Tarea 8

Jonathan Alejandro Casas Bocanegra
Sergio David Sierra Marín

Marzo 2019

1. Punto 1

Enunciado

Dado el sistema $Ax = b$, describa un algoritmo y desarrolle un programa que permita ingresar A, b, el vector inicial $x^{(0)}$ y la precisión ϵ para:

1. Determinar el valor de ω (use una sola cifra decimal) del método de relajación que genera el menor número de iteraciones.
2. Repita el ejercicio anterior usando el método SOR.

Implementación

Con el fin de llevar a cabo la implementación de los puntos descritos previamente, es necesario plantear un algoritmo que permita encontrar el valor de ω que genere el menor número de iteraciones para el método de relajación y el método de sobre-relajación. Partiendo del enunciado, es posible destacar un criterio de diseño importante correspondiente a la resolución para encontrar ω, el cual debe ser encontrado con una resolución de no más de una cifra decimal. Así, se plantea el siguiente algoritmo:

1. Inicializar variables para limitar el valor mínimo y máximo de ω, $limit_{min}$ y $limit_{max}$.
2. Inicializar variable ω con valor inicial igual $limit_{min}$.
3. Inicializar variable $iter_{min}$ en 0 para almacenar el valor de iteraciones mínimas encontrado.
4. Calcular matriz B:
   - Para el método de relajación: $B = A_D$, donde A_D proviene de la descomposición LDR de A.
   - Para el método de sobre-relajación: $B = A_D + A_L$, donde A_D y A_L provienen de la descomposición LDR de A.
5. Calcular matriz de iteración G como $G = I - \omega B^{-1}A$.
6. Realizar el método de iteración general y obtener el número de iteraciones requeridas.
7. Comparar el número de iteraciones con el valor mínimo de iteraciones almacenado previamente. En caso de ser la primera vez, asignar las iteraciones obtenidas así: $iter_{min} := iteraciones$. En caso de obtener un valor de iteraciones menor a $iter_{min}$, almacenar el valor de ω actual como ω_{min}.
8. Aumentar ω en 0.1 y repetir desde el paso 5 siempre que $\omega \leq limit_{max}$. Si $\omega \geq limit_{max}$ continuar.
9. Si $limit_{min} < \omega_{min} < limit_{max}$ ó $\omega_min = limit_{min}$, salir. Si no, asignar $limit_{min} := limit_{max}$, $limit_{max} := limit_{min} + limit_{max}$, $w := limit_{min}$ y repetir desde el paso 5.

Los pasos descritos, permiten realizar la implementación de los métodos de relajación y sobre-relajación. En primer lugar, el algoritmo 1 ilustra un programa en C correspondiente al método de relajación.

```c
double relaxation_method(Matrix_ptr A, Matrix_ptr b, Matrix_ptr x_0, double error){
  double limit_max = 2.0;
  double limit_min = 0.1;
  double w = 0.1;
  double w_min = 0.1;
  int ext = 0;
  int iter_min = 0;
  int iter = 0;
  LDR_Matrix_ptr ldr = LDR_decomposition(A);
  Matrix_ptr B = ldr->D;
  do {
    while (w <= limit_max) {
        iter = general_interation_method_2(A, B, b, x_0, error, w);
        //printf("\n\n %d\n", iter);
        if (iter_min == 0 || iter < iter_min) {
          iter_min = iter;
          w_min = w;
        }
        w = w + 0.01;
    }
    if ((w_min != limit_min && w_min != limit_max) || w_min == limit_min){
      ext = 1;
    } else {
      limit_min = limit_max;
      limit_max = limit_min + 2.0;
      w = limit_min;
      w_min = w;
      iter_min = 0;
      iter = 0;
    }
  } while (ext != 1);
  printf("\n%s\n", "      Utilizando metodo de relajacion" );
  printf("      w optimo: %f\n", w_min );
  printf("      iteraciones: %d\n", iter_min );
  return w_min;
}
```

Algoritmo 1: Método de relajación

Para probar el funcionamiento del método de sobre-relajación, se utilizó el sistema $Ax = b$:

$$A = \begin{pmatrix} 5 & -3 & 0 & 2 \\ 2 & 6 & -3 & 0 \\ -1 & 2 & 4 & -1 \\ -2 & -3 & 2 & 7 \end{pmatrix} \; y \; b = \begin{pmatrix} -12 \\ -18 \\ 9 \\ 13 \end{pmatrix}$$

Tomando como vector inicial $(x^{(0)})^T = (0, 0, 0, 0, 0)$ y una precisión de 10^{-5}, se obtuvieron los resultados presentados en la tabla 1.

