LA SOLUCIÓN DEL CÁLCULO ECONÓMICO [VI]: PROGRAMACIÓN ENTERA MIXTA

Tomas Härdin

ciber-comunismo

Introducción a la Primera Parte

En el primer artículo de esta serie, señalé que, aunque la programación lineal nos lleva bastante lejos en la planificación, no puede hacer frente a las no linealidades que existen en las economías reales. Para hacerme una idea de lo difícil que es esto, me he sumergido en la programación entera mixta (MIP, por sus siglas en inglés).

Un programa en MIP es como un programa lineal normal con la restricción adicional de que algunas variables deben ser enteras. Esto aparece siempre que no podemos hacer cantidades fraccionarias de algo. Por ejemplo, no tiene sentido construir medio coche. Muchos problemas de optimización combinatoria pueden formularse como MIP, como el problema de la mochila y el del viajante de comercio. Un problema muy relevante para la planificación es el de la localización de instalaciones, que abarca aspectos como cuándo y dónde construir nuevas instalaciones de producción. Utilizaré el juego de ordenador Workers & Resources: Soviet Republic como modelo de economía para ilustrar este post. He escrito un programa que analiza los archivos de datos del juego, sobre el que pienso escribir en otro post.

¿Cuándo es útil MIP?

```
max: cement;

// resources on hand
concrete = 1500;
gravel = 150;

cement = 2.7 plants;
739 plants <= concrete;
70 plants <= gravel;
int plants;
```

La MIP es útil siempre que haya que planificar cuándo y dónde realizar inversiones de capital. La instalación de una planta de cemento debe hacerse en su totalidad antes de ponerla en marcha. Si una planta puede hacer 2,7 toneladas de cemento por hora, dos plantas pueden hacer 5,4, pero 1½ plantas sólo pueden hacer 2,7. Obviamente, dos plantas incompletas no pueden fabricar cemento. Para simplificar, asumo que las plantas se construyen secuencialmente. El siguiente programa entero mixto ilustra el concepto:

Introduciéndolo en lp_solve, devolverá la solución plantas = 2, cemento = 5,4. Pero si el hormigón se reduce ligeramente a 1400, la solución óptima es plantas = 1, cemento = 2,7. Eliminando la restricción entera, manteniendo el hormigón = 1400 y la solución será plantas = 1,89445, cemento = 5,11502.

Sin embargo, este ejemplo es aburrido. No tenemos ninguna noción de tiempo. Añadamos una.

Tiempo discretizado

Una forma de introducir el tiempo en la MIP es usar la discretización. Ésta puede ser monótona para todas las variables en un intervalo de tiempo determinado, o podemos utilizar una rejilla no monótona, o incluso una rejilla no monótona distinta para cada variable. Para simplificar, utilizaremos la misma rejilla monótona para todas las variables.

La siguiente MIP cubre tres periodos de planificación/cuatro instantes de tiempo en una rejilla monótona:

```
max: cement_3;

// initial resources
cement_0 = 0;
concrete_0 = 1500;
gravel_0 = 200;
coal_0 = 10;

// size of the working population
workers = 6000;

// initial plants
plants_0 = 0;

// 0 -> 1
workers_cement_0 <= 30 plants_0;
cement_prod_0 = 0.09 workers_cement_0;
coal_cement_0 = 0.025 workers_cement_0;
gravel_cement_0 = 0.233 workers_cement_0;
5676 new_plants_0 = workers_construct_0;
739 new_plants_0 = concrete_construct_0;
70 new_plants_0 = gravel_construct_0;
workers_cement_0 + workers_construct_0 <= workers;
plants_1 = plants_0 + new_plants_0;
cement_1 = cement_0 + cement_prod_0;
coal_1 = coal_0 - coal_cement_0;
gravel_1 = gravel_0 - gravel_cement_0 - gravel_construct_0;
concrete_1 = concrete_0 - concrete_construct_0;

// 1 -> 2
workers_cement_1 <= 30 plants_1;
cement_prod_1 = 0.09 workers_cement_1;
coal_cement_1 = 0.025 workers_cement_1;
gravel_cement_1 = 0.233 workers_cement_1;
5676 new_plants_1 = workers_construct_1;
739 new_plants_1 = concrete_construct_1;
70 new_plants_1 = gravel_construct_1;
workers_cement_1 + workers_construct_1 <= workers;
plants_2 = plants_1 + new_plants_1;
cement_2 = cement_1 + cement_prod_1;
coal_2 = coal_1 - coal_cement_1;
gravel_2 = gravel_1 - gravel_cement_1 - gravel_construct_1;
concrete_2 = concrete_1 - concrete_construct_1;

// 2 -> 3
workers_cement_2 <= 30 plants_2;
cement_prod_2 = 0.09 workers_cement_2;
coal_cement_2 = 0.025 workers_cement_2;
gravel_cement_2 = 0.233 workers_cement_2;
cement_3 = cement_2 + cement_prod_2;
coal_3 = coal_2 - coal_cement_2;
gravel_3 = gravel_2 - gravel_cement_2;

int new_plants_0;
int new_plants_1;
```

