Métodos Numéricos

Métodos Iterativos para Sistemas de Ecuaciones Lineales y Método de las Potencias

Tarea 5

September 21, 2020

Erika Rivadeneira Pérez
erika.rivadeneira@cimat.mx
Matemáticas Aplicadas - CIMAT

1 Resumen

En el presente reporte se exponen dos métodos iterativos para resolver sistemas de ecuaciones lineales los cuales son: método de Jacobi y Gauss-Seidel. Además, se presenta el método iterativo de las potencias, específicamente se presentan dos versiones de esta herramienta. La primera encuentra el autovalor más grande de una matriz con su respectivo autovector asociado y la segunda versión encuentra varios de los autovalores más grandes con sus respectivos autovectores de una matriz. Finalmente se realiza un análisis de comparación de los métodos utilizados.

2 Introducción

Un método iterativo es un método que progresivamente va calculando aproximaciones a la solución de un problema. El proceso se repite sobre esta nueva solución hasta que el resultado más reciente satisfaga ciertos requisitos. A diferencia de los métodos directos, en los cuales se debe terminar el proceso para tener la respuesta, en los métodos iterativos se puede suspender el proceso al termino de una iteración y se obtiene una aproximación a la solución.

Las técnicas iterativas rara vez se utilizan para resolver sistemas lineales de pequeñas dimensiones, ya que el tiempo requerido para una precisión suficiente

excede el requerido para técnicas directas, como la eliminación gaussiana. Sin embargo, para sistemas grandes con un alto porcentaje de 0 entradas, estas técnicas son eficientes tanto en términos de almacenamiento como de cálculo [1].

En la siguiente sección se describen dos métodos iterativos para resolver sistemas de ecuaciones, Jacobi y Gauss-Seidel, también se describe el método de la potencia para encontrar el autovalor más grande en magnitud de una matriz y su generalización para encontrar varios autovalores deseados con esta característica.

3 Metodología

3.1 Métodos iterativos para sistemas de ecuaciones lineales

Si las entradas diagonales de A son distintas de cero, podemos señalar en cada ecuación la incógnita correspondiente, obteniendo el sistema lineal equivalente

$$x_i = \frac{1}{a_{ii}} \left[b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j \right], \quad i = 1, \ldots, n \tag{1}$$

3.1.1 Método de Jacobi

El método de Jacobi puede ser escrito de la siguiente manera $\mathbf{x}^{(k)} = T\mathbf{x}^{(k-1)} + \mathbf{c}$ dividiendo A en sus partes diagonales y fuera de la diagonal. Para ver esto, sea D la matriz diagonal cuyas entradas diagonales son las de $A, -L$ la parte estrictamente triangular inferior de A, y $-U$ la parte estrictamente triangular superior de A. Con esta notación,

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}$$

es dividida en

$$A = \begin{bmatrix} a_{11} & 0 & \ldots & 0 \\ 0 & a_{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & a_{nn} \end{bmatrix} - \begin{bmatrix} 0 & \ldots & \ldots & 0 \\ -a_{21} & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ -a_{n1} & \ldots & -a_{n,n-1} & 0 \end{bmatrix} - \begin{bmatrix} 0 & -a_{12} & \ldots & -a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & -a_{n-1,n} \\ 0 & \ldots & \ldots & 0 \end{bmatrix}$$

$$= D - L - U$$

La ecuación $A\mathbf{x} = \mathbf{b}$, o $(D - L - U)\mathbf{x} = \mathbf{b}$, es entonces transformada en

$$D\mathbf{x} = (L + U)\mathbf{x} + \mathbf{b}$$

y, si D^{-1} existe, esto es, si $a_{ii} \neq 0$ para cada i, entonces

$$\mathbf{x} = D^{-1}(L + U)\mathbf{x} + D^{-1}\mathbf{b}$$

Esto da como resultado la forma matricial de la técnica iterativa de Jacobi [2]:

$$\mathbf{x}^{(k)} = D^{-1}(L + U)\mathbf{x}^{(k-1)} + D^{-1}\mathbf{b}, \quad k = 1, 2, \ldots$$

