Métodos Numéricos Avanzados
Model Eigenvalue Problem - MV - MODE

Facundo Edgardo Alderete (51063), Federico Elli (50817), Leandro Matías Rivas (51274),
Germán Rodrigo Romarión (51296)
Trabajo Práctico 1 - Instituto Tecnológico de Buenos Aires (ITBA)

Resumen - En el presente artículo se describen los procedimientos para obtener autovalores de una ecuación diferencial ordinaria.

Palabras clave - Autovalores, matrices, ecuación diferencial ordinaria, diferencias finitas, eliminación gaussiana, edo, Octave.

---

1 - INTRODUCCIÓN

El trabajo práctico que nos fue asignado consiste en modelar el problema de una ecuación diferencial ordinaria utilizando autovalores. Para ello seguiremos el ejemplo propuesto por G.W. Stewart.

En la primera parte se construirán las matrices A, B y C (detalladas en el paper de G.W. Stewart) dados n y γ. En la segunda parte se calcularán numéricamente los valores μ dados, nuevamente, n y γ.

El funcionamiento de los algoritmos se verifica utilizando las matrices ODEP400A y ODEP400B, que se encuentran en Matrix Market.

2 - METODOLOGÍA UTILIZADA

Consideramos la siguiente ecuación diferencial ordinaria de segundo orden.

$$y'' + \mu^2 y = 0 \quad (1)$$

Con las condiciones iniciales

$$y(0) = 0 \text{ y } y'(0) + \gamma y'(1) = 0\,,\ 0 < \gamma < 1\,. \quad (2)$$

Este problema puede ser aproximado mediante diferencias finitas si se redistribuyen los elementos de la ecuación (1) y se toman puntos aislados de la función en lugar de la misma en su totalidad.

$$y'' = -\mu^2 y \quad (3)$$

Entonces, si y_i es la imagen aproximada para la ordenada $x_i = \frac{i}{(n+1)}\ (i = 0, 1, \ldots, n)$ y se toma el vector $y = (y_1, y_2, \ldots, y_n)^T$, puede expresarse la segunda derivada de y a través de la matriz A, como se muestra en la ecuación (4).

$$Ay = -\mu^2 By \quad (4)$$

La resolución de este problema involucra tres matrices. La matriz A es cuadrada y de tamaño (n+1)x(n+1), estando compuesta por el valor -2 en su diagonal, el valor 1 en sus semidiagonales y su última fila por valores dependientes de gamma.

$$A = \begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ 4 & -1 & \cdots & \gamma & -4\gamma & 3\gamma \end{pmatrix}$$

(5)

Por su parte, la matriz B tiene el mismo tamaño que A y el valor h^2 en su diagonal. Esta matriz podría llamarse diagonal, de no ser porque en su última fila todos los valores son iguales a cero.

$$B = h^2 diag(1, 1, ..., 1, 0) \quad (6)$$

Finalmente, la matriz C surge de la multiplicación de A^{-1} con B, como se muestra en la ecuación (7).

$$C = A^{-1}B \quad (7)$$

Lo cual deriva en la ecuación (8).

$$Cy = -\frac{1}{\mu^2} y \quad (8)$$

Los autovalores μ surgen de la ecuación (9).

$$\mu = i\, cosh^{-1}(-\gamma^{-1}) \quad (9)$$

Cuyas soluciones son de la forma correspondiente a la ecuación (10).

$$\mu^2 = ((2k + 1)^2\pi^2 - ln^2\sigma) - i(2(2k + 1)\,\pi\, ln\,\sigma) \quad (10),$$

$$\text{donde } k = 0,\ \pm 1,\ \pm 2,\ \ldots \quad (11)$$

$$\text{y } \sigma = 1/\gamma \pm \sqrt{1/\gamma^2 - 1} \quad (12)$$

Entonces, utilizando la ecuación (10), podemos encontrar todos los posibles valores de μ , y así, los autovalores de la matriz C.

3 - RESULTADOS OBTENIDOS

Mediante los algoritmos desarrollados fuimos capaces de generar las matrices A, B y C de acuerdo a los parámetros especificados en el enunciado, de modo que siempre sigan la forma necesaria para modelar el problema, sin importar su tamaño o el valor de gamma que quiera utilizarse.

