連続最適化特論　期末レポート

M1-55-17 高野昂平

課題について

課題内容として、不動点近似法の 3 つのアルゴリズム (POCS, Krasnosel'skii-Mann, Halpern) について挙動を調べた。

言語は Python を用いた。

POCS

POCS アルゴリズムを実装する上で、閉球を用いるため、以下のような簡単な閉球クラスを作成した。

Listing 1　閉球クラス

```
class ClosedBall:
    def __init__(self, center, radius):
        self.center = np.array(center)
        self.radius = radius

    def projection(self, coordinate):
        norm = np.linalg.norm(coordinate - self.center)
        if norm <= self.radius:
            return coordinate
        else:
            return self.center + (self.radius / norm) * (coordinate - self.center)
```

この閉球クラスは中心 center と半径 radius を元に作成されるクラスで、射影を行うメソッド projection を持つ。

$$P(\boldsymbol{x}) = \begin{cases} \boldsymbol{c} + \frac{r}{\|\boldsymbol{x}-\boldsymbol{c}\|}(\boldsymbol{x}-\boldsymbol{c}) & (\boldsymbol{x} \notin B) \\ \boldsymbol{x} & (\boldsymbol{x} \in B) \end{cases} \tag{1}$$

projection メソッドはこの射影 $P(\boldsymbol{x})$ に相当し、projection メソッドの取る引数 coordinate は $P(\boldsymbol{x})$ の $\boldsymbol{x}$ にあたる。

挙動を調べるにあたって、2 次元平面上の 4 つの円における不動点近似を例に見ていく。

Listing 2　4 つの円における不動点近似

```
1 def main():
2     cb1 = ClosedBall((3, 3), 5)
3     cb2 = ClosedBall((0, 0), 3)
4     cb3 = ClosedBall((-3, -3), 10)
5     cb4 = ClosedBall((2, 1), 7)
6
7     coordinate = cb4.projection(np.array((21, 5)))
8     print(coordinate)
9     coordinate = cb3.projection(coordinate)
10    print(coordinate)
11    coordinate = cb2.projection(coordinate)
12    print(coordinate)
13    coordinate = cb1.projection(coordinate)
14    print(coordinate)
```

Listing 3　出力結果

```
[8.84984849 2.44207337]
[6.08748157 1.17344926]
[2.9457696  0.56783928]
[2.9457696  0.56783928]
```

4 つの円として、cb1,cb2,cb3,cb4 を生成した (Listing1 2-5 行目)。この生成した円はそれぞれ projection メソッドを用いて、射影を返すことができ、その射影をそれぞれ P_1, P_2, P_3, P_4 とすると、不動点 T は $T = P_1P_2P_3P_4(\boldsymbol{x})$ となる。今回は挙動を調べるため、$P_4(\boldsymbol{x}), P_3P_4(\boldsymbol{x}), P_2P_3P_4(\boldsymbol{x}), P_1P_2P_3P_4(\boldsymbol{x})$ を順に出力している (Listing2 7-14 行目)。$\boldsymbol{x} = (21, 5)$ とした。

$P_4(\boldsymbol{x}), P_3P_4(\boldsymbol{x}), P_2P_3P_4(\boldsymbol{x})$ の計算においては、与えられた座標が円外にあるため、計算が行われるが、$P_1P_2P_3P_4(\boldsymbol{x})$ の計算においては与えられた座標が cb1 の円内にあるため、与えられた座標がそのまま cb1 の射影となっている (式 (1))。

Krasnosel'skii-Mann

POCS と同様に 4 つの円を生成し、どのように不動点近似が行われていくのかその挙動を見ていく。まず、非拡大写像として以下のようなメソッドを実装した。

Listing 4　非拡大写像

```
def non_expansion_map(cb_list, coordinate):
    sum = np.array((0, 0))
    for cb in cb_list[1:]:
        sum = sum + cb.projection(coordinate)
    else:
        average = sum / (len(cb_list) - 1)
    return cb_list[0].projection(average)
```

このメソッドは以下の式

$$T(\boldsymbol{x}) = P_0\left(\frac{1}{m}\sum_{i=1}^{m} P_i(\boldsymbol{x})\right)$$

をプログラム上に落とした式である。

次に、再帰的に不動点近似を行うメソッド km を以下に示す。

Listing 5　Krasnosel'skii-Mann メソッド

```
def km(coordinate, cb_list, alpha = 0.5):
    new_coordinate = alpha * coordinate + (1 - alpha) * non_expansion_map(
        cb_list, coordinate
    )
    norm = np.linalg.norm(new_coordinate - coordinate)
    print(new_coordinate, norm)
    if norm < 10**-6:
        return new_coordinate
    else:
        return km(new_coordinate, cb_list)
```

