第5回：Break-even Inflation(BEI)の定義と計算例

`(財務省が2026年1月22日に発表したBEI値での検証を追記)`

今後3回の投稿でQuantLibを使った物価連動国債の分析例を紹介する。今回は通常国債の名目利回りと物価連動国債の実質利回りの差である**Break-even Inflation**(以下`BEI`)の計算法を説明する。

1. BEIの定義

この`利回りの差`であるBEIの別の表現は「通常国債と物価連動国債(以下`物国`)の投資利回りが等しくなるような**将来のインフレ率**」で、直感的には市場が織り込んだ**期待インフレ率**である

- BEIの理想的な計算はゼロレートz、同一満期Tの債券に対し、
  - 名目利回り$z_n(T)$、実質利回り$z_r(T)$とすると、次式で算出する

$$\mathrm{BEI}(T) = z_n(T) - z_r(T)$$

  - ただ、この定義ではゼロレートの算出に補間やブートストラップが必要
- 実務的にはゼロレートを債券満期までの複利利回りYTMで置換えた次式が使われる

$$\mathrm{BEI}(T) = \mathrm{YTM}_n(T) - \mathrm{YTM}_r(T) \tag{5-1}$$

- 注意点として、BEI と期待インフレ率は異なり、両者の関係は一般的に次式となる ```math \begin{aligned} \text{BEI} &= \text{期待インフレ率} \ &\qquad \qquad
- \underbrace{\text{インフレ リスクプレミアム}}_{\text{Risk Premium}}
- \underbrace{\text{流動性プレミアム}}_{\text{Liquidity Premium}} \end{aligned} ```
- 右辺の2，3項目の計測はかなり困難で、筆者は`BEI ≒ 期待インフレ率`と見做している。ただし次の点を忘れないように！
  - 物価連動債は流動性が低く、ショック時に BEI は急低下する
  - この現象は、期待インフレ率の低下ではなく、流動性プレミアムの変動となり、`BEI ≠ 期待インフレ率`となる

2. 物国30回債のBEI計算例

- では 計算例を示そう。下図はJSDAが発表した2026年1月6日引けのマーケットデータ。(左上の2026年1月7日とは翌日に発表する日付を意味)
  - 867行目の物国30回債のクーポンが `*` 印で表示されているが、正しくは0.005%
  - 164行目がベンチマークの378回債

	A	B	C	D	E	F	G	H	I
25	**2026年1月7日 (水)　発表**								
26									
27	銘柄種別 Issue Type	銘柄コード Code	銘柄名 Issues	償還期日 Due Date	利率 Coupon Rate	平均値 Average			
28						単価 Price(Yen)	前日比(銭) Change(0.01Yen)	複利利回り(%) Compound Yield	単利利回り(%) Simple Yield
64									
164	02	003780067	長期国債 378	2035/03/20	1.4	94.56	-10	2.051	2.105
857									
867	05	000300083	物価連動国債 30	2035/03/10	*	97.55	+25	———	———

(図5-1：JSDA 2026年1月6日引けデータ)

- 第2回記事 JGBクラスの日本式複利メソッドの実装 で伝えたように、JSDAはT+0で日本式複利を計算し、2.051%

- 下のセルでは、第2回記事の復習を兼ね、T+0で378回債の価格94.56から日本式と欧米式複利を計算した

```
1  from myABBR import * ; import myUtil as mu
2
3  jb378 = mu.makeJGB(jDT(2025,9,20),jDT(2035,3,20),0.014)
4  trdDT = jDT(2026,1,5)
5  setEvDT(trdDT) ; print(f'EvDT({getEvDT().ISO()}', end=') ')
6  print(f'jpnCmpYld: {jb378.JapanCompoundYield(94.56):.5%}', end=', ')
7  print( 'stdCmpYld: {:.5%}'.format(
8                  jb378.bondYield(cP(94.56),dcA365n, cmpdCMP, freqSA)))
```

✓ 1.3s　Python

```
EvDT(2026-01-05) jpnCmpYld: 2.05190%, stdCmpYld: 2.05179%
```

  - 3行目のmakeJGB関数は第2回 4節の説明を参照
  - この関数の引数は最低3つ必要となり、`(378回債の前回利払日、満期日、クーポン)` で構成
    - 5行目 `getEvDT関数` はQuantLibのシステム日付を取得する関数として、新たにmyABBRモジュールに追加 `(これでsetter, getterが揃った)`
    - 第2回記事でも記したが、日本式複利と欧米式複利の差は大きくない。問題がない場合、筆者の投稿はQuantLib標準の複利メソッドを使用する

- T+0は邪魔なので、次のセルは通常のT+1で複利を算出させた

  - 2行目のsetEvDT関数で、取引日 `(trdDTはtrade dateの略)` を1月6日にセットし直し、複利利回り 2.052%を算出

```python
trdDT = jDT(2026,1,6) ;          stlDT = calJP.advance(trdDT,Tp1,DD)
setEvDT(trdDT) ; print(f'EvDT({getEvDT().ISO()}', end=') ')
print(f'jpnCmpYld: {jb378.JapanCompoundYield(94.56):.5%}', end=', ')
print( 'stdCmpYld: {:.5%}'.format(
                    jb378.bondYield(cP(94.56),dcA365n, cmpdCMP, freqSA)))
```

✓ 0.0s Python

```
EvDT(2026-01-06) jpnCmpYld: 2.05209%, stdCmpYld: 2.05197%
```

- これらを前提に次のセルで **BEI 1.7770%** を算出
  - 上で算出した各複利利回りは第2回記事の復習であり、本題からはズレている
  - BEIの計算は下のたった10行のコードのみで完了する

```python
# BEI計算: 初期値  (effDT:前回利払日   nmlYLD: ベンチマーク国債利回り)
effDT,               matDT,       cpnRT,   rBndPRC,  nmlYLD         =\
jDT(2025,9,10), jDT(2035,3,10), 0.005/100,  97.55,  2.052/100

# real bond object
rBndOBJ = mu.makeJGB(effDT,matDT,cpnRT)
# real yield
realYLD = rD(rBndOBJ.bondYield(cP(rBndPRC), dcA365n, cmpdCMP, freqSA),5)
beiRT   = nmlYLD -realYLD
print(f'real YLD:{realYLD:.5%}, (BEI)Break-even Infl.: {beiRT:.5%}')
```

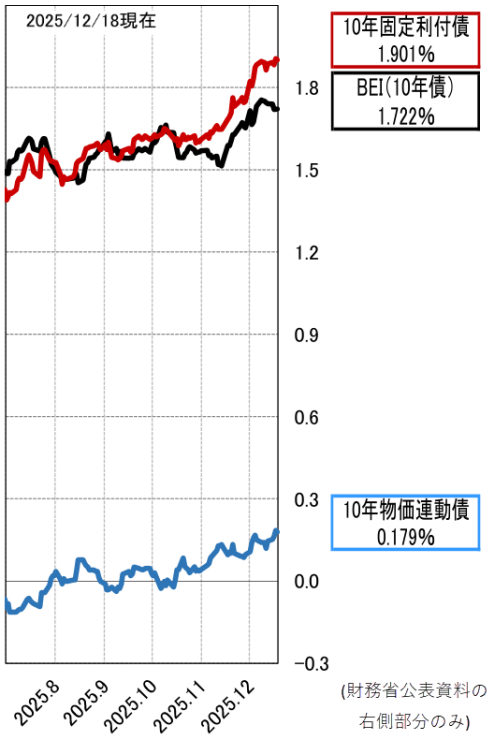
✓ 0.0s Python

```
real YLD:0.27500%, (BEI)Break-even Infl.: 1.77700%
```

(図5-2：実質利回り債オブジェクトとBEI算出)

  - 日本の財務省は「ブレーク・イーブン・インフレ率の推移」を物価連動国債のWebで発表しているが、2025/12/18現在で1.722%と公表し、ほぼ同じ水準 (注：評価日が異な

る）



- では上のコードを概観する
  - 2,3行目は初期値の設定
    - `effDT`, `matDT` は物国30回債の前回利払日と満期日
    - `rBndPRC`, `nmlYld` は それぞれreal yield bond price `（実質利回り債の価格）`, nominal yieldの略
  - 6行目は `makeJGB関数` を使い、**インフレで元本が増減しない実質利回り債**のオブジェクト `rBndOBJ` を物国30回債の条件から設定
    - `rBndOBJ` とはクーポン0.005%、償還100円の国債
  - 8行目はJSDAの価格 97.55からYTM利回り 0.2750% を算出
    - `cP(...)関数` はクリーンプライスを意味し、myABBRモジュールで定義
    - この97.55は実質利回り債の価格なので、**Real clean price** `（実質価格）` と呼ばれる
  - 9行目は YTMベースの式(1)を使い、BEI 1.777%を算出
    - このBEI値は第6回記事で利用するので、変数 `beiRT` に保存

QuantLibを使えば、BEIは驚くほど簡単に計算できることが判ったと思う。

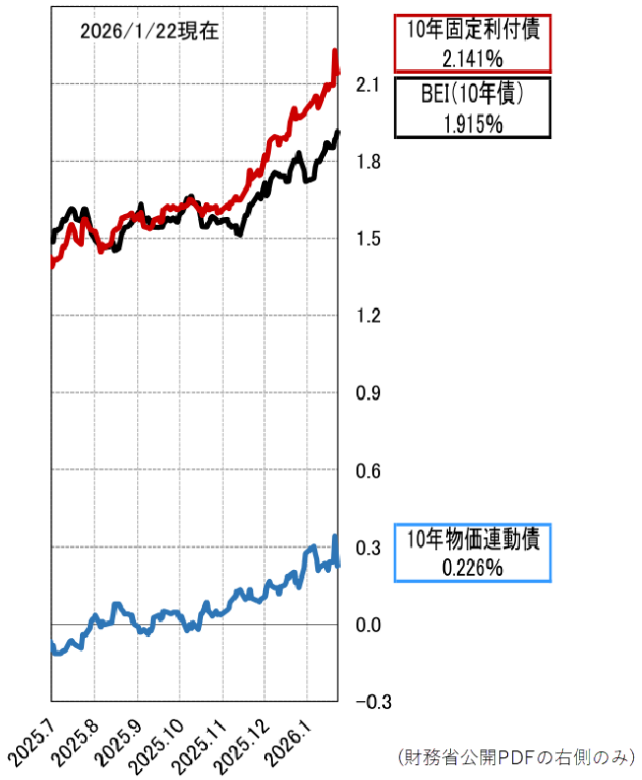
まとめ

- BEIを定義し、YTMベースのBEI算出のコード例を示した
- そのコードでは**実質利回り債**を第2回記事で作成したmakeJGB関数で作成

- 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)`
  - `(ファイル名はinfBond-JP.ipynbへ修正し、第6回コードが含まれる`

追記

- 財務省のWeb ブレーク・イーブン・インフレ率の推移は2026年1月22日付のBEI値を下図ように**1.915%**と発表



(財務省公開PDFの右側のみ)

- 次図がJSDAデータ(T+1ベース)で算出したBEI値で**1.916%**

- 0.1bp ズレている理由は、財務省の価格ソースが日本相互証券で異なるため

	A	B	C	D	E	F	G	H	I
25	2026年1月23日 (金)　発表								
26									
27	銘柄種別 Issue Type	銘柄コード Code	銘柄名 Issues	償還期日 Due Date	利率 Coupon Rate	平均値 Average			
28						単価 Price(Yen)	前日比(銭) Change(0.01Yen)	複利利回り(%) Compound Yield	単利利回り(%) Simple Yield
62									
164	02	003780067	長期国債 378	2035/03/20	1.4	93.86	+32	2.142	2.205
864									
874	05	000300083	物価連動国債 30	2035/03/10	*	98.00	+60	———	———

```python
from myABBR import * ; import myUtil as mu

jb378 = mu.makeJGB(jDT(2025,9,20),jDT(2035,3,20),0.014)

trdDT = jDT(2026,1,22) ;          stlDT = calJP.advance(trdDT,Tp1,DD)
setEvDT(trdDT) ; print(f'EvDT({getEvDT().ISO()}', end=') ')
print(f'jpnCmpYld: {jb378.JapanCompoundYield(93.86):.5%}', end=', ')
print( 'stdCmpYld: {:.5%}'.format(
                    jb378.bondYield(cP(93.86),dcA365n, cmpdCMP, freqSA)))
```

Python

```
EvDT(2026-01-22) jpnCmpYld: 2.14228%, stdCmpYld: 2.14218%
```

```python
# BEI計算: 初期値  (effDT:前回利払日    nmlYLD: ベンチマーク国債利回り)
effDT,                matDT,        cpnRT,    rBndPRC,  nmlYLD          =\
jDT(2025,9,10), jDT(2035,3,10), 0.005/100,  98.00,  2.142/100

# real bond object
rBndOBJ = mu.makeJGB(effDT,matDT,cpnRT)
# real yield
realYLD = rD(rBndOBJ.bondYield(cP(rBndPRC), dcA365n, cmpdCMP, freqSA),5)
beiRT   = nmlYLD -realYLD
print(f'real YLD:{realYLD:.5%}, (BEI)Break-even Infl.: {beiRT:.5%}')
```

Python

```
real YLD:0.22600%, (BEI)Break-even Infl.: 1.91600%
```

第6回：物価連動国債のキャッシュフローと名目利回り (その1)

(1月26日 2-3節にゼロインフレ率の説明を追加)

第5回記事ではYTMベースの**Break-even Inflation** (以下 `BEI`)を算出した。今回はBEIでインフレが発生した場合 (つまり 前回算出した1.777%で、インフレが発生した場合) 、物価連動国債 (以下 `物国`) の名目利回りはベンチマーク国債の利回り 2.052%になるかを確認する。

この確認作業は、インフレ後の物国のキャッシュフローの作成と利回り算出となるが、この2つの作業は物国を理解する王道である。

かなり長い記事となり、我慢が必要となることを最初に断っておく。

1. Reference CPI (適用指数)、 Index Ratio (連動係数)

初めに物国のキャッシュフロー作成の下準備をする。

- 物国キャッシュフローの基礎となる**生鮮食品を除く全国消費者物価指数** (以下 `CPI`) は総務省から、前月分のデータが**毎月20日前後**に発表され、CPIの数値は 政府統計ポータルサイト e-Stat で確認できる
- 財務省は発表されたCPIを基に、 物価連動国債（１０年）の適用指数及び連動係数 という次のExcelを公表

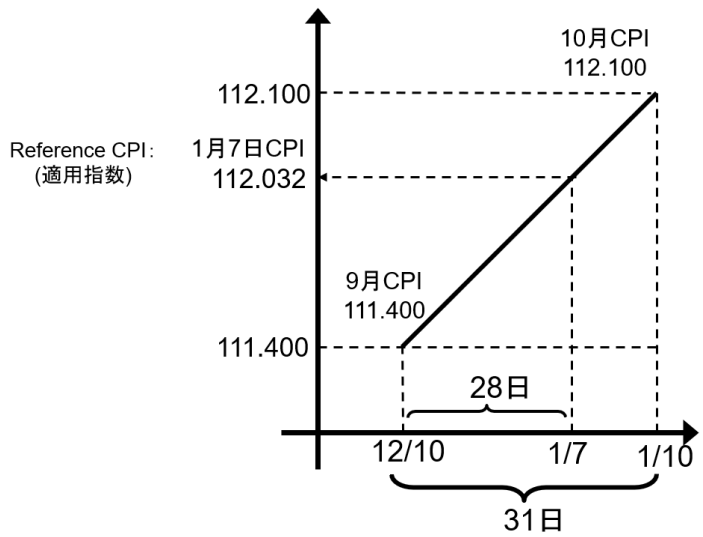
	A	B	G	H	I	J	K	L
1	令和7年12月10日～令和8年1月10日の適用指数・連動係数							
2			連動係数					
3		基準CPI／適用指数	第２５回	第２６回	第２７回	第２８回	第２９回	第３０回
4			102.2	101.1	100	104.1	106.4	109.6
5	2025/12/10	111.400	1.11412	1.12624	1.11400	1.07012	1.04699	1.01642
6	2025/12/11	111.423	1.11435	1.12648	1.11423	1.07035	1.04721	1.01663
7	2025/12/12	111.445	1.11457	1.12670	1.11445	1.07056	1.04742	1.01683
	(途中省略)							
29	2026/1/3	111.942	1.11954	1.13172	1.11942	1.07533	1.05209	1.02137
30	2026/1/4	111.965	1.11977	1.13195	1.11965	1.07555	1.05230	1.02158
31	2026/1/5	111.987	1.11999	1.13218	1.11987	1.07576	1.05251	1.02178
32	2026/1/6	112.010	1.12022	1.13241	1.12010	1.07598	1.05273	1.02199
33	2026/1/7	112.032	1.12044	1.13263	1.12032	1.07620	1.05293	1.02219
34	2026/1/8	112.055	1.12067	1.13286	1.12055	1.07642	1.05315	1.02240
35	2026/1/9	112.077	1.12089	1.13309	1.12077	1.07663	1.05336	1.02260
36	2026/1/10	112.100	1.12112	1.13332	1.12100	1.07685	1.05357	1.02281
37							(注:表示をコンパクトに編集した)	

まず、このExcelについて 説明するが、物国の取引日 / 受渡日は第5回記事同様 2026年1月6日 / 7日とする。重要な点は、**受渡日のCPI**がB33セルの112.032と算出されている点である。

1-1. Reference CPI (適用指数) `B33セル：112.032`

- 物国が参照するCPIは3ヶ月前のデータ。このことを「**3ヶ月ラグ**のCPI」などと言う

- `B5セル 111.40`：2025/12/10 `(A5セル)` の3ヶ月前の2025年9月10日のCPIが111.40を意味
- `B36セル 112.10`：同様に2026/1/10 `(A36セル)` の3ヶ月前の2025年10月10日のCPIが112.10
    - 本来のCPIの解釈にしたがえば、9月CPIが111.40、10月CPIが112.10であり、9月10日や10月10日という**各月10日**のCPIではない
    - たぶん、この `10日` は物国の利払日が10日なため、適当に財務省が決めたものか？
    - 欧米の標準仕様は月初の `1日` で、この違いがあとあと コーディングに影響
- `B6〜B35セル`：2025/12/11〜2026/1/9の各日付けのCPIは111.40と112.10を単純に線形補間した値
    - これら補間したCPIを **Reference CPI (適用指数)**と呼ぶ
    - 下は2026年1月7日のReference CPI計算のイメージ図



(図6-0：Reference CPI `適用指数`)

- この線形補間は次式のように 初めにウエイト w を算出し、

$$w = \frac{1/7 - 12/10}{1/10 - 12/10} = \frac{28日}{31日}$$

    - この w を利用して、

$$(1-w) \times 111.400 + w \times 112.100 = 112.032$$

        - `テキスト p.11 脚注4 式(1.3)` に線形補間の原型の式を記したが、その右辺が上のウエイトに対応
        - このウエイト算出で注意すべき点は2025/12/10〜2026/1/10を使うこと。3ヶ月ラグの期間 2025/9/10〜10/10ではない `(ここはよく間違える！)`

- 線形補間の式は使わないとすぐに忘れるので、メモしておく

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

- 同値な形： $y(x) = y_0 + \frac{x-x_0}{x_1-x_0}(y_1 - y_0)$ （テキスト p.11 脚注4 と同じ式）
- ウエイトは12/10からの経過日数に対応し、12/10では $w = 0$ 、1/10では $w = 1$

1-2. Base CPI (基準CPI) `L4セル：109.6`

- `G4～L4セル：102.2, ... , 109.6` 物国25回債～30回債が発行された時のCPI
  - CPIの改定があれば、このBase CPIの値は改定されるが、それ以外は不変
  - 図6-1の2行目では、baseCPIとせず、`発行時CPI` としての**issCPI**という変数名を使用 `(baseという用語が他の状況で使われるため)`

1-3. Index Ratio (連動係数) `L33セル：1.02219`

- Index Ratioは次式のように、Reference CPI ÷ Base CPIで算出
  - 発生したインフレの結果として、物国額面の増減を表す数値

$$\text{Index Ratio (連動係数)} = \frac{\overset{\text{(適用指数)}}{Ref.CPI}}{\underset{\text{(基本}CPI\text{)}}{BaseCPI}} = \frac{112.032}{109.6} = 1.02219 \qquad (6\text{-}2)$$

- 上式では インフレーションにより、額面が1.0から1.02219に増加

1-4. 2つのクリーン価格： `Real clean price` と `Clean price`

- 第5回 2節では市場で取引されている価格 `97.55` を**実質価格 (Real Clean Price)**と呼んだ
- 物国の**Clean Price** `（クリーン価格）` はIndex Ratioを使い、次式で算出する

$$\text{クリーン価格} = \text{Real Clean Price} \times \text{Index Ratio} \qquad (6\text{-}3)$$

- つまり、物国は2つの利回り(名目と実質)に対応し、2つのクリーン価格を持つ
- `実質価格` に対応し、クリーン価格を**名目価格**とも言う
- このクリーン価格に経過利息を加算したものがダーティー価格であり、受渡代金となる `(ダーティー価格は1つのみ)`

2. CPI指数オブジェクト、 インフレカーブ オブジェクト

Reference CPI や Base CPI は大した計算を行っていないが、ここまでの部分をQuantLibで計算するには、**インフレカーブを持ったCPI指数**のオブジェクト`(図6-1 10行目のcpiIX)`を作成し、そのメソッドで計算させる。

- インフレカーブとCPI指数の関係は、スワップカーブとIborの関係に対応
  - Iborのフォワードレートを計算する際、スワップカーブが必要と同様に、
  - CPI指数のフォワードを計算させるのに、インフレカーブが必要となる
    - この回の目標であるインフレ後の物国キャッシュフローを作成するには、CPI指数のフォワードの計算が必須

- 図6-1がコードと出力例 `(見慣れないキャメルケースは"myABBRモジュールで定義"と考えよう)`

```
# CPIデータ、日付の調整
issCPI,       stCPI,          enCPI,          =\
109.60,  (2025,9,111.40), (2025,10,112.10)

aTrdDT,       aStlDT                          =\
trdDT-ds9,  stlDT-ds9;     setEvDT(aTrdDT)

# CPI指数オブジェクト設定と関連数値の計算
infCvHDL = ql.RelinkableZeroInflationTermStructureHandle()
cpiIX    = ql.ZeroInflationIndex("Japan CPI",
                                 jpRegion, reviseF, freqM, lag0M, jpyFX, infCvHDL)
for yy, mm, xx in [stCPI, enCPI]: cpiIX.addFixing( jDT(yy,mm,1), xx )
enCpiDT = cpiIX.lastFixingDate()
refCPI  = round( ql.CPI.laggedFixing(cpiIX, aStlDT, lag3M, cpiLNR ),3)
ixRTO   = round( refCPI / issCPI , 5)
print(f'lastFixing DT:{enCpiDT.ISO()}, '
                                   f'refCPI:{refCPI:.8f}, ixRatio:{ixRTO:.8f}')
```

✓ 0.0s Python

```
lastFixing DT:2025-10-01, refCPI:112.03200000, ixRatio:1.02219000
```

(図6-1：CPIオブジェクトコード)

  - インフレ関連で新たに追加した`myABBRモジュール`の短縮形

```
# bond, CPI, クリーン価格等
parPR    = 100.0
parAMT   = 100.0
cpiLNR   = ql.CPI.Linear
cpiFLT   = ql.CPI.Flat
ds9      = 9
lag3M    = ql.Period('3M')
lag0M    = ql.Period('0M')
gwOLY    = False
reviseF  = False
jpRegion = ql.CustomRegion("Japan", "JP")
usRegion = ql.CustomRegion("USA",   "US")

def cP(prc): return ql.BondPrice(prc, ql.BondPrice.Clean)
def dP(prc): return ql.BondPrice(prc, ql.BondPrice.Dirty)
```

(図6-2：`myABBR`でのインフレ関連の短縮形)

2-1. Reference CPI（適用指数）、Index Ratio（連動係数）の算出

まず、図6-1の出力結果が第1節のReference CPI、Index Ratioと一致している点を確認しよう。それぞれの数値は14行、15行目で計算されている。

- `14行目 refCPI`：CPIクラス（C++の構造体）の静的メソッド`laggedFixing`で3ヶ月ラグでの2026年1月7日のReference CPIを算出（静的メソッドはテキスト p.7を参照）
  - 14行目 右辺先頭の`round`はCPI指数を少数3桁へラウンド（財務省ルール）
- `15行目 ixRTO`：Index Ratioを式(6-2)より、`refCPI`÷`issCPI`で算出
  - 同様に先頭の`round`でindex Ratioを少数5桁へラウンド（財務省ルール）

14行目の**laggedFixing**メソッドは最近追加されたようで、ドキュメントが間に合っていないが、**QuantLib.pyファイル**では次のようなコードである（QuantLib.pyファイルはテキスト p.7を参照）

```
23730    @staticmethod
23731    def laggedFixing(index, date, observationLag, interpolationType):
23732        r"""laggedFixing(ext::shared_ptr< ZeroInflationIndex > const & index,
         Date date, Period observationLag, CPI::InterpolationType
         interpolationType) -> Real"""
23733        return _QuantLib.CPI_laggedFixing(index, date, observationLag,
         interpolationType)
```

- 引数は4つで、（CPIインデックス、ターゲットの日付、3ヶ月ラグ、補間方法）
- 図6-1の14行目で与えた4つの変数から、これらの引数の内容を推測しよう。以下が簡単な説明（ds9, lag3M, cpiLNRはmyABBRの短縮形、図6-2参照）
  - `cpiIX`：10行目で作成されたCPI指数のオブジェクト
  - `aStlDT`：adjusted settle dateの略で5行目で作成。
    - 本来の受渡日（`stlDT`）2026年1月7日から9日（`ds9`）を引いた日付（dsはdaysの略）
  - `lag3M`：ラグ3ヶ月を意味する短縮形
    - ラグ0ヶ月の`lag0M`もmyABBRでは定義
  - `cpiLNR`：CPIの補間を線形に指定する短縮形（LNRはlinearの略）
    - １ヶ月単位でCPIが更新されるため、月内を線形補間させる
    - 補間計算を行わない`cpiFLT`もmyABBRで定義

2-2. ZeroInflationIndexクラスのCPI指数オブジェクト

では図6-1のコードを1行目から見て行く

- `2,3行目 初期値設定`：3ヶ月ラグで参照する9月と10月のCPIを`stCPI, enCPI`に設定`(st, en はstard, endの略)`
    - Quantlibでは、CPIデータは月初～月末とするため、`年、月`という日付で設定
- `5,6行目 ds9による調整`：毎月10日という日本のCPIの日付を欧米の標準仕様に修正するため、取引日と受渡日から9日`(ds9)`を引いて、`aTrdDT, aStlDT`に設定`(aはadjustedの略)`
    - `setEvDT(aTrdDT)`によって、QuantLibのシステム日付も9日前に修正
    - この修正法は、図6-0で示した「Reference CPI計算のイメージ図」の日付を9日ずらしても同じ計算となり、結果は正しい
        - 国債の場合、利払い日・償還日が曜日に関係なく、固定されているため、このようなシフトが可能
- `9,10,11行目 CPI指数オブジェクトの設定`：**ZeroInflationIndexコンストラクタ**によって、日本のCPI指数オブジェクト`cpiIX`を作成
    - コンストラクタの引数に関しては、QLPドキュメントの説明`(下図)`と10,11行目を見較べて、推測しよう`(QLPドキュメントはテキスト p.6を参照)`

    class **ZeroInflationIndex**(*familyName: str, region: ql.Region, revised: bool, frequency: ql.Frequency, availabilityLag: ql.Period, currency: ql.Currency, h: ql.ZeroInflationTermStructureHandle | None*)

    Base class for zero inflation indices.

    Parameters:
    - **familyName** (*str*) – The name of the zero inflation index family (e.g., “CPI”, “HICP”).
    - **region** (*ql.Region*) – The geographical region for which the index is published.
    - **revised** (*bool*) – Whether the index can be revised after publication.
    - **frequency** (*ql.Frequency*) – The frequency with which the index is published (e.g., Monthly, Quarterly).
    - **availabilityLag** (*ql.Period*) – The lag between the reference period and the publication date.
    - **currency** (*ql.Currency*) – The currency in which the index is quoted.
    - **h** (*Optional[ql.ZeroInflationTermStructureHandle]*) – (Optional) The zero inflation term structure handle used for forecasting.

    (若干の補足説明)

    - `第2引数 jpRegion`：myABBRモジュールで、CustomRegionコンストラクタを使い、`ql.CustomRegion("Japan", "JP")`と定義
    - `第3引数 revisedF`：myABBRモジュールで`False`を指定し、CPI指数が改定がされていないと設定

        - `第5引数 lag0M`：インデックス自体にラグは無いので、myABBRモジュールで、`ql.Period('0M')`を設定
            - QLPドキュメントの説明とは異なる。ここを`lag3M`にするとエラーで身動きできなくなる
        - `第7引数 infCvHDL`：9行目で設定した空のインフレカーブ用ハンドル
            - インフレカーブの設定は2-3節で行う
            - 空のハンドルの設定はTiborと同じで、`テキスト p.43の図2.3 7,8行目`を参照

- `12行目 cpiIX.addFixing`：9月と10月に確定したCPI値を`cpiIX`オブジェクトに登録
    - addFixingメソッドはTiborと同じで、`テキスト p.59 2.4.4節`を参照
    - `jDT(yy,mm,1)`と毎月1日を指定　`(ここを10日にしても、1日でフィクシングされる)`
- `13行目 lastFixingDate`：最後にフィクシングした日付を戻すメソッド
    - 12行目で9月と10月のCPI値をフィクシングさせたので、最後にフィクシングした日付は2025年10月1日

2-3. ゼロインフレ率 と インフレカーブ 設定

直前に説明した13行目`lastFixingDateメソッド`が意味することは「`cpiIX`オブジェクトは10月1日以降のインフレデータを持っていない」ことである。インフレカーブの設定は、その日付以降の**将来のインフレ率**を`cpiIX`に教える。

図6-3の2行目 **ZeroInflationCurve** はインフレカーブを作成する最も簡便なコンストラクタである。

- CPI指数を作成した ZeroInflationIndex やこの ZeroInflationCurve の名前の由来である**Zero coupon inflation rate**`(以下 ゼロインフレ率)`の定義は 式(6-4)の z_t となる。`(金利の場合も、ゼロクーポンレートをゼロレートと言う点は同じ)`
    - 基準日 b と時点 tのCPIをそれぞれ I_b、I_t 、[b, t] 間の年数を τ として、**ゼロインフレ率** z_t は次式より算出

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

    - I_bを左辺に移せば、式(6-2)のIndex Ratioが左辺に現れる。右辺はゼロインフレ率 z_tを使って、増減した物国額面が計算される

$$\frac{I_t}{I_b} = (1 + z_t)^\tau$$

- 図6-3では BEI 1.7770% を ゼロインフレ率に設定し、インフレカーブとするコード

- つまり、10月1日以降 物国額面は年率1.7770% で増加させる設定となる

```
# インフレカーブ                                    (BEIでフラットカーブを設定)
infCvOBJ = ql.ZeroInflationCurve( aStlDT,
                [enCpiDT,enCpiDT+pD('10y')], [beiRT,beiRT], freqM, dcA365)  #dcA365n
infCvHDL.linkTo(infCvOBJ)
# カーブ表示
dfInfCv  = pd.DataFrame( infCvOBJ.nodes(),columns=["node", "rate"] )
dfInfCv.node = dfInfCv.node.map(lambda x: x.ISO())
dfInfCv[:5].style.format({"rate": "{:.4%}"})
```

✓ 0.0s Python

	node	rate
0	2025-10-01	1.7770%
1	2035-10-01	1.7770%

(図6-3：インフレカーブ オブジェクト)

コードを簡単に説明しておく。

- `3行目` ：日付リスト `[enCpiDT,enCpiDT+pD('10y')]` とインフレ率リスト `[beiRT,beiRT]` を与えて、インフレカーブのオブジェクト `infCvOBJ` を作成
  - 日付リストでは、2025年10月1日から10年間を指定し、インフレ率は第5回記事で計算したbeiRT=1.7770%をフラットで設定
  - インフレカーブの日数計算法は `dcA365` とした　　`(dcA365nの可能性もある)`
- `5行目 infCvHDL.linkTo(infCvOBJ)` ：図6-1の9行目で作成した空のハンドル `infCvHDL` へinfCvOBJオブジェクトをリンク
  - `infCvHDL` ハンドルは `cpiIX` オブジェクト作成時の引数であった `(同図 11行目)`
  - このリンクにより、 `cpiIX` オブジェクトはフォワードCPIの算出が可能になる
- `6～8行 infCvOBJ.nodes` ： 作成された `infCvOBJ` オブジェクトは金利カーブと同様なnodesメソッドを持ち、nodes情報を出力 `(nodesの例は、テキスト 図2.3など 参照)`

3. 物国の名目利回りとオブジェクト

長い準備が終わり、図6-4の2～4行目で物国30回債用の**インフレで元本が増大する物国オブジェクト**の作成にたどり着けた。

```python
# inf bond object    >> スケジュールをds9で調整
iScdOBJ = ql.Schedule(effDT-ds9,matDT-ds9,pdFreqSA,calNL,unADJ,unADJ,dtGENb,EoMf)
iBndOBJ = ql.CPIBond (Tp1, parAMT, gwOLY, issCPI, lag3M, cpiIX,
                                                    cpiLNR, iScdOBJ, [cpnRT], dcA365n, unADJ)
# discount curve
dsCvOBJ, dsCvHDL = mu.ffTSH(aStlDT, nmlYLD, dcA365n, cmpdCMP, freqSA)
iBndOBJ.setPricingEngine(ql.DiscountingBondEngine(dsCvHDL))
# pricing
clnPRC  = iBndOBJ.cleanPrice()  ;        rClnPRC = clnPRC / ixRTO
accAMT  = iBndOBJ.accruedAmount()
infYLD  = iBndOBJ.bondYield(cP(clnPRC),  dcA365n, cmpdCMP, freqSA)
print(f'clnPRC:{clnPRC:.5f},  '      f'accAMT:{accAMT:.7f},  '
  f'nominalYLD:{infYLD:.7%},  '  f'realClnPRC:{rClnPRC:.7f} ')
```

✓ 0.0s Python

```
clnPRC:99.73804,  accAMT:0.0016663,  nominalYLD:2.0520002%,  realClnPRC:97.5728931
```

(図6-4：物国オブジェクトとプライシング)

3-1. 物国30回債のプライシング

図6-4では ベンチマーク国債の利回り`nmlYLD=2.052%`で元本が増大した物国30回債をプライシングした。ここで出力結果を確認する。

- 物国の名目金利 `nominalYLD` は 2.05200%とベンチマーク国債の利回りとほぼ一致
    - どのような計算から、名目利回り 2.052%が計算されたかは、3-2節で概説し、詳細は第7回記事で説明
- 実質価格(Real Clean Price)を表すrealClnPRC=97.5728931は第5回記事 2節の97.55と0.02円ほどのズレ
    - この実質価格は97.55に一致すべきであり、図6-3で設定したBEIのインフレカーブを修正する。この修正法は第7回記事で説明
    - BEIの1.777%はYTMベースであり、インフレカーブはゼロクーポンベースで設定しているため、この差異が発生
    - 物国30回債のクーポンが0.005%とゼロに近いため、物国の名目利回り2.052%とベンチマーク国債の利回りが近くなったが、本来は数ベーシスから数十ベーシスの差が生ずる

次にどのメソッドがこの計算結果を算出したかを確認する。`(使用されている債券のメソッドの詳細はテキスト4.2節を参照)`

- 9行目の `cleanPrice` メソッドは、名目価格(Clean Price)を戻し、99.73804円
- 実質価格は式(6-3)より、9行目 2つ目のコードで `rClnPRC=clnPRC/ixRTO` で手計算

- 9行目で計算された名目価格 `clnPRC` を11行目の `bondYield` メソッドに与え、名目利回り 2.052% を算出

3-2. CPIBond クラスとメソッド

では 図6-4のコードで残った部分を確認する。

- `2行 iScdOBJ` ： 第5回記事で使用した `mu.makeJGB関数` は使わず、 `Scheduleコンストラクタ` を手打ちし、物国30回債用のiScdOBJオブジェクトを作成 `(iはinflationの略)`
  - 注意すべきは、 `effDT` と `matDT` を `ds9` で調整させている
  - その他は `mu.makeJGB関数` のScheduleコンストラクタと同じ
- `3,4行 iBndOBJ` ：**CPIBondコンストラクタ**によって、物国30回債のiBndOBJを作成

  このコンストラクタの引数に関しては、QLPドキュメントの説明 (下図)と3, 4行目を見較べて、推測しよう

  > **ql.CPIBond**(*settlementDays, notional, growthOnly, baseCPI, contractObservationLag, inflationIndex, observationInterpolation, fixedSchedule, fixedRates, fixedDayCounter, fixedPaymentConvention*)

  - 3番目の引数 **growthOnly** は常にFalseとし、図6-2で `gwONY=False` を設定
  - その他の引数は自明であろう
- `6,7行` ：6行目でディスカウントカーブを作成し、7行目でiBndOBJオブジェクトにエンジンをセット
  - この手順は `テキスト 図4.11 (p.128)` と同じ
  - 注意すべきはディスカウントカーブの受渡日が `aStlDT` としている。つまりディスカウントカーブも9日間シフトさせている
- `テキスト 図4.11` ではNPVメソッドで利含み価格を計算させているが、エンジンを搭載した債券オブジェクトの場合、この図6-4の9,10行目のように `cleanPrice, accruedAmount, dirtyPrice` 等は引数無しで計算できる
  - `テキスト 図4.3 (p.115)` では、エンジンを持たない債券オブジェクトを使い、これらのメソッドを計算させた点と比較

3-3. 物国30回債のキャッシュフロー表 概観

図6-5は 物国30回債のiBndOBJオブジェクトを用い、2, 3行目で `mu.cpiBondCashFlow関数` から、キャッシュフロー表を出力させた。4行目ではキャッシュフロー表のamount列とDF列から、クリーン価格を手計算させ、図6-4と同じ価格 99.73804を得ている。

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

(図6-5：物国30回債キャッシュフロー表)

このキャッシュフロー表から、まず 理解すべき点を列挙する。

- 最も重要な列が、キャッシュフロー表 中央にある**cpi列**。この列はBEI 1.7770%でインフレが発生した場合の将来のCPIを式(6-4)で計算した値で、1-1節の `Reference CPI(適用指数)` となる
- 左3列の日付は9日間シフトさせたもので、本来はすべてが `10日`
  - index20の物国30回債の債券償還日はpayDate列の日付を９日間シフトさせ、2035年3月10日
  - `1日` という表示が嫌いな場合、下のコードのように データフレーム上で9日を足して表示させれば済む

```python
# 日付を10日に戻す (dfBNDのコピーでの処理)
dfBND2 = dfBND.copy()
for cc in ['payDate', 'accruStart', 'accruEnd'] :
    dfBND2[cc] = pd.to_datetime(dfBND2[cc]) + pd.Timedelta(days=ds9)
    dfBND2[cc] = dfBND2[cc].dt.strftime("%Y-%m-%d")
dfBND2.style.format(fmtFUT)
```

- index20は物国30回債の償還価格が 120.2160 と算出
  - この償還価格は `Index Ratio(連動係数)` として、 `償還日CPI 131.756719 ÷ 発行時CPI 109.60` から計算

    - 各利払い日の増大した元本額は、cpi列の各利払い日CPIから `Index Ratio` を計算すれば済む
    - 各利払い額は `増大した元本額 x クーポン(0.005%)` で計算
- 図6-4で出力された `名目利回り(nominalYLD) 2.0520%` はpayDate列とamount列から計算した年2回複利利回り `(この手計算例は次回記事で説明)`

この第6回記事はかなり長くなったので、この辺りで一旦休憩し、残りは第7回記事で説明しよう。

- 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)`

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。

```
# 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 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

```
# 年2回複利の計算
cfLeg = ql.Leg()
for iso, amt in zip(dfBND.payDate[1:], dfBND.amount[1:]):
        cfLeg.append(ql.SimpleCashFlow(amt, iDT(iso)))
yldSA = ql.CashFlows.yieldRate(
        cfLeg, clnPRC+accAMT, dcA365n, cmpdCMP, freqSA, False)
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

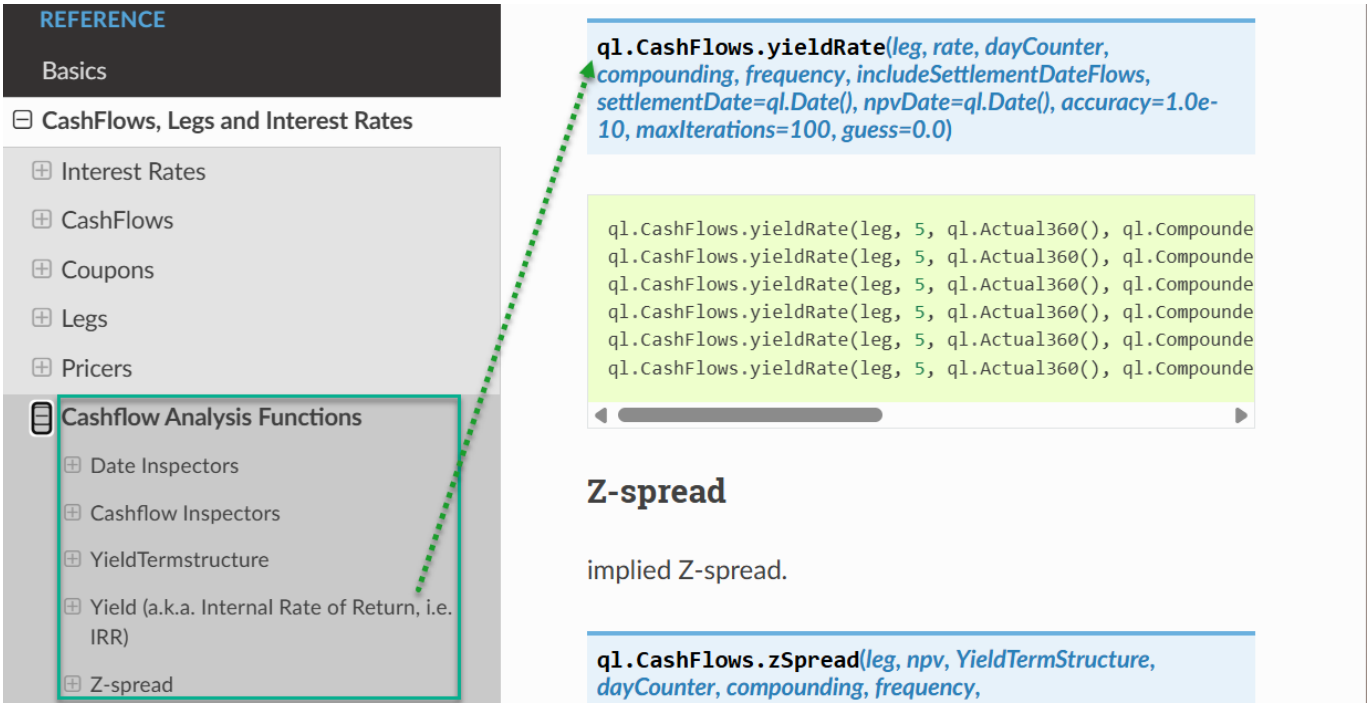
```
1  #リスト内包表記で最後の3つを表示
2  [ (lg.date(), lg.amount()) for lg in cfLeg ][-3:]
```

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

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

- 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 τ 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)`

第1章　インフレーション関連商品

この章ではインフレーションにリンクして支払額が変動する商品について説明する。インフレに連動して元本が増減する債券は物価連動債またはリンカーと言われ、インフレーション関連商品の中核に位置している。
一方、デリバティブとしてはインフレーションスワップやインフレーション キャップ・フロアー等が取引されているが、最も基本的なゼロクーポンインフレーションスワップのみを取り上げる。
また インフレーションスワップを利用して、物価連動債をアセットスワップする考え方を紹介する。

1.1　物価連動債

(Qiita 参照)

1.2　ゼロクーポン インフレーションスワップ

次にゼロクーポン インフレーションスワップについて概観しよう。まず、固定金利と CPI の変化率を交換するスワップを**インフレーション スワップ** (Inflation swap) と呼ぶ。また スワップの期間中 キャッシュフローの交換を行わず、満期時に 1 回のみの交換をするスワップを**ゼロクーポンスワップ** (Zero Coupon swap) と呼ぶ。**ゼロクーポン インフレーションスワップ** (以下 **ZC インフレ スワップ**) とはこの 2 つが組み合わさったスワップである。


SWIL

9D 保存　9Q アセット　93 更新　物価連動債/スワップ設定

1) カーブ　2) コンベクシティ&シーズナリティ　3) YoYボラティリティ　4) ZCボラティリティ　5) 債券

レート　米国　Date 08/19/22　市場サイド ミッド　CPI指数　1D CPURNSA

ゼロクーポンカーブ　タイプ 提供社　情報提供社 BGN

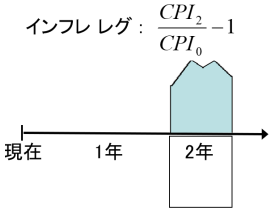
ラグ 3 月　補間 Daily　基準指数 5/23/2022 295.145

期間	ZCビッド	ZCミッド	ZCアスク	YoYビッド	YoYミッド	YoYアスク	CPI ビッド	CPIミッド	CPI アスク	Cvx(bp)
1 YR	3.07615	3.16120	3.24625	3.07615	3.16120	3.24625	304.22447	304.47549	304.72651	0.00
2 YR	3.11766	3.15800	3.19834	3.11915	3.16017	3.20122	313.83549	314.08108	314.32677	0.21
3 YR	3.02803	3.11120	3.19437	3.02723	3.10976	3.19234	322.77671	323.55899	324.34254	-0.26
4 YR	2.96234	3.03405	3.10576	2.96544	3.03721	3.10902	331.70314	332.62818	333.55515	-0.13

図 1.1: インフレーション スワップレート (SWIL 機能)　(データ出所:Bloomberg)

ZC インフレスワップは金利スワップのように市場で建値されている。図 1.1 の「ZC ビッド、ZC ミッド、ZC ミッド」列が USD 建て ZC インフレ スワップレートで、取引日 2022 年 8 月 19 日 (受渡日 8 月 23 日) の建値である[*1] 。

例えば、「2YR」行のミッド 3.1580%とは 2 年間の CPI の変化率、すなわち 2 年間のインフレーションを受け取る見返りに、固定金利 $3.1580\%_{(P.A.)}$ を 2 年分 支払うことを意味している。ただし、ゼロクーポンスワップの為、インフレーションと固定金利の交換はスワップ満期時の 1 回のみとなる。



インフレーション側のキャッシュフローは**インフレーションレグ** (またはインフレレグ) として参照される。

[*1]図 1.1 の「YOY」(Year On Year)、「Cvx」(Convexity Adjust) 列に関しては Mercurio and Zhang [3] の説明が入門的であり、よくまとまっている資料である。また YOY インフレスワップは市場での建値は無く、価格計算にはインフレーションモデル (Jarrow-Yildirim Mode 等が有名) が必要となる。モデルの導出に関しては Brigo and Mercurio [1] を参照。

ここで 現在と 2 年後の CPI をそれぞれ CPI_0 と CPI_2 で表すと、想定元本 1 ドルのインフレレグの満期時点の価値は CPI の 2 年間の変化率より、

$$\textbf{インフレレグの価値}_{(満期)} \quad = \quad \underbrace{\frac{CPI_2}{CPI_0} - 1}_{\substack{CPI \text{ 変化率 または} \\ 2 \text{ 年間のインフレ率}}} \tag{1.1}$$

と計算できる。この現在価値が ZC インフレスワップのインフレ レグの時価となる。

一方 固定金利側 (以下 固定レグ) の満期時の価値は

$$\textbf{固定レグの価値}_{(満期)} \quad = \quad (1 + \underbrace{3.158\%}_{\substack{ZC \text{ ミッド} \\ (\text{図 } 1.1)}})^2 - 1 \tag{1.2}$$

のように計算する。

すると 市場で建値されているスワップレートの受け払い額は等しいので、

$$\underbrace{\frac{CPI_2}{CPI_0} - 1}_{\text{インフレレグ: 式 (1.1)}} \quad = \quad \underbrace{(1 + 3.158\%)^2 - 1}_{\text{固定レグ: 式 (1.2)}} \tag{1.3}$$

が成立し、CPI_2 について 解く。

$$\underbrace{CPI_2}_{\text{フォワード } CPI} \quad = \quad CPI_0 \times (1 + \underbrace{3.158\%}_{ZC \text{ ミッド}})^2 \tag{1.4}$$

このように 2 年後の CPI_2 を ZC インフレスワップレートから計算することが可能になる。つまり ZC インフレ スワップを利用すれば、市場が予想している将来の CPI(以降 **プロジェクト CPI**) を簡単に計算することが出来てしまう。

図 1.2: 2 年ゼロクーポン インフレーション スワップ　(データ出所:Bloomberg)

図 1.2 は 2 年 ZC インフレ スワップを計算させた SWPM 機能 (元本交換は無い) であり、下側の画面にインフレレグのキャッシュフローを追加した。このスワップでは起算日を 8 月 23 日とし、図 **??** TIPS と同じ受渡日とさせた為、インフレレグ (レグ 1) の “ Base Index(基準金利) ” 欄に表示された 295.14535 は式 (**??**) で計算した 参照 CPI と同じ値となっていることを確認しよう。

この 8 月 23 日 CPI が式 (1.4) の CPI_0 となるので、2 年後の CPI_2 は

$$\underbrace{CPI_2}_{\text{フォワード } CPI} = \underbrace{295.14535}_{CPI_0} \times (1 + \underbrace{3.158\%}_{ZC \text{ ミッド}})^2 = 314.08108 \tag{1.5}$$

と計算される。図 1.2 インフレレグ キャッシュフローの「リセット価格」欄に表示された値である。

CPI_2 の計算結果を受けて、インフレレグから受け取ることが出来る 2 年後の金額は

インフレレグの時価

$$= \underbrace{\left[\frac{\overset{(CPI_2)}{314.08108}}{\underset{(CPI_0)}{295.14535}} - 1 \right]}_{\text{2 年間のインフレ率: 式 (1.1)}} \times \underbrace{10Mil \text{ ドル}}_{\text{想定元本}} \times \underbrace{0.93615}_{\substack{\text{ディスカウント} \\ \text{ファクター} \\ \text{(図 1.2)}}}$$

$$= \underset{\text{(図 1.2インフレレグの時価)}}{600,608.23 \text{ ドル}}$$

と計算される。

参考文献

[1] Brigo, D and Mercurio, F. (2001) Interest Rate Models: Theory and Practice Second ed., Springer-Verlag

[2] Kerkhof, Jeroen (2005) Inflation Derivatives Explained Markets,Products, and Pricing, Lehman Brothers Fixed Income Quantitative Research, July (available at http://www.scribd.com/doc/19601807/Lehman-Brothers-Kerkhof-Inflation-Derivatives-Explained-Markets-Products-And-Pricing)

[3] Mercurio, F. and Zhang, J. (2012) Pricing Inflation Derivatives, Bloomberg DOCS 2056795<GO>