😎 Método gráfico de la programación lineal

Se puede abordar el método gráfico como alternativa de solución de modelos de programación lineal, como un aporte a la investigación de operaciones, al tiempo que se genera código en Python que nos permita automatizar procedimientos gráficos y cálculos.

Más información (https://economipedia.com/definiciones/programacion-lineal.html)

El problema

Un micro que realiza un cierto recorrido, ofrece asientos para fumadores al precio de 10.000 pesos y a no fumadores al precio 6.000 pesos. Al no fumador se le deja llevar 50 Kg. de peso y al fumador 20 Kg. Si el autobús tiene 90 asientos y admite un equipaje de hasta 3.000 Kg. ¿Cuál ha de ser la oferta de asientos de la compañía para cada tipo de pasajeros, con la finalidad de optimizar el beneficio?. Tener en cuenta que por políticas de la empresa, deben ofrecerse cómo mínimo 10 asientos para pasajeros no fumadores.

Modelamiento mediante programación lineal

Variables

x: Cantidad de asientos reservados a fumadores.

y: Cantidad de asientos reservados a no fumadores.

Restricciones

20x + 50y <= 3000 (Equipaje permitido)

x + y <= 90 (Cantidad de asientos disponibles)

y >= 10 (Política de asientos mínimos para no fumadores)

y >= 0 (No negatividad)

x >= 0 (No negatividad)

Función objetivo

z = 10000x + 6000y (Maximizar)

Paso 1: Importar las librerías

In [34]:
```
import matplotlib.pyplot as plt
import numpy as np
from shapely.geometry import LineString
```

Paso 2: Ecuaciones e intervalos

Para poder graficar las restricciones (paso esencial del método gráfico), es necesario representar las ecuaciones por medio de líneas en nuestra gráfica.

Cada restricción estará representada por una línea recta. El paso que antecede trazar una línea recta consiste en la determinación de un mínimo de dos puntos que al unirse la conformen.

Cada punto en el plano cartesiano se encuentra conformado por una coordenada en x y una coordenada en y (en el caso que así definamos llamar a la abscisa y a la ordenada).

Recordemos que a partir del modelo algebraico inicial, nuestras restricciones se encuentran representadas por ecuaciones. Así entonces, la tarea consiste en, a partir de una ecuación, obtener un conjunto mínimo de dos coordenadas.

Lo primero que debemos hacer es despejar cada una de las inecuaciones convertidas en ecuaciones del problema, para eso, en nuestro caso, vamos a despejar en función de y:

Primera restricción: (Equipaje permitido)

20x + 50y <= 3000 (Inecuación)

20x + 50y = 3000 (Ecuación)

y = (3000 – 20x) / 50 (Despejamos y)

y1 = (3000 – 20x) / 50 (Asignamos un identificador único a y)

Segunda restricción: (Cantidad de asientos disponibles)

x + y <= 90 (Inecuación)

x + y = 90 (Ecuación)

y = 90 – x (Despejamos y)

y2 = 90 – x (Asignamos un identificador único a y)

Tercera restricción: (Política de asientos mínimos para no fumadores)

y >= 10 (Inecuación)

y = 10 (Ecuación)

Dado que la intención es graficar, es fundamental que cada ecuación contenga las dos variables. Dado que en la ecuación original la variable x no hace parte, podemos incluirla multiplicándola por 0. Es decir, para todos los valores que tome x la ecuación permanecerá inalterable.

y = 10 + (0 * x) (Ecuación, y ya se encuentra despejada)

y3 = 10 + (0 * x) (Asignamos un identificador único a y)

Cuarta restricción: (No negatividad)

y >= 0 (Inecuación)

y = 0 (Ecuación)

En este caso, dado que la intención es graficar, es fundamental que cada ecuación contenga las dos variables. Dado que en la ecuación original la variable x no hace parte, podemos incluirla multiplicándola por 0. Es decir, para todos los valores que tome x la ecuación permanecerá inalterable.

y = 0 * x (Ecuación, y ya se encuentra despejada)

y4 = 0 * x (Asignamos un identificador único a y)

Todas las ecuaciones anteriores se encuentran despejadas para obtener el valor de y a partir de x, de manera que necesitamos alguna función que nos permita asignar diferentes valores a x en un rango dado:

In [2]:
```
x = np.arange(-100, 200, 50)
x
```

Out[2]:
```
array([-100,  -50,    0,   50,  100,  150])
```

Cuarta restricción: (No negatividad)

Está dada en función de hallar x a partir y, de manera que efectuamos la misma tarea en un sentido inverso:

x >= 0 (Inecuación)

x = 0 (Ecuación)

En este caso, dado que la intención es graficar, es fundamental que cada ecuación contenga las dos variables. Dado que en la ecuación original la variable y no hace parte, podemos incluirla multiplicándola por 0. Es decir, para todos los valores que tome y la ecuación permanecerá inalterable.

x = 0 * y (Ecuación, x ya se encuentra despejada)

x1 = 0 * y (Asignamos un identificador único a x)

En este caso, requerimos que y tome diversos valores:

In [3]:
```
y = np.arange(-100, 200, 50)
y
```

Out[3]:
```
array([-100,  -50,    0,   50,  100,  150])
```

In [4]:
```
y1 = (3000 - (20 * x))/ 50
y2 = 90 - x
y3 = 10 + (0 * x)
y4 = 0 * x
x1 = 0 * y

y1, y2, y3, y4, x1
```

Out[4]:
```
(array([100.,  80.,  60.,  40.,  20.,   0.]),
 array([190, 140,  90,  40, -10, -60]),
 array([10, 10, 10, 10, 10, 10]),
 array([0, 0, 0, 0, 0, 0]),
 array([0, 0, 0, 0, 0, 0]))
```

