

三、Interest Rate 类别

◆ 利率(Interest Rate)对象旨在封装一个利率信息，并提供与之相关的转换运算。

- 如折现因子(Discout Factor)与复利因子(Compound Factor)。
- 它也提供此对象的一些信息访问方法。
 - ✓ 存取子(Accessor)

◆ 实务上市场计算复利的方式有下面五种方式，

- 在 QuantLib 中其所对应的字符串如下表所示。

```
enum Compounding
{
    Simple,
    Compounded,
    Continuous,
    SimpleThenCompounded,
    CompoundedThenSimple
};
```

Andy M. Dong 18

◆ 令 t 表计息期间(以年为单位)， f 表计息频率(一年几次)， r 表计息利率(年利率)，则此四种复利方式求得的折现因子，可以表示如下。

- 简单复利(Simple): $DF_t = \frac{1}{(1+r \times t)}$

- 一般复利(Compounded): $DF_t = \frac{1}{\left(1 + \frac{r}{f}\right)^{t \times f}}$

- 连续复利(Continuous): $DF_t = \frac{1}{e^{r \times t}}$

- 简单再一般复利(SimpleThenCompounded): $DF_t = \begin{cases} \frac{1}{(1+r \times t)} & , t \leq \frac{1}{f} \\ \frac{1}{\left(1 + \frac{r}{f}\right)^{t \times f}} & , t > \frac{1}{f} \end{cases}$

Andy M. Dong 19

(一)InterestRate 类别

◆ 建构子

● InterestRate()

✧ 语法

```
def __init__(self, *args):
    _QuantLib.InterestRate_swiginit(self, _QuantLib.new_InterestRate(*args))
```

✧ 使用

```
>>> DC360 = ql.Actual360()
>>> QRate = ql.InterestRate(0.1, DC360, ql.Compounded, ql.Quarterly)
```

The screenshot displays a code editor with two panes. The left pane shows a list of classes and methods in the QuantLib namespace, including InstrumentVector, InterestRate, and various engines. The right pane shows the definition of the InterestRate class, including its constructor and various methods like compoundFactor, discountFactor, and impliedRate. The constructor is highlighted in blue.

```
public InterestRate(double r, QuantLib::DayCounter dc, QuantLib::Compounding comp, QuantLib::Frequency freq)
    QuantLib::InterestRate 的成員
```

● Function : compoundFactor()

✧ 计算输入的开始日到结束日之间的复利因子

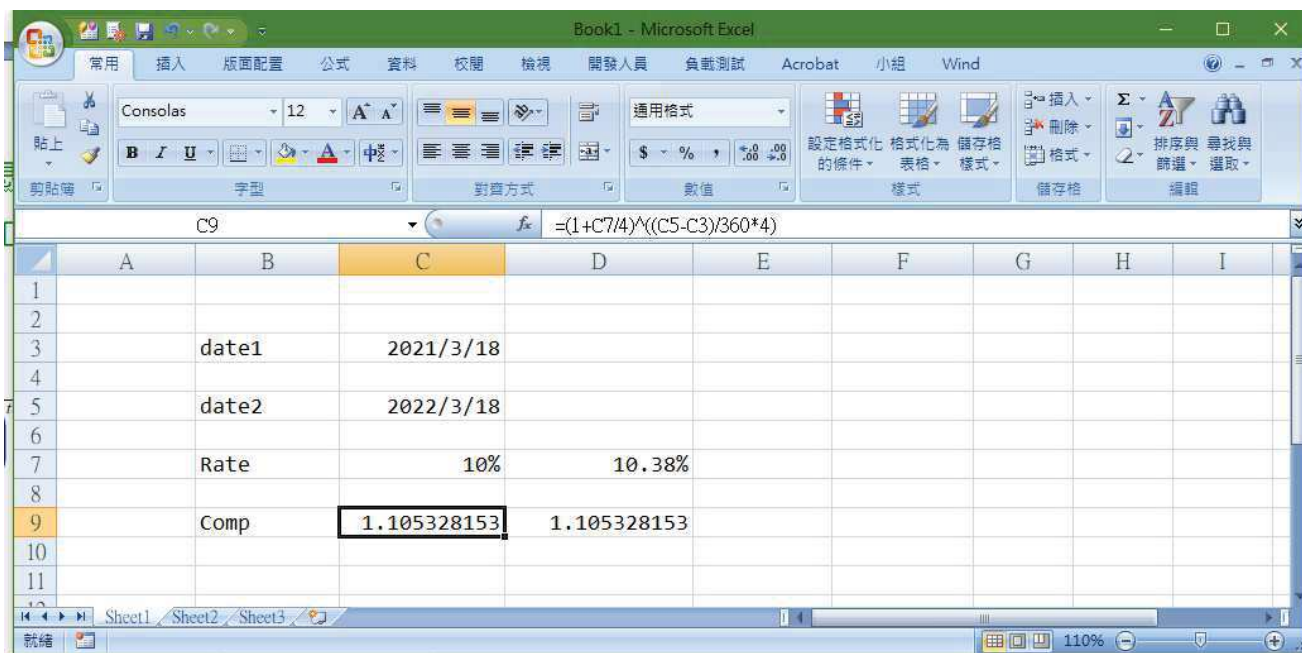
✧ 语法

```
def compoundFactor(self, *args):  
    return _QuantLib.InterestRate_compoundFactor(self, *args)
```

