

Tarea 6 Métodos Numéricos

Carlos Giovanni Encinia González

September 11 2021

Resumen

Se implementaron cuatro maneras de solucionar matrices, las primeras que se muestran son para matrices con estructuras específicas, 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 pseudocódigo.

Algorithm 1: Algoritmo que resuelve 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 ecuacion 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.

$$x_0 = y_0 / \text{matrix}_{0,0};$$

Despues se utilizara

$$x_i = \frac{1}{\text{matrix}_{i,i}} * (y_i - \sum_{j=1}^i \text{matrix}_{i,j} * x_j)$$

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  
           número 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 = comprobar_ceros_diagonal(matrix);  
8   if ceros then  
9     print(Hay ceros en diagonal, sin solucion unica);  
10    return 0;  
11  end  
12   $x_0 = y_0 / \text{matrix}_{0,0};$   
13  i ← 1;  
14 end  
15 while i ≤ m do  
16    $x_i = \frac{1}{\text{matrix}_{i,i}} * (y_i - \sum_{j=1}^i \text{matrix}_{i,j} * x_j);$   
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 elementos 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
           número 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 = comprobar_ceros_diagonal(matrix);
8   if ceros then
9     print(Hay ceros en diagonal, sin solución única);
10    return 0;
11  end
12  i ← 1;
13  n ← rows − 1;
14  xn = yn/matrixn,n;
15 end
16 while i ≥ m do
17    $x_i = \frac{1}{matrix_{i,i}} * (y_i - \sum_{j=i+1}^n matrix_{i,j} * x_j)$ ;
18   i ← i − 1;
19 end
20 return x;

```

Gauss Pivoteo Total

Para realizar este método primero se encuentra el número 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 la diagonal, después se comienzan a hacer ceros en las columnas, 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 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  $\leftarrow$  rows;  
3 matrix  $\leftarrow$  matriz_aumentada(matrix, y);  
4 cambios_x  $\leftarrow$  array(0 : m);  
5 for i:n do  
6   | ic  $\leftarrow$  i;  
7   | jc  $\leftarrow$  i;  
8   | (ic, jc) = encuentra_maximo(matrix);  
9   | if i  $\neq$  ic and i = jc or i = ic and i  $\neq$  jc then  
10  | | signo  $\leftarrow$  signo * (-1)  
11  | end  
12  | if i  $\neq$  ic then  
13  | | intercambia_filas(i, ic);  
14  | end  
15  | if i  $\neq$  jc then  
16  | | intercambia_columnas(j, jc);  
17  | | intercambia_x(i, jc);  
18  | end  
19  | for l=i+1:m do  
20  | | factor  $\leftarrow$  matrixl,i/matrixi,i;  
21  | | for j=i+1 do  
22  | | | matrixl,j  $\leftarrow$  matrixl,j - factor * matrixi,j;  
23  | | end  
24  | end  
25 end  
26 x  $\leftarrow$  solve_superior(matrix, m, x);  
27 x  $\leftarrow$  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 resultados 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

```
=====
                        INFERIOR
=====

La matriz es:
1.000000 0.000000 0.000000 0.000000
2.000000 3.000000 0.000000 0.000000
4.000000 5.000000 6.000000 0.000000
7.000000 8.000000 9.000000 10.000000

El array es:
1.000000 2.000000 3.000000 4.000000

La solucion es
1.000000 0.000000 -0.166667 -0.150000
```

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

```
=====
                        SUPERIOR
=====

La matriz es:
1.000000 2.000000 3.000000 4.000000
0.000000 5.000000 6.000000 7.000000
0.000000 0.000000 8.000000 9.000000
0.000000 0.000000 0.000000 10.000000

El array es:
1.000000 2.000000 3.000000 4.000000

La solucion es
-0.235000 -0.070000 -0.075000 0.400000

Memory array free
Memory matrix free
```

Gauss Pivoteo Total

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

Figure 4: Input