Además, basándonos en la ecuación 10, desarrollamos un algoritmo que nos permite hallar todos los posibles valores de μ, a partir de los cuales pudimos encontramos los valores de $-\frac{1}{\mu^2}$, los cuales se corresponden con los que devuelve la función *eig* de Octave para la matriz C. De este modo verificamos el correcto funcionamiento de nuestro algoritmo. Finalmente verificamos las limitaciones de nuestro algoritmo imponiendo un tamaño excesivo para las matrices A y B, llegando a un límite final de *n = 100000*, por cuestiones de memoria. Aunque un valor menor, como *n = 10000*, tampoco fue posible porque el algoritmo tardó demasiado en completarse.

4 - ALGORITMOS

Para realizar

- MATRIZ A
    - El cálculo de la matriz se halla en la función homónima que se encuentra en el archivo ABC.m en la línea 7 (consultar *APÉNDICE*).
      Lo primero que hicimos fue usar la función eye(n + 1), que devuelve la matriz identidad de (n + 1)x(n + 1). Luego multiplicamos a dicha matriz por el valor escalar -2, y en las semidiagonales colocamos el valor 1, salvo en la última fila. Para concluir, en la última fila colocamos los valores 4 y -1 en la primer y segunda columna respectivamente, y los valores γ, -4γ y -3γ.

- MATRIZ B

- El cálculo de la matriz se halla en la función homónima que se encuentra en el archivo ABC.m en la línea 23 (consultar *APÉNDICE*).
  Como se indica en (6), B no es más que una matriz con valores únicamente en la diagonal iguales a h^2, con $h = \frac{1}{(n+1)}$, excepto la última fila que está compuesta únicamente por ceros. Por lo tanto, B será nuevamente la matriz eye(n + 1) multiplicada por h, y colocando en (n + 1, n + 1) el valor 0.

- MATRIZ C
  - El cálculo de la matriz se halla en la función homónima que se encuentra en el archivo ABC.m en la línea 28 (consultar *APÉNDICE*)
    Como se indica en (7), C es equivalente a la multiplicación entre la transpuesta de la matriz A y la matriz C.

- VALORES DE μ
  - El cálculo de valores se halla en la función *mus* que se encuentra en el archivo mus.m en la línea 1 (consultar *APÉNDICE*).
    Para hallar los valores de μ seguimos la ecuación (10). Primero calculamos las constantes en la ecuación, tales como los valores de σ, el $ln(\sigma)$ y el $ln(\sigma)^2$, de manera de no tener que hacerlo cada vez para un valor distinto de *k*. Como lo que estamos queriendo buscar es la raíz cuadrada de la ecuación (10), los valores que obtendremos serán $\pm\mu$. Habiendo aclarado esto, procedemos a calcular los primeros cuatro valores de mu: dos negativos y dos positivos, cada uno con k = 0. Tratamos a estos cuatro valores como un caso particular para evitar meter la ecuación dentro del ciclo definido en la línea 16 y obtener valores repetidos. Los valores de mu vienen de a pares debido a los posibles valores que puede tomar σ en la ecuación (12). Luego iteramos por la cantidad de veces solicitadas por el usuario (determinadas por el parámetro *limit* de la función) para obtener en cada caso ocho valores extra de mu: cuatro con signo positivo y cuatro con signo negativo.
    Los valores obtenidos pueden ser usados para calcular los autovalores de la matriz C, mencionada en la ecuación (7). Para esto, el usuario puede ejecutar la siguiente línea en Octave:

```
-1./mus(400, 0.01, 3).^2
```

    donde *n* = 400, γ = 0.01 y *limit* = 3.

APÉNDICE

1 - CÓDIGO

Funciones para la obtención de las matrices A, B y C en base a *n* y γ. Si desea ver el código fuente con mayor claridad, puede visitar https://github.com/gromarion/MNA_TP1.