✧ 使用

```
>>> date1 = ql.Date(18, 3, 2021)  
>>> date2 = ql.Date(18, 3, 2022)  
>>> CompFactor = QRate.compoundFactor(date1, date2)  
>>> print(CompFactor)  
1.105328153274998
```

$$CF_t = \left(1 + \frac{10\%}{4}\right)^{4 \times \frac{(d2-d1)}{360}} = 1.105328153$$



- **Function : discountFactor()**

- ✧ 计算输入的开始日到结束日之间的折现因子

- ✧ 语法

```
def discountFactor(self, *args):  
    return _QuantLib.InterestRate_discountFactor(self, *args)
```

- ✧ 使用

```
>>> DiscFactor = QRate.discountFactor(date1, date2)  
>>> print(DiscFactor)  
0.9047087030553604
```

$$DF_t = \frac{1}{\left(1 + \frac{10\%}{4}\right)^{4 \times \frac{(d2-d1)}{360}}} = 0.904708703$$



The screenshot shows a Jupyter Notebook window titled 'Lec_1_03_3' with 'unsaved changes'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for saving, running, and other actions. The code is written in Python and uses the QuantLib library. It defines a QRate object with a 0.1 interest rate and DC360 day count convention, then calculates a compound factor and a discount factor for the period from March 18, 2021, to March 18, 2022.

```
In [1]: import QuantLib as ql  
        DC360 = ql.Actual360()  
  
In [2]: QRate = ql.InterestRate(0.1, DC360, ql.Compounded, ql.Quarterly)  
        date1 = ql.Date(18, 3, 2021)  
        date2 = ql.Date(18, 3, 2022)  
  
In [3]: CompFactor = QRate.compoundFactor(date1, date2)  
        print(CompFactor)  
1.105328153274998  
  
In [4]: DiscFactor = QRate.discountFactor(date1, date2)  
        print(DiscFactor)  
0.9047087030553604
```

- **Function : equivalentRate()**

- ✧ 根据输入的利率对象以及开始日与结束日，计算此段复利期间的相当利率

- ✧ 语法

```
def equivalentRate(self, *args):  
    return _QuantLib.InterestRate_equivalentRate(self, *args)
```

- ✧ 使用

- **Function : impliedRate()**

- ✧ 根据输入的复利因子，计算从开始日到结束日之间，基于此复利因子所隐含的利率大小。

- ✧ 语法

```
@staticmethod  
def impliedRate(*args):  
    return _QuantLib.InterestRate_impliedRate(*args)
```

- ✧ 使用

```
In [5]: annualRate = QRate.equivalentRate(DC360, ql.Compounded, ql.Anual, date1, date2)  
print(annualRate)
```

```
10.381289 % Actual/360 Annual compounding
```

```
In [6]: impRate = QRate.impliedRate(CompFactor, DC360, ql.Compounded, ql.Quarterly, date1, date2)  
print(impRate)
```

```
10.000000 % Actual/360 Quarterly compounding
```

```
In [7]: rate = annualRate.rate()  
print(rate)
```

```
0.10381289062499954
```

```
In [8]: print(QRate.dayCounter())
```

```
Actual/360 day counter
```

```
In [9]: print(QRate.frequency())
```

```
4
```

● Function : rate()

- ✧ 传回一个利率对象的利率值
- ✧ 语法

```
def rate(self):  
    return _QuantLib.InterestRate_rate(self)
```

- ✧ 使用

```
>>> rate = annualRate.rate()  
0.10381289
```

● Function : dayCounter()

- ✧ 传回计息对象
- ✧ 语法

```
def dayCounter(self):  
    return _QuantLib.InterestRate_dayCounter(self)
```

- ✧ 使用

```
>>> print(QRate.dayCounter())  
Actual/360 day counter
```

● Function : frequency()

- ✧ 传回计息频率
- ✧ 语法

```
def frequency(self):  
    return _QuantLib.InterestRate_frequency(self)
```

- ✧ 使用

```
>>> print(QRate.frequency())  
4
```