# 第7回：物価連動国債のキャッシュフローと名目利回り (その2)

この記事は第6回記事の続きで、前回の3-2節「物国30回債のキャッシュフロー表 概観」で示した図6-5への説明から始める。

```python
# cash flows
dfBND = mu.cpiBondCashFlow(iBndOBJ, yts=dsCvOBJ, past=1)
display( dfBND.style.format(fmtFUT) )
HCclnPRC = (dfBND.amount *dfBND.DF).sum() - accAMT
print(f'(hc) clnPRC:{HCclnPRC:.5f}')
```

✓ 0.0s Python

| | payDate | accruStart | accruEnd | cpi | coupon | amount | DF |
|---|---|---|---|---|---|---|---|
| 0 | 2025-09-01 | nan | nan | nan | nan% | nan | 1.000000 |
| 1 | 2026-03-01 | 2025-09-01 | 2026-03-01 | 112.430475 | 0.0051% | 0.0025 | 0.996538 |
| 2 | 2026-09-01 | 2026-03-01 | 2026-09-01 | 113.422284 | 0.0052% | 0.0026 | 0.986335 |
| 3 | 2027-03-01 | 2026-09-01 | 2027-03-01 | 114.428365 | 0.0052% | 0.0026 | 0.976400 |
| | | | (途中省略) | | | | |
| 18 | 2034-09-01 | 2034-03-01 | 2034-09-01 | 130.598284 | 0.0060% | 0.0030 | 0.837710 |
| 19 | 2035-03-01 | 2034-09-01 | 2035-03-01 | 131.756719 | 0.0060% | 0.0030 | 0.829272 |
| 20 | 2035-03-01 | nan | nan | nan | nan% | 120.2160 | 0.829272 |

```
(hc) clnPRC:99.73804
```

<再掲> (図6-5：物国30回債キャッシュフロー表)

## 4. 物国30回債のキャッシュフロー表の詳細

### 4-1. 将来のReference CPIの計算

図6-5で出力された `cpi列` のReference CPIの計算例を示そう。この値は図6-3で**ゼロインフレ率=1.777%**が設定され、以下が計算されている。 `(前回記事の2-3節参照)`

- index1 (2026-03-01) 112.430475
  - 基準日とCPI：b = 2025-10-01 `(= lastFixing DT)` 、基準CPI：$I_b = 112.10$
  - 計算日：t = 2025-12-01 `(= 2026-03-01 の3ヶ月ラグ)`
  - 日数：2025-10-01 → 2025-12-01 = 61日、**dcA365** で$\tau = 61/365$

- ゼロインフレ率 $z_t = 1.777\% = BEI$ で、式(6-4)より

$$I_t = 112.10\ (1 + 1.777\%)^{61/365} = 112.4304752\ldots$$

- index2 (2026-09-01) 以降もゼロインフレ率=1.777%で同様に計算

### 4-2. cpiBondCashFlow関数とas_cpi_couonのキャスト

図6-5 コードの2行目 右辺の `mu.cpiBondCashFlow` 関数は `as別名のmu` で判るように、 `myUtil` モジュールで定義した関数であり、物価連動国債用のキャッシュフローを作成する。

- この関数の元になる関数は `テキスト 4.3.4節 (p.120)` で説明した `図4.6` であり、 `myUtil` モジュールでのコードは `テキスト p.394` を参照
- `テキスト p.394` の `mu.bondCashFlow` 関数との相違点は以下の図で四角く囲った2つの部分

```
225 # 物価連動債 (past=0は過去キャッシュフローの非表示)
226 def cpiBondCashFlow(bondOBJ, ir='', yts='', past=0):
227     '''1:(ir='',yts='')=No DF      2:(ir=irOBJ, yts='')=ir DF
228        3:(ir='', yts=ytOBJ)=yt DF
229        4:( , ,past=0)=futureCF      5:( , ,past=1)=past+futureCF    '''
230     dfCPN = pd.DataFrame({
231         'payDate':     cpn.date(),           # no ISO
232         'accruStart': cpn.accrualStartDate().ISO(),
233         'accruEnd':   cpn.accrualEndDate().ISO(),
234         'cpi':        cpn.indexFixing(),
235         'coupon':     cpn.rate(),
236         'amount':     cpn.amount(),
237         } for cpn in map(ql.as_cpi_coupon, bondOBJ.cashflows()) if cpn is not None )
```
(以下 省略)

(図6-6：as_cpi_couonのキャスト)

  - `237行目 as_cpi_coupon` ：関数の第1引数のbondOBJの各cashflowオブジェクトを `as_cpi_coupon` で `cpnオブジェクト` にキャスト `(このキャストのコードが不明な場合、テキスト 2.4.2節を復習しよう)`
  - `234行目 indexFixing` ：キャストで生まれたcpnオブジェクトの `indexFixing` メソッドで、図6-5 キャッシュフロー表のcpi列に表示される `Reference CPI` を取得
  - (実は241行目の元本部分も `as_cpi_coupon` でキャストしているが、上の説明と同じとなり、省略)

- なお、これまで物価連動国債の説明は「元本の増減」と説明したが、上図 235行目では、「クーポンが増減」するコードとなっている
  - 図6-5では coupon列がindex ratioに従って 増加しているが、本来は0.005%で一定

- これはQuantLibのコーディング上の制約のためであり、「元本の増減」も「クーポンの増減」も同じキャッシュフローとなり、問題はない

#### 4-3. CashFlowsユーティリティ と 年2回複利の手計算

前回の[図6-4](https://qiita.com/retrieveram/items/2f7cee84999e1fffd7b4#3-%E7%89%A9%E5%9B%BD%E3%81%AE%E5%90%8D%E7%9B%AE%E5%88%A9%E5%9B%9E%E3%82%8A%E3%81%A8%E3%82%AA%E3%83%96%E3%82%B8%E3%82%A7%E3%82%AF%E3%83%88)で出力された`名目利回り(nominalYLD) 2.0520%` を 図6-5のpayDate列とamount列から、年2回複利として、手計算させたコードが図6-7である。

- 出力結果の2.05138%は、図6-4と0.07bpほどズレているが大目に見て頂きたい

```
1  # 年2回複利の計算
2  cfLeg = ql.Leg()
3  for iso, amt in zip(dfBND.payDate[1:], dfBND.amount[1:]):
4          cfLeg.append(ql.SimpleCashFlow(amt, iDT(iso)))
5  yldSA = ql.CashFlows.yieldRate(
6          cfLeg, clnPRC+accAMT, dcA365n, cmpdCMP, freqSA, False)
7  print(f'semi-annual yield:{yldSA:.6%}')
```

✓ 0.0s Python

```
semi-annual yield:2.051384%
```

(図6-7：年2回複利の手計算)

ここから下の記述は物価連動国債との関係は薄く、QuantLibのコーディング関係となるため、読み飛ばしても問題ない。ただ、図6-7は不規則なキャッシュフローの利回り等を計算する典型的なコードとなり、QuantLibを使いこなしたい場合、一度は理解しよう。

- `4行目` ： **SimpleCashFlow** コンストラクタに対する QLPドキュメントの説明が下図

  **ql.SimpleCashFlow**(*amount, date*)

  ```
  amount = 105
  date = ql.Date(15,6,2020)
  cf = ql.SimpleCashFlow(amount, date)
  ```

  - SimpleCashFlowクラスは `テキスト p54の最初の図` で出力された 金利スワップ 変動レグの4つのCashFlowと同じ
  - `テキスト 2.4.1節 p53` ではスワップオブジェクトの構成が「3階建てのピラミッド」と説明したが、 `SimpleCashFlow` コンストラクタはその1階部分を直接 作成する

- `2, 4行目` ：2行目のcfLeg=**ql.Leg()** は「3階建てのピラミッド」の2階部分を作成する「**コンテナ**」をcfLeg変数に設定し、4行目で各CashFlowオブジェクトをappend

- Pythonで「コンテナ `container` 」という用語はポピュラーではない。説明をすると、
    - 「コンテナ」とは複数の要素 `(オブジェクト)` をまとめて、保持する「入れ物」のこと
    - `list` や `dict` などがコンテナの例
- 下図は `Leg` に対する QLPドキュメントの説明で、最後の行の左辺 `leg` は `cf1, cf2, cf3` のコンテナとなっている

  **Leg**

  ```
  date = ql.Date().todaysDate()
  cf1 = ql.SimpleCashFlow(5.0, date+365)
  cf2 = ql.SimpleCashFlow(5.0, date+365*2)
  cf3 = ql.SimpleCashFlow(105.0, date+365*3)
  leg = ql.Leg([cf1, cf2, cf3])
  ```

- `テキスト 2.4.1節` では筆者が造語として `レグオブジェクト` と呼んだが、**レグコンテナ**が正しい呼び方 `(この点はテキストを訂正予定)`
- SimpleCashFlowクラスは `date` , `amount` のメソッドを持ち、レグコンテナの中身は次のようなコードで確認できる

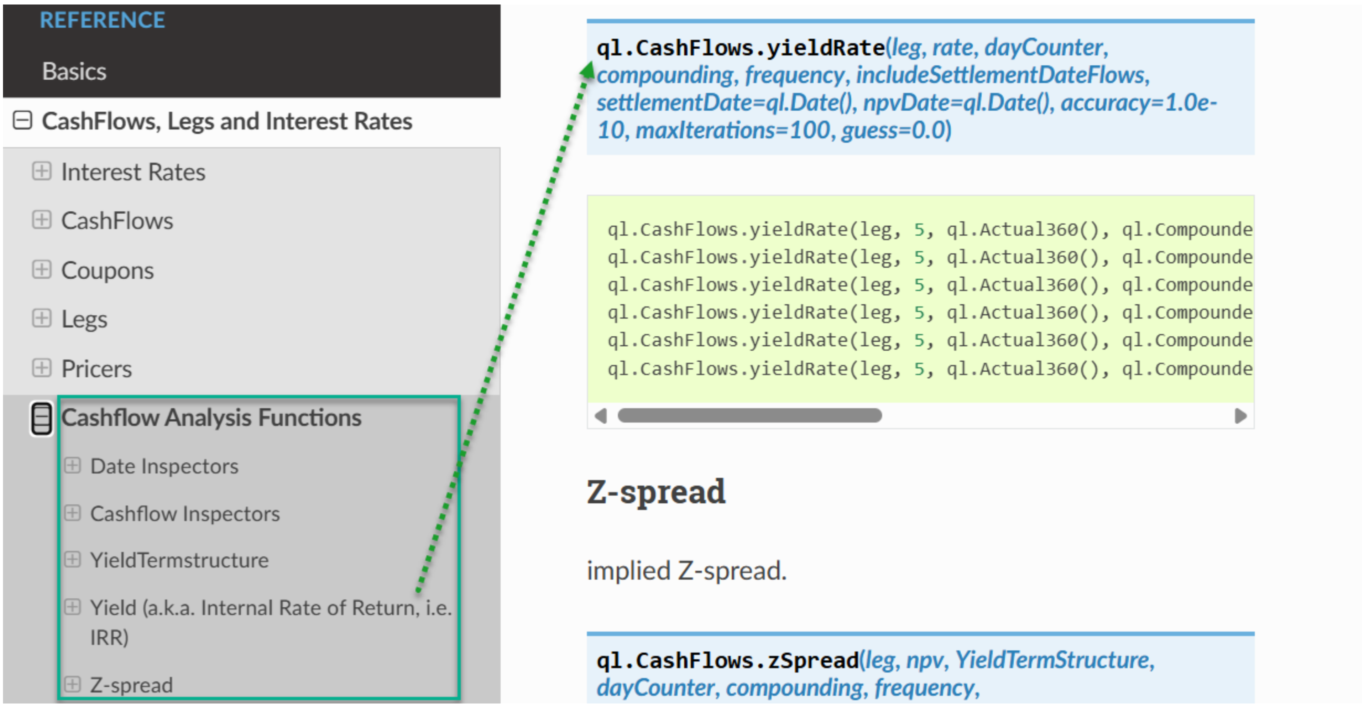
  ```python
  #リスト内包表記で最後の3つを表示
  [ (lg.date(), lg.amount()) for lg in cfLeg ][-3:]
  ```

  Python

  ```
  [(Date(1,9,2034), 0.0030034601755009336),
   (Date(1,3,2035), 0.002980697691933767),
   (Date(1,3,2035), 120.21598426031213)]
  ```

以上の説明で、2～4行目が理解できるだろう。スワップの場合、固定と変動の2つのレグをまとめるため、3階建てとなったが、ここではレグが1つなので 2 階建て。

この 2 階建てに計算機能を提供する**静的メソッドの集まり**が5行目の `CashFlows` ユーティリティ `("ユーティリティ"とは状態を持たず、処理だけを提供する関数群)`




上図は `テキスト p.6 図1.6` に掲載したQLPドキュメントで、左側の `REFERENCE` 部分も含めたスクショ。

- 左の四角く囲った部分が `CashFlows` ユーティリティの関数群のメニュー
- 右側に表示された `yieldRate` メソッドの仕様に従い、図6-7の5行目が書かれている
    - 2番目の引数は `rate` ではなく、 `npv` が正しい

      ```
      (method) def yieldRate(*args: Any) -> Any
      ```

      yieldRate(Leg arg1, Real npv, DayCounter dayCounter, Compounding compounding, Frequency frequency, bool includeSettlementDateFlows, Date settlementDate=Date(), Date npvDate=Date(), Real accuracy=1.0e-10, Size maxIterations=10000, Rate guess=0.05) -> Rate

        - 上図はVS Codeで5行目のyieldRateにカーソルを合わせて、ポップアップするスクショ `(テキスト p.7 脚注1 参照)`
        - QLPドキュメントは入門用であり、多少の誤りは大目に見よう
- 6番目の引数 `includeSettlementDateFlows` は受渡日のキャッシュフローを計算に含めるかを指定するブール変数で、図6-7の5行目では `False`

## 5. 実質価格と等しくなるインフレ率の算出

図6-4の出力を検討した3-1節で指摘したように、BEI＝1.777%をゼロインフレ率に設定した場合、実質価格=97.5728 となり、市場価格の97.55 からズレている。

図6-8はこのズレを修正し、市場価格と一致するゼロインフレ率1.77440%をBrent法で算出させたコード例となる。

```python
1  # find implied inflation  infRT:implied インフレ率
2  def infSLVR(iRT):
3      # Flat inf-curve
4      infCvOBJ = ql.ZeroInflationCurve( aStlDT,
5                              [enCpiDT,enCpiDT+pD('10y')], [iRT,iRT], freqM, dcA365)
6      infCvHDL.linkTo(infCvOBJ)
7      # pricing
8      clnPRC = iBndOBJ.cleanPrice() ;      rClnPRC = clnPRC / ixRTO
9      return rBndPRC - rClnPRC
10                                                  # accuracy   guess xMin  xMax
11 infRT = ql.Brent().solve(infSLVR,  1e-8,      0,  -0.1, 0.5)
12 print(f'real bond価格(rBndPRC):{rBndPRC:.5f}での'
13       f'インフレーション率(infRT):{infRT:.5%}'           )
```

✓ 0.0s Python

```
real bond価格(rBndPRC):97.55000でのインフレーション率(infRT):1.77440%
```

(図6-8：実質価格と等しくなるインフレ率)

