Tarea 8

Jonathan Alejandro Casas Bocanegra
Sergio David Sierra Marín

Marzo 2019

1. Punto 1

Enunciado
Dado el sistema $Ax = b$ donde:

$$A := \begin{pmatrix} 5 & -1 & 1 \\ 2 & 8 & -1 \\ -1 & 1 & 4 \end{pmatrix} \quad b = \begin{pmatrix} 10 \\ 11 \\ 3 \end{pmatrix} \tag{1}$$

con $x^{(0)} = (0, 0, 0)^T$

- **1.1** Construya tres matrices B, diferentes de A_D y $A_D + A_L$, de tal forma que $\rho(G) < 1$ donde $G := I - B^{-1}A$.
- **1.2** Usando el método iterativo general, determine el número de iteraciones necesarias para obtener una aproximación a la solución con una presición de 10^{-5} tomando $||x^{(k)} - x^{(k-1)}||_2$.
- **1.3** Repita 1.2 usando el método de Jacobi y Gauss-Seidel.

1.1. Teoría

1.1.1. Método de Jacobi

El método de Jacobi puede escribirse como:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^k \; - \sum_{j=i+1}^{n} a_{ij} x_j^k \right], \; i = 1 \; \ldots \; n, \; k = 0, 1, \ldots \tag{2}$$

La expresión del algoritmo se puede interpretar como la aproximación que se obtiene al despejar x_i. Para este método se puede considerar $B = A_D$, por lo que:

$$x^{k+1} = x^k + A_D^{-1}(b - Ax^k) \tag{3}$$

Reescribiendo, se obtiene:

$$A_D x^{k+1} = A_D x^k + b - Ax^k \tag{4}$$

Utilizando la descomposición aditiva de A y simplificando se llega a:

$$A_D x^{k+1} = b - A_L x^k - A_R x^k \tag{5}$$

lo que corresponde a la forma matricial del algoritmo. Adicionalmente, se tienen algunas observaciones correspondientes a este método:

- ***Observación 1***: El método de Jacobi no se puede utilizar si la matriz tiene algún coeficiente nulo en la diagonal.
- ***Observación 2***: Una condición suficiente para convergencia del método de Jacobi es que A sea diagonalmente dominante. Una matriz A es diagonalmente dominante si se verifica que $|a_i i| > \sum_{j \neq i} \ \forall i$.

1.1.2. Método de Gauss-Seidel

En el método de Jacobi cada componente x_i^{k+1} se calcula utilizando las componentes de la aproximación anterior, x^k. Sin embargo, a la hora de calcular la componente $x^k + 1_i$ ya se han calculado previamente las componentes anteriores. En el método de Gauss-Seidel se aprovechan estas $i1$ componentes para el cálculo de x_i^{k+1}. Así, se puede expresar el cálculo de los coeficientes de x como:

$$x_i^{k+1} = \frac{1}{a_{ii}} \left[b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+i} \ - \ \sum_{j=i+1}^{n} a_{ij} x_j^k \right], \ i = 1 \ \ldots \ n, \ k = 0, 1, \ldots \tag{6}$$

En este método, en cada paso se utiliza la información más actual que se dispone. Para este método se considera $B = A_D + A_L$, por lo que:

$$x^{k+1} = x^k + (A_D + A_L)^{-1}(b - Ax^k) \tag{7}$$

Pre-multiplicando por $(A_D + A_L)$ se obtiene:

$$(A_D + A_L)x^{k+1} = (A_D + A_L)x^k + (b - Ax^k) \tag{8}$$

Lo que finalmente se puede escribir como:

$$A_D x^{k+1} = b - A_L x^{k+1} - A_R x^k \tag{9}$$

1.2. Implementación 1.1

El enunciado solicita encontrar tres matrices B no singulares, de tal forma que el radio de la espectral de la matriz de iteración $G := I - B^{-1}A$, sea menor a 1. Para tal fin, se propone una aproximación basada en la generación de matrices aleatorias. El algoritmo implementado puede describirse por los siguientes pasos:

1. Inicializar matrices A, b y matriz identidad I.
2. Crear un contador para validar la cantidad de matrices encontradas que cumplan la condición del enunciado,
3. Generar una matriz B de tamaño *3x3* aleatoria, con elementos en el intervalo *[-5, 5]*.
4. Calcular el determinante de la matriz B.
5. Sí el determinante es diferente de cero continuar, sí no volver al paso 3.
6. Calcular la inversa de la matriz B.
7. Calcular la matriz $G := I - B^{-1}A$.
8. Calcular los valores eigen de la matriz G,
9. Encontrar el radio espectral de G, como el máximo valor de los valores absolutos de los valores eigen.
10. Sí el radio espectral es menor a 1 continuar, sí no volver al paso 3.
11. Mostrar la matriz B encontrada.
12. Aumentar el contador.

