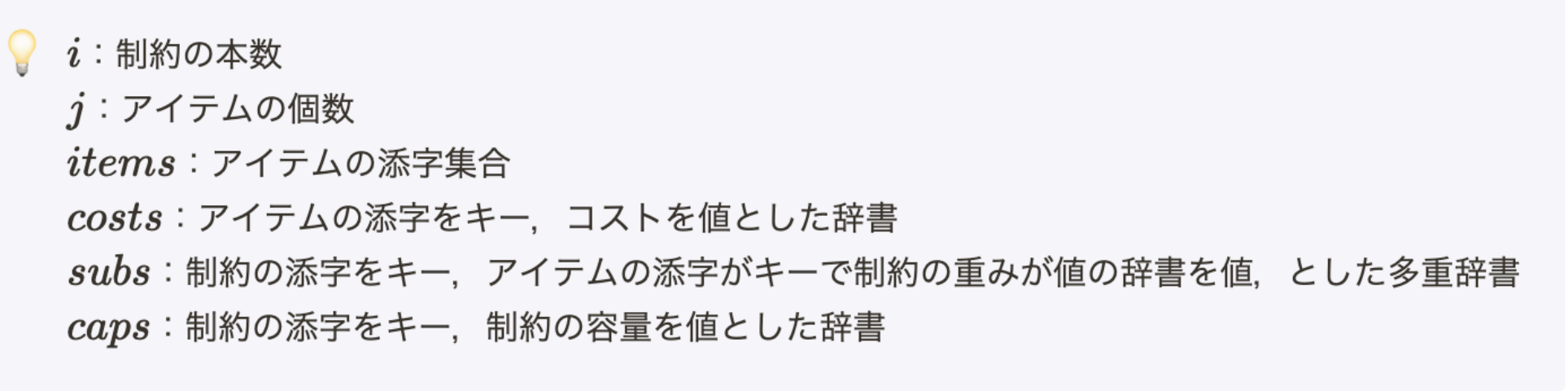
情報メディア実験
数理最適化02

情報メディア創成学類
201911419　土佐凜斗

実装したアルゴリズムの紹介

下界を貪欲法で，上界をシンプレックス法で求める分枝限定法

以下，



i：制約の本数
j：アイテムの個数
$items$：アイテムの添字集合
$costs$：アイテムの添字をキー，コストを値とした辞書
$subs$：制約の添字をキー，アイテムの添字がキーで制約の重みが値の辞書を値，とした多重辞書
$caps$：制約の添字をキー，制約の容量を値とした辞書

下界を求める方法

貪欲算法の真似事

> 💡 c：cost
> w_{ij}：j番目の品物のi番目の制約

1. 品物を価格/重量（$\frac{c_j}{w_{ij}}$）の大きい順に並べる.
2. 1の順番で品物をナップサックに入れて（$x_j = 1$として）いく
   このとき，一度入れた品物はその都度残りのリストから消去しておく（同じ品物が入らないように）
   容量オーバーなら次の品物へ．これを品物がなくなるまで繰り返す.

▶ code

上界を求める方法

線形緩和問題をシンプレックス法で解く

$0 \leq x_n \leq 1$の制約は，以下のようにして扱う

$$
\begin{aligned}
\text{maximize} \quad & 16x_1 + 19x_2 + 23x_3 + 28x_4 \\
\text{subject to} \quad & 2x_1 + 3x_2 + 4x_3 + 5x_4 \leq 7 \\
& 3000x_1 + 3500x_2 + 5100x_3 + 7200x_4 \leq 10000 \\
& x_1 \leq 1 \\
& x_2 \leq 1 \\
& x_3 \leq 1 \\
& x_4 \leq 1 \\
& x_j > 0 \ (j = 1, 2, \cdots, 4)
\end{aligned}
$$

- 行列作成

  以下の形の情報から,

```
items = {1,2,3,4}
costs = {1:16, 2:19, 3:23, 4:28}
subs = {1:{1:2, 2:3, 3:4, 4:5}, 2:{1:3000, 2:3500, 3:5100, 4:7200}}
caps = {1:7, 2:10000}
```