Introduciendo la notación $T_j = D^{-1}(L + U)$ y $\mathbf{c}_j = D^{-1}\mathbf{b}$ da el método de Jacobi de la forma

$$\mathbf{x}^{(k)} = T_j\mathbf{x}^{(k-1)} + \mathbf{c}_j$$

3.1.2 Método de Gauss-Seidel

Los componentes de $\mathbf{x}^{(k-1)}$ se utilizan para calcular todos los componentes $x_i^{(k)}$ de $\mathbf{x}^{(k)}$. Pero, para $i > 1$ los componentes $x_1^{(k)}, \ldots, x_{i-1}^{(k)}$ de $\mathbf{x}^{(k)}$ ya se han calculado y se espera que sean mejores aproximaciones a las soluciones reales $x_1, \ldots, x_{i-1}$ que las $x_1^{(k-1)}, \ldots, x_{i-1}^{(k-1)}$. Parece razonable, entonces, calcular $x_i^{(k)}$ utilizando estos valores calculados más recientemente. Es decir, usar

$$x_i^{(k)} = \frac{1}{a_{ii}}\left[-\sum_{j=1}^{i-1}\left(a_{ij}x_j^{(k)}\right) - \sum_{j=i+1}^{n}\left(a_{ij}x_j^{(k-1)}\right) + b_i\right] \tag{2}$$

para cada $i = 1, 2, \ldots, n$. Esta modificación se denomina técnica iterativa de Gauss-Seidel [2].
Para escribir el método de Gauss-Seidel en forma de matriz, multiplique ambos lados de ecuación (2) por a_{ii} y recolecta los k términos iterados para dar como resultado

$$a_{i1}x_1^{(d)} + a_{i2}x_2^{(i)} + \cdots + a_{ij}x_i^{(i)} = -a_{i,i+1}x_{i+1}^{(i-1)} - \cdots - a_{in}x_n^{(k-1)} + b_i$$

para cada $i = 1, 2, \ldots, n$. Escribiendo todas las n ecuaciones nos da

$$\begin{aligned}
a_{11}x_1^{(k)} &= -a_{12}x_2^{(k-1)} - a_{13}x_3^{(k-1)} - \cdots - a_{1n}x_n^{(k-1)} + b_1 \\
a_{21}x_1^{(k)} + a_{22}x_2^{(k)} &= a_{23}x_3^{(k-1)} - \cdots - a_{2n}x_n^{(k-1)} + b_2 \\
\vdots &= \vdots \\
a_{n1}x_1^{(k)} + a_{n2}x_2^{(k)} + \cdots + a_{nn}x_n^{(k)} &= b_n;
\end{aligned}$$

con las definiciones de D, L, y U dadas previamente, se obtiene el método de Gauss-Seidel representado por

$$(D - L)\mathbf{x}^{(i)} = U\mathbf{x}^{(1-1)} + \mathbf{b}$$

y

$$\mathbf{x}^{(i)} = (D-L)^{-1}U\mathbf{x}^{(l-1)} + (D-L)^{-1}\mathbf{b}, \quad \text{para cada } k = 1, 2, \ldots$$

Haciendo $T_g = (D-L)^{-1}U$ y $\mathbf{c}_g = (D-L)^{-1}\mathbf{b}$, da el método de Gauss-Seidel de la forma

$$\mathbf{x}^{(k)} = T_g\mathbf{x}^{(k-1)} + \mathbf{c}_g$$

Para que la matriz triangular inferior $D-L$ no sea singular, es necesario y suficiente que $a_{ii} \neq 0$, para cada $i = 1, 2, \ldots, n$.

3.2 Método de la Potencia

El método conocido como el método de la potencia es utilizado para calcular el valor propio dominante, es decir el valor propio de mayor módulo de una matriz cuadrada A. Sea A una matriz cuadrada de orden n y la cual tiene n vectores propios linealmente independientes (es decir, A es diagonalizable) y un valor propio estrictamente dominante, es decir, si sus valores propios son

$$\lambda_1, \lambda_2, \ldots, \lambda_n$$

Entonces ocurre que:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$$

No hay pérdida de generalidad en suponerlos ordenados, de forma que el de mayor módulo corresponde a λ_1
Sea v_i al vector propio asociado a cada valor propio λ_i para $i = 1, 2, \ldots, n$. Entonces por la hipótesis de la que hemos partido tenemos que: $\{v_1, v_2, \ldots, v_n\}$ son linealmente independientes. Esto significa que cualquier vector v del espacio vectorial $\mathbb{R}^n$ se puede expresar como combinación lineal de los vectores $\{v_1, v_2, \ldots, v_n\}$. Es decir existirán valores reales $a_i \in \mathbb{R}$, tales que:

