TRABAJO PRÁCTICO N°5
Sistemas de ecuaciones lineales

Métodos aproximados Vs. métodos exactos: necesidad Vs. recursos

Si bien el uso de aritmética finita implica necesariamente la aparición de error numérico, se puede hablar de la existencia de *métodos directos* (o *exactos*) para la resolución de sistemas de ecuaciones lineales, en contraste con los **métodos iterativos**. Los métodos directos darán un "resultado exacto", en el sentido de que es el mejor que se puede obtener con la precisión ofrecida por el sistema de cálculo (nota del JTP: con aritmética finita, casi nada es exacto).

A diferencia de los métodos directos, los métodos iterativos permiten *aproximar* la solución paso a paso, hasta obtener un resultado con un error aceptable ***en un tiempo aceptable***.

Como ejemplo de los métodos exactos se verán **descomposición LU**, el método de eliminación de **Gauss** por **sustitución hacia atrás** y el de **Gauss-Jordan** de diagonalización. Como ejemplos de los métodos iterativos, se verán los métodos de **Jacobi** y de **Gauss-Seidel**.

Lo que se busca en esta práctica es que sean capaces de distinguir los distintos métodos, sus ventajas y desventajas, y que profundicen el uso de estructuras de datos como matrices y vectores.

1. **Método de eliminación de Gauss**

   a) Realice un diagrama de flujo del método de Gauss para la solución de sistemas de ecuaciones lineales. Introduzca en el diagrama los chequeos necesarios para la existencia de solución y la utilización del método. ¿Cuáles son esos requisitos?

   b) ¿Para qué matrices puede chequearse la existencia de una solución al principio? ¿Y si no se puede al principio, cuándo?

   c) Tome una rutina de solución por Gauss (puede ser una de las suministradas por la cátedra, bajada de internet o programada por Ud. mismo). Utilizando los comandos **tic** y **toc**, y resolviendo matrices de tamaño creciente, estime cuánto tardaría la computadora en resolver una matriz de 1015x1015 (tal vez requiera herramientas fuera de Matlab/Octave para dar esta respuesta; por ejemplo, una hoja de cálculos). Nota: para generar una matriz cuadrada de 100x100 con números aleatorios entre 0 y 1000 puede usarse el comando.

```
randx = randi([0,1000],100,100)
```

   Para generar un vector columna de 100 números aleatorios del 0 al 1000, se puede usar el comando

```
randb = randi([0,1000], 100, 1)
```

   donde randx almacenará la matriz generada y randb almacenará el vector. El comando en sí es "randi([rango_de_valores], cantidad_de_filas, cantidad_de_columnas)".

d) Errores. Si bien el método de Gauss es un método directo o exacto, lo es salvo los errores intrínsecos al cálculo con aritmética finita. En ese sentido, ¿Hay partes de la solución que sean más precisas que las otras? De ser así, ¿Cuáles? Si la máquina cometiese un error relativo del orden de 10-15, ¿Cuál sería el sistema más grande a resolver con un error del orden del 1 %? De un tamaño aproximado.

e) Validación de los resultados: plantee dos métodos para validar el resultado (verificar que el resultado obtenido sea realmente el resultado).

2. **Método de Gauss-Jordan**

a) Repita el punto 1-a) para el método de Gauss-Jordan.

b) El siguiente código fue tomado de la web:

```
clc
clear all
a = [6 2 2 4;-1 2 2 -3;1 0 2 3 ; 0 1 1 4];
b = [1;-1;1;2];
% a is the matrix of co-efficient
% b is the matrix of constants
% m is the row
% c is the column
[m,c]=size(a);
j=1;  %j th column
k=1;%k th row
n=1;% n th pivoting row
z=2;% Row operation always starts from 2
% Forward Elimination
% Shifting Loop for column operation
for i=1:c-1
  %Making each desired column component zero
  for r=z:m
    % Checking if any Lower triangle is Already Zero or not
    if a(r,j)==0
      % If any is zero left the entire row as it is and skip
         the step
      a(r,:)=a(r,:);
      b(r,:)=b(r,:);
    else
      b(r,:)=((a(r,j)/a(k,j))*b(n,:))-b(r,:);
      a(r,:)=((a(r,j)/a(k,j))*a(n,:))-a(r,:);
    end
  end
  k=k+1;% Changing row after completion of inner loop
  n=n+1;% Changing the pivoting row
  z=z+1;% Setting a new condition for inner loop
  j=j+1;% Changing column after completion of inner loop
end
% Converting all the diagonal elements to one
for row=1:m
  b(row,:)=b(row,:)/a(row,row);
  a(row,:)=a(row,:)/a(row,row);
```

```
end
D=c;
for q=1:D-1
  for rowJ=m-1:-1:1
    if a(rowJ,c)==0
      a(rowJ,:)=a(rowJ,:);b(rowJ,:)=b(rowJ,:);
    else
      b(rowJ,:)=b(rowJ,:)-(a(rowJ,c)*b(m,:));
      a(rowJ,:)=a(rowJ,:)-(a(rowJ,c)*a(m,:));
    end
  end
  m=m-1;
  c=c-1;
end
```

Modifique el código anterior para que tome como entradas una matriz arbitraria y un vector de términos independientes.

c) Agregue los chequeos necesarios para verificar que tanto matriz como vector de entrada sean aptos para el cálculo.

d) Modifique el código anterior para que verifique si la solución obtenida es realmente la solución

e) Utilizando las funciones **tic** y **toc** calcule el tiempo empleado para resolver sistemas de hasta 1000x1000. Compare estos tiempos con los obtenidos para el método de Gauss.

f) ¿Cómo evoluciona el error en el cálculo por Gauss-Jordan?

g) Realice un diagrama de flujo para el código del inciso b). ¿Se parece a lo que realizó en el inciso a)? ¿Pueden llevarse partes de un diseño al otro para mejorarlo? De ser posible, plantee un diagrama de flujo del algoritmo mejorado, escriba el programa y compare el tiempo de ejecución para resolver un sistema de 1000x1000 con el programa del inciso b)

3. **Descomposición LU**

a) Plantee un diagrama de flujo del método.

b) El código siguiente tiene un problema: calcula la matriz L con ceros en la diagonal (en lugar de unos).

```
function[L U]=LU2(A)
%Descomposicion de matriz AA: A = L U
  z=size(A,1);
  L=zeros(z,z);
  U=zeros(z,z);
  for i=1:z
    % Obtencion de L
    for k=1:i-1
      L(i,k)=A(i,k);
        for j=1:k-1
          L(i,k)= L(i,k)-L(i,j)*U(j,k);
        end
      L(i,k) = L(i,k)/U(k,k);
```

```
14
15      end
16
17      % Obtencion de U
18
19      for k=i:z
20        U(i,k) = A(i,k);
21        for j=1:i-1
22          U(i,k)= U(i,k)-L(i,j)*U(j,k);
23        end
24      end
25    end
26
27    U
28
29    L
30  end
```

¿Cuál es el problema? ¿Cómo puede arreglarse? ¿Puede darse una "solución sucia (muchas veces conocida como un hack)" (sin entender el problema)? Repare el código para que la matriz L tenga unos en la diagonal.

c) Plantee un código que verifique que la descomposición es correcta en forma automática para matrices de 5x5 aleatorias (a esto se le llama "testeo", y es muy importante al programar; idealmente, deberían escribirse o programarse pruebas automáticas para las distintas funciones o subrutinas que componen un programa).

d) Se sabe que, una vez obtenidas las matrices L y U, estas sirven para cualquier conjunto de términos independientes. Lo que resta para que la función anterior sirva, es escribir otra función que utilice las matrices L y U para resolver un sistema de ecuaciones lineales tipo $Ax = b$, donde b es el vector de términos independientes. Escriba esta función, y su test automático. Preste atención a lo que debe cambiar en cada test, ya que hay cosas que, se supone, fueron previamente testeadas...

e) Utilice código previamente escrito para escribir una función que ***valide*** los datos de entrada de la función LU2 (en especial, la matriz).

f) Construya una función que tome como entrada una matriz de tamaño arbitrario y un vector apropiado y que, utilizando las funciones de validación, descomposición y solución escritas previamente resuelva cualquier sistema de ecuaciones lineales.

g) Use **tic** y **toc** para evaluar el tiempo de proceso en función del tamaño de las matrices.

4. **Métodos iterativos**

a) Realice un diagrama de flujo de los métodos iterativos de Jacobi y Gauss-Seidel.

b) El siguiente código resuelve sistemas de ecuaciones lineales por el método de Jacobi, o por el de Gauss-Seidel.

```
1  function [resultado] = jacobi(matriz_A, vector_b, vector_x0,
       cotaError)
2    long_A = length(matriz_A);
3    k = 1;
4    error = 1.2*cotaError;
5    while(error > cotaError)
```

```
6       parte_suma_vec_x = 0;
7       for i=1:long_A
8         parte_suma_vec_x(i)=0;
9         for j=1:long_A
10          parte_suma_vec_x(i) = parte_suma_vec_x(i) + ...
11            matriz_A(i, j)*vector_x0(j)*(1-delta(i,j));
12        end
13        vector_x(i) = 1/matriz_A(i,i)*( vector_b(i) -
              parte_suma_vec_x(i) );
14      end
15      disp(vector_x);
16      error = max(abs(vector_x - vector_x0));
17      vector_x0 = vector_x;
18      k=k+1;
19    end
20    resultado = vector_x;
21
22  end
23
24  function d=delta(i,j)
25    if i != j
26      d = 0;
27    else
28      d = 1;
29    end
30  end
```

Identifique el método empleado en el código, y realice en él las modificaciones necesarias para que calcule por el método restante.

c) Modifique ambos códigos (Jacobi y Gauss-Seidel) para que grafiquen el error en función del número de iteraciones.

d) Usando **tic** y **toc** realice un programa que grafique el tiempo de cálculo en función del error para ambos métodos.

5. **Consumo de recursos** Un factor muy importante a tener en cuenta cuando se decide qué método se usará, es el consumo de recursos computacionales. Por ejemplo, un algoritmo que sea más eficiente (tarda menos, usa menos procesador) que otro pero utilice más memoria RAM limitará más el tamaño de los problemas a resolver que uno menos eficiente (usa más procesador) pero que ocupa menos espacio en memoria.

   a) Suponiendo que un número ocupa 64 bits (8 bytes) en memoria, estime el consumo de RAM de los algoritmos usados en el caso de resolver un sistema de 1000, 1.000.000 y 1.000.000.000 de grados de libertad. Estime el tamaño máximo de la matriz que puede resolver una computadora con 8GB de RAM. ¿Y una con 1TB (1000 GB)?

   b) ¿Cómo variaría la necesidad de RAM si se tratase de matrices triangulares?

   c) ¿Y si se tratase de matrices **banda**, con valores distintos de cero solamente una columna atrás y una adelante de la diagonal?

   d) ¿Y si se tratase de matrices simétricas? Este es un tema importante, porque en física e ingeniería hay muchos problemas que dan origen a matrices simétricas.

e) Compare los tiempos de resolución de sistemas de 1000x1000 (el mismo sistema) de todos los programas usados (para los métodos iterativos, use un error porcentual de 0.0001).

f) Compare los tiempos obtenidos en el punto f) con el que toma a las funciones intrínsecas de Matlab/Octave (investigar cuáles son y cómo se usan).