13. Sí el contador es menor a 3, repetir desde el paso 3, de lo contrario terminar.

El algoritmo anterior fue implementado utilizando el entorno de programación *Octave*. El programa desarrollado se puede observar en el algoritmo 1.

```
%Inicializacion de Variables Iniciales
A = [5 -1  1;
     2  8 -1;
    -1  1  4];
b = [10; 11; 3];
I = eye(3);
success = 0;

while(success < 3)
  B = randi([-5, 5], 3, 3); %Generacion de matriz B de forma aleatoria
  if (det(B) != 0)          %Confirmar que la matriz sea no singular
    B_inv = inv(B);         %Inversa de B
    G = I - B_inv*A;        %Calculo de matriz G
    eigen = eig(G);         %Calculo de valores eigen
    pG = max(abs(eigen));   %Calculo Radio espectral
    if (pG < 1)             %Verificar que el radio espectral cumpla
       printf("La matriz encontrada es ");
       display(B);          %Mostrar la matriz B encontrada
       printf("El radio espectral es %f \n", pG);
       success = success + 1;
    end
  end
end
```

Algoritmo 1: Programa para generar matrices B.

A partir de la implementación del algoritmo anterior se obtuvieron las siguientes matrices:

$$B_1 = \begin{bmatrix} 3 & 1 & -5 \\ -3 & 5 & -4 \\ 0 & 0 & 5 \end{bmatrix} \tag{10}$$

$$B_2 = \begin{bmatrix} 5 & 5 & 5 \\ 0 & 2 & -4 \\ -1 & 3 & 5 \end{bmatrix} \tag{11}$$

$$B_3 = \begin{bmatrix} 4 & 0 & 4 \\ -2 & 4 & 5 \\ -1 & -3 & 5 \end{bmatrix} \tag{12}$$

1.3. Implementación 1.2

El algoritmo que implementa el método general de iteración para resolver el sistema de ecuaciones se presenta en Algoritmo 2. El algoritmo consiste en incorporar una matriz B no singular, tal que:

$$A = (B + (A - B))$$

Considerando el sistema de ecuaciones $Ax = b$, se tiene:

$$(B + (A - B))x = b$$

$$x = (I - B^{-1}A)x + B^{-1}b$$

En donde el resultado de $(I - B^{-1}A)$ se considera como la matriz de iteración G:

$$G = (I - B^{-1}A)$$

Basado en esta estructura, la primera parte del algoritmo consiste en generar la matriz G. De esta forma la función *general_iteration_method*(A, B, b, x_0) recibe la matriz A, B el vector solución b y la condición inical del método de iteración, donde:

$$x^{(k)} = Gx^{(k-1)} + B^{-1}b$$

Al generar la matriz G, se implementa un ciclo *while* que realiza las iteraciones necesarias. La condición de salida se haya calculando la resta entre los vectores $x^k - x^{k-1}$ y posteriormente al vector resultante se le haya la norma 2:

$$||x^{(k)} - x^{(k-1)}||_2$$

$$||X|| = \sqrt{\sum_{i=0}^{n} x_i^2}$$

Este valor es comparado con el error permitido (10^{-5}).

```
Matrix_ptr general_interation_method(Matrix_ptr A, Matrix_ptr B, Matrix_ptr b, Matrix_ptr
    x_0){

  int n = A->n;
  int m = A->m;
  double error = 1e-5;
  double norm_err;
  int iteration = 0;
  //allox x as initial condition (0 0 0)
  if(x_0 == NULL){
    Matrix_ptr x_0 = matrix_alloc(m, 1);
  }

  Matrix_ptr x;
  //get Identity matrix
  Matrix_ptr B_inv =  matrix_inverse_3x3(B);
  Matrix_ptr I = get_identity_matrix(m,n);
  //Matrix_ptr G = matrix_alloc(m,n);

  //generate iteration matrix G
  Matrix_ptr G = mult_matrix(B_inv, A);
  G = scalar_mult(G, -1);
  G = sum_matrix(I, G);


  do{
    //iteration x =Gx_0 + B_inv*b
    x = mult_matrix(G,x_0);
    Matrix_ptr b_product = mult_matrix(B_inv, b);
    x = sum_matrix(x, b_product);
    //substract x-x_0
    x_0 = scalar_mult(x_0, -1);
    Matrix_ptr err = sum_matrix(x,x_0);

    norm_err = matrix_norm(err);
    x_0 = x;
    printf("Iteration error: %f, vs expected error: %f\n", norm_err, error );
    free(err);
    free(b_product);
    iteration ++;
  }
  while(norm_err > error);

  printf("Total number of iterations: %d\n",iteration );


  return x;
```

```
47 }
```