$$v = \sum_{i=1}^{n} a_i v_i$$

Por lo tanto,

$$Av = \sum_{i=1}^{n} a_i A v_i = \sum_{i=1}^{n} a_i \lambda_i v_i$$

Es decir,

$$Av = a_1\lambda_1 v_1 + a_2\lambda_2 v_2 + \cdots + a_n\lambda_n v_n$$

Si multiplicamos esta expresión de nuevo por A, se tiene que

$$A^2 v = a_1\lambda_1 A v_1 + a_2\lambda_2 A v_2 + \cdots + a_n\lambda_n A v_n$$

así

$$A^2 v = a_1\lambda_1^2 v_1 + a_2\lambda_2^2 v_2 + \cdots + a_n\lambda_n^2 v_n$$

Si continuamos multiplicando por A, de forma sucesiva se tiene que

$$A^k v = a_1 \lambda_1^k v_1 + a_2 \lambda_2^k v_2 + \cdots + a_n \lambda_n^k v_n$$

Como el valor propio λ_1 es el de mayor módulo, sacando factor común λ_1^k en la expresión anterior

$$A^k v = \lambda_1^k \left[a_1 v_1 + a_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k v_2 + \cdots + a_n \left(\frac{\lambda_2}{\lambda_1} \right)^k v_n \right],$$

tomando límites cuando $k \to \infty$, y considerando que al ser λ_1 el valor propio dominante entonces

$$\lim_{k \to \infty} \left(\frac{\lambda_i}{\lambda_1} \right)^k = 0 \qquad i = 2, 3, \ldots, n,$$

en consecuencia se tiene que

$$\lim_{k \to \infty} \left[a_1 v_1 + a_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k v_2 + \cdots + a_n \left(\frac{\lambda_2}{\lambda_1} \right)^k v_n \right] = a_1 v_1$$

Para valores "grandes" de k

$$A^k v \approx \lambda_1^k a_1 v_1$$

es decir la sucesión $\left\{ A^k v \right\}_{k \in \mathbb{N}}$ es una sucesión de valores que cumple que la sucesión

$$\left\{ \lambda_1^{-k} A^k v \right\}_{k \in \mathbb{N}} \text{ converge a } a_1 v_1$$

Por lo tanto,

$$\lim_{k \to \infty} \frac{\left(A^{k+1} \right)_1}{\left(A^k v \right)_1} = \lambda_1$$

Además, existen métodos conocidos como **técnicas de deflación** para obtener aproximaciones a los otros valores propios de una matriz una vez que se ha calculado una aproximación al valor propio dominante. Las técnicas de deflación implican formar una nueva matriz B cuyos valores propios son los mismos que los de A, excepto que el valor propio dominante de A se reemplaza por el valor propio 0 en B.

4 Implementación

A continuación se presentan los pseudo-códigos de los métodos presentados en la sección anterior. Los algoritmos (2) y (1) requieren como datos de entrada las matrices A y b del sistema de ecuaciones, el número máximo de iteraciones y la tolerancia que el usuario desee.

Algorithm 1: Algortimo para la Iteración Jacobi

```
Result: iteración, xn/* Número de iteraciones y vector
    solución                                                    */
n,m=shape(A);/* extraemos la dimensión de la matriz             */
error = 1 /* Inicializamos error                               */
cont = 0 /* Inicializamos contador                             */
xn = np.zeros(n)/* Inicializamos vector nuevo                   */
xv = np.zeros(n)/* Inicializamos vector viejo                   */
while error < tolerancia and cont < max − iter do
    con+=1;
    suma=0;
    for i in (1:n) do
        for j in (1:m) do
            if j!=i then
                suma+=A[i,j]*xv[j]
            end
        end
        xn[i]=(b[i]-suma)/A[i,i]
    end
    xv=xn;
    error = np.linalg.norm(np.dot(A,x)-np.transpose(b))
end
```

Algorithm 2: Algortimo para la Iteración Gauss-Seidel

```
Result: iteración, xj/* Número de iteraciones y vector solución
    */
n,m=shape(A);/* extraemos la dimensión de la matriz             */
error = 1 /* Inicializamos error                               */
cont = 0 /* Inicializamos contador                             */
xj = np.zeros(n);
while error < tolerancia and cont < max − iter do
    con+=1;
    suma=0;
    for i in (1:n) do
        for j in (1:m) do
            if j!=i then
                suma+=A[i,j]*x[j]
            end
        end
        xj[i]=(b[i]-suma)/A[i,i]
    end
    error = np.linalg.norm(np.dot(A,x)-np.transpose(b))
end
```

A continuación se pueden observar los algoritmos para calcular uno o varios

autovalores más grandes en módulo de una matriz A en (3) y (4) respectivamente, donde los input son la matriz A de la cual se desea obtener el/los autovalores, tolerancia y número máximo de iteraciones. Por otro lado, para el algoritmo (4) también tiene un argumento de entrada r que son el número de autovalores más grandes que se desea obtener.

