**DEPARTAMENTO DE INGENIERÍA INDUSTRIAL**
**Optimización Avanzada 202420 – Tarea 1**
PROFESOR: Andrés Medaglia
ASISTENTE: Laura Juliana Sánchez
MONITOR: Felipe de Jesús Liévano

| Apellidos | Nombres | Código | Login | Quién entrega (Bloque Neón) |
|---|---|---|---|---|
| Bayona Vergara | William Andres | 202011494 | w.bayona | X |
| Vasquez Cristancho | Juan Martin | 202113314 | j.vasquezc | |

**Problema 1: Migrando Facultades**

Para plantear la solución de este problema es posible pensarlo en principio a través de una formulación no lineal, como se presenta a continuación:

### 1.1. Formulación matemática (No Lineal):

**Conjuntos:**

$$F: Conjunto\ de\ Facultades$$
$$F: \{Ingeniería, Ciencias, Administración, Artes, Derecho\ y\ Economía\}$$

$$F: Conjunto\ de\ Sedes$$
$$S: \{Oriente, Norte, Sur, Occidente\}$$

**Parámetros:**

$$B_{fs}: Beneficio\ de\ tener\ la\ facultad\ f\ en\ la\ sede\ s$$
$$(en\ millones\ de\ pesos\ al\ año)$$

$$C_{fa}: Comunicaciones\ entre\ la\ facultad\ f\ y\ la\ facultad\ a$$
$$(en\ millones\ de\ unidades\ al\ año)$$

$$K_{se}: Costos\ de\ las\ comunicaciones\ entre\ la\ sede\ s\ y\ la\ sede\ e$$
$$(en\ millones\ de\ pesos\ colombianos)$$

**Variables de decisión:**

$$x_{fs}: \begin{cases} 1, si\ la\ facultad\ f\ \in F\ se\ encuentra\ ubicada\ en\ la\ sede\ s\ \in S \\ 0, dlc \end{cases}$$

**Función Objetivo:**

$$Max \sum_{f \in F} \sum_{s \in S} x_{fs} * B_{fs} - \sum_{f \in F} \sum_{a \in F} \sum_{s \in S} \sum_{e \in S} x_{fs} * x_{ae} * C_{fa} * K_{se}$$

**Restricciones:**

1. Todas las facultades se encuentran dentro de una sede
$$\sum_{s \in S} x_{fs} = 1\ \ \forall f \in F$$
2. Todas las sedes tienen al menos una facultad
$$\sum_{f \in F} x_{fs} \geq 1\ \ \forall s \in S$$
3. Todas las sedes tienen máximo tres facultades
$$\sum_{f \in F} x_{fs} \leq 3\ \ \forall s \in S$$
4. Naturaleza de las variables
$$x_{fs} \in \{0,1\}\ \forall s \in S, f \in F$$

En esta formulación, específicamente en la función objetivo, podemos observar la no-linealidad a través de la multiplicación entre variables $x_{fs}$ y $x_{ae}$. Con el fin de pasar este problema por un optimizador lineal y recibir una respuesta valida; se vuelve necesario transformar la formulación en una versión lineal. Esto lo realizaremos por medio de la creacion de una variable auxiliar $y_{fsae}$, cuyo valor depende de los valores que pueden adquirir las variables $x_{fs}$ y $x_{ae}$.
Definimos el valor que puede tener $y_{fsae}$ por medio de la siguiente tabla de verdad:

| $y_{fsae}$ ($x_{fs} \wedge x_{ae}$) | $x_{fs}$ | $x_{ae}$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |

Una vez tenemos definido los valores que puede adquirir $y_{fsae}$ basado en los valores que pueden adquirir $x_{fs}$ y $x_{ae}$ es posible incluir esta relacion por medio de 3 nuevas restricciones, las cuales presentamos dentro de la nueva formulación:

## 1.2. Formulación matemática (Lineal):

**Conjuntos:**

$$F: Conjunto\ de\ Facultades$$
$$F: \{Ingeniería, Ciencias, Administración, Artes, Derecho\ y\ Economía\}$$

$$F: Conjunto\ de\ Sedes$$
$$S: \{Oriente, Norte, Sur, Occidente\}$$

**Parámetros:**

$$B_{fs}: Beneficio\ de\ tener\ la\ facultad\ f\ en\ la\ sede\ s$$
$$(en\ millones\ de\ pesos\ al\ año)$$

$$C_{fa}: Comunicaciones\ entre\ la\ facultad\ f\ y\ la\ facultad\ a$$
$$(en\ millones\ de\ unidades\ al\ año)$$

$$K_{se}: Costos\ de\ las\ comunicaciones\ entre\ la\ sede\ s\ y\ la\ sede\ e$$
$$(en\ millones\ de\ pesos\ colombianos)$$

**Variables de decisión:**

$$x_{fs}: \begin{cases} 1, si\ la\ facultad\ f\ se\ encuentra\ ubicada\ en\ la\ sede\ s \\ 0, dlc \end{cases}$$

$$y_{fsae}: Variable\ auxiliar\ que\ representa\ el\ comportamiento\ de$$
$$x_{fs}\ \wedge\ x_{ae}\ donde\ f, a\ \in\ F;\ \ s, e\ \in\ S$$

