Universidad Nacional de Colombia
Facultad de Ciencias
Análisis Numérico I

Mateo Andrés Manosalva Amaris
Edgar Santiago Ochoa Quiroga
Sergio Alejandro Bello Torres

Ejercicio 1

Sea

$$A = \begin{bmatrix} 0 & -20 & 14 \\ -3 & 27 & 4 \\ -4 & 11 & 2 \end{bmatrix}$$

a) Aplique las transformaciones de Householder, para calcular $A = QR$.

b) Aplique el método de ortogonalización de Gram-Schmidt, para calcular $A = QR$.

c) Implemente en Matlab los métodos de ortogonalización de Gram-Schmidt y Householder, para calcular $A = QR$, compare los resultados numéricos con los encontrados en las partes (a) y (b).

Ejercicio 2

Descargue el archivo `Datos.txt` de la página del curso. En este encontrará un conjunto de 21 datos. Copie estos datos y calcule el polinomio de ajuste de grado 5

$$p(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4 + c_5 x^5$$

utilizando los métodos de ecuaciones normales y factorización QR. Compare sus resultados con los valores certificados $c_i = 1$ para $i = 0, 1, \ldots, 5$. Encuentre el residual $\|Ac - y\|_2$ en cada caso, así como la diferencia relativa con respecto a los valores certificados. Escriba sus conclusiones.

Solución.

Ejercicio 3

Sean

$$A = \begin{bmatrix} a & 0 & 0 \\ a\delta & a & 0 \\ 0 & a\delta & a \end{bmatrix}, \quad a < 0, \delta > 0, \quad b = \begin{bmatrix} -1 \\ -1{,}1 \\ 0 \end{bmatrix}$$

a) Obtenga el número de condición de A.

b) Estudie el condicionamiento del sistema $Ax = b$ en función de los valores de δ. Interprete su resultado.

c) Si $a = -1$, $\delta = 0{,}1$ y se considera $x^* = (1, 9/10, 1)^T$ como solución aproximada del sistema $Ax = b$ (sin obtener la solución exacta), determine un intervalo en el que esté comprendido el error relativo. ¿Es coherente con la respuesta dada en el apartado anterior?

d) Si $a = -1$ y $\delta = 0{,}1$, ¿es convergente el método de Jacobi aplicado a la resolución del sistema $Ax = b$? Realice tres iteraciones a partir de $x_0 = (0, 0, 0)^T$.

Ejercicio 4

Sea $A \in \mathbb{R}^{n \times n}$. Probar que $\lambda = 1$ es un valor propio de la matriz de iteración del método de Jacobi (o Gauss-Seidel) de A si y solo si A no es invertible.

Ejercicio 5

Considere el sistema $Ax = b$ donde

$$A = \begin{bmatrix} 3 & -1 & -1 & 0 & 0 \\ -1 & 4 & 0 & -2 & 0 \\ -1 & 0 & 3 & -1 & 0 \\ 0 & -2 & -1 & 5 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ -26 \\ 3 \\ 47 \\ -10 \end{bmatrix}$$

a) Investigue la convergencia de los métodos de Jacobi, Gauss-Seidel y sobrerelajación.

Solución. Para determinar la convergencia del método de Jacobi basta observar que la matriz A es estrictamente diagonal-dominante por filas, por lo tanto el método de Jacobi converge. Al programar el método y con una tolerancia de 0.001 respecto al error en $\|\cdot\|_\infty$ se observa que el método se detiene luego de 28 iteraciones y se obtiene un error de aproximadamente $7{,}7359 \times 10^{-4}$.