Función objetivo:

10000x + 6000y = 0 (Ecuación)

y = (- 10000x) / 6000 (Despejamos y)

z = (- 10000x) / 6000 (Asignamos un identificador único a y)

En este caso, la función que ya asignamos a x (np.arange), nos prestará los valores necesarios para tabular z

In [5]:
```
z = (-10000 * x) / 6000
z
```

Out[5]:
```
array([ 166.66666667,   83.33333333,    0.        ,  -83.33333333,
       -166.66666667, -250.        ])
```

Paso 3: Tabular coordenadas e identificar las líneas

En el paso anterior desarrollamos unas líneas de código que representan los cálculos correspondientes a cada una de las ecuaciones del modelo. Si bien con fines prácticos mostramos la forma en que se tabularían los datos, es en este paso en que la información se organiza en tablas (propiamente matrices de dos columnas: x y y):

In [6]:
```
np.column_stack((x, y1))
```

Out[6]:
```
array([[-100.,  100.],
       [ -50.,   80.],
       [   0.,   60.],
       [  50.,   40.],
       [ 100.,   20.],
       [ 150.,    0.]])
```

Identificadores para las líneas: El siguiente paso consiste en generar la línea correspondiente a cada ecuación de acuerdo al tabulado.

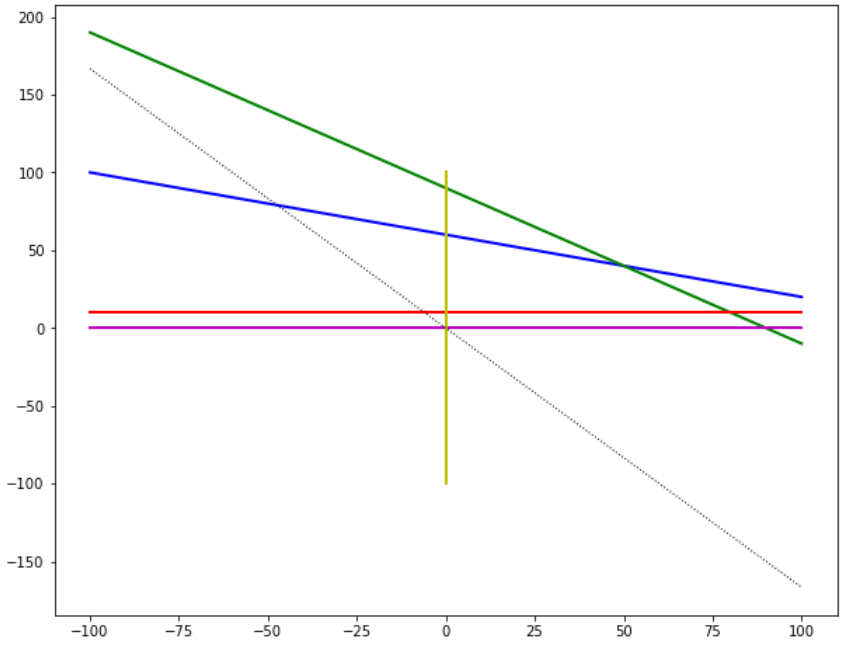
LineString es una instancia de la librería Shapely que permite unir cada punto (coordenada), de manera que genera una línea. Ahora bien, cada línea de código deberá corresponder o asociarse a una variable única que nos permitirá identificar cada una de las líneas:

In [7]:
```
primera_linea = LineString(np.column_stack((x, y1)))
segunda_linea = LineString(np.column_stack((x, y2)))
tercera_linea = LineString(np.column_stack((x, y3)))
quinta_linea = LineString(np.column_stack((x, y4)))
cuarta_linea = LineString(np.column_stack((x1, y)))
sexta_linea = LineString(np.column_stack((x, z)))
```

Paso 4: Graficar las líneas

In [35]:
```
plt.figure(figsize=(10,8))
plt.plot(x, y1, '-', linewidth=2, color='b')
plt.plot(x, y2, '-', linewidth=2, color='g')
plt.plot(x, y3, '-', linewidth=2, color='r')
plt.plot(x, y4, '-', linewidth=2, color='m')
plt.plot(x1, y, '-', linewidth=2, color='y')
plt.plot(x, z, ':', linewidth=1, color='k')
```

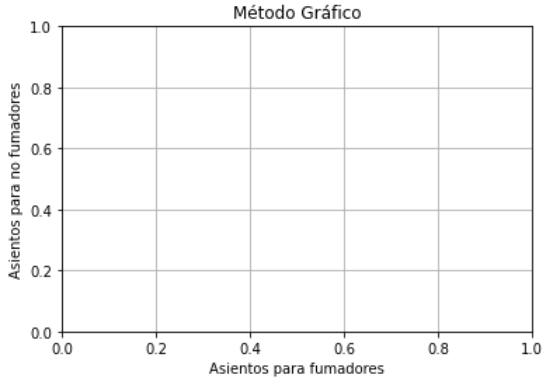
Out[35]:
```
[<matplotlib.lines.Line2D at 0x28591edb760>]
```



Configurar el gráfico

In [37]:

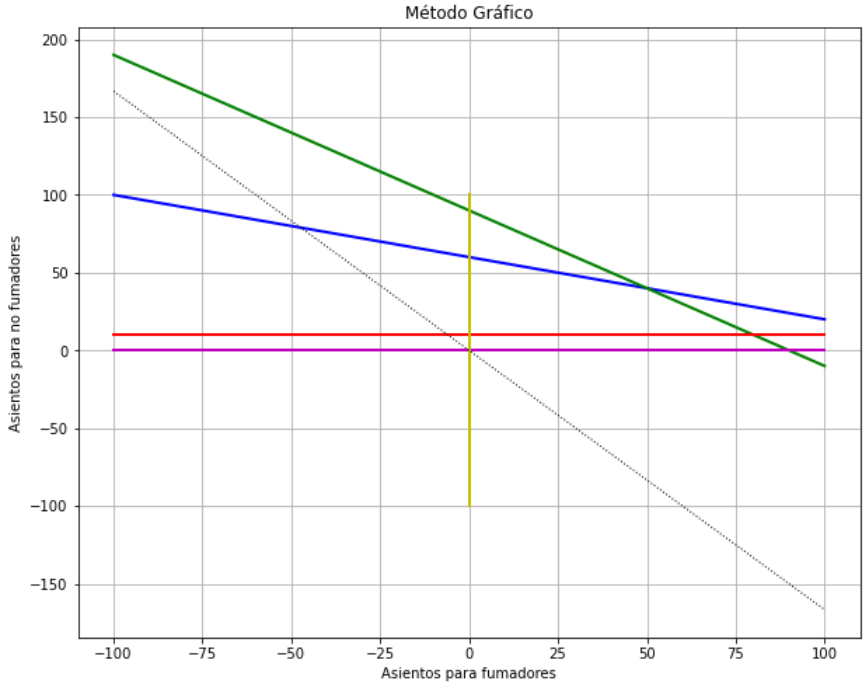
```
plt.grid()
plt.xlabel('Asientos para fumadores')
plt.ylabel('Asientos para no fumadores')
plt.title('Método Gráfico')

plt.show()
```



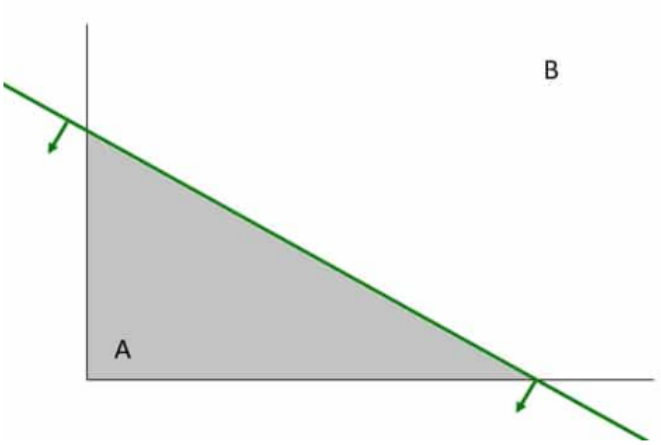
Hasta aquí todo el programa:

In [38]:

```
import matplotlib.pyplot as plt
import numpy as np
import shapely
from shapely.geometry import LineString

x = np.arange(-100, 150, 50)
y = np.arange(-100, 150, 50)

y1 = (3000 - (20 * x))/ 50
y2 = 90 - x
y3 = 10 + (0 * x)
x1 = 0 * y
y4 = 0 * x
z = (-10000 * x) / 6000

primera_linea = LineString(np.column_stack((x, y1)))
segunda_linea = LineString(np.column_stack((x, y2)))
tercera_linea = LineString(np.column_stack((x, y3)))
cuarta_linea = LineString(np.column_stack((x1, y)))
quinta_linea = LineString(np.column_stack((x, y4)))
sexta_linea = LineString(np.column_stack((x, z)))

plt.figure(figsize=(10,8))
plt.plot(x, y1, '-', linewidth=2, color='b')
plt.plot(x, y2, '-', linewidth=2, color='g')
plt.plot(x, y3, '-', linewidth=2, color='r')
plt.plot(x, y4, '-', linewidth=2, color='m')
plt.plot(x1, y, '-', linewidth=2, color='y')
plt.plot(x, z, ':', linewidth=1, color='k')

plt.grid()
plt.xlabel('Asientos para fumadores')
plt.ylabel('Asientos para no fumadores')
plt.title('Método Gráfico')

plt.show()
```



Podemos identificar que en la gráfica se pueden apreciar algunos polígonos, pero: ¿Cuál es nuestro polígono factible?. Para eso debemos considerar la naturaleza de las restricciones; en cuyo caso, cada línea trazada dividirá el cuadrante solución en dos semiplanos: uno a cada lado de la línea trazada. Solo una de estas dos mitades satisface cada desigualdad, de manera que debemos considerar si cada restricción es «menor o igual», «mayor o igual», o en su defecto «igual».



en esta figura, podemos observar el concepto de semiplanos: A y B (Uno a cada lado de la línea trazada). Como se trata de una restricción «menor o igual» (líneas hacia abajo y hacia la izquierda), solo la mitad A satisface la desigualdad. Las restricciones «menores o igual» apuntarán sus flechas hacia abajo y/o a la izquierda; los «mayores o igual» apuntarán sus flechas hacia arriba y/o a la derecha. Así entonces encontramos con suma facilidad nuestro polígono factible.

Paso 5: Determinar y graficar las intersecciones

En este ejemplo el criterio es maximizar. Recordemos que en programación lineal, la solución óptima siempre está asociada con un punto de esquina o vértice del polígono factible. Y que estos vértices se generan en cada punto de intersección de dos o más líneas, es decir, en el punto en que se igualan dos o más ecuaciones.

