



Centro de Investigación en Matemáticas, A.C.

Descripción tarea - Métodos numéricos

Erick Salvador Alvarez Valencia

8 de Octubre de 2017

1. Introducción

2. Método de potencia inversa

2.1. Descripción

El primer método a describir es el de la potencia inversa, el cuál nos ayuda a encontrar el eigenvalor más pequeño, aunque con una pequeña modificación puede encontrar algún valor que sea muy cercano a un δ dado.

Sea $A \in R^{n \times n}$ una matriz no necesariamente simétrica, estamos interesados en encontrar un eigenpar (ϕ, λ) , aplicando el método de la potencia inversa encontraremos un eigenpar tal que el valor de λ sea el más pequeño que existe. Ahora definimos a δ como un valor que está dentro del rango $[-d, d]$ donde $d = \|A\|_\infty$. Podemos ver entonces que si sumamos δ a la matriz A obtenemos:

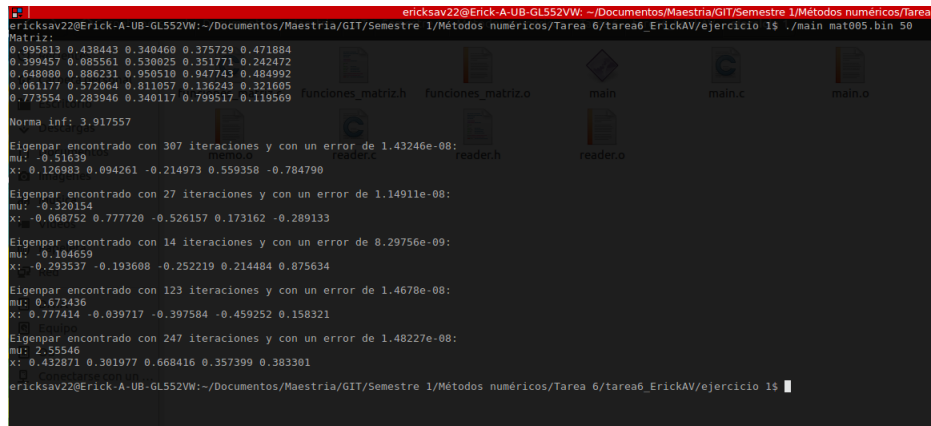
$$(A + \delta I)\phi = A\phi + \delta\phi = \lambda\phi + \delta\phi = (\lambda + \delta)\phi$$

De lo anterior se obtiene un nuevo eigenvalor asociado al mismo eigenvector. Por lo que podemos concluir que eligiendo valores de δ apropiados, podremos encontrar posiblemente todos los eigenpares de la matriz mediante el método de la potencia.

En el código proporcionado se realiza una discretización uniforme dentro del rango $[-d, d]$ con un tamaño de paso Δd , dicha constante la obtenemos como $\Delta d = 2d/N$, el usuario brindará el valor de N al programa. Ahora por cada valor δ_i y con una cierta tolerancia τ llamamos a la función de la potencia inversa, la cual nos devolverá un eigenvalor μ_i y su eigenvector ϕ_i asociado, si se cumple que $|\mu_0 - \mu_i| > 0,0001$ entonces podemos considerar a μ_i y a su respectivo ϕ_i como un eigenpar válido. Para asegurarnos de tener una muy buena probabilidad de encontrar todos los eigenpares de una matriz, ejecutamos el método de la potencia con un máximo de 1000 iteraciones.

2.2. Ejemplo de ejecución

A continuación se mostrará el resultado de la ejecución del método de la potencia con una matriz de 5×5 .



```
ericksav22@Erick-A-UB-GL552VW: ~/Documentos/Maestria/GIT/Semestre 1/Métodos numéricos/Tarea 6
Matriz:
0.995813 0.438443 0.348468 0.375729 0.471884
0.399457 0.085561 0.538025 0.351771 0.242472
0.648880 0.886231 0.350510 0.947723 0.484892
0.061177 0.572864 0.811057 0.136243 0.321695
0.773554 0.283946 0.348117 0.799517 0.119569
Norma inf: 3.917557
Eigenpar encontrado con 307 iteraciones y con un error de 1.43246e-08:
mu: -0.51639
x: 0.126983 0.094261 -0.214973 0.559358 -0.784790
Eigenpar encontrado con 27 iteraciones y con un error de 1.14911e-08:
mu: -0.320154
x: -0.068752 0.777720 -0.526157 0.173162 -0.289133
Eigenpar encontrado con 14 iteraciones y con un error de 8.29756e-09:
mu: -0.104659
x: -0.293537 -0.193688 -0.252219 0.214484 0.875634
Eigenpar encontrado con 123 iteraciones y con un error de 1.4678e-08:
mu: 0.673436
x: 0.777414 -0.039717 -0.397584 -0.459252 0.158321
Eigenpar encontrado con 247 iteraciones y con un error de 1.48227e-08:
mu: 2.55546
x: 0.432871 0.301977 0.668416 0.357399 0.383301
ericksav22@Erick-A-UB-GL552VW: ~/Documentos/Maestria/GIT/Semestre 1/Métodos numéricos/Tarea 6/tarea6_ErickAV/ejercicio 1$
```

(a) Figura 1. Ejecución del algoritmo de la potencia inversa con 50 particiones.

