Centro de Investigación en Matemáticas A. C.

Maestría en Ciencias con especialidad en Computación y Matemáticas Industriales

Métodos Numéricos (2019)

Tarea 05 – Solvers iterativos, raíces de funciones y eigensolvers

Reporte

Marco Antonio Esquivel Basaldua (MCMESQ)

Los métodos numéricos iterativos consisten en métodos cuya función es aproximarse cada vez más al resultado buscado conforme las iteraciones en el programa van avanzando, entiéndase aquí una iteración como una "vuelta" en un código de programa o parte de él.

En el presente reporte de tarea se muestran y explican los métodos iterativos de Jacobi y Gauss-Seidel para la solución de sistemas de ecuaciones lineales, la localización de raíces o ceros en una función matemática con los métodos de la bisección y el método de Newton-Raphson, y el método de la potencia para encontrar el máximo eigenvalor de una matriz cuadrada y su aigenvector asociado.

Los códigos utilizados para estas aplicaciones están desarrollados en el lenguaje C y creados y compilados en el entorno de desarrollo Visual Studio Code.

1. Método de Jacobi para la solución de sistemas de ecuaciones lineales

Al tener un sistema de ecuaciones lineales el objetivo principal es obtener los valores de las variables x_i que dan solución a cada una de las ecuaciones en el sistema. Para el presente trabajo se van a considerar únicamente sistemas de ecuaciones con el mismo número de incógnitas que de ecuaciones. Dicho sistema de ecuaciones se puede representar en forma de multiplicación de matrices de la forma Ax = b, con la matriz A como la matriz de coeficientes, el vector x como el vector de incógnitas y el vector b como el vector de soluciones a cada una de las ecuaciones:

Sistema de ecuaciones lineales:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \cdots + a_{2n}x_n = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n = b_3$$

$$a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n$$

Su representación en la forma de multiplicación de matrices se da de la siguiente forma:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

Dadas las n ecuaciones lineales, el método de Jacobi determina el valor de las incógnitas x de la siguiente forma y por filas:

$$x_1 = \frac{b_1 - (a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n)}{a_{11}}$$

$$x_2 = \frac{b_2 - (a_{21}x_1 + a_{23}x_3 + \cdots + a_{2n}x_n)}{a_{22}}$$

$$x_3 = \frac{b_3 - (a_{31}x_1 + a_{32}x_2 + \cdots + a_{3n}x_n)}{a_{33}}$$

De forma general:

$$x_n = \frac{b_n - (a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{n(n-1)}x_{n-1})}{a_{nn}}$$

(Nota: en la implementación de los códigos en C hay que tomar en cuenta que los índices comienzan en 0 y terminan en n-1)

Ya que es evidente que para calcular cada una de las incógnitas es necesario de las demás, este método calcula los valores de las incógnitas un determinado número de veces, o iteraciones, y en cada una de ellas se logra una mejor aproximación de los valores de las incógnitas, para esto es necesario inicializar los valores de todas las x_i siendo una buena aproximación inicializar con los valores del vector b.

El código generado para implementar este método debe determinar el número de iteraciones máximas que los cálculos serán llevados a cabo para llegar a las soluciones. Sin embargo, no es necesario que en cada implementación se utilicen todas las iteraciones. Existe una condición adicional para detener la ejecución y mostrar los resultados obtenidos, esta condición consiste en considerar los resultados de las incógnitas en la iteración actual y la iteración anterior como dos vectores diferentes y calcular la diferencia entre las normas de dichos vectores, si esta diferencia en valor absoluto es menor o igual a una tolerancia cercana a cero determinada al inicio de la ejecución del programa.

Existe la posibilidad que la implementación requiera de todas las iteraciones posibles y que al terminar la diferencia entre las normas de los vectores antes mencionados sea mayor que la tolerancia establecida. En este caso se dirá que las soluciones no convergen y no habrán resultados concluyentes con la implementación. Una forma de evitar este escenario es asegurarse que la matriz de coeficientes en el sistema de ecuaciones sea diagonalmente dominante.

