

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

9-12-2020

Métodos Numericos para la Computación

Entrega 5

4ºCurso

Grupo Prácticas 11

Several thin, curved lines in dark blue and light grey that sweep upwards from the bottom left towards the center of the page.

Alejandro Daniel Herrera Cardenes
Carlos Eduardo Pacichana Bastidas
UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Indice

Practica 1 2

 Descripción..... 2

 Trabajo realizado..... 2

Practica 2 9

 Descripción..... 9

 Trabajo realizado..... 9

Práctica 3 17

 Descripción..... 17

 Trabajo realizado..... 17

Practica 1

Descripción

Realizar los siguientes ejercicios usando la librería LAPACK. Emplear Octave para verificar que el resultado es correcto.

1. Generar con Octave una matriz aleatoria de 6x6 con números de 1 a 10, comprobando que su determinante no sea nulo. Realizar la factorización LU con pivotamiento.
2. Utilizar LAPACK a nivel computacional para obtener:
 - a) Factorización LU
 - b) Determinante
 - c) Matriz inversa a partir de resolver el sistema $AX = I$
 - d) Calcular la inversa usando la rutina `_dgetri()`

Trabajo realizado

Para octave simplemente usamos la función ***randi*** que genera números aleatorios del rango que queramos, en este caso del 1 al 10 y usamos la función LU con '***[L, U, P] = lu(A)***', donde en L devuelve la matriz triangular inferior L, en U la triangular superior U y en P la matriz de permutación indicando que filas han sido movidas.

Con la rutina ***LAPACKE_dgetrf*** conseguimos las matrices triangulares L y U sobrescribiendo la matriz A.

Para calcular la matriz inversa tanto del apartado c) como el d) simplemente usamos el método ***LAPACKE_dgetrf*** sustituyendo los datos y el método ***LAPACKE_dgetrfi***.

Código Programa:

Los códigos de la librería **Lapack** de los distintos apartados serán puesto a continuación con la referencia del apartado en el pie de la foto.

```
int N = 6;

double A[36] = { 7.8255, 2.2781, 2.8602, 2.7904, 6.8964, 0.7336,
                2.9553, 3.2102, 6.9913, 6.7538, 1.3183, 8.2233,
                1.5185, 8.2956, 7.9626, 9.0366, 1.2350, 7.2290,
                8.4791, 8.2218, 4.4159, 9.0853, 1.9090, 9.2586,
                7.8485, 5.7068, 4.4622, 7.4720, 1.4573, 4.9264,
                2.7083, 5.7183, 4.6566, 2.6051, 5.8504, 6.5488};

//double* A = (double*)mkl_malloc(N * N * sizeof(double), 64);
double* A2 = (double*)mkl_malloc(N * N * sizeof(double), 64);
lapack_int ipiv[6];

srand((unsigned int)time(NULL));

for (int i = 0; i < N * N; i++) {
    //A[i] = (double)rand() / (double)RAND_MAX*(double)10;
    A2[i] = A[i];
}

printf("\n\n\n      Matrix Aleatoria de 6x6");
printf("\n-----");

for (int e = 0; e < N*N ; e++) {
    if (e%6==0) {
        printf("\n|  %f", A[e]);
    }
    else {
        printf("|  %f", A[e]);
    }
}
}
```

Código principal 1

```
printf("\n\n\n      Factorizacion LU");
printf("\n-----");
lapack_int m = 6, n = 6, lda = 6;

LAPACKE_dgetrf(LAPACK_ROW_MAJOR, m, n, A, lda, ipiv);

printf("\n");

for (int e = 0; e < 6; e++) {
    printf("|  %i", ipiv[e]);
}
}
```

Código principal 2

```

printf("\n\n\n      Determinante");
printf("\n-----");
//Determinante (detA = detL * detU => detL = 1 y detU = {A11*A22*A33...Ann})

double detA = 1.0;

for (int i = 0; i < N; i+=N+1) {
    detA *= A[i];
}
printf("\n");
printf(" El determinante es %f", detA);