Veamos ahora que A es simétrica definida positiva, pues así garantizamos la convergencia del método de sobrerelajación para $\omega \in (0, 2)$ y por lo tanto del método de Gauss - Seidel. Procederemos a calcular $x^T Ax$ para un vector no nulo arbitrario:

$$
\begin{aligned}
x^T Ax &= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix} \begin{bmatrix} 3 & -1 & -1 & 0 & 0 \\ -1 & 4 & 0 & -2 & 0 \\ -1 & 0 & 3 & -1 & 0 \\ 0 & -2 & -1 & 5 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \\
&= \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix} \begin{bmatrix} 3x_1 - x_2 - x_3 \\ -x_1 + 4x_2 - 2x_4 \\ -x_1 + 3x_3 - x_4 \\ -2x_2 - x_3 + 5x_4 - x_5 \\ -x_4 + 2x_5 \end{bmatrix} \\
&= 3x_1^2 + 4x_2^2 + 3x_3^2 + 5x_4^2 + 2x_5^2 - 2x_1x_2 - 2x_1x_3 - 2x_1x_4 - 2x_2x_4 - 2x_3x_4 - 2x_4x_5 \\
&= (x_1 - x_2)^2 + (x_1 - x_3)^2 + (x_1 - x_4)^2 + (x_2 - x_4)^2 + (x_3 - x_4)^2 + (x_4 - x_5)^2 + 2x_2^2 + x_3^2 + x_4^2 + x_5^2 \\
&> 0
\end{aligned}
$$

Con esto vemos que $x^T Ax > 0$ y por lo tanto A es definida positiva, así, los métodos de sobrerelajación con $\omega \in (0,2)$ y de Gauss-Seidel convergen. Al programar el método de Gauss-Seidel y fijar la misma tolerancia que con el método de Jacobi, éste se detiene luego de 14 iteraciones y un error de aproximadamente $6{,}8814 \times 10^{-4}$, con lo cual notamos que el método converge significativamente más rápido que el de Jacobi. Ahora, si programamos el método de sobrerelajación tomando $\omega = 1{,}18$ usando las mismas condiciones de parada anteriores, éste se detiene luego de 9 iteraciones, por tanto podemos decir que de los tres métodos, es el que converge más rápido.

b) ¿Cuál es el radio espectral de la matriz J y de la matriz S?

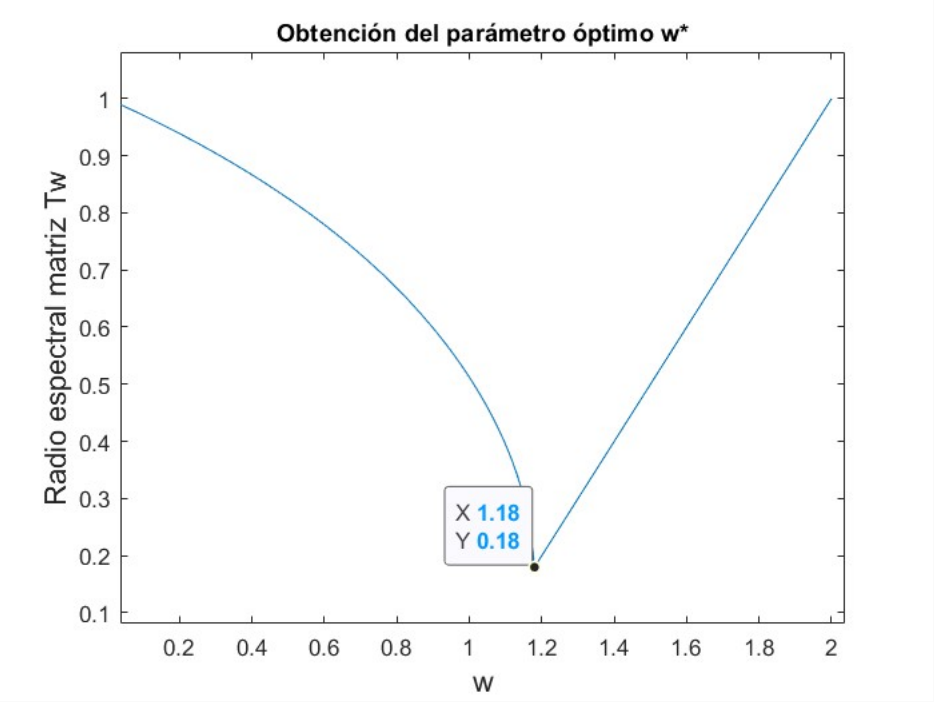
Solución. Para esto primero calculamos las matrices T_J y $T_G S$ en Matlab de acuerdo a las fórmulas $T_J = D^{-1}(U+L)$ y $T_G S = (D-L)^{-1}U$ y luego calculamos el radio espectral usando el siguiente código

```
1    T_J=inv(D)*(L+U);
2    T_GS=inv(D-L)*U;
3    p_J=max(abs(eig(T_J))); %radio espectral de T_J
4    p_GS = max(abs(eig(T_GS))); %radio espectral de T_GS
```

