GPU 平行運算與財務工程實作班

Heston 模型應用於結構商品之開發設計

昀騰金融科技

技術長

董夢雲 博士

dongmy@ms5.hinet.net

Part I Heston 模型與結構商品設計開發(15hrs)

一、Heston 模型介紹	案例一
二、蒙地卡羅模擬法	案例二
三、CPU 多線程的實作	案例三
四、結構商品的實例	案例四
五、結構商品的程式實作	案例五

Part II GPU 架構下的結構商品開發(15hrs)

六、GPU 與 CUDA 介紹	案例六
七、C#與 CUDA 的整合開發	案例七
八、CUDA 的變量與記憶體管理	案例八
九、CUDA 下的模擬與 cuRand 程式庫	案例九
十、GPU 版的結構商品模擬	案例十

Part I Heston 模型與結構商品設計開發

主題二 蒙地卡羅模擬法

一、古典資產模型的模擬

二、亂數的產生

三、相關性的處理

四、Heston 模型的模擬

五、實作案例二

一、古典資產模型

(一)Black-Scholes 對資產行為的假設

◆ Black-Scholes 模型之下股票價格變化的程序

➢ 金融資產價格的假設是它遵行著所謂的擴散程序(diffusion process)

$$dS/S = \mu \cdot dt + \sigma \cdot dZ$$

✓ $\dfrac{dS}{S} = \dfrac{S_{t+dt} - S_t}{S_t}$ ＝金融資產的報酬率，

✓ dt ＝單位時間，

✓ μ ＝單位時間內預期金融資產的報酬率，

✓ σ ＝單位時間內預期金融資產的標準差。

◆ Z＝一隨機變數，為平均數為零，變異數為 t 之常態分配，$Z \sim \Phi(0,t)$。

➢ Z 稱之為韋恩程序。

➢ dZ ＝單位時間內，Z 的變動量，為一期望值為零，變異數為 dt 之常態分配，$dZ \sim \Phi(0,dt)$。

(二)資產價格路徑的模擬

- 現代財務模型大都以連續交易作為分析的架構，亦即，交易是連續進行，兩次交易間的時間間隔為無限小，因此稱之為連續時間財務。
  - 當我們要進行模擬時，必須將之離散化。
  - 在實際進行模擬時，只能以有限的步數，模擬期末資產可能的價格。
  - 每次模擬的時間跨距(Time Interval)是有限的，而非無限小。
- 這種以有限間隔的模擬實作，取代模型中間隔無限小的假設，稱之為離散化(Discretization)。
  - 實務最常使用的是尤拉法(Euler Schemes)。
  - 另外尚有兩個較為精細的方法，分別是 Milstein Schemes 與二階法(Second-Order Method)，
    - ✓ 由於效率上的考量，實務上使用的機會不大。

(三)尤拉法

◆ 以傳統的 Black-Scholes 模型為例，

$$dS = r \bullet S \bullet dt + \sigma \bullet S \bullet dZ \quad \text{..........(1.1)}$$

➢ 最簡單的離散化為尤拉法的離散化，

$$\Delta S = r \bullet S \bullet \Delta t + \sigma \bullet S \bullet \Delta Z$$

$t_2 = t_1 + \Delta t$ ， $Z_2 = Z_{t_2}$ ， $Z_2 = Z_1 + \Delta Z$ ， $S_2 = S_{t_2}$ ， $S_2 = S_1 + \Delta S$

➢ 因此，可得下面的迭代模擬方程式。

$$S_2 - S_1 = r \bullet S_1 \bullet (t_2 - t_1) + \sigma \bullet S_1 \bullet (Z_2 - Z_1) \quad \text{..........(1.2)}$$

◆ (1.1)式可以改寫如下，

$$\frac{dS}{S} = r \bullet dt + \sigma \bullet dZ \quad \text{......(1.3)}$$

➢ (1.3)式的隨機微分方程式，在給定期初資產價格 S_0 下，可以求得期末價格 S_T 的移轉方程式如下，

$$S_T = S_0 \exp\left(\left(r - \tfrac{1}{2}\sigma^2\right) \bullet T + \sigma\sqrt{T} \bullet \varepsilon\right) \quad \text{......(1.4)}$$

$$\varepsilon \sim N(0,1)$$

➢ 由於，(1.4)式是公式解，因此不論時間跨距長短，都可以直接用來模擬期末價格。

- (1.1)式的差分方程式，在給定期初資產價格 S_1 下，可以求得期末價格 S_2 的移轉方程式如下，

$$S_2 = S_1 \exp\left(\left(r - \frac{1}{2}\sigma^2\right) \bullet \Delta t + \sigma\sqrt{\Delta t} \bullet \varepsilon\right) \qquad \text{(1.5)}$$

