AYUDANTÍA S13.2

GMRES APLICADO EN NEWTON Y SYLVESTER

Ayudante: Francisco Manríquez Novoa

Jueves 5 de junio del 2025

Computación Científica

¿Cuándo usar GMRes?

Ya que GMRes se basa en multiplicar repetidas veces por una matriz *A* en la iteración de Arnoldi, se vuelve útil cuando calcular el producto *Ac* entre *A* y un vector c es la mejor opción.

Por lo general, lo anterior ocurre cuando no es factible calcular *A*.

¿Cuándo usar GMRes?

Ejemplo 1: Método de Newton en $\mathbb{R}^n$

$$\mathbf{x}_{i+1} = \mathbf{x}_i - J^{-1}(\mathbf{x}_i)F(\mathbf{x}_i)$$

Se puede obtener $\mathbf{u} = J^{-1}(\mathbf{x}_i)F(\mathbf{x}_i)$ resolviendo $J(\mathbf{x}_i)\mathbf{u} = F(\mathbf{x}_i)$, pero suele ser infactible obtener el jacobiano J.

Sin embargo, se puede aproximar el producto $J(\mathbf{x}_i)\mathbf{v}$, sin calcular J. Por lo tanto, GMRes es una buena opción.

¿Cuándo usar GMRes?

Ejemplo 2: Ecuación de Sylvester

$$AX + XB = C$$

Se puede expresar el sistema de la forma $M\mathbf{x} = \mathbf{c}$, pero, si las matrices son de tamaño nxn, la matriz M tiene tamaño $n^2 x n^2$. No es factible almacenar M.

Sin embargo, se puede calcular fácilmente el producto $M\mathbf{v}$ con $O(n^3)$ operaciones, sin almacenar M. **Por lo tanto, GMRes es una buena opción.**

¿Cuándo usar GMRes?

Para los anteriores ejemplos, en vez de definir explícitamente la matriz A para calcular el producto Av, se puede definir una función que calcule Av sin definir explícitamente A.

A esta función la llamamos "Afun": función que representa el producto por A.

En los ejemplos anteriores, hay que definir el afun según corresponda y pasárselo al algoritmo GMRes para hacer los cálculos.

GMRes y Newton [1/4]

$$\mathbf{x}_{i+1} = \mathbf{x}_i - J^{-1}(\mathbf{x}_i)F(\mathbf{x}_i)$$

Se puede obtener $\mathbf{u} = J^{-1}(\mathbf{x}_i)F(\mathbf{x}_i)$ resolviendo $J(\mathbf{x}_i)\mathbf{u} = F(\mathbf{x}_i)$, pero suele ser infactible obtener el jacobiano J. Sin embargo, se puede aproximar su producto con un vector $\mathbf{v}$, aproximando una derivada direccional:

$$J(\mathbf{x})\mathbf{v} \approx \frac{F(\mathbf{x} + \epsilon\mathbf{v}) - F(\mathbf{x})}{\epsilon}$$

GMRes y Newton [2/4]

Por tanto, para el sistema

$$J(\mathbf{x}_i)\mathbf{u} = F(\mathbf{x}_i)$$

se puede encontrar soluciones aproximadas para u mediante GMRes.

Partiendo con un vector inicial $q_1 = F(x_i)/|F(x_i)|$, se puede multiplicar repetidamente por $J(x_i)$ en la iteración de Arnoldi para generar vectores q_2, ..., q_n con los cuales aproximar u, sin almacenar *J*. Ejemplo:

$$J(\mathbf{x_i})\mathbf{q}_1 \approx \frac{F(\mathbf{x}_i + \epsilon\mathbf{q}_1) - F(\mathbf{x}_i)}{\epsilon}, \quad \epsilon = 0.001$$

GMRes y Newton [3/4]

Por lo tanto, el afun que se debe definir es la función que representa el producto por la matriz J. Dado un vector v, afun calcula:

$$J(\mathbf{x}_i)\mathbf{v} \approx \frac{\mathbf{F}(\mathbf{x}_i + \varepsilon\mathbf{v}) - \mathbf{F}(\mathbf{x}_i)}{\varepsilon}, \quad \varepsilon = 0.001$$

```
def afun(v):
    e = 0.001
    return (F(xi - e*v) - F(xi)) / e
```

GMRes y Newton [4/4]