Esto puede parecer intimidante, pero los tres bloques más grandes de ecuaciones son en su mayoría copias unos de otros. El último bloque está simplificado. Veamos el primer bloque con más detalle:

```
workers_cement_0 <= 30 plants_0;
cement_prod_0 = 0.09 workers_cement_0;
coal_cement_0 = 0.025 workers_cement_0;
gravel_cement_0 = 0.233 workers_cement_0;
```

Esta parte cubre la producción en las plantas de cemento. Cada planta puede emplear hasta 30 trabajadores, y cada trabajador produce 0,09 toneladas de cemento por hora dadas 0,025 toneladas de carbón y 0,233 toneladas de grava.

```
5676 new_plants_0 = workers_construct_0;
739 new_plants_0 = concrete_construct_0;
70 new_plants_0 = gravel_construct_0;
```

Esta parte abarca la construcción de nuevas plantas. Es similar a lo que teníamos en la sección anterior, salvo que se añade una restricción para los trabajadores de la construcción. Para simplificar, cada planta se construye completamente en un periodo de planificación (¡una hora!).

```
workers_cement_0 + workers_construct_0 <= workers;
```

Esto limita los lugares en los que se puede emplear a los trabajadores, ya sea en la construcción o en la producción de cemento.

```
plants_1 = plants_0 + new_plants_0;
cement_1 = cement_0 + cement_prod_0;
coal_1 = coal_0 - coal_cement_0;
gravel_1 = gravel_0 - gravel_cement_0 - gravel_construct_0;
concrete_1 = concrete_0 - concrete_construct_0;
```

Esto describe cómo cambian los diferentes medios de producción de un instante de tiempo a otro. La solución tiene este aspecto, después de ordenar las variables para poder ver cómo cambian en el tiempo:

```
Value of objective function: 8.10000000
cement_0                       0
cement_1                       0
cement_2                     2.7
cement_3                     8.1
cement_prod_0                  0
cement_prod_1                2.7
cement_prod_2                5.4
coal_0                        10
coal_1                        10
coal_2                      9.25
coal_3                      7.75
coal_cement_0                  0
coal_cement_1               0.75
coal_cement_2                1.5
concrete_0                  1500
concrete_1                   761
concrete_2                    22
concrete_construct_0         739
concrete_construct_1         739
gravel_0                      200
gravel_1                      130
gravel_2                    53.01
gravel_3                    39.03
gravel_cement_0                0
gravel_cement_1             6.99
gravel_cement_2            13.98
gravel_construct_0            70
gravel_construct_1            70
new_plants_0                   1
new_plants_1                   1
plants_0                       0
plants_1                       1
plants_2                       2
workers                     6000
workers_cement_0               0
workers_cement_1              30
workers_cement_2              60
workers_construct_0         5676
workers_construct_1         5676
```

Como estamos maximizando la producción de cemento, vemos que la cantidad de cemento aumenta con el tiempo. La grava se gasta tanto en la construcción como en la producción de cemento. Te animo, querido lector, a experimentar un poco con este programa respondiendo a las siguientes preguntas.