Algoritmo 2: Método de iteración general

Al implementar el método general de iteración para la matriz A propuesta con la matriz B, la cual es una matriz no singular con radio espectral menor a 1:

$$B := \begin{pmatrix} 4 & -2 & 1 \\ 5 & 4 & 2 \\ -5 & 2 & 4 \end{pmatrix}$$

Se obtienen los resultados presentados en la Figura 1. Donde se muestra que la solución encontrada por el método es $x = (1{,}999998, 0{,}999998, 0{,}999997)^T$, obtenido en 36 iteraciones.

```
------------ PUNTO 1 ------------

Matriz A:
5.000000        -1.000000       1.000000
2.000000        8.000000        -1.000000
-1.000000       1.000000        4.000000

Matriz B:
4.000000        -2.000000       1.000000
5.000000        4.000000        2.000000
-5.000000       2.000000        4.000000

Vector b:
10.000000
11.000000
3.000000


    ........METODO GENERAL.......
4.000000        -2.000000       1.000000
5.000000        4.000000        2.000000
-5.000000       2.000000        4.000000
Numero total de iteraciones: 36
Iteracion de x:
1.999998
0.999998
0.999997
```

Figura 1: Resultado prueba del algoritmo de iteración general.

1.4. Implementación 1.3

Para implementar el algoritmo de Jacobi y Seidel, es neceario en primer lugar generar la descomposición LDR de la matriz A, donde:

$$A = L + D + R$$

Siendo L es una matriz triangular inferior con diagonal principal nula, R una matriz triangular superior con diagonal nula y D es una matriz diagonal. El código para implementar dicha descomposición se muestra en el algoritmo 3.

```
LDR_Matrix_ptr LDR_decomposition(Matrix_ptr M){

  int m = M->m;
  int n = M->n;
  double val;
  //allocate memory for the ldr structure
  LDR_Matrix_ptr ldr = (LDR_Matrix_ptr)malloc(sizeof(LDR_Matrix));
  //allocate memory for the three matrices L,R and def
  ldr->L = matrix_alloc(m,n);
  ldr->D = matrix_alloc(m,n);
  ldr->R = matrix_alloc(m,n);

  for(int i = 0; i<m; i++){
    for(int j = 0; j< n; j++){
      val = get_matrix_value(M,i,j);
      if(i == j){
        set_matrix_value(ldr->D,i,j,val);
      }
      else if(i > j){
        set_matrix_value(ldr->L, i,j,val);
      }
      else{
        set_matrix_value(ldr->R,i,j,val);
      }
    }

  }
  return ldr;
}
```

Algoritmo 3: Descrimposición LDR para matriz A

Con la descomposición de la matriz se puede ahora realizar el método de Jacobi de la siguiente forma:

$$B = D$$

La matriz B del método general se escoge como la matriz D de la descomposición LDR y se calcula el método de iteración general. La implementación de este algoritmo se muestra en Algoritmo **??**. Donde se realiza la descomposición ldr, se asigna la matriz D a la matriz B y se utiliza la función del método general de iteración.

```
Matrix_ptr jacobi_iteration_method(Matrix_ptr A, Matrix_ptr b, Matrix_ptr x_0){
  //this method sets the B matrix as Ad
  //1. get ldr decomposition
  LDR_Matrix_ptr ldr=  LDR_decomposition(A);
  //set B as the D matrix;
  Matrix_ptr B = ldr->D;

  Matrix_ptr x= general_interation_method(A,B,b, x_0);
  printf("Jacobi method solution\n");
  print_matrix(x);
  return x;
}
```

Algoritmo 4: Implementación del método de iteración de Jacobi

Resultados del algoritmo se muestran en la Figura 2. Para la matriz A, el método de Jacobi llego a la solución $x = (1{,}999998, 1{,}000002, 1{,}000000)^T$ en 13 iteraciones.

