f"c = {c:.20f}")
    print(f"a + b + c = {resultado:.20f}")
    print(f"Error: {abs(resultado):.2e}")

demostracion_fpu()
```

Salida

```
a = 1.00000000000000000000
b = 1.00000000000000022204
c = -2.00000000000000044409
a + b + c = -0.00000000000000044409
Error: 4.44e-16
```

2.5 Distintas formas de operar

Cuando se ejecuta un código, es posible que la CPU realice las operaciones en distintas unidades o módulos en diferentes ejecuciones. Lo anterior puede ocasionar que, para mayor eficiencia, la CPU conmute el orden de algunas operaciones que, aunque algebraicamente deberían ser invariantes, en una computadora pueden producir resultados diferentes.

Para ejemplificar esto, se muestra el siguiente código.

Ejemplo 8

Código

```python
def demostrar_cancelacion_catastrofica():
    print("=== CANCELACIÓN CATASTRÓFICA ===")

    # Números diseñados para cancelación
    x = 1.0e15
    y = 1.0e-5
    z = -1.0e15

    # Diferentes órdenes - resultados DRAMÁTICAMENTE diferentes
    orden1 = (x + y) + z  # (1e15 + 1e-5) - 1e15 = 1e-5
    orden2 = x + (y + z)  # 1e15 + (1e-5 - 1e15) = 0.0
    orden3 = (x + z) + y  # (1e15 - 1e15) + 1e-5 = 1e-5

    print(f"x = {x:.1e}, y = {y:.1e}, z = {z:.1e}")
    print(f"(x + y) + z = {orden1:.20f}")
    print(f"x + (y + z) = {orden2:.20f}")
    print(f"(x + z) + y = {orden3:.20f}")

demostrar_cancelacion_catastrofica()
```

Salida

```
=== CANCELACIÓN CATASTRÓFICA ===
x = 1.0e+15, y = 1.0e-05, z = -1.0e+15
(x + y) + z = 0.00000000000000000000
x + (y + z) = 0.00000000000000000000
(x + z) + y = 0.00001000000000000000
```

Cambiando x, y, z a los valores del ejemplo 7:

```
x = 1.0000000000000001
y = 1.0000000000000002
z = -2.0000000000000003
```

Salida