así obtuvimos que $\rho(T_J) = 0{,}7158$ y $\rho(T_G S) = 0{,}5123$, observe que $\rho(T_J) < \rho(T_G S)$, lo cual es esperable pues el radio espectral determina la velocidad de convergencia de cada método.

c) Aproxime con dos cifras decimales el parámetro de sobrerelajación ω^*.

Solución. Teniendo en cuenta que el radio espectral determina la velocidad de convergencia, podemos calcular el radio espectral de la matriz de iteración, variando ω entre 0 y 2, con pasos de 0,01 para obtener la precisión deseada, luego escogemos el ω que minimice el radio espectral de la matriz T_ω. En este caso obtuvimos que el parámetro ω^* es aproximadamente 1.18, como se observa en la gráfica:



d) ¿Qué reducción en el costo operacional ofrece el método de sobrerelajación con el parámetro ω^*, en comparación con el método de Gauss-Seidel?

Solución. La reducción en el costo operacional puede interpretarse como la diferencia en el número de iteraciones de cada método, pues en general cada iteración tiene el mismo número de operaciones, sin tener en cuenta el cálculo de las matrices de iteración, pues éste se puede realizar antes de implementar cada método. En ese caso, programamos ambos métodos variando la tolerancia al error, de tal manera que se pudiera evidenciar la diferencia entre las iteraciones de cada método. De esta forma obtenemos la siguiente tabla:

Tolerancia	1	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-7}	10^{-8}	10^{-9}	10^{-10}
Sobrerelajación	4	6	7	9	10	11	12	14	15	16	17
Gauss-Seidel	4	8	11	15	18	22	25	28	32	35	39

Aquí observamos que a medida que la tolerancia se hace más pequeña, la relación entre la cantidad de iteraciones de cada método es aproximadamente 0.43, lo cual significa que con el metodo de Sobrerelajación reducimos la cantidad de operaciones en un 56% con respecto al método de Gauss-Seidel.

e) ¿Cuántas iteraciones más requiere el método de Gauss-Seidel para lograr una precisión mejorada en una cifra decimal? ¿Cuántas necesita el método de sobrerelajación con ω^*?

Solución. Si observamos nuevamente la tabla del inciso anterior, variamos la tolerancia justamente para que cada vez se mejore la precisión en una cifra decimal, y notamos que para el método de Gauss-Seidel se necesitan al rededor de 3 o 4 iteraciones más para lograr esto. En cambio, para lograr esta misma mejoría en la precisión con el método de

sobrerrelajación, basta iterar 1 o 2 veces más, lo cual también indica que éste método converge mucho más rápido que el anterior.

Ejercicio 6

El operador Laplaciano en 2D se define como:

$$\nabla^2 u(x,y) = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

donde $u(x,y)$ es una función que representa la intensidad de los pixeles de la imagen en el dominio espacial. El operador Laplaciano se utiliza para detectar bordes porque responde a las variaciones de la intensidad de los pixeles en la vecindad de cada punto. En áreas de la imagen donde la intensidad varía rápidamente (bordes), el Laplaciano tiene un valor alto, mientras que en áreas homogéneas (sin bordes) el Laplaciano es cercano a cero. El proceso de detección de bordes implica calcular el Laplaciano de la imagen $u(x,y)$, lo que da como resultado un mapa de bordes.
En este ejercicio estamos interesados en reconstruir la imagen original $u(x,y)$ a partir de los bordes detectados. Para esto debemos resolver la ecuación de Poisson en 2D

$$\nabla^2 u(x,y) = f(x,y)$$

donde $f(x,y)$ es el mapa de bordes. El problema (1) puede ser discretizado en un sistema de ecuaciones lineales de la forma:

$$Au = f$$

donde A es una matriz que representa el operador Laplaciano discreto, u es el vector que contiene los valores de la imagen original en cada pixel (a reconstruir), y f es el vector que contiene los valores de los bordes detectados.