ω	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1
iteraciones	104	54	36	28	24	22	22	24	28	36	53

Cuadro 1: Comparación de las iteraciones obtenidas con diferentes valores de ω para el método de relajación.

Finalmente, el algoritmo 2 ilustra un programa en C para el método de sobre-relajación.

```c
double over_relaxation_method(Matrix_ptr A, Matrix_ptr b, Matrix_ptr x_0, double error){
  double limit_max = 2.0;
  double limit_min = 0.1;
  double w = 0.1;
  double w_min = 0.1;
  int ext = 0;
```

```
  int iter_min = 0;
  int iter = 0;
  LDR_Matrix_ptr ldr = LDR_decomposition(A);
  Matrix_ptr B = sum_matrix(ldr->D, ldr->L);
  do {
    while (w <= limit_max) {
        iter = general_interation_method_2(A, B, b, x_0, error, w);
        //printf("\n\n %d\n", iter);
        if (iter_min == 0 || iter < iter_min) {
          iter_min = iter;
          w_min = w;
        }
        w = w + 0.1;
    }
    if ((w_min != limit_min && w_min != limit_max) || w_min == limit_min){
      ext = 1;
    } else {
      limit_min = limit_max;
      limit_max = limit_min + 2.0;
      w = limit_min;
      w_min = w;
      iter_min = 0;
      iter = 0;
    }
  } while (ext != 1);
  printf("\n%s\n", "      Utilizando metodo de sobre-relajacion" );
  printf("      w optimo: %f\n", w_min );
  printf("      iteraciones: %d\n", iter_min );
  return w_min;
}
```

Algoritmo 2: Método de sobre-relajación

Para probar el funcionamiento del método de sobre-relajación, se utilizó el sistema $Ax = b$:

$$A = \begin{pmatrix} 3 & -1 & -1 & 0 & 0 \\ -1 & 4 & 0 & -2 & 0 \\ -1 & 0 & 3 & -1 & 0 \\ 0 & -2 & -1 & 4 & -1 \\ 0 & 0 & 0 & -1 & 2 \end{pmatrix} \quad y \quad b = \begin{pmatrix} 2 \\ -26 \\ 3 \\ 47 \\ -10 \end{pmatrix}$$

Tomando como vector inicial $(x^{(0)})^T = (0, 0, 0, 0, 0)$ y una precisión de 10^{-5}, se obtuvieron los resultados presentados en la tabla 2.

ω	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4
iteraciones	137	70	47	34	27	22	18	15	12	10	8	10	13	17

Cuadro 2: Comparación de las iteraciones obtenidas con diferentes valores de ω para el método de sobre-relajación.

2. Punto 2

Describa un algoritmo y genere un programa para determinar el valor y el vector propio dominantes para cualquier matriz $A = (a_{ij})_{nxn}$

El algoritmo se desarrolló en base al método de potencias, en el cual se tiene $\{\lambda_1, \lambda 2, ..., \lambda_n\}$ como el conjunto de valores propios de la matriz A, en donde $|\lambda_1| > |\lambda_i|$. Se dice que λ_1 es el valor propio dominante junto con el vector propio asociado.

El método de las potencias utiliza una sucesión recursiva, expresada de la siguiente forma:

$$y^{(k)} = Ax^{(k)}$$

$$x^{(k+1)} = \frac{1}{C_{k+1}} y^{(k)}$$

C_{k+1} es la componente de mayor (max $|y_i|$) valor del vector $y^{(k)}$.
Al realizar el proceso iterativo, se tiene que:

$$\lim_{k\to\infty} x^k = v$$
$$\lim_{k\to\infty} C_k = \lambda$$

Donde v y λ son el vector y valor propio dominantes de la matriz A.