**Función Objetivo:**

$$Max \sum_{f \in F} \sum_{s \in S} x_{fs} * B_{fs} - \sum_{f \in F} \sum_{a \in F} \sum_{s \in S} \sum_{e \in S} y_{fsae} * C_{fa} * K_{se}$$

**Restricciones:**

1. Todas las facultades se encuentran dentro de una sede

$$\sum_{s \in S} x_{fs} = 1\ \ \forall\ f\ \in F$$

2. Todas las sedes tienen al menos una facultad

$$\sum_{f \in F} x_{fs} \geq 1\ \ \forall\ s \in S$$

3. Todas las sedes tienen máximo tres facultades

$$\sum_{f \in F} x_{fs} \leq 3 \;\; \forall \, s \in S$$

4. La variable auxiliar $y_{fsae}$ siempre será menor o igual que $x_{fs}$

$$y_{fsae} \leq x_{fs} \;\; \forall \;\; f,a \in F; \;\; s,e \in S$$

5. La variable auxiliar $y_{fsae}$ siempre será menor o igual que $x_{ae}$

$$y_{fsae} \leq x_{ae} \;\; \forall \;\; f,a \in F; \;\; s,e \in S$$

6. Restricción que cubre el ultimo caso de la tabla de verdad

$$y_{fsae} \geq x_{fs} + x_{ae} - 1 \;\; \forall \;\; f,a \in F; \;\; s,e \in S$$

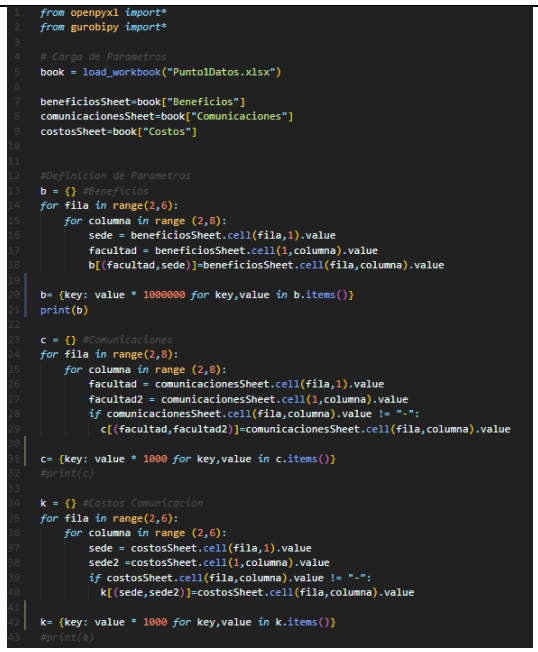
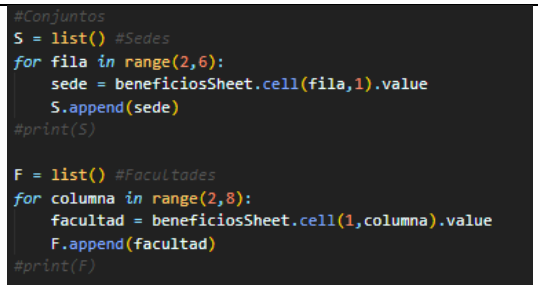
7. Naturaleza de las variables

$$x_{fs} \in \{0,1\} \, \forall \, s \in S, f \in F$$

$$y_{fsae} \geq 0 \, \forall \, s,e \in S; \; f,a \in F$$

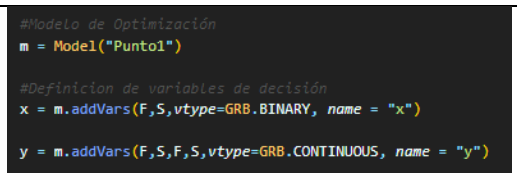
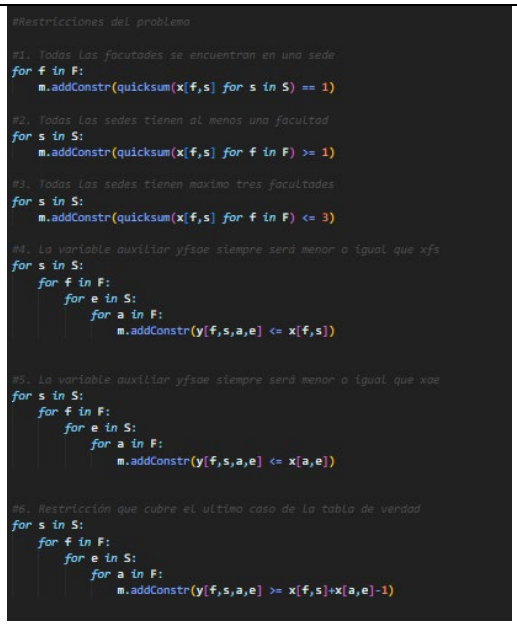
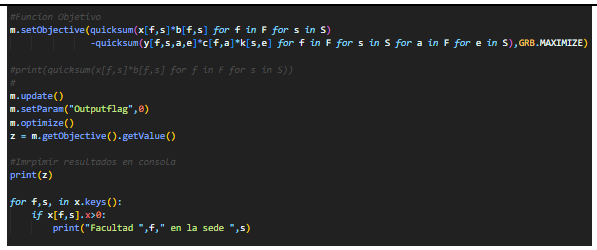
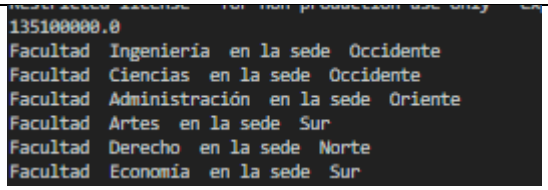
### 1.3. Implementación del modelo en Python-Gurobi

