UNIVERSIDAD DE CONCEPCION
FACULTAD DE CIENCIAS
FISICAS Y MATEMATICAS
DEPARTAMENTO DE INGENIERIA MATEMATICA

Cálculo Numérico (521230)

Laboratorio 3

Sistemas de ecuaciones lineales

El objetivo de este laboratorio es aprender a utilizar eficientemente métodos directos e iterativos para la solución de sistemas de ecuaciones lineales $\mathbf{A}\mathbf{x} = \mathbf{b}$, así como el correcto tratamiento de las matrices dispersas (sparse).

1. El *método de las potencias inversas* permite calcular el valor propio menor en valor absoluto de una matriz. El algoritmo de este método es el siguiente:

> $\mathbf{y}_0$: dato inicial normalizado de manera que $\|\mathbf{y}_0\|_2 = 1$;
> $\mathbf{x}_0$: solución de $\mathbf{A}\mathbf{x}_0 = \mathbf{y}_0$;
> $\lambda_0 = 1/(\mathbf{y}_0^{\mathrm{t}}\mathbf{x}_0)$;
> $k = 1, 2, 3, \ldots$
> $\quad \mathbf{y}_k = \mathbf{x}_{k-1}/\|\mathbf{x}_{k-1}\|_2$;
> $\quad \mathbf{x}_k$: solución de $\mathbf{A}\mathbf{x}_k = \mathbf{y}_k$;
> $\quad \lambda_k = 1/(\mathbf{y}_k^{\mathrm{t}}\mathbf{x}_k)$;
> hasta que $\lambda_k = \lambda_{k-1}$.

El λ_k con el que finaliza este algoritmo es el valor calculado del autovalor de menor valor absoluto de la matriz $\mathbf{A}$.

(a) Escriba un programa MATLAB que implemente adecuadamente este algoritmo. Para ello tenga en cuenta lo siguiente:

i. Aproveche el comando *lu* para reducir significativamente el costo computacional del método.
ii. No almacene todos los $\mathbf{x}_k$, $\mathbf{y}_k$ y λ_k, $k = 1, 2, 3, \ldots$, sino sólo los necesarios.

(b) Testee el programa con la matriz tridiagonal del ejercicio 4.1 del Laboratorio 2

$$\mathbf{A} = \begin{pmatrix} a & b & & 0 \\ c & \ddots & \ddots & \\ & \ddots & \ddots & b \\ 0 & & c & a \end{pmatrix}.$$

correspondiente a $n = 10$, $a = 2$ y $b = c = -1$, cuyos valores propios se conocen exactamente:

$$\lambda_j = 2\left[1 - \cos\left(\frac{j\pi}{n+1}\right)\right], \quad j = 1, \ldots, n.$$

Calcule también estos valores propios con el comando MATLAB *eig*.

2. Las matrices de *Hilbert*

$$\mathbf{H}_n = \left(h_{ij}^n\right) \in \mathbb{R}^{n\times n}, \qquad \text{con } h_{ij}^n = \frac{1}{i+j-1}, \quad i,j = 1,\ldots,n,$$

son matrices muy mal condicionadas. Estas matrices se generan en MATLAB con el comando *hilb*.

(a) Tabule los números de condición de estas matrices para $n = 2,\ldots,10$ (utilizar para ello el comando *cond*) y las estimaciones del mismo que se obtienen a partir del comando *rcond*.

(b) Sean

$$\mathbf{b}_0 = \begin{pmatrix} 0.7487192 \\ 0.4407175 \\ 0.3206968 \\ 0.2543113 \\ 0.2115308 \\ 0.1814429 \end{pmatrix} \quad \text{y} \quad \mathbf{b}_1 = \mathbf{b}_0 + \delta\mathbf{b},$$

con $\delta\mathbf{b}$ un vector de perturbaciones aleatorias de valor absoluto menor o igual a 10^{-6} (Note que la sentencia MATLAB `(2*rand(n,1)-1)*a` genera un vector columna aleatorio de dimensión n, uniformemente distribuido en el intervalo $[-a,a]$).
Resuelva los sistemas $\mathbf{H}_6\mathbf{x}_0 = \mathbf{b}_0$ y $\mathbf{H}_6\mathbf{x}_1 = \mathbf{b}_1$. Compare la diferencia de las soluciones $\delta\mathbf{x} = \mathbf{x}_1 - \mathbf{x}_0$ con la de los segundos miembros $\delta\mathbf{b}$. Describa lo qué se observa.

