Centro de Investigación en Matemáticas A.C.

Maestría en Ciencias con especialidad en Computación y Matemáticas Industriales
Métodos Numéricos (2019)

Tarea 07 - Eigensolvers y Solvers Iterativos

Reporte

Marco Antonio Esquivel Basaldua (MCMESQ)

En el presente reporte se muestran y explican los métodos iterativos del algoritmo de Rayleigh, el algoritmo de iteración en subespacio ya la fatorización QR para encontrar valores y vectores propios. De igual forma se presenta el método de gradiente conjugado para la solución de sistemas de ecuaciones lineales.

1. Algoritmo de Rayleigh

El método de Rayleigh o también llamado método del cociente de Rayleigh es muy similar al método de la potencia inversa pero que presenta un desplazamiento en cada iteración. El objetivo de este método es hacer que el residuo sea lo más pequeño posible en cada iteración. El residuo se define como:

$$r = ||\lambda x - Ax||$$

Donde λ y x son el valor y el vector propio aproximados que se calculan en cada iteración. El objetivo del método de Rayleigh es encontrar un valor para λ para el que se hace mínimo el residuo, es decir, se quiere encontrar

$$min_{\alpha \in \mathbb{C}} ||\lambda x - Ax||$$

Por lo tanto el valor que minimiza el residuo es:

$$\lambda_0 = \frac{x^T Ax}{x^T x}$$

A este valor se le conoce como cociente de Rayleigh:

$$\rho(x, A) = \frac{x^T Ax}{x^T x}$$

entonces el cociente de Rayleigh de A en x es el número donde se alcanza el mínimo residuo:

$$min_{\alpha \in \mathbb{C}} ||\lambda x - Ax|| = ||\rho(x, A)x - Ax||$$

Algoritmo

- Dar un vector inicial x_0 normalizado, una tolerancia τ y un numero máximo de iteraciones M
- calcular $\rho_0 = \rho(x_0, A)$
- Para $k = 1, \ldots, M$
  - Resolver $(A - \rho_{k-1} I) y_k = x_{k-1}$
  - $x_k = \frac{y_k}{||y_k||}$
  - $\rho_k = \rho(x_k, A)$
  - Terminar de iterar si $||Ax_k - \rho_k x_k|| < \tau$
- Devolver x_k y ρ_k como una estimación de un eigenpar de A

Resultados

El vector x_0 indicado en el algoritmo es inicializado con valores aleatorios por lo que en cada ejecución del programa se va a converger al valor y vector propio más cercanos a ese vector aleatorio. Para coroborar que se haya convergido en un eigenpar válido se pueden buscar los valores en los archivos generados por el método de Jacobi realizado como parte de la Tarea 06.

En seguida se muestra un ejemplo de los resultados obtenidos al implementar el archivo "M_BIG.txt" en el algoritmo, por razones de espacio no se muestra el vector propio generado aún así se puede visualizar su resultado por la pantalla de la terminal una vez ejecutado el programa.

```
Matrix A:
Rows: 125, Columns: 125
Matrix A is too large to be displayed

Iterations taken: 8
Found Eigenvalue:
 0.000098
Eigenvector assotiated to found eigenvalue is:
-0.2839248634
0.2261835791
1.152864763e-49
0.1618074016
-0.01735968529
-0.1574187743
-0.008363502088
0.01587946548
9.393712884e-50
0.0101402225
0.1454729137
-0.1284317718
-0.01174834566
-0.01719659667
0.1648997781
-0.04250714666
0.0266112368
0.07056252106
-0.1317566105
-0.0199316068
-0.02445897933
-0.05911945522
0.04464063845
0.09612920296
1.124398966e-49
-0.003615283765
0.03330944103
```

El tiempo ejecución del algoritmo para este ejemplo fue de 93.803 milisegundos. Debido a que se inicializa con un vector aleatorio este valor puede oscilar entre los 50 y los 150 milisegundos.

Conclusiones

Debido a que se inicia el proceso mediante un vector aleatorio x_0 no se sabe con certeza hacia qué valor y vector propio se convergirá y tampoco se tiene una medida firme del tiempo que toma el algoritmo para encontrar dichos resultados, una mejora o alternativa a esto es solicitar al usuario ingresar una aproximación al vector propio que se esté buscando. Para el caso de la matriz "M_BIG.txt", ya que se trata de una matriz grande e ingresar los datos por la terminal puede ser tedioso, se puede ingresar un nuevo archivo de texto al programa que contenga la aproximación al vector propio que se esté buscando.