Dentro de la implementación se ejecutaron los siguientes pasos dentro del archivo Punto1.py

| | |
|---|---|
| Partimos con la carga de datos y la definición de los distintos diccionarios de parámetros |  |
| Definimos conjuntos de Sedes y de Facultades |  |

```
from openpyxl import*
from gurobipy import*

# Carga de Parametros
book = load_workbook("Punto1Datos.xlsx")

beneficiosSheet=book["Beneficios"]
comunicacionesSheet=book["Comunicaciones"]
costosSheet=book["Costos"]


#Definicion de Parametros
b = {} #Beneficios
for fila in range(2,6):
    for columna in range (2,8):
        sede = beneficiosSheet.cell(fila,1).value
        facultad = beneficiosSheet.cell(1,columna).value
        b[(facultad,sede)]=beneficiosSheet.cell(fila,columna).value

b= {key: value * 1000000 for key,value in b.items()}
print(b)

c = {} #Comunicaciones
for fila in range(2,8):
    for columna in range (2,8):
        facultad = comunicacionesSheet.cell(fila,1).value
        facultad2 = comunicacionesSheet.cell(1,columna).value
        if comunicacionesSheet.cell(fila,columna).value != "-":
          c[(facultad,facultad2)]=comunicacionesSheet.cell(fila,columna).value

c= {key: value * 1000 for key,value in c.items()}
#print(c)

k = {} #Costos Comunicacion
for fila in range(2,6):
    for columna in range (2,6):
        sede = costosSheet.cell(fila,1).value
        sede2 =costosSheet.cell(1,columna).value
        if costosSheet.cell(fila,columna).value != "-":
          k[(sede,sede2)]=costosSheet.cell(fila,columna).value

k= {key: value * 1000 for key,value in k.items()}
#print(k)
```

```
#Conjuntos
S = list() #Sedes
for fila in range(2,6):
    sede = beneficiosSheet.cell(fila,1).value
    S.append(sede)
#print(S)

F = list() #Facultades
for columna in range(2,8):
    facultad = beneficiosSheet.cell(1,columna).value
    F.append(facultad)
#print(F)
```

| | |
|---|---|
| Creamos el modelo de optimización y nuestras dos variables de interés |  |
| Definimos las restricciones del modelo |  |
| Definimos la función objetivo, ejecutamos el optimizador sobre el modelo creado e imprimimos resultados |  |
| Se ejecuta el problema y se obtiene la siguiente solución. En la primera línea se observa el valor de la funcion objetivo (La utilidad en el contexto del problema), siendo esta **$135,100,000** y las diferentes ubicaciones que tendrían las facultades en las sedes. |  |

```
#Modelo de Optimización
m = Model("Punto1")

#Definicion de variables de decisión
x = m.addVars(F,S,vtype=GRB.BINARY, name = "x")

y = m.addVars(F,S,F,S,vtype=GRB.CONTINUOUS, name = "y")
```

```
#Restricciones del problema

#1. Todas las facutades se encuentran en una sede
for f in F:
    m.addConstr(quicksum(x[f,s] for s in S) == 1)

#2. Todas las sedes tienen al menos una facultad
for s in S:
    m.addConstr(quicksum(x[f,s] for f in F) >= 1)

#3. Todas las sedes tienen maximo tres facultades
for s in S:
    m.addConstr(quicksum(x[f,s] for f in F) <= 3)

#4. La variable auxiliar yfsae siempre será menor o igual que xfs
for s in S:
    for f in F:
        for e in S:
            for a in F:
                m.addConstr(y[f,s,a,e] <= x[f,s])


#5. La variable auxiliar yfsae siempre será menor o igual que xae
for s in S:
    for f in F:
        for e in S:
            for a in F:
                m.addConstr(y[f,s,a,e] <= x[a,e])


#6. Restricción que cubre el ultimo caso de la tabla de verdad
for s in S:
    for f in F:
        for e in S:
            for a in F:
                m.addConstr(y[f,s,a,e] >= x[f,s]+x[a,e]-1)
```