- ¿Qué ocurre si coal_0 se limita a 2 toneladas? ¿1 tonelada?
- ¿Qué ocurre si grave_0 se limita a 150?
- ¿Qué ocurre si se duplica el número de trabajadores? ¿Se limita a 5676? 5675?
- ¿Qué ocurre si plants_0 se fija a 1?

El siguiente script de python3 generaliza la MIP dado anteriormente a cualquier número de períodos:

```
import sys
n = int(sys.argv[1])

print(f"""max: cement_{n:06d};

// initial resources
cement_000000 = 0;
concrete_000000 = 1500;
gravel_000000 = 200;
coal_000000 = 10;

// size of the working population
workers = 6000;

// initial plants
plants_000000 = 0;""")

for i in range(n-1):
  print(f"""// {i:06d} -> {i+1:06d}
workers_cement_{i:06d} <= 30 plants_{i:06d};
cement_prod_{i:06d} = 0.09 workers_cement_{i:06d};
coal_cement_{i:06d} = 0.025 workers_cement_{i:06d};
gravel_cement_{i:06d} = 0.233 workers_cement_{i:06d};
5676 new_plants_{i:06d} = workers_construct_{i:06d};
739 new_plants_{i:06d} = concrete_construct_{i:06d};
70 new_plants_{i:06d} = gravel_construct_{i:06d};
workers_cement_{i:06d} + workers_construct_{i:06d} <= workers;
plants_{i+1:06d} = plants_{i:06d} + new_plants_{i:06d};
cement_{i+1:06d} = cement_{i:06d} + cement_prod_{i:06d};
coal_{i+1:06d} = coal_{i:06d} - coal_cement_{i:06d};
gravel_{i+1:06d} = gravel_{i:06d} - gravel_cement_{i:06d} - gravel_construct_{i:06d};
concrete_{i+1:06d} = concrete_{i:06d} - concrete_construct_{i:06d};""")

print(f"""// {n-1:06d} -> {n:06d}
workers_cement_{n-1:06d} <= 30 plants_{n-1:06d};
cement_prod_{n-1:06d} = 0.09 workers_cement_{n-1:06d};
coal_cement_{n-1:06d} = 0.025 workers_cement_{n-1:06d};
gravel_cement_{n-1:06d} = 0.233 workers_cement_{n-1:06d};
cement_{n:06d} = cement_{n-1:06d} + cement_prod_{n-1:06d};
coal_{n:06d} = coal_{n-1:06d} - coal_cement_{n-1:06d};
gravel_{n:06d} = gravel_{n-1:06d} - gravel_cement_{n-1:06d};""")

for i in range(n-1):
  print(f"int new_plants_{i:06d};")
```

El script anterior toma un único argumento posicional para el número de períodos del plan. Para generar y resolver un programa para N períodos, haga lo siguiente:

```
$ python generate.py $N | lp_solve
```

