

Actividades 7

Prácticas de Cálculo Numérico I (doble grado)

27 de abril de 2021

1. El método de la potencia

En lo sucesivo usaremos como ejemplo las matrices A_1 , A_2 y A_3 definidas con `matlab/octave` como:

```
1 % Autovalores: 4, 1\pm i, 1
2 A1 = [-7,6,-8,5; -16,10,-13,10; -5,2,-2,3; -8,6,-8,6];
3
4 % Autovalores: 4, 3, 2, 1
5 A2 = [-1,3,-5,2; -7,7,-7,4; -1,1,1,0; -2,3,-5,3];
6
7 % Autovalores: 8, 6, 4, 2
8 A3 = [5,0,-2,1; 0,5,-1,2; -2,-1,5,0; 1,2,0,5];
```

Los autovalores indicados se pueden comprobar con `eig`. Recordemos que con `eig(A)` se obtienen los autovalores y con `[C,D]=eig(A)` la diagonalización $A = CDC^{-1}$. Para A_1 y A_2 es posible elegir C con elementos enteros pero como `eig` normaliza, con este comando no nos percataremos de ello. La matriz A_3 es simétrica y por tanto diagonaliza en una base ortonormal.

Aunque disponemos del comando `eig`, este es un curso para matemáticos y nuestro objetivo será entender algunos métodos para el cálculo, aunque sea parcial, de autovectores y autovalores de una matriz cuadrada. Una cuestión a tener en cuenta es que, desde el punto de vista numérico, todas las matrices parecen diagonalizables sobre \mathbb{C} porque cada matriz no diagonalizable está infinitamente cerca en norma de una infinidad de ellas que sí lo son. Un enemigo fundamental del cálculo numérico de autovalores y autovectores es la existencia de multiplicidades.

Uno de los métodos más sencillos para calcular un autovalor y un autovector de una matriz cuadrada A es la iteración $\vec{x}_{n+1} = A\vec{x}_n$, lo que se llama *método de la potencia* porque $\vec{x}_n = A^n\vec{x}_0$. Habitualmente se normalizan los \vec{x}_n para evitar divergencias. Si A tiene un solo *autovalor dominante* λ_1 , es decir, es simple y $|\lambda_1| > |\lambda_j|$ para $j \neq 1$, para casi cualquier elección

de \vec{x}_0 , \vec{x}_n se acerca paulatinamente a un autovector correspondiente al autovalor λ_1 y, consecuentemente, λ_1 se aproxima por el llamado *cociente de Rayleigh* $\langle A\vec{x}_n, \vec{x}_n \rangle / \|\vec{x}_n\|^2$, donde el denominador es 1 si hemos normalizado. Por ejemplo, $-A_1$ tiene autovalor dominante -4 que podemos aproximar, junto con su autovector con el código:

```

1  A1 = [-7,6,-8,5; -16,10,-13,10; -5,2,-2,3; -8,6,-8,6];
2  A = -A1;
3  N = 8;
4  v = rand(size(A,1),1);
5  v = v/norm(v);
6
7  for k = 1:N
8      vn = A*v;
9      v = vn/norm(vn);
10 end
11
12 disp('Aproximación del autovector:')
13 disp(v)
14
15 disp('Aproximación del autovalor:')
16 disp(v'*A*v)

```

Normalmente queremos establecer un criterio de parada para saber si se ha conseguido cierta tolerancia en el error relativo estimando autovalores o autovectores, antes de haber gastado todas las iteraciones máximas disponibles. Para los primeros, una posibilidad es cambiar el bucle por:

```

1  tol = 0.01
2  fl = false;
3  for k = 1:N
4      vn = A*v;
5      vn = vn/norm(vn);
6      la = v'*A*v;
7      lan = vn'*A*vn;
8      v = vn;
9      if abs(la/lan-1) < tol
10         fl = true;
11         break
12     end
13 end
14
15
16 if fl
17     disp(['Se ha alcanzado la tolerancia exigida con N
18         ↔ = ' num2str(k)])
19 else
20     disp('No se ha alcanzado la tolerancia exigida')

```

Si queremos usar un criterio similar con los autovalores hay que tener cuidado porque aunque informalmente decimos que \vec{x}_n “converge” al autovector, los autovectores solo están definidos salvo multiplicar por constantes. Si por ejemplo λ_1 es negativo, \vec{x}_n cambiará alternativamente de sentido y algo más complicado ocurre para autovalores complejos. Una solución es normalizar poniendo una de las coordenadas iguales a 1, típicamente la mayor en valor absoluto.

ACTIVIDAD 7.1.1. Siguiendo la idea anterior, trata de hacer un criterio de parada que tome en cuenta el error relativo de los autovectores.