Solución. Lo primero que debemos hacer es discretizar el laplaciano, para esto usamos el teorema de Taylor. A saber

$$u(x+h,y) = u(x,y) + hu_x(x,y) + \frac{h^2\partial_x^2 u(x,y)}{2} + O(h^3),$$
$$u(x-h,y) = u(x,y) - hu_x(x,y) + \frac{h^2\partial_x^2 u(x,y)}{2} + O(h^3),$$

realizando esto para $u(x,y+h)$ y $u(x,y-h)$ obtenemos que

$$\Delta u(x,y) \approx \frac{u(x-h,y) + u(x+h,y) + u(x,y-h) + u(x,y+h) - 4u(x,y)}{h^2}.$$

Para reconstruir la imágen debemos encontrar $u(x,y)$, la función que nos da la intensidad de cada pixel, esto es resolver la ecuación de Poisson $\Delta u(x,y) = f(x,y)$ donde $f(x,y)$ es el mapa de bordes. Primero consideremos para el problema una malla regular, tomaremos $h = 1$ ya que cada pixel representa un paso, en una dimensión la matriz que obtenemos es tridiagonal, sin embargo al discretizar la ecuación de Poisson en 2D debemos mover f en coordenadas x y y, por lo tanto debemos analizarlo con detalle.

Como usamos usa malla regular de $n \times n$ (porque el problema es 2D), cada fila de la matriz sería de $n \times n$, obteniendo

$$\Delta u_{ij} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij} = f_{ij}$$

donde $2 \le i \le m-1$ y $2 \le j \le n-1$. Esto da lugar a un sistema lineal de dimensión $n^2 \times n^2$, $Au = f$, donde

$$A = \begin{bmatrix} D & I & 0 & 0 & 0 & \cdots & 0 \\ I & D & I & 0 & 0 & \cdots & 0 \\ 0 & I & D & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & D & I & 0 \\ 0 & \cdots & \cdots & 0 & I & D & I \\ 0 & \cdots & \cdots & \cdots & 0 & I & D \end{bmatrix}, \text{ donde } D = \begin{bmatrix} -4 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & -4 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & -4 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & -4 & 1 & 0 \\ 0 & \cdots & \cdots & 0 & 1 & -4 & 1 \\ 0 & \cdots & \cdots & \cdots & 0 & 1 & -4 \end{bmatrix},$$

y como f es una matriz, se debe aplanar como vector columna. Por ejemplo en 3×3

$$A = \left[\begin{array}{ccc|ccc|ccc} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{array}\right].$$

Ahora debemos resolver los sistemas implementando los métodos de Jacobi,Gauss-Seidel y SOR, esto lo hicimos en Matlab y el código es extenso, por lo que lo adjuntamos en el cuaderno de Jupyter, sin embargo vamos a ver aquí los resultados obtenidos.

Para el método de Jacobi aprovechamos que la inversa de la matriz diagonal es simplemente invertir la diagonal, a saber $1/a_{ii}$, luego el código es bastante eficiente y puede correr 100 iteraciones muy rápido.