```

Código principal 3

```

//Inversa AX=I
double I[36] = { 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
                 0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
                 0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
                 0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
                 0.0, 0.0, 0.0, 0.0, 0.0, 1.0};

lapack_int nrhs = 6, ldb = 6;

LAPACKE_dgetrs(CblasRowMajor, 'N', n, nrhs, A, lda, ipiv, I, ldb);

printf("\n\n\n      Inversa AX=I");
printf("\n-----");

for (int e = 0; e < N * N; e++) {
    if (e % 6 == 0) {
        printf("\n| %f", I[e]);
    }
    else {
        printf("| %f", I[e]);
    }
}

```

Código principal 4

```

//Inversa _dgetri()

LAPACKE_dgetri(CblasRowMajor, n, A, lda, ipiv);

printf("\n\n Inversa con la funcion _dgetri");
printf("\n-----");

for (int e = 0; e < N * N; e++) {
    if (e % 6 == 0) {
        printf("\n| %f", A[e]);
    }
    else {
        printf("| %f", A[e]);
    }
}

```

Código principal 5

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (*Resultado de ejecución*).

```

Matrix Aleatoria de 6x6
-----
| 7.825500| 2.278100| 2.860200| 2.790400| 6.896400| 0.733600|
| 2.955300| 3.210200| 6.991300| 6.753800| 1.318300| 8.223300|
| 1.518500| 8.295600| 7.962600| 9.036600| 1.235000| 7.229000|
| 8.479100| 8.221800| 4.415900| 9.085300| 1.909000| 9.258600|
| 7.848500| 5.706800| 4.462200| 7.472000| 1.457300| 4.926400|
| 2.708300| 5.718300| 4.656600| 2.605100| 5.850400| 6.548800|

Factorizacion LU
-----
| 4| 3| 3| 6| 6| 6|

Determinante
-----
El determinante es 8.479100

Inversa AX=I
-----
| -0.126108| 0.004224| -0.181639| -0.178825| 0.432760| 0.136600|
| -0.165252| -0.206203| 0.021538| -0.124804| 0.303421| 0.201860|
| -0.215433| 0.074312| -0.105001| -0.500041| 0.658116| 0.258603|
| 0.367080| 0.038296| 0.279431| 0.444884| -0.757484| -0.456808|
| 0.289348| 0.005836| 0.140747| 0.257540| -0.544549| -0.149571|
| -0.054880| 0.105018| -0.105921| 0.131443| -0.124073| 0.051403|

Inversa con la funcion _dgetri
-----
| -0.126108| 0.004224| -0.181639| -0.178825| 0.432760| 0.136600|
| -0.165252| -0.206203| 0.021538| -0.124804| 0.303421| 0.201860|
| -0.215433| 0.074312| -0.105001| -0.500041| 0.658116| 0.258603|
| 0.367080| 0.038296| 0.279431| 0.444884| -0.757484| -0.456808|
| 0.289348| 0.005836| 0.140747| 0.257540| -0.544549| -0.149571|
| -0.054880| 0.105018| -0.105921| 0.131443| -0.124073| 0.051403|

```

Resultado de la ejecución

Matlab

A continuación los códigos de los distintos apartados pero esta vez en **Matlab**

```
a = rand(6)*10  
d = det(a)  
[L U P] = lu(a);
```