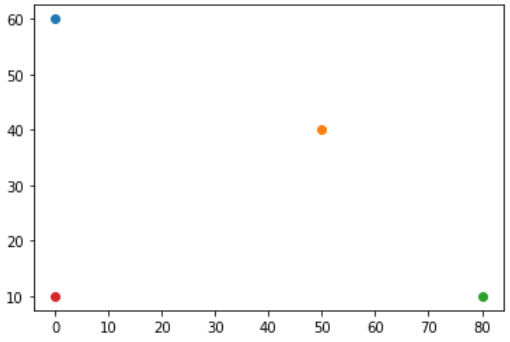
Determinar el punto de intersección de dos líneas requiere de la solución de un sistema de ecuaciones de 2 x 2, ya sea por el método de sustitución, igualación, eliminación, etc, aunque utilizaremos la función intersection() de Python:

In [11]:
```
primera_interseccion = cuarta_linea.intersection(primera_linea)
segunda_interseccion = primera_linea.intersection(segunda_linea)
tercera_interseccion = segunda_linea.intersection(tercera_linea)
cuarta_interseccion = tercera_linea.intersection(cuarta_linea)
```

El paso siguiente consiste en graficar dichas restricciones:

In [12]:
```
plt.plot(*primera_interseccion.xy, 'o')
plt.plot(*segunda_interseccion.xy, 'o')
plt.plot(*tercera_interseccion.xy, 'o')
plt.plot(*cuarta_interseccion.xy, 'o')
```

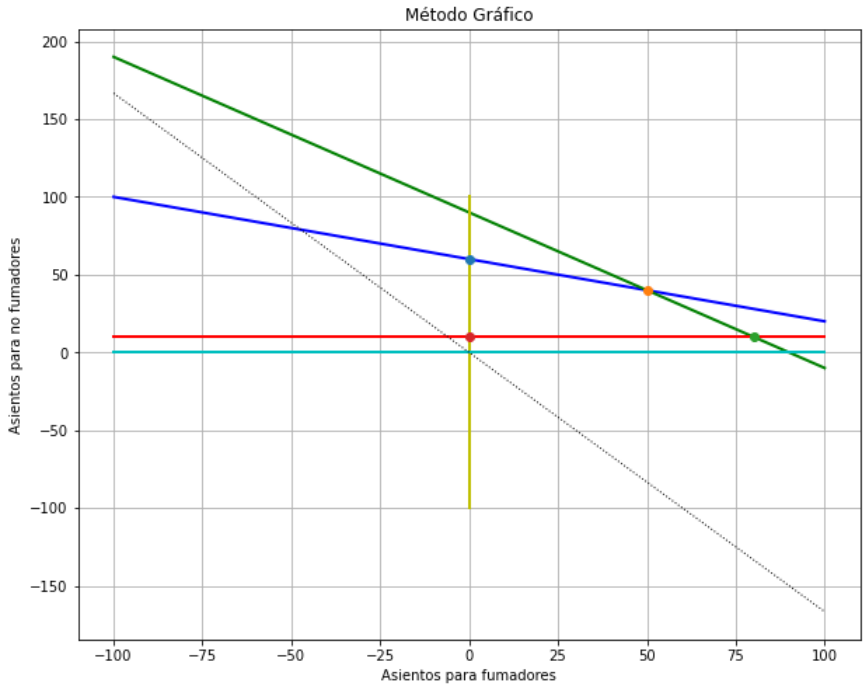
Out[12]:
```
[<matplotlib.lines.Line2D at 0x28590cf4b50>]
```



Hasta aquí todo el programa:

In [39]:

```
import matplotlib.pyplot as plt
import numpy as np
import shapely
from shapely.geometry import LineString

x = np.arange(-100, 150, 50)
y = np.arange(-100, 150, 50)

y1 = (3000 - (20 * x))/ 50
y2 = 90 - x
y3 = 10 + (0 * x)
x1 = 0 * y
y4 = 0 * x
z = (-10000 * x) / 6000

primera_linea = LineString(np.column_stack((x, y1)))
segunda_linea = LineString(np.column_stack((x, y2)))
tercera_linea = LineString(np.column_stack((x, y3)))
cuarta_linea = LineString(np.column_stack((x1, y)))
quinta_linea = LineString(np.column_stack((x, y4)))
sexta_linea = LineString(np.column_stack((x, z)))

plt.figure(figsize=(10,8))
plt.plot(x, y1, '-', linewidth=2, color='b')
plt.plot(x, y2, '-', linewidth=2, color='g')
plt.plot(x, y3, '-', linewidth=2, color='r')
plt.plot(x1, y, '-', linewidth=2, color='y')
plt.plot(x, y4, '-', linewidth=2, color='c')
plt.plot(x, z, ':', linewidth=1, color='k')

plt.grid()
plt.xlabel('Asientos para fumadores')
plt.ylabel('Asientos para no fumadores')
plt.title('Método Gráfico')

primera_interseccion = cuarta_linea.intersection(primera_linea)
segunda_interseccion = primera_linea.intersection(segunda_linea)
tercera_interseccion = segunda_linea.intersection(tercera_linea)
cuarta_interseccion = tercera_linea.intersection(cuarta_linea)

plt.plot(*primera_interseccion.xy, 'o')
plt.plot(*segunda_interseccion.xy, 'o')
plt.plot(*tercera_interseccion.xy, 'o')
plt.plot(*cuarta_interseccion.xy, 'o')

plt.show()
```



Es importante determinar el valor de las coordenadas en cada uno de los vértices, ya que con ello podemos evaluar la ecuación de la función objetivo para cada vértice:

In [14]:
```
print('\n COORDENADAS DE LAS INTERSECCIONES')
print('Coordenadas de la primera intersección: {} '.format(primera_interseccion))
print('Coordenadas de la segunda intersección: {} '.format(segunda_interseccion))
print('Coordenadas de la tercera intersección: {} '.format(tercera_interseccion))
print('Coordenadas de la cuarta intersección: {} '.format(cuarta_interseccion))
```

```
 COORDENADAS DE LAS INTERSECCIONES
Coordenadas de la primera intersección: POINT (0 60)
Coordenadas de la segunda intersección: POINT (50 40)
Coordenadas de la tercera intersección: POINT (80 10)
Coordenadas de la cuarta intersección: POINT (0 10)
```

Paso 6: Determinar la solución óptima

Es el objetivo de la solución gráfica. Lo único que debemos hacer es, de acuerdo a los valores de cada vértice, evaluar la ecuación de la función objetivo en cada uno de ellos, de manera que podamos determinar cual es el mejor resultado (de acuerdo al criterio de optimización, en este caso: maximizar).