ABC.m

```
function [A, B, C] = ABC(n, gamma)
    A = A(n, gamma);
    B = B(n);
    C = C(A, B);
end

function A = A(n, gamma)
    A = -2 * eye(n + 1);
    for i = 2 : n
        A(i - 1, i) = 1;
        A(i + 1, i) = 1;
    end
    A(2, 1) = 1;
    A(n, n + 1) = 1
    A(n, n - 1) = 1;
    A(n + 1, 1) = 4;
    A(n + 1, 2) = -1;
    A(n + 1, n - 1) = gamma;
    A(n + 1, n) = -4 * gamma;
    A(n + 1, n + 1) = 3 * gamma;
end

function B = B(n)
    B = h(n)^2 * eye(n + 1);
    B(n + 1, n + 1) = 0;
end

function C = C(A, B)
    C = A^-1 * B;
end

function h = h(n)
    h = 1 / (n + 1);
end
```

Función para la obtención de los μ

mus.m

```
function result = mus(n, gamma, limit)
    mus = zeros(1, 8 * limit + 4);
    sigma_plus = 1 / gamma + sqrt(1 / gamma^2 - 1);
    sigma_minus = 1 / gamma - sqrt(1 / gamma^2 - 1);
    ln_sigma_plus = log(sigma_plus);
    ln_sigma_minus = log(sigma_minus);
```

```
7       ln_sigma_plus_2 = ln_sigma_plus^2;
8       ln_sigma_minus_2 = ln_sigma_minus^2;
9
10      mus(1) = - sqrt((pi^2 - ln_sigma_plus_2) - (2 * pi * ln_sigma_plus) * i);
11      mus(2) = - sqrt((pi^2 - ln_sigma_minus_2) - (2 * pi * ln_sigma_minus) * i);
12      mus(3) = sqrt((pi^2 - ln_sigma_plus_2) - (2 * pi * ln_sigma_plus) * i);
13      mus(4) = sqrt((pi^2 - ln_sigma_minus_2) - (2 * pi * ln_sigma_minus) * i);
14      l = 5;
15      result_size = 0;
16      for k = 1 : limit
17              for j = 0 : 1
18                      [mu1, mu2, mu3, mu4] = partial_mus(k, mod(j, 2), ln_sigma_plus,
ln_sigma_minus, ln_sigma_plus_2, ln_sigma_minus_2);
19                      [mus, size_1] = add_if_not_added_yet(mus, l, mu1);
20                      [mus, size_2] = add_if_not_added_yet(mus, l + 1, mu2);
21                      [mus, size_3] = add_if_not_added_yet(mus, l + 2, mu3);
22                      [mus, size_4] = add_if_not_added_yet(mus, l + 3, mu4);
23                      result_size = result_size + size_1 + size_2 + size_3 + size_4;
24                      l = l + 4;
25              end
26      end
27      result = eliminate_null_values(mus, result_size);
28  end
29
30  function [mu1, mu2, mu3, mu4] = partial_mus(k, j, ln_sigma_plus, ln_sigma_minus,
ln_sigma_plus_2, ln_sigma_minus_2)
31      mu1 = (-1)^j * sqrt(((2 * k  + 1)^2 * pi^2 - ln_sigma_plus_2) - ((2 * k + 1) * 2
* pi * ln_sigma_plus) * i);
32      mu2 = (-1)^j * sqrt(((2 * k  + 1)^2 * pi^2 - ln_sigma_minus_2) - ((2 * k + 1) *
2 * pi * ln_sigma_minus) * i);
33      mu3 = (-1)^j * sqrt(((2 * (-k) + 1)^2 * pi^2 - ln_sigma_plus_2) - ((2 * (-k) +
1) * 2 * pi * ln_sigma_plus) * i);
34      mu4 = (-1)^j * sqrt(((2 * (-k) + 1)^2 * pi^2 - ln_sigma_minus_2) - ((2 * (-k) +
1) * 2 * pi * ln_sigma_minus) * i);
35  end
36
37  function [mus, size] = add_if_not_added_yet(mus, l, mu)
38      exists = 0;
39      size = 0;
40      for k = 1 : length(mus)
41              if (mus(k) - mu > 0 && mus(k) - mu <= 0.001)
42                      exists = 1;
43                      break;
44              end
45      end
46      if (exists == 0)
47              mus(l) = mu;
48              size = 1;
49      end
50      mus;
51  end
52
53  function result = eliminate_null_values(mus, size)
54      result = zeros(1, size);
55      result_index = 1;
56      for k = 1 : length(mus)
57              if (mus(k) != 0)
58                      result(result_index) = mus(k);
59                      result_index = result_index + 1;
60              end
```

```
61      end
62 end
```

2 - EJECUCIÓN

Ejemplo de ejecución para obtener las matrices A, B y C.

```
[A, B, C] = ABC(400, 0.01)
```

Ejemplo de ejecución para obtener los mu.

```
m = mus(C, 0.01, 4)
```