2. Iteración en el subespacio

El objetivo de este método es determinar en forma simultanea los K vectores propios asociados a los valores propios de mayor módulo. La idea básica es que es mucho más fácil iterar para obtener un subespacio que contenga a estos vectores que iterar para obtener a cada uno de ellos por separado.

Algoritmo

- Dar la matriz cuadrada A de dimensión n, una tolerancia τ y un máximo de iteraciones M
- Dar una matriz P de dimensión $n \times K$ con entradas aleatorias
- Para $i = 1...M$
    - Factorizar P en Q y R: $P = QR$
    - $P = AQ$
    - $\Lambda = Q^T P$
    - Si Λ es diagonal
        - Terminar de iterar
- Normalizar los vectores en P
- Regresar la diagonal de Λ como valores propios y las columnas de P como vectores propios

Resultados

Al ejecutar el algoritmo con un valor $K = 15$, se obtiene los siguientes resultados para los valores propios:

Eigenvalues are:
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
1.000000
0.000598

0.000581
0.000561

Con un tiempo de ejecución que va de los 500 milisegundos al segundo, aproximadamente. Esto es debido a que se trabaja con una matriz de valores inicializados aleatoriamente.

Para una mejor visualización y manejo de los datos generados, los valores y vectores propios son almacenados en archivos de texto con nombre : "Eigenvalues.txt" y "Eigenvectors.txt".

Conclusiones

El método de iteraciones en el subespacio es un buen método para localizar los valores y vectores propios más grandes de una matriz cuadrada, tomando en cuenta su tiempo de ejecución, aunque este puede variar un poco debido a que se trabaja con una matriz de valores aleatorios.

3. Algoritmo QR para valores y vectores propios

La fatorización QR de una matriz es una descomposición de la misma como un producto de una matriz ortonormal por una triangular superior. Para asegurar que la fatorización arroje como resultado una matriz de vectores ortonormales en Q se utiliza el algoritmo modificado de Gram Schmidt. Esta descomposición es utilizada posteriormente para encontrar los valores y vectores propios de la matriz que inicialmente fue factorizada, para encontrarlos, ya que se cuenta con vectores ortonormales, se procede de una manera iterativa muy similar al método de Jacobi.

Algoritmo

Para realizar la fatorización QR, utilizando el algoritmo de Gram Schmidt modificado:

- Dar una matriz A simétrica de tamaño n para factorizar en $Q * R$
- Para $k = 0...(n-1)$
    - $sum = 0$
    - Para $j = 0...(n-1)$
        - $sum = sum + a_{jk}^2$
    - $r_{kk} = sqrt(s)$
    - Para $j = 0...(n-1)$
        - $q_{jk} = a_{jk} / r_{kk}$
    - Para $i = (k+1)...(n-1)$
        - $sum = 0$
        - Para $j = 0...(n-1)$
            - $sum = sum + a_{ji} * q_{jk}$
        - $r_{ki} = sum$
        - Para $j = 0...(n-1)$
            - $a_{ji} = a_{ji} - r_{ki} * q_{jk}$
- Regresar las Matrices Q y R como resultado de la fatorización

Para encontrar los valores y vectores propios

- Dar la matriz A y una matriz FI de mismo tamaño de A para guardar los vectores propios, los valores propios se sobre escriben en A
- Dar el máximo de iteraciones M
- Para $i = 1...M$
    - Factorizar A en Q y R
    - $A = R * Q$
    - $FI = Q * FI$
    - Terminar de iterar si A es diagonal
- Regresar la diagonal de A como los valores propios y FI como los vectores propios

Resultados

Dadas las dimensiones de la matriz "M_BIG.txt" los resultados, en lugar de ser mostrados por consola, son guardados en dos archivos de texto, "Eigenvalues.txt" y "Eigenvectors.txt", ubicados en la carpeta donde el archivo ejecutable fue generado. De esta manera es mucho más sencillo recuperar los valores y visualizarlos.

