

Python 金融工程套件 QuantLib 介紹

衍生商品評價與風險計算

台灣科技大學財務金融研究所

2019/5/13，14:30 - 17:30

昀騰金融科技

技術長

董夢雲 博士

目 錄

零、QuantLib-Python 安裝

一、QuantLib 基礎類別

二、金融工具與選擇權評價引擎

三、Greeks 數值計算

四、市場報價與債券價格

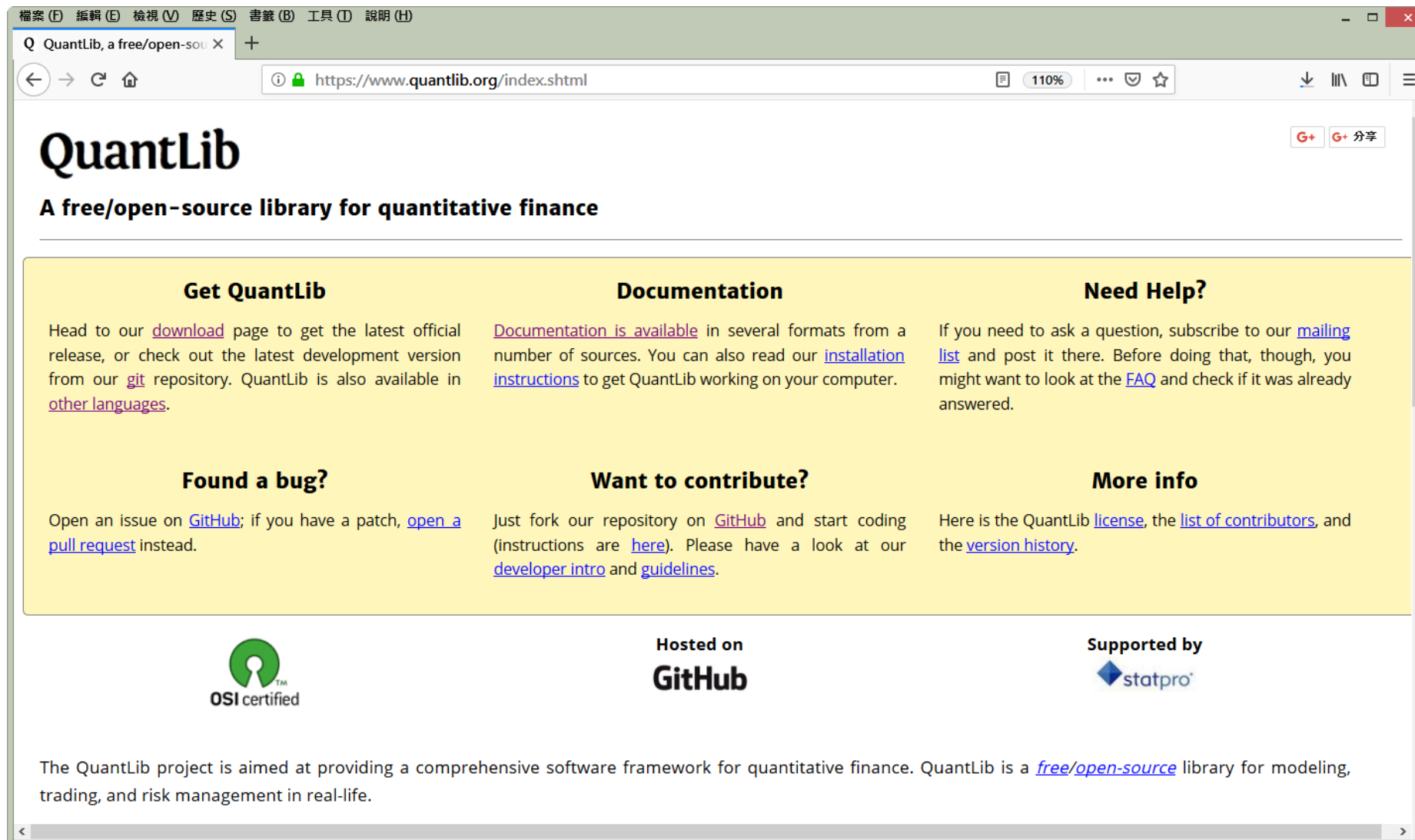
五、利率期限結構

零、QuantLib 與 QuantLib-Python 安裝

(一)QuantLib 程式庫的由來

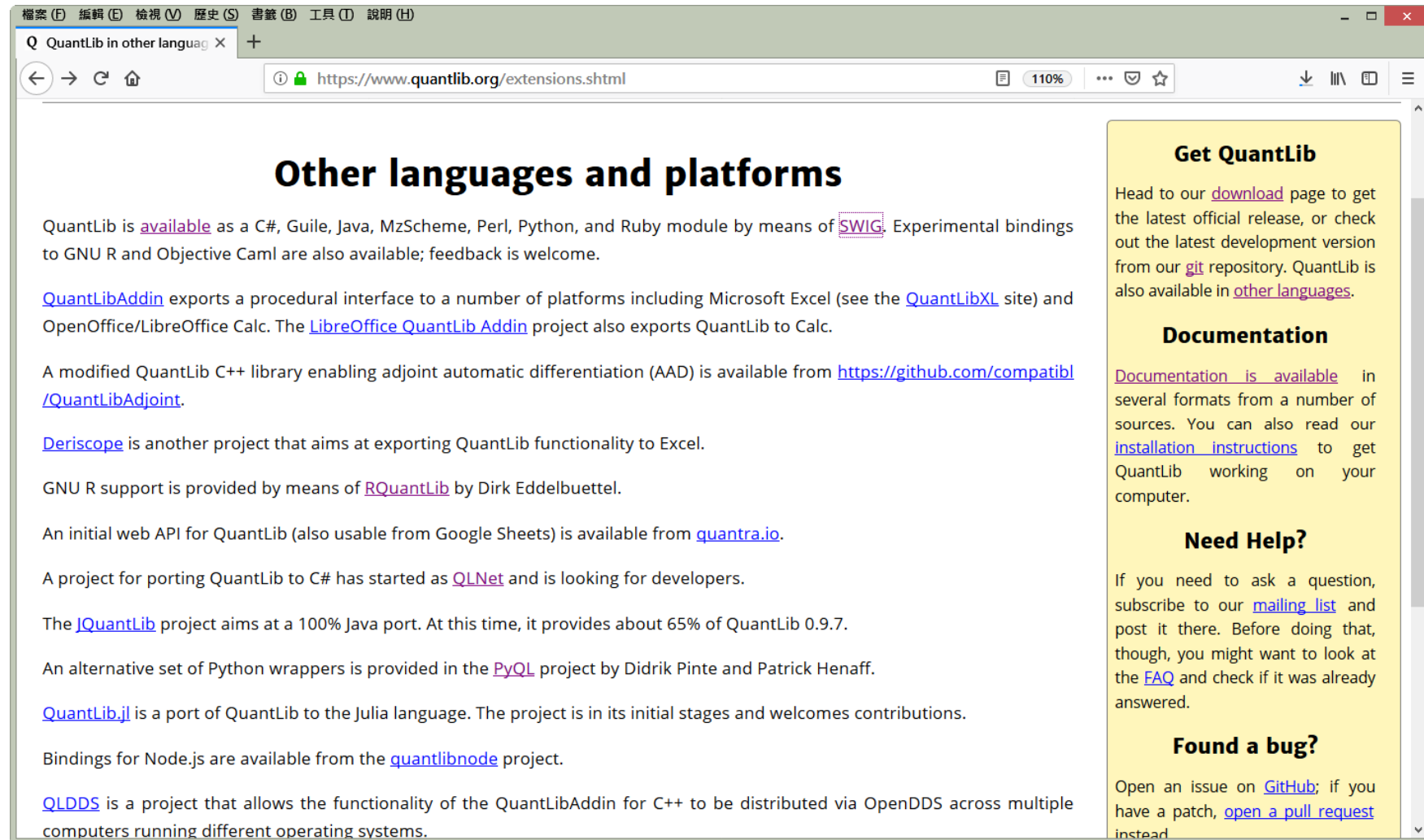
◆QuantLib 是一個開放源碼程式庫(Open-Source Software Library)。

- 目的在提供對金融工具評價和相關主題有興趣的軟體開發人員，適合的開發工具。
- 可在 QuantLib 的網站，<https://www.quantlib.org>，了解專案的相關內容，2018/09/19。



◆ QuantLib 是以 C++ 開發的程式庫，以此為基礎，被轉寫成其他不同的語言。

➤ 包括 C#、Java、Perl、Python、R、Ruby 與 Scheme。



(二)QuantLib 的歷史

◆由一群在 Cabota Banca Intesa 的利率衍生商品 Desk 工作的數量分析專家們，於西元 2000 年時，所開創的一個財務金融程式庫開發專案。

➤ 目前這些專家們已經成立了一家公司，之前稱之為 RiskMap，現在則名為 StatPro Italia。

◆QuantLib 專案目前是由 Luigi Ballabio 與 Ferdinando Ametrano 兩位專家所領導。

➤ 當他們在實作 Black-Scholes 公式的時候，Ferdinando Ametrano 分享他在另一個結構化專案中的開發經驗。

➤ 該專案允許一種新的合作方式，專案中的任何人可以改進、更正來開發一個免費的共用基礎框架 (Free Common-base Framework)。

✓ 此意見為 RiskMap 所支持並加以採用。

◆QuantLib 的創建者秉持著這樣的信念，公開的標準最適合於科學與技術的演進。

➤ 尤其在學術界之內，好的實務技能與工具只有被教育界所接受，方可在長期的演化中勝出。

◆雖然最早專案的目標對象為學術界與實務界，然而最後卻是進一步促成這兩方人士進一步的交流。

- QuantLib 所提供的工具，不論對實務上的實作或是進階的建模，都是相當有用的。
- 國內主要銀行的交易室都有在使用。
- 國外銀行也有在使用，德國工業銀行有持續支持此專案的活動。

◆下表列示了 QuantLib 各版本釋出的時間。目前最新的版本是 1.14 版(2018.10.1)。

- 除了 C++之外，其他語言的 QuantLib 專案也陸續成立運作，
 - ✓ 使用 C#的 QLNet，<https://sourceforge.net/projects/qlnet/>；
 - ✓ 使用 R 的 RQuantLib，<https://dirk.eddelbuettel.com/code/rquantlib.html>；
 - ✓ 使用 Java 的 JQuantLib，<https://www.jquantlib.org>；
 - ✓ 使用 Python 的 PyQL，<https://github.com/enthought/pyql>；
 - ✓ 以及可配合 Excel 使用的增益集，本專案的子專案 QuantLibXL。
- 透過 SWIG，可供多種語言使用的模組，本次演說使用此版本。
 - ✓ 包含 C#，Java，Perl，Python，R，Ruby，Scala。

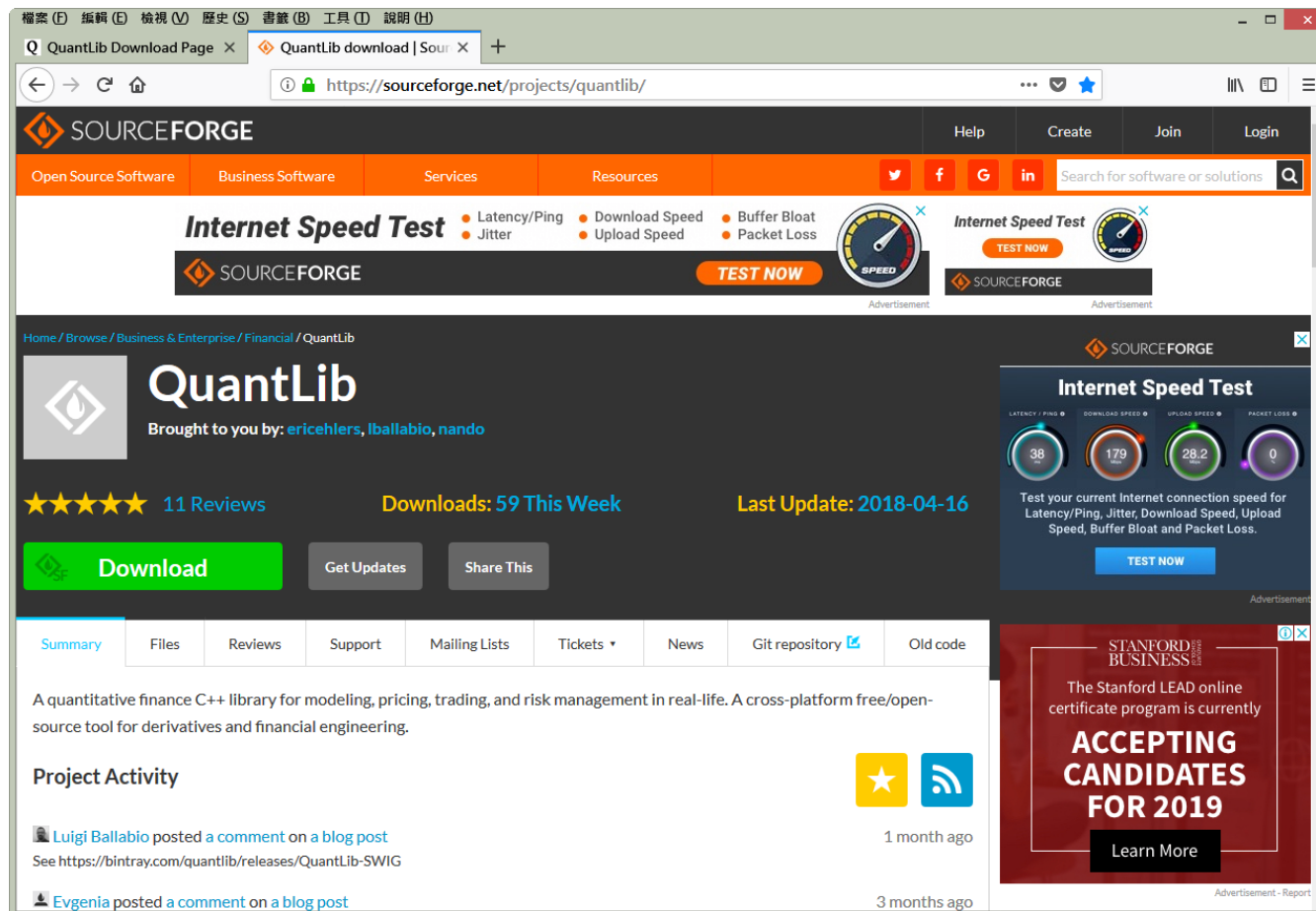
◆ QuantLib 各早期版本的演進與推出時間表。

Version	Release date	Notes
0.1.1	Nov 21, 2000	
0.2.0	Sep 18, 2001	
0.3.4	Nov 21, 2003.	本人首次發現，在 KGI。
0.3.7	Jul 23, 2004.	此版之後 QuantLib 需要 Boost 程式庫。
0.4.0	Feb 20, 2007.	
0.8.0	May 30, 2007.	版本的大幅更動，目的在加快收斂到 1.0 版。
0.9.0	Dec 24, 2007.	
0.9.9	Nov 2009.	
1.0.0	Feb 24, 2010	
1.0.1	Sep 17, 2010	
1.1.0	May 23, 2011	
1.2.0	March 6, 2012	