Considere el siguiente sistemas de ecuaciones no-lineales:

$$\mathbf{F}_1(\mathbf{x}) = \mathbf{1}, \tag{4}$$

donde $\mathbf{F}_1(\mathbf{x}) : \mathbb{R}^{2n} \to \mathbb{R}^n$, $\mathbf{x} \in \mathbb{R}^{2n}$, y $\mathbf{1}$ corresponde al vector de 1's de dimensión n. Lamentablemente no se puede resolver directamente con el método de Newton en alta dimensión porque tenemos una inconsistencia con el número de ecuaciones disponibles y la cantidad de incógnitas. En particular, tenemos el doble de incógnitas que la cantidad de ecuaciones. Para resolver este problema, se tiene el siguiente sistema de ecuaciones no-lineales complementario,

$$\mathbf{F}_2(\mathbf{x}) = \mathbf{0}, \tag{5}$$

donde $\mathbf{F}_2(\mathbf{x}) : \mathbb{R}^{2n} \to \mathbb{R}^n$, y $\mathbf{0}$ corresponde al vector de 0's de dimensión n. Ahora, considerando al mismo tiempo las ecuaciones (4) y (5), sí tenemos la misma cantidad de incógnitas que de ecuaciones. El paso natural sería aplicar el método de Newton en alta dimensión de la siguiente forma, por simplicidad solo incluiremos un paso,

$$\begin{aligned} \mathbf{x}_0 &= \text{“Initial guess”}. \\ J_{\mathbf{F}}(\mathbf{x}_0)\,\Delta\mathbf{x}_0 &= -\mathbf{F}(\mathbf{x}_0), \\ \mathbf{x}_1 &= \mathbf{x}_0 + \Delta\mathbf{x}_0, \end{aligned}$$

Ecuación de Sylvester [1/2]

$$AX + XB = C$$

Si, por ejemplo, todas las matrices son de 2x2:

$$X = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \qquad C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$$

es posible expresar el sistema de la forma $M\mathbf{x} = \mathbf{c}$:

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \end{pmatrix} = \begin{pmatrix} c_{11} \\ c_{12} \\ c_{21} \\ c_{22} \end{pmatrix}$$

Ecuación de Sylvester [2/2]

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \end{pmatrix} = \begin{pmatrix} c_{11} \\ c_{12} \\ c_{21} \\ c_{22} \end{pmatrix}$$

Sin embargo, construir la matriz *M* puede ser muy ineficiente. Por lo tanto, hay que buscar alternativas.

Para ello, debemos hablar sobre la vectorización, y no la de NumPy.

Vectorización [1/4]

El operador vec toma una matriz y apila sus columnas en un solo vector.

$$\operatorname{vec}\begin{pmatrix} | & & | \\ \mathbf{v}_1 & \cdots & \mathbf{v}_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | \\ \mathbf{v}_1 \\ | \\ \\ \vdots \\ \\ | \\ \mathbf{v}_n \\ | \end{pmatrix}$$

Ejemplo:

$$\operatorname{vec}\begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{21} \\ x_{12} \\ x_{22} \end{pmatrix}$$

Vectorización [2/4]

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \\ x_{21} \\ x_{22} \end{pmatrix} = \begin{pmatrix} c_{11} \\ c_{12} \\ c_{21} \\ c_{22} \end{pmatrix}$$

La ecuación de Sylvester AX = B se puede expresar como Mx = c, aplicando vec a ambos lados:

$$AX + XB = C$$

$$\text{vec}(AX + XB) = \text{vec}(C)$$

$$M\,\text{vec}(X) = \text{vec}(C)$$

$$M\mathbf{x} = \mathbf{c}$$

Vectorización [3/4]

Propiedades:

1. vec(A+B) = vec(A) + vec(B), por lo que el lado izquierdo vec(AX + XB) = vec(AX) + vec(XB).
2. Debido a que

$$AX = \begin{pmatrix} | & & | \\ A\mathbf{x}_1 & \cdots & A\mathbf{x}_n \\ | & & | \end{pmatrix}$$

Vectorización [4/4]

se cumple que

$$\operatorname{vec}(AX) = \operatorname{vec}\begin{pmatrix} | & & | \\ A\mathbf{x}_1 & \cdots & A\mathbf{x}_n \\ | & & | \end{pmatrix} = \begin{pmatrix} | \\ A\mathbf{x}_1 \\ | \\ \vdots \\ | \\ A\mathbf{x}_n \\ | \end{pmatrix} = \begin{pmatrix} A & & \\ & \ddots & \\ & & A \end{pmatrix} \begin{pmatrix} | \\ \mathbf{x}_1 \\ | \\ \vdots \\ | \\ \mathbf{x}_n \\ | \end{pmatrix}$$

Es como haber multiplicado por una matriz diagonal, pero sus coeficientes son otras matrices.