このコードはこれまでの説明から理解できるので、コードの概観を記そう。なお Brent法やソルバー関数は`テキスト 7.2.5節(p.229)` を参照。

- `2~9行目` ：Brentクラスに与えるソルバー関数`infSLVR`の定義
  - `infSLVR`関数 の引数 `iRT` はゼロインフレ率用の変数。11行目のBrentクラスは市場価格97.55となる`iRT`を見つける
  - この`iRT`変数を使い、4,5行目でインフレカーブを作成　　(図6-3参照)
  - 作成された新しいインフレカーブで物国30回債の名目価格`clnPRC`と実質価格`rClnPRC`を算出　　(図6-4参照)
  - 9行目は新しいインフレカーブで計算されたrClnPRCとrBndPRC=97.55との差を戻す
- `11,12行目` ：Brent法で見つかったゼロインフレ率を`infRT`変数に設定し、出力

筆者の認識では、このような計算で見つかったインフレ率を**インプライド インフレ率**と呼び、BEIとは区別されている。この類の計算は各国の中央銀行が得意とし、彼らが詳細なレポートを発表しているので、興味のある読者はそれらのレポートを参照しよう。

最後にインフレーションで参照すべきインフレ率として、**ゼロクーポン インフレーション スワップ** `(ZC インフレ スワップ)` **レート** がある。ただ、筆者は まだ納得の行くコードが書けていないため、ZC インフレスワップに関しては、記事を書く予定が無い。

## まとめ

#### S-1. 物国の基本計算

- Base CPI (基準CPI) ： 発行時のCPI
- Reference CPI (適用指数)：補間したCPI

  `線形補間の式`

$$y(x) = (1-w)\,y_0 + w\,y_1, \quad \text{ウエイト } w := \frac{x-x_0}{x_1-x_0} \tag{6-1}$$

- Index Ratio (連動係数) ：額面の増減を表す数値

$$\text{Index Ratio (連動係数)} = \frac{\overset{(\text{適用指数})}{Ref.CPI}}{\underset{(\text{基本}CPI)}{BaseCPI}} \tag{6-2}$$

- 2つのクリーン価格： 実質クリーン価格 (Real Clean Price) と 名目クリーン価格(Clean Price)
  - 市場は実質クリーン価格で建値

$$\text{名目クリーン価格} = \text{実質クリーン価格} \times \text{Index Ratio} \tag{6-3}$$

- ゼロインフレ率 (Zero coupon inflation rate)： 式(6-4)の $z_t$
  - 基準日 $b$ と時点 $t$のCPIをそれぞれ $I_b$、$I_t$ 、[ $b, t$ ] の年数 $\tau$ で

$$I_t = I_b(1+z_t)^\tau \tag{6-4}$$

### S-2. CPIBond コーディング

- `ZeroInflationIndex`クラス：インフレカーブを持たせたCPI指数
- `ZeroInflationCurve`クラス：インフレカーブ
  - インフレカーブはCPI指数のフォワードを計算
- `laggedFixing`静的メソッド：3ヶ月ラグのReference CPIを戻す
  - Index Ratioは式(6-2)で手計算 `(QLにはBaseCPIが無いため)`
- `cpiIX.addFixing`：確定したCPI値を`cpiIX`オブジェクトに登録
  - `lastFixingDate`：フィクシングした最後日付を戻す
- `CPIBond`クラス：物国オブジェクト
- `mu.cpiBondCashFlow`関数：物国キャッシュフロー表

- `as_cpi_coupon`：クーポン情報取得のため、CPIBondクラスをキャスト
  - `indexFixing`：キャストされたオブジェクトからReference CPIを取得

### S-3. CashFlow コーディング

- `SimpleCashFlow`クラス：キャッシュフローの1階部分を作成
- `ql.Leg()`：レグ コンテナで2階部分を作成
- `CashFlows ユーティリティ`：２階部分に計算機能を提供する静的メソッド群
  - `yieldRate` メソッド等

上記コードはテキストのサポートページ より、取得可能。`(ファイル名はinfBond-JP.ipynb, myABBR.py, myUtil.py)`