En Gauss-Seidel no fue posible implementar el método de manera matricial, invertir matriz $(D-L)$ es un trabajo costoso teniendo en cuenta que la primera imagen es de 240×240, es decir una matriz de 57600×57600, una sola iteración costaba varios minutos. En vista de esto optamos por implementar el método resolviendo cada ecuación y sin definir la matriz A, usando claramente que

$$\Delta u_{ij} = u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{ij} = f_{ij},$$

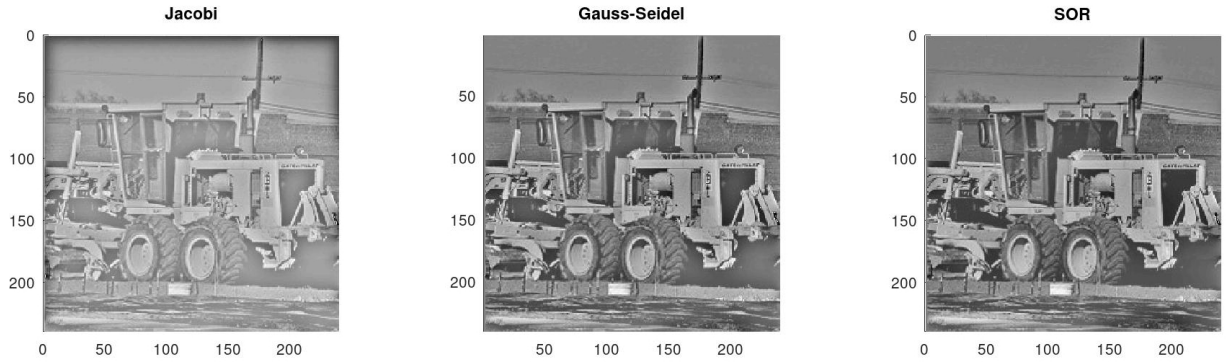
se obtiene una fórmula eficiente para hace Gauss-Seidel, a saber

$$u_{ij}^{(k+1)} = \frac{1}{4}\left(\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k+1)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k+1)}\right) - f_{ij}\right),$$

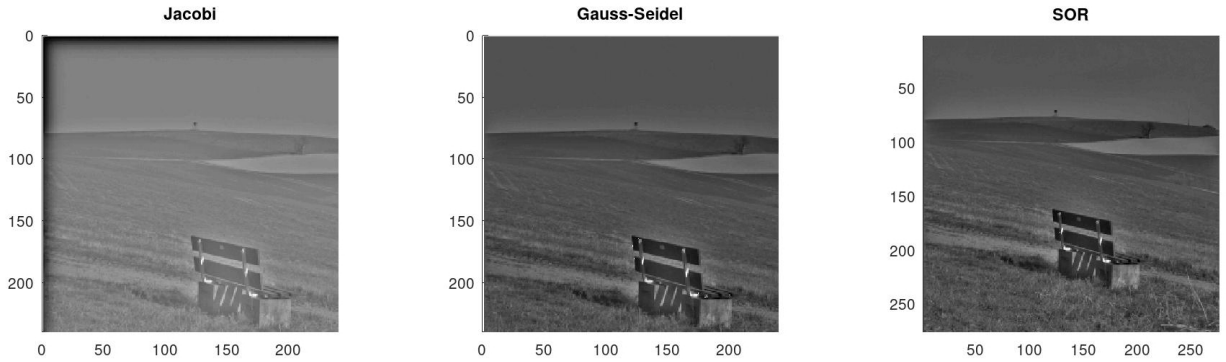
esto nos permitió implementar todo de manera eficiente. Para SOR la idea fue exactamente la misma pero aplicando la relajación

$$u_{ij}^{k+1} = (1-\omega)\,u_{ij}^{k} + \frac{\omega}{4}\left(\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k+1)} + u_{i,j+1}^{(k)} + u_{i,j-1}^{(k+1)}\right) - f_{ij}\right),$$

en este caso tomamos $\omega = 1{,}5$ ya que la idea era acelerar la convergencia, esto se logra con $1 < \omega < 2$. Finalmente para la primera imagen, dada por el archivo Bordes1, obtuvimos los siguientes resultados aplicando 100 iteraciones.



Para la segunda imagen también aplicamos 100 iteraciones, obtuvimos



Aplicamos 100 iteraciones para que se observara diferencia entre los métodos ya que con muchas iteraciones el problema se estabiliza de manera que no notaremos diferencia entre aplicar un método y otro.