(三)QuantLib 程式庫的下載

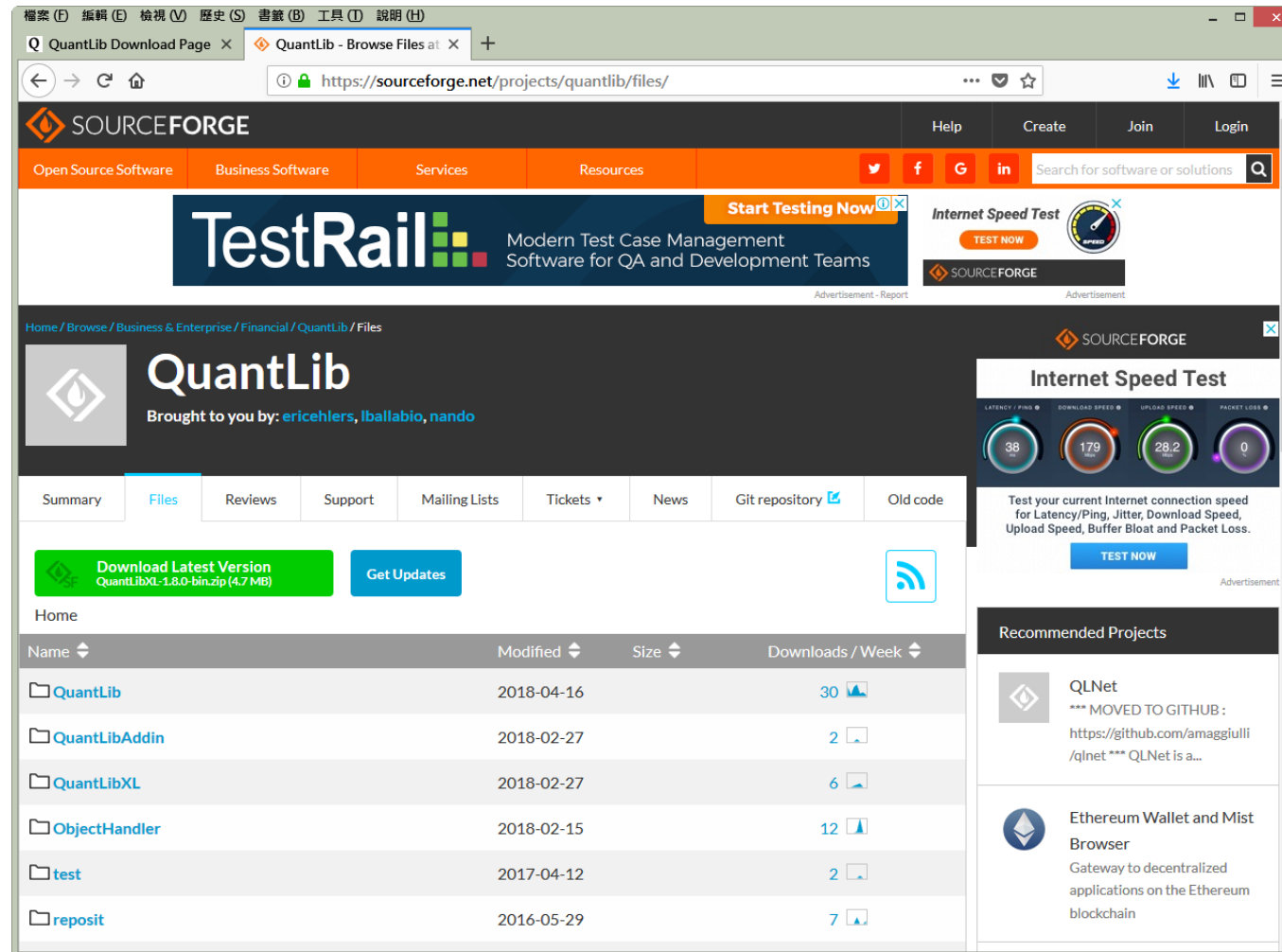
◆可在 Open Source Software 網站內的 QuantLib 專案網頁下載，

➤ <http://sourceforge.net/projects/quantlib/>，2018/09/16。



◆點 Files，可以看到下面目錄結構。

➤ 點選 QuantLib，



The screenshot shows the SourceForge project page for QuantLib. The 'Files' tab is selected, displaying a list of files and folders. The table below summarizes the content:

Name	Modified	Size	Downloads / Week
QuantLib	2018-04-16		30
QuantLibAddin	2018-02-27		2
QuantLibXL	2018-02-27		6
ObjectHandler	2018-02-15		12
test	2017-04-12		2
reposit	2016-05-29		7

Additional details visible on the page include a 'Download Latest Version' button for QuantLibXL-1.8.0-bin.zip (4.7 MB) and a 'Get Updates' button. The right sidebar features an 'Internet Speed Test' and 'Recommended Projects' like QLNet and Ethereum Wallet and Mist Browser.

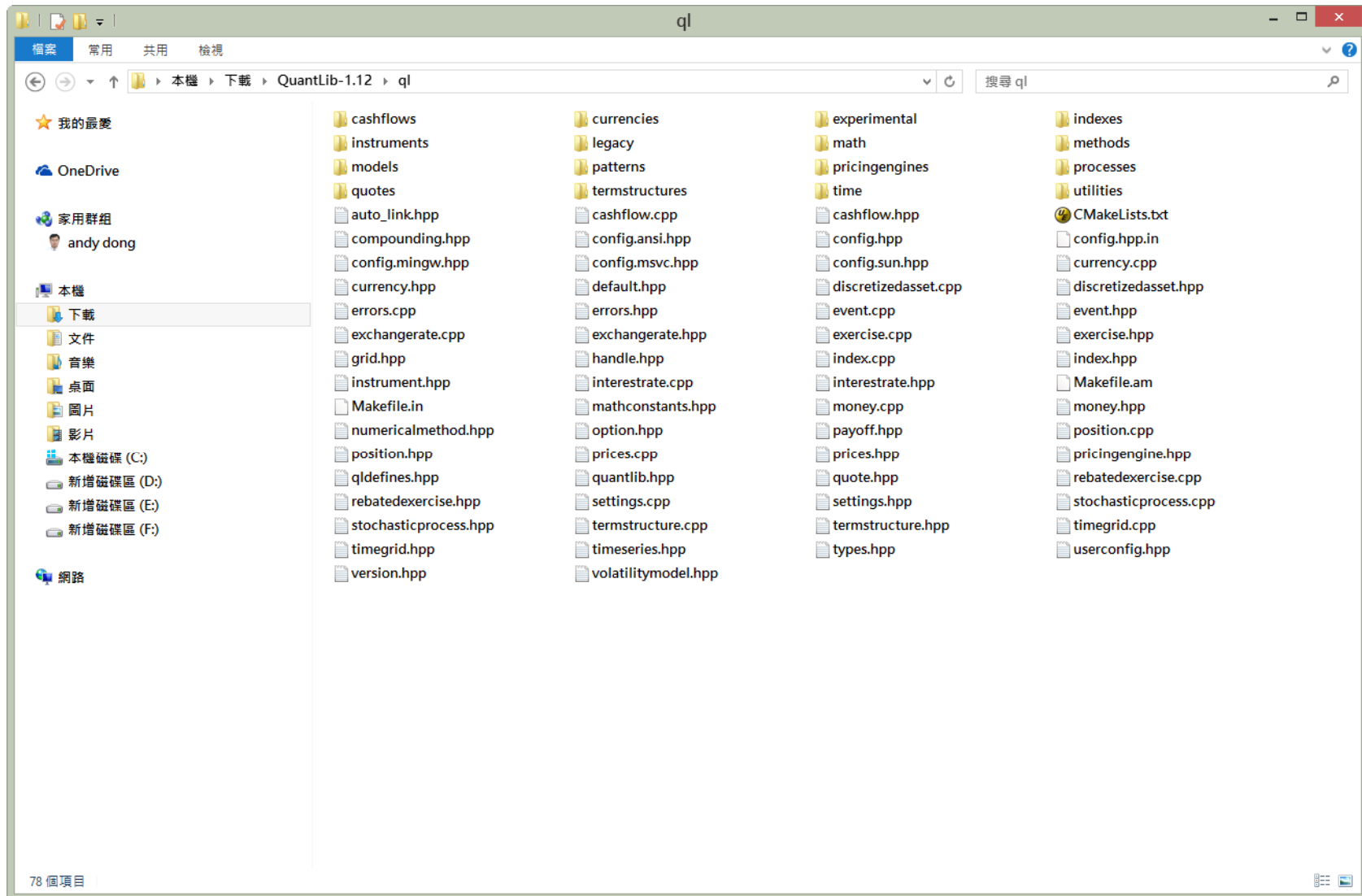
◆ 最新版 1.12 版。

The screenshot shows the SourceForge project page for QuantLib. The browser address bar displays the URL <https://sourceforge.net/projects/quantlib/files/QuantLib/>. The page features a navigation bar with links to Open Source Software, Business Software, Services, and Resources. A search bar is also present. The main content area includes a banner for 'Easy data viz in Python with Dash' and an 'Internet Speed Test' advertisement. Below the banner, the QuantLib project is highlighted, with a note 'Brought to you by: ericehlers, lballabio, nando'. The 'Files' tab is selected, showing a list of versions. A green button 'Download Latest Version' is visible, along with a 'Get Updates' button. The version list table is as follows:

Name	Modified	Size	Downloads / Week
Parent folder			
1.12	2018-02-01		13
1.11	2017-10-02		6
1.10.1	2017-08-31		0
1.10	2017-05-16		0
1.9.2	2017-02-27		0

On the right side, there is a 'Recommended Projects' section listing 'QLNet' (noted as moved to GitHub) and 'Ethereum Wallet and Mist Browser'.

