

## Práctica 2: Algoritmo del simplex.

### Pregunta 1:

Crea un código del tipo function que dado un problema de optimización de la forma

$$\max z = c^T x$$

$$Ax \leq b$$

$$x \geq 0$$

(donde las componentes del vector  $b$  cumplen que  $b_i \geq 0$ ) te devuelva una solución óptima o en caso contrario te especifique que no está acotada la solución. Para ello los inputs deberán ser:

- El vector de costes de la función objetivo, es decir, los coeficientes de las variables de la función objetivo.
- La matriz  $A$  de los coeficientes de las restricciones.
- El vector de recursos  $b$ .

Limpieza del entorno de trabajo

```
clear, clc
```

Definimos un problema de programación lineal (acotado) a resolver mediante el la función simplex() definida al final de este documento

$$\max z = 5x_1 + 4x_2 + 3x_3 + x_4$$

$$x_1 + x_2 + x_3 + x_4 \leq 12$$

$$3x_1 + 2x_2 + 3x_3 + x_4 \leq 5$$

$$2x_1 + x_2 + x_3 + 4x_4 \leq 7$$

$$x_1, x_2, x_3, x_4 \geq 0$$

```
c = [5, 4, 3, 1];  
A = [  
    1, 1, 1, 1;  
    3, 2, 3, 1;  
    2, 1, 1, 4;  
    ];  
b = [  
    12;  
    5;  
    7;  
    ];  
[x, z] = simplex_max(c, A, b);  
fprintf("Solución:")
```

Solución:

```
for i=1:length(x)  
    fprintf("x%d = %.2f\n", i, x(i))
```

```
end
```

```
x1 = 0.00  
x2 = 2.50  
x3 = 0.00  
x4 = 0.00
```

```
fprintf("z = %.2f", z)
```

```
z = 10.00
```

El programa permite también detectar si una solución no está acotada, el siguiente problema de programación lineal no está acotado:

$$\max z = 2x_1 + x_2$$

$$3x_1 - x_2 \leq 6$$

$$-2x_1 + x_2 \leq 3$$

$$x_1, x_2 \geq 0$$

```
c = [2, 1];  
A = [  
    3, -1;  
   -2, 1  
];  
b = [  
    6;  
    3  
];  
[x, z] = simplex_max(c, A, b);
```

ERROR: Se ha encontrado una dirección utilizada para optimizar no acotada

```
fprintf("Solución trivial:")
```

Solución trivial:

```
for i=1:length(x)  
    fprintf("x%d = %.2f\n", i, x(i))  
end
```

```
x1 = 0.00
```

```
fprintf("z = %.2f", z)
```

```
z = 0.00
```

### **Pregunta 2:**

Apoyándote en el código anterior, crea un nuevo código del tipo function que te permita maximizar o minimizar un problema dado, cuyo output sea una solución óptima o el mensaje de que no está acotada. Para ello los inputs deberán ser:

- Un vector de caracteres para especificar si queremos maximizar o minimizar. (Use las cadenas max o min respectivamente como inputs y en caso de no usar ninguna de las anteriores devolver un mensaje con el error correspondiente).
- El vector de costes de la función objetivo, es decir, los coeficientes de las variables de la función objetivo.
- La matriz  $A$  de los coeficientes de las restricciones.
- El vector de recursos  $b$ .

Limpieza del entorno de trabajo

```
clear, clc
```

Reutilizando el problema de programación lineal acotado anterior:

$$\max z = 5x_1 + 4x_2 + 3x_3 + x_4$$

$$x_1 + x_2 + x_3 + x_4 \leq 12$$

$$3x_1 + 2x_2 + 3x_3 + x_4 \leq 5$$

$$2x_1 + x_2 + x_3 + 4x_4 \leq 7$$

$$x_1, x_2, x_3, x_4 \geq 0$$

```
c = [5, 4, 3, 1];
A = [
    1, 1, 1, 1;
    3, 2, 3, 1;
    2, 1, 1, 4;
];
b = [
    12;
    5;
    7;
];
[x, z] = simplex("max", c, A, b);
fprintf("Solución:")
```

Solución:

```
for i=1:length(x)
    fprintf("x%d = %.2f\n", i, x(i))
end
```

```
x1 = 0.00
x2 = 2.50
x3 = 0.00
x4 = 0.00
```

```
fprintf("z = %.2f", z)
```

```
z = 10.00
```

Probamos también minimización

$$\min z = 3x_1 - 2x_2$$

$$2x_1 + x_2 \leq 18$$

$$2x_1 + 3x_2 \leq 42$$

$$3x_1 - 2x_2 \leq 5$$

$$x_1, x_2 \geq 0$$

```
c = [3, -2];
A = [
    2, 1;
    2, 3;
    3, -2;
];
b = [
    18;
    42;
    5;
];
[x, z] = simplex("min", c, A, b);
fprintf("Solución:")
```

Solución:

```
for i=1:length(x)
    fprintf("x%d = %.2f\n", i, x(i))
end
```

```
x1 = 0.00
x2 = 14.00
```

```
fprintf("z = %.2f", z)
```

```
z = -28.00
```

En caso de opción incorrecta se muestra un mensaje de error

```
[x, z] = simplex("mean", c, A, b);
```