Algoritmo:

- A -> matriz de coeficientes
- b -> vector de soluciones a las ecuaciones lineales
- xO -> vector con valores de las incógnitas calculadas en la iteración anterior

- xN -> vector con valores de las incógnitas calculadas en la iteración actual
- se define el valor de la tolerancia y el máximo de iteraciones permitidas

- *Inicializar valores de xO*
- *Mientras no se alcance el máximo de iteraciones y la diferencia entre las normas sea mayor que la tolerancia*
    - *Para i=1 hasta n*
        - *sum = 0*
        - *para j = 1:n*
            - *sum += A[i][j] * xO[j]*
        - *end*
        - *xN[i] = (b[i][0]-sum)/A[i][i];*
    - *end*
    - *Calcular diferencia* en valor absoluto *entre normas*
    - *Si la diferencia es mayor que la tolerancia*
        - *xO toma los vaores de xN*
    - *Si la diferencia es mayor que la tolerancia*
        - *xN son las soluciones de las incognitas*
- *end*

(Nota: en la implementación de los códigos en C hay que tomar en cuenta que los índices comienzan en 0 y terminan en n-1)

2. Método de Gauss-Seidel para la solución de sistemas de ecuaciones lineales

El método de Gauss-Seidel es un método muy similar al método de Jacobi, con la única diferencia que para calcular las incógnitas desde x_2 hasta x_n se utilizan los valores de las x_i calculadas hasta ese entonces sobre la misma iteración (el método de Jacobi tomaba todos los valores de las incógnitas a partir de los resultados de la iteración anterior), solamente para x_1 se consideran todos los valores de las incógnitas de la iteración anterior. Ya que los resultados obtenidos sobre una misma iteración resultan ser mejores aproximaciones a los resultados buscados, el método de Gauss-Seidel converge (encuentra las soluciones de las incógnitas) en menos iteraciones que el método de Jacobi.

De igual forma que en el método anterior, en el presente, un criterio de paro de la ejecución del programa es calcular la diferencia en valor absoluto entre las normas de los vectores con las soluciones a las incógnitas de la iteración anterior y la actual y evaluar si esta es menor a una tolerancia cercana a cero definida antes de iniciar las iteraciones. Para el caso de este método no es necesario que la matriz de coeficientes sea diagonalmente dominante, como en el caso del método anterior.

Algoritmo:

- A -> matriz de coeficientes
- b -> vector de soluciones a las ecuaciones lineales
- xO -> vector con valores de las incógnitas calculadas en la iteración anterior

- xN -> vector con valores de las incógnitas calculadas en la iteración actual
- se define el valor de la tolerancia y el máximo de iteraciones permitidas

- *Inicializar valores de xO y xN*
- *Mientras no se alcance el máximo de iteraciones y la diferencia entre las normas sea mayor que la tolerancia*
    - *Para i=1 hasta n*
        - *sum = 0*
        - *para j = 1:n*
            - *sum += A[i][j] * xN[j]*
        - *end*
        - *xN[i] = (b[i][0]-sum)/A[i][i];*
    - *end*
    - *Calcular diferencia* en valor absoluto *entre normas*
    - *Si la diferencia es mayor que la tolerancia*
        - *xO toma los valores de xN*
    - *Si la diferencia es mayor que la tolerancia*
        - *xN son las soluciones de las incógnitas*
- *end*

(Nota: en la implementación de los códigos en C hay que tomar en cuenta que los índices comienzan en 0 y terminan en n-1)

3. Método de bisección para encontrar raíces de funciones

Dado un intervalo de valores en el eje x, limitado por los valores *l_max* y *l_min*, y una función definida sobre el mismo, una forma de encontrar una raíz entre ese intervalo, si existe, es evaluar la función en el punto intermedio del intervalo, en el punto *l_med*, si la función evaluada en l_med es igual a cero se encontró la ubicación de la raíz buscada, en caso de que no se igual a cero se procede con lo siguiente:

- Si la función evaluada en l_min y l_med tienen mismo signo, l_min se actualiza con el valor de l_med.
- En caso contrario l_max se actualiza con el valor de l_med.

