Catálogo Grupal de Algoritmos

Integrantes:

- Brayan Alfaro González - Carné 2019380074
- Sebastián Alba Vives - Carné 2017097108
- Kevin Zeledón Salazar - Carné 2018076244
- Daniel Camacho González - Carné 2017114058

1 Sistemas de ecuaciones

1.1 Método de Eliminación Gaussiana

Código 1: Método de eliminación Gaussiana en Lenguaje M

```
% Resuelve el sistema de ecuaciones Ax=B
% Entradas: A: Matriz de coeficientes
%           B: Matriz de t rminos independientes
% Salidas:  X: Matriz resultante
function [X] = gaussiana(A, B)
  n = length(A);
  X=zeros(1,n);
  Ab=[A transpose(B)]

  for k=1:n-1
    for i=k+1:n
      mik = Ab(i,k)/Ab(k,k)
      for j=k:n+1
        Ab(i,j)=Ab(i,j)-mik*Ab(k,j);
      end
    end
  end
  X=sust_atras(Ab(:,1:n),B);
end

function [X] = sust_atras(A, B)
  n = length(A);
  X=zeros(1,n);
  for i=n:-1:1
    sum = 0;
    for j=i+1:n
      sum = sum + A(i,j)*X(j);
    end
    X(i) = (1/(A(i,i)))*(B(i)-sum);
  end
end
```

1.2 Método de Factorizacion LU

Código 2: Método de Factorización LU en Python

```
import numpy as np

#Librer as: Se usa la libreria numpy

# Funcion para calcular la solucion de un sistema de ecuaciones de acuerdo
# al m todo de la factorizaci n LU
# Entradas:
#       A : matriz de numpy del sistema a resolver
#       b : vector de numpy para el cual resolver el sistema
# Salida:
#       x : vector de solucion

def fact_lu(A,b):
    [m, n] = A.shape
    [k,] = b.shape

    if not verificacion_lu(A):
        return np.zeros(b.shape)
    elif k!=m:
        return np.zeros(b.shape)
    else:
        (L,U) = fact_lu_aux(A)
        y = sustitucion_adelante(L, b)
        x = sustitucion_atras(U, y)
        return x

# Funcion para verificar si la matriz de entrada cumple las condiciones para la
# factorizacion LU.
# Entradas:
#       A : matriz de numpy del sistema a resolver
# Salida:
#       booleano : indica si A cumple las condiciones

def verificacion_lu(A):
    [a,b] = A.shape
    if a!=b :
        return False
    for i in range(1,a):
        if(det(A[0:i,0:i])==0):
            return False
    return True

# Funcion para realizar la factorizaci n LU de una matriz
# Entradas:
#       A : matriz de numpy a la cual calcular la factorizacion LU
# Salida:
#       L : matriz L de la factorizacion
#       U : matriz U de la factorizacion
def fact_lu_aux(A):
    [a, b] = A.shape
    U = A.astype(np.float64)
```

```
    L = np.zeros(A.shape)
    np.fill_diagonal(L, 1)
    for j in range(b):
        for i in range(j+1,a):
            x = U[i, j]/U[j, j]
            U[i,0:b] -= x*U[j, 0:b]
            L[i, j] = x
    return (L,U)

# Funcion para realizar la sustitucion hacia adelante en un sistema de ecuaciones
# triangular inferior
# Entradas:
#       A : matriz de numpy triangular inferior
#           a la aplicarle la sustitucion hacia adelante
#       b : vector de numpy para el cual resolver el sistema
# Salida:
#       x : solucion del sistema
def sustitucion_adelante(A, b):
    [n, m] = A.shape
    x = np.zeros(m)
    for i in range(n):
        x[i] = (b[i] - np.dot(A[i,0:m],x))/A[i,i]
    return x

# Funcion para realizar la sustitucion hacia atras en un sistema de ecuaciones
# triangular superior
# Entradas:
#       A : matriz de numpy triangular superior
#           a la aplicarle la sustitucion hacia atras
#       b : vector de numpy para el cual resolver el sistema
# Salida:
#       x : solucion del sistema
def sustitucion_atras(A, b):
    [n, m] = A.shape
    x = np.zeros(m)
    for i in range(-(n-1),1):
        x[-i] = (b[-i] - np.dot(A[-i,0:m],x))/A[-i,-i]
    return x

# Funcion para calcular el determinante de una matriz cuadrada
# Entradas:
#       A : matriz de numpy cuadrada
# Salida:
#       determinante : determinante de A
def det(A):
    [a,b] = A.shape
    determinante = 0
    sign = 1
    if b == 1:
        determinante = A[0,0]
    else:
        # Se calcula el determinante de manera recursiva
        for i in range(b):
```

```
            determinante += sign*A[0, i]*det(np.concatenate((A[1:a,0:i], A[1:a,i+1:b
                ]), axis=1))
            sign = -1*sign
    return determinante

#Ejemplo Num rico
A = np.array([[4,-2, 1],
              [20,-7, 12],
              [-8, 13, 17]])
b = np.array([11,70,17])
x = fact_lu(A, b)
print("Matriz: ")
print(A)
print("\nVector b: ")
print(b)
print("\nSolucion x: ")
print(x)
```