Código principal 1

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (*Resultado de ejecución*).

```
a =  
  
    7.8255    2.2781    2.8602    2.7904    6.8964    0.7336  
    2.9553    3.2102    6.9913    6.7538    1.3183    8.2233  
    1.5185    8.2956    7.9626    9.0366    1.2350    7.2290  
    8.4791    8.2218    4.4159    9.0853    1.9090    9.2586  
    7.8485    5.7068    4.4622    7.4720    1.4573    4.9264  
    2.7083    5.7183    4.6566    2.6051    5.8504    6.5488  
  
d =  
  
    1.6349e+04  
  
L =  
  
    1.0000         0         0         0         0         0  
    0.1791    1.0000         0         0         0         0  
    0.3485    0.0505    1.0000         0         0         0  
    0.3194    0.4532   -0.0008    1.0000         0         0  
    0.9229   -0.7782    0.8577    0.7076    1.0000         0  
    0.9256   -0.2790    0.4667    0.1013   -0.4423    1.0000  
  
U =  
  
    8.4791    8.2218    4.4159    9.0853    1.9090    9.2586  
         0    6.8232    7.1718    7.4095    0.8931    5.5709  
         0         0    5.0900    3.2130    0.6078    4.7150  
         0         0         0   -3.6522    4.8364    1.0706  
         0         0         0         0    1.8860   -8.2777  
         0         0         0         0         0   -8.0598  
  
P =  
  
         0         0         0         1         0         0  
         0         0         1         0         0         0  
         0         1         0         0         0         0  
         0         0         0         0         0         1  
         1         0         0         0         0         0  
         0         0         0         0         1         0
```

Resultado de la ejecución

Practica 2

Descripción

1. Repetir la factorización LU y el cálculo de la matriz inversa realizado en la práctica anterior utilizando la rutina LAPACKE_dgesv().
2. Realizar una comparación entre la operación sobre matrices generales y sobre matrices banda.
 - a) Crear matrices A (tridiagonal) y B, y rellenarlas con valores aleatorios (media 0 y varianza 1)
 - b) Codificar la matriz A en forma compacta A_banda, añadiendo una fila auxiliar nula al principio.
 - c) Resolver a partir de la matriz general (LAPACKE_dgesv, comprobar el código de error).
 - d) Resolver a partir de la matriz banda (LAPACKE_dgbv, comprobar el código de error).
 - e) Comparar los tiempos c y d promediando entre diferentes ejecuciones.

Trabajo realizado

Generamos una matriz aleatoria tridiagonal de tamaño 6x6 y una matriz general del mismo tamaño y la rellenamos con valores aleatorios.

A partir de la matriz tridiagonal realizamos la codificación utilizando un método que extrae las bandas de la matriz tridiagonal y las almacena en la matriz equivalente I y añadimos la fila auxiliar rellenándolas con ceros.

Y usamos los métodos *LAPACKE_dgesv* y *LAPACKE_dgbv* rellenándolas con los valores pertinentes que se requieren y comprobando los tiempos de ejecución y los resultados de ejecución de la función.

Código Programa:

Los códigos de la librería **Lapack** de los distintos apartados serán puesto a continuación con la referencia del apartado en el pie de la foto.

```
//Inversa _dgesv()

printf("\n\n\n      Inversa _dgesv");
printf("\n-----");

double I2[36] = { 1.0, 0.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 1.0, 0.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 1.0, 0.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 1.0, 0.0,
                  0.0, 0.0, 0.0, 0.0, 0.0, 1.0 };

LAPACKE_dgesv(CblasRowMajor, n, nrhs, A2, lda, ipiv, I2, ldb);

printf("\n");
for (int e = 0; e < N * N; e++) {
    if (e % 6 == 0) {
        printf("\n| %f", I2[e]);
    }
    else {
        printf("| %f", I2[e]);
    }
}

std::getchar();
return 0;
```

Código principal 1

```

std::default_random_engine generator;
std::normal_distribution<double> rand(0.0,1.0);

int N = 6;

double* A = (double*)mkl_malloc(N * N * sizeof(double), 64);
double* B = (double*)mkl_malloc(N * N * sizeof(double), 64);

//Generar matriz de banda (A) y matriz normal (B)

for (int i = 0; i < N * N; i++) {
    if ((i-1)%(N+1)==0 || i%(N+1)==0 || (i+1)%(N+1)==0 ) {
        A[i] = rand(generator);
    }
    else {
        A[i] = 0.0;
    }
    B[i] = rand(generator);
}

