MÉTODO: is_pareto_efficient_simple(costs):

is_efficient = np.ones(costs.shape[0], dtype=bool)

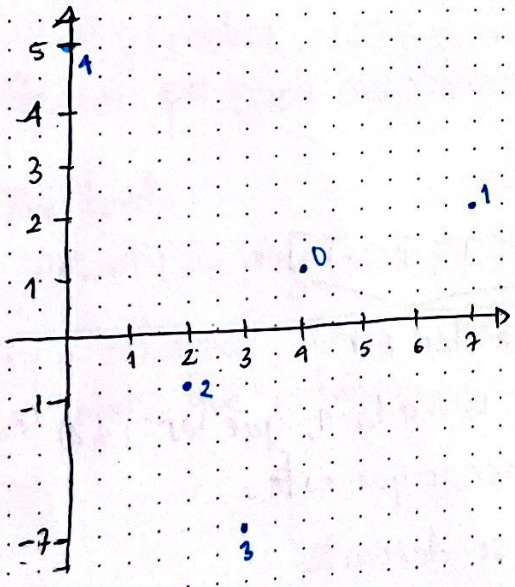
Primero se genera un arreglo booleano, de una fila, el cual todas las entradas serán True y su longitud es el número de filas del arreglo costs, que ingreso como parametro.

Este método minimiza los puntos, para el caso de que se necesite maximizar, ingresamos los valores del arreglo (costs) con signo contrario (-costs)

Ahora, vamos a considerar un ejemplo con puntos en el plano Cartesiano, para una mejor visualización.

costs = [[4 1], [7 2], [2 -1], [3 -7], [0 5]]

Entonces, antes de ingresar a for por primera vez
is_efficient = [True True True True True], una lista de longitud cinco



for i, c in enumerate(costs):
la variable i enumera cada punto c, en este caso de 0 a 4.

Vamos a comenzar a descartar todos los puntos que tengan a alguien por encima de él y lo marcara como False, por tal motivo si una entrada ya tiene un False no entrará al if.

if is_efficient[i]: El if siempre va caer cuando sea necesario comparar al punto, es decir, cuando is_effient[i] tenga el valor de True

is_efficient[is_efficient] = np.any(costs[is_effient] < c, axis=1)

Llamo toda la lista booleana, donde se guarda la información si los puntos pertenecen o no al conjunto de pareto minimizado. Esta se actualiza en cada for

En el lado derecho, esta la asignación, que se hace.
np.any: crea un arreglo donde se le asigna True si el predicado de adentro es correcto y False en caso contrario.

costs[is_efficient] < c, axis=1
Compara entrada por entrada las filas de la matriz costs, únicamente en los valores que is_effient tenga como True, esta comparación se hace con c, que es la fila del índice seleccionado

Para poder verificar la entrada i-esima, está siempre saldrá del for con el valor de True, así is_effient[i] = True

Volviendo al ejemplo:
is_efficient = [T T T T T]

En el for:
i=0
[F F T T T] ⟶ [T F T T T]
este se genera en el if — Así sale del for
Esto significa que el punto que esta en la entrada 1, que es (7,2) será descartado porque ya hay alguien menor que este.

i=1 No entra en el if, porque es False, ya se descarto

i=2
[F F F T T] ⟶ [F F T T T] así sale de for
este se genera — ya descartamos los primeros dos puntos
Y así hasta terminar

MÉTODO: is_pareto_efficient_dumb(costs):

```
is_efficient = np.ones(costs.shape[0], dtype=bool)
for i,c in enumerate(costs):
    is_efficient[i] = np.all(np.any(costs[:i]>c, axis=1)
                             and np.all(np.any(costs[i+1:]>c, axis=1))
    return is_efficient
```

Nuevamente generamos una lista de Trues, de longitud número de filas del arreglo, el cual se irá modificando, con el objetivo que cada una de las entradas de is_efficient diga True si la fila corresponde al conjunto de pareto minimizado y en caso contrario False.

Vamos a actualizar entrada por entrada con el comando is_efficient[i]

np.any(costs[:i]>c, axis=1) retornara un lista booleana, tomando como True los casos donde la desigualdad se cumpla para todos las entradas del arreglo desde 0 hasta i-1, esta comparación se hace por columnas.
Geometricamente se puede pensar como si hay una fila mayor que c, retornará True

np.any(costs[i+1:]>c, axis) hace la verificación para los valores que están enlistados luego de la fila c.

Por tanto:
np.all(np.any(costs[:i]>c, axis=1)) and np.(np.any(costs[i+1:]>c, axis=1)) va a retornar True todos, en el caso que c sea menor que todos las filas que están antes y después de c.

MÉTODO (is_pareto_efficient (costs, return_mask = True))

is_efficient = np.arange (costs.shape[0]) genera un arreglo de una longitud de número de filas del arreglo costs, y sus entradas serán: 0, 1, ..., costs.shape[0]-1

n_points = costs.shape[0] el número de puntos que hay

next_point_index = 0 es el siguiente índice en el arreglo is_efficient

while next_point_index < len(costs):

El while se moverá por todos los índices de las filas de costs

nondominated_point_mask = np.any(costs < costs[next_point_index], axis=1)

nondominated_point_mask[next_point_index] = True

Como ya lo hemos visto en los anteriores métodos, la variable nondominated_point_mask está guardado un arreglo booleano, donde se guardarán como True los puntos no dominados y Así:

is_efficient = is_efficient[nondominated_point_mask]

únicamente guardará los índices no dominados, es decir que estaremos eliminando los puntos dominados y haciendo más rápido el código.

Para terminar, se guardan la información de toda la fila en

costs = costs[nondominated_point_mask]