El mismo procedimiento fue realizado para el método Gauss-Seidel. En este caso, la matriz B que se escoge es $B = D + L$. Por lo tanto, el primer paso es generar la descomposición LDR. Posteriormente, se asigna la matriz B y se ejecuta el método de iteración general. Este procedimiento es presentado en el Algoritmo 5.

```
    ........JACOBI.......
5.000000        0.000000        0.000000
0.000000        8.000000        0.000000
0.000000        0.000000        4.000000
Numero total de iteraciones: 13
Solucion de x por Jacobi
1.999998
1.000002
1.000000
```

Figura 2: Resultado prueba del algoritmo del método Jacobi

```
Matrix_ptr gauss_seidel_iteration_method(Matrix_ptr A, Matrix_ptr b, Matrix_ptr x_0){
  //this method set the B matrix as D+L
  //1. get ldr decomposition
  LDR_Matrix_ptr ldr=  LDR_decomposition(A);
  //add R and L matrices
  Matrix_ptr B = sum_matrix(ldr->D, ldr->L);

  Matrix_ptr x= general_interation_method(A,B,b, x_0);
  printf("Gauss-seidel method solution\n");
  print_matrix(x);
  return x;

}
```

Algoritmo 5: Implementación del método de iteración de Gauss Seidel

Al ejecutar el algoritmo de Seidel, se obtiene la solución $x = (1{,}999999, 1{,}00000, 100000)^T$ en un total de 8 iteraciones. Los resultados se muestran en la Figura 3.

```
    ........GAUSS-SEIDEL.......
5.000000        0.000000        0.000000
2.000000        8.000000        0.000000
-1.000000       1.000000        4.000000
Numero total de iteraciones: 8
Solucion de x por Gauss-seidel
1.999999
1.000000
1.000000
```

Figura 3: Resultado prueba del algoritmo con Gauss-Seidel

2. Punto 2

2.1. Enunciado

Resolver el sistema $Hx = b$ usando uno de los métodos iterativos, donde H es la matriz de Hilbert de orden 3 y $b = (1, 2, 3)^T$, tomando $x^{(0)} = (30, -190, 200)^T$.

En primer lugar se realizó el algoritmo para obtener la matriz H de orden n, la cual se muestra en el Algoritmo 6.

```
Matrix_ptr generate_hilbert_matrix(m){

  Matrix_ptr H = matrix_alloc(m,m);
  double val;

  for(int i =  0; i < m ;i++){
    for(int j = 0; j < m; j++){
      val = 1.0/(i+1+j+1-1);
      printf("%f\n", val);
      set_matrix_value(H, i,j, val);
    }
  }

  printf("HILBERT Matrix\n");
  print_matrix(H);
  return H;

}
```

Algoritmo 6: Algoritmo para obtener matriz H de orden n

Posteriormente se realizó la solución del sistema $Hx = b$ por medio del método de Gauss-Seidel (ver Algoritmo 7). Cabe resaltar que, no es posible implementar el método de Jacobi para la matriz $A := H$ de orden 3, ya que una condición de convergencia del método es que la matriz A sea diagonalmente dominante.

```
Matrix_ptr H =  generate_hilbert_matrix(3);
Matrix_ptr b_1 = matrix_alloc(3,1);
set_matrix_value(b_1,0, 0, 1);
set_matrix_value(b_1,1, 0, 2);
set_matrix_value(b_1,2, 0, 3);


set_matrix_value(x_0,0, 0, 30);
set_matrix_value(x_0,1, 0, -190);
set_matrix_value(x_0,2, 0, 200);
printf("%s\n","       GAUSS       -SEIDEL para Hilbert           " );
x_0 = gauss_seidel_iteration_method(H,b_1,x_0);
```

Algoritmo 7: Algoritmo para calcular la descomposición de Cholesky

Para el método de Gauss-Seidel se obtuvo la solución $x = (26{,}999927, -191{,}999627, 209{,}999658)^T$ en 516 iteraciones (ver Figura 4).

```
Iteration error: 0.000012, vs expected error: 0.000010
Iteration error: 0.000011, vs expected error: 0.000010
Iteration error: 0.000011, vs expected error: 0.000010
Iteration error: 0.000011, vs expected error: 0.000010
Iteration error: 0.000011, vs expected error: 0.000010
Iteration error: 0.000011, vs expected error: 0.000010
Iteration error: 0.000010, vs expected error: 0.000010
Iteration error: 0.000010, vs expected error: 0.000010
Iteration error: 0.000010, vs expected error: 0.000010
Total number of iterations: 516
Gauss-seidel method solution
26.999927
-191.999630
209.999658
```

Figura 4: Resultado prueba del algoritmo