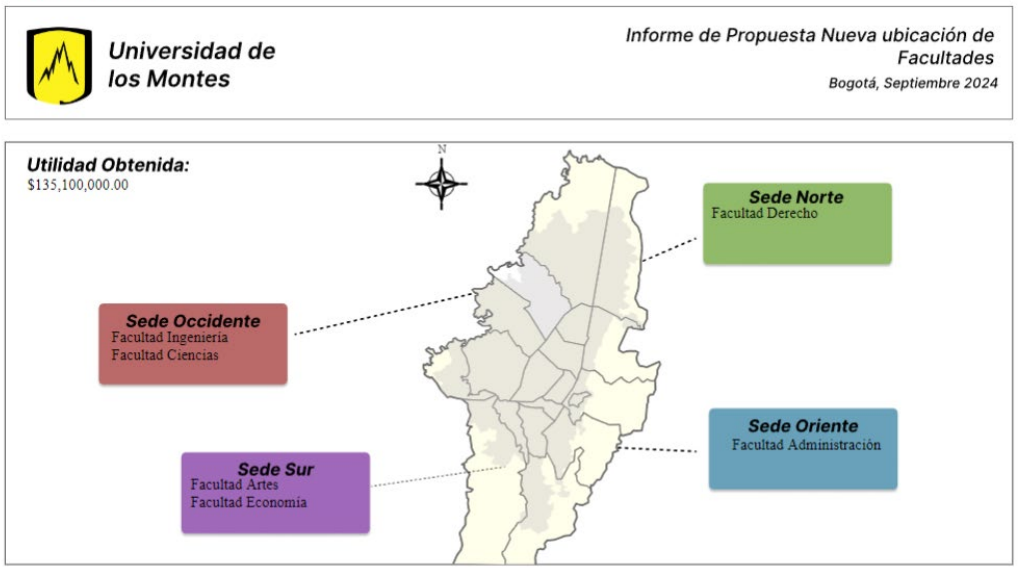
```
#Funcion Objetivo
m.setObjective(quicksum(x[f,s]*b[f,s] for f in F for s in S)
               -quicksum(y[f,s,a,e]*c[f,a]*k[s,e] for f in F for s in S for a in F for e in S),GRB.MAXIMIZE)

#print(quicksum(x[f,s]*b[f,s] for f in F for s in S))

m.update()
m.setParam("Outputflag",0)
m.optimize()
z = m.getObjective().getValue()

#Imprimir resultados en consola
print(z)

for f,s, in x.keys():
    if x[f,s].x>0:
        print("Facultad ",f," en la sede ",s)
```

```
135100000.0
Facultad  Ingeniería  en la sede  Occidente
Facultad  Ciencias  en la sede  Occidente
Facultad  Administración  en la sede  Oriente
Facultad  Artes  en la sede  Sur
Facultad  Derecho  en la sede  Norte
Facultad  Economía  en la sede  Sur
```

### 1.4. Presentación de visualización de la respuesta obtenida

Con el fin de presentar los resultados obtenidos al Consejo Superior de la Universidad de los Montes, se ha creado el archivo **Punto1Visualizador.py** el cual ejecuta el mismo modelo de optimización, con la diferencia de que se incluye una nueva parte en el código la cual genera un informe en pdf destinado al Consejo Superior.

El pdf contiene las ubicaciones de cada una de las facultades a través de las distintas sedes, así como la utilidad obtenida bajo esa combinación de ubicaciones.




Informe de Propuesta de traslado de Facultades

Si en algún momento se desea reevaluar el problema bajo nuevos parámetros, el archivo Punto1Visualizador.py esta en la capacidad de generar nuevos informes con los nuevos resultados obtenidos por el modelo de optimización.

Lo único que se debe realizar es cambiar los valores de los parámetros iniciales en el archivo Punto1Datos.xlsx y volver a correr el modelo.

**Observación:** En los anexos a este documento se encuentra el informe generado por el programa para los parámetros asignados bajo este contexto; sin embargo si desea realizar la prueba de funcionamiento por usted mismo, el único prerrequisito es la instalación de la librería **Aspose pdf**, la cual permite la edición de pdfs a través de Python. Se instala por medio del siguiente comando en consola: "pip install aspose-pdf"

## 1.5. Conclusión

El modelo de programación lineal planteado para la reubicación de las facultades de la Universidad de los Montes permitió **maximizar las utilidades anuales** asociadas a dicha reubicación, definidas por los beneficios potenciales y los costos de comunicación entre facultades. Se logró una maximización de la utilidad total anual de COP $135,100,000.

La solución obtenida, que asigna la Facultad de Ingeniería y la Facultad de Ciencias a la sede Occidente, la Facultad de Administración a la sede Oriente, la Facultad de Artes y la Facultad de Economía a la sede Sur, y la Facultad de Derecho a la sede Norte, resulta ser la **más eficiente en términos de utilidad neta** para la universidad.

Bajo este resultado se distribuyeron equitativamente las facultades entre las nuevas sedes y la actual sede oriental, **cumpliendo con las restricciones impuestas por la universidad**, de modo que no solo se permitió un uso más eficiente de los nuevos espacios disponibles, sino que también se contribuyó a fortalecer la posición de la universidad en términos de alcance y eficiencia operativa, al aprovechar al máximo los recursos y las infraestructuras disponibles.