En la Figura 1. podemos observar que el programa se ejecutó con una matriz de tamaño 5×5 y con 50 particiones. Lo primero que podemos observar es que el algoritmo nos arrojó todos los eigenpares existentes y el error obtenido en los mismos, dicho error es muy pequeño por lo que podemos concluir que la calidad de los resultados es buena.

2.3. Compilación y ejecución

Para compilar: En la carpeta encontraremos los archivos `.c` y `.h` con los que se podrá compilar el ejecutable. De la misma forma, en conjunto con los archivos anteriores, también podremos encontrar un Makefile para, en caso de encontrarse en linux, compilar de manera sencilla.

1. **Compilar usando Makefile:** En la terminal, nos colocamos en el directorio donde se encuentre el programa, y ejecutamos el comando `make`, automáticamente se realizará la compilación y se generará el ejecutable. El Makefile también contiene el comando `makeclean` el cual limpiará los archivos generados por la compilación, incluyendo el ejecutable.
2. **Compilar directamente:** De la misma forma, podemos compilar directamente usando los siguientes comandos (en terminal):

- `gcc -c main.c`
- `gcc -c memo.c`
- `gcc -c funciones_matriz.c`
- `gcc -c reader.c`
- `gcc -o main main.o memo.o funciones_matriz.o reader.o -lm`

Para ejecutar: Únicamente debemos de usar el comando `./main` para ejecutar el programa en consola.

El programa recibe dos argumentos: Un string que representa el nombre del archivo binario (incluida la extensión) donde se encuentra la matriz; Un entero que representa el número de particiones que se realizará en la discretización. A continuación se mostrará un ejemplo de cómo ejecutar el programa:

```
./main mat005.bin 50
```

3. Método de Jacobi para eigenvalores

3.1. Descripción

Este método también nos va a servir para encontrar todos los eigenpares de una matriz $A \in R^{n \times n}$ la cual si debe cumplir la restricción de ser simétrica. La idea de este método parte de la siguiente igualdad $A\Phi = \Phi\Lambda$ donde Φ es una matriz que contiene todos los eigenvectores de A y Λ es una matriz diagonal que contiene los eigenvalores asociados a su propio eigenvector. Para obtener lo anterior usamos un tipo de matrices, las cuales llamaremos rotaciones de Givens, las cuales son matrices de tamaño $n \times n$ que son iguales a la identidad, con la excepción que en las posiciones $[i, j]$, $[j, i]$, $[i, i]$, $[j, j]$ contienen senos y cosenos que permiten realizar una rotación. Los índices i, j son las posiciones donde encontraremos al elemento más grande en la matriz A , la idea es convertir en cero ese elemento, pero hay que recordar que estaremos afectando dos filas y dos columnas para lograr esto, por lo que si ya habíamos convertido a cero en otro elemento en alguna de las filas y columnas afectadas mediante una rotación, esto se perderá, pero a su vez se generará un número pequeño, entonces podemos hacer muchas iteraciones hasta lograr convertir en ceros (o números muy cercanos a cero) todos los elementos que no se encuentren en la diagonal de la matriz.

3.2. Ejemplos de ejecución

A continuación se mostrará un ejemplo de ejecución del método de Jacobi para una matriz de 8×8 con un total de 20000 iteraciones propuestas.

(b) Figura 2. Ejecución del algoritmo de Jacobi con una matriz 5x5.

(b) Figura 2. Ejecución del algoritmo de Jacobi con una matriz 5x5.

Podemos observar en la Figura 1. que pese a que se propusieron 20000 iteraciones como límite sólo se necesitaron 86 para encontrar todos los eigenpares de la matriz con un error del orden de menos quince.

Ahora se mostrará una parte del resultado de la ejecución del mismo algoritmo aplicado a una matriz de tamaño 100x100 con un total de 20000 iteraciones.

[illegible]

(c) Figura 3. Ejecución del algoritmo de Jacobi con una matriz 100x100.

Para este caso la imagen genera un resultado muy abrumador ya que se están mostrando todos los eigenvectores encontrados, y pese a que no se muestra, se necesitaron un total de 17254 iteraciones para que el algoritmo terminara, y esto anterior con un error del orden de menos trece.

3.3. Compilación y ejecución

Para la compilación se proporcionan las mismas instrucciones que en el programa anterior, se cuenta con un archivo Makefile o con los mismos comandos de compilación del método de la potencia inversa.

Para ejecutar: Únicamente debemos de usar el comando `./main` para ejecutar el programa en consola.

El programa recibe dos argumentos: Un string que representa el nombre del archivo binario (incluida la extensión) donde se encuentra la matriz; Un entero que representa el número máximo de iteraciones que realizará el algoritmo.

A continuación se mostrará un ejemplo de cómo ejecutar el programa:

```
./main mat008.bin 20000
```