Ya que este proceso acorta el intervalo en el eje x con el que se va trabajando, los pasos anteriores se pueden repetir un cierto número de veces hasta encontrar la raíz de la función. Debido a que se trabaja bajo un cierto número de repeticiones de instrucciones, este método también es considerado un método iterativo, por tanto se debe especificar una cantidad de veces que las iteraciones se deben repetir antes de mostrar resultados o desistir en buscar las soluciones.

De igual forma a los métodos anteriormente citados se puede añadir un criterio adicional para detener la ejecución del programa. Este criterio consiste en definir una tolerancia cercana a cero, si la función evaluada en l_med para alguna de las iteraciones es igual o menor a esta tolerancia se detiene el programa y l_med se convierte en la raíz buscada.

Para el presente trabajo se dan distintas funciones y el intervalo en x para trabajar en cada una de ellas:

- $f(x) = x^2 \quad x \in [-10, -1]$
- $f(x) = x^2 - 2 \quad x \in [-10{,}10]$
- $f(x) = \sin(x) \quad x \in [-\pi, \pi]$
- $f(x) = \frac{1}{x^2} \quad x \in [-1{,}1]$
- $f(x) = x^3 + 3x^2 + 2x \quad x \in [-3{,}3]$
- $f(x) = \frac{1}{x^2} \quad x \in [-10000{,}10000]$

El usuario debe elegir con cuál de las funciones anteriores se trabajará y a partir de ella se buscará una de sus raíces.

El código desarrollado es capaz, además, de graficar la función elegida y la raíz encontrada y guardar la gráfica en un archivo de imagen png.

Algoritmo:

- *Se selecciona la función con la que se va a trabajar*
- *Mientras no se alcance el máximo de iteraciones y la función evaluada en l_med sea mayor que la tolerancia*
    - *l_med = (l_max + l_min)/2*
    - *evaluar la función en el valor de l_med*
    - *si este valor es menor o igual a la tolerancia*
        - *l_med es el valor en el eje x donde se encuentra la raíz*
    - *si se alcanzó el máximo de iteraciones*
        - *no se encontró la raíz*
    - *en otro caso*
        - *si la funcion evaluada en l_min y l_med tiene mismo signo*
            - *l_min se actualiza con el valor de l_med*
        - *en caso contrario*
            - *l_max se actualiza con el valor de l_med*
    - *end*
- *end*

Observaciones:

El método iterativo presentado en este trabajo solo es capaz de encontrar una raíz, si existe, por cada intervalo propuesto debido a la misma naturaleza de corte del método. Para encontrar todas las raíces de una función se debe modificar el programa para que trabaje sobre distintos intervalos propuestos cercano a cada raíz y es importante que entre cada intervalo dado solo exista una raíz.

4. Método de Newton-Raphson para encontrar raíces de funciones

El método de Newton-Raphson es un método iterativo más para la localización de raíces de una función. Dado un valor inicial en el eje x cercano a una raíz llamado x0, este método aprovecha la definición de la derivada para crear la tangente que pasa por el valor de la función evaluada en el valor inicial y toma su

cruce por cero como el valor de x1 que será utilizado de igual forma que x0 para calcular x2 y acercarse cada vez más a la raíz de la función. Este proceso se puede repetir un número n de veces para encontrar la raíz. Este método se puede apreciar en la figura 1.

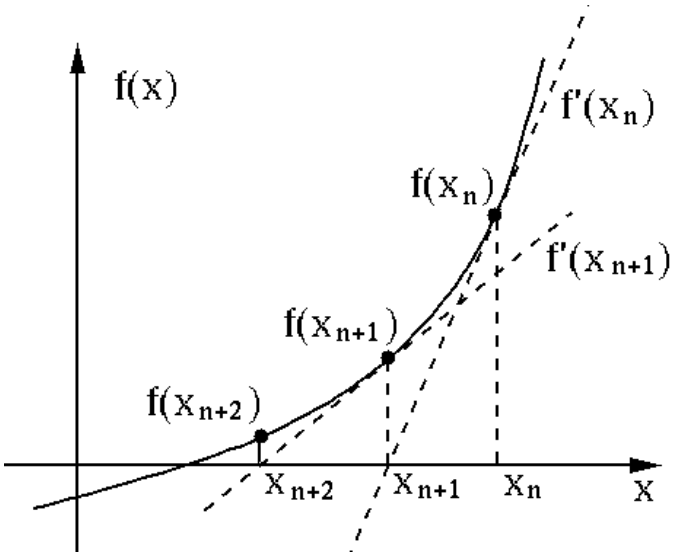

Figure 1

El valor de cada x se calcula como:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