**Problema 2: Sudoku Avanzado**

Para plantear la solución de este problema es posible pensarlo en principio a través de una formulación no lineal, como se presenta a continuación:

**2.1 Formulación matemática:**

**Conjuntos:**

$$K: Conjunto\ de\ numeros\ hasta\ N\ (tamanio\ tablero)$$
$$K: \{1,2,3 \dots N\}$$

$$R: Conjunto\ de\ regiones$$
$$S: \{1,2,3,4,5\}$$

$C1: Conjunto\ de\ las\ celdas\ en\ el\ tablero\ (la\ celda\ superior\ izquierda\ se\ cuenta\ como\ (1,1),\ las\ X\ aumetan\ a\ la\ derecha\ y\ las\ Y\ aumentan\ hacia\ abajo)$

$$C1: \{(1,1), (1,2), (1,3), (1,4) \dots (N, N)\}$$

**Parámetros:**

$$O_r: Operacion\ esperada\ en\ la\ region\ r\ \in\ R$$
$$O_r: \{1: Suma, 2: Resta: \dots\}$$

$$E_r: Resultado\ esperado\ en\ la\ region\ r\ \in\ R$$
$$E_r: \{1: 2,3: 4 \dots\}$$

$$C_r: Celdas\ en\ la\ region\ r \in R$$
$$C_r: \{1: [(1,1), (2,2)], 2: [(4,2), (3,1)], \dots\}$$

**Variables de Decisión:**

$$x_{k(i,j)}: \begin{cases} 1, si\ el\ numero\ k \in K\ se\ encuentra\ en\ la\ posicion\ (i,j) \in C1 \\ 0, dlc \end{cases}$$

$$Y_{(i,j)}: corresponde\ al\ numero\ k \in K\ ubicado\ en\ la\ posicion (i,j) \in C1$$

$$Z1_r: \begin{cases} 1, si\ el\ y_{(i,j)} \leq y_{(m,l)}\ en\ las\ restricciones\ de\ resta\ (se\ delcara\ para\ todas\ las\ r \in R) \\ 0, dlc \end{cases}$$

$$Z2_r: \begin{cases} 1, si\ el\ y_{(i,j)} \leq y_{(m,l)}\ en\ las\ restricciones\ de\ division\ (se\ delcara\ para\ todas\ las\ r \in R) \\ 0, dlc \end{cases}$$

**Función Objetivo**

$$\max \sum_{(i,j)\in C1} y_{(i,j)}$$

En si no interesa la función objetivo, sino que se cumplan todas las restricciones. Esta función debería ser igual a 126 dadas las características del sudoku (para una instancia 6x6).

**Restricciones**

1. Solo un numero por casilla

$$\sum_{k\in K} x_{k,(i,j)} = 1 \ \forall\, (i,j) \in C1$$

2. No repetir números en filas

$$\sum_{i\in N} x_{k,(i,j)} = 1 \ \forall\, k,j \in K$$

3. No repetir números en columnas

$$\sum_{j\in N} x_{k,(i,j)} = 1 \quad \forall\, k,i \in K$$

4. Restricción para igualar las X y las Y

$$y_{(i,j)} = \sum_{k\in K} k * \ x_{k,(i,j)} \quad \forall\, (k,i) \in C1$$

5. Restricción para la suma

$$\sum_{(i,j)\in C_r} y_{(i,j)} = E_r \quad \forall\, r \in R \mid O_r = Suma$$

6. Restricción para la resta
   6.1 no lineal:

$$\left|y_{(i,j)} - y_{(m,l)}\right| = E_r$$

$$\forall\, r \in R \ \wedge \ \forall\, (i,j),(m,l) \in C_r \mid O_r = Resta \ \wedge ((m = i+1 \wedge l = j) \vee (m = 1 \wedge l = j+1))$$

   6.2 lineal:

Para linealizar esta restricción se utiliza una de las variables auxiliares (z1) esta variable capturara si $y_{(i,j)} \geq y_{(m,l)}$ se sabe que si inecuación es cierta $E_r$ debería ser positivo de lo contrario negativo, este comportamiento se modela de la restricción de la siguiente manera:

$$y_{(i,j)} - y_{(m,l)} = E_r - 2E_r * Z1_r$$

$$\forall\, r \in R \ \wedge\ \forall\, (i,j),(m,l) \in C_r |\ O_r = Resta\ \wedge ((m = i+1 \wedge l = j) \vee (m = 1 \wedge l = j+1))$$

en este caso donde la variable extra se active el valor de a la derecha de la restricción será $-E_r$. Además, como se tiene un segundo recorrido sobre $C_r$ se debe únicamente tomar las combinaciones de m,l,i,j que interesan, si están en columna o en fila.