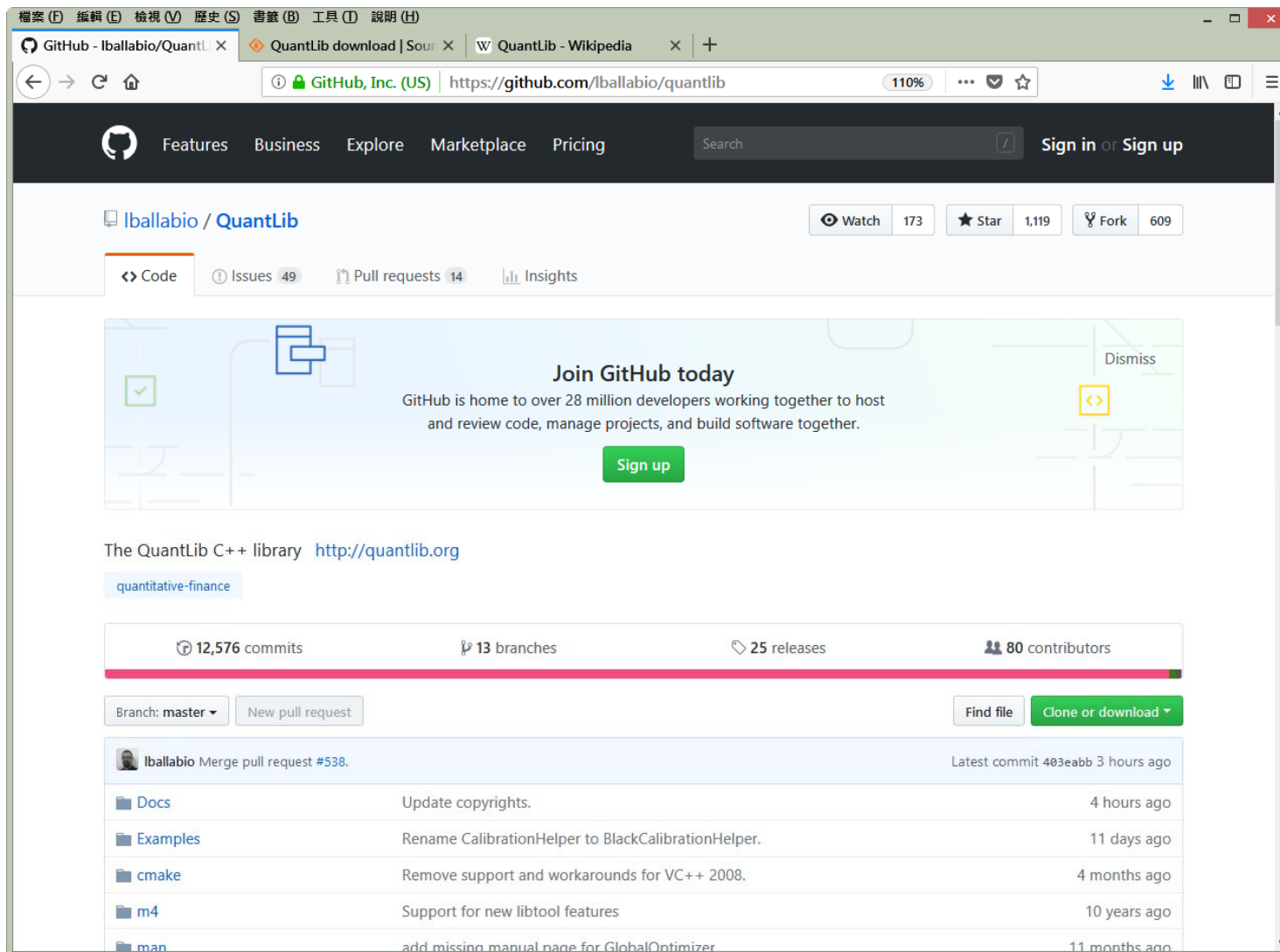
◆完整目錄



◆QuantLib 的 Wiki 網頁為，<http://en.wikipedia.org/wiki/QuantLib>。



◆ Github 網站，2018/09/16。



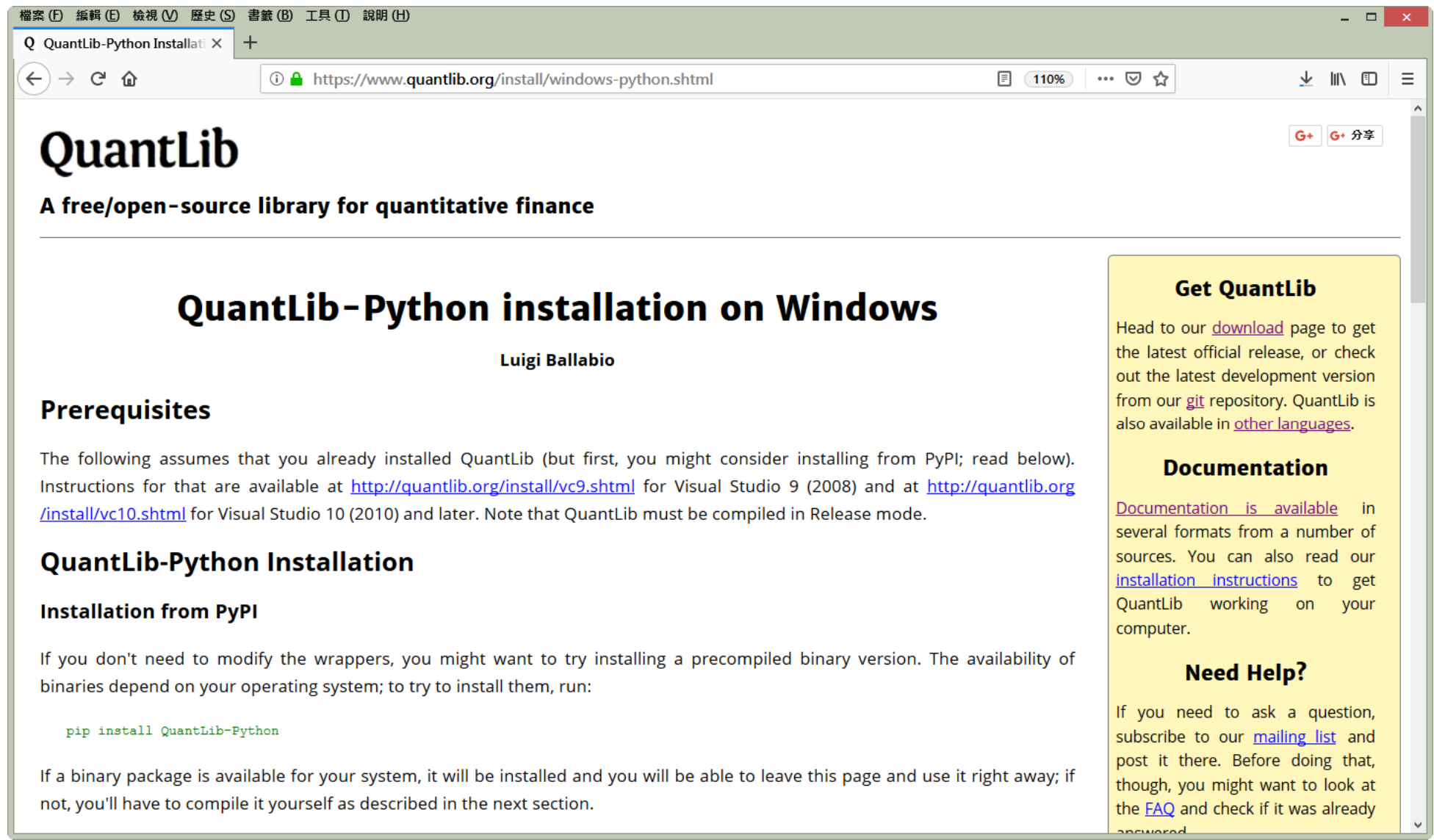
(四)QuantLib 程式庫的內容

◆ QuantLib 程式庫內容豐富，包含的檔案兩千餘個，可以概分 15 類模組。

- 1.Numeric types：主要在定義各類資料的資料型別，例如，利率(Rate)、利差(Spread)、波動性(Volatility)是實數(Real)。
- 2.Currencies and FX rates：包括 66 種貨幣類別，以及匯率轉換計算與管理的物件。
- 3.Date and time calculations：包括 37 種不同國家/地區日曆類別、7 種不同計息方式類別，以及相關日期計算的物件。
- 4.Pricing engines：九大類評價引擎，包括 Asian option、Barrier option、Basket option、Cap/Floor、Cliquet option、Forward option、Quanto option、Swaption、Vanilla option 等。每一類都有解析解、二元樹、有限差分法與蒙地卡羅模擬的實作類別。
- 5.Finite-differences framework：三大類有限差分法的實作類別。
- 6.Short-rate modelling framework：包括單因子模型，Vasicek、CIR、Hull-White、Black-Karasinski、Extended CIR 等模型，以及雙因子模型的 G2 模型。

- 7.Financial instruments：40 多種的金融工具，含 Swap、Vanilla Option、Exotic Option、Stock、Forward、Cap、Floor、Color、Bond、Future、Callable Bond 以及相對應的 Quanto 產品、Inflation Bond。
- 8.Lattice methods：一維與二維的二元樹和三元樹模型。
- 9.Math tools：包括分配、積分、相關性、內差、矩陣、最適化、亂數、求解、統計等九類數學工具。
- 10.Monte Carlo framework：單變數與多變數的歐式與美式模擬方法物件。
- 11.Design patterns：Singleton、Observer/Observable、Lazy Object、Composite、Curiously Recurring Template、Acyclic Visitor 樣式。
- 12.Stochastic processes：幾何布朗運動、隨機波動模型、Square Root Process、Ornstein Uhlenbeck Process、Hull White Process、G2 Process。
- 13.Term structures：包括利率、波動性、信用與通膨的期限結構物件。
- 14.Models：分別包括 Equity Model、Short Rate Model、Volatility Model 中使用的相關模型，以及針對 Libor/Swap Market Model 涉及的金融工具與模擬方法的相關物件上百個，。
- 15.QuantLib macros：一些數值極限與除錯所需的 Macros。

(五)QuantLib-Python Package下載與安裝



The screenshot shows a web browser window with the URL <https://www.quantlib.org/install/windows-python.shtml>. The page title is "QuantLib" with the subtitle "A free/open-source library for quantitative finance". The main heading is "QuantLib-Python installation on Windows" by Luigi Ballabio. The page is divided into sections: "Prerequisites", "QuantLib-Python Installation", and "Installation from PyPI". The "Installation from PyPI" section contains a code block for installing QuantLib-Python using pip. A right sidebar contains sections for "Get QuantLib", "Documentation", and "Need Help?".

QuantLib
A free/open-source library for quantitative finance

QuantLib-Python installation on Windows

Luigi Ballabio

Prerequisites

The following assumes that you already installed QuantLib (but first, you might consider installing from PyPI; read below). Instructions for that are available at <http://quantlib.org/install/vc9.shtml> for Visual Studio 9 (2008) and at <http://quantlib.org/install/vc10.shtml> for Visual Studio 10 (2010) and later. Note that QuantLib must be compiled in Release mode.

QuantLib-Python Installation

Installation from PyPI

If you don't need to modify the wrappers, you might want to try installing a precompiled binary version. The availability of binaries depend on your operating system; to try to install them, run:

```
pip install QuantLib-Python
```

If a binary package is available for your system, it will be installed and you will be able to leave this page and use it right away; if not, you'll have to compile it yourself as described in the next section.

Get QuantLib

Head to our [download](#) page to get the latest official release, or check out the latest development version from our [git](#) repository. QuantLib is also available in [other languages](#).

Documentation

[Documentation is available](#) in several formats from a number of sources. You can also read our [installation instructions](#) to get QuantLib working on your computer.

Need Help?


If you need to ask a question, subscribe to our [mailing list](#) and post it there. Before doing that, though, you might want to look at the [FAQ](#) and check if it was already answered.



◆ Source Code



The screenshot shows a web browser window displaying the Bintray page for the QuantLib-SWIG package. The browser's address bar shows the URL <https://bintray.com/quantlib/releases/QuantLib-SWIG>. The page features a green header with the JFrog logo and navigation links for API, User Guide, Pricing, and Sign In. Below the header, the package name 'quantlib / releases / QuantLib-SWIG' is displayed, along with a download link <https://dl.bintray.com/quantlib/QuantLib-SWIG>. The package is owned by QuantLib, indicated by a star rating and a 'Report' button. A description states 'Wrappers for QuantLib in a number of languages'. A 'SET ME UP!' button with a wrench icon is visible. At the bottom, there are tabs for General, Readme, Release Notes, Reviews (0), Statistics, and Files. Additional links include 'About This Package', 'Version Notification Links', 'Versions', and 'Latest Version Badge'.


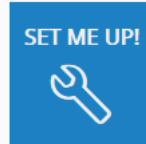
Package QuantLib-SWIG - x

← → ↻ 🏠 <https://bintray.com/quantlib/releases/QuantLib-SWIG> 133% ⋮ 🛡️ ☆ ⬇️ 📄 📖 ☰

 API User Guide Pricing Sign In



 quantlib / releases / QuantLib-SWIG  <https://dl.bintray.com/quantlib/QuantLib-SWIG>

Owned by [QuantLib](#) ★★★★★  Report 

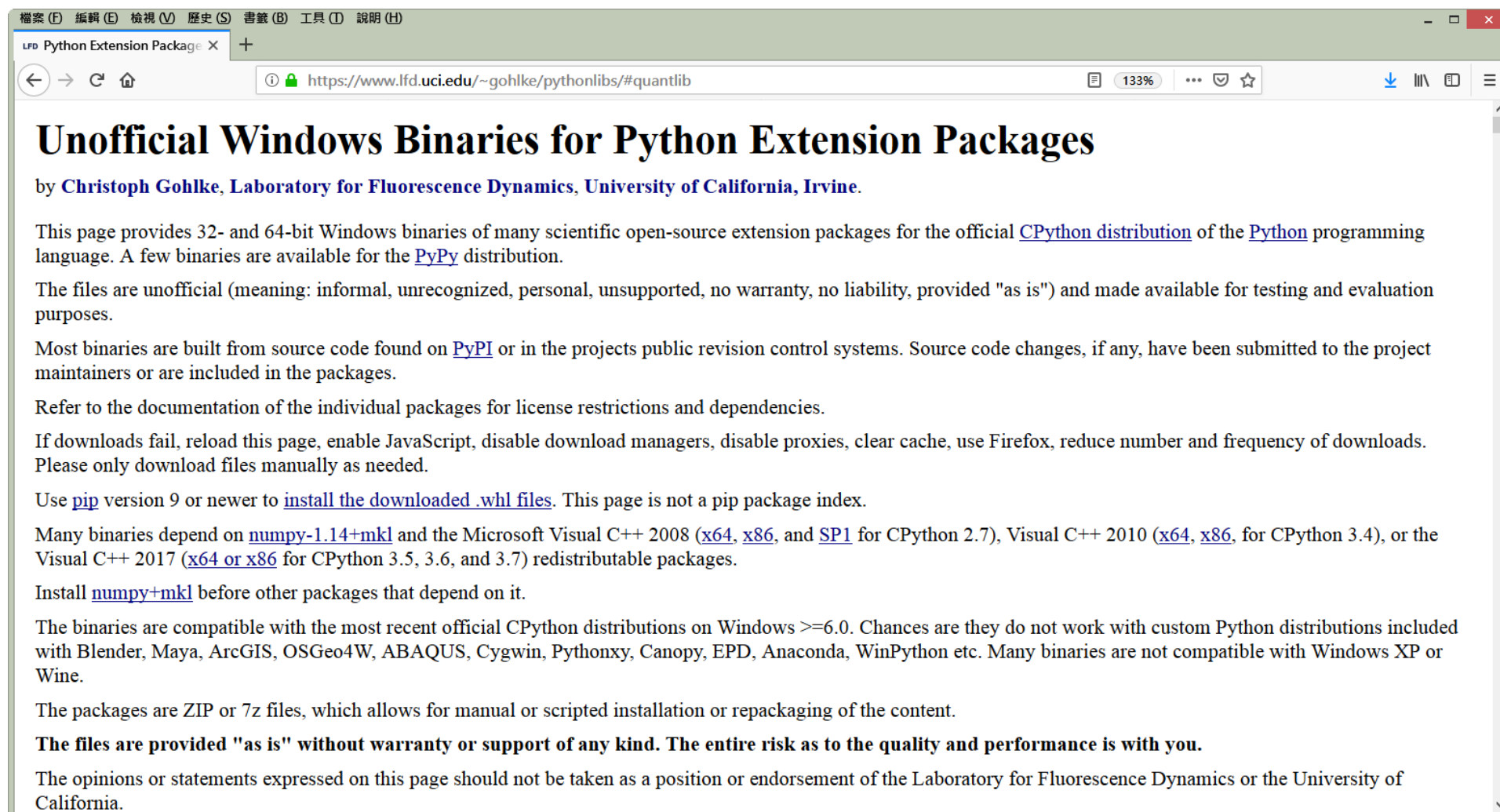
 Wrappers for QuantLib in a number of languages 

Feedback

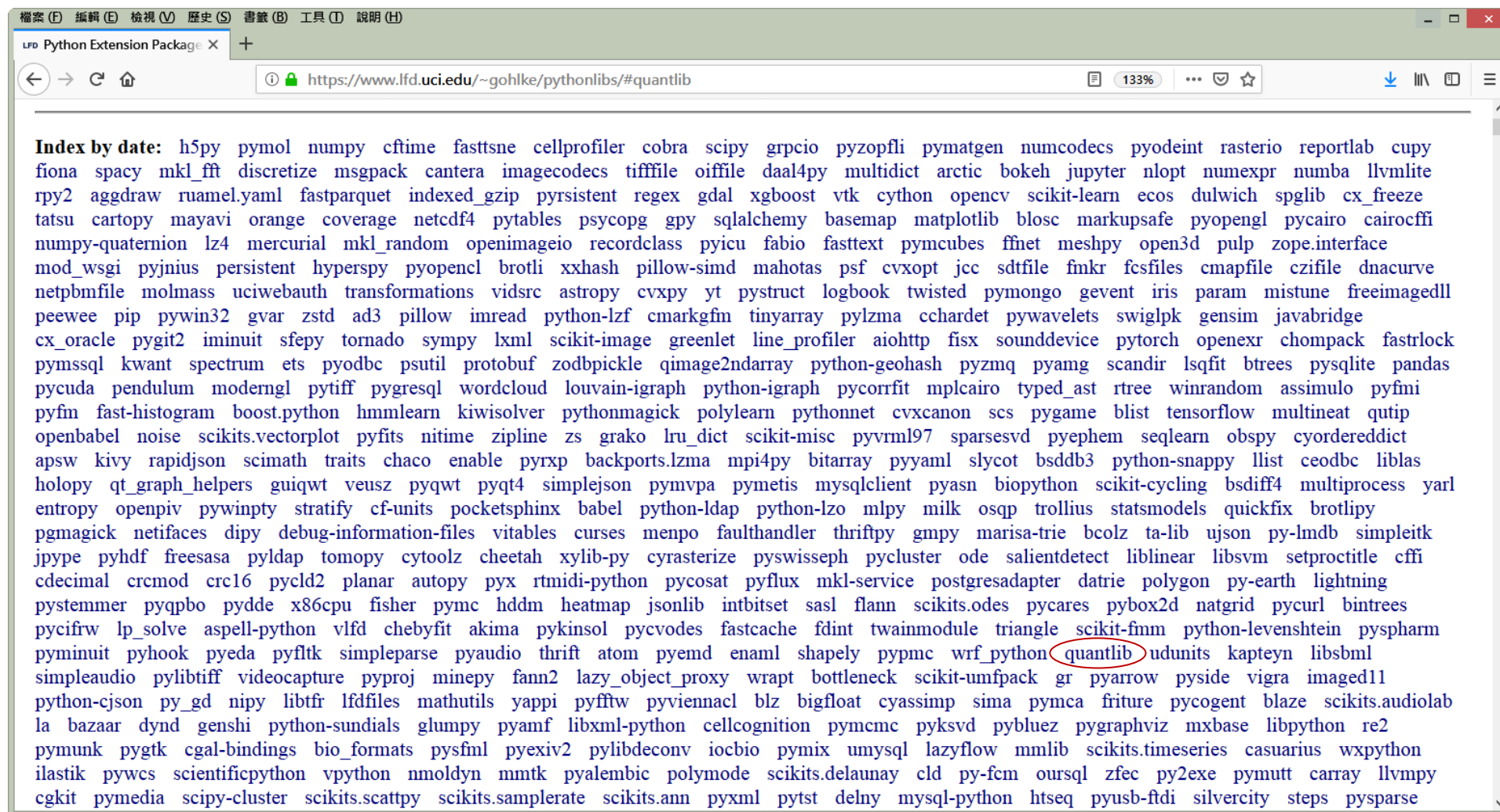
General Readme Release Notes Reviews (0) Statistics Files

