24 de Mayo de 2025

Instituto Politécnico Nacional

“Escuela Superior de Cómputo”

“Reto Programación asíncrona.”

Profesor: Luis Alberto Ibáñez Zamora

Alumno:

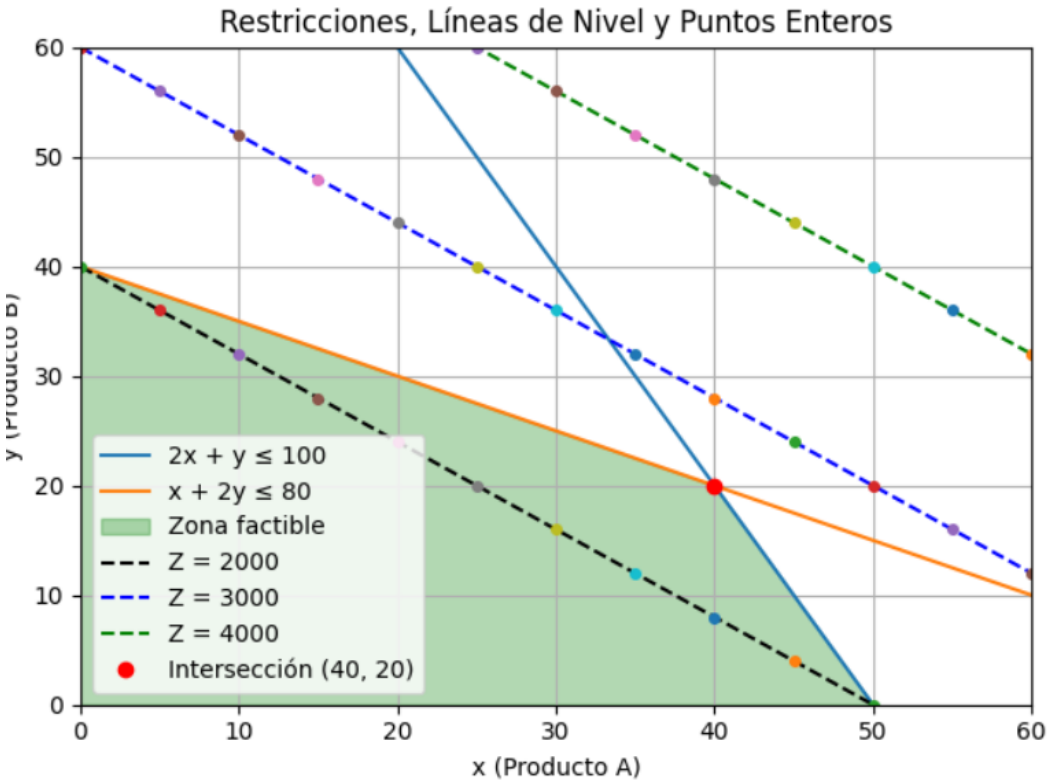
Ignacio Cortés Atzin Maxela

email:

ignacio.cortes.atzin.maxela@gmail.com

Grupo: 7BM2

Este programa grafica un problema de programación lineal con dos restricciones, tres líneas de nivel de una función objetivo y los puntos enteros que cumplen con dicha función. Calcula la intersección exacta de las restricciones resolviendo el sistema de ecuaciones, lo cual es clave para identificar posibles soluciones óptimas. Utiliza asyncio para organizar el código en funciones asíncronas, permitiendo una estructura modular y escalable, útil si se desea paralelizar cálculos en el futuro. La visualización incluye la región factible sombreada, las líneas de nivel y los puntos viables, lo que facilita interpretar gráficamente el comportamiento del modelo.



Las líneas azules representan las restricciones:

.2x+y≤ 100

.x+2y≤ 80

La zona sombreada en gris es la región factible, donde se cumplen ambas restricciones y x, y ≥ 0.

Las líneas punteadas representan diferentes niveles de la función objetivo Z = 40x + 50y. Cuanto más arriba y a la derecha se encuentre una línea dentro de la región factible, mayor es la ganancia.

El programa inicia importando las librerías necesarias: matplotlib.pyplot para graficar, numpy y pandas para manipulación de datos, y asyncio para permitir el uso de funciones asíncronas. Luego define dos funciones llamadas restriccion_1 y restriccion_2, que representan las restricciones del problema lineal. Estas funciones devuelven el valor de y despejado en función de x, derivado de las inecuaciones $2x + y \leq 100$ y $x + 2y \leq 80$, respectivamente. Esto permite graficar directamente las líneas de restricción.