  - 通常，我們以一天走一步的方式來進行模擬。
  - 到期時間為一年，則一條模擬的路徑要走 365 步。
  - 十萬條的路徑應該是可以接受的模擬量。

二、亂數的產生

- 程式中，需要亂數時，通常可以由內建程式庫中取得亂數。
  - 這類亂數可稱之為假亂數(Pseudo-Random Number)。
    - ✓ 這是因為我們是採用確定的方法(Deterministic Method)去模仿產生亂數，
    - ✓ 本質上他們並不是真正的亂數，只是刻意地使其看起來像是亂數而已。
    - ✓ 他們具備了一些亂數該有的性質。
  - 另一類在程式設計中可充當亂數來源的是所謂的準亂數(Quasi-Random Number)，
    - ✓ 這類亂數在本質上就不是以模仿亂數的方式去產生，
    - ✓ 他們是以數論的理論去產生均勻分佈的數列(Low-discrepancy Sequence)，

- 事實上，如果要產生”真正”的亂數，那還是要求助於大自然，
  - 原子核衰變過程中，放射出來的粒子的方位，便是一個天然的亂數源。
    - 網路上有網站提供(氪 85)產生的數列，這是一個有趣的網站。http://www.fourmilab.ch/hotbits/
  - Toshiba 使用半導體上的熱騷動(Thermal Noise)，以 PCI 板方式販售 Random Master。
    - 可相容用於 PC 與超級電腦。
    - 使用磁碟機轉動產生的空氣亂流(Air Turbulence)。

(一)亂數產生邏輯

- 電腦使用的假亂數通常是以迭代的方式去產生，我們可以表示如下，

$$x_{i+1} = f(x_i) \text{ ， } x_0 = seed \quad \text{.....(2.1)}$$

  - 此假亂數都是以均等分配亂數的方式產生的。
  - 由於電腦的離散性質，上式中的數值 x_i 原則上都是整數，而起始值 x_0 由外生給定。
  - 我們可以 1234 為起始種子，初始化亂數物件。

- 以 C#的亂數物件為例，
  - 當我們呼叫 Next()方法，產生的亂數，是介於 0 與 maxValue 之間的整數。
  - maxValue 是 32 位元整數的最大值，2,147,483,647，以十六進位表示為 7FFFFFFF。
  - 將此產生的亂數除以 maxValue 便可產生範圍介於[0, 1]的均等亂數。成為我們產生其他亂數的來源。

- 實務上，線性同餘法是最為常見的亂數產生器，其迭代的公式如下，

$$x_{i+1} = (ax_i + c) \mod m \quad \text{...(2.2)}$$

  - 其中，a, c, m 都是正常數，mod 表取餘數運算。
    - ✓ 在著名的 IBM 360 系統中，常數選擇如下，
    - ✓ $a = 7^5 = 16807, c = 0, m = 2^{31} - 1 = 2147483674$
  - 讀者可以根據(2.2)式自行撰寫自己的產生器。

(二)Mersenne Twister 亂數產生器

- 一個良好的亂數產生器，應該具備下面三項性質，
  - 產出的亂數要能均勻分佈，
  - 亂數數列的週期要夠長，
  - 產生的速度要夠快。

- 均勻分佈
  - 理論上的均等分佈亂數，要能均勻的分佈於範圍之內，
    - ✓ 對於一維的亂數，這應該不是太大的問題。
    - ✓ 然而，如果我們要的是 100 維的亂數，則演算法的品質就可能有差異了。
    - ✓ 一些演算法在高維度時，會出現群聚(Clustering)的現象，造成高維空間分佈不均勻的現象。
  - 通過一些統計檢定，我們可以判斷亂數均勻分佈的性質。
    - ✓ 建議有興趣的讀者，可以參考 Knuth Donald 教授的巨著，The Art Of Computer Programming, Vol. 2: Semi-numerical Algorithms, Addison-Wesley, Reading, MA, 2^{nd} Ed, 1981。
    - ✓ 計算機程序設計的藝術，第 2 卷半數值算法，蘇運霖，國防工業出版社，2002。

長週期

- 由(2.1)的迭代式可知，亂數的產生是一個有序的過程，最終將會回到源頭，自我重複。
  - ✓ 然而，重點是多久後會自我重複，也就是亂數的週期有多長。
- 在現代的商品模擬計算中，抽取上億個亂數是相當常見的。
  - ✓ 如果亂數週期不能達到十億以上的數量級，則模擬的品質自也堪慮。

速度快

- 由於抽取上億個亂數是相當常見的，如果演算法的計算效率不高，則實用價值也就不大。
  - ✓ 通常為求加速，可以考慮盡可能使用位元運算的方式，來完成演算法。
  - ✓ 有時使用組合語言來撰寫程式碼，以求性能的要求。

Mersenne Twister Home Page

A very fast random number generator
Of period $2^{19937}-1$

Japanese Version

News:

- TinyMT is released. (2011/06/20)
- MTGP is released(2009/11/17)
- we released SIMD-oriented Fast Mersenne Twister (SFMT). SFMT is roughly twice faster than the original Mersenne Twister, and has a better equidistibution property, as well as a quicker recovery from zero-excess initial state. Click here! (2007/1/31)

If you send an email when you use Mersenne Twister (MT), it will encourage us - even if we are unable to reply to all the messages we receive.
Please let us know when you find any problems.
m-mat @ math.sci.hiroshima-u.ac.jp (remove the spaces adjacent to @. I hate 100 SPAMs everyday.)

Caution: Mersenne Twister is basically for Monte-Carlo simulations - it is not cryptographically secure "as is". Please read FAQ.

- Before asking us questions, please first take a look into FAQ.
- What/how is Mersenne Twister?
- 2002-version (mt19937ar.c): explanations and C-codes (Newest at present, 2004/3.)
- MT for 64-bit machines (2005/2/23)
- license (2001/4/6)
- Various versions/languages links/codes

- 實務上，一個在過去 20 年頗受歡迎的演算法，已逐漸為各個主要數值程式庫所採用。
  - 日本學者 Makoto Matsumoto(松本真)與 Takuji Nishimura(西村拓士)於 1997 年發表的 Mersenne Twister (MT)亂數產生器。
    - ✓ MT 已實作於 SPSS 與 SAS 軟體中，被認為是較可信賴的產生器。
    - ✓ Boost C++ Library，GNU Scientific Library，NAG Numerical Library，Matlab，GAUSS，Python 與其他軟體都已實作 MT。

- 一般使用的 MT 亂數產生器產生 32 位元的整數數列，它具有下面的優點。
  - 首先，它具有非常長的周期，$2^{19937}-1$，足以滿足一般的使用。
    - ✓ 這也是我們通常稱其為 MT19937 的原因，一般早期的亂數產生器週期多為 2^{32}。
  - 其次，MT 可以達到 623 維的均勻亂數分佈。
  - 第三，MT 通過許多隨機性的統計測試。

- MT 的一些缺點，
  - 首先，演算法中有很大的狀態空間，其狀態變數位元數很長，造成 CPU 快取負荷很大。
  - 其次，MT 的計算速率不高，除非使用平行運算版本 SFMT。
  - 第三，MT 並沒有通過最嚴格的隨機性統計測試，TestU01。
- MT 已有多個語言的實作版本，下面這個 C#版本是由 Mitil Ooyama 所改寫，網址為

  http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/VERSIONS/C-LANG/mt19937ar.cs。

◆ MT19937 Main Page。

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

Papers on random number ge | What & how is MT?

www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/ewhat-is-mt.html

What is Mersenne Twister (MT)?

Mersenne Twister(MT) is a pseudorandom number generating algorithm developped by Makoto Matsumoto and Takuji Nishimura (alphabetical order) in 1996/1997. An improvement on initialization was given on 2002 Jan.
MT has the following merits:

- It is designed with consideration on the flaws of various existing generators.
- The algorithm is coded into a C-source downloadable below.
- Far longer period and far higher order of equidistribution than any other implemented generators. (It is proved that the period is 2^19937-1, and 623-dimensional equidistribution property is assured.)
- Fast generation. (Although it depends on the system, it is reported that MT is sometimes faster than the standard ANSI-C library in a system with pipeline and cache memory.) (Note added in 2004/3: on 1998, usually MT was much faster than rand(), but the algorithm for rand() has been substituted, and now there are no much difference in speed.)
- Efficient use of the memory. (The implemented C-code mt19937.c consumes only 624 words of working area.)

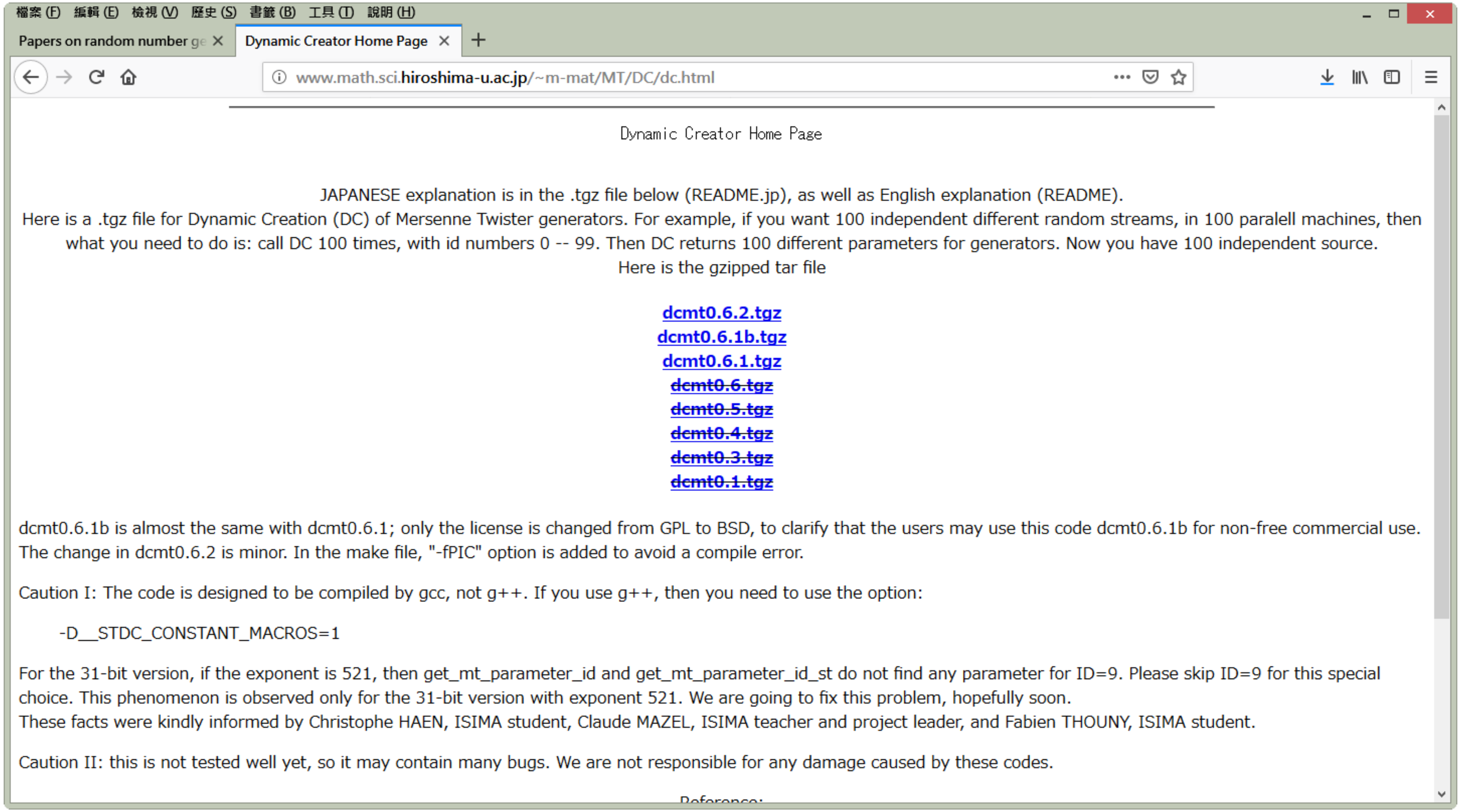
Asks and acknowledgements
The origin of the name MT

Why MT? More Reasons

- Mathematical/Computational reasons
- Reputations by some researchers
- Reputations by some users

Return to MT's homepage

➢ MT19937 有多個進階改良版，DCMT：Dynamic Create MT



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

Papers on random number ge × | Dynamic Creator Home Page ×

www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/DC/dc.html

Dynamic Creator Home Page

JAPANESE explanation is in the .tgz file below (README.jp), as well as English explanation (README).
Here is a .tgz file for Dynamic Creation (DC) of Mersenne Twister generators. For example, if you want 100 independent different random streams, in 100 paralell machines, then what you need to do is: call DC 100 times, with id numbers 0 -- 99. Then DC returns 100 different parameters for generators. Now you have 100 independent source.
Here is the gzipped tar file

dcmt0.6.2.tgz
dcmt0.6.1b.tgz
dcmt0.6.1.tgz
~~dcmt0.6.tgz~~
~~dcmt0.5.tgz~~
~~dcmt0.4.tgz~~
~~dcmt0.3.tgz~~
~~dcmt0.1.tgz~~

dcmt0.6.1b is almost the same with dcmt0.6.1; only the license is changed from GPL to BSD, to clarify that the users may use this code dcmt0.6.1b for non-free commercial use.
The change in dcmt0.6.2 is minor. In the make file, "-fPIC" option is added to avoid a compile error.

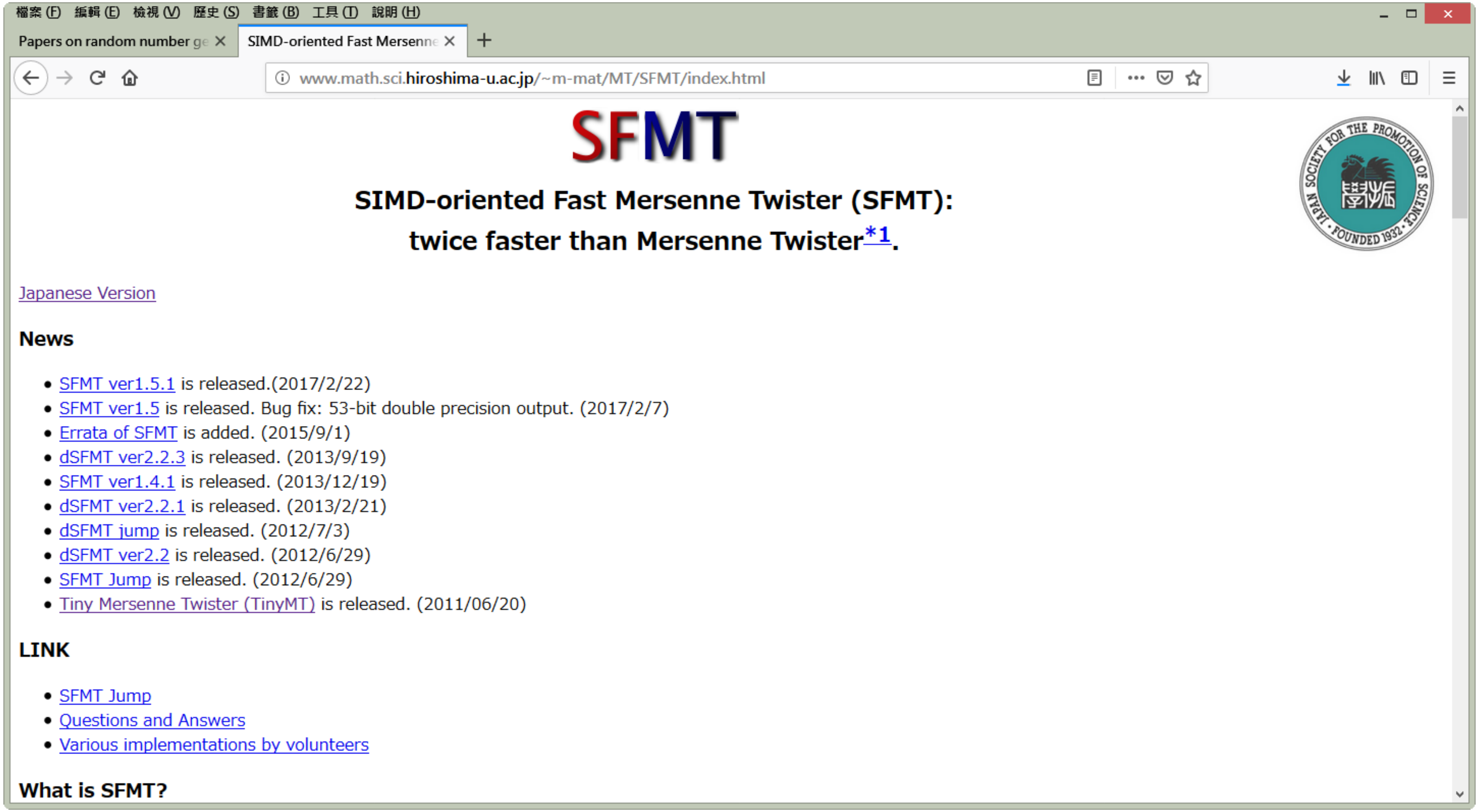
Caution I: The code is designed to be compiled by gcc, not g++. If you use g++, then you need to use the option:

-D__STDC_CONSTANT_MACROS=1

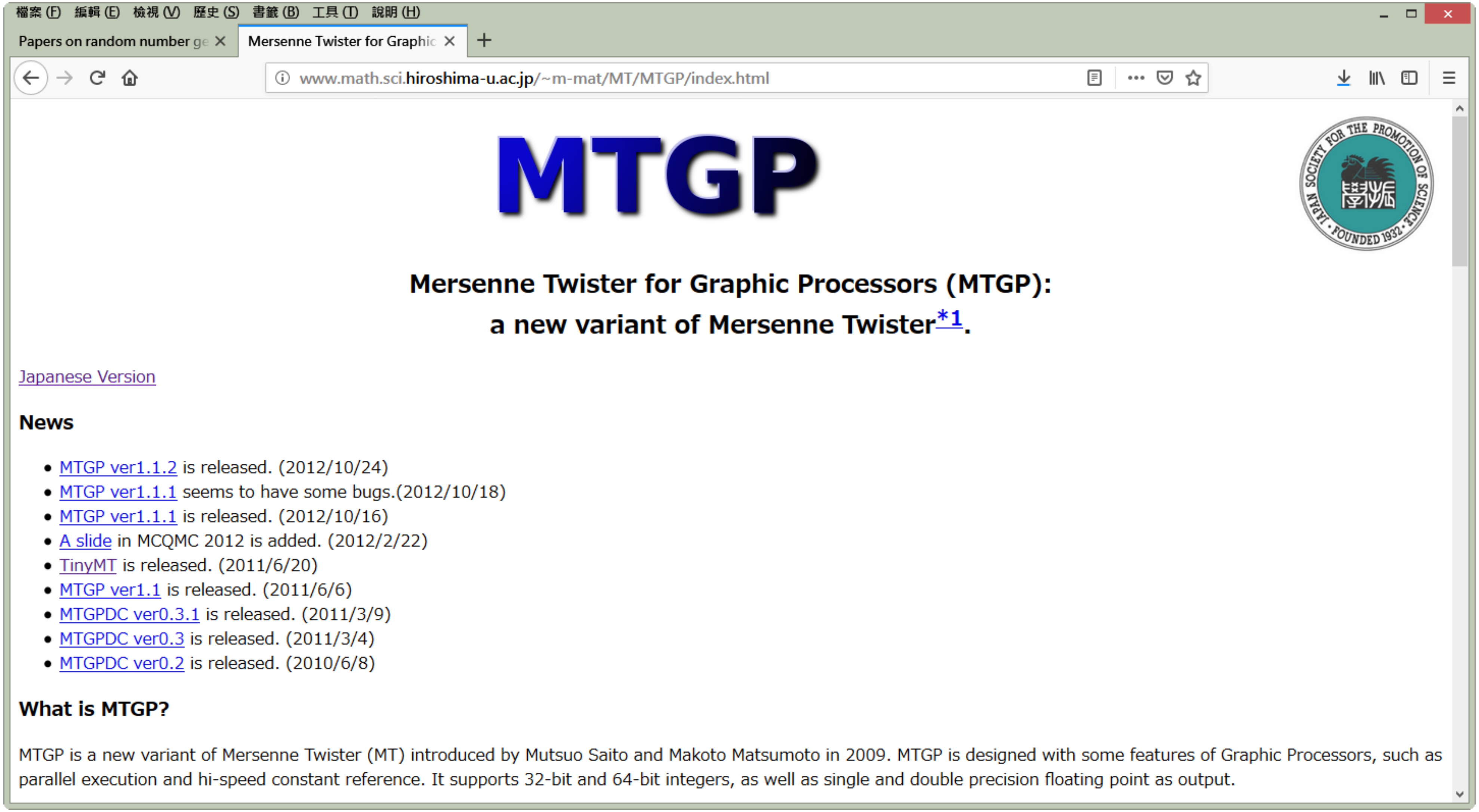
For the 31-bit version, if the exponent is 521, then get_mt_parameter_id and get_mt_parameter_id_st do not find any parameter for ID=9. Please skip ID=9 for this special choice. This phenomenon is observed only for the 31-bit version with exponent 521. We are going to fix this problem, hopefully soon.
These facts were kindly informed by Christophe HAEN, ISIMA student, Claude MAZEL, ISIMA teacher and project leader, and Fabien THOUNY, ISIMA student.

Caution II: this is not tested well yet, so it may contain many bugs. We are not responsible for any damage caused by these codes.

◆ SFMT：SIMD-Fast MT



◆ MTGP：MT for Graphic Processor



➢ MTGPDC：Dynamic Create MT for Graphic Processor

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

Papers on random number ge | Mersenne Twister for Graphic

www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MTGP/index.html#mtgpdc

MTGP Dynamic Creator (MTGPDC)

MTGPDC creates parameter sets for MTGP. MTGP has 128 parameter sets for each period. If you need more parameter sets, use MTGPDC to create them. MTGPDC can create 2^{32} parameter sets for each period.

MTGPDC can create parameter sets of MTGPs which have period of 2^{3217},2^{4423},2^{11213}, 2^{23209} or 2^{44497}.

MTGPDC requires some libraries.

- Victor Shoup's NTL: A Library for doing Number Theory.
- C++ TR1 Libraries. (g++ ver. 4.0 or later)
- Boost C++ Libraries.
- OpenSSL.

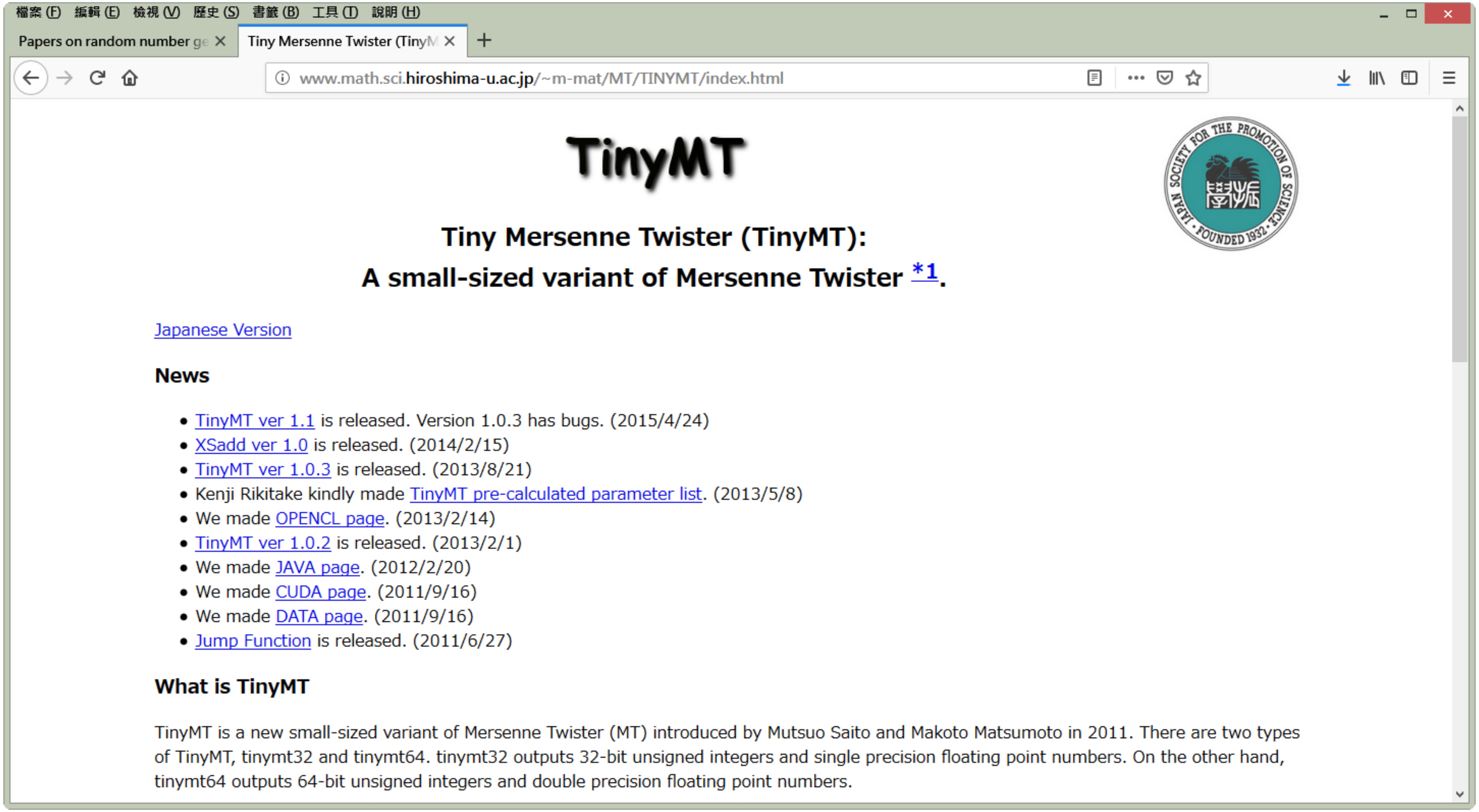
And MTGP requires UNIX like Operating System with GNU C++ compiler(g++).

Download MTGP

This code is released on March 26th in 2010, so it may contain some bugs. Any feedback is welcome (send an email to Mutsuo Saito, saito **"at sign"** math.sci.hiroshima-u.ac.jp and m-mat **"at sign"** math.sci.hiroshima-u.ac.jp)

version	archive	changes
new! 0.3.1	MTGPDC-src-0.3.1.zip MTGPDC-src-0.3.1.tar.gz	Fix the bug in mtgp64dc mentioned below. (2011/3/9)
	MTGPDC-	A small bug in search_temper.hpp is fixed. The bug effects only on the least significant 5 bits, but it makes the dimension of equidistribution of the LSBs smaller than they should be. In addition, we modified the algorithm of searching tempering parameters, so that the algorithm tempers the dimensions of equidistribution of

◆ TinyMT：Tiny(Small Size) MT


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

Papers on random number ge ×   Tiny Mersenne Twister (TinyM ×

www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/index.html

TinyMT

Tiny Mersenne Twister (TinyMT): A small-sized variant of Mersenne Twister *1.

Japanese Version

News

- TinyMT ver 1.1 is released. Version 1.0.3 has bugs. (2015/4/24)
- XSadd ver 1.0 is released. (2014/2/15)
- TinyMT ver 1.0.3 is released. (2013/8/21)
- Kenji Rikitake kindly made TinyMT pre-calculated parameter list. (2013/5/8)
- We made OPENCL page. (2013/2/14)
- TinyMT ver 1.0.2 is released. (2013/2/1)
- We made JAVA page. (2012/2/20)
- We made CUDA page. (2011/9/16)
- We made DATA page. (2011/9/16)
- Jump Function is released. (2011/6/27)

What is TinyMT

TinyMT is a new small-sized variant of Mersenne Twister (MT) introduced by Mutsuo Saito and Makoto Matsumoto in 2011. There are two types of TinyMT, tinymt32 and tinymt64. tinymt32 outputs 32-bit unsigned integers and single precision floating point numbers. On the other hand, tinymt64 outputs 64-bit unsigned integers and double precision floating point numbers.

(三)Quasi-Random Number

- 準亂數又稱之為低差異數列(Low-discrepancy Sequence, LDS)，
  - 此數列中的點，落於特定區域 B 的比率，與該區域 B 的空間測度成正比。
  - 空間測度可以有不同的定義，
    - ✓ 但在一維空間中，常用測度為距離；
    - ✓ 在二維空間中，常用測度為面積；
    - ✓ 在三維空間中，常用測度為體積。
  - 此一性質也可見於均勻分佈數列之中。
  - 產生 LDS 的方法有很多種，下面舉兩種方式為例說明之。

◆ Halton 數列

- 對任意 k≧1，可以用唯一的方式，如下表示之，

$$k = a_0 + a_1 m + a_2 m^2 + \ldots + a_r m^r$$

  - ✓ 其中，a_i 為整數，$a_i \in [0, m-1]$ 。
  - ✓ r 滿足下面條件，$m^r \le k < m^{r+1}$ 。

- 定義 Radical Inverse Function in Base m 如下，

$$\phi_m(k) \equiv a_0 m^{-1} + a_1 m^{-2} + a_2 m^{-3} + \ldots + a_r m^{-(r+1)}$$

  - ✓ 上式為一個介於[0, 1)的有理數，藉由將 k 反射，以 m 為基底之分數而產生之。

- 我們可以如下定義一個 d 維之 Halton Points，

$$z_k \equiv \left(\phi_{p1}(k), \phi_{p2}(k), \ldots, \phi_{pd}(k)\right), \quad k \ge 1$$

  - ✓ 其中，$p_1, p_2, \ldots, p_d$ 為前 d 個質數。

◆ Halton 範例

➢ 以 2,3,5,7 為質數之四維 Halton 數列，第 37 個點由下產生，

Base	$n=37_{10}$		
2	100101_2	0.101001_2	=0.640625
3	1101_3	0.1011_3	=0.382716
5	122_5	0.221_5	=0.488000
7	52_7	0.25_7	=0.387755

◆ Sobol 數列

- 首先，針對每一維度，需要一組方向數(direction number) v_i 與一個質數多項式。
  - ✓ 令 v_i 為二元分數，可表示為

$$v_i = \frac{m_i}{2^i}, \quad i = 1,2,\ldots,\text{MAXBIT}$$

  - ✓ 選擇正確的起始 v_i 是非常重要的。對 m_i 的正式要求只有 mi 為奇數，且

$$0 < m_i < 2^i, \quad i < \text{MAXBIT} \sim 30$$

  - ✓ 在 Sobol 產生器中 MAXBIT = 30。

- 每一維度需要一個質數多項式如下，

$$P \equiv x^q + a_1 x^{q-1} + \ldots + a_1 x^{q-1} + 1, \quad a_i \in \{0,1\}$$

- 對每一多項式，需要 q 個起始方向數 v_i。
  - ✓ 一但起始方向數 v_i 選好後，v_i 可由下是遞迴產生。

$$v_i = a_1 v_{i-1} \oplus a_2 v_{i-2} \oplus \ldots \oplus a_{q-1} v_{i-q+1} \oplus v_{i-q} \oplus \left[\frac{v_{i-q}}{2^q} \right], \quad i > q$$

  - ✓ 其中，$\oplus$ 為位元 XOR 運算，上式亦可表示為

$$m_i = 2a_1 m_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus \ldots \oplus 2^{q-1} a_{q-1} m_{i-q+1} \oplus 2^q m_{i-q} \oplus m_{i-q}$$ 。

- 在求得所有的 vi 後，可由下式產生 Sobol 數列，

$$x^{n+1} = x^n \oplus v_c, \quad x^0 = 0$$

  - ✓ c 為 n 以二元表示下，最右方 0 位元的位置。

◆ Sobol 範例

- 以下面質數多項式產生第四維之 Sobol 數列，

  $P(x)=x^3+x^2+1$，$q=3, a_1=1, a_2=0$ 。

- 前三個方向數如下表

i	1	2	3
m_i	1	1	5
v_i(Binary)	0.1	0.01	0.101

- 可得

$$m_i=2m_{i-1}\oplus 8m_{i-3}\oplus m_{i-3}$$

$$m_4=2m_3\oplus 8m_1\oplus m_1=10\oplus 8\oplus 1=1010\oplus 1000\oplus 0001=0011_2=3_{10}$$

$$m_5=2m_4\oplus 8m_2\oplus m_2=6\oplus 8\oplus 1=0110\oplus 1000\oplus 0001=1111_2=15_{10}$$

$$m_6=2m_5\oplus 8m_3\oplus m_3=30\oplus 40\oplus 5=11110\oplus 101000\oplus 101=110011_2=51_{10}$$

- 可得

i	4	5	6
m_i	3	15	51
v_i(Binary)	0.0011	0.01111	0.110011

- 可得

$x