About This Package Version Notification Links  Versions Latest Version Badge 

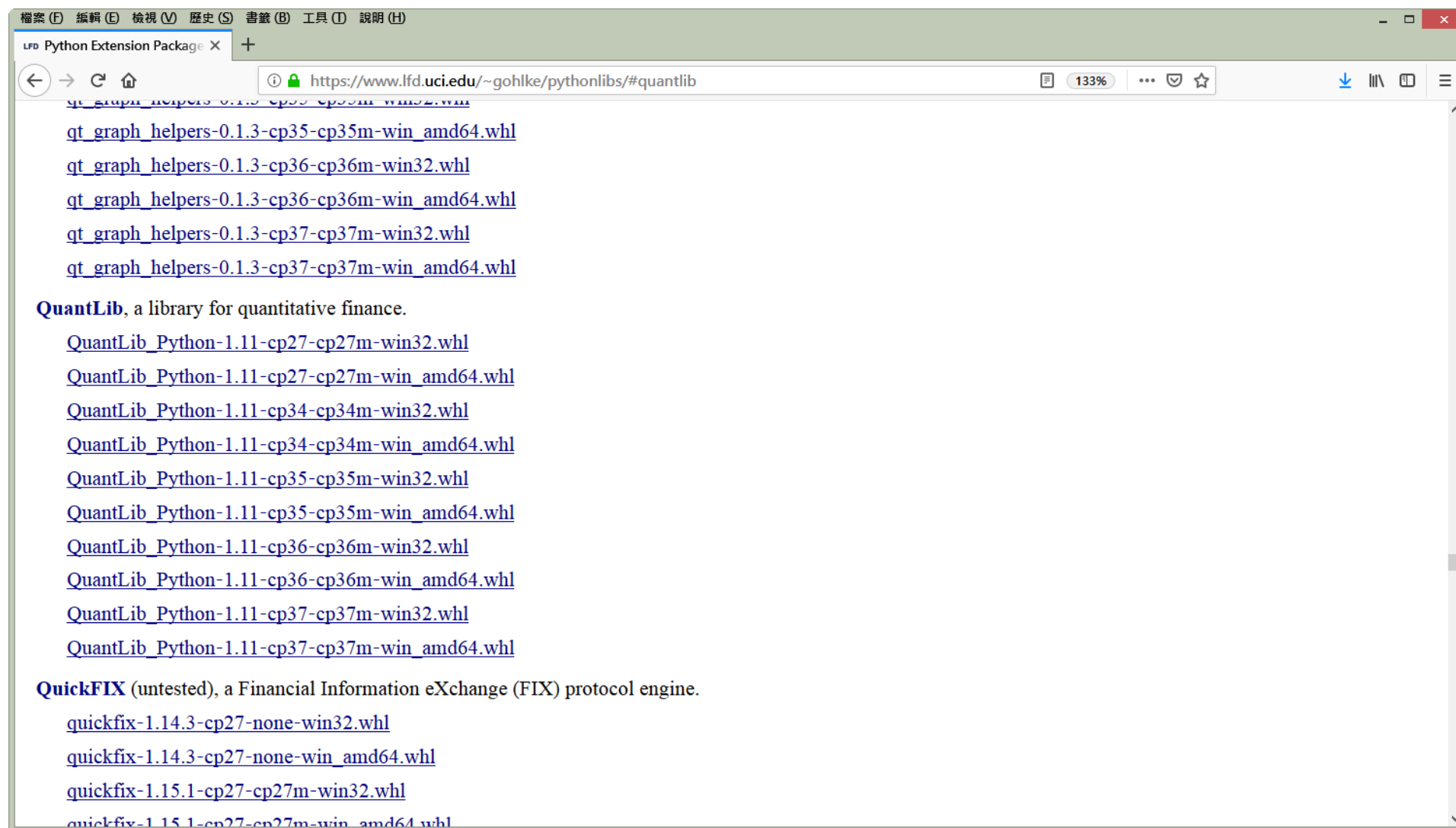
◆ 下載網址



➤ Package 索引

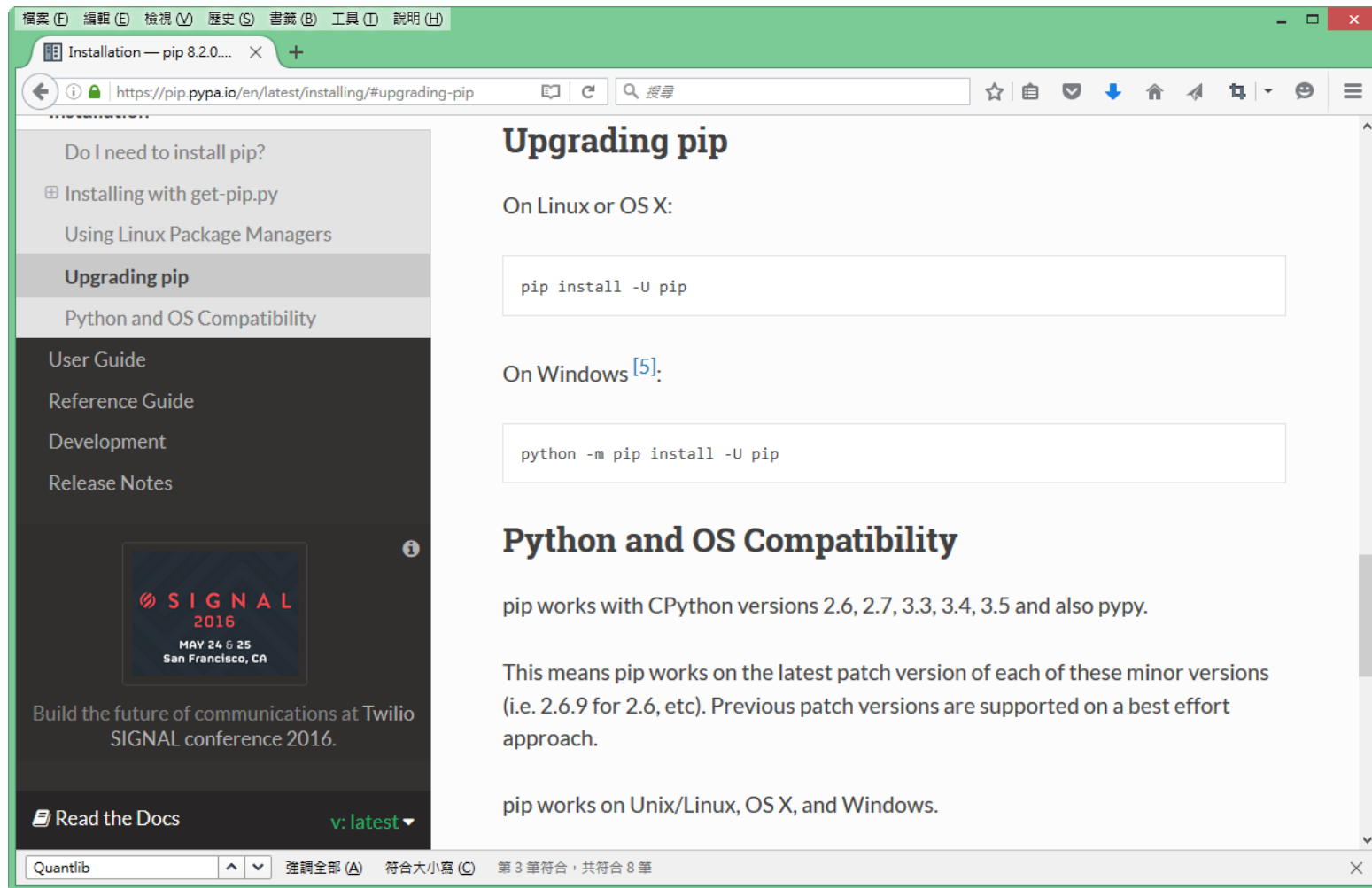


➤ QuantLib 檔案



◆升級 pip

```
C:\>python -m pip install -U pip
```



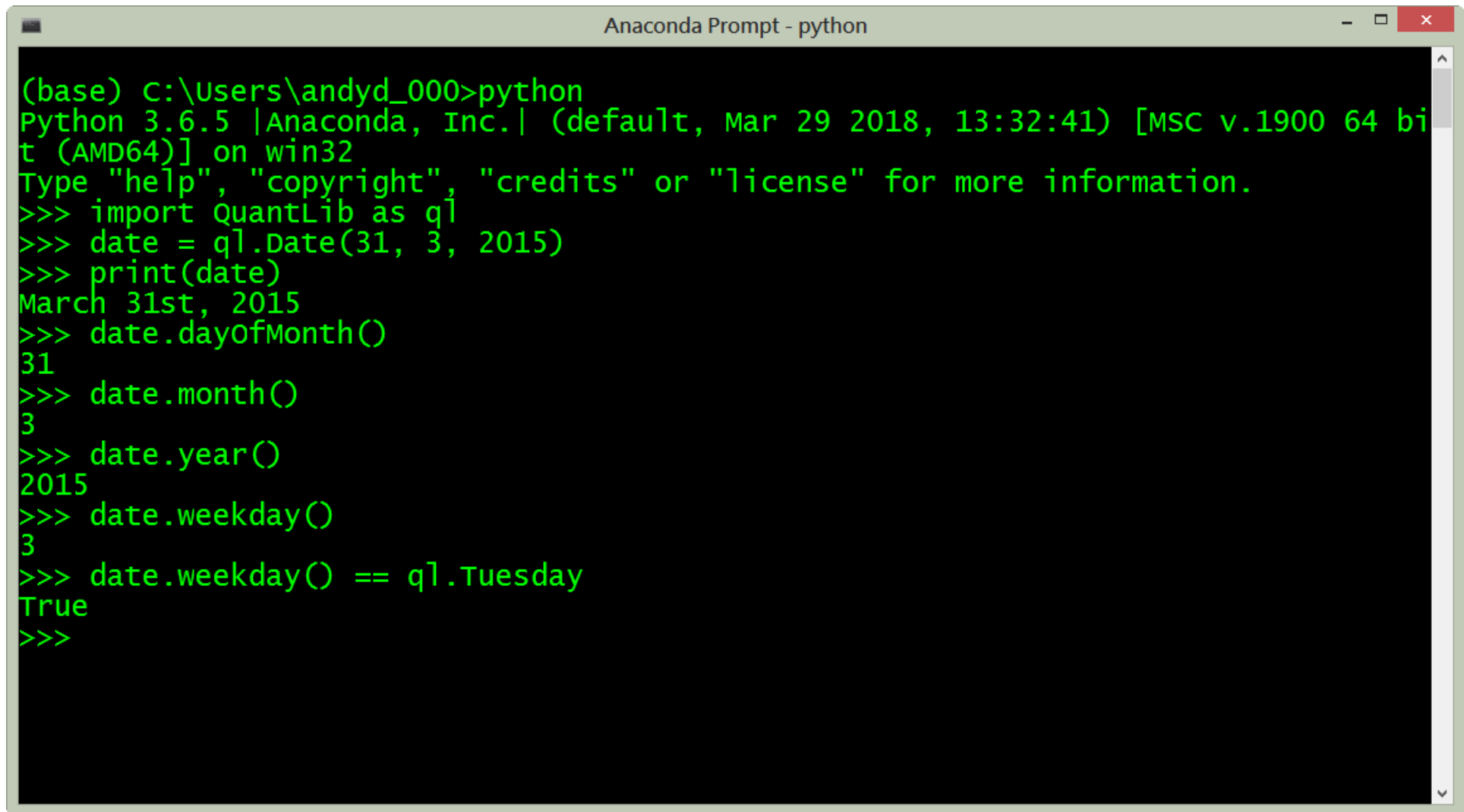
◆安裝 QuantLib_Python Package , whl 格式。

```
C:\>pip install QuantLib_Python-1.11-cp36-cp36m-win_amd64.whl
```

◆最簡單的安裝方式

```
C:\>pip install QuantLib-Python
```

(六)QuantLib Package 使用測試

A screenshot of an Anaconda Prompt window titled "Anaconda Prompt - python". The window has a dark background with green text. The prompt shows a user running a Python script. The script imports QuantLib as 'ql', creates a date object for March 31, 2015, and then uses various methods to extract date components and check the day of the week. The output shows the date as "March 31st, 2015", the day of the month as 31, the month as 3, the year as 2015, and the weekday as 3. Finally, it checks if the weekday is equal to ql.Tuesday, which returns True.

```
(base) C:\Users\andyd_000>python
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bi
t (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import QuantLib as ql
>>> date = ql.Date(31, 3, 2015)
>>> print(date)
March 31st, 2015
>>> date.dayOfMonth()
31
>>> date.month()
3
>>> date.year()
2015
>>> date.weekday()
3
>>> date.weekday() == ql.Tuesday
True
>>>
```


◆ 主要參考網站

The screenshot shows a web browser window with the address bar displaying `gouthamanbalaraman.com/blog/quantlib-basics.html`. The page title is "Introduction to QuantLib Python". The author is Goutham Balaraman, dated March 24, 2015. The page content includes a brief introduction to the QuantLib Python library and a code snippet for importing the library.

Introduction to QuantLib Python

March 24, 2015 by [Goutham Balaraman](#)
Share on: [Diaspora*](#) / [Twitter](#) / [Facebook](#) / [Google+](#) / [Email](#) / [Bloglovin](#)

This post will walk through some of the basics of QuantLib Python library.

Visit here for other [QuantLib Python examples](#). If you found these posts useful, please take a minute by providing some [feedback](#).

I installed the latest version of QuantLib (V1.5) and the python wrapper to QuantLib. My experiments lately have been to get a feel for the QuantLib API. The library itself is so extensive, that it is rather hard for a new comer to get going. In this post we will look into some of the basic classes and functionality in QuantLib.

Let us import QuantLib as:

```
import QuantLib as ql
```

Checkout my book

檔案 (F) 編輯 (E) 檢視 (V) 歷史 (S) 書籤 (B) 工具 (T) 說明 (H)

QuantLib Python Tutorials With Examples

gouthamanbalaraman.com/blog/quantlib-python-tutorials-with-examples.html 120%

GB About

QuantLib Python Tutorials With Examples


October 30, 2015 by [Gouthaman Balaraman](#)
Share on: [Diaspora*](#) / [Twitter](#) / [Facebook](#) / [Google+](#) / [Email](#) / [Bloglovin](#)

This post is a collection of links to all my quantlib python tutorial

Visit here for other [QuantLib Python examples](#). If you found these posts useful, please take a minute by providing some [feedback](#).


I have written a lot of little tutorials on using QuantLib python bindings. In these posts I explain some of the QuantLib concepts using minimal examples. Following are the links to these posts:

- [Introduction to QuantLib Python](#): This post will walk through some of the basics of QuantLib Python library.
- [Modeling Fixed Rate Bonds in QuantLib Python](#): This post will walk through an example of modeling fixed rate bonds using QuantLib Python.
- [An Introduction to Interest Rate Term Structure in QuantLib Python](#): This post will walk through the basics of bootstrapping yield curve in QuantLib Python.
- [Hull White Term Structure Simulations with QuantLib Python](#): Discusses simulation of the Hull



I am Goutham Balaraman, and I explore topics in quantitative finance, programming, and data science. You can follow me [@gsbalaraman](#).