Algorithm 3: Algortimo de Método de la Potencia

```
Result: lambda, iteracion/* Autovalor más grande y número de
    iteraciones                                              */
n,m=shape(A);/* extraemos la dimensión de la matriz          */
num=0;
error = 1 /* Inicializamos error                            */
cont = 0 /* Inicializamos contador                         */
vo = np.ones(n);
while error < tolerancia and cont < max − iter do
    vo = norma(vo);
    v1 = A * vo/* Producto matricial                        */
    for i in (1:n) do
        num +=v1[i]*vo[i];
        vo[i]=v1[i]
    end
    error = abs(lambda-num);
    lambda=num
end
```

Algorithm 4: Algortimo de Método de la Potencia con Deflación

```
Result: w, v/* Vector de autovalores más grandes y matriz de
    autovectores asociados                                    */
n,m=shape(A);/* extraemos la dimensión de la matriz           */
num=0;
w = []/* Inicializamos matriz para guardar autovalores       */
v = []/* Inicializamos matriz para guardar autovectores      */
error = 1 /* Inicializamos error                             */
cont = 0 /* Inicializamos contador                          */
vo = np.ones(n);
for i in (1:r) do
    suma=0;
    while error < tolerancia and cont < max − iter do
        ai=transpose(vo)*vo;
        suma = -ai*vo;
        vo = vo + suma;
        vo = norma(vo);
        v1 = A * vo/* Producto matricial                      */
        for i in (1:n) do
            num +=v1[i]*vo[i];
            vo[i]=v1[i]
        end
        error = abs(lambda-num);
        lambda=num;
        w = w.append(lambda);
        v = v.append(v1)
    end
end
```

5 Resultados

Se implementaron los métodos de Jacobi y Gauss-Seidel en las matrices

$$A_{3\times 3} = \begin{bmatrix} 3 & -0.1 & -0.2 \\ 0.1 & 7 & -0.3 \\ 0.3 & -0.2 & 10 \end{bmatrix}; \quad b_{3\times 1} = \begin{bmatrix} 7.85 \\ -19.3 \\ 71.4 \end{bmatrix}$$

$$A_{125\times 125} = \begin{bmatrix} 9.566296520595209e-05 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0.0003522280923904741 \end{bmatrix};$$

$$b_{125\times 1} = \begin{bmatrix} 1.30884747e-06 \\ \vdots \\ 0 \end{bmatrix}$$

Las soluciones obtenidas de los sistemas $Ax = b$ ocupando métodos iterativos se muestran en la tabla (1), en donde **G-S** denota el método de Gauss-Seidel, **Iter** denota el número de iteraciones realizadas para obtener el vector solución y **Tol** la tolerancia impuesta por el usuario, en este caso se usó una tolerancia de $1e-7$. El número máximo de iteraciones propuesto en todos los casos fue de 1000.

Método	**Sist. de Ecuaciones**	**Solución**	**Error**	**Tol.**	**Iter.**
G-S	$A_{3\times3}x = b_{3\times1}$	[3, -2.5, 7]'	6.722e-09	1e-7	5
Jacobi		[3,-2.5,7]'	2.698e-08	1e-7	8
G-S	$A_{125\times125}x = b_{125\times1}$	[0.59839197,...,0.46134888]'	9.99e-08	1e-7	340
Jacobi		[0.59731758,...,0.46042692]'	9.944e-08	1e-7	656

Table 1: Resultados y comparación de métodos iterativos para resolver sistemas de ecuaciones

Además, se encontraron autovalores con sus respectivos autovectores de las matrices

$$B_{3\times3} = \begin{bmatrix} 5 & -1.778 & 0 \\ -1.778 & 9 & -1.778 \\ 0 & -1.778 & 10 \end{bmatrix};$$

$$B_{125\times125} = \begin{bmatrix} 9.566296520595209e-05 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0.0003522280923904741 \end{bmatrix}$$

Utilizando el método de la potencia se obtuvieron los autovalores más grandes en valor absoluto y sus respectivos autovectores mostrados a continuación en la tabla (2). Estos resultados se obtuvieron tomando como 1000 el número máximo de iteraciones. Nuevamente, **Iter.** representa el número de iteraciones realizadas para encontrar el autovalor y **Tol.** es la tolerancia impuesta.