Teniendo nuestra función objetivo: $40x+50y=Z$

$50y=Z-40x$

se despeja y:

$y=Z-40x/50$

y se grafica esa ecuación como una línea recta para cada valor de z

Cada valor distinto de ZZZ da una línea distinta pero paralela, que permite visualizar cómo se comporta la función objetivo al moverse en el plano.

Estas líneas nos permiten ver cómo se desplaza la función objetivo sobre la región factible. Al moverlas hacia arriba o abajo (aumentar o reducir Z), podemos ver cuál es el punto más extremo que todavía toca la zona válida, y así encontrar la máxima o mínima ganancia.

La función interseccion() se encarga de hallar el punto exacto donde se cruzan ambas restricciones. Para ello, iguala las dos ecuaciones resultantes: $100 - 2x = (80 - x)/2$. Resolviendo algebraicamente se obtiene $x = 40$ y luego $y = 20$. Este punto (40, 20) se considera crítico porque puede ser un punto óptimo en la solución del problema. La función devuelve estos valores para usarlos posteriormente en la gráfica.

La ganancia que representa ese punto es:

$Z=40(40)+50(20)=1600+1000=2600$

Este es el valor de la función objetivo en el punto de intersección. Este punto se encuentra dentro de la zona factible, entonces es candidato a la ganancia máxima.

La función puntos_enteros_nivel(z) busca todos los puntos (x, y) con valores enteros que cumplen la ecuación de la línea de nivel $40x + 50y = z$, con valores de z como 2000, 3000 y 4000. Itera sobre posibles valores de x y verifica si el valor correspondiente de y también es entero y positivo. Esto es útil en problemas donde las soluciones deben ser discretas, como el número de unidades a producir de un producto.

La función principal graficar_completo() genera la visualización del problema. Usa numpy.linspace para generar valores de x entre 0 y 60, y calcula y usando las restricciones. Luego grafica las dos restricciones como líneas y rellena el área entre ellas para marcar la región factible. Para cada valor de z en las líneas de nivel, grafica la línea respectiva y los puntos enteros encontrados. También marca la intersección de las restricciones en rojo y ajusta los límites del gráfico.

Una vez completado el gráfico, la función crea una tabla con los puntos ***enteros*** válidos usando pandas.DataFrame y la imprime. Todo el flujo del programa está estructurado usando funciones async y se ejecuta con await graficar_completo(). Aunque no hay paralelismo real en este ejemplo, usar asyncio permite que estas funciones puedan integrarse fácilmente en flujos más complejos con tareas concurrentes, manteniendo el código limpio y modular.

	Nivel	x	y
0	Z = 2000	0	40
1	Z = 2000	5	36
2	Z = 2000	10	32
3	Z = 2000	15	28
4	Z = 2000	20	24
5	Z = 2000	25	20
6	Z = 2000	30	16
7	Z = 2000	35	12
8	Z = 2000	40	8
9	Z = 2000	45	4
10	Z = 2000	50	0
11	Z = 3000	0	60
12	Z = 3000	5	56
13	Z = 3000	10	52
14	Z = 3000	15	48
15	Z = 3000	20	44
16	Z = 3000	25	40
17	Z = 3000	30	36
18	Z = 3000	35	32
19	Z = 3000	40	28
20	Z = 3000	45	24
21	Z = 3000	50	20
22	Z = 3000	55	16
23	Z = 3000	60	12
24	Z = 3000	65	8
25	Z = 3000	70	4
26	Z = 3000	75	0
27	Z = 4000	0	80
28	Z = 4000	5	76
29	Z = 4000	10	72
30	Z = 4000	15	68
31	Z = 4000	20	64
32	Z = 4000	25	60
33	Z = 4000	30	56
34	Z = 4000	35	52
35	Z = 4000	40	48
36	Z = 4000	45	44
37	Z = 4000	50	40
38	Z = 4000	55	36
39	Z = 4000	60	32
40	Z = 4000	65	28
41	Z = 4000	70	24
42	Z = 4000	75	20
43	Z = 4000	80	16
44	Z = 4000	85	12
45	Z = 4000	90	8
46	Z = 4000	95	4
47	Z = 4000	100	0