ERROR: Opción no reconocida

```
function [x, z] = simplex_max(c, A, b)
% Función que calcula la solución al problema de programación lineal:
%     max z = c'x
%     Ax <= b
%     x >= 0 para todo xi
%     b >= 0 para todo bi
% Mediante el algoritmo Simplex a partir de las tablas de Charnes, Cooper y
Henderson
% INPUTS:
```

```

%      A = matriz de coeficientes (matriz de restricciones)
%      b = vector columna de recursos (término independiente restricciones)
%      c = vector fila de costes (coeficientes función objetivo)
% OUTPUTS:
%      x = vector fila solución del problema de programación lineal
%      z = valor de la función objetivo para la solución obtenida

% Determinamos el tamaño de la matriz de restricciones
[n, m] = size(A);

% A la función objetivo se añaden variables de holgura con coeficientes igual a
cero
cj = horzcat(c, zeros(1, n));

% Guardamos los índices de las variables que pertenecen a la base
% Inicialmente las variables de holgura estarán en la base
base = m+1:m+n; % Tenemos m variables y a estas se añaden n de holgura

% Con las variables de holgura el sistema  $Ax \leq b$  pasa a ser  $Ax = b$ 
% Se procede a definir la tabla de Charnes, Cooper y Henderson
% Esta tabla contendrá en su primera columna los valores de las variables de la
base
aij = horzcat(b, A, eye(n));

% Definimos las últimas filas de la tabla que determinarán la columna pivote
% La primera posición de zj contendrá el valor de la función objetivo
% zj se obtiene como el producto escalar  $aj * cj'$ 
% Con  $cj'$  coeficientes de las variables en la base
zj = zeros(1, 1+m+n);
% NOTA: Los valores de las variables que no están en la base son cero
% Por eso hacemos  $aj * cj'$  en vez de todos los valores por sus respectivos costes

%  $cj - zj$  inicialmente siempre igual a c al ser zj vector de ceros inicialmente
cj_zj = cj;

% Solución trivial
x = 0;
z = 0;
% Mientras que todos los  $cj - zj$  sean mayores que cero se repite el algoritmo
while any(cj_zj > 0)
    % Obtención columna pivote k
    % Se busca el valor más grande positivo de  $cj - zj$ 
    [~, k] = max(cj_zj);
    k = k + 1; % MATLAB indexa a partir de 1

    % Obtención fila pivote h
    % Se busca el menor de dividir el valor de las variables que están en la
base
    % entre las coordenadas del vector entrará en la base
    a = aij(:, 1)./aij(:, k);

```

```

[~, h] = min(a(a>0));
% NOTA: Se contempla el caso de dividir por cero
% MATLAB devolverá infinito pero buscamos el mínimo valor positivo evitando
el error

% Entrará en la base la variable xk
% Saldrá la variable con posición h en la base
% NOTA: Para comprobar si la solución es acotada guardamos la variable que
sale
aux = base(h);
base(h) = k-1;
% Para obtener la siguiente tabla de Charnes, Cooper y Henderson se realiza
Gauss
for i=1:n
    for j = 1:1+m+n
        % Omitimos cálculos en la fila y columna pivote
        % Debido a que sabemos que solo hay que dividir y hacer
        % cero respectivamente
        if i==h || j==k
            continue
        end
        aij(i, j) = aij(i, j) - aij(h, j) * aij(i, k) / aij(h, k);
    end
end
% Finalmente se modifican la fila y columna pivote
% Se divide la fila pivote entre el pivote
aij(h, :) = aij(h, :) ./ aij(h, k);
% Se hacen cero los elementos en la columna pivote
aij(:, k) = 0;
% Excepto el pivote que vale 1
aij(h, k) = 1;

% Obtenemos cada zj haciendo el producto escalar aj*cj'
% Con cj' los coeficientes de las variables en la base
for j=1:1+m+n
    zj(j) = cj(base)*aij(:, j);
end

% Finalmente se obtiene cj-zj
% Que servirá para calcular el siguiente pivote o detener el algoritmo
cj_zj = cj-zj(2:end);
% NOTA: El primer valor de zj no interviene para elegir la columna pivote
% El primer valor de zj es el valor de la función objetivo para esa base

% Definimos también una tolerancia
% En caso de obtener valores muy pequeños los sustituimos por cero
cj_zj(abs(cj_zj)<=100*eps) = 0;

% Comprobamos que la solución está acotada

```

```

        % No puede ocurrir que todos los elementos de la columna del pivote
        anterior sean todos positivos
        % En ese caso la dirección no está acotada por las restricciones
        if all(aij(:, aux+1)>0)
            fprintf("ERROR: Se ha encontrado una dirección utilizada para optimizar
no acotada\n")
            return
        end
    end
end

% Se obtiene el vector solución de todas las variables (incluyendo holgura)
x = zeros(1, m+n);
for i=1:length(base)
    x(base(i)) = aij(i, 1);
end
% Finalmente descartamos los valores de las variables de holgura, para no
devolverlas
x = x(1:m);
% Devolvemos a su vez el valor de la función objetivo
z = zj(1);
end

function [x, z] = simplex(opt, c, A, b)
% Función que calcula la solución al problema de programación lineal:
%     opt z = c'x
%     Ax <= b
%     x >= 0 para todo xi
%     b >= 0 para todo bi
% Mediante el algoritmo Simplex a partir de las tablas de Charnes, Cooper y
Henderson
% INPUTS:
%     opt = opción a realizar (maximizar o minimizar)
%     A = matriz de coeficientes (matriz de restricciones)
%     b = vector columna de recursos (término independiente restricciones)
%     c = vector fila de costes (coeficientes función objetivo)
% OUTPUTS:
%     x = vector fila solución del problema de programación lineal
%     z = valor de la función objetivo para la solución obtenida
% NOTA: Se requiere la función simplex_max
    if opt == "max"
        [x, z] = simplex_max(c, A, b);
    elseif opt == "min"
        [x, z] = simplex_max(-c, A, b);
        z = -z;
    else
        fprintf("ERROR: Opción no reconocida\n")
        x = NaN;
        z = NaN;
    end
end
end

```