Producto de Kronecker

Lo anterior se puede expresar como un producto de Kronecker. Para entenderlo, puedes “fingir” que la 2° matriz es un escalar, y multiplicar todos los coeficientes de la 1° por ese “escalar”, formando una matriz por bloques:

$$A \otimes B = \begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \ddots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}$$

Producto de Kronecker

Por lo tanto:

$$\begin{pmatrix} A & & \\ & \ddots & \\ & & A \end{pmatrix} = I \otimes A$$

y, por consiguiente:

$$\mathrm{vec}(AX) = (I \otimes A)\mathrm{vec}(X)$$

Producto de Kronecker

Por otro lado:

$$\text{vec}(XB) = (B^T \otimes I)\text{vec}(X)$$

Por lo tanto:

$$\text{vec}(AX + XB) = ((I \otimes A) + (B^T \otimes I))\text{vec}(X)$$

PALTA: solo recuerda este producto. Para encontrar vec(AX), reemplaza B = I. Para encontrar vec(XB), reemplaza A = I.

$$\text{vec}(AXB) = (B^T \otimes A)\text{vec}(X)$$

Producto de Kronecker

Así, la ecuación AX + BX se puede escribir como Mx = c, donde

$$M = (I \otimes A) + (B^T \otimes I)$$

$$\mathbf{x} = \mathrm{vec}(X)$$

$$\mathbf{c} = \mathrm{vec}(C)$$

El producto Mv [1/3]

La matriz M es muy costosa de almacenar, pero se puede calcular el producto $M\mathbf{v}$ sin almacenar M.

Recordemos que $M\mathbf{x} = \text{vec}(AX + XB)$.
Si $\mathbf{v} = \text{vec}(V)$, entonces $M\mathbf{v} = \text{vec}(AV + VB)$.

Pasos para calcular Mv en $O(n^3)$ operaciones:

1. "Desvectorizar" v. Se debe convertir en una matriz cuadrada V tal que $\mathbf{v} = \text{vec}(V)$.
2. Calcular $AV + VB$.
3. Vectorizar $AV + VB$ para obtener $M\mathbf{v}$.

El producto Mv [2/3]

Por lo tanto, para resolver Mx = c, es conveniente usar GMRes.

Partiendo de un $\mathbf{q}_1 = \mathbf{c}/|\mathbf{c}|$, se realiza la iteración de Arnoldi, multiplicando repetidamente por M, es decir, calculando productos $M\mathbf{v}$ de la manera eficiente descrita anteriormente, para generar vectores $\mathbf{q}_2, \ldots, \mathbf{q}_n$ con los cuales aproximar $\mathbf{u}$, sin almacenar M.

Luego de obtener x, se debe "desvectorizar" para obtener la matriz X que soluciona $AX + XB = C$.

El producto Mv [3/3]

El afun necesario para GMRes sería:

```
def afun(v: np.ndarray):
    # Como la vectorización es por columnas y NumPy ordena valores
    # consecutivos por filas, hay que obtener traspuestas de matrices.

    # Desvectorización (requiere trasponer)
    V = v.reshape((n, n)).T

    # Cálculo de matriz C = AV + VB
    C = A @ V + V @ B

    # Vectorización (requiere trasponer) para obtener c = Mv.
    Mv = C.T.reshape(n*n)

    return Mv
```

Ejercicio de Sylvester

2. Sea $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{C}^{n\times n}$, $C \in \mathbb{C}^{n\times n}$, y $Z \in \mathbb{C}^{n\times n}$. Considere la siguiente variante de la ecuación de Sylvester:

$$A\,Z + \mathrm{Conj}(Z)\,B = C \tag{1}$$

donde A es simétrica y definida positiva, B es Hermitiana, C es una matriz no nula, y $\mathrm{Conj}(\cdot)$ corresponde al operador de conjugación, i.e. si $z = x + \mathrm{i}\,y$ donde x e y son números reales y $\mathrm{i}^2 = -1$, entonces $\mathrm{Conj}(z) = x - \mathrm{i}\,y$ y en el caso matricial se aplica elemento a elemento. Para estandarizar la notación, se sugiere considerar $Z = X + \mathrm{i}\,Y$, $B = B_1 + \mathrm{i}\,B_2$, y $C = C_1 + \mathrm{i}\,C_2$, donde X, Y, B_1, B_2, C_1, y C_2 pertenecen a $\mathbb{R}^{n\times n}$. El desafío es proponer un algoritmo que solo dependa de aritmética real para encontrar Z, el cual se define al obtener X e Y, es decir, es suficiente obtener X e Y. **Recordar que no está permitido obtener la inversa explícita de ninguna de las matrices involucradas, lo que se debe hacer es resolver el sistema de ecuaciones lineales asociado**.

(a) **[30 puntos]** Dada la restricción de que solo se puede utilizar aritmética real, re-escriba la ecuación (1) para que solo dependa de aritmética real. *Hint: You should get a linear system of equations where the unknowns are the matrices X and Y.*

¿Dudas?