Método	**Matriz**	**Autovalor**	**Autovector**	**Error**	**Tol.**	**Iter.**
Potencia	$B_{3\times3}$	11.53840278	$[0.23528713, -0.86524375, 1]'$	$8.64e-07$	$1e-6$	45
	$B_{125\times125}$	1	$[-3.23e-12, ..., -1.72e-11]'$	0.0	$1e-6$	3

Table 2: Resultados del Método de la Potencia

Por otro lado, en la tabla (3) se pueden observar los autovalores más grandes de las matrices $B_{3\times3}$ y $B_{125\times125}$. Para la matriz 3×3 se obtuvieron los 2 autovalores más grandes y para la matriz con dimensión 125×125 se obtuvieron sus 121 autovalores más grandes junto con sus respectivos autovectores.

Método	Matriz	Autovalores max{2,n-4}	Autovector	Error	Tol.	Iter.
Potencia con deflación	$A_{3\times3}$	1°: 11.538404914009085	[0.23528713, -0.86524375, 1]'	8.134e-08	1e-7	46
		2°: 8.213809393238977	[-0.36503844, 0.65982236, 0.6567963]'	8.095e-08	1e-7	1
	$A_{125\times125}$	1°: 1	[-3.23e-12,...,-1.72e-11]	1.128e-10	1e-7	4
		121°: 0.00433311147340	[-4.25e-12,...,-3.24-12]	2.348e-8	1e-7	423

Table 3: Resultados del Método de la Potencia con Deflación

6 Discusión

Como se puede obserar en la tabla (1) para el caso de la matriz $A_{3\times3}$ el método de Gauss-Seidel converge en 5 iteraciones mientras que Jacobi convergió en 8 iteraciones. Esta diferencia se observa mejor para el caso de la matriz $A_{125\times125}$ ya que con el método de Jacobi se obtuvo convergencia en casi el doble de iteraciones que con Gauss-Seidel. En cuanto a los errores ambos métodos mostraron tener un error similiar de aproximadamente $9e-08$ utilizando una tolerancia de $1e-7$ en ambos casos.

Por otro lado, utilizando el método de las potencias se obtuvo el autovalor más grande y sus autovectores para las matrices $B_{3\times3}$ y $B_{125\times125}$ con un error aproximado de $8.64e07$ y con pocas iteraciones. Es importante mencionar que el método de la potencia con deflación tiene un gran costo computacional debido a que cada que se obtiene un autovalor debe repetirse el ciclo del método de la potencia excluyendo las contribuciones de los autovectores encontrados previamente. Igualmente, si se restrige el método a una tolerancia aún menor y aumentando el número máximo de iteraciones entonces el método demorará mucho más en converger.

7 Conclusiones

Las técnicas iterativas como Jacobi y Gauss-Seidel para resolver sistemas lineales garantizan una precisión suficiente en la solución requerida por el usuario. Estos métodos funcionan muy bien para sistemas grandes. Además, son eficientes tanto en términos de almacenamiento como de cálculo. En matrices de cualquier dimensión el método de Gauss-Seidel converge más rápido que Jacobi, ambos teniendo una precisión similar. Finalmente, el método de la potencia es de gran ayuda cuando se requiere conocer un autovalor con mayor magnitud para matrices de cualquier dimensión ya que tiene una gran presición y converge rápidamente. Por otro lado, al encontrar varios autovalores se requiere de la

repetición del método de la potencia en un ciclo, dependiendo del número de valores que se desee encontrar y esto genera un gran costo computacional pero se garantiza el encontrar todos los autovalores deseados incluso los que tienen multiplicidad.

References

[1] A. Quarteroni, P. Quarteroni, F. Riccardo Sacco, R. Sacco, and F. Saleri, Numerical Mathematics, ser. Texts in Applied Mathematics. Springer, 2007.

[2] R.L. Burden and J.D. Faires, Análisis Numérico, Grupo Editorial Iberoamérica, México 1985.