Es necesario aclarar que el valor de los vértices obtenidos corresponde a un formato de tipo point (punto), y que no es posible operar cada una de las variables (x, y) que contiene sin antes extraerlas. Por esa razón, lo siguiente que debemos hacer es extraer el valor de cada variable contenida en cada punto de manera independiente:

In [15]:
```
xi1m, yi1m = primera_interseccion.xy
xi2m, yi2m = segunda_interseccion.xy
xi3m, yi3m = tercera_interseccion.xy
xi4m, yi4m = cuarta_interseccion.xy
xi1m, yi1m
```

Out[15]: (array('d', [0.0]), array('d', [60.0]))

In [16]:
```
xi2m, yi2m
```

Out[16]: (array('d', [50.0]), array('d', [40.0]))

In [17]:
```
xi3m, yi3m
```

Out[17]: (array('d', [80.0]), array('d', [10.0]))

In [18]:
```
xi4m, yi4m
```

Out[18]: (array('d', [0.0]), array('d', [10.0]))

Cambiamos el formato de matriz a float

In [19]:
```
xi1 = np.float64(np.array(xi1m))
xi2 = np.float64(np.array(xi2m))
xi3 = np.float64(np.array(xi3m))
xi4 = np.float64(np.array(xi4m))
yi1 = np.float64(np.array(yi1m))
yi2 = np.float64(np.array(yi2m))
yi3 = np.float64(np.array(yi3m))
yi4 = np.float64(np.array(yi4m))

xi1, xi2, xi3, xi4, yi1, yi2, yi3, yi4
```

Out[19]: (0.0, 50.0, 80.0, 0.0, 60.0, 40.0, 10.0, 10.0)

Evaluando y emitiendo la función objetivo en cada vértice

In [20]:
```
FOi1 = (xi1 * 10000) + (yi1 * 6000)
FOi2 = (xi2 * 10000) + (yi2 * 6000)
FOi3 = (xi3 * 10000) + (yi3 * 6000)
FOi4 = (xi4 * 10000) + (yi4 * 6000)

print('\n EVALUACIÓN DE LA FO EN LOS VÉRTICES')
print('Función objetivo en la intersección 1: {} pesos'.format(FOi1))
print('Función objetivo en la intersección 2: {} pesos'.format(FOi2))
print('Función objetivo en la intersección 3: {} pesos'.format(FOi3))
print('Función objetivo en la intersección 4: {} pesos'.format(FOi4))
```

```
 EVALUACIÓN DE LA FO EN LOS VÉRTICES
Función objetivo en la intersección 1: 360000.0 pesos
Función objetivo en la intersección 2: 740000.0 pesos
Función objetivo en la intersección 3: 860000.0 pesos
Función objetivo en la intersección 4: 60000.0 pesos
```

Se debe determinar su valor máximo o mínimo, según el caso. En nuestro problema, requerimos determinar el valor máximo.

Calcula y emitir el mejor resultado (Maximizar)

In [21]:
```
ZMAX = max(FOi1, FOi2, FOi3, FOi4)

print('\n SOLUCIÓN ÓPTIMA')
print('Solución óptima: {} pesos'.format(ZMAX))
```

```
 SOLUCIÓN ÓPTIMA
Solución óptima: 860000.0 pesos
```

Se necesita también emitir el valor de las variables de decisión, es decir, el valor de las coordenadas x e y en el vértice donde se encuentra la solución óptima (punto óptimo).

Lo primero que vamos a hacer es organizar la información, es decir, crear un vector que almacene los valores de cada variable (sea x o y) en todos los vértices. El vector m contendrá los valores (ordenados) de las variables x en cada uno de los vértices, empezando por el vértice 1 y finalizando en el vértice 4. Lo mismo se realiza con las variables y.

Ordenando las coordenadas de los vértices (Las coordenadas x en m y las coordenadas y en n)

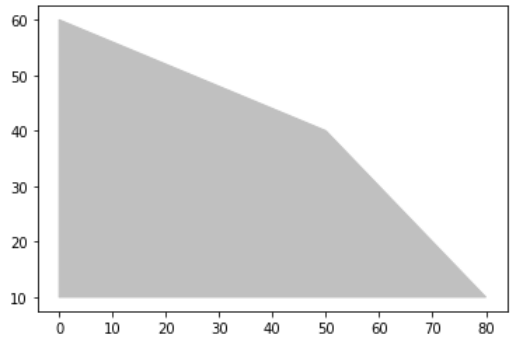
In [22]:
```
m = [xi1, xi2, xi3, xi4]
n = [yi1, yi2, yi3, yi4]

m, n
```

Out[22]: ([0.0, 50.0, 80.0, 0.0], [60.0, 40.0, 10.0, 10.0])

Graficar el polígono solución a partir de las coordenadas de los vértices

In [23]:
```
plt.fill(m, n, color='silver')
```

Out[23]: [<matplotlib.patches.Polygon at 0x28590e07a30>]