7. Restricción para la división:
   7.1 no lineal:

$$\frac{y_{(m,l)}}{y_{(ij)}} = E_r$$

$$\forall\, r \in R \ \wedge\ \forall\, (i,j),(m,l) \in C_r |\ O_r = Division\ \wedge ((m = i+1 \wedge l = j) \vee (m = 1 \wedge l = j+1))$$

   7.2 lineal:

La anterior restricción no logra capturar la escénica de la división en el sudoku además se debe tener en cuenta el caso en el cual $y_{(i,j)} > y_{(m,l)}$ para poder linealizar en primer lugar se despeja el numerador en ambos casos:

$$y_{(m,l)} = y_{(i,j)*E_r}$$

$$y_{(i,j)} = y_{(m,l)*E_r}$$

Sin embargo, ambas restricciones no podrían cumplirse al tiempo, por lo tanto, se plantean como inecuaciones y se hace una relajación basada en la variable binaria z2 que se usa para modelar el caso en que $y_{(i,j)} > y_{(m,l)}$ y que las otras restricciones no afecten el desarrollo del problema. Quedando así el grupo de restricciones:

$$y_{(i,j)} \geq y_{(m,l)*E_r-1} - 10^6 Z2_r$$
$$y_{(i,j)} \leq y_{(m,l)*E_r-1} + 10^6 Z2_r$$

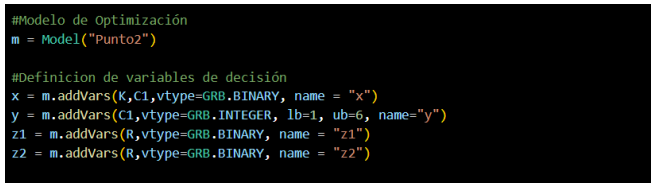
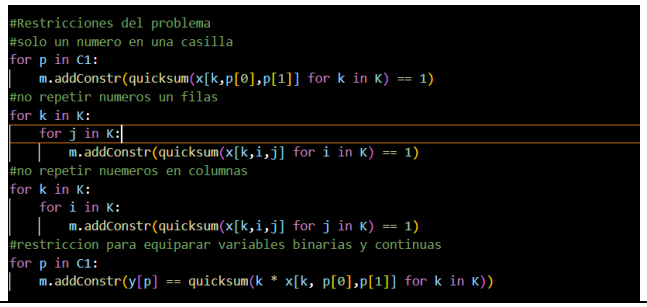
$$y_{(m,l)} \geq y_{(i,j)*E_r-1} - 10^6 Z2_r$$
$$y_{(m,l)} \leq y_{(i,j)*E_r-1} + 10^6 Z2_r$$

$$\forall\, r \in R \ \wedge\ \forall\, (i,j),(m,l) \in C_r |\ O_r = Division\ \wedge ((m = i+1 \wedge l = j) \vee (m = 1 \wedge l = j+1))$$

**2.2 implementación en Gurobi Python y respuesta de instancias:**

Implementación en el archivo Punto2.1.py .

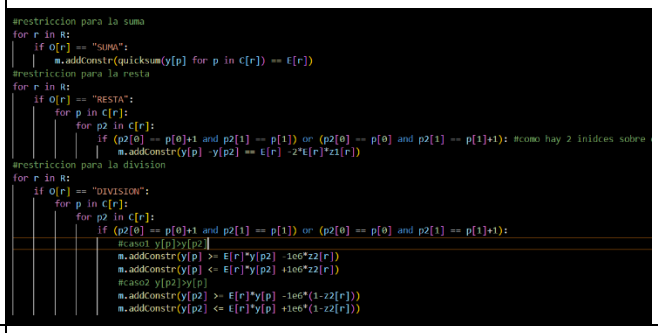
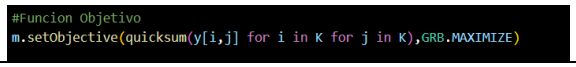
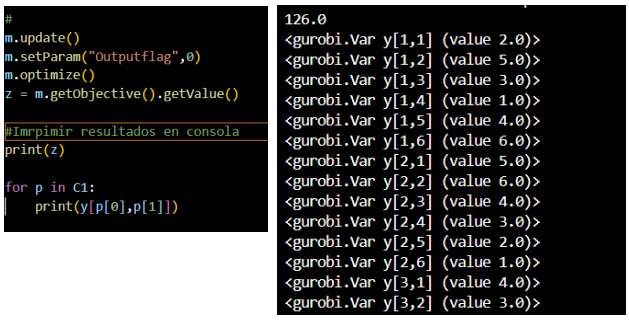
**2.2.1. Instancia A:**