1.3 Método de la Factorizacion de Cholesky

Código 3: Método de Factorizacion de Cholesky en C++

```
#include <iostream>
#include <vector>
#include <cmath>
using namespace std;



vector<vector<double> > cholesky(double **A, int n){

  // Inicializacion de la matriz decompuesta, como un vector de vectores
  // El objeto esta lleno de ceros
  vector<vector<double> > L(n, vector<double>(n, 0));

    double sum1 = 0.0, sum2 = 0.0;
    L[0][0] = sqrt(A[0][0]);

    for (int i = 0; i < n; i++){
      L[i][0] = A[i][0]/L[0][0];
  }


    for (int i = 1; i < n; i++){

        for (int k = 0; k <= i - 1; k++){
            sum1 += pow(L[i][k], 2.0);
        }

        L[i][i] = sqrt(A[i][i] - sum1);

        for (int j = i + 1; j < n; j++){
            for (int k = 0; k <= i - 1; k++){
                sum2 += L[j][k]*L[i][k];
            }

            L[j][i] = 1.0/L[i][i]*(A[j][i] - sum2);
            sum2 = 0;
        }

        sum1 = 0;
    }

    return L;
}

// Esta funcion devuelve la matriz transpuesta de una matriz dada
vector<vector<double> > transpose(vector<vector<double> > A, int n){

  vector<vector<double> > T(n, vector<double>(n, 0));

  for (int i = 0; i < n; i++){
    for (int j = 0; j < n; j++){
```

```
      T[i][j] = A[j][i];
    }
  }
  return T;
}

vector<double> fact_cholesky(double **A, double *b, int n){

  // Vector of vectors (double)
  vector<vector<double> > L(n, vector<double>(n, 0));
  vector<vector<double> > Lt(n, vector<double>(n, 0));
  L = cholesky(A, n);
    Lt = transpose(L, n);

  vector<double> x(n, 0);
  vector<double> z(n, 0);

  double sum1, sum2;

  // Sustitucion hacia adelante
  z[0] = b[0]/L[0][0];
  for (int i = 1; i < n; i++){
    sum1 = 0.0;
    for (int j = 0; j < i; j++){
      sum1 += L[i][j]*z[j];
    }

    z[i] = (b[i] - sum1)/L[i][i];
  }

  for (int i = 0; i < n; i++){
      cout << "y[" << i << "] = " << z[i] << "\n";
    }
  cout << "\n";
  // Sustitucion hacia atras

  x[n-1] = z[n-1]/Lt[n-1][n-1];
  for (int i = n - 2; i > -1; i--){
    sum2 = 0.0;
    for (int j = i + 1; j < n; j++){
      sum2 += Lt[i][j]*x[j];
    }
    x[i] = (z[i] - sum2)/Lt[i][i];

  }

  return x;


}

int main(){

  // n: Tamao de la matriz (simetrica)
```

```
int n = 4;
// A: Matriz a ser descompuesta por la Descomposicion de Choleski
  double A[][4] = {{25,15,-5,-10}, {15, 10, 1, -7}, {-5, 1, 21, 4}, {-10, -7, 4,
     18}};
  // b: Vector de valores independientes del sistema de ecuaciones lineales  A x  =
      b
  double b[] = {-25, -19, -21, -5};

/* La funcion choleski() y la funcion solve_chol()
reciben como argumento la matriz de coeficientes en forma de un puntero de punteros
debido a esto, no se puede pasar a esta los arrays A[][n]. El vector de
    coeficientes
independientes b se recibe de igual forma

Para esto, se inicializan los punteros de punteros A1 y b1 en las siguientes lineas
    ,
esta es una forma de crear arrays dinamicos
*/

  double **A1 = NULL;
  double *b1 = NULL;
  A1 = new double *[n];
  b1 = new double [n];
  for(int i = 0; i < n; i++){
    A1[i] = new double [n];
}

  for(int i = 0; i < n; i++){
    b1[i] = b[i];
    for (int j = 0; j < n; j++){
      A1[i][j] = A[i][j];
  }
}

  // Vector de vectores (double) que representa una matriz
vector<vector<double> > L(n, vector<double>(n, 0));
// CLlamada a la funcion que descompone la matriz dada  s e g n  la descomposicion de
    Choleski
  L = cholesky(A1, n);

// Se imprime en pantalla la matriz descompuesta
  for (int i = 0; i < n; i++){
  for (int j = 0; j < n; j++){
    cout << "L[" << i << "][" << j << "] = " << L[i][j] << "\t";
  }
  cout << "\n";
}

cout << "\n";

  // Solucion del sistema lineal  A x  = b
  vector<double> u(n, 0);
  u = fact_cholesky(A1, b1, n);
```

```
  // Se imprime en pantalla la solucion del sistema lineal  A x  = b
    for (int i = 0; i < n; i++){
      cout << "x[" << i << "] = " << u[i] << "\n";
    }

    cout << "\nPrograma terminado con exito :D\n";

    return 0;

}
```