El tiempo de ejecución del algoritmo fue de 294636.838 milisegundos (aproximadamente cinco minutos) el cual es un tiempo de relativamente grande comparado a otros métodos y, obviamente, a matrices más pequeñas. Esto es debido a que la matriz debe pasar por el algoritmo de fatorización en cada iteracion en el algoritmo para encontrar los valores y vectores propios. La complejidad del algoritmo es entonces de n^3

Conclusiones

Debido al tiempo de ejecución es más recomendable utilizar el algoritmo de Jacobi, ya que trabajan bajo criterios muy similares (Multiplicación de matrices para encontrar los vectores propios y diagonalizar la matriz A para encontrar los valores propios) pero el tiempo de ejecución para el algoritmo de Jacobi es de solo 908.8 milisegundos (cerca de un segundo). Otra alternativa es encontrar formas de optimizar los procedimientos en el algoritmo QR, sobre todo la multiplicación de matrices que es lo más costoso computacionalmente.

4. Gradiente Conjugado para Solución de Sistemas de Ecuaciones

Este método se aplica para solucionar sistemas de ecuaciones lineales de la forma $Ax = b$ donde A es simétrica y definida positiva. El método de gradiente conjugado está basado en un principio de conjugación teniendo un almacenamiento modesto y siempre converge.

Dos vectores son conjugados si son ortogonales con respecto al producto interior $< u, Av >= u^T Av$. La conjugación es una relación simétrica: si u es conjugado a v entonces v es conjugado a u. Supongamos que $\{p_k\}$ es una secuencia de n direcciones mutuamente conjugadas. Entonces los p_k forman una base de R^n por lo tanto se puede extender la solución de x^* de $Ax = b$ en esta base:

$$x^* = \sum_{i=1}^{n} \alpha_i p_i$$

Los coeficientes están dados por:

$$b = Ax^* = \sum_{i=1}^{n} \alpha_i A p_i$$

$$p_k^T b = p_k^T A x^* = \sum_{i=1}^{n} \alpha_i p_k^T A p_i = \alpha_k p_k^T A p_k$$

$$\alpha_k = \frac{p_k^T b}{p_k^T A p_k} = \frac{< p_k, b >}{< p_k, p_k >_A} = \frac{< p_k, b >}{||p_k||^2}$$

Esto da el método para resolver $Ax = b$. Primero se encuentra una secuencia de n direcciones conjugadas y luego se computan los coeficientes de α_k.

Algoritmo

- Dar vectores de inicio x_0, r_0 y p_0. Dar un valor máximo de iteraciones M
- Iniciar el vector x_0 con ceros y los vectores r_0 y p_0 con las entradas de b
- Para $k = 0...M$
  - $\omega = Ap_k$
  - $\alpha = \frac{p_k^T r_k}{p_k^T \omega}$
  - $x_{k+1} = x_k + \alpha_k p_k$
  - $r_{k+1} = r_k - \alpha\omega$
  - Si la norma de r_{k+1} es menor que la tolerancia
    - Dejar de iterar
  - Si no
    - $\beta = \frac{p_k^T r_{k+1}}{p_k^T p_k}$
    - $p_{k+1} = r_{k+1} + \beta p_k$
- Regresar x_{k+1} como la solución de las incógnitas

Resultados

Al ejecutar el programa con los archivos de texto "M_BIG.txt" y "V_BIG.txt" que corresponden a la matriz de coeficientes y al vector de soluciones a las ecuaciones lineales del sistema $Ax = b$ se tiene un tiempo de ejecución de 1103.962 milisegundos. Los resultados son desplegados en la pantalla de la terminal para ser visualizados. Al comparar estos resultados con algoritmos ya validados para la solución de sistemas de ecuaciones, por ejemplo el algoritmo de Doolittle, se puede asegurar la funcionalidad del método de gradiente conjugado.

Conclusiones

Al comparar el tiempo de ejecución de este método con métodos aplicados con anterioridad se puede apreciar que se trata de un método más tardado. Por ejemplo, con el algoritmo de Doolittle el sistema es resuelto en 6.893 milisegundos. Teniendo en cuenta además el limitante de que es un método que solo se aplica para matrices simétricas y definidas positivas se puede concluir que existen métodos más eficientes que el del gradiente conjugado.