  シンプレックス法が使えるように以下の形に整形する

```
# 制約の係数行列
A: [[2.0e+00 3.0e+00 4.0e+00 5.0e+00]
 [3.0e+03 3.5e+03 5.1e+03 7.2e+03]
 [1.0e+00 0.0e+00 0.0e+00 0.0e+00]
 [0.0e+00 1.0e+00 0.0e+00 0.0e+00]
 [0.0e+00 0.0e+00 1.0e+00 0.0e+00]
 [0.0e+00 0.0e+00 0.0e+00 1.0e+00]]
# コストベクトル
Cost: [16 19 23 28]
# 右側ベクトル
B: [7.e+00 1.e+04 1.e+00 1.e+00 1.e+00 1.e+00]
```

自分で工夫した部分

- 貪欲法で一度入れたアイテムをもう一度入れないようにする

  一度入れたアイテムの添字をリストに入れ，あたらに比較するときはそのアイテムがリストに入っているかどうかを確認してから実行する（上の「下界を求める方法」で記載）

- シンプレックス法の$0 \leq x \leq 1$の制約

  $x_n \leq 1$の制約を加える
  （制約の係数行列の後ろに単位行列を追加して，右側ベクトルにアイテムの数だけ 1 を追加する）

  （上の「上界を求める方法」で記載）

- 入力の形からシンプレックス法で使える行列を生成する

  入力の辞書や集合の形から添字をでfor文を回して行列に追加していく
  0-1を固定したアイテムを省く必要があるので`self.zeros`や`self.ones`に入っている添字を使い行列を工夫して作成した（上の「上界を求める方法」で記載）

- 最小添字規則の実装（入る変数の場合と出る変数の場合）

  入る変数の場合：

  教科書のコードではnp.argmax()を使っていたがこれでは添字が最小になことが保証できないので，以下のようにして正かつ最小の添字のものを選んだ

```
# 候補の中で最も添字が小さいものをsに代入する
# ccは変数が入った行列
s = -100
# 添字をカウントする変数
ns = 0
for ss in cc:
    if ss > MEPS:
        s = ns
        break
    else:
        pass
    ns += 1
# エラー処理（ccに正の値が存在しない）
if s == -100:
    print("erorr")
```

出る変数の場合：

出る変数は値が正でなおかつ最小値をとるものを選ばなければならないので，入る変数のような手法ではうまくいかなかった.
そもそも考え直すとnp.argmax()は最小値をとるものが2つ以上存在するとき，添字が小さい方を返すので，元の教科書のコードで最小添字規則を満たしていたことのに気づいた.

- シンプレックス法でbb/dの時dがあまりにも小さい時0割りのエラーが出る対処

  bb/dの段階では一旦無視してその後で0割りをした要素を除外する(値をnp.infに変更)

```
import warnings

# 0割りのwarningで処理を中止しない
warnings.simplefilter('ignore', category=RuntimeWarning)
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    bb = linalg.solve(AI[:,basis], b)
    # 一旦無視して計算
    ratio = bb/d
    ratio[ratio<-MEPS] = np.inf
    # ここではじく
    ratio[d<MEPS] = np.inf

    # 出る変数（r）を選ぶ
    r = np.argmin(ratio)
```

- biを選ぶとき，最もどっちつかずな値(0.5にちかい値)をbiに選んで0-1固定する

  これによって，分枝回数を大きく減らせると考えたが，逆に実行速度が大幅に落ちてしまった.

  シンプレックス法で分数解が出る数がそれほど多くなく，biを選ぶ順番はさほど影響しないのだろうと推測した.

```
# 被約費用ベクトルが0以下（十分に小さいとき）(x_B,x_N)=(A^{-1}_Bb,0)が最適基底解
bi_vals = {}
if np.all(cc <= MEPS):
    x = np.zeros(n+m)
    x[basis] = linalg.solve(AI[:,basis], b)
    for j in self.items:
        if j in self.zeros:
            self.xub[j] = 0.0
        elif j in self.ones:
            self.xub[j] = 1.0
        else:
            self.xub[j] = x[j-1]
            # print(x[j-1])
            if x[j-1] != 0.0 and x[j-1] != 1.0:
                bi_vals[j] = abs(x[j-1]-0.50)
            self.bi = min(bi_vals.items(), key = lambda x:x[1])[0]
```