| | |
|---|---|
| En primer lugar, se plantea el problema con sus respectivos conjuntos y parámetros **(instancia A):** |  |
| En siguiente lugar se crea la instancia del optimizador y se declaran las variables: |  |
| Se plantean las restricciones básicas de un sudoku: |  |

```
from openpyxl import*
from gurobipy import*

#conjuntos
K= [i for i in range(1,7)] #numeros
R = [i for i in range(1,17)] #regiones
C1 = [(i,j)for i in K for j in K] #celdas


#parametros
#OPERACIONES
O={1:"SUMA",2:"SUMA",3:"SUMA",4:"SUMA",5:"SUMA",6:"SUMA",7:"SUMA",8:"SUMA",
  9:"SUMA",10:"SUMA",11:"SUMA",12:"SUMA",13:"SUMA",14:"SUMA",15:"SUMA",16:"SUMA"}
#RESULTADOS_ESPERADOS
E={1:7,2:9,3:9,4:6,5:5,6:10,7:4,8:6,9:7,10:8,11:10,12:5,13:10,14:3,15:11,16:16}

#CELDAS POR REGION
C={1:[(1,1),(1,2)], 2:[(2,1),(3,1)], 3:[(4,1),(5,1)], 4:[(6,1),(6,2),(5,2)],5:[(4,2),(3,2)],
  6:[(2,2),(2,3)],7:[(1,3),(1,4)],8:[(3,3),(3,4)],9:[(5,3),(4,3)],10:[(6,3),(6,4)],
  11:[(4,4),(5,4)],12:[(2,5),(2,4)],13:[(1,6),(1,5)],14:[(2,6),(3,6)],15:[(3,5),(4,5),(4,6)],
  16:[(5,5),(5,6),(6,6),(6,5)]}
```

```
#Modelo de Optimización
m = Model("Punto2")

#Definicion de variables de decisión
x = m.addVars(K,C1,vtype=GRB.BINARY, name = "x")
y = m.addVars(C1,vtype=GRB.INTEGER, lb=1, ub=6, name="y")
z1 = m.addVars(R,vtype=GRB.BINARY, name = "z1")
z2 = m.addVars(R,vtype=GRB.BINARY, name = "z2")
```

```
#Restricciones del problema
#solo un numero en una casilla
for p in C1:
    m.addConstr(quicksum(x[k,p[0],p[1]] for k in K) == 1)
#no repetir numeros un filas
for k in K:
    for j in K:
        m.addConstr(quicksum(x[k,i,j] for i in K) == 1)
#no repetir nuemeros en columnas
for k in K:
    for i in K:
        m.addConstr(quicksum(x[k,i,j] for j in K) == 1)
#restriccion para equiparar variables binarias y continuas
for p in C1:
    m.addConstr(y[p] == quicksum(k * x[k, p[0],p[1]] for k in K))
```

| | |
|---|---|
| Se plantean las restricciones de operaciones |  |
| Se plantea la función objetivo |  |
| Se imprimen los resultados |  |

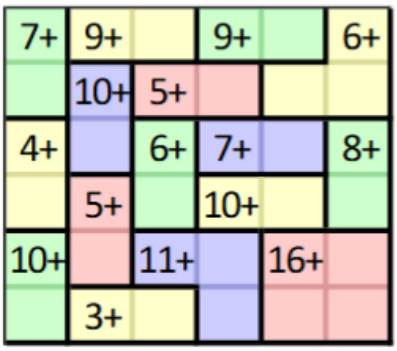
```
#restriccion para la suma
for r in R:
    if O[r] == "SUMA":
        m.addConstr(quicksum(y[p] for p in C[r]) == E[r])
#restriccion para la resta
for r in R:
    if O[r] == "RESTA":
        for p in C[r]:
            for p2 in C[r]:
                if (p2[0] == p[0]+1 and p2[1] == p[1]) or (p2[0] == p[0] and p2[1] == p[1]+1): #como hay 2 inidces sobre
                    m.addConstr(y[p] -y[p2] == E[r] -2*E[r]*z1[r])
#restriccion para la division
for r in R:
    if O[r] == "DIVISION":
        for p in C[r]:
            for p2 in C[r]:
                if (p2[0] == p[0]+1 and p2[1] == p[1]) or (p2[0] == p[0] and p2[1] == p[1]+1):
                    #caso1 y[p]>y[p2]
                    m.addConstr(y[p] >= E[r]*y[p2] -1e6*z2[r])
                    m.addConstr(y[p] <= E[r]*y[p2] +1e6*z2[r])
                    #caso2 y[p2]>y[p]
                    m.addConstr(y[p2] >= E[r]*y[p] -1e6*(1-z2[r]))
                    m.addConstr(y[p2] <= E[r]*y[p] +1e6*(1-z2[r]))
```

```
#Funcion Objetivo
m.setObjective(quicksum(y[i,j] for i in K for j in K),GRB.MAXIMIZE)
```

```
#
m.update()
m.setParam("Outputflag",0)
m.optimize()
z = m.getObjective().getValue()

#Imrpimir resultados en consola
print(z)

for p in C1:
    print(y[p[0],p[1]])
```

```
126.0
<gurobi.Var y[1,1] (value 2.0)>
<gurobi.Var y[1,2] (value 5.0)>
<gurobi.Var y[1,3] (value 3.0)>
<gurobi.Var y[1,4] (value 1.0)>
<gurobi.Var y[1,5] (value 4.0)>
<gurobi.Var y[1,6] (value 6.0)>
<gurobi.Var y[2,1] (value 5.0)>
<gurobi.Var y[2,2] (value 6.0)>
<gurobi.Var y[2,3] (value 4.0)>
<gurobi.Var y[2,4] (value 3.0)>
<gurobi.Var y[2,5] (value 2.0)>
<gurobi.Var y[2,6] (value 1.0)>
<gurobi.Var y[3,1] (value 4.0)>
<gurobi.Var y[3,2] (value 3.0)>
```

