Tarea 6 Métodos Numéricos

Carlos Giovanny Encinia González

September 11 2021

Resumen

Se implementaron cuatro maneras de solucionar matrices, las primeras que se muestran son para matrices con estructuras especificas, para ser mas precisos, son métodos para la solución de matrices diagonales, matrices triangulares inferiores y matrices triangulares superiores. El ultimo es un método general llamado método de Gauss, además de esto se utiliza la técnica del pivoteo para estabilizar la solución a ele sistema matricial.

Se nos fueron entregados varios archivos con algunos datos para poder probar cada uno de los métodos, entonces se mostraran los resultados obtenidos para cada archivo y su correspondiente método.

Desarrollo

Matriz Diagonal

Para resolver una matriz diagonal se requiere de un método muy sencillo. Debido a que la información de importancia recae en la diagonal de la matriz podemos colapsar el sistema a una matriz unidimensional, teniendo en cuenta esto el problema se reduce a una sola operación aritmética para cada variable a encontrar, siempre es importante asegurarse que la matriz no contenga ceros en su diagonal.

En seguida se muestra el pseudocodigo.

Algorithm 1: Algoritmo que resulve una matriz diagonal

```
1 solve_diagonal (matrix, size, x, y);
  Input  : matriz colapsada de size * 1 matrix, el número de filas de la
           matriz size, arreglo de tamaño igual a size donde se
           almacenará el resultado x, arreglo que contiene las variables
           dependientes y
  Output: x
2 i ← 0;
3 while i ≤ m do
4 |   x_i ← y_i/matrix_{i,0};
5 |   i ← i + 1;
6 end
7 return x;
```

Matriz Triangular Inferior

Para resolver la matriz triangular Inferior, se implemento la ecuaccion que vimos en la materia de metodos numericos, en la cual se necesita que para que el sistema matricial tenga solucion unica es necesario que la diagonal de la matriz no contenga ceros, una vez revisado esto se procede a solucionar mediante un algoritmo iterativo. Al resolver el problema se encuentra que seran necesarios dos terminos. Se espera que la matriz dada sea cuadrada.

El termino siguiente es de suma importancia para la resolucion del problema.

$$x_0 = y_0/matrix_{0,0};$$

Despues se utilizara

$$x_i = \frac{1}{matrix_{i,i}} * (y_i - \sum_{j=1}^{i} matrix_{i,j} * x_i)$$

En seguida se muestra el pseudocodigo.

Algorithm 2: Algoritmo que resuelve una matriz triangular inferior

```
1  solve_inferior (matrix, rows, cols, x, y);
   Input  : matriz matrix, el número de filas de la matriz rows, el
            numero de columnas cols, arreglo de tamaño igual a size
            donde se almacenará el resultado x, arreglo que contiene las
            variables dependientes y
   Output: x
2  i ← 0;
3  if cols ≠ rows then
4  |  print(No es un sistema cuadrado);
5  |  return 0;
6  else
7  |  ceros = combrobar_ceros_diagonal(matrix);
8  |  if ceros then
9  |  |  print(Hay ceros en diagonal, sin solucion unica);
10 |  |  return 0;
11 |  end
12 |  x_0 = y_0/matrix_{0,0};
13 |  i ← 1;
14 end
15 while i ≤ m do
16 |  x_i = 1/matrix_{i,i} * (y_i − Σ_{j=1}^{i} matrix_{i,j} * x_i);
17 |  i ← i + 1;
18 end
19 return x;
```

Matriz Triangular Superior

Para resolver la matriz triangular superior, se realizo un método muy similar al que se realizo al de la matriz triangular inferior. Pero comenzando desde la

parte inferior de la matriz, ya que aquí se encuentra solo un elemento distinto de cero. Al igual que el anterior método se necesita que los elemento sobre la diagonal sean distintos de cero. En seguida se muestra el pseudocódigo.

Algorithm 3: Algoritmo que resuelve una matriz triangular superior

```
1  solve_superior (matrix, rows, cols, x, y);
   Input  : matriz matrix, el número de filas de la matriz rows, el
            numero de columnas cols, arreglo de tamaño igual a size
            donde se almacenará el resultado x, arreglo que contiene las
            variables dependientes y
   Output: x
2  i ← 0;
3  if cols ≠ rows then
4  |  print(No es un sistema cuadrado);
5  |  return 0;
6  else
7  |  ceros = combrobar_ceros_diagonal(matrix);
8  |  if ceros then
9  |  |  print(Hay ceros en diagonal, sin solucion unica);
10 |  |  return 0;
11 |  end
12 |  i ← 1;
13 |  n ← rows − 1;
14 |  x_n = y_n/matrix_{n,n};
15 end
16 while i ≥ m do
17 |  x_i = 1/matrix_{i,i} * (y_i − Σ_{j=i+1}^{n} matrix_{i,j} * x_i);
18 |  i ← i − 1;
19 end
20 return x;
```

Gauss Pivoteo Total

Para realizar este método primero se encuentra el numero cuyo valor absoluto es mayor a los demás elementos de la matriz, este se pasa al inicio de la matriz en el primer elemento de l diagonal, después se comienzan a hacer ceros en las columnas, y y esto se realiza iterativamente hasta tener una matriz triangular superior. Después se utiliza el método para solucionar una matriz triangular superior. Además se debe de tener en cuenta las permutaciones que se realizan en la matriz, ya que esto cambia el orden de nuestras variables, y el orden de nuestra variable dependiente. En seguida se muestra el pseudocódigo.

Algorithm 4: Método Gauss por pivoteo total

```
1  solve_gauss_total (matrix, rows, cols, x, y);
   Input  : matriz matrix, el número de filas de la matriz m, arreglo de
            tamaño igual a size donde se almacenará el resultado x,
            arreglo que contiene las variables dependientes y, variable que
            se puede usar para determinar el valor del determinante signo
   Output: x, signo
2  m ← rows;
3  matrix ← matriz_aumentada(matrix, y);
4  cambios_x ← array(0 : m);
5  for i:n do
6  |  i_c ← i;
7  |  j_c ← i;
8  |  (i_c, j_c) = encuentra_maximo(matrix);
9  |  if i ≠ i_c and i = j_c or i = i_c and i ≠ j_c then
10 |  |  signo ← signo * (-1)
11 |  end
12 |  if  i ≠ i_c then
13 |  |  intercambias_filas(i, i_c);
14 |  end
15 |  if i ≠ j_c then
16 |  |  intercambia_columnas(j, j_c);
17 |  |  intercambia_x(i, j_c);
18 |  end
19 |  for l=i+1:m do
20 |  |  factor ← matrix_{l,i}/matrix_{i,i};
21 |  |  for j=i:m+1 do
22 |  |  |  matrix_{l,j} ← matrix_{l,j} − factor * matrix_{i,j};
23 |  |  end
24 |  end
25 end
26 x ← solve_superior(matrix, m, x);
27 x ← regresa_posicion_original(x, cambios_x);
28 return x, signo;
```

Resultados

Matriz Diagonal

Para la matriz diagonal se introdujeron datos de dos archivos, uno llamado M_DIAG.txt, y otro archivo llamado V_DIAG.txt, el algoritmo se desarrollo en el lenguaje de programación C, para optimizar el espacio de almacenamiento se utilizó memoria dinámica y se colapso la matriz dimensional a una unidimensional. En seguida se muestran los resultado s obtenidos al ejecutar el programa. También es importante mencionar que si se requieren hacer pruebas, se debe de ejecutar el programa desde codeblocks, ya se tiene creado un archivo main.c, que llama a las diferentes implementaciones.

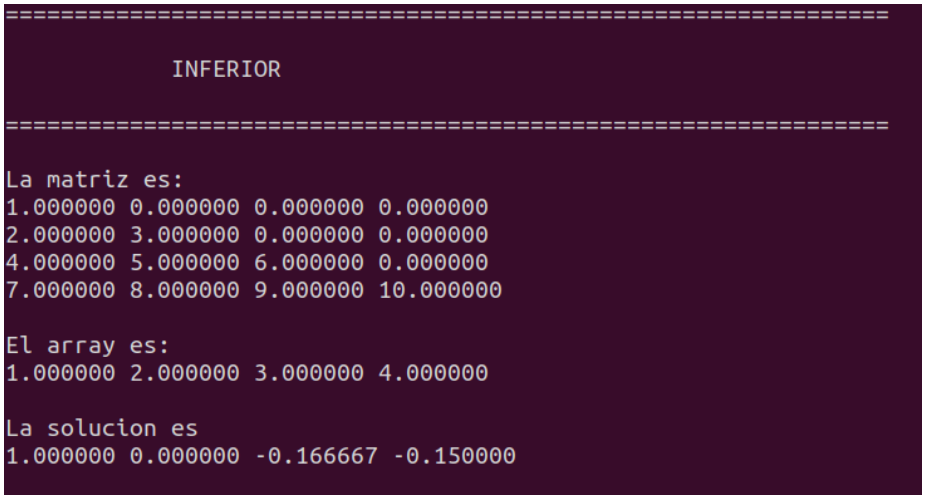
Figure 1: Resultado

```
================================================================

            DIAGONAL

================================================================

La matriz es:
1.000000 0.000000 0.000000 0.000000
0.000000 2.000000 0.000000 0.000000
0.000000 0.000000 3.000000 0.000000
0.000000 0.000000 0.000000 4.000000

El array es:
1.000000 2.000000 3.000000 4.000000

La solucion es
1.000000 1.000000 1.000000 1.000000

Memory array free
Memory matrix free
```

Matriz Triangular Inferior

Para la matriz triangular inferior se usaron como datos de entrada los archivos M_TINF.txt y el archivo llamado V_TINF, dentro del código se implementaron funciones especiales que guardan únicamente los valores distintos a cero de matrices triangulares, esto por el momento solo aplica para la lectura de archivos. En seguida se muestran los resultados obtenidos.

Figure 2: Resultado



Matriz Triangular Superior

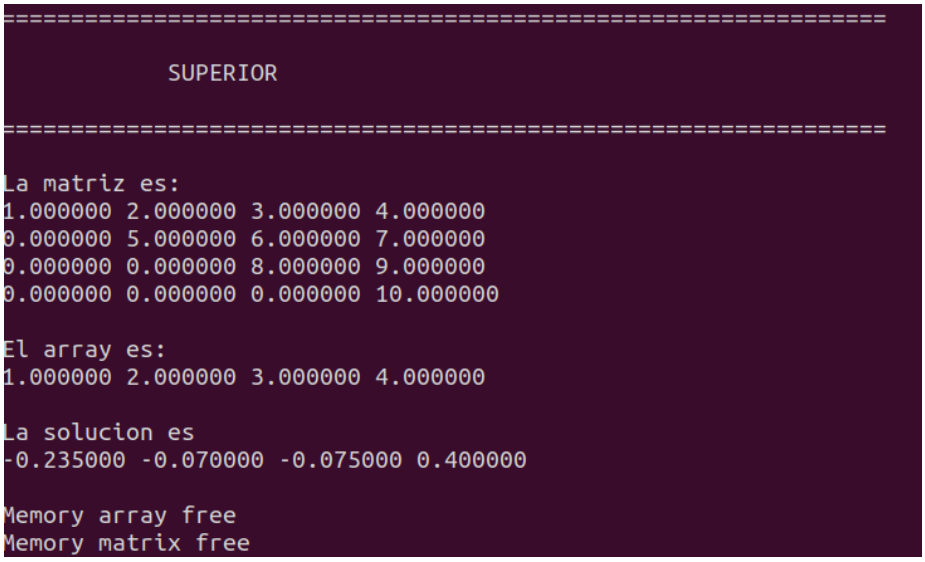
Para la matriz triangular superior se realizo una metodología casi identica que para la triangular inferior. Los archivos de entrada fueron M_TSUP.txt y V_TSUP.txt, una vez que se leyeron estos datos la matriz con coeficientes de el sistema de ecuaciones se guardaron en una estructura que evita los valores cero de la matriz, debido a esto se debe de considerar un desfase al acceder a los elementos de la matriz, el desfase se encuentra en las columnas, y se debe considerar la siguiente ecuación para acceder a la columna original.

$$j = j - i$$

donde j, hace referencia a la j-ésima columna y la i hace referencia a la i-ésima fila.

En seguida se muestra el resultado obtenido al ejecutar el programa.

Figure 3: Resultado



Gauss Pivoteo Total

Para este método se utilizaron los archivos M_LARGE.txt y V_LARGE.txt, para este método se utilizo un tipo de almacenamiento distinto a comparación de los demás métodos. En seguida se muestran los resultados obtenidos.

Figure 4: Input

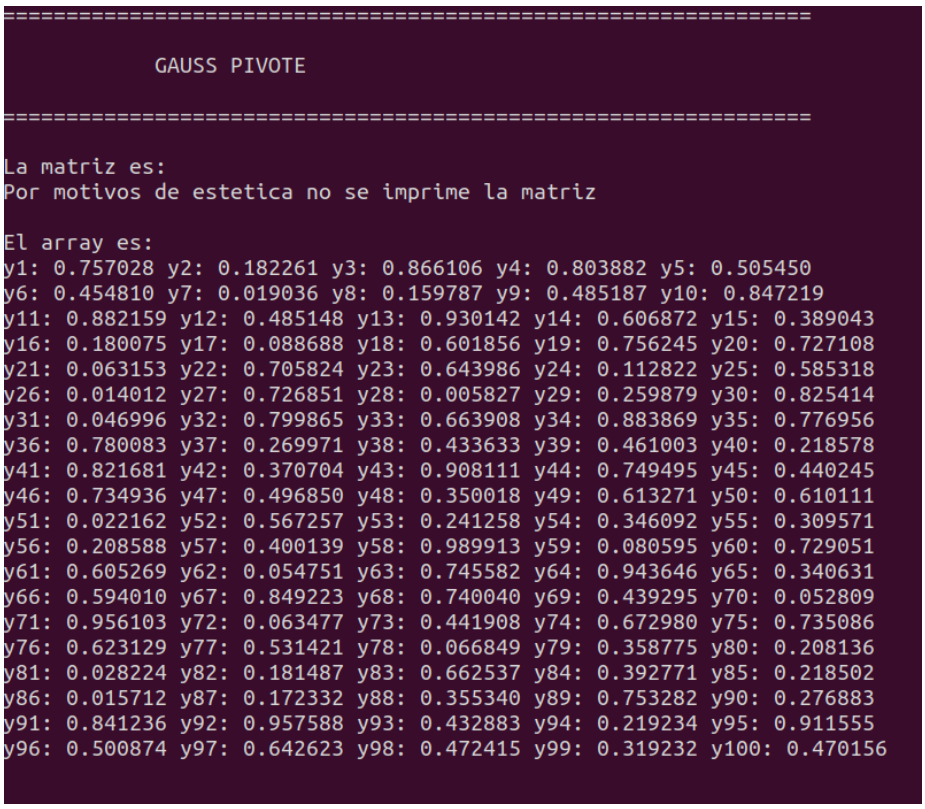
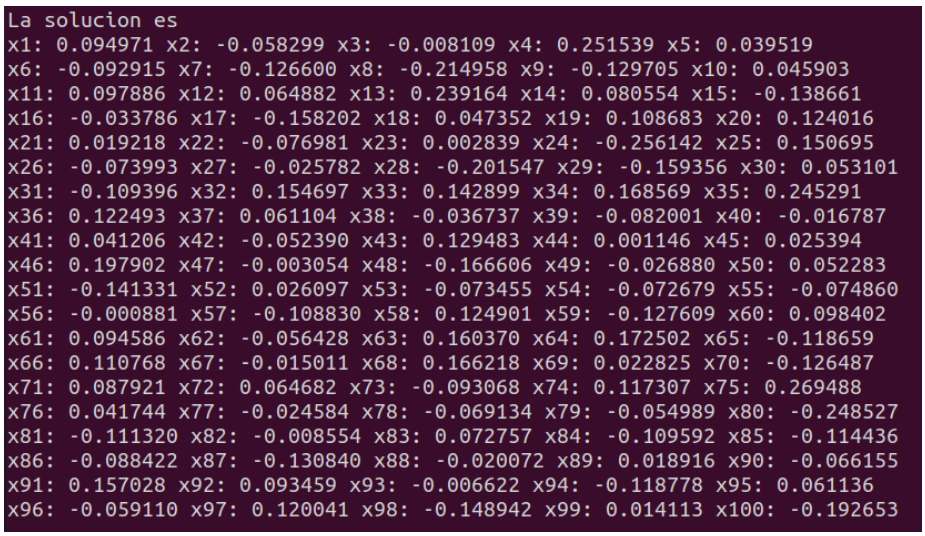


Figure 5: Resultado



Conclusión

Los primeros tres métodos son para poder solucionar, sistemas matriciales los cuales tienen una estructura especifica, aunque es posible solucionar un sistema matricial diagonal por medio de algún método como el que se utiliza para una matriz inferior y una matriz superior recomendaría que no se hiciera ya que se necesitan mas operaciones y estas son innecesarias.

Para el método de Gauss se implemento también la técnica de pivoteo, aunque en este momento no me parece que la haya implementado de una manera óptima, espero poder encontrar alguna manera de optimizarla, ya que lo que hago en este momento en el algoritmo es intercambiar los elementos de columnas uno por uno. También es importante decir que este método es uno de los mas tardados, por lo que no recomendaría utilizarlo para encontrar alguna solución.

En los primeros tres métodos se implemento una estructura particular (para cada uno) para el almacenamiento de los datos, esta misma metodología se puede aplicar después, si bien no se leerían los datos desde archivos, ayudaria mucho como cuando se hacen factorizaciones tipo LU o Cholesky.