La rapidez de convergencia depende de lo grande que sea el cociente $|\lambda_1|/|\lambda_2|$ donde λ_2 es el *autovalor subdominante*, el siguiente en tamaño del módulo, que podría ser múltiple. Cuanto más cercano sea a uno este cociente, peor para la convergencia. Esto lleva a una situación curiosa. Matemáticamente tiene la misma complicación hallar los autovalores y autovectores de A que los de $A - \mu I$ pero numéricamente los cocientes $|\lambda_1|/|\lambda_2|$ pueden ser bien distintos. Por ejemplo, para A_2 es $4/3$ y para $A_2 - 2I$ es 2 . Por tanto, es más conveniente aplicar el algoritmo a $A_2 - 2I$. Comprobémoslo:

```

1 A2 = [-1,3,-5,2; -7,7,-7,4; -1,1,1,0; -2,3,-5,3];
2 A = A2;
3 N = 10;
4 n = size(A,1)
5 v1 = rand(n,1);
6 v1 = v1/norm(v1);
7 v2 = v1;
8
9 for k = 1:N
10     vn1 = A*v1;
11     vn2 = (A-2*eye(n))*v2;
12     v1 = vn1/norm(vn1);
13     v2 = vn2/norm(vn2);
14     disp([num2str(k) ' : aprox 1 -> '
           ↪ num2str(v1'*A*v1, '%.6f') ' aprox 2 -> '
           ↪ num2str(v2'*A*v2, '%.6f')])
15 end

```

2. Localización de autovalores

Según lo que acabamos de ver, sabremos acerca de la velocidad de convergencia con una idea previa de la localización de los autovalores. Los *círculos de Gershgorin* dan una información burda que se vuelve muy precisa para pequeñas perturbaciones de matrices diagonales. Un resultado [QS07, §6.3] afirma que los autovalores de una matriz A están, en el plano complejo dentro de la unión de los círculos (de Gershgorin) $|z - a_{ii}| \leq \sum_{j \neq i} |a_{ij}|$ y que si la unión de m de estos círculos está aislada de los otros, entonces dicha unión contiene exactamente m autovalores.

El siguiente código dibuja los círculos de Gershgorin para una matriz aleatoria con a_{11} mucho mayor que el resto de los elementos, con ello λ_1 estará en el círculo aislado a la derecha y λ_2 en la unión de los otros.

```

1 N = 5;
2 A = rand(N,N);
3 A(1,1) = A(1,1) + 2*N;
4

```

```

5  n = size(A,1);
6
7  t = linspace(0,2*pi,100);
8  figure(1)
9  for k = 1:n
10     r = sum( abs(A(k,:)) ) - abs(A(k,k));
11     plot( real(A(k,k))+ r*cos(t), imag(A(k,k))+
           ↪ r*sin(t) )
12     hold on
13 end
14 axis('equal')
15 hold off

```

Usar `real`, `imag` en la línea 11 permite que el programa funcione para matrices complejas.

ACTIVIDAD 7.2.1. *Añade alguna línea de código para que muestre también las posiciones reales de los autovalores. Intenta extremar la brevedad. Se puede hacer con una sola línea.*

En física se presenta la situación en la que uno conoce un autovalor (simple) λ de una matriz hermítica A y su autovector normalizado \vec{v} y quiere estimar sus análogos para $A' = A + P$ donde P es una matriz hermítica pequeña. La teoría de perturbación dice que las fórmulas relevantes son

$$\lambda' \approx \lambda + \vec{v}^\dagger P \vec{v} \quad \vec{v}' \approx \vec{v} + \sum_{\mu \neq \lambda} \frac{\vec{v}_\mu^\dagger P \vec{v}}{\lambda - \mu} \vec{v}_\mu$$

donde μ recorre los autovalores de A distintos de λ y \vec{v}_μ los correspondientes autovectores normalizados. Esto permite encontrar un buen punto de partida para el método de la potencia o la extensión que veremos a continuación.

ACTIVIDAD 7.2.2. *Comprueba que la primera de las fórmulas anteriores dan buenas aproximaciones para los autovalores de $A3+ep*(R+R')$ con $R = \text{rand}(4)$ y ep pequeño, considerada como una perturbación de $A3$.*

3. El método de la potencia inversa

Si aplicamos el método de la potencia a A^{-1} el autovalor de menor valor absoluto pasa a ser el dominante. En general, aplicando el método a $(A - \mu I)^{-1}$ con μ cercano a λ_j (supuesto autovalor simple), conseguiremos que λ_j sea dominante y podremos aplicar el método de la potencia. Esta es la forma más básica del *método de la potencia inversa* y funciona tanto para autovalores reales como complejos, como se puede apreciar en el siguiente ejemplo:

```

1 A1 = [-7,6,-8,5; -16,10,-13,10; -5,2,-2,3; -8,6,-8,6];
2 A = A1;
3 N = 8;
4 v = rand(size(A,1),1);
5 v = v/norm(v);
6
7 la0 = 0.8+0.8*i;
8 Ai = inv(A-la0*eye(size(A,1)));
9
10 for k = 1:N
11     vn = Ai*v;
12     v = vn/norm(vn);
13 end
14
15 disp('Aproximación del autovector:')
16 disp(v)
17
18 disp('Aproximación del autovalor:')
19 disp(v'*A*v)

