Tarea 4. Calculo de eigenvalores

Ricardo Chávez Cáliz

October 4, 2017

Problema 1. Dado el siguiente
Teorema (Gershgorin):
Dada una matriz $A = a_{ij}$ de $m \times m$, cada eigenvalor de A está en al menos uno de los discos en el plano complejo con centro en a_{ii} y radio $\sum_{j \neq i} |a_{ij}|$. Además, si n de estos discos forman un dominio conexo, disjunto de los otros $m - n$ discos, entonces hay exactamente n eigenvalores en ese dominio.

Deduce estimaciones de los eigenvalores de

$$A = \begin{pmatrix} 8 & 1 & 0 \\ 1 & 4 & \epsilon \\ 0 & \epsilon & 1 \end{pmatrix}$$

Para la matriz A las estimaciones dependen completamente del valor de ϵ. Mientras más pequeño sea ϵ las estimaciones serán más precisas. Conversamente para grandes valores de ϵ las estimaciones que provee el teorema son imprácticas. De manera concreta, tenemos $\sum_{j \neq i} |a_{ij}| = 2 + 2|\epsilon|$, en este caso los valores propios se encuentran en el interior de los discos con centros en $1, 4, 8$. Como $\sum_{j \neq i} |a_{ij}| \geq 2$ y $max\{d(a,b)|a,b \in \{1,4,8\}\} = 4$ los discos cerrados siempre forman un dominio conexo, por lo tanto la segunda parte del teorema no nos da información adicional. La siguiente figura ejemplifica el dominio que hace referencia el teorema.

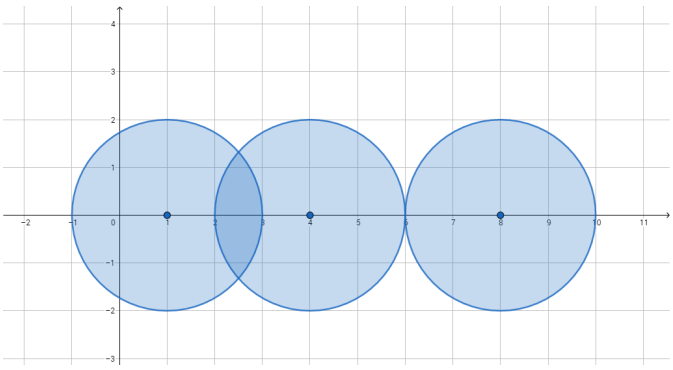

Figure 1: Se muestran discos en el plano complejo con centros en 1, 4 y 8 de radio 2 (mayor precisión a obtener)

En el caso en que A es hermitiana y por tanto sus eigenvalores son reales, entonces se tiene que $-1-2\epsilon \leq \lambda_i \leq 10+2\epsilon$

Problema 2. Implementa la iteración QR con shift. Aplícala a la matriz A del Ejercicio 1 con $\epsilon = 10^N$ para $N = 1, .., 5$.

Se implementaron los algortimos de iteración QR sin desplazamiento, con desplazamiento simple ($\sigma = a_{n,n}$) y con el desplazamiento de Wilkinson. Para este último si B es el menor de 2×2 del extremo inferior derecho de la matriz:

$$B = \begin{pmatrix} a_{n-1,n-1} & b_{n-1} \\ b_{n-1} & a_{n,n} \end{pmatrix}$$

El desplazamiento de Wilkinson es el eigenvalor de B más cercano a $a_{n,n}$. Una formula numéricamente estable para calcular el desplazamiento de Wilkinson es:

$$\mu = a_{n,n} - \frac{signo(\delta) b_{n-1}^2}{|\delta| + \sqrt{\delta^2 + b_{n-1}^2}}$$

donde $\delta = (a_{n-1,n-1} - a_{n,n})/2$

En caso de que para la iteración 100,000 no se tenga A_k cercana a una matriz diagonal ($||A_k - diag(A)||_\infty < 1e-8$) entonces se detiene el proceso y se imprime $||A_k - diag(A)||_\infty$ para saber que tan cerca se estaba de la convergencia.

Se obtuvieron los siguientes resultados

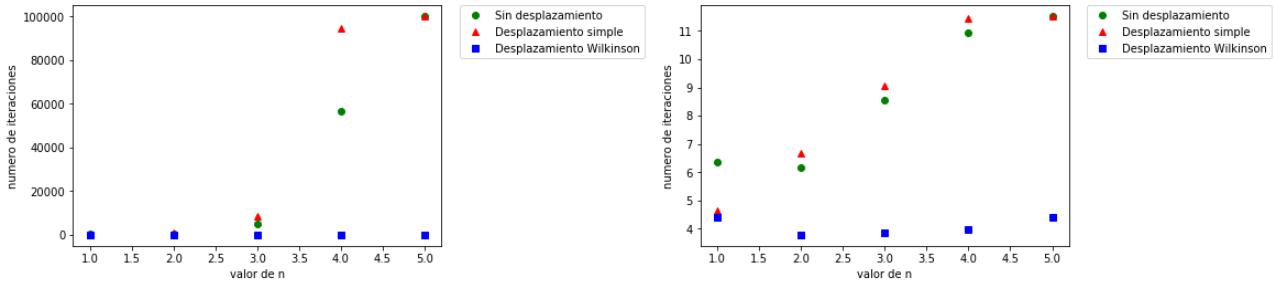

Figure 2: Se muestran el número de iteraciones requeridas en cada caso, hasta que A_k fue suficientemente cercana a una matriz diagonal ($||A_k - diag(A)||_\infty < 1e-8$), en escala normal y logaritmica

Para $n = 5$ la iteración QR sin desplazamiento y con desplazamiento simple rebasaron el límite establecido. Con $k = 100,000$ se obtuvo para iteración QR $||A_k - diag(A)||_\infty = 1347.3598064$, mientras que para iteración QR con desplazamiento simple $||A_k - diag(A)||_\infty = 9931.55732038$. Es decir que en ambos casos hacían falta aún muchas iteraciones para asegurar un buen cálculo de eigenvalores. Esto aunado con lo que se puede observar en la gráfica, se puede decir que la iteración QR sin desplazamiento fue más eficiente que la iteración QR con desplazamiento simple.

En la Figura 2, debido a la gran diferencia de iteraciones de cada método, no es posible apreciar la ejecución con el desplazamiento de Wilkinson, el cual es de nuestro interés por ser el más eficiente. Por eso se muestra la siguiente gráfica.

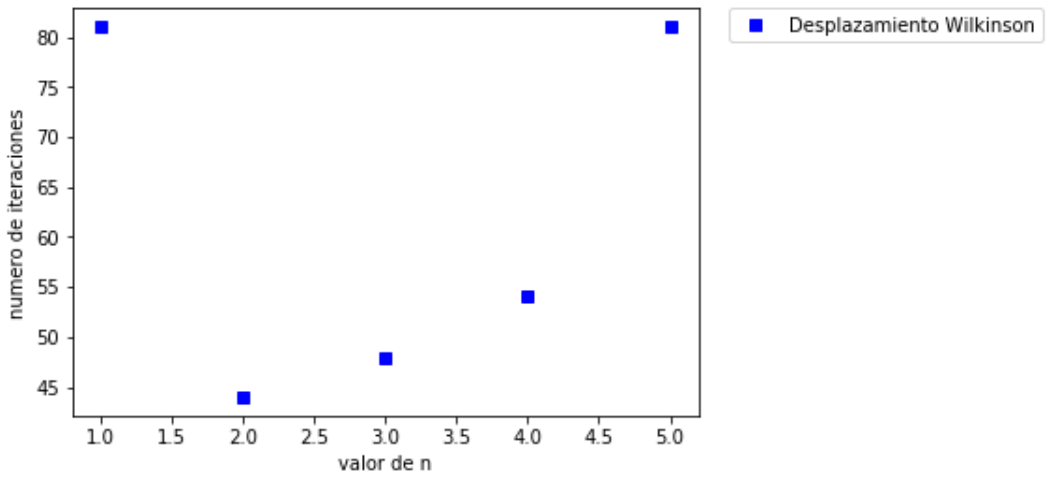

Figure 3: Se muestran el número de iteraciones requeridas en cada caso, hasta que A_k fue suficientemente cercana a una matriz diagonal ($||A_k - diag(A)||_\infty < 1e-8$)

Se observa una anomalía para $n = 1$, dado que requiere una cantidad elevada de iteraciones para lograr la convergencia. Por esto se comparó su tiempo de ejecución con el tiempo de ejecución del método para calcular eigenvalores de la librería de *numpy*. En el que se presenta la misma anomalía, a continuación se muestran los resultados.

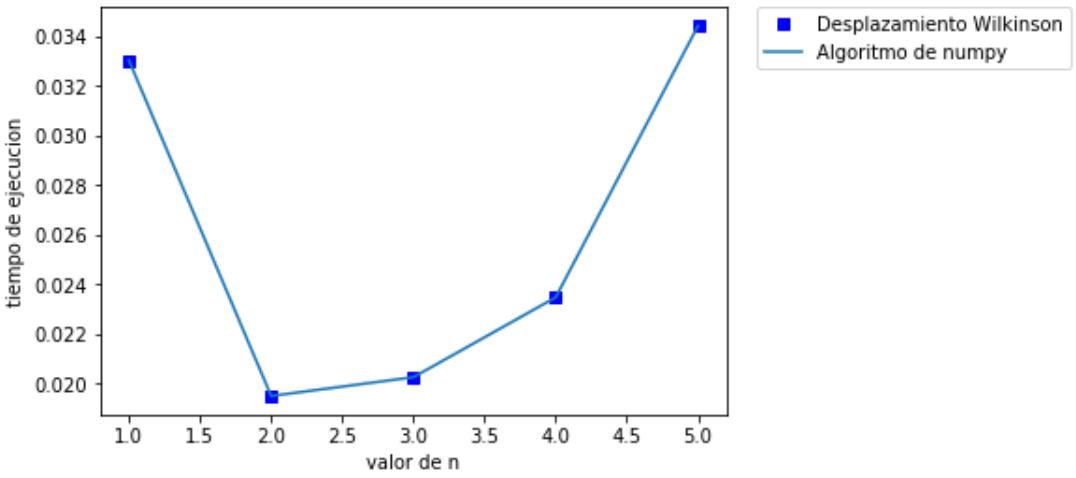

Figure 4: Se muestran tiempos de ejecución de el algortimo de iteración Qr con desplazamiento de Wilkinson contra el algoritmo de numpy

También se compararon los eigenvalores obtenidos en ambos métodos usando el método *allclose* de numpy, obteniendo *true* en cada caso.

Problema 3. Determina todos los eigenvalores y eigenvectores de una matriz de Householder.

Las matrices de Householder son ortogonales y simétricas.

Si A es ortogonal entonces $A \cdot A^t = Id$ y si λ es un eigenvalor de A con eigenvector v entonces $Av = \lambda v$, y tenemos que:

$$\|v\| = vv^t = v^t A^t Av = (Av)^t Av = (\lambda v)^t \lambda v = |\lambda| vv^t = |\lambda| \cdot \|v\|$$

Por lo tanto los eigenvalores de matrices ortogonales tienen valor absoluto 1 (norma compleja), es decir que la multiplicación por matrices ortogonales es una isometría.

Dado que la matriz de Housholder es real y simétrica, sus eigenvalores son reales y por lo tanto deben ser 1 o -1.

Dado que $Hu = u - 2u(u^t u) = -u$ (reflexión), hay al menos algún eigenvalor -1, y en efecto los vectores v perpendiculares a u satisfacen $Hv = v - 0u = v$ (proyección). De manera que hay una base completa de eigenvectores, uno para el eigenvalor -1 y el resto para los eigenvalores 1.

Es decir que una matriz de Householder tiene por valores propios $(-1, 1, \ldots, 1)$

Para verificar lo anterior se generó una muestra de 10 vectores aleatorios de tamaño 5 y se obtuvieron los eigenvalores para las matrices de Housholder construidas a partir de dicha muestra,usando el algoritmo de iteración QR con desplazamiento implementado anteriormente. Obteniendo en cada caso $[-1., 1., 1., 1., 1.]$

Problema 4. Demuestra que no es posible construir la transformación de similaridad del teorema de Schur con un número finito de transformaciones de similaridad de Householder.

Para dar contexto a la pregunta, recordemos que el determinante y los eigenvalores son invariantes bajo transformaciones de similitud. Por tanto una buena idea para aumentar la eficiencia de los algoritmos que calculan eigenvalores es buscar encontrar un representante sencillo en la misma clase de equivalencia por similitud de la matriz dada. Recordemos el teorema de Schur.

Teorema. Para toda matriz compleja A de $n \times n$ existen matrices S y T de $n \times n$ con S invertible y T triagular superior, tales que:

$$A = S^{-1} \cdot T \cdot S$$

La idea es aplicar transformaciones de similaridad a A de manera que podamos introducir ceros bajo la diagonal. Conocemos la matriz de Householder asociada a un vector v, la cual es una proyección ortogonal sobre v y una reflexión, por lo que una idea muy natural es usar dichas matrices una tras otra para lograr dicha descomposición.

La primer reflexión de Householder Q_1^*, multiplicada por la izquierda a A, introduce ceros por debajo de la diagonal en la primer columna de A. Ejemplificamos esto en los siguientes diagramas.

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix} \xrightarrow[Q_1^*]{} \begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix}$$

Para completar la transformación de similaridad, también debemos multiplicar por Q_1 en la derecha de A:

$$\begin{pmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{pmatrix} \xrightarrow[Q_1^*]{} \begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}$$

Lo cual tiene el efecto de reemplazar cada columna de la matriz por una combinación lineal de todas las columnas. El resultado es que los ceros inducidos anteriormente no vuelven a aparecer.

Problema 5. ¿Qué pasa si aplicas la iteración QR sin shift a una matriz ortogonal?

Recordemos que la iteración QR calcula una sucesión de matrices A_k de la siguiente manera. Se empieza con $A_0 = A$ y en la iteración k se calcula la descomposición QR de A

$$Q_k R_k = A_{k-1}$$

y luego se toma el producto invertido

$$A_k = R_k Q_k$$

Pero si A es una matriz ortogonal entonces $Q_1 = A$ y $R_1 = Id$ por lo tanto $A_1 = R_1 \cdot Q_1 = Id \cdot A = A$, y por lo tanto $A_k = A$ para toda $k \in \mathbb{N}$. Por lo que los elementos fuera de la diagonal no decrecen, y a menos de que hayamos empezado con el caso trivial ($A = Id$), A_k no converge a una matriz diagonal.

Para comprobar lo recién demostrado se construyó una matriz ortogonal y al intentar aplicar la iteración sin desplazamiento se obtuvo después de 100,000 iteraciones que los elementos fuera de a diagonal permanecieron igual.

```
In [266]: runfile('C:/Users/Pc/Documents/CC/tarea4.py', wdir='C:/Users/Pc/Documents/
CC')
Matriz ortogonal de ejemplo
[[ 5.  5.]
 [-5.  5.]]
El número de iteraciones ha exedido el límite establecido
norma de A-diag(A) es 5.0
```

Figure 5: Mensaje obtenido en la terminal al tratar de aplicar iteración QR sin desplazamiento a una matriz ortgonal