```

Código principal 2

```

//Codificar Matriz A (A_banda)
int N_banda_x = 6;
int N_banda_y = 4;

double* A_banda = (double*)mkl_malloc(N_banda_x * N_banda_y * sizeof(double), 64);

for (int i = 0; i < 24;i++) {
    A_banda[i] = 0;
}

ceroMatrix(A_banda, N_banda_x * N_banda_y);

addBand(A, 1, A_banda, 6 , 1);
addBand(A, 0, A_banda, 6, 2);
addBand(A, -1, A_banda, 6, 3);

printf("\n\n");
for (int e = 0; e < N_banda_x * N_banda_y; e++) {
    if (e % 6 == 0) {
        printf("\n| %f", A_banda[e]);
    }
    else {
        printf("| %f", A_banda[e]);
    }
}
}

```

Código principal 3

```

int ceroMatrix(double* matrix, int length) {
    for (int i = 0; i < length ; i++) {
        matrix[i] = 0;
    }
    return 0;
}

```

Código principal 3.1

```

int addBand(double* matrix, int bandPosition, double* matrixResult, int matrixResultY, int matrixResultRow) {
    if(bandPosition>0){
        for (int i = 0; i < matrixResultY; i++) {
            matrixResult[matrixResultRow* matrixResultY+ i + bandPosition] = matrix[i* (matrixResultY+1)+1];
        }
    }
    else if (bandPosition < 0) {
        bandPosition *= -1;
        for (int i = 0; i < matrixResultY; i++) {
            matrixResult[matrixResultRow * matrixResultY + i] = matrix[bandPosition*matrixResultY+i*(matrixResultY+1)];
        }
    }
    else {
        for (int i = 0; i < matrixResultY; i++) {
            matrixResult[matrixResultRow * matrixResultY + i] = matrix[i * (matrixResultY + 1)];
        }
    }
    return 0;
}

```

Código principal 3.2

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (*Resultado de ejecución*).

```
Inversa _dgesv
-----
-0.126108| 0.004224| -0.181639| -0.178825| 0.432760| 0.136600
-0.165252| -0.206203| 0.021538| -0.124804| 0.303421| 0.201860
-0.215433| 0.074312| -0.105001| -0.500041| 0.658116| 0.258603
0.367080| 0.038296| 0.279431| 0.444884| -0.757484| -0.456808
0.289348| 0.005836| 0.140747| 0.257540| -0.544549| -0.149571
-0.054880| 0.105018| -0.105921| 0.131443| -0.124073| 0.051403_

| -0.146382 | -1.871384 | 0.000000 | 0.000000 | 0.000000 | 0.000000
| 1.055466 | 1.174568 | -1.049443 | 0.000000 | 0.000000 | 0.000000
| 0.000000 | -0.888707 | 1.019224 | -0.482589 | 0.000000 | 0.000000
| 0.000000 | 0.000000 | -2.301443 | -0.061527 | 1.354326 | 0.000000
| 0.000000 | 0.000000 | 0.000000 | 0.439499 | 0.645743 | -1.685987
| 0.000000 | 0.000000 | 0.000000 | 0.000000 | -0.103692 | 0.624049

| 0.134530| 0.460650| -0.214253| 0.163712| -0.827944| 0.298595
| 0.010215| -0.546841| 0.660682| -0.625276| 1.485964| -0.829081
| -2.559122| -0.539781| -0.628956| 0.339587| -0.121306| 2.108857
| -0.371003| -0.287389| -1.059349| 1.455024| 0.925328| -0.243275
| 1.515609| 0.197497| 1.008865| 0.438945| -0.128149| 1.776432
| -0.613857| 0.469861| -0.582398| 0.668493| 0.149386| 1.537275

| 0.000000| 0.000000| 0.000000| 0.000000| 0.000000| 0.000000
| 0.000000| -1.871384| -1.049443| -0.482589| 1.354326| -1.685987
| -0.146382| 1.174568| 1.019224| -0.061527| 0.645743| 0.624049
| 1.055466| -0.888707| -2.301443| 0.439499| -0.103692| 0.000000

Tiempo de ejecucion de la funcion DGESV 1081 ms valor de retorno de la funcion 0
Tiempo de ejecucion de la funcion DGBSV 1488 ms valor de retorno de la funcion 0_
```

Resultado de la ejecución

Matlab

A continuación los códigos de los distintos apartados pero esta vez en **Matlab**

```
%a
N = 6;
a = normrnd(0,1);
b = normrnd(0,1);
c = normrnd(0,1);
A = diag(a*ones(1,N)) + diag(b*ones(1,N-1),1) + diag(c*ones(1,N-1),-1)
B = normrnd(0,1,[N,N])
time_DGESV = 0;
time_DGBSV = 0;

%b
A_compacta = [[0;diag(A,-1)],diag(A), [diag(A,1);0]]

%c
X = linsolve(A,B)

%d
X1 = linsolve(A_compacta,B)

%e
for i = 1:100
    time_DGESV = time_DGESV + Lapacke_DGESV(A,B);
end

time_DGESV = time_DGESV / 100

for i = 1:100
    time_DGBSV = time_DGBSV + Lapacke_DGBSV(A_compacta,B);
end

time_DGBSV = time_DGBSV / 100

function time_DGESV = Lapacke_DGESV(A,B)
    tic;
    X = linsolve(A,B);
    time_DGESV = toc;
end

function time_DGBSV = Lapacke_DGBSV(A_compacta,B)
    tic;
    X = linsolve(A_compacta,B);
    time_DGBSV = toc;
end
```

Código principal 1

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (*Resultado de ejecución*).

```
>> Practica2

A =

    -0.5174    0.0243         0         0         0         0
    -0.1624   -0.5174    0.0243         0         0         0
         0   -0.1624   -0.5174    0.0243         0         0
         0         0   -0.1624   -0.5174    0.0243         0
         0         0         0   -0.1624   -0.5174    0.0243
         0         0         0         0   -0.1624   -0.5174

B =

    -0.6057   -0.0076   -0.8015    0.7291   -0.9392    1.5245
     0.2717    0.7103   -1.8529    0.9008    1.2374   -0.2620
     0.3586   -0.4625    0.0532   -1.3256    0.0330   -1.7307
    -0.3523   -0.5279   -0.4364   -1.5246    0.1236    1.0135
     0.1324   -0.3458    0.2800    0.5024    0.2739   -0.9277
     2.6736    0.5646    0.0846    0.3064    0.7542    1.2437

A_compacta =

         0   -0.5174    0.0243
    -0.1624   -0.5174    0.0243
    -0.1624   -0.5174    0.0243
    -0.1624   -0.5174    0.0243
    -0.1624   -0.5174    0.0243
    -0.1624   -0.5174         0

X =

     1.1285   -0.0463     1.6905   -1.4629     1.6799   -2.8745
    -0.8970   -1.2959     3.0043   -1.1404   -2.8806     1.5371
    -0.3758     1.3300   -0.9934     3.0108     0.8160     2.7359
     0.7647     0.6225     1.1137     1.9276   -0.5152   -2.7005
    -0.7280     0.4156   -0.8854   -1.5806   -0.4300     2.4910
    -4.9393   -1.2217     0.1143   -0.0962   -1.3228   -3.1857
```


X1 =

-4.3621	0.9171	-1.9247	6.7186	-8.3523	12.3254
-3.7988	-1.3791	0.4405	-2.7008	1.1636	-6.2721
-105.7367	-29.6539	-23.5921	-27.4789	-13.8674	-70.7554

time_DGESV =

3.7650e-06

time_DGBSV =

3.0250e-06

Resultado de la ejecución

Práctica 3

Descripción

Generar ejemplos sencillos para otros esquemas de factorización con Octave:

- a) Cholesky
- b) QR
- c) SVD
- d) Cálculo de autovalores y autovectores

Trabajo realizado

Para la realización de este ejercicio práctico simplemente se basa en simplemente usar las instrucciones que provee Octave para las diferentes factorizaciones como son: chol (cholesky), qr (QR), svd (SVD) y eig (autovalores y autovectores).

Matlab

A continuación los códigos de los distintos apartados pero esta vez en **Matlab**

```
A = [100, 2, 3, 4, 5, 6,
      2, 200, 3, 4, 5, 6,
      3, 3, 300, 4, 5, 6,
      4, 4, 4, 400, 5, 6,
      5, 5, 5, 5, 500, 6,
      6, 6, 6, 6, 6, 600]

R = chol(A);
fprintf("\nFactorizazion con Cholesky\nR*R':\n");
Cholesky = A*A'

[Q,R] = qr(A);
fprintf("\nFactorizazion con QR\nQ*R:\n");
QR = A*R

[U, S, V] = svd(A);
fprintf("\nFactorizazion con SVD\nU*S*V':\n");
SVD = U*S*V'

fprintf("\nCalculo de auto-valores y auto-vectores.\n");
[autoVectores,autoValores] = eig(A)
```

Código principal 1

Resultado Ejecución:

El resultado de la ejecución lo podemos ver en la imagen inferior (*Resultado de ejecución*).

A =

100	2	3	4	5	6
2	200	3	4	5	6
3	3	300	4	5	6
4	4	4	400	5	6
5	5	5	5	500	6
6	6	6	6	6	600

Factorizazion con Cholesky

R*R':

Cholesky =

10090	686	1283	2081	3081	4284
686	40090	1583	2481	3581	4884
1283	1583	90095	2885	4086	5490
2081	2481	2885	160109	4596	6102
3081	3581	4086	4596	250136	6720
4284	4884	5490	6102	6720	360180

Factorizazion con QR

Q*R:

QR =

1.0e+05 *

-0.1004	-0.0108	-0.0219	-0.0372	-0.0567	-0.0087
-0.0020	-0.4004	-0.0242	-0.0400	-0.0600	-0.0123
-0.0030	-0.0062	-0.9000	-0.0423	-0.0624	-0.0148
-0.0040	-0.0083	-0.0128	-1.5990	-0.0640	-0.0161
-0.0050	-0.0103	-0.0160	-0.0220	-2.4968	-0.0162
-0.0060	-0.0124	-0.0192	-0.0264	-0.0340	3.5805

Factorizazion con SVD

$U \cdot S \cdot V'$:

SVD =

100.0000	2.0000	3.0000	4.0000	5.0000	6.0000
2.0000	200.0000	3.0000	4.0000	5.0000	6.0000
3.0000	3.0000	300.0000	4.0000	5.0000	6.0000
4.0000	4.0000	4.0000	400.0000	5.0000	6.0000
5.0000	5.0000	5.0000	5.0000	500.0000	6.0000
6.0000	6.0000	6.0000	6.0000	6.0000	600.0000

Calculo de auto-valores y auto-vectores.

autoVectores =

0.9995	0.0168	0.0133	-0.0123	-0.0122	0.0130
-0.0178	0.9991	0.0263	-0.0184	-0.0162	0.0163
-0.0138	-0.0280	0.9983	-0.0365	-0.0243	0.0217
-0.0125	-0.0189	-0.0390	-0.9973	-0.0479	0.0324
-0.0118	-0.0158	-0.0244	0.0513	-0.9962	0.0634
-0.0113	-0.0143	-0.0195	0.0304	0.0659	0.9970

autoValores =

99.7462	0	0	0	0	0
0	199.7088	0	0	0	0
0	0	299.7237	0	0	0
0	0	0	399.8297	0	0
0	0	0	0	500.1084	0
0	0	0	0	0	600.8831

Resultado de la ejecución