Creamos un arreglo con las variables que contienen los resultados de las funciones objetivo (FOi1, FOi2, FOi3 y FOi4), y a cada uno le asignaremos un índice de acuerdo a su orden:

In [24]:
```
dict1 = {0:FOi1, 1:FOi2, 2:FOi3, 3:FOi4}
dict1
```

Out[24]: {0: 360000.0, 1: 740000.0, 2: 860000.0, 3: 60000.0}

Conocemos que el mayor valor es el de FOi3, por esta razón, la función debe arrojarnos como respuesta el índice 2. La siguiente función nos traerá el índice de acuerdo al mayor valor de la variable:

In [25]:
```
posicion = max(dict1, key=dict1.get)
posicion
```

Out[25]: 2

Tenemos 2 vectores donde se almacenan las variables x e y de cada uno de los vértices. Con la variable posicion podemos acceder a la variable específica dentro de estos vectores, es decir, a la posición en la que se encuentra cada una de las variables asociadas al vértice solución.

Según lo trabajado hasta ahora, podemos inferir que si la función objetivo solución se encuentra en la variable FOi3, y sus coordenadas serán xi3 y y13. Así entonces, si tenemos la posición de la función objetivo dentro de un arreglo (dict1), esa misma posición corresponderá a la de cada variable dentro del vector que la contenga (el vector m contiene las variables x de los vértices, mientras el vector n contiene las variables y de los vértices).

Obteniendo y emitiendo las coordenadas del vértice de la mejor solución de acuerdo al índice del paso anterior

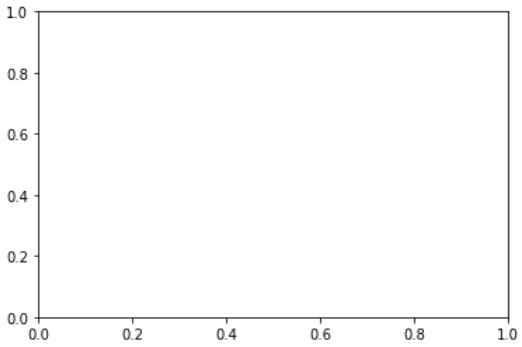
In [26]:
```
XMAX = m[posicion]
YMAX = n[posicion]

print('\n VARIABLES DE DECISIÓN')
print('Cantidad de asientos a reservar para fumadores: {} '.format(XMAX))
print('Cantidad de asientos a reservar para no fumadores: {} '.format(YMAX))
```

```
 VARIABLES DE DECISIÓN
Cantidad de asientos a reservar para fumadores: 80.0
Cantidad de asientos a reservar para no fumadores: 10.0
```

Generando las anotaciones de las coordenadas y solución óptima en el gráfico

In [27]:
```
plt.annotate('  X: {0} / Y: {1} (Coordenadas)'.format(XMAX, YMAX), (XMAX, YMAX))
plt.annotate('  Solución óptima: {}'.format(ZMAX), (XMAX, YMAX+3))
```

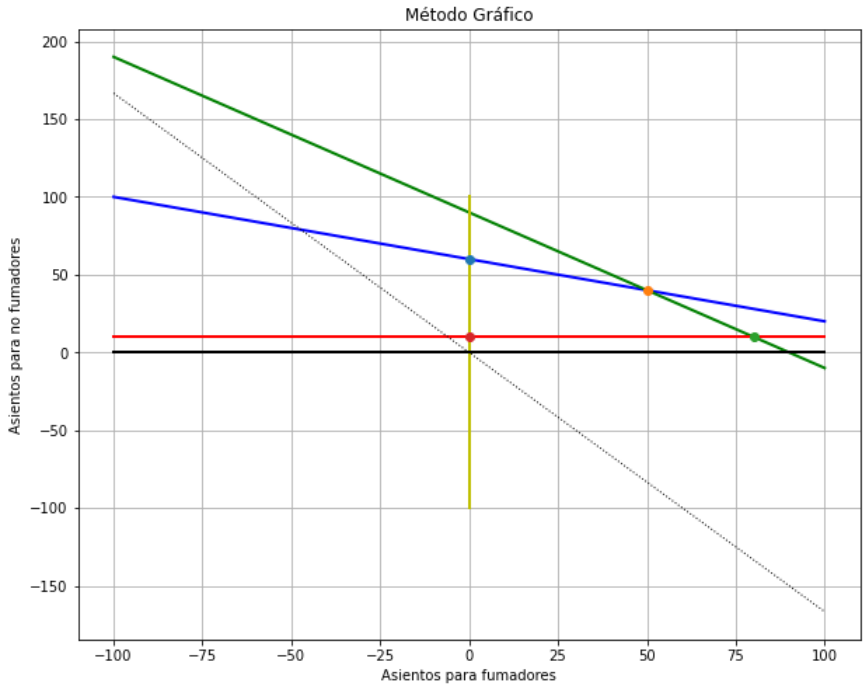
Out[27]: Text(80.0, 13.0, ' Solución óptima: 860000.0')