1.4 Método de Thomas

Código 4: Método deThomas en Python

```
import numpy as np
from math import *

# Esta funcion toma una matriz y determina si es tridiagonal
#  Entradas  : A: Matriz a determinar tridiagonalidad
# Salidas: Verdadero o falso, dependiendo de la tridiagonalidad
def is_tridiagonal(a):
    dims=a.shape

    for i in range(dims[0]):
        for j in range(dims[1]):
            if(j>i+1 and a[i,j]!=0):
                return False
            elif(j<i-1 and a[i,j]!=0):
                return False
    return True

# Este metodo calcula la matriz solucion al sistema de ecuaciones Ax=B
# Entradas: a: Matriz de coeficientes
#           b: Matriz de termino independientes
# Salidas: Matriz solucion al sistema
def thomas(a,b):
    if(is_tridiagonal(a)):
        n=len(b)
        a_s=[]
        b_s=[]
        c_s=[]
        d_s=b.transpose();
        p_s=[]
        q_s=[]
        xk =np.zeros((1,n))
        for i in range(n):
            if(i==0):
                a_s.append(0)
                b_s.append(a[i,i])
                c_s.append(a[i,i+1])
            elif(i==n-1):
                a_s.append(a[i,i-1])
                b_s.append(a[i,i])
                c_s.append(0)
            else:
                a_s.append(a[i, i - 1])
                b_s.append(a[i, i])
                c_s.append(a[i, i + 1])
        n=len(b)
        qi=0
        pi=0
        for i in range(n):
            if(i==0):
                p_s.append(c_s[i]/b_s[i])
                q_s.append(d_s[0,i]/b_s[i])
```

```
            else:
                if(i!=n-1):
                    p_s.append(c_s[i]/(b_s[i]-p_s[i-1]*a_s[i]))
                    q_s.append((d_s[0,i]-q_s[i-1]*a_s[i])/(b_s[i]-p_s[i-1]*a_s[i]))
                else:
                    q_s.append((d_s[0, i] - q_s[i - 1] * a_s[i]) / (b_s[i] - p_s[i - 1] * a
        for j in range(n-1,-1,-1):
            if(j==n-1):
                xk[0,j]=q_s[j]
            else:
                xk[0,j]=q_s[j]-p_s[j]*xk[0,j+1]
        return xk
    else:
        print("La matriz no es tridiagonal")
        return None

a=np.matrix(([-4,1,0,0,0],
             [1,-4,1,0,0],
             [0,1,-4,1,0],
             [0,0,1,-4,1],
             [0,0,0,1,-4]))
b=np.matrix(([1],
             [1],
             [1],
             [1],
             [1]))

xk=thomas(a,b)
if(xk is not None):
    print("Las soluciones del sistema de ecuaciones son: \nxk="+str(xk))
```