```
=== CANCELACIÓN CATASTRÓFICA ===
x = 1.0e+00, y = 1.0e+00, z = -2.0e+00
(x + y) + z = -0.00000000000000044409
x + (y + z) = -0.00000000000000022204
(x + z) + y = -0.00000000000000022204
```

2.5.1 Desfase entre la malla y $CF_{\min}$

Otra fuente de error significativo puede surgir cuando la relación propuesta por la ecuación (61) es muy dispar. Esto puede causar que la malla esté desajustada respecto al tamaño de $CF_{\min}$, de modo que, aunque las secciones construidas sean similares, no se resuelvan de manera uniforme.

La cantidad de puntos en cada sección puede variar, dando lugar a segmentos con menos divisiones. Para visualizar esto, se muestra el siguiente diagrama:

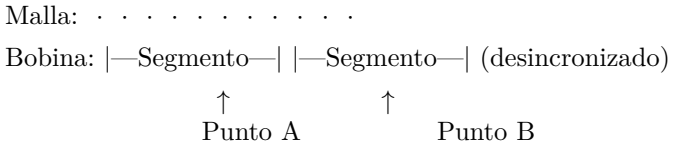

Figura 1: Diagrama de desajuste de la malla

Esto conlleva a diferencias numéricas debidas a la desalineación, lo que resulta en evaluaciones inconsistentes de ψ. Como consecuencia, los cálculos pueden verse afectados por redondeos, truncamientos o superposiciones indeseadas en comparación con otras secciones.

2.5.2 Propagación del error

Aunque estas diferencias numéricas pueden parecer pequeñas e inofensivas, es bien sabido que el error se propaga y puede generar inconsistencias en los cálculos. Para ilustrar esto, considérese el siguiente ejemplo.

Ejemplo 9

Supóngase un error inicial debido al ruido numérico del orden de 1×10^{-15}. Si se tiene un error por desfase en $F(\psi)$ de $\sim 1 \times 10^{-10}$, este error persistirá ya que, al sumar el ruido numérico, la magnitud del error se mantendrá en ese orden.

Este error se amplifica en el esquema JFNK (ver ecuación (55)). Supóngase que se requiere un error $\varepsilon \approx 1 \times 10^{-8}$. Entonces, $\frac{1\times10^{-10}}{1\times10^{-8}} = 1 \times 10^{-2}$. Por otra parte, si GMRES resuelve un sistema mal condicionado, es posible que el error final sea del orden de $\sim 1 \times 10^{-1}$.

2.6 Conclusión de por qué se obtienen múltiples soluciones y cómo evitarlas

La combinación de la propagación del error y el hecho de que la CPU puede reordenar las operaciones, obteniendo así distintos errores en distintas ejecuciones, explica por qué pueden obtenerse soluciones diferentes aun cuando el solucionador es teóricamente determinista.

Para evitar esto, la mejor estrategia es mantener una buena relación dada por (61), minimizando en lo posible este error, y utilizar el mismo valor para secciones pequeñas, incluso si pertenecen a componentes diferentes (como bobinas).

Si se logra minimizar estas propagaciones, un resultado aceptable se alcanzaría cuando el código converja con un $SSC \leq 1 \times 10^{-5}$.

Cabe recordar que el método de Static Forward determina la forma del plasma a partir de las corrientes de las bobinas, por lo que este método será más sensible a los errores debido a la mayor libertad en el cálculo de la solución.

Nota final

Estas notas no tienen puestas aún las referencias de forma rigurosa, pero para profundizar en algún tema se puede revisar [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]

Referencias

[1] Ben Dudson. *FreeGS 0.8.3.DeV13+Gdd8701e Documentation*. 2023. URL: https://freegs.readthedocs.io/en/latest/.

[2] Freegs-Plasma. *GitHub - freegs-plasma/freegs: Free boundary Grad-Shafranov solver*. URL: https://github.com/freegs-plasma/freegs.

[3] Andrés Camacho López y Uriel Yafté Sánchez Almaguer. *GitHub - Andr3s3/Freegs-pruebas*. 2025. URL: https://github.com/Andr3s3/Freegs-pruebas.git.

[4] Young Mu Jeon. «Development of a free-boundary tokamak equilibrium solver for advanced study of tokamak equilibria». en. En: *Journal of the Korean Physical Society* 67.5 (2015), págs. 843-853. DOI: 10.3938/jkps.67.843. URL: http://dx.doi.org/10.3938/jkps.67.843.

[5] Marco Ariola y Alfredo Pironti. *Magnetic Control of Tokamak Plasmas*. Springer, oct. de 2010. ISBN: 1849967830. URL: http://books.google.com/books?id=I9PGcQAACAAJ&dq=isbn:1849967830&hl=&source=gbs_api.

[6] Mohammedi R. Abdel-Aziz y Mahmoud M. El-Alem. «Newton-Krylov Type Algorithm for Solving Nonlinear Least Squares Problems». En: *International Journal of Mathematics and Mathematical Sciences* 2009.1 (ene. de 2009). DOI: 10.1155/2009/435851. URL: https://doi.org/10.1155/2009/435851.

[7] D.A. Knoll y D.E. Keyes. «Jacobian-free Newton–Krylov methods: a survey of approaches and applications». En: *Journal of Computational Physics* 193.2 (oct. de 2003), págs. 357-397. DOI: 10.1016/j.jcp.2003.08.010. URL: https://doi.org/10.1016/j.jcp.2003.08.010.

[8] Zenaida Natividad Castillo Marrero et al. «Sobre el uso de métodos de Arnoldi para la continuación numérica de puntos estacionarios». En: *Ciencia Digital* 4.3 (jul. de 2020), págs. 378-390. DOI: 10.33262/cienciadigital.v4i3.1385. URL: https://doi.org/10.33262/cienciadigital.v4i3.1385.

[9] Bartolomé Moreno y Christian Manuel. *Métodos de Krylov para sistemas lineales*. Ene. de 2019. URL: https://riull.ull.es/xmlui/handle/915/15736.

[10] Universidad de Zaragoza y Blanca Sayas Ladaga. *Métodos iterativos en Krylov*. Inf. téc. Jul. de 2024.

[11] Stephen H. Friedberg. *Linear Algebra 4Th Ed.* Ene. de 2003.

[12] N. C. Amorisco et al. «FreeGSNKE: A Python-based dynamic free-boundary toroidal plasma equilibrium solver». En: *Physics of Plasmas* 31.4 (2024), pág. 042517. DOI: 10.1063/5.0188467.