El algoritmo desarrollado para encontrar los valores y vectores propios dominantes se muestra en Algoritmo 3. La función *get_eigen_values*() recibe la matriz A, el vector x de condición inicial y el valor de precisión deseados. Este método realiza las iteraciones por medio de un ciclo While, en donde la condición de salida se define como $||x^{k+1} - x^k|| < e$

```
EigenStruct_ptr get_eigen_values(Matrix_ptr A, Matrix_ptr x, double prec ){
  //get matrix size
  int m = A->m;
  int n = A->n;
  //alloc memory for eigen struct
  EigenStruct_ptr eigen = (EigenStruct_ptr)malloc(sizeof(EigenStruct));
  Matrix_ptr y = matrix_alloc(m,1);
  Matrix_ptr x_new = matrix_alloc(m,1);
  double C;
  double err;
  double e = 10e-5;

  int cont = 0;
  do{
    /* code */
    y = mult_matrix(A,x);
    //get max of y
    C = get_max(y);
    x = scalar_mult(x,-1);
    x_new = scalar_mult(y, 1/C);
    Matrix_ptr diff = sum_matrix(x_new, x);
    err = matrix_norm(diff);
    x = x_new;

    printf("Error obtained %f, Error desired %f\n",err,e );
    printf(" C variable %f\n",C );
    cont ++;

  }while(err > e);

  eigen->v = x_new;
  eigen->lambda = C;

  printf("%s\n","##################################" );
  printf("Max eigen value: %f\n", eigen->lambda);
  printf("%s\n", "Eigen vector: ");
  print_matrix(eigen->v);
  printf("Numero de iteraciones: %d\n", cont);
  printf("%s\n","##################################" );
  return eigen;
}
```

Algoritmo 3: Método de iteración general

El Algoritmo 4 muestra la función principal que realiza la petición de la matriz deseada y ejecuta el calculo del Algoritmo 3.

```

typedef struct EigenStruct{
```

```
  Matrix_ptr v;
  double lambda;
}EigenStruct, *EigenStruct_ptr;


int main(){
  printf("%s\n", "INGRESE LA MATRIZ");
  Matrix_ptr A = user_request_matrix();

  printf("%s\n", "INGREASE LA CONDICION INICIAL" );
  Matrix_ptr x = user_request_matrix();


  double e = 10e-5;
  EigenStruct_ptr eigen =  get_eigen_values(A,x,e);

}
```

Algoritmo 4: función main para ejecución del algoritmo

El algoritmo se evaluó con la matriz de prueba A:

$$A = \begin{pmatrix} 0 & 7 & -5 \\ -2 & 17 & -7 \\ -4 & 26 & -10 \end{pmatrix}$$

Resultados del algoritmo se muestran en la Figura 1. Donde se observa que el algoritmo llega al resultado $v = [0{,}400059, 0{,}600039, 1{,}0]^T$ y $\lambda = 4{,}001564$ en un total de 11 iteraciones para un error de 10^{-5}

```
Error obtained 0.600925, Error desired 0.000100
 C variable 12.000000
Error obtained 0.075116, Error desired 0.000100
 C variable 5.333333
Error obtained 0.025039, Error desired 0.000100
 C variable 4.500000
Error obtained 0.010543, Error desired 0.000100
 C variable 4.222222
Error obtained 0.004866, Error desired 0.000100
 C variable 4.105263
Error obtained 0.002341, Error desired 0.000100
 C variable 4.051282
Error obtained 0.001148, Error desired 0.000100
 C variable 4.025316
Error obtained 0.000569, Error desired 0.000100
 C variable 4.012579
Error obtained 0.000283, Error desired 0.000100
 C variable 4.006270
Error obtained 0.000141, Error desired 0.000100
 C variable 4.003130
Error obtained 0.000071, Error desired 0.000100
 C variable 4.001564
################################
Max eigen value: 4.001564
Eigen vector:
0.400059
0.600039
1.000000
Numero de iteraciones: 11
################################
```

Figura 1: Resultado prueba del algoritmo.