Checkout my book



一、QuantLib 基礎類別

◆Date、Period、Calendar 與 Schedule 等類別，

- 在產生金融工具(Instrument)、模型(Model)與期限結構(Term Structure)方面，成為 QuantLib 的基本構件。

```
In [1]:from QuantLib import *  
import pandas as pd
```

(一)Date & Period Class

◆日期物件以建構子 `Date(day, month, year)`產生。

➤ 與 Python `datetime` 物件實例化不同，先日、中月、後年。

```
In [2]: date = Date(31, 3, 2015)

        print(date)
```

```
Out[2]: March 31st, 2015
```

◆可以 `month()`, `dayOfMonth()`與 `year()`等方法取得其欄位。

➤ `weekday()`方法取得其星期值。

```
In [3]: print("%d-%d-%d" %(date.month(), date.dayOfMonth(), date.year()))
```

```
Out[3]: 3-31-2015
```

```
In [4]: date.weekday() == Tuesday
```

```
Out[4]: True
```

◆Date 物件可以進行算數運算，如前進日、周、月等，

◆星期、月等周期時間，可以 Period 物件表示，

➤ 周期物件建構子為 Period(num_periods, period_type)。

✓ num_periods 為整數，表示周期數目，

✓ period_type 表周期單位，可為 Weeks, Months and Years。

```
In [5]: type(date+1)
```

```
Out[5]: QuantLib.QuantLib.Date
```

```
In [6]: print("Add a day : {0}".format(date + 1))
```

```
print("Subtract a day : {0}".format(date - 1))
```

```
print("Add a week : {0}".format(date + Period(1, Weeks)))
```

```
print("Add a month : {0}".format(date + Period(1, Months)))
```

```
print("Add a year : {0}".format(date + Period(1, Years)))
```

```
Out[6]:Add a day : April 1st, 2015
        Subtract a day : March 30th, 2015
        Add a week : April 7th, 2015
        Add a month : April 30th, 2015
        Add a year : March 31st, 2016
```

◆Date 物件可以進行邏輯運算。

```
In [7]: print(date == Date(31, 3, 2015))
        print(date > Date(30, 3, 2015))
        print(date < Date(1, 4, 2015))
        print(date != Date(1, 4, 2015))
```

```
Out[7]: True
        True
        True
        True
```

- ◆Date 物件用於設定評價日，工具的發行日與到期日。
- ◆Period 物件用於設定期限(tenors)，例如債息的支付或建立支付的時程(payment schedules)。

(二)Calendar Class

◆上述 Date 運算沒有考慮到假日(Holidays)的問題。

➤ 然而不同工具需考慮特定交易所與國家的假日。Calendar 類別實作主要交易所的這些功能。

```
In [8]: date = Date(31, 3, 2015)
        us_calendar = UnitedStates()
        italy_calendar = Italy()
        period = Period(60, Days)
        raw_date = date + period
        us_date = us_calendar.advance(date, period)
        italy_date = italy_calendar.advance(date, period)

        print("Add 60 days: {0}".format(raw_date))

        print("Add 60 business days in US: {0}".format(us_date))

        print("Add 60 business days in Italy: {0}".format(italy_date))
```

```
Out[8]: Add 60 days: May 30th, 2015

        Add 60 business days in US: June 24th, 2015

        Add 60 business days in Italy: June 26th, 2015
```


◆addHoliday 與 removeHoliday 方法可用來加入或移除特定日曆的假日。

◆businessDaysBetween 方法可用來計算兩日期間的假日數目。

```
In [9]: us_busdays = us_calendar.businessDaysBetween(date, us_date)
        italy_busdays = italy_calendar.businessDaysBetween(date, italy_date)

        print("Business days US: {}".format(us_busdays))

        print("Business days Italy: {}".format(italy_busdays))
```

```
Out[9]: Business days US: 60
        Business days Italy: 60
```

◆ 特定交易的評價，可能需要觀察一個以上的日曆，

➤ QuantLib 有 JointCalendar 類別，合併兩個以上的日曆。

```
In [10]: joint_calendar = JointCalendar(us_calendar, italy_calendar)
         joint_date = joint_calendar.advance(date, period)
         joint_busdays = joint_calendar.businessDaysBetween(date, joint_date)

         print("Add 60 business days in US-Italy: {0}".format(joint_date))

         print("Business days US-Italy: {0}".format(joint_busdays))
```

```
Out[10]: Add 60 business days in US-Italy: June 29th, 2015
         Business days US-Italy: 60
```

(三)Schedule類別

◆Schedule 物件在產生債息與贖回時程上，是必要的工具，有下述的建構子：

```
Schedule(const Date& effectiveDate, const Date& terminationDate, const Period& tenor,  
         const Calendar& calendar, BusinessDayConvention convention,  
         BusinessDayConvention terminationDateConvention, DateGeneration::Rule rule,  
         bool endOfMonth, const Date& firstDate = Date(), const Date& nextToLastDate = Date())
```

```
Schedule(const std::vector<Date>&, const Calendar& calendar, BusinessDayConvention  
         rollingConvention)
```

➤ 使用範例

```
In [11]: effective_date = Date(1, 1, 2015)  
        termination_date = Date(1, 1, 2016)  
        tenor = Period(Monthly)  
        calendar = UnitedStates()  
        business_convention = Following  
        termination_business_convention = Following  
        date_generation = DateGeneration.Forward  
        end_of_month = False
```

```
schedule = Schedule(effective_date, termination_date, tenor, calendar,  
                    business_convention, termination_business_convention, date_generation,  
                    end_of_month)  
pd.DataFrame({'date': list(schedule)})
```

Out[11]:

date

0 January 2nd, 2015

1 February 2nd, 2015

2 March 2nd, 2015

3 April 1st, 2015

4 May 1st, 2015

5 June 1st, 2015

6 July 1st, 2015

7 August 3rd, 2015

8 September 1st, 2015

9 October 1st, 2015

10 November 2nd, 2015

11 December 1st, 2015

12 January 4th, 2016

◆Schedule 物件包含介於 effective_date 與 termination_date 間的日期，tenor 說明週期 Period 為 Monthly。

- calendar 物件用來決定 holidays。
- 此處使用慣例為 following，因此假日被排除於日期中。

◆Schedule 類別可以處理不規律的日期產生。

- 兩個額外的參數，firstDate 與 nextToLastDate，搭配 forward 與 backward 日期產生規則，可以產生 short 或 long stub 的支付時程。

◆例如，下述 firstDate 與 backward 產生規則，產生一個 January 15, 2015 為前端的 short stub。

```
In [12]: # short stub in the front
```

```
    effective_date = Date(1, 1, 2015)
    termination_date = Date(1, 1, 2016)
    first_date = Date(15, 1, 2015)
    schedule = Schedule(effective_date, termination_date, tenor, calendar,
                        business_convention, termination_business_convention, DateGeneration.Backward,
                        end_of_month, first_date)
    pd.DataFrame({'date': list(schedule)})
```

```
Out[12]:
```

```
date
0 January 2nd, 2015
1 January 15th, 2015
2 February 2nd, 2015
3 March 2nd, 2015
4 April 1st, 2015
5 May 1st, 2015
```

6 June 1st, 2015
7 July 1st, 2015
8 August 3rd, 2015
9 September 1st, 2015
10 October 1st, 2015
11 November 2nd, 2015
12 December 1st, 2015
13 January 4th, 2016

◆使用 `nextToLastDate` 參數配合 `forward` 產生規則，產生一個在時程尾端的 `short stub`。

```
In [13]: # short stub at the back
```

```
    effective_date = Date(1, 1, 2015)
    termination_date = Date(1, 1, 2016)
    penultimate_date = Date(15, 12, 2015)
    schedule = Schedule(effective_date, termination_date, tenor, calendar,
                        business_convention, termination_business_convention, DateGeneration.Forward,
                        end_of_month, Date(), penultimate_date)
    pd.DataFrame({'date': list(schedule)})
```

```
Out[13]:
```

```
date
0 January 2nd, 2015
1 February 2nd, 2015
2 March 2nd, 2015
3 April 1st, 2015
4 May 1st, 2015
5 June 1st, 2015
6 July 1st, 2015
```


7 August 3rd, 2015

8 September 1st, 2015

9 October 1st, 2015

10 November 2nd, 2015

11 December 1st, 2015

12 December 15th, 2015

13 January 4th, 2016

◆使用 backward 產生規則，搭配 firstDate，允許我們在前端產生一個 long stub。

In [14]: # long stub in the front

```
first_date = Date(1, 2, 2015)
effective_date = Date(15, 12, 2014)
termination_date = Date(1, 1, 2016)
schedule = Schedule(effective_date, termination_date, tenor, calendar,
                    business_convention, termination_business_convention, DateGeneration.Backward,
                    end_of_month, first_date)
pd.DataFrame({'date': list(schedule)})
```

Out[14]:

```
date
0 December 15th, 2014
1 February 2nd, 2015
2 March 2nd, 2015
3 April 1st, 2015
4 May 1st, 2015
5 June 1st, 2015
6 July 1st, 2015
```

7 August 3rd, 2015
8 September 1st, 2015
9 October 1st, 2015
10 November 2nd, 2015
11 December 1st, 2015
12 January 4th, 2016

◆使用 `nextToLastDate` 參數與 `forward` 日期產生規則，可以用來在尾端產生 long stub 的時程。

```
In [15]: # long stub at the back
```

```
    effective_date = Date(1, 1, 2015)
    penultimate_date = Date(1, 12, 2015)
    termination_date = Date(15, 1, 2016)
    schedule = Schedule(effective_date, termination_date, tenor, calendar,
                        business_convention, termination_business_convention, DateGeneration.Forward,
                        end_of_month, Date(), penultimate_date)
    pd.DataFrame({'date': list(schedule)})
```

```
Out[15]:
```

```
date
0 January 2nd, 2015
1 February 2nd, 2015
2 March 2nd, 2015
3 April 1st, 2015
4 May 1st, 2015
5 June 1st, 2015
```

6 July 1st, 2015

7 August 3rd, 2015

8 September 1st, 2015

9 October 1st, 2015

10 November 2nd, 2015

11 December 1st, 2015

12 January 15th, 2016

◆亦可由一串日期來產生 Schedule ◦

```
In [16]: dates = [Date(2,1,2015), Date(2, 2,2015), Date(2,3,2015), Date(1,4,2015), Date(1,5,2015),  
                  Date(1,6,2015), Date(1,7,2015), Date(3,8,2015), Date(1,9,2015), Date(1,10,2015),  
                  Date(2,11,2015), Date(1,12,2015), Date(4,1,2016)]
```

```
rolling_convention = Following schedule = Schedule(dates, calendar, rolling_convention)  
pd.DataFrame({'date': list(schedule)})
```

Out[16]:

date

0 January 2nd, 2015

1 February 2nd, 2015

2 March 2nd, 2015

3 April 1st, 2015

4 May 1st, 2015

5 June 1st, 2015

6 July 1st, 2015

7 August 3rd, 2015

8 September 1st, 2015

9 October 1st, 2015

10 November 2nd, 2015

11 December 1st, 2015

12 January 4th, 2016

(四)Interest Rate

◆InterestRate 類別可用來儲存利率與複利類型(compounding type)，計日方式(day count)與複利頻率(frequency of compounding)。

➤ 下例為 5.0%利率，年複利，使用 Actual/Actual 計日方式。

```
In [17]: annual_rate = 0.05
         day_count = ActualActual()
         compound_type = Compounded
         frequency = Annual
         interest_rate = InterestRate(annual_rate, day_count, compound_type, frequency)
         print(interest_rate)
```

```
Out[17]: 5.000000 % Actual/Actual (ISDA) Annual compounding
```


◆compound_factor 表示複利次數，下例為 2 次，因頻率為年，表示 2 年投資。

```
In [18]: t = 2.0  
         print(interest_rate.compoundFactor(t))  
         print((1+annual_rate)*(1.0+annual_rate))
```

```
Out[18]: 1.1025  
         1.1025
```

◆discountFactor 提供 compoundFactor 的倒數，可以用來算現值。

```
In [19]: print(interest_rate.discountFactor(t))  
         print(1.0/interest_rate.compoundFactor(t))
```

```
Out[19]: 0.9070294784580498  
         0.9070294784580498
```

◆一個利率可以使用 `equivalentRate` 方法，轉換為其他複利方式與複利頻率的利率。

```
In [20]: new_frequency = Semiannual
         new_interest_rate = interest_rate.equivalentRate(compound_type, new_frequency, t)
         print(new_interest_rate)
```

```
Out[20]: 4.939015 % Actual/Actual (ISDA) Semiannual compounding
```

◆`interest_rate` 與 `new_interest_rate` 的折現因子皆相同。

```
In [21]: print(interest_rate.discountFactor(t))
         print(new_interest_rate.discountFactor(t))
```

```
Out[21]: 0.9070294784580498
         0.9070294784580495
```

◆`InterestRate` 類別的 `impliedRate` 方法，接受 `compound factor` 為輸入，傳出 `implied rate`。
`impliedRate` 方法是靜態方法。`equivalentRate` 方法有喚起 `impliedRate` 方法，進行計算。

二、金融工具與選擇權評價引擎

(一)Setup

◆載入 QuantLib 模組，設定全域的評價日。

```
In [1]: from QuantLib import *
```

```
In [2]: today = Date(7, March, 2014)
```

```
        Settings.instance().evaluationDate = today
```

(二)金融工具(Instrument)

◆以歐式選擇權為工具的範例，建立一個選擇權契約只需要，

- 它的 payoff (一個 call 選擇權，執行價格 strike 為 100)
- 三個月後到期的歐式執行日期
- 市場資料之後傳入

```
In [3]: option = EuropeanOption(PlainVanillaPayoff(Option.Call, 100.0),  
                                EuropeanExercise(Date(7, June, 2014)))
```

(三)第一種定價方法：Black-Scholes解析公式

◆收集市場資料。

- 使用 SimpleQuote 來包裝市場資料，標的價格為 100，無風險利率 1%，波動性為 20%。

```
In [4]: u = SimpleQuote(100.0)
        r = SimpleQuote(0.01)
        sigma = SimpleQuote(0.20)
```

◆將報價資料包裝成 Black-Scholes process 的物件。

- 首先，建立水平的利率與波動性曲線。

```
In [5]: riskFreeCurve = FlatForward(0, TARGET(), QuoteHandle(r), Actual360())
        volatility = BlackConstantVol(0, TARGET(), QuoteHandle(sigma), Actual360())
```

◆然後，實例化程序

- 需要 underlying value 與 curves。
- 輸入資料儲存在 handles 中，方便之後的改變。

```
In [6]: process = BlackScholesProcess(QuoteHandle(u), YieldTermStructureHandle(riskFreeCurve),  
                                     BlackVolTermStructureHandle(volatility))
```

◆有了程序，便可建構 engine。

```
In [7]: engine = AnalyticEuropeanEngine(process)
```

◆將引擎設定給 option，便可計算。

```
In [8]: option.setPricingEngine(engine)
```

```
In [9]: print(option.NPV())
```

```
Out[9]: 4.155543462156206
```

◆根據工具與引擎，可以要求其他結果，此例為 Greeks.

```
In [10]: print(option.delta())
```

```
print(option.gamma())
```

```
print(option.vega())
```

```
Out[10]: 0.5302223303784392
```

```
0.03934493301271913
```

```
20.109632428723106
```

(四)Market changes

◆市場資料儲存於 Quote 的實例之中，當它有異動時會自行通知選擇權。

- 我們無須明白告知該選擇權需要重新計算，一旦我們重設新值給標的資產，只需再向選擇權要 NPV，便可得到異動的新價值。

```
In [11]: u.setValue(105.0)
```

```
print(option.NPV())
```

```
Out[11]: 7.27556357927846
```

◆為求顯示此效果，可畫出選擇權價值對標的資產價格的作圖。

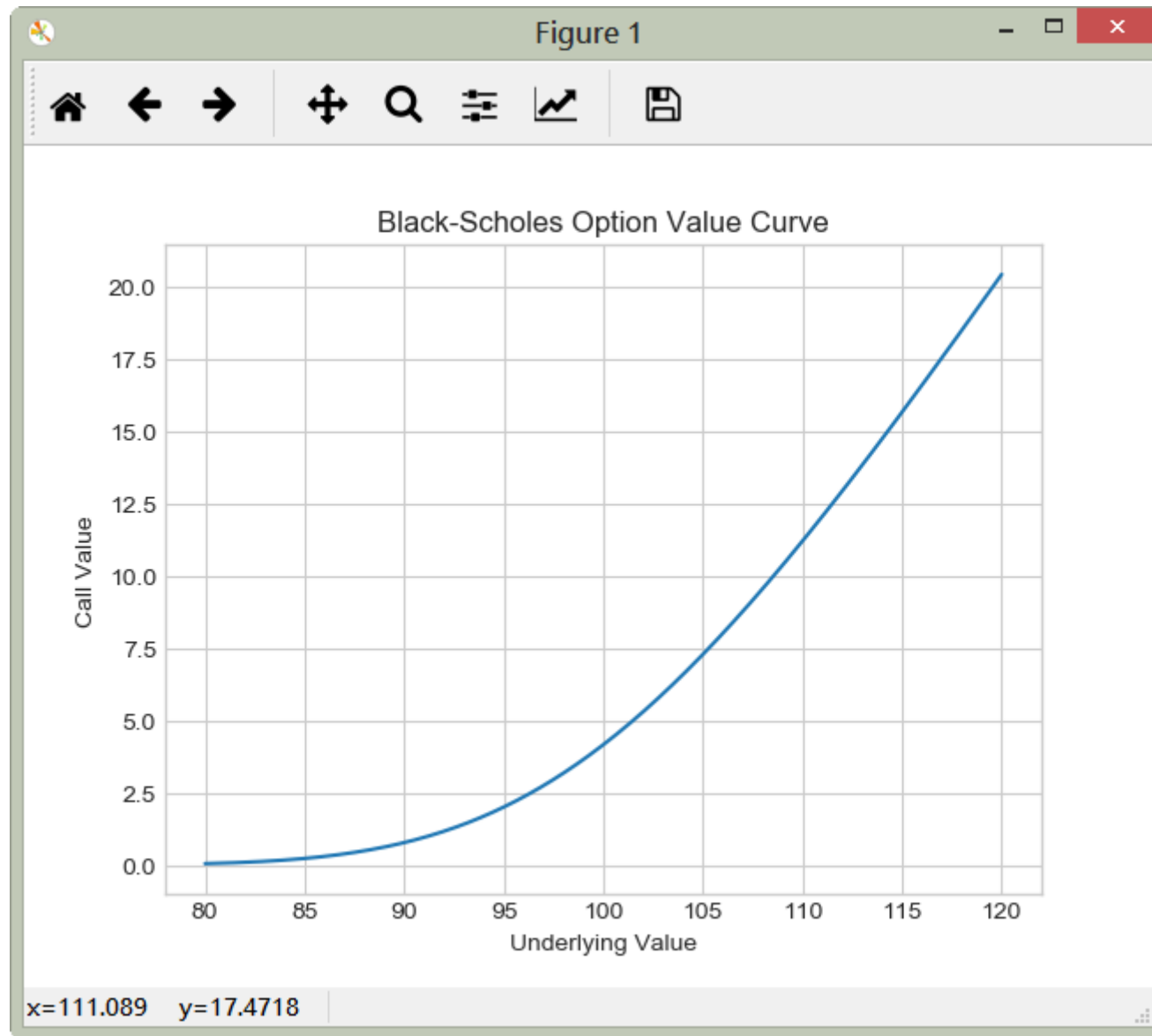
```
In [12]: %matplotlib inline
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```


◆取用 80 到 120 的陣列，將其設定標的資產的價值，收集相對應的選擇權價值，畫出結果。

```
In [13]: plt.style.use("seaborn-whitegrid")
        fig = plt.figure()
        ax = plt.axes()
        plt.title("Black-Scholes Option Value Curve")
        plt.xlabel("Underlying Value")
        plt.ylabel("Call Value")
        xs = np.linspace(80.0, 120.0, 400)
        ys = []
        for x in xs:
            u.setValue(x)
            ys.append(option.NPV())
        plt.plot(xs, ys)
        plt.show()
```



◆其他市場資料亦會影響價值。

```
In [14]: u.setValue(105.0)
         r.setValue(0.01)
         sigma.setValue(0.20)
```

```
In [15]: print(option.NPV())
```

```
Out[15]: 7.27556357927846
```

◆無風險利率的效果

```
In [16]: r.setValue(0.03)
```

```
In [17]: print(option.NPV())
```

```
Out[17]: 7.624029148527754
```

◆波動性的效果

```
In [18]: sigma.setValue(0.25)
```

```
In [19]: print(option.NPV())
```

```
Out[19]: 8.531296969971573
```

(五)Date變動

◆我們也可推進評價日，看看價值的變動。

➤ 首先，三個月後到期，標的資產 105。

```
In [20]: u.setValue(105.0)
         r.setValue(0.01)
         sigma.setValue(0.20)
         print(option.NPV())
```

```
Out[20]: 7.27556357927846
```

➤ 推進一個月評價。

```
In [21]: Settings.instance().evaluationDate = Date(7, April, 2014)
```

```
In [22]: print(option.NPV())
```

```
Out[22]: 6.560073820974377
```

◆繪圖示之。

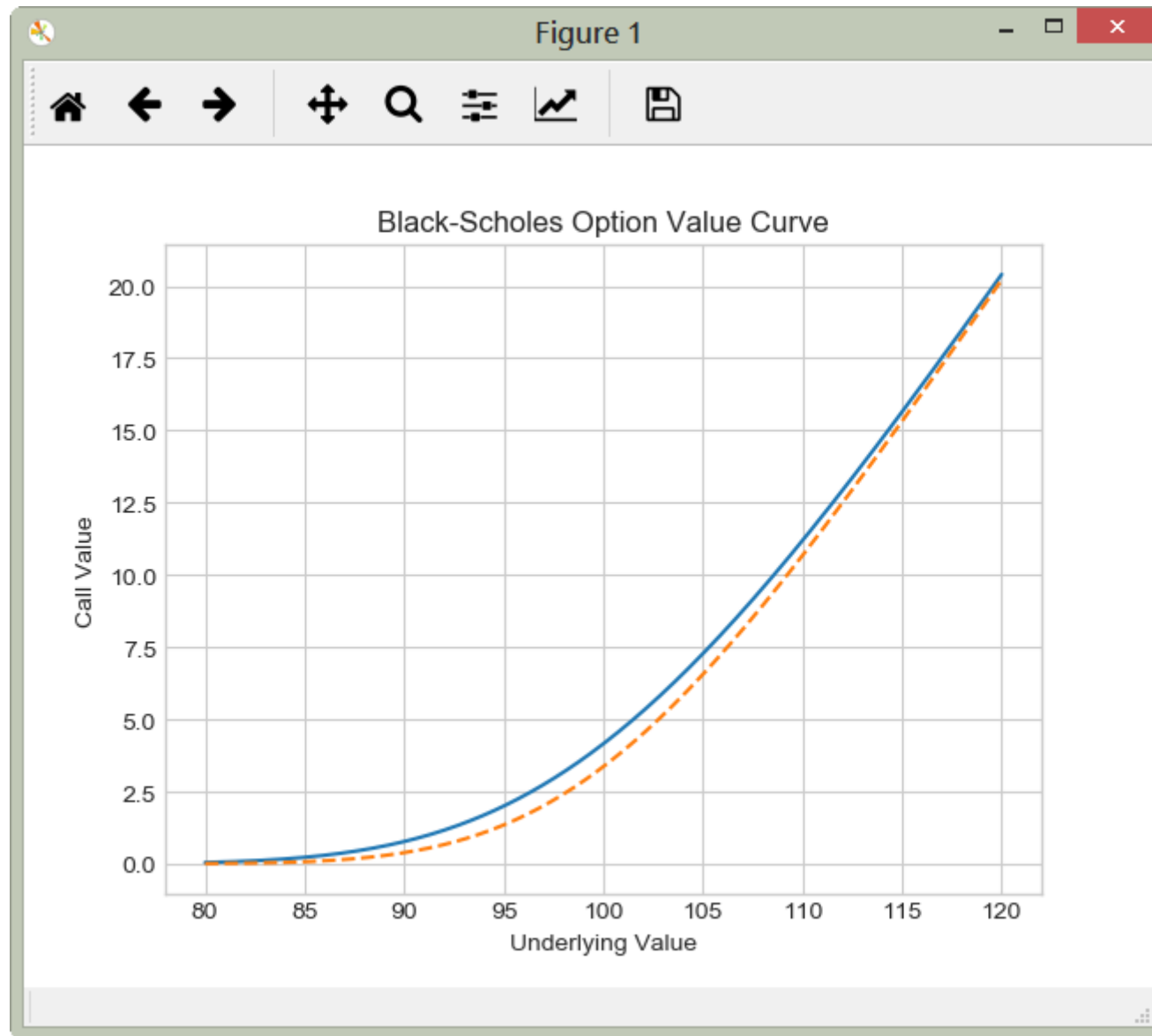
```
In [23]: ys = []  
         for x in xs:  
             u.setValue(x)  
             ys.append(option.NPV())  
         plt.plot(xs, ys, '--')  
         plt.show()
```

◆在到期日時，傳回價值為預設值：0。

```
In [24]: Settings.instance().evaluationDate = Date(7, June, 2014)
```

```
In [25]: print(option.NPV())
```

```
Out[25]: 0.0
```



(六)Other pricing methods

◆pricing-engine 機制允許我們使用不同的評價方法，首先退回原來的資料。

```
In [26]: Settings.instance().evaluationDate = today
         u.setValue(105.0)
         r.setValue(0.01)
         sigma.setValue(0.20)
```

```
In [27]: print(option.NPV())
```

```
Out[27]: 7.27556357927846
```

◆其次，使用 Heston model 來進行評價。

➤ 實例化一個相應的類別，輸入想要的參數。

```
In [28]: model = HestonModel(HestonProcess(YieldTermStructureHandle(riskFreeCurve),
         YieldTermStructureHandle(FlatForward(0, TARGET(), 0.0, Actual360()))),
         QuoteHandle(u), 0.04, 0.1, 0.01, 0.05, -0.75))
```

◆傳給對應引擎，並設定給選擇權。

```
In [29]: engine = AnalyticHestonEngine(model)
         option.setPricingEngine(engine)
```

◆根據新模型，計算選擇權 NPV。

```
In [30]: print(option.NPV())
```

```
Out[30]: 7.295356086978629
```


三、Greeks 數值計算

(一)Setup

◆導入 QuantLib 模組，設定評價日。

```
In [1]: from QuantLib import *
```

```
In [2]: today = Date(8, October, 2014)
```

```
        Settings.instance().evaluationDate = today
```

(二)較複雜的Exotic Option

◆使用 knock-in barrier option 阻隔選擇權為例。

```
In [3]: option = BarrierOption(Barrier.UpIn, 120.0, # barrier
    0.0, # rebate
    PlainVanillaPayoff(option.Call, 100.0),
    EuropeanExercise(Date(8, January, 2015)))
```

◆市場資料為 u , r , σ , 包入報價 Quote 中。

```
In [4]: u = SimpleQuote(100.0)
    r = SimpleQuote(0.01)
    sigma = SimpleQuote(0.20)
```

◆建立 flat curves 與 process，供 engine 使用。建立 term structures，使其可與評價日一並移動。

```
In [5]: riskFreeCurve = FlatForward(0, TARGET(), QuoteHandle(r), Actual360())
        volatility = BlackConstantVol(0, TARGET(), QuoteHandle(sigma), Actual360())
In [6]: process = BlackScholesProcess(QuoteHandle(u), YieldTermStructureHandle(riskFreeCurve),
        BlackVolTermStructureHandle(volatility))
```

◆最後，建立 engine(使用解析公式)設給 option。

```
In [7]: option.setPricingEngine(AnalyticBarrierEngine(process))
```

◆可以計算價格。

```
In [8]: print(option.NPV())
Out[8]: 1.3657980739109867
```

◆如果要計算 Greeks，會出現問題。

```
In [9]: print(option.delta())
```

```
Out[9]: -----
```

```
RuntimeError Traceback (most recent call last)
```

```
<ipython-input-9-dcaa26b2b456> in <module>()----> 1 print(option.delta())
```

```
/usr/local/lib/python3.6/dist-packages/QuantLib/QuantLib.py in delta(self)
```

```
11432
```

```
11433 def delta(self):
```

```
> 11434 return _QuantLib.BarrierOption_delta(self)
```

```
11435
```

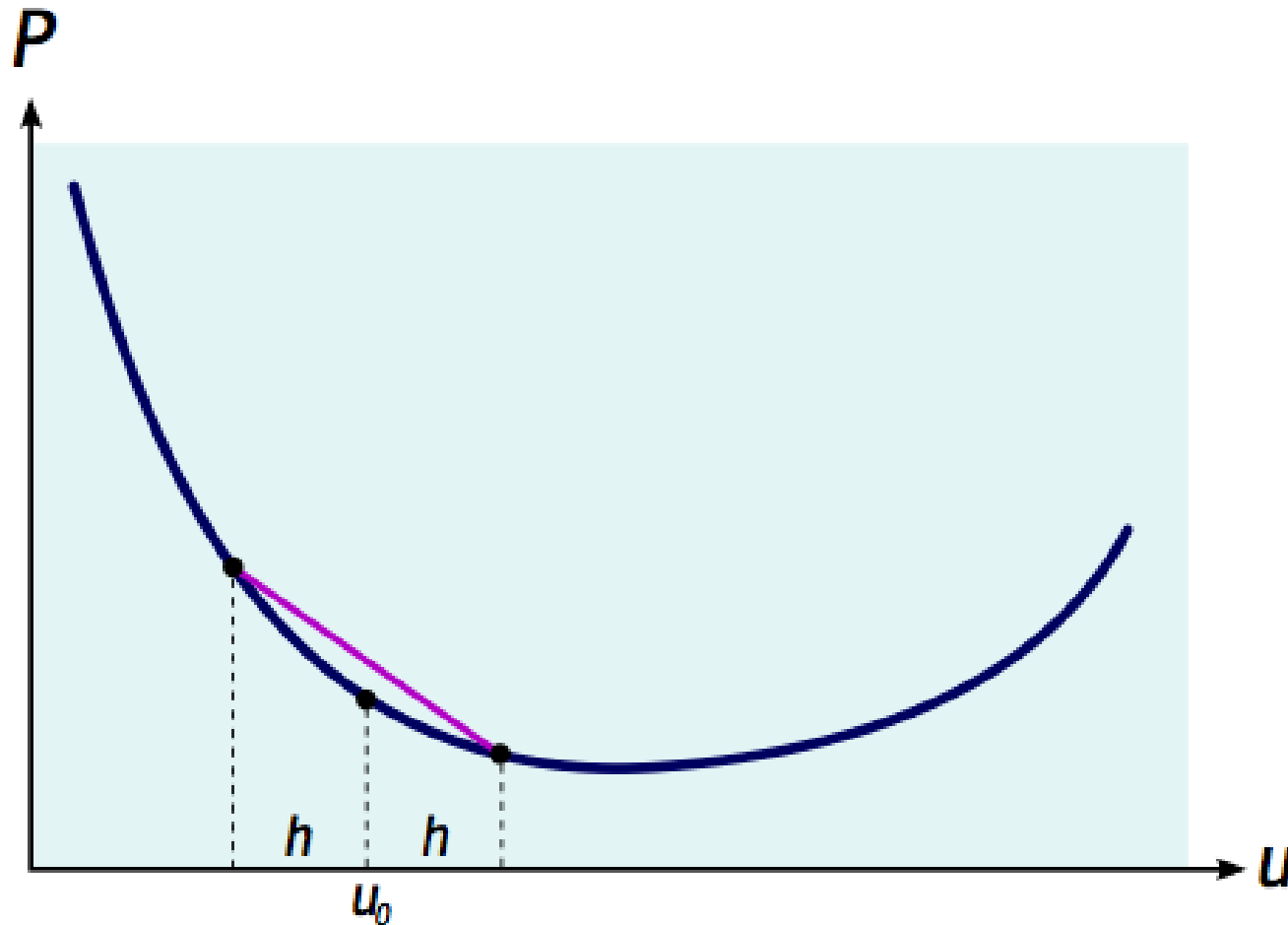
```
11436 def gamma(self):
```

```
RuntimeError: delta not provided
```

◆此 engine 不提供 delta，故產生 error。

(三) 數值計算

◆Quant 如何解決？ 使用數值微分來近似。



◆近似公式為：

$$\Delta = \frac{P(u_0 + h) - P(u_0 - h)}{2h} \quad \Gamma = \frac{P(u_0 + h) - 2P(u_0) + P(u_0 - h)}{h^2}$$

◆首先，紀錄當前價值。

```
In [10]: u0 = u.value(); h = 0.01
```

```
In [11]: P0 = option.NPV(); print(P0)
```

```
Out[11]: 1.3657980739109867
```

◆其次，增加 underlying 價格，取得價值。

```
In [12]: u.setValue(u0 + h)
```

```
        P_plus = option.NPV(); print(P_plus)
```

```
Out[12]: 1.3688112201958083
```

◆再次，減少 underlying 價格。

```
In [13]: u.setValue(u0 - h)

         P_minus = option.NPV(); print(P_minus)
```

```
Out[13]: 1.3627900998610207
```

◆最後，設定 underlying value 回目前價值。

```
In [14]: u.setValue(u0)
```

◆使用公式，計算 Greeks。

```
In [15]: Delta = (P_plus - P_minus)/(2*h)
         Gamma = (P_plus - 2*P0 + P_minus)/(h*h)

         print(Delta)

         print(Gamma)
```

```
Out[15]: 0.3010560167393761
         0.05172234855521651
```

◆此法適用於任何的 Greek ◦

$$\frac{\partial P}{\partial x} = \frac{P(x_0 + h) - P(x_0)}{h}$$

◆舉例，Rho 與 Vega：

```
In [16]: r0 = r.value(); h = 0.0001
          r.setValue(r0 + h); P_plus = option.NPV()
          r.setValue(r0)

          Rho = (P_plus - P0)/h; print(Rho)

Out[16]: 6.531038494277386

In [17]: sigma0 = sigma.value(); h = 0.0001
          sigma.setValue(sigma0 + h); P_plus = option.NPV()
          sigma.setValue(sigma0)

          Vega = (P_plus - P0)/h; print(Vega)

Out[17]: 26.52519924198904
```


◆Theta 的計算稍有不同。

```
In [18]: Settings.instance().evaluationDate = today+1  
         P1 = option.NPV()  
         h = 1.0/365  
         Theta = (P1 - P0)/h; print(Theta)
```

```
Out[18]: -10.770888399441302
```

四、市場報價與債券價格

◆此例中，將顯示如何避免當多重報價需要更新時，可能產生的陷阱。

```
In [1]: import numpy as np
```

```
In [2]: from QuantLib import *
```

```
import matplotlib.pyplot as plt
```

```
In [3]: today = Date(17, October, 2016)
```

```
Settings.instance().evaluationDate = today
```

(一)Setting the stage

◆配合 QuantLib C++範例，產生一個 Bond Curve，

➤ Nelson-Siegel model 配湊出的資料，下面為到期年限與債息利率。

```
In [4]: data = [ (2, 0.02), (4, 0.0225), (6, 0.025), (8, 0.0275), (10, 0.03), (12, 0.0325),  
                (14, 0.035), (16, 0.0375), (18, 0.04), (20, 0.0425), (22, 0.045), (24, 0.0475),  
                (26, 0.05), (28, 0.0525), (30, 0.055)]
```

◆使用相同起始日、頻率、慣例，所有債券皆價格 100。

```
In [5]: calendar = TARGET()  
        settlement = calendar.advance(today, 3, Days)  
        quotes = []  
        helpers = []  
  
        for length, coupon in data:  
            maturity = calendar.advance(settlement, length, Years)  
            schedule = Schedule(settlement, maturity, Period(Annual), calendar,  
                                ModifiedFollowing, ModifiedFollowing, DateGeneration.Backward, False)  
            quote = SimpleQuote(100.0)
```

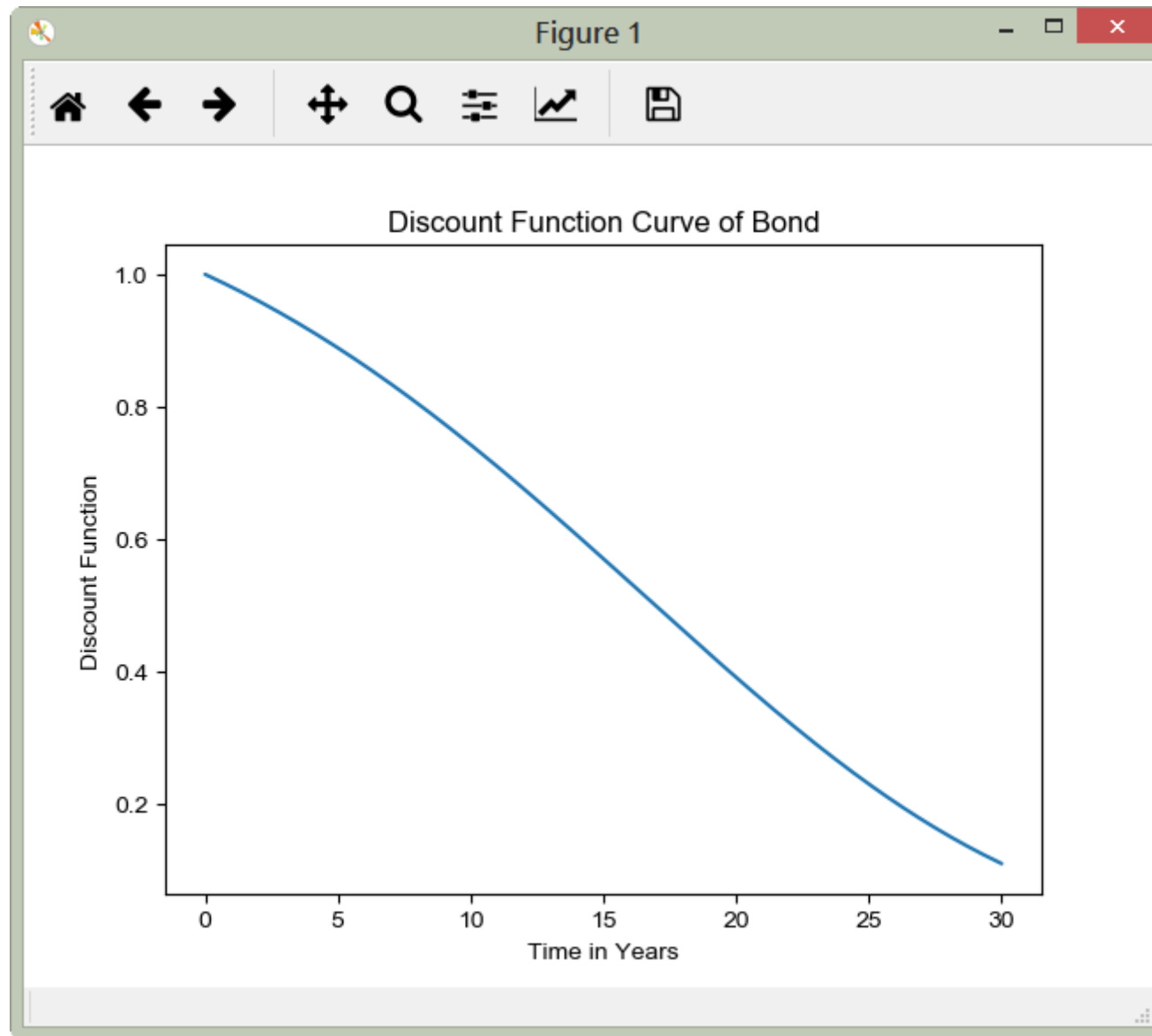
```
quotes.append(quote)
helpers.append(FixedRateBondHelper(QuoteHandle(quote), 3, 100.0, schedule, [coupon],
    SimpleDayCounter(), ModifiedFollowing))
curve = FittedBondDiscountCurve(0, calendar, helpers, SimpleDayCounter(),
    NelsonSiegelFitting())
```

◆下圖為折現因子對時間做圖，以年為單位。

```
In [6]: sample_times = np.linspace(0.0, 30.0, 301)

sample_discounts = [ curve.discount(t) for t in sample_times ]

fig = plt.figure()
plt.title("Discount Function Curve of Bond")
plt.xlabel("Time in Years")
plt.ylabel("Discount Function")
plt.style.use('seaborn-whitegrid')
xs = sample_times
ys = sample_discounts
plt.plot(xs, ys)
plt.show()
```



◆使用此 Cureve，計算一債券的價格。

➤ 三年到期，100 元面值，4% 債息，Actual/360 計息方式。

```
In [7]: schedule = Schedule(today, calendar.advance(today, 15, Years), Period(Semiannual),  
    calendar, ModifiedFollowing, ModifiedFollowing, DateGeneration.Backward, False)  
    bond = FixedRateBond(3, 100.0, schedule, [0.04], Actual360())  
    bond.setPricingEngine(DiscountingBondEngine(YieldTermStructureHandle(curve)))  
    print(bond.cleanPrice())
```

```
Out[7]: 105.77449628297312
```

(二) “It looked like a good idea at the time”

◆ 新增一個觀察者，檢查債券是否過期，如果過期，重算價格並輸出之。

➤ 在 Python 中，可定義一個被通知觸發的函數，此函數被傳遞給觀察者，再將觀察者註冊此債券。

✓ 觀察者觀察債券，一旦債券異動，觀察者被通知，然後觸發他的動作，就是建構子傳入的觸發函數。

➤ 市場報價的任何變動，會通知 the helper，然後通知 the curve，然後通知 the pricing engine，the bond，最後到我們的觀察者。

```
In [8]: prices = []

def print_price():
    p = bond.cleanPrice()
    prices.append(p)

    print(p)

o = Observer(print_price)
o.registerWith(bond)
```

◆此函數會附加新價格到 List，作為之後價格歷史之用，看其作用。

```
In [9]: quotes[2].setValue(101.0)
```

```
Out[9]: 105.77449628297312
```

```
105.8656042875337
```

◆此函數被呼叫兩次，這是由於多重繼承的毛病所造成的。Curve 發送兩次通知給工具。第一次，工具重算，但 Curve 沒有(因此價格不便)；第二次後，Curve 異動，價格改變。

➤ 此問題未來應修正，將價格改回來。

```
In [10]: quotes[2].setValue(100.0)
```

```
Out[10]: 105.8656042875337
```

```
105.77449634664224
```

◆假設市場改變，債券價格皆走高至 101。異動所有報價。

```
In [11]: prices = []
```

```
for q in quotes:
```

```
    q.setValue(101.0)
```

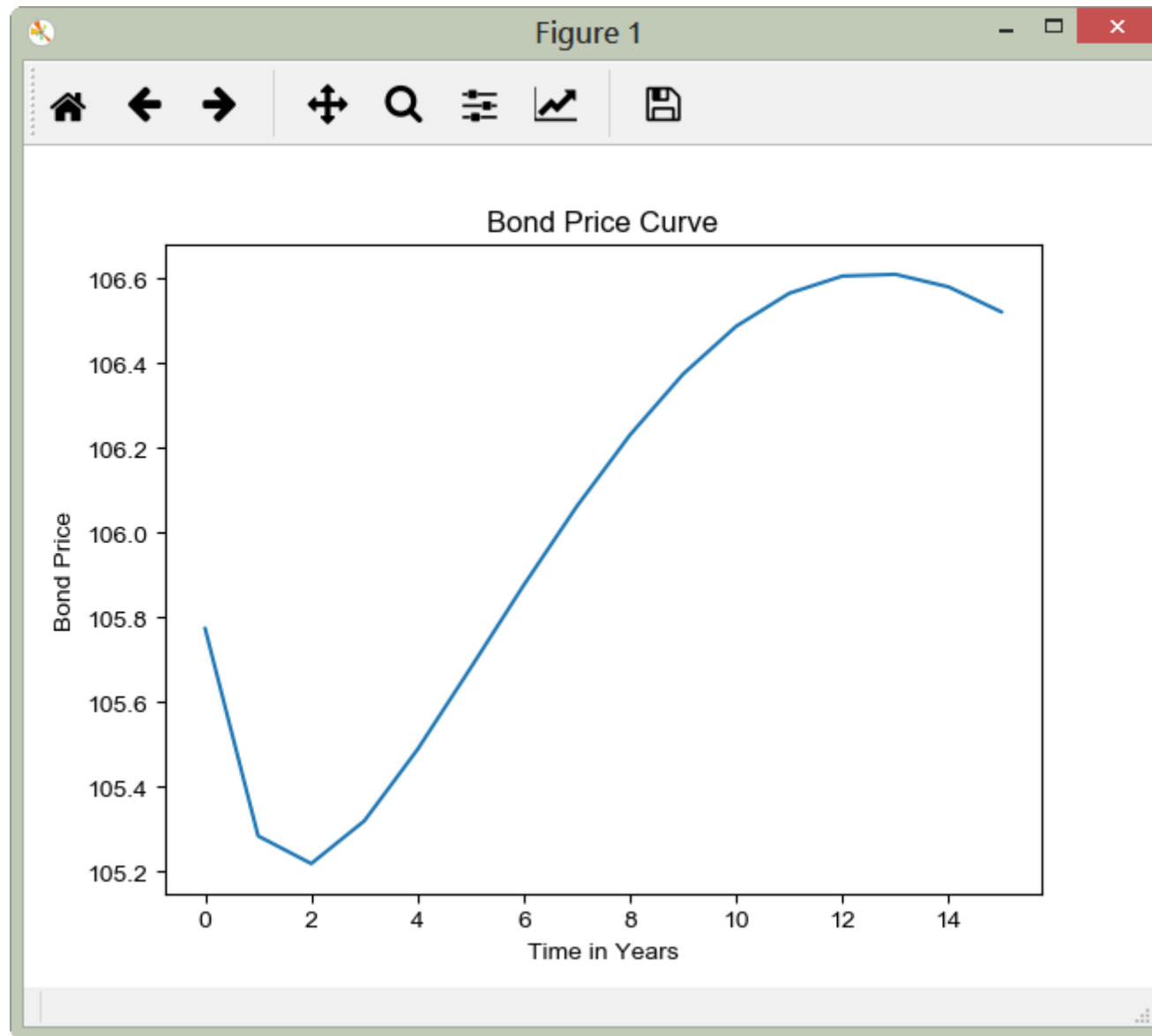


```
Out[11]: 105.77449634664224  
105.28388426272507  
105.28388426272507  
105.2186288679219  
105.2186288679219  
105.3195906444377  
105.3195906444377  
105.4878663448759  
105.4878663448759  
105.68032070200927  
105.68032070200927  
105.87580370787278  
105.87580370787278  
106.06201680440225  
106.06201680440225  
106.23044624497663  
106.23044624497663  
106.37409230798896
```

106.37409230798896
106.48708840758337
106.48708840758337
106.56505206364592
106.56505206364592
106.60570726105742
106.60570726105742
106.60980187075381
106.60980187075381
106.58011186582736
106.58011186582736
106.52070699740128

◆ 每一個異動發送通知，觸發重算。畫圖如下。

```
In [12]: unique_prices = prices[:,2]+prices[-1:]  
        plt.plot(unique_prices, '-');  
        plt.show()
```



- ◆第一個價格為原來的，最後價格為最後一個。中央的價格為不完全的變動，有些報價異動完成，有些沒有。因此那些價格是不正確的。

(三) Alternatives?

◆有些變通方法可以使用。

- 例如，暫時凍結債券，防止通知前傳。

```
In [13]: bond.freeze()
```

◆如此，不會傳遞到觀察者。

```
In [14]: for q in quotes:
```

```
    q.setValue(101.5)
```

◆復原債券，發送一個通知。

- 觸發一次重算，產生正確最後價格。

```
In [15]: bond.unfreeze()
```

```
Out[15]: 106.85839373944943
```

五、利率期限結構

(一)Setup

◆導入 QuantLib 模組，設定評價日。

```
In [1]: from QuantLib import *
```

```
In [2]: Settings.instance().evaluationDate = Date(3, October, 2014)
```

(二)明示Term Structure參考日

◆參考日是一 term structure 的起始日。它可為 evaluation date，但你也可能希望它自 spot date 開始。

◆第一種方法是有目前評價日的差距來表示，以 “two business days after the evaluation date” 定義其為 spot date；或是 “no business days” 來定義其為 evaluation date 本身。

➤ 此處藉由建立一個 swaps curve 來定義。

```
In [3]: helpers = [ SwapRateHelper(QuoteHandle(SimpleQuote(rate/100.0)),  
                                Period(*tenor), TARGET(), Annual, Unadjusted, Thirty360(), Euribor6M()),  
                    for tenor, rate in [((2,Years), 0.201), ((3,Years), 0.258), ((5,Years), 0.464),  
                                         ((10,Years), 1.151), ((15,Years), 1.588)] ]
```

➤ 注意 0 與 TARGET()引數，明示日數與使用決定營業日的日曆。

```
In [4]: curve1 = PiecewiseFlatForward(0, TARGET(), helpers, Actual360())
```

◆第二種方法事明確說明參考日期。

- 例如，ForwardCurve 類別接受特定日期向量與相對應的利率，內插這些資料。
- 第一個傳入的日期被認為是參考日，
- 由上述曲線取得節點資料，以之建立 ForwardCurve 實例。

```
In [5]: dates, rates = zip(*curve1.nodes())
```

```
In [6]: curve1.nodes()
```

```
Out[6]: ((Date(3,10,2014), 0.0019777694879293093),  
         (Date(7,10,2016), 0.0019777694879293093),  
         (Date(9,10,2017), 0.0036475517704509294),  
         (Date(7,10,2019), 0.007660760701876805),  
         (Date(7,10,2024), 0.018414773669420893),  
         (Date(8,10,2029), 0.025311634328221498))
```


◆定義：zip([iterable, ...])

`zip()` 是 Python 的一個內建函數，它接受一系列可迭代的對象作為參數，將對象中對應的元素打包成一個個 tuple (元組)，然後返回由這些 tuples 組成的 list (列表)。若傳入參數的長度不等，則返回 list 的長度和參數中長度最短的對象相同。利用 * 號操作符，可以將 list unzip (解壓)。

◆用法示例：參考下面的例子，對 zip() 函數的基本用法就可以明白了：

```
>>> a = [1,2,3]
>>> b = [4,5,6]
>>> c = [4,5,6,7,8]
>>> zipped = zip(a,b)
>>> zipped
[(1, 4), (2, 5), (3, 6)]
>>> zip(a,c)
[(1, 4), (2, 5), (3, 6)]
>>> zip(*zipped)
[(1, 2, 3), (4, 5, 6)]
```

◆由這些資料建構出的曲線，與第一條是相同的。

➤ 其參考日則是我們明示的 October 3rd (第一個傳入日期)。

```
In [7]: curve2 = ForwardCurve(dates, rates, Actual360())
```

◆兩條曲線定義於相同的日期區間。

```
In [8]: print("{0} to {1}".format(curve1.referenceDate(), curve1.maxDate()))
```

```
print("{0} to {1}".format(curve2.referenceDate(), curve2.maxDate()))
```

```
Out[8]: October 3rd, 2014 to October 8th, 2029
```

```
October 3rd, 2014 to October 8th, 2029
```

◆當我們詢問特定日期時，傳回相同利率。(例如，5 years)

```
In [9]: print(curve1.zeroRate(5.0, Continuous))
```

```
print(curve2.zeroRate(5.0, Continuous))
```

```
Out[9]: 0.452196 % Actual/360 continuous compounding
```

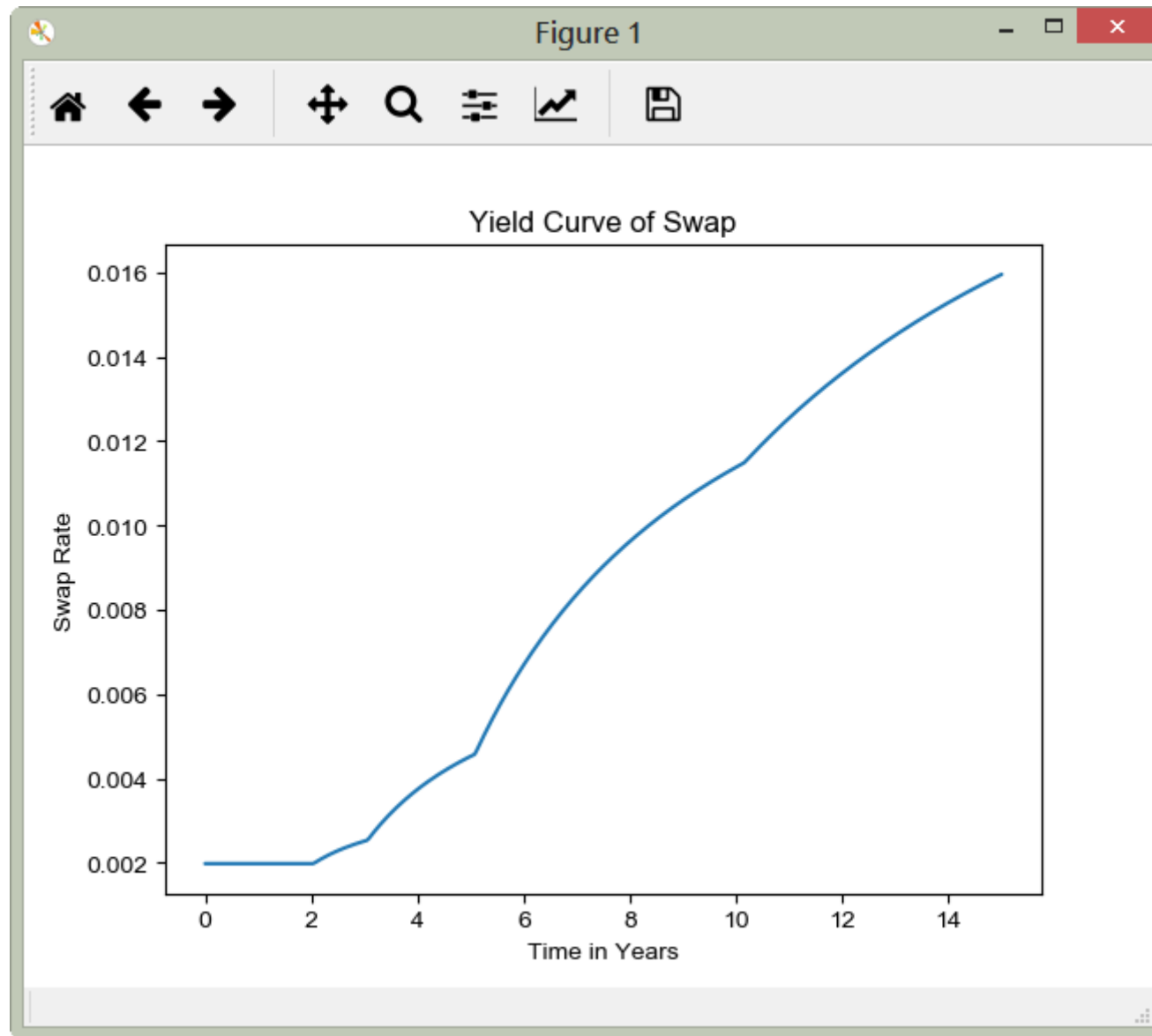
```
0.452196 % Actual/360 continuous compounding
```

◆或是特定日期。

```
In [10]: print(curve1.zeroRate(Date(7, September, 2019), Actual360(), Continuous))  
         print(curve2.zeroRate(Date(7, September, 2019), Actual360(), Continuous))  
Out[10]: 0.452196 % Actual/360 continuous compounding  
         0.452196 % Actual/360 continuous compounding
```

◆匯出完整的利率曲線。

```
In [11]: from matplotlib.ticker import FuncFormatter  
         import numpy as np  
In [12]: times = np.linspace(0.0, 15.0, 400)  
         rates = [ curve1.zeroRate(t, Continuous).rate() for t in times ]  
         _, ax = utils.plot()  
         ax.yaxis.set_major_formatter(FuncFormatter(lambda r,pos: utils.format_rate(r,2)))  
         ax.plot(times, rates);
```



(三)移動評價日

◆如果我們移動評價日，則此二曲線有何變化？

```
In [13]: Settings.instance().evaluationDate = Date(19, September, 2014)
```

◆如你所預期，第一條曲線的參考日會變動，第二條則不會。

➤ 我們可以看到第一條曲線定義的區間如何改變，第二條則否。

```
In [14]: print("{0} to {1}".format(curve1.referenceDate(), curve1.maxDate()))
```

```
print("{0} to {1}".format(curve2.referenceDate(), curve2.maxDate()))
```

```
Out[14]: September 19th, 2014 to September 24th, 2029
```

```
October 3rd, 2014 to October 8th, 2029
```

◆當然，利率也因之改變。

```
In [15]: print(curve1.zeroRate(5.0, Continuous))
```

```
print(curve2.zeroRate(5.0, Continuous))
```

```
Out[15]: 0.452196 % Actual/360 continuous compounding
```

```
0.452196 % Actual/360 continuous compounding
```

◆由於整條曲線往後移動數周，如果我們詢問一定時間後的利率，其值不變，如果詢問特定日期的利率，則自然不同。

```
In [16]: print(curve1.zeroRate(Date(7, September, 2019), Actual360(), Continuous))
```

```
print(curve2.zeroRate(Date(7, September, 2019), Actual360(), Continuous))
```

```
Out[16]: 0.454618 % Actual/360 continuous compounding
```

```
0.452196 % Actual/360 continuous compounding
```

(四)Notifications

◆最後，我們看看這兩條曲線的通知如何進行。製造兩個觀察者。

```
In [17]: def make_observer(i):  
  
    def say():  
  
        s = "Observer %d notified" % i  
  
        print('-'*len(s))  
  
        print(s)  
  
        print('-'*len(s))  
  
        return Observer(say)  
  
obs1 = make_observer(1)  
obs2 = make_observer(2)
```

◆連結到一些報價，看看是否工作正常。

```
In [18]: q1 = SimpleQuote(1.0)
         obs1.registerWith(q1)
         q2 = SimpleQuote(2.0)
         obs2.registerWith(q2)
         q3 = SimpleQuote(3.0)
         obs1.registerWith(q3)
         obs2.registerWith(q3)
```

◆如果第一個報價變動，第一個觀察者會被通知。

```
In [19]: q1.setValue(1.5)
Out[19]: -----
         Observer 1 notified
         -----
```


◆如果第二個報價變動，第二個觀察者會被通知。

```
In [20]: q2.setValue(1.9)
```

```
Out[20]: -----  
         Observer 2 notified  
         -----
```

◆如果第三個報價變動，二個觀察者都會被通知。

```
In [21]: q3.setValue(3.1)
```

```
Out[21]: -----  
         Observer 2 notified  
         -----  
         -----  
         Observer 1 notified  
         -----
```

◆將兩人分別連到兩條曲線，

```
In [22]: obs1.registerWith(curve1)
         obs2.registerWith(curve2)
```

◆評價日改變，只有第一人被告知。

```
In [23]: Settings.instance().evaluationDate = Date(23, September, 2014)
```

```
Out[23]: -----
         Observer 1 notified
         -----
```

金融研訓院 特約講師
證券暨投資分析人員合格(CSIA)
希奇資本 技術長(CTO)

董 夢 雲 財務博士

Mobil: (Taiwan)0988-065-751 (China)1508-919-2872

Email: dongmy@ms5.hinet.net

Line ID/WeChat ID: andydong3137

專長

GPU 平行運算與財務工程，C#、.Net Framework、CUDA、OpenCL、C、C++。

外匯與利率結構商品評價實務，股權與債權及衍生商品評價實務。

風險管理理論與實務，資本配置與額度規劃。

經歷

中國信託商業銀行交易室研發科主管

凱基證券風險管理部主管兼亞洲區風險管理主管

中華開發金控、工業銀行風險管理處處長

永豐金控、商業銀行風險管理處處長

永豐商業銀行結構商品開發部副總經理

著作

金融選擇權：市場、評價與策略，第二版，1997，新陸書局。

財務工程與 Excel VBA 的應用：選擇權評價理論之實作，2005，證券暨期貨發展基金會。

翻譯

衍生性金融商品與內部稽核，2003，金融研訓院。