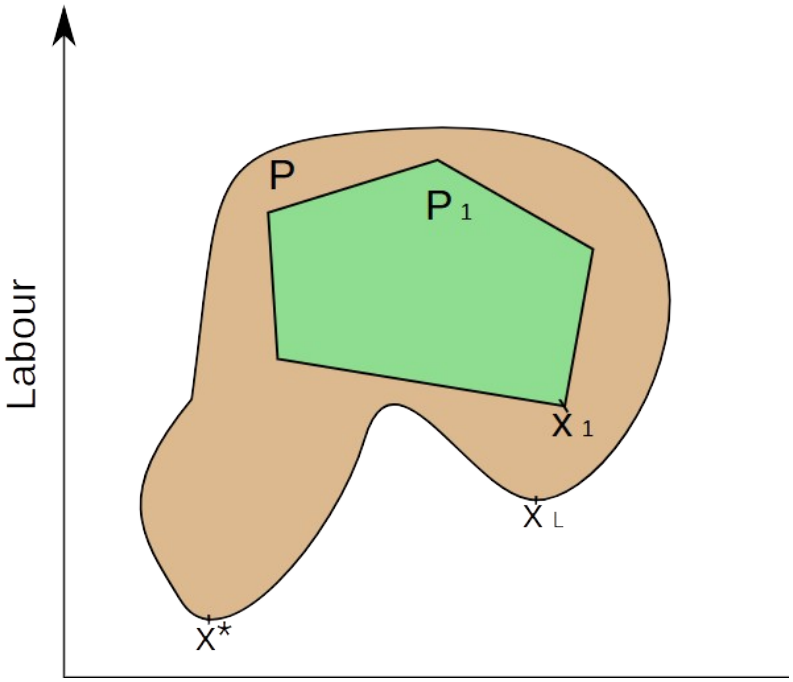
La versión 5.5.2.5-2 de lp_solve resuelve esta clase de programas en tiempo $O(N^2)$, incluso cuando se eliminan las restricciones enteras. Esto no demuestra la NP-complejidad como me gustaría. También he pasado estos problemas por la versión 5.0-1 de glpsol, que forma parte del paquete glpk, y los resuelve en tiempo O(N).

La introducción de más industrias en el programa anterior se deja como ejercicio para el lector. La clase de programas resultante es del tipo diagonal por bloques.

Planificación no convexa

Una de las limitaciones del esquema anterior es que parte de la base de que la productividad es totalmente lineal, o posiblemente "grumosa" si hay que optimizar las inversiones de capital. No hay forma de abordar la economía de escala, que es una de las principales debilidades de la propuesta de Kantorovich de utilizar la programación lineal para la planificación. Más recientemente se ha señalado la debilidad de la PL para la planificación en la entrada de blog *In Soviet Union, Optimization Problem Solves You* por Cosma Shalizi.

Encontrar soluciones óptimas para programas no convexos es, en general, difícil. Sabemos que encontrar soluciones aproximadas a programas lineales es P, y que una solución que esté dentro del 1% del óptimo es probablemente lo suficientemente buena para la planificación. Para cualquier programa no convexo con una región factible no vacía podemos inscribir un programa lineal que a su vez puede resolverse de forma aproximada en P. La siguiente figura ilustra el concepto:

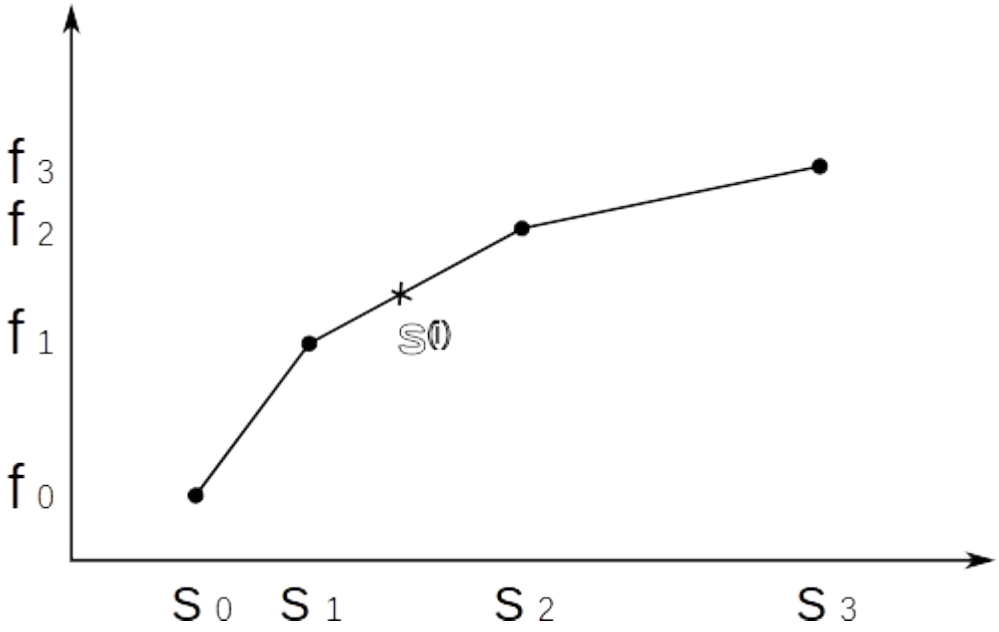


Es evidente que este proceso acabaría por conducir al óptimo local x_L después de algún tiempo. Y esto no es diferente a cómo el capitalismo tiende a encontrar óptimos locales. Pero podemos aplicar ideas de la optimización no lineal para salir de los óptimos locales, por ejemplo el recocido simulado o el reinicio aleatorio. Podemos utilizar diferentes programas lineales inscritos: hay un número infinito de ellos para elegir.

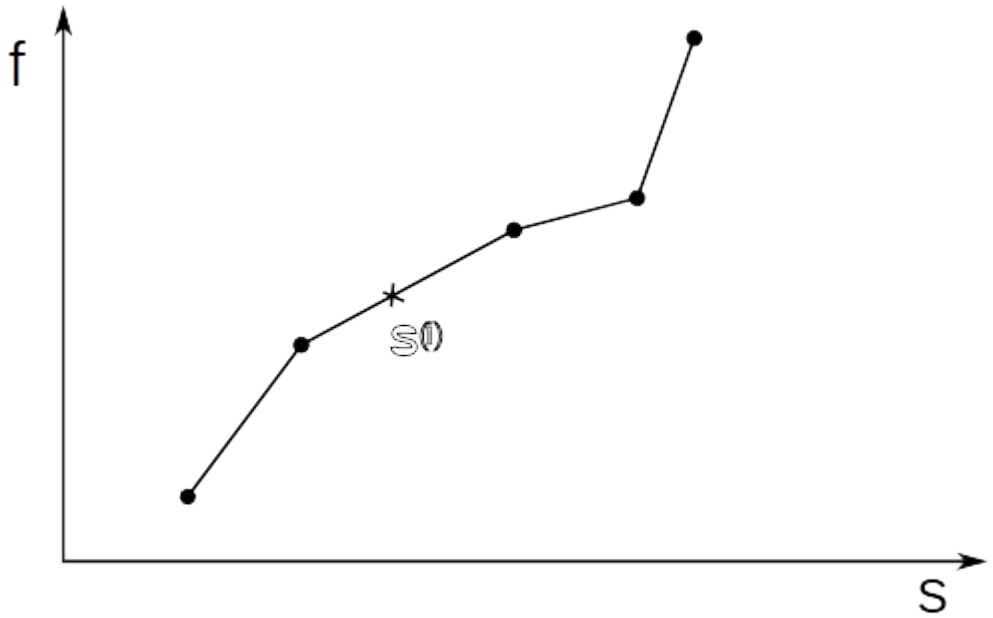
¿Pero cómo se aplica esto a la planificación?