1.5 Método de Jacobi

Código 5: Método de Jacobi en Lenguaje C++

```
#include <iostream>
#include <armadillo>
#include "matplotlibcpp.h"
using namespace std;
using namespace arma;
namespace plt = matplotlibcpp;
/**
 * Funcion para calcular la solucion de un sistema de ecuaciones
 * mediante el metodo de jacobi
 * @param A Matriz cuadrada de armadillo
 * @param b Vector de terminos independientes de armadillo
 * @param x0 Solucion inicial
 * @param tol Tolerancia
 * @param iterMax Cantidad maxima de iteraciones
 * @return <x, error> Tupla del vector de solucion y el error de la aproximacion
 */
tuple<vec, double> jacobi(mat A, vec b, vec x0, double tol, int iterMax){
    mat L(size(A), fill::zeros);
    mat D(size(A), fill::zeros);
    mat U(size(A), fill::zeros);
    //Se separa A en L , D , U
    for(int i = 0; i< A.n_rows; i++){
        for(int j = 0; j< A.n_cols; j++){
            if(i>j){
                U(i,j) = A(i,j);
            }else if(i<j){
                L(i,j) = A(i,j);
            }else{
                D(i,j) = A(i,j);
            }
        }
    }

    // Se realiza el metodo iterativo
    mat T = -D.i()*(L+U);
    mat c = D.i()*b;
    vector<double> graphX(iterMax), graphY(iterMax);

    int i = 0;
    vec x = std::move(x0);
    double error;
    do{
        x = T*x + c;
        i++;
        error = norm(A*x-b);
        graphX.insert(graphX.begin(),i);
        graphY.insert(graphY.begin(), error);
    }while  (error> tol && i<iterMax);

    //Se grafica
    graphX.resize(i);
```

```
    graphY.resize(i);
    plt::plot(graphX,graphY);
    plt::suptitle("Error en el M todo de Jacobi");
    plt::xlabel("Iteraciones");
    plt::ylabel("Error");
    plt::show();

    // Se retorna el resultado
    return make_tuple(x, error);
}
/**
 * Ejemplo Numerico
 */
int main(int argc, char const *argv[])
{
    mat A("5 1 1; 1 5 1; 1 1 5");
    vec b("7 7 7");
    vec x0("0 0 0");

    tuple<vec, double> result  = jacobi(A, b, x0, 1e-8, 50);

    // Se imprime el resultado

    vec x = get<0>(result);
    cout.precision(15);
    cout.setf(ios::fixed);
    x.raw_print("X: ");

    double error = get<1>(result);
    cout<< "Error: ";
    cout<< error <<endl;

    return 0;
}
```

1.6 Método de Gauss-Seidel

Código 6: Método de Gauss-Seidel en Lenguaje M

```
function [xs, errorf] = gauss_seidel( A, b, xo, tol, imax )

    L = tril(A);
    U = triu(A);
    D = -(A - L - U);
    L = L - D;
    U = U - D;
    x(:, 1) = xo;
    error = zeros(1, imax);
    i = 1;
    check = 0;


    if abs(inv(D + U)*L) < 1

        while check == 0 && i < imax
            i = i + 1;
            x(:, i) = -inv(D + U)*L*x(:,i-1) + inv(D + U)*b;
            error(1, i-1) = abs((sum(x(:,i)) - sum(x(:, i-1)))/sum(x(:,i-1)));
            if error(1, i-1) < tol
                check = 1;
            end
        end
        errorf = error(1,end);
        xs = x(:, i);
        fprintf('x = \n')
        display(xs)
        fprintf('\nerror = %5.8f', error)
        plot(1:i-1,error(1, 2:i), 'marker', 'o', 'linewidth', 1.5)
        xlabel('Iteraciones')
        ylabel('Error')
    else
        fprintf('The Matrix A is not suitable for Gauss - Seidel Iteration method\n')
    end


end
```

1.7 Método de Relajación

Código 7: Método de Relajación en Python

```
import numpy as np
from math import *
import random
import matplotlib.pyplot

'''
    Funcion desarrollada para realizar la sustitucion hacia atras en una matriz triangular

    Entradas: a=Matriz de coeficientes.
              b=Vector de terminos independientes.

    Salidas: xk=Vector columna de soluciones que se obtiene de resolver el sistema formado
'''
def sust_atras(a,b):
    n = len(b)
    xk = np.zeros((1, n))
    for i in range(n):
        suma = 0
        for j in range(i):
            suma += a[i, j] * xk[0, j]
        xi = (1 / a[i, i]) * (b[i] - suma)
        xk[0, i] = xi
    return np.asmatrix(xk).transpose()

'''
    Funcion desarrollada para determinar si una matriz es simetrica.

    Entradas: a= Matriz a analizar.

    Salidas:  Booleano resultante de analizar la matriz a.
'''
def is_symetric(a):
    dims=a.shape
    a_t=a.transpose()
    for i in range(dims[0]):
        for j in range(dims[1]):
            if(a[i,j]!=a_t[i,j]):
                print("La matriz no es simetrica!")
                return False
    return True


'''
    Funcion desarrollada para determinar si una matriz es positiva definida.

    Entradas: a= Matriz a analizar.

    Salidas:  Booleano resultante de analizar la matriz a.
'''
def is_positive_defined(a,n):
    dims = a.shape
```

```
    n = dims[0]
    for i in range(n):
        temp = a[0:i + 1, 0:i + 1]
        if (np.linalg.det(temp) < 0):
            print("La matriz no es positiva definida!")
            return False
    return True


'''
    Funcion que implementa el metodo de relajacion estudiado.

    Entradas:
              a= Matriz de coeficientes.
              b=Matriz de terminos independientes.
              x0=Vector columna de valores x iniciales.
              tol=Tolerancia permitida para el metodo.
              iterMax= Numero de iteraciones maximas a realizar.
    Salidas:
              xk=Vector de valores en x que representan la solucion del sistema de ecuacion
              error=Error obtenido del resultado xk encontrado.
              k= Numero de iteraciones que le tomo al metodo para encontrar la solucion xk.
'''
def relajacion(a,b,x0,tol,iterMax):
    dims = a.shape
    n = dims[0]

    #Verificacion para determinar si a es cuadrada.
    if(dims[0]!=dims[1]):
        print("La matriz no es cuadrada!")
        return (None,None,None)
    #Verificacion para determinar si a es simetrica,invertible y positiva definida.
    if(np.linalg.det(a)!=0 and is_symetric(a) and is_positive_defined(a,n)):

        # Definicion de un valor random para w en ]0,2[.
        w=0
        while(w==2 or w==0):
            w=random.uniform(0,2)

        #Definicion de los valores D,L,U.
        d=np.asmatrix(np.diag(np.diag(a)))
        l=np.asmatrix(np.tril(a,-1))
        u=np.asmatrix(np.triu(a,1))

        #Valor que acompana la incognita en el sistema de ecuaciones e*zK=f.
        e=d+w*l

        #Valor del sistema de ecuaciones e*zK=f.
        f=((1-w)*d-w*u)*x0

        #Resolucion del sistema de ecuaciones e*zK=f.
        zk=sust_atras(e,f)

        #Valor del sistema de ecuaciones e*c=g.
```

```
        g=w*b

        #Resolucion del sistema de ecuaciones e*c=g.
        c=sust_atras(e,g)

        k=1
        xk=x0
        iteraciones=[]
        errores=[]

        #Ciclo iterativo para realizar las iteraciones requeridas por el metodo para aproxi
        while(k<iterMax):
            #Recalculado los valores f y zk.
            f=((1-w)*d-w*u)*xk
            zk = sust_atras(e,f)

            #Calculando x_k+1.
            xk=zk+c

            #Calculo del error asociado a la iteracion actual.
            error=np.linalg.norm(a*xk-b)

            #Verificacion de la condicion de parada asociada a la tolerancia definida.
            if(error<=tol):
                break
            k+=1
            iteraciones.append(k)
            errores.append(error)

       #Graficacion de los errores
        matplotlib.pyplot.plot(iteraciones,errores)
        matplotlib.pyplot.ylabel("Errores |f(Xk)|")
        matplotlib.pyplot.title("Graficaci n Errores vs Iteraciones")
        matplotlib.pyplot.xlabel("Iteraci n")

        return (xk,error,k)

    return (None,None,None)


#---> Caso de ejemplo <---#

a=np.matrix(([4,3,0],
             [3,4,-1],
             [0,-1,4]))
b=np.matrix(([24],
             [30],
             [-24]))

x0=np.asmatrix(np.zeros(a.shape[0])).transpose()
tol=1e-10
iterMax=2500
valores=relajacion(a,b,x0,tol,iterMax)
if(valores[0] is not None):
```

```
    print("Las soluciones del sistema de ecuaciones obtenidas en "+str(valores[2])+" iterac
    matplotlib.pyplot.show()
```

1.8 Método de Pseudoinversa

Código 8: Método de Pseudoinversa en C++

```
#include <iostream>
#include <armadillo>

using namespace arma;

// Este metodo iterativo calcula la pseudoinversa de una matriz y la utiliza para dar
// solucion al sistema de ecuaciones Ax=B
// Entradas: A: Matriz de coeficientes
//           B: Matriz de t rminos independientes
//           iterMax: Cantidad maxima de iteraciones
//           tol: Tolerancia de las iteraciones
// Salidas: La matriz solucion x al sistema de ecuaciones Ax=B
Mat<double> pseudoinversa(Mat<double> A, Mat<double> B, int iterMax, double tol){
  int n = A.n_rows;
  int m = A.n_cols;

  double alpha = eig_sym(A*trans(A)).max();
  cout << alpha<<endl;

  Mat<double> X, Xo, Xkm1;
  X = (1/alpha)*trans(A);
  Mat <double> I(n,m, fill::eye); // MATRIX IDENTIDAD

  int k=0;

  while(k<iterMax){
    cout <<  -(A*X) << endl;
    X=X*(2*I-A*X);

    if(norm((A*X*A)-A)<tol){
      break;
    }

    k=k+1;
  }

  return X*B;
}

int main(void){
  Mat<double> A = {{1,2,4},{2,-1,1},{1,0,1}};
  Col<double> B = {4,3,9};
  cout << pseudoinversa(A,B,200, 0.0000000001)<<endl;;

  return 0;
}
```