このメソッドは、以下の式をプログラム上に落としたもので、終了条件を $||T(\boldsymbol{x}) - \boldsymbol{x}|| < 10^{-6}$ としている。また、ステップ幅 α はデフォルトで 0.5 としている。

$$\boldsymbol{x}_{n+1} = \alpha\boldsymbol{x}_n - (1-\alpha)T(\boldsymbol{x}_n)$$

実行時の main 文は以下の通りで、POCS の実験の時と同様の円を 4 つ生成している。また、初期点も同様のものとした。

Listing 6　main 文

```
def main():
    cb0 = ClosedBall((3, 3), 5)
    cb1 = ClosedBall((0, 0), 3)
    cb2 = ClosedBall((-3, -3), 10)
    cb3 = ClosedBall((2, 1), 7)
    cb_list = [cb0, cb1, cb2, cb3]
    km(np.array((21, 5)), cb_list)
```

Listing 7　出力結果

```
[13.54251669  3.04986877] 7.708246829925411
[9.80634186 2.07789922] 3.8605345646038773
[7.93066159 1.58316718] 1.939828931025051
[6.81445695 1.29779227] 1.1521074738158075
[6.06084541 1.12728637] 0.7726594470796265

                ........

[2.94942643 0.54857829] 1.8142098167533117e-06
[2.94942495 0.54857802] 1.511841513954326e-06
[2.94942371 0.54857779] 1.259867928364655e-06
[2.94942268 0.54857759] 1.0498899403072627e-06
[2.94942182 0.54857743] 8.74908283890604e-07
```

出力行数が非常に長くなるため、一部割愛したが、実行結果としては 79 回の再帰にて終了条件を満たし、不動点近似を完了した。POCS での近似と近い結果となっていることがわかる。

Halpern

このアルゴリズムにおいても、Krasnosel'skii-Mann アルゴリズムと同様の非拡大写像を用いた (Listing4)。以下が Halpern アルゴリズムを実装したものである。

Listing 8　Halpern メソッド

```
def halpern(coordinate, cb_list, step, initial):
    alpha = 1 / (2**step)
    new_coordinate = alpha * initial + (1 - alpha) * non_expansion_map(
        cb_list, coordinate
    )
    next_step = step + 1
    norm = np.linalg.norm(new_coordinate - coordinate)
    print(new_coordinate, norm)
```

```
9         if norm < 10**-6:
10            return new_coordinate
11        else:
12            return halpern(new_coordinate, cb_list, next_step, initial)
```

1-4 行目は以下の式をプログラム上に起こしたものである。

$$\alpha_k = \frac{1}{2^k}$$
$$\boldsymbol{x}_{k+1} = \alpha_k \boldsymbol{x}_0 + (1 - \alpha_k) T(\boldsymbol{x}_k)$$

Halpern の挙動確認においても同様に 4 つの円と初期点を元に不動点近似を行った。円の実装については POCS にて使用した ClosedBall クラスを用いている。

Listing 9　main 文

```
def main():
    cb0 = ClosedBall((3, 3), 5)
    cb1 = ClosedBall((0, 0), 3)
    cb2 = ClosedBall((-3, -3), 10)
    cb3 = ClosedBall((2, 1), 7)
    cb_list = [cb0, cb1, cb2, cb3]
    coordinate = np.array((21, 5))
    halpern(coordinate, cb_list, 1, coordinate)
```

Listing 10　出力結果

```
[13.54251669  3.04986877] 7.708246829925411
[9.80262528 2.07944725] 3.8637424430288365
[7.92277551 1.57865309] 1.9454125408529224
[6.65232618 1.25963155] 1.3098916863739631
[5.74176409 1.07823057] 0.9284555086952968

                ........

[2.94470308 0.57338062] 3.4641723959355285e-06
[2.94470081 0.57338018] 2.3094820396481456e-06
[2.9446993  0.57337988] 1.5396715812279111e-06
[2.9446983  0.57337969] 1.0264561652676736e-06
[2.94469762 0.57337956] 6.843083314337784e-07
```

こちらも同様に、出力結果が長いので一部割愛した。実行結果としては 42 回の再帰にて終了条件を満たした。ステップ幅が Krasnosel'skii-Mann アルゴリズムと異なり、最初は大きく不動点に近づくため、再帰回数が少なくなっていると考えられる。