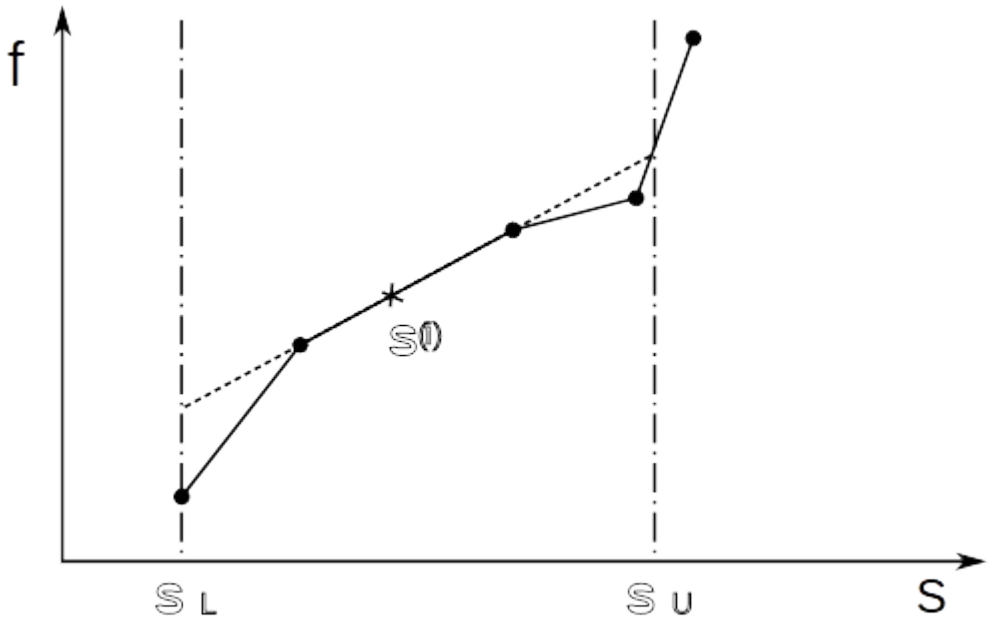
Linealización a trozos

En MIP hay algo llamado conjuntos especiales ordenados que se puede utilizar para resolver programas no convexos a través de la linealización a trozos. lp_solve puede hacer esto a través del comando SOS. Para estas linealizaciones también podemos encontrar funciones lineales que resultan en programas lineales inscritos como se discutió en la sección anterior. Podemos utilizar la solución factible actual para "apretar" el programa lineal inscrito acercando estas funciones lineales a las linealizaciones. La siguiente figura ilustra la idea:



s es la cantidad de algo que hay que fabricar y *f* es la cantidad de algún recurso que se necesita. La figura ilustra una economía de escala: a medida que S aumenta, la demanda marginal de *f* disminuye ($d^2f/ds^2 < 0$). En cada iteración tenemos un s*(i)* que podemos utilizar para elegir la linealización, siempre que no subestime *f*. También podemos añadir límites a s para poder ser más agresivos con nuestra elección de la linealización. Esto es útil si d^2f/ds^2 cambia de signo, como a continuación:

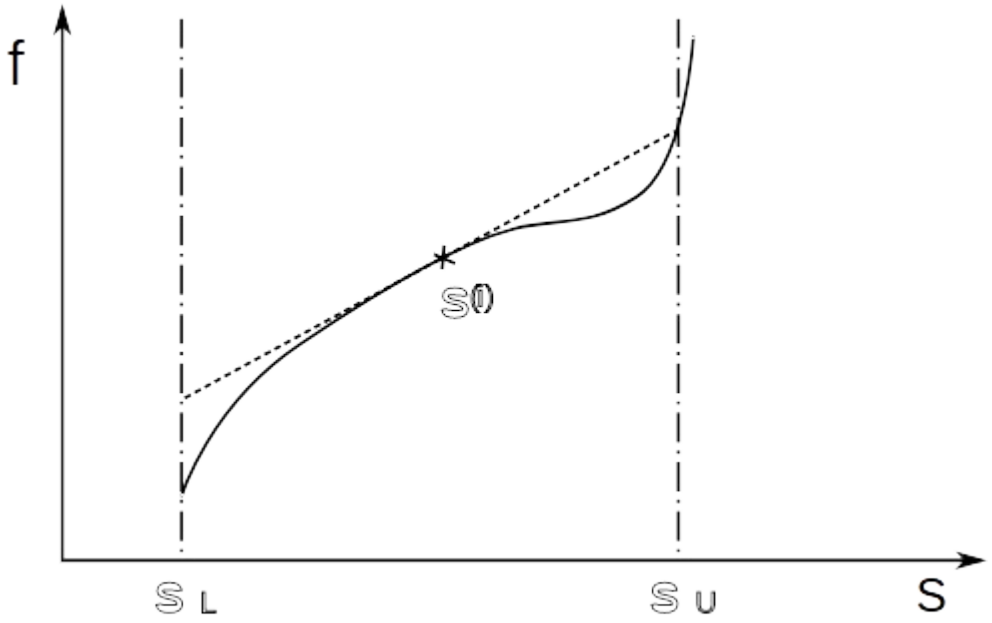




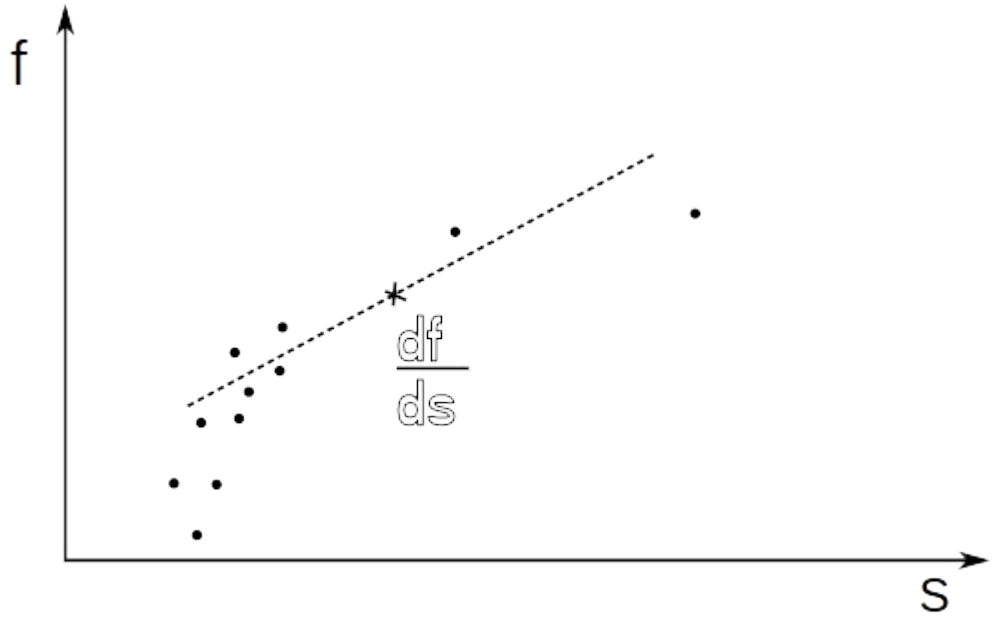
Aquí la linealización de f es válida para $s_L \leq s \leq s_U$.

Funciones continuas

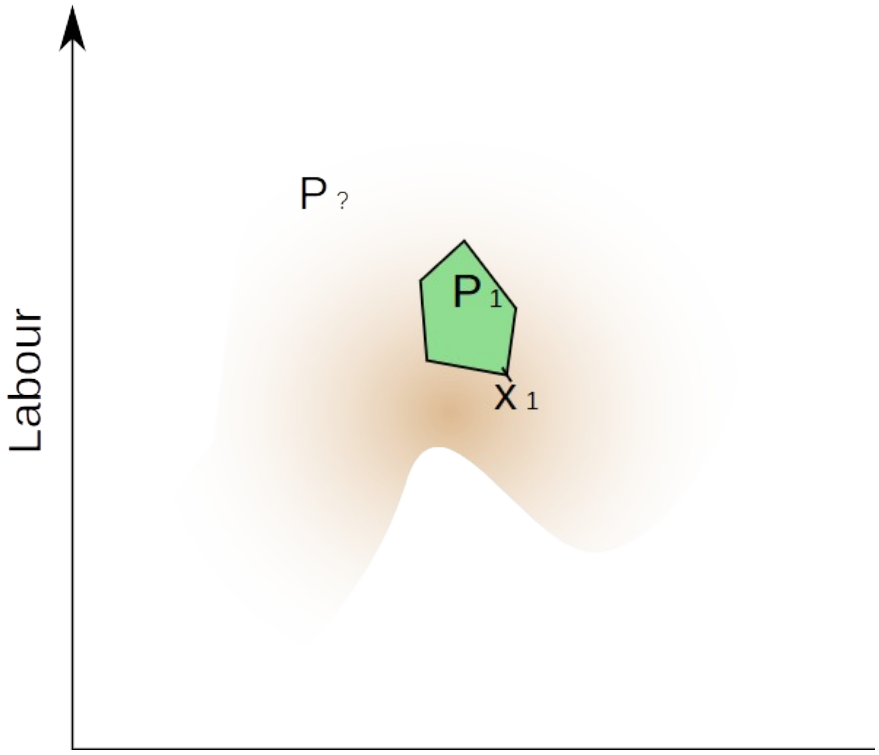
Hasta ahora sólo hemos visto las linealizaciones a trozos. La razón es que podemos resolverlas exactamente con cualquier solucionador de MIP. Otra razón es que podemos preguntar a la gente en cada puesto de trabajo cuánto material creen que necesitan para efectuar grandes cambios en la producción. "¿Cuánto necesitas ahora? ¿Cuánto para hacer el doble? ¿La mitad?". Estas preguntas, si se pueden responder, son puntos discretos. Una opción obvia para la interpolación es una función lineal a trozos. Pero también podemos elegir alguna función continua, ¡y la estrategia de linealización funciona igual de bien!