De esta manera se puede ver que una posible respuesta a la instancia A es:




| | | | | | |
|---|---|---|---|---|---|
| 2 | 5 | 4 | 3 | 6 | 1 |
| 5 | 6 | 3 | 2 | 1 | 4 |
| 3 | 4 | 1 | 5 | 2 | 6 |
| 1 | 3 | 5 | 6 | 4 | 2 |
| 4 | 2 | 6 | 1 | 3 | 5 |
| 6 | 1 | 2 | 4 | 5 | 3 |

Resta

Suma

Division

**2.2.2. Instancia B:**

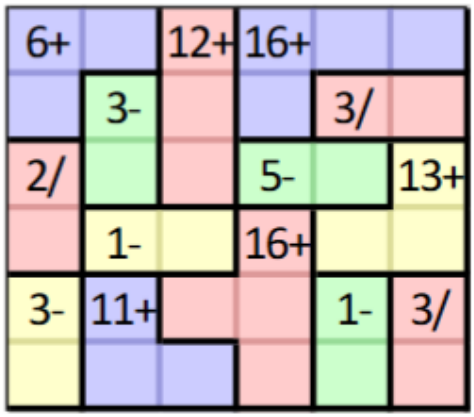
El procedimiento para realizar en la instancia B es el mismo, siendo la única diferencia los datos ingresados como parámetros. Implementación en el archivo Punto2.2.py .

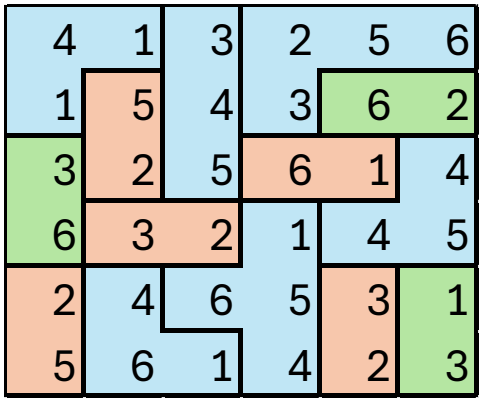
| | |
|---|---|
| Se utilizan los siguientes parámetros y conjuntos sobre el planteamiento anterior | |
| Llegando a los siguientes resultados: | |

```
#parametros
#OPERACIONES
O={1:"SUMA",2:"SUMA",3:"SUMA",4:"DIVISION",5:"RESTA",6:"RESTA",7:"DIVISION",8:"RESTA",
  9:"SUMA",10:"DIVISION",11:"RESTA",12:"SUMA",13:"SUMA",14:"RESTA"}
#RESULTADOS_ESPERADOS
E={1:6,2:12,3:16,4:3,5:5,6:3,7:2,8:1,9:13,10:3,11:1,12:16,13:11,14:3}

#CELDAS POR REGION
C={1:[(1,1),(1,2),(2,1)],2:[(3,1),(3,2),(3,3)],3:[(4,2),(4,1),(5,1),(6,1)],
  4:[(5,2),(6,2)],5:[(5,3),(4,3)],6:[(2,2),(2,3)],7:[(1,3),(1,4)],8:[(2,4),(3,4)],
  9:[(6,3),(6,4),(5,4)],10:[(6,6),(6,5)],11:[(5,6),(5,5)],12:[(4,6),(4,5),(4,4),(3,5)],
  13:[(2,6),(3,6),(2,5)],14:[(1,6),(1,5)]}
```

```
126.0
<gurobi.Var y[1,1] (value 4.0)>
<gurobi.Var y[1,2] (value 1.0)>
<gurobi.Var y[1,3] (value 3.0)>
<gurobi.Var y[1,4] (value 6.0)>
<gurobi.Var y[1,5] (value 2.0)>
<gurobi.Var y[1,6] (value 5.0)>
<gurobi.Var y[2,1] (value 1.0)>
<gurobi.Var y[2,2] (value 5.0)>
<gurobi.Var y[2,3] (value 2.0)>
<gurobi.Var y[2,4] (value 3.0)>
<gurobi.Var y[2,5] (value 4.0)>
```

De esta manera se puede ver que una posible respuesta a la instancia B es:





**Conclusión**

Luego de lograr plantear un programa líneal e implementarlo para la resolución del sudoku avanzado se puede concluir que se ha desarrollado un método que r**esuelve instancias cualesquiera de tamaño variable** y estos resultados son acertados, respetando las restricciones propias del sudoku y las restricciones de áreas y operaciones del sudoku avanzado.

Un resultado interesante al que se llegó es la **nula importancia de la función objetivo**, en este problema mientras la solución cumpla las restricciones la respuesta será válida para lo que se requiere, podría ser de maximización o minimización con cualesquiera variables y no importaría en la solución. Se hayo un óptimo para el problema.