```

Como A_i no varía, para un número grande de iteraciones, nos conviene calcular la inversa una sola vez, aunque esto sea costoso pero también podríamos ir actualizando el valor de μ con las aproximaciones del autovalor para conseguir una convergencia más rápida. En esos casos, el cálculo de la inversa es menos ventajoso que la resolución del sistema en cada iteración. El siguiente código, que alguno diría que corresponde al algoritmo llamado *iteración del cociente de Rayleigh* (véase también la variante [PTVF92, (11.7.10)]), ilustra la situación:

```

1 A1 = [-7,6,-8,5; -16,10,-13,10; -5,2,-2,3; -8,6,-8,6];
2 A = A1;
3 N = 5;
4 v = rand(size(A,1),1);
5 v = v/norm(v);
6
7 la = 0.8+0.8*i;
8
9 for k = 1:N
10     vn = (A-la*eye(n))\v;
11     v = vn/norm(vn);
12     la = v'*A*v;
13 end
14
15 disp('Aproximación del autovector:')
16 disp(v)
17
18 disp('Aproximación del autovalor:')
19 disp(v'*A*v)

```

La convergencia es mucho más rápida que en el caso anterior pero a cambio hay que pagar con resolver un sistema en cada iteración.

ACTIVIDAD 7.3.1. *Escribe un programa que compare el error en la aproximación del autovalor en las dos formas del algoritmo para este ejemplo.*

4. El algoritmo QR (teórico)

Según la teoría, partiendo de $A_0 = A$ real con autovalores reales simples, la iteración $A_{n+1} = R_n Q_n$ donde Q_n y R_n son los factores en la descomposición QR de A_n , converge a una matriz triangular superior T igual a cambiar A a cierta base ortonormal, cuya diagonal son los autovalores de A . Si además A es simétrica, T será diagonal. Esta es la teoría bajo el algoritmo QR pero como cada paso requiere del orden de n^3 operaciones, es bastante poco eficiente para matrices medianamente grandes. Por otro lado, es ridículamente simple de programar en `matlab/octave`:

```
1 A2 = [-1,3,-5,2; -7,7,-7,4; -1,1,1,0; -2,3,-5,3];
2 A = A2;
3 N = 15;
4 n = size(A,1);
5 %Q = eye(n);
6
7 for k = 1:N
8     [Qp,R] = qr(A);
9     A = R*Qp;
10    %    Q = Q*Qp;
11 end
12
13 disp('Aproximación de los autovalores')
14 disp(diag(A))
```

El método se puede adaptar con pocos cambios al caso de autovalores complejos [SB93, 6.6.4.15] pero no lo veremos aquí. Esencialmente lo que ocurre es que aparece un valor $T_{i,i+1} \neq 0$ por cada pareja de autovalores conjugados.

Si quitamos el comentario en las líneas 5 y 10, en Q tendremos una matriz unitaria tal que $Q^\dagger A_2 Q$ es aproximadamente triangular superior T , cuyos autovectores son fáciles de hallar y hallaríamos una aproximación de los autovectores de A_2 multiplicando los de T por Q .

ACTIVIDAD 7.4.1. *Escribe una función `eigt` tal que `[C,D] = eig(T)` funcione como `eig` para una matriz triangular superior R con todos los elementos de su diagonal distintos.*

Sin entrar en detalles, la forma de volver eficiente el algoritmo es reducir primero A a una matriz de Hessenberg superior con un cambio de base unitario. Estas matrices son las que tienen $a_{ij} = 0$ para $i > j + 1$. La particularidad de ellas es que la descomposición QR se puede hacer de forma mucho más económica. Así que el esquema sería:

```
1 A2 = [-1,3,-5,2; -7,7,-7,4; -1,1,1,0; -2,3,-5,3];
2 A = A2;
3 N = 10;
4 n = size(A,1);
5
```

```

6 [P,A] = hess(A);
7 %Q = P;
8
9 for k = 1:N
10     [Qp,R] = qr(A);
11     A = R*Qp;
12     Q = Q*Qp;
13 end
14
15 disp('Aproximación de los autovalores')
16 disp(diag(A))

```

El comando `hess` de la línea 6 halla P unitaria con $P^\dagger AP$ de Hessenberg que se vuelve a almacenar en A .

Referencias

- [PTVF92] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in C*. Cambridge University Press, Cambridge, second edition, 1992. The art of scientific computing.
- [QS07] A. Quarteroni and F. Saleri. *Cálculo Científico con MATLAB y Octave*. Springer, Milan, 2007.
- [SB93] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*, volume 12 of *Texts in Applied Mathematics*. Springer-Verlag, New York, second edition, 1993. Translated from the German by R. Bartels, W. Gautschi and C. Witzgall.