Pero la mayoría de las veces no es posible que la gente responda a las preguntas sobre los cambios a gran escala en la producción. Lo único que sabemos con certeza son los valores históricos de *s* y *f*. Si tenemos suerte, podríamos obtener una estimación decente de *df/ds*, la respuesta a la pregunta: ¿cuánto más/menos se necesita para cambiar la producción en una pequeña cantidad? Esto se parece a lo siguiente:



Debo esta noción a una reciente discusión con Spyridon Samothrakis y Dave Zachariah. Así que una mejor visión de la situación podría ser algo así:



El programa lineal es más pequeño porque estamos menos seguros de cuáles son las funciones que componen nuestras restricciones. Por lo tanto, ¿llamamos al programa incierto que estamos tratando de resolver P?

Conclusiones

Una característica de la formalización que aquí se discute es que mejora las propuestas "Leontief-y" para la planificación no convexa. Por Leontief me refiero aquí a los sistemas cuadrados basados en tablas input-output (IO). Resulta que necesitamos sistemas rectangulares en la planificación, pues de lo contrario no podemos sopesar múltiples métodos para alcanzar el mismo objetivo. Las tablas IO no pueden ayudarnos a elegir entre quemar antracita, madera o uranio a la hora de generar electricidad.

Las incertidumbres en las funciones que componen nuestras restricciones implican límites en las optimizaciones que podemos aplicar realmente. Sólo podemos acercarnos a las restricciones que estamos seguros que no son optimistas. Podemos tener en cuenta esta incertidumbre al resolver asignando "fuerzas" apropiadas a cada restricción, es decir, diferentes constantes antes de cada término $c_i(x)$ en la función de barrera utilizada (ver métodos de punto interior). Esto tenderá a empujar las soluciones hacia las restricciones más seguras.

La física puede utilizarse para verificar aspectos de algunas restricciones. La termodinámica es un ejemplo de ello. También podemos aplicar la conservación de la masa y la energía, exigiendo que se tenga en cuenta el flujo de éstas. Si un aserradero se alimenta de troncos, esperamos que la masa de tablas, serrín y corteza que se produzca no sea mayor que la suministrada. También esperamos que no sea mucha menos de la que se introduce, restando la humedad que se pierde en el secado.

Si violamos una restricción que existe en la realidad pero que no hemos modelado, entonces tenemos un plan inviable. Digamos que hoy descubrimos que el CO2 atmosférico afecta al equilibrio energético de la Tierra. Ese nuevo conocimiento de la realidad haría inmediatamente inviable nuestro plan actual, y tendríamos que cambiar el plan en consecuencia. Cuanto más "rígido" sea nuestro plan, más difícil será reorientar la economía.

Sospecho que el objetivo más importante a corto plazo puede no ser la optimización, sino la viabilidad. Sólo podemos hacer cosas como reducir las horas de trabajo si estamos seguros de que no se romperá nada por hacerlo. Sin duda, tenemos que regular la cantidad de gases de efecto invernadero en la atmósfera antes de poder esperar una optimización del trabajo.