```
=====
GAUSS PIVOTE
=====

La matriz es:
Por motivos de estetica no se imprime la matriz

El array es:
y1: 0.757028 y2: 0.182261 y3: 0.866106 y4: 0.803882 y5: 0.505450
y6: 0.454810 y7: 0.019036 y8: 0.159787 y9: 0.485187 y10: 0.847219
y11: 0.882159 y12: 0.485148 y13: 0.930142 y14: 0.606872 y15: 0.389043
y16: 0.180075 y17: 0.088688 y18: 0.601856 y19: 0.756245 y20: 0.727108
y21: 0.063153 y22: 0.705824 y23: 0.643986 y24: 0.112822 y25: 0.585318
y26: 0.014012 y27: 0.726851 y28: 0.005827 y29: 0.259879 y30: 0.825414
y31: 0.046996 y32: 0.799865 y33: 0.663908 y34: 0.883869 y35: 0.776956
y36: 0.780083 y37: 0.269971 y38: 0.433633 y39: 0.461003 y40: 0.218578
y41: 0.821681 y42: 0.370704 y43: 0.908111 y44: 0.749495 y45: 0.440245
y46: 0.734936 y47: 0.496850 y48: 0.350018 y49: 0.613271 y50: 0.610111
y51: 0.022162 y52: 0.567257 y53: 0.241258 y54: 0.346092 y55: 0.309571
y56: 0.208588 y57: 0.400139 y58: 0.989913 y59: 0.080595 y60: 0.729051
y61: 0.605269 y62: 0.054751 y63: 0.745582 y64: 0.943646 y65: 0.340631
y66: 0.594010 y67: 0.849223 y68: 0.740040 y69: 0.439295 y70: 0.052809
y71: 0.956103 y72: 0.063477 y73: 0.441908 y74: 0.672980 y75: 0.735086
y76: 0.623129 y77: 0.531421 y78: 0.066849 y79: 0.358775 y80: 0.208136
y81: 0.028224 y82: 0.181487 y83: 0.662537 y84: 0.392771 y85: 0.218502
y86: 0.015712 y87: 0.172332 y88: 0.355340 y89: 0.753282 y90: 0.276883
y91: 0.841236 y92: 0.957588 y93: 0.432883 y94: 0.219234 y95: 0.911555
y96: 0.500874 y97: 0.642623 y98: 0.472415 y99: 0.319232 y100: 0.470156
```

Figure 5: Resultado

```
La solucion es
x1: 0.094971 x2: -0.058299 x3: -0.008109 x4: 0.251539 x5: 0.039519
x6: -0.092915 x7: -0.126600 x8: -0.214958 x9: -0.129705 x10: 0.045903
x11: 0.097886 x12: 0.064882 x13: 0.239164 x14: 0.080554 x15: -0.138661
x16: -0.033786 x17: -0.158202 x18: 0.047352 x19: 0.108683 x20: 0.124016
x21: 0.019218 x22: -0.076981 x23: 0.002839 x24: -0.256142 x25: 0.150695
x26: -0.073993 x27: -0.025782 x28: -0.201547 x29: -0.159356 x30: 0.053101
x31: -0.109396 x32: 0.154697 x33: 0.142899 x34: 0.168569 x35: 0.245291
x36: 0.122493 x37: 0.061104 x38: -0.036737 x39: -0.082001 x40: -0.016787
x41: 0.041206 x42: -0.052390 x43: 0.129483 x44: 0.001146 x45: 0.025394
x46: 0.197902 x47: -0.003054 x48: -0.166606 x49: -0.026880 x50: 0.052283
x51: -0.141331 x52: 0.026097 x53: -0.073455 x54: -0.072679 x55: -0.074860
x56: -0.000881 x57: -0.108830 x58: 0.124901 x59: -0.127609 x60: 0.098402
x61: 0.094586 x62: -0.056428 x63: 0.160370 x64: 0.172502 x65: -0.118659
x66: 0.110768 x67: -0.015011 x68: 0.166218 x69: 0.022825 x70: -0.126487
x71: 0.087921 x72: 0.064682 x73: -0.093068 x74: 0.117307 x75: 0.269488
x76: 0.041744 x77: -0.024584 x78: -0.069134 x79: -0.054989 x80: -0.248527
x81: -0.111320 x82: -0.008554 x83: 0.072757 x84: -0.109592 x85: -0.114436
x86: -0.088422 x87: -0.130840 x88: -0.020072 x89: 0.018916 x90: -0.066155
x91: 0.157028 x92: 0.093459 x93: -0.006622 x94: -0.118778 x95: 0.061136
x96: -0.059110 x97: 0.120041 x98: -0.148942 x99: 0.014113 x100: -0.192653
```


Conclusión

Los primeros tres métodos son para poder solucionar, sistemas matriciales los cuales tienen una estructura **específica**, 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.