Al igual que el método de la bisección, se puede añadir un criterio adicional para detener la ejecución del programa. Este criterio consiste en definir una tolerancia cercana a cero, si la función evaluada en x_i para alguna iteración i es igual o menor que la tolerancia, entonces se detiene la ejecución del programa y x_i es considerada la raíz de la función. De igual forma, si se ejecuta el programa una cantidad de iteraciones máxima definida y la función evaluada es mayor a la tolerancia entonces se dice que no se ha encontrado la raíz.

Para el presente trabajo, igual que en el método por bisección, se dan distintas funciones y el intervalo en x para trabajar en cada una de ellas:

- $f(x) = x^2 \quad x \in [-10, -1]$
- $f(x) = x^2 - 2 \quad x \in [-10,10]$
- $f(x) = \sin(x) \quad x \in [-\pi, \pi]$

- $f(x) = \frac{1}{x^2} \quad x \in [-1,1]$
- $f(x) = x^3 + 3x^2 + 2x \quad x \in [-3,3]$
- $f(x) = \frac{1}{x^2} \quad x \in [-10000,10000]$

El usuario debe elegir con cuál de las funciones anteriores se trabajará y a partir de ella se buscará una de sus raíces, en caso de encontrar una raíz que no se encuentre en el intervalo establecido, el programa lo notificará.

El código desarrollado es capaz, además, de graficar la función elegida y la raíz encontrada y guardar la gráfica en un archivo de imagen png.

Algoritmo:

- *Se selecciona la función con la que se va a trabajar*
- *Se elige un valor inicial para xold (este valor puede ser 0)*
- *Mientras no se alcance el máximo de iteraciones*
    - *xnew = xold - f(xold)/f'(xold)*
    - *y_xnew = f(xnew)*
    - *si y_new es igual o menor a la tolerancia*
        - *xnew es la raíz*
        - *se detiene la ejecución del programa*
    - *si se alcanzó el máximo de iteraciones*
        - *no se encontró la raíz*
    - *en otro caso*
        - *xold toma el valor de xnew*
- *end*

Observaciones:

El método iterativo presentado en este trabajo solo es capaz de encontrar una raíz, si existe, por cada valor inicial de x propuesto debido a la misma naturaleza de aproximación del método. Para encontrar todas las raíces de una función se debe modificar el programa para que sean ingresados tantos valores de inicialización de x como raíces en la función. Es importante que los valores de inicialización sean cercanos a las raíces a localizar.

5. Método de la potencia para encontrar el valor y vector propio más grande de una matriz

El método de las potencias es un método iterativo que calcula sucesivas aproximaciones al autovalor màs grande de una matriz y su autovector asociado.

Para aplicar este método se tiene una matriz cuadrada de nxn y con n valores propios, cada uno con su correspondiente vector propio.

El método comienza por tomar cualquier vector V0 inicializado aleatoriamente, en cada paso i se calcula:

$$V_{i+1} = A * V_i$$

$$\lambda_i = \frac{V_{i+1}^2}{V_{i+1} * V_i}$$

Entonces V1 converge al autovector de mayor autovalor.

El método termina cuando se cumplan todas las iteraciones establecidas o bien cuando la deferencia entre el autovalor generado en la iteración actual y el generado en la iteración anterior sea menor o igual que una tolerancia establecida cercana de 0.

Algoritmo:

- *Se inicializa V0 con valores aleatorios*
- *Se inicializa lambda_old a cero*
- *Se define la tolerancia*
- *Mientras no se alcance el máximo de iteraciones*
    - Normalizar V0
    - V1 = A*V0
    - Lambda = $V1^2/(V0*V1)$
    - Si la diferencia en valor absoluto entre lambda y lambda_old es menor o igual a una tolerancia
        - lambda es el autovalor más grande
        - V1 es el autovector más grande
    - En otro caso
        - lambda_old = lambda
        - V0 = V1
    - End
- End