Programa completo

In [40]:

```
import matplotlib.pyplot as plt
import numpy as np
import shapely
from shapely.geometry import LineString

x = np.arange(-100, 150, 50)
y = np.arange(-100, 150, 50)
y1 = (3000 - (20 * x))/ 50
y2 = 90 - x
y3 = 10 + (0 * x)
x1 = 0 * y
y4 = 0 * x
z = (-10000 * x) / 6000

primera_linea = LineString(np.column_stack((x, y1)))
segunda_linea = LineString(np.column_stack((x, y2)))
tercera_linea = LineString(np.column_stack((x, y3)))
cuarta_linea = LineString(np.column_stack((x1, y)))
quinta_linea = LineString(np.column_stack((x, y4)))
sexta_linea = LineString(np.column_stack((x, z)))

plt.figure(figsize=(10,8))
plt.plot(x, y1, '-', linewidth=2, color='b')
plt.plot(x, y2, '-', linewidth=2, color='g')
plt.plot(x, y3, '-', linewidth=2, color='r')
plt.plot(x1, y, '-', linewidth=2, color='y')
plt.plot(x, y4, '-', linewidth=2, color='k')
plt.plot(x, z, ':', linewidth=1, color='k')


plt.grid()
plt.xlabel('Asientos para fumadores')
plt.ylabel('Asientos para no fumadores')
plt.title('Método Gráfico')

primera_interseccion = cuarta_linea.intersection(primera_linea)
segunda_interseccion = primera_linea.intersection(segunda_linea)
tercera_interseccion = segunda_linea.intersection(tercera_linea)
cuarta_interseccion = tercera_linea.intersection(cuarta_linea)

plt.plot(*primera_interseccion.xy, 'o')
plt.plot(*segunda_interseccion.xy, 'o')
plt.plot(*tercera_interseccion.xy, 'o')
plt.plot(*cuarta_interseccion.xy, 'o')


plt.show()

print('\n COORDENADAS DE LAS INTERSECCIONES')
print('Coordenadas de la primera intersección: {} '.format(primera_interseccion))
print('Coordenadas de la segunda intersección: {} '.format(segunda_interseccion))
print('Coordenadas de la tercera intersección: {} '.format(tercera_interseccion))
print('Coordenadas de la cuarta intersección: {} '.format(cuarta_interseccion))
```




```
 COORDENADAS DE LAS INTERSECCIONES
Coordenadas de la primera intersección: POINT (0 60)
```

```
Coordenadas de la segunda intersección: POINT (50 40)
Coordenadas de la tercera intersección: POINT (80 10)
Coordenadas de la cuarta intersección: POINT (0 10)
```

In [29]:

```
xi1m, yi1m = primera_interseccion.xy
xi2m, yi2m = segunda_interseccion.xy
xi3m, yi3m = tercera_interseccion.xy
xi4m, yi4m = cuarta_interseccion.xy

xi1 = np.float64(np.array(xi1m))
xi2 = np.float64(np.array(xi2m))
xi3 = np.float64(np.array(xi3m))
xi4 = np.float64(np.array(xi4m))
yi1 = np.float64(np.array(yi1m))
yi2 = np.float64(np.array(yi2m))
yi3 = np.float64(np.array(yi3m))
yi4 = np.float64(np.array(yi4m))

FOi1 = (xi1 * 10000) + (yi1 * 6000)
FOi2 = (xi2 * 10000) + (yi2 * 6000)
FOi3 = (xi3 * 10000) + (yi3 * 6000)
FOi4 = (xi4 * 10000) + (yi4 * 6000)

print('\n EVALUACIÓN DE LA FO EN LOS VÉRTICES')
print('Función objetivo en la intersección 1: {} pesos'.format(FOi1))
print('Función objetivo en la intersección 2: {} pesos'.format(FOi2))
print('Función objetivo en la intersección 3: {} pesos'.format(FOi3))
print('Función objetivo en la intersección 4: {} pesos'.format(FOi4))
```

```
 EVALUACIÓN DE LA FO EN LOS VÉRTICES
Función objetivo en la intersección 1: 360000.0 pesos
Función objetivo en la intersección 2: 740000.0 pesos
Función objetivo en la intersección 3: 860000.0 pesos
Función objetivo en la intersección 4: 60000.0 pesos
```

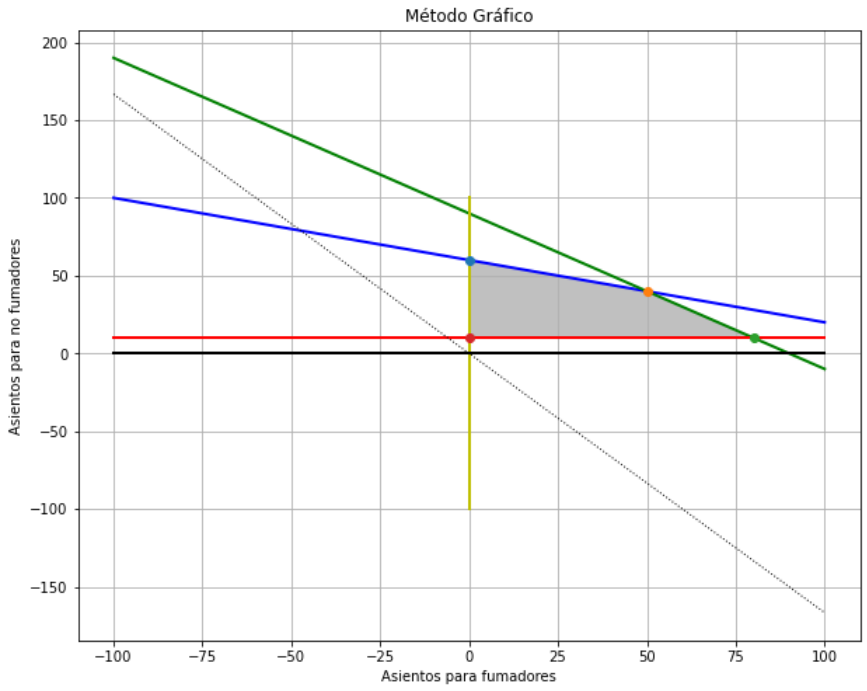
In [30]:

```
ZMAX = max(FOi1, FOi2, FOi3, FOi4)
print('\n SOLUCIÓN ÓPTIMA')
print('Solución óptima: {} pesos'.format(ZMAX))
```

```
 SOLUCIÓN ÓPTIMA
Solución óptima: 860000.0 pesos
```

