PRÁCTICA 1 – EIGENVALORES Y -VECTORES

1. Implementación

Esta sección describe los métodos que tienen que implementar.
Todos los funciones deben ser implementados en un ambiente: Matlab/Octave ó Python.

1.1. Método de la Potencia.

Implementa ese método en una función encabezada:

```
function [lam1, w] = mPI(A, q0, k, tol).
```

Los argumentos son una matriz `A`, un vector inicial `q0`, un número máximo `k` de iteraciones y una tolerancia `tol` para el error relativo. La función debe devolver la ultima aproximación del eigenpar dominante $(\lambda_1, \boldsymbol{v}_1)$, digamos $(\mu, \boldsymbol{w})$ con $\|\boldsymbol{w}\|_2 = 1$. Además, la función debe parar por el criterio relativo cuando `k` es grande.

1.2. Método de la Potencia inversa con *shift*.

Implementa ese método en una función encabezada:

```
function [lamn, w] = mPIinvShift(A, q0, rho, k, tol),
```

donde `A, q0, k, tol` son como arriba y `rho` es un *shift*. La función debe devolver una aproximación del eigenvalor más cercano a ρ y una aproximación del eigenvector asociado. Además, la función debe parar por el criterio relativo cuando `k` es grande.
(Note que el método de potencia inversa es un caso especial de esa función.)

1.3. Método de la Potencia inversa con cociente de Rayleigh.

Implementa ese método en una función encabezada:

```
function [lamj, w] = mPIinvRayleigh(A, q0, k)
```

donde `A, q0, k` son como arriba. La función debe devolver una aproximación de un eigenpar, digamos $(\lambda_j, \boldsymbol{v}_j)$. Para parar la iteración, deben implementar el siguiente criterio relativo

$$\|A\boldsymbol{w} - \rho_j \boldsymbol{w}\| \leq \texttt{tol} \cdot \|\boldsymbol{w}\| \quad \text{con} \quad \texttt{tol} = 10^{-7},$$

donde ρ_j es el cociente de Rayleigh. (Para A simétrica se debe reducir la tolerancia a 10^{-5}.)

(Esto se debe a lo siguiente. Cuando este método converge, el cociente de Rayleigh tiende hacia un eigenvalor. Por lo tanto la matriz $(A - \rho_i I)$ tiende a una matriz singular. Esa convergencia es cuadrática y cúbica si A es simétrica.)

Date: Fecha de entrega: 4 de Marzo de 2018 hora 23:55 (por equipos de 2 a 4 personas).

1.4. Método QR simple. Implementa ese método en una función encabezada:

```
function [lambdas, Q] = mQRsimple(A, k, tol)
```

donde los argumentos `A, k, tol` son como arriba. La función debe devolver una aproximación de los eigenvalores $\lambda_j, (j = 1, \cdots, n)$ de la matriz A y la matriz ortogonal acumulada tal que $Q^\top AQ = T$. El método debe parar después de `k` pasos ó cuando el siguiente criterio aplica

```
norm(diag(Am, -1)) <tol
```

(recuerda que $A_m \to T$ con T triangular superior).

1.5. Método QR con *shifts* dinámicos para matrices simétricas. Implementa ese método en una función encabezada:

```
function [lambdas] = mQRdyna(A, k, tol)
```

donde los argumentos `A, k, tol` son como arriba. La función debe devolver una aproximación de los eigenvalores $\lambda_j, (j = 1, \cdots, n)$ de la matriz A. El criterio para parar la aproximación de cada eigenvalor se encuentra en las notas de la clase. El parámetro `k` debe ser usado para limitar las iteraciones hacia un eigenvalor. Para no tener problemas con la simetría use el `shift` de Wilkinson, dado por:

$$\mu = a_m - \operatorname{sign}(\delta)\frac{b_{m-1}^2}{|\delta| + \sqrt{\delta^2 + b_{m-1}^2}},$$

donde

$$\delta = \frac{1}{2}(a_{m-1} - a_m) \quad \text{and} \quad B = \begin{pmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{pmatrix} \quad \text{es la ultima submatriz de } A_m.$$

Atención: El método `mQRdyna` NO da los eigenvalores ordenados según magnitud. Hay que ordenar los eigenvalores y eigenvectores con una permutación. Es permutación de puede obtener con `[v, P] = sort(lams, 'descend')`. Después, las permutaciones de los eigenvalores y -vectores se obtienen por `lams(P), Q(:,P)`.

Optativo: Acumular Q_m para obtener los eigenvectores (es posible, las A son simétricas).

1.6. Método SVD economizada. El método QR (que devuelve también la matriz ortogonal Q), da los eigenpares para matrices simétricas. Cuando lo aplican a la más pequeña de las matrices $A^\top A$ ó $AA^\top$, pueden encontrar los vectores singulares por la izquierda ($A^\top A$ es la pequeña) o por la derecha ($AA^\top$ es la pequeña). Supongamos que valores singulares son

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r \geq 10^{-7} > \sigma_{r+1} \geq \ldots \geq \sigma_k \geq 0$$

donde $k = \min\{m, n\}$. Entonces, podemos definir los vectores singulares que no hemos calculado por

$$\boldsymbol{u}_i = \frac{1}{\sigma_i} A\boldsymbol{v}_i \qquad \text{ó} \qquad \boldsymbol{v}_i = \frac{1}{\sigma_i} A^\top \boldsymbol{u}_i \qquad (i = 1, \ldots, r).$$

Así obtenemos la siguiente aproximación numérica de la SVD

$$A = U\Sigma V^\top \quad \text{con} \quad U \in \mathbb{R}^{m\times r},\ \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_r),\ V \in \mathbb{R}^{n\times r}.$$

Implementa ese método en una función encabezada:

```
function [U, S, V] = mSvdEco(A)
```

donde las matrices $U, S = \Sigma, V$ son como arriba.

2. Verificación de la implementación y teoría

Ejecuta el comando `format long` para mostrar más dígitos.

2.1. Ejercicio. Sea

$$A = \begin{pmatrix} 1 & 1 & 2 \\ -1 & 9 & 3 \\ 0 & -1 & 3 \end{pmatrix}.$$

1. Usando tu programa, aplica el método de la potencia a la matriz A, empezando con el vector inicial $q_0 = (1, 1, 1)$. Haz exactamente 10 iteraciones.
2. Usa el comando `[V,D] = eig(A)` de Matlab para obtener los eigenvalores y -vectores "exactos" de A. Compara el valor propio dominante de `D` con tu resultado del apartado 1.
3. Sea v el eigenvector dominante en `V` (dado por `eig` arriba). Ese está normalizado y tiene la norma euclidiana uno. Calcula las razones

$$\frac{||q_{j+1} - v||_\infty}{||q_j - v||_\infty}, \qquad j = 1, 2, 3, \ldots.$$

4. Usando `D` (dado por `eig` arriba), calcula la razón $|\lambda_2/\lambda_1|$ y compáralo con las razones del apartado 3.

2.2. Ejercicio. Repite el Ejercicio 1 con el método de la potencia inversa con *shift*:

- $\rho = 0$: Recuerde la razón exacta en este caso es $|\lambda_n/\lambda_{n-1}|$.
- $\rho = 3.3$: La razón exacta para ese caso se deja calcular usando `eig`.

2.3. Ejercicio. ¿Cuántos iteraciones requiere el método de potencia inversa con *shift* $\rho = 3.6$ para cuando la tolerancia relativa es 10^{-15}?

2.4. Ejercicio. Usa la matriz del Ejercicio 1 y tu programa de potencia inversa con cociente de Rayleigh con varios vectores iniciales q_0 para calcular tres eigenvectores distintos y sus eigenvalores asociados.

Además, verifica para uno de los tres vectores q_0 y su límite (un eigenvector v_j) que la convergencia es cuadrática.

2.5. Ejercicio. Decide si el método de la potencia inversa funciona para la matriz siguiente:

$$A = \begin{pmatrix} 1 & -4 & -6 \\ -12 & -8 & -6 \\ 11 & 10 & 10 \end{pmatrix}.$$

2.6. Ejercicio. Comparen los resultados (errores en eigenvalores) de su implementación del método QR simple y los del QR con *shifts* dinámicos con el resultado del ambiente Matlab `eig`. Para ello use la matriz dada por el siguiente comando de Matlab:

```
gallery( 'fiedler', 25 )
```

y aplica los métodos como sigue:

- `mQRsimple( gallery( 'fiedler', 25 ), 2000, 1E-10)`
- `mQRdyna( gallery( 'fiedler', 25 ), 20, 1E-10)`
- `eigs( gallery( 'fiedler', 25 ), 25)`

Además, cuenta las iteraciones requeridas por tu implementación del método QR simple *vs* QR con *shift* dinámico.

2.7. Ejercicio. Hay dos archivos en la práctica `image_compression_svd.m` y `small.jpg`. Esos les deben ayudar a ver si su implementación del método SVD es correcta.