(c) Verifique que se satisface la relación

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \text{cond}(\mathbf{H}_6)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

3. En muchas aplicaciones es necesario resolver varios sistemas de ecuaciones $\mathbf{A}\mathbf{x}_i = \mathbf{b}_i$, con la misma matriz $\mathbf{A} \in \mathbb{R}^{n\times n}$ y distintos segundos miembros $\mathbf{b}_i$, $i = 1,\ldots,m$. Para hacer esto en MATLAB resulta conveniente generar la matriz de segundos miembros

$$\mathbf{B} = \left[\begin{array}{c|c|c} & & \\ \mathbf{b}_1 & \cdots & \mathbf{b}_m \\ & & \end{array}\right] \in \mathbb{R}^{n\times m}$$

y resolver el sistema matricial $\mathbf{A}\mathbf{X} = \mathbf{B}$, cuya solución

$$\mathbf{X} = \left[\begin{array}{c|c|c} & & \\ \mathbf{x}_1 & \cdots & \mathbf{x}_m \\ & & \end{array}\right] \in \mathbb{R}^{n\times m}$$

es la matriz de vectores solución $\mathbf{x}_i$, $i = 1,\ldots,m$, de los sistemas anteriores.

El siguiente programa MATLAB tiene por objeto verificar esto experimentalmente:

```
function [t1,t2,dif]=compara(n,m)
A=rand(n);
B=rand(n,m);

t0=cputime;
X=A\B;
t1=cputime-t0;

t0=cputime;
for i=1:m
    Y(:,i)=A\B(:,i);
end
t2=cputime-t0;

dif=norm(X-Y,inf);
```

(a) Escriba y ejecute el programa anterior.

(b) Analice lo que hace este programa. (Note que el comando *cputime* entrega el tiempo de CPU utilizado desde que se inició MATLAB.)

(c) Justifique la diferencia de tiempos de ejecución que se observa.

(d) Utilice la sentencia MATLAB

```
>> X=A\B;
```

con una matriz $\mathbf{B}$ adecuada para calcular $\mathbf{A}^{-1}$. Compare el resultado obtenido con el del comando MATLAB *inv*.

4. (a) Haga un programa *function* que genere una matriz de la forma

$$\mathbf{B} = \begin{pmatrix} 2\,\mathbf{I} & -\mathbf{I} \\ -\mathbf{I} & 2\,\mathbf{I} \end{pmatrix} \in \mathbb{R}^{2n\times 2n}$$

para $n = 100$. Compruebe que $\mathbf{B}$ es una matriz dispersa mediante el comando `nnz` (*Number of Non Zeros*) que da la cantidad total de entradas no nulas de la matriz.

(b) Determine la cantidad de memoria para almacenar en forma *sparse* otra matriz $\mathbf{A}$ igual a $\mathbf{B}$.

(c) Genere un vector aleatorio $\mathbf{b} \in \mathbb{R}^{2n}$ y compare los tiempos necesarios para resolver los sistemas $\mathbf{Bx} = \mathbf{b}$ y $\mathbf{Ax} = \mathbf{b}$. Indique cuál resulta más conveniente.

(d) Muchos comandos, cuando se aplican a matrices *sparse*, generan matrices *sparse*. Por ejemplo, `diag`, `tril` y `triu`.

Utilice estos comandos para obtener las matrices $\mathbf{D}$, $\mathbf{L}$ y $\mathbf{U}$ de la descomposición $\mathbf{A} = \mathbf{D}-\mathbf{L}-\mathbf{U}$. Calcule la matriz de iteración del método de Jacobi $\mathbf{J} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$. Compruebe que las matrices $\mathbf{D}$, $\mathbf{L}$, $\mathbf{U}$ y $\mathbf{J}$ tambien se almacenan como matrices *sparse*.

(e) Si bien el almacenamiento de la matriz $\mathbf{A}$ requiere mucha menos memoria que el de la matriz $\mathbf{B}$, para generar $\mathbf{A}$ como se ha descrito resulta necesario haber almacenado previamente $\mathbf{B}$.

A veces, la memoria del computador no alcanza para almacenar $\mathbf{B}$ en forma llena. En tal caso, es necesario generar directamente la matriz dispersa $\mathbf{A}$ sin pasar nunca por la matriz llena $\mathbf{B}$. Indique alguna forma de hacer esto y compruébelo.

(f) Para algunas aplicaciones es necesario dar la forma llena de una matriz dispersa. El comando inverso de `sparse` que almacena una matriz dispersa en forma de matriz llena es `full`. Compruébelo.

5. Haga un programa que resuelva mediante el método de **Gauss-Seidel** un sistema de ecuaciones $\mathbf{Ax} = \mathbf{b}$ con error menor que una tolerancia dada `tol`. Parta de un dato inicial $\mathbf{x}^0$ nulo y utilice el criterio de detención estudiado en las clases teóricas. El programa también debe detenerse si se alcanza un número máximo de iteraciones `maxit` sin que se satisfaga el criterio de convergencia.

Testee el método con la matriz del Ej. 4.2 del Lab. 2 con una tolerancia `tol` $= 10^{-6}$. Verifique que la solución obtenida aproxima a la solución exacta con la tolerancia del error estipulada. Determine la cantidad de iteraciones realizadas.

6. Baje de la página web del curso (o bien solicite al ayudante) los archivos de datos `data-1.mat`, `data-2.mat` y `data-3.mat` y el programa `show.m`. En los archivos se han almacenado mediante un comando `save` una matriz $\mathbf{A}$ (en forma *sparse*) y un vector $\mathbf{b}$ generados mediante un programa MATLAB para la resolución por el método de **elementos finitos** de un problema de EDPs que modela la deformación de una membrana bajo la acción de una fuerza. Estos archivos contienen también datos geométricos para la visualización de los resultados mediante el programa `show.m`.

Cada uno de los tres archivos de datos corresponde a modelaciones de distintos grados de precisión de la membrana: `data-1.mat` es una modelación grosera, `data-2.mat` una normal y `data-3.mat` una bien fina. Por ello las dimensiones de las matrices y vectores son sustancialmente distintas.

(a) Cargue el archivo `data-1.mat` mediante un comando `load`. Determine las dimensiones del problema y "espíe" la matriz $\mathbf{A}$.

(b) Resuelva el sistema de ecuaciones $\mathbf{Ax} = \mathbf{b}$ correspondiente al archivo `data-1.mat` mediante el programa de Gauss-Seidel del ejercicio anterior, con error menor que 10^{-6}. Indique cuántas iteraciones son necesarias.

(c) Aplique el método del gradiente conjugado mediante el comando `pcg`. La sentencia para la aplicación elemental de este comando es la siguiente:

```
>> x=pcg(A,b);
```

Indique lo qué ocurre.

(d) Utilice el comando `help` para ver como pueden modificarse la tolerancia del error y el número máximo de iteraciones en el comando `pcg` . Aplique otra vez el método del gradiente conjugado con tolerancia 10^{-6} y con un número de iteraciones suficiente para lograr la convergencia.

(e) Visualice la deformación calculada de la membrana ejecutando el programa `show`. (Ojo! El programa presupone que la solución calculada está en un vector llamado `x`.)

(f) Repita los pasos anteriores (a), (d) y (e) con los otros dos archivos (si es que entran en memoria y si la solucion de los sistemas de ecuaciones no requiere excesivo tiempo).

GBG/MCP/RRS/MSC
http://www.ing-mat.udec.cl/pregrado/asignaturas/521230/
03/09/03

Soluciones propuestas

1. File `pot.m`

```
n=10;
A=trid(n,2,-1,-1);
[L,U]=lu(A);
y=rand(10,1);
y=y/norm(y);
x=U\(L\y);
lant=0;
l=1/(x'*y);
while(l~=lant)
   lant=l;
   y=x/norm(x);
   x=U\(L\y);
   l=1/(x'*y);
end
l
k=1:10;
autov=2*(1-cos(k*pi/(n+1)))'
```

Salida:

```
>> pot

l =

   0.08101405277101

autov =

   0.08101405277101
   0.31749293433764
   0.69027853210943
   1.16916997399623
   1.71537032345343
   2.28462967654657
   2.83083002600377
   3.30972146789057
   3.68250706566236
   3.91898594722899
```

2. File `hilbert.m`

```
n=6;
Hn=hilb(n);

b0=[0.7487192; 0.4407175; 0.3206968; 0.2543113; 0.2115308; 0.1814429];
db=(2*rand(n,1)-1)*1.0e-06;
b1=b0+db;

x0=Hn\b0;
x1=Hn\b1;
dx=x1-x0;

format long
seg_miembro=[b1 b0 db]
solucion=[x1 x0 dx]
normas=[norm(db) norm(b0) norm(dx) norm(x0)]
compar=[norm(dx)/norm(x0) cond(Hn)*(norm(db)/norm(b0))]
```

Salida:

```
>> hilbert

seg_miembro =

   0.74872010025857   0.74871920000000   0.00000090025857
   0.44071696227703   0.44071750000000  -0.00000053772297
   0.32069701368517   0.32069680000000   0.00000021368517
   0.25431127196494   0.25431130000000  -0.00000002803506
   0.21153158259793   0.21153080000000   0.00000078259793
   0.18144342419367   0.18144290000000   0.00000052419367

solucion =

   0.46857987754213   0.46281539999986   0.00576447754227
   0.10031791680389   0.26254200000410  -0.16222408320020
   1.35354857524713   0.26476799997211   1.08878057527503
  -2.83447617833927  -0.01814399992728  -2.81633217841199
   3.42102103729396   0.32545799991967   3.09556303737428
  -1.18072054185112   0.03492720003165  -1.21564774188277

normas =

   0.00000142593735   0.99999991770093   4.49486706292068   0.67874922747620

compar =

   6.62227945309639  21.31927470184091
```

3. Ej. 4(a,b,c,d). File `Ejer1.m` :

```
n=100;

B=[2*eye(n) -eye(n); -eye(n) 2*eye(n)];
A=sparse(B);

b=rand(2*n,1);

t0=cputime;
x=B\b;
Tiempo_full=cputime-t0

t0=cputime;
y=A\b;
Tiempo_sparse=cputime-t0

D=diag(diag(A));
L=-(tril(A)-D);
U=-(triu(A)-D);
J=D\(L+U);
```

Salida:

```
>> Ejer1
Tiempo_full =
    0.0300
Tiempo_sparse =
     0
>> whos
  Name                Size          Bytes  Class

  A                 200x200          5604  sparse array
  B                 200x200        320000  double array
  D                 200x200          3204  sparse array
  J                 200x200          3204  sparse array
  L                 200x200          6804  sparse array
  Tiempo_full         1x1               8  double array
  Tiempo_sparse       1x1               8  double array
  U                 200x200          6804  sparse array
  b                 200x1            1600  double array
  n                   1x1               8  double array
  t0                  1x1               8  double array
  x                 200x1            1600  double array
  y                 200x1            1600  double array
```

4. Ej. 4(e,f).

```
>> clear all
>> n=100;
>> A=[2*speye(n) -speye(n); -speye(n) 2*speye(n)];
>> B=full(A);
>> whos
  Name      Size          Bytes  Class

  A       200x200          5604  sparse array
  B       200x200        320000  double array
  n         1x1                8  double array
```

5. Ej. 5. File `GS.m`:

```
n=length(A);
N=tril(A);        % Gauss-Seidel
P=N-A;

x=zeros(n,1);
corr=1;
errest=1;
iter=1;

while errest>tol & iter<maxit
    iter=iter+1;
    x0=x;
    corr0=corr;
    x=N\(P*x0+b);
    corr=norm(x-x0,inf);
    normest=corr/corr0;
    if normest>=1
        error('norma de la matriz de iteracion > 1')
    end
    errest=normest/(1-normest)*corr;
end
```

Salida:

```
>> n=10;
>> A=sparse(trid(n,4,1,1));
>> b=rand(n,1);
>> tol=1.e-6;
>> maxit=100;

>> Jac
>> Num_iter=iter
Num_iter =
    19
>> Norm_Error=norm(x-A\b,inf)
Norm_Error =
   3.0344e-07
```

6. Ej. 6(a,b,c,d,e).

```
>> load data-1.mat
>> whos
  Name               Size         Bytes  Class

  A                145x145        12596  sparse array
  Coordinates      185x2           2960  double array
  Elements3        328x3           7872  double array
  Elements4          0x0              0  double array
  FreeNodes          1x145         1160  double array
  b                145x1           1160  double array

>> spy(A)

>> tol=1.e-6;
>> maxit=1000;
>> GS
>> Num_iter=iter
Num_iter =
   188
>> Norm_Error=norm(x-A\b,inf)
Norm_Error =
   9.5904e-07

>> x=pcg(A,b);
pcg stopped at iteration 20 without converging to the desired tolerance 1e-06
because the maximum number of iterations was reached.
The iterate returned (number 20) has relative residual 0.00053

>> x=pcg(A,b,tol,maxit);
pcg converged at iteration 32 to a solution with relative residual 8.2e-07

>> show
```