In [41]:

```
plt.figure(figsize=(10,8))
plt.plot(x, y1, '-', linewidth=2, color='b')
plt.plot(x, y2, '-', linewidth=2, color='g')
plt.plot(x, y3, '-', linewidth=2, color='r')
plt.plot(x1, y, '-', linewidth=2, color='y')
plt.plot(x, y4, '-', linewidth=2, color='k')
plt.plot(x, z, ':', linewidth=1, color='k')

primera_interseccion = cuarta_linea.intersection(primera_linea)
segunda_interseccion = primera_linea.intersection(segunda_linea)
tercera_interseccion = segunda_linea.intersection(tercera_linea)
cuarta_interseccion = tercera_linea.intersection(cuarta_linea)

plt.plot(*primera_interseccion.xy, 'o')
plt.plot(*segunda_interseccion.xy, 'o')
plt.plot(*tercera_interseccion.xy, 'o')
plt.plot(*cuarta_interseccion.xy, 'o')

plt.grid()
plt.xlabel('Asientos para fumadores')
plt.ylabel('Asientos para no fumadores')
plt.title('Método Gráfico')

m = [xi1, xi2, xi3, xi4]
n = [yi1, yi2, yi3, yi4]

plt.fill(m, n, color='silver')

plt.show()
```



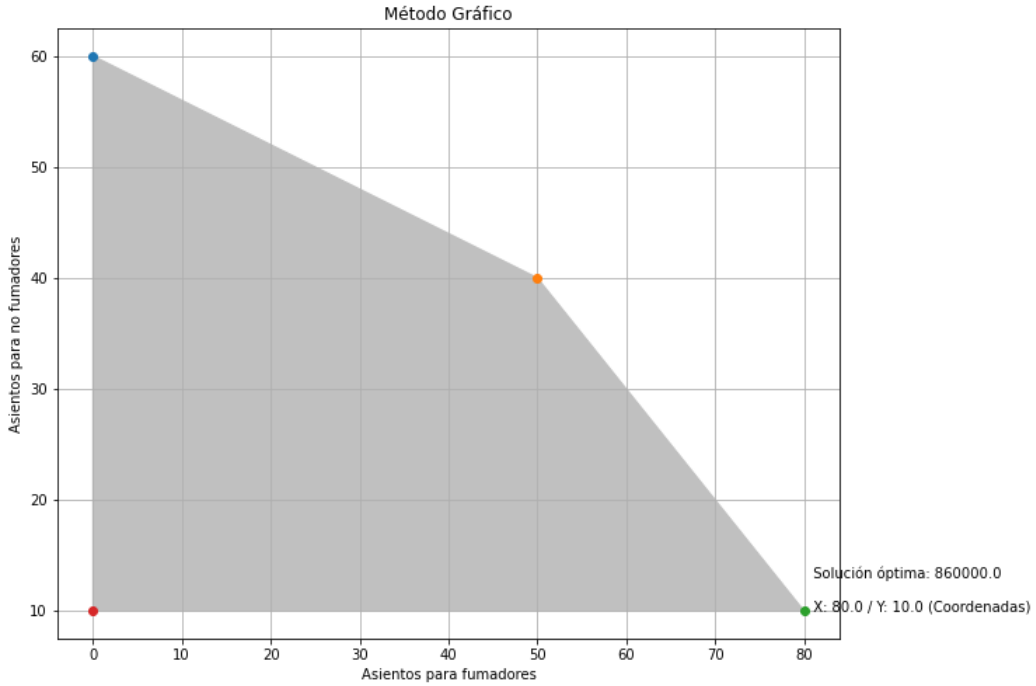
In [32]:

```
dict1 = {0:FOi1, 1:FOi2, 2:FOi3, 3:FOi4}
posicion = max(dict1, key=dict1.get)

XMAX = m[posicion]
YMAX = n[posicion]

print('\n VARIABLES DE DECISIÓN')
print('Cantidad de asientos a reservar para fumadores: {} '.format(XMAX))
print('Cantidad de asientos a reservar para no fumadores: {} '.format(YMAX))
```

```
 VARIABLES DE DECISIÓN
Cantidad de asientos a reservar para fumadores: 80.0
Cantidad de asientos a reservar para no fumadores: 10.0
```

In [42]:

```
primera_interseccion = cuarta_linea.intersection(primera_linea)
segunda_interseccion = primera_linea.intersection(segunda_linea)
tercera_interseccion = segunda_linea.intersection(tercera_linea)
cuarta_interseccion = tercera_linea.intersection(cuarta_linea)

plt.figure(figsize=(10,8))
plt.plot(*primera_interseccion.xy, 'o')
plt.plot(*segunda_interseccion.xy, 'o')
plt.plot(*tercera_interseccion.xy, 'o')
plt.plot(*cuarta_interseccion.xy, 'o')

plt.grid()
plt.xlabel('Asientos para fumadores')
plt.ylabel('Asientos para no fumadores')
plt.title('Método Gráfico')

m = [xi1, xi2, xi3, xi4]
n = [yi1, yi2, yi3, yi4]

plt.fill(m, n, color='silver')

plt.annotate('  X: {0} / Y: {1} (Coordenadas)'.format(XMAX, YMAX), (XMAX, YMAX))
plt.annotate('  Solución óptima: {}'.format(ZMAX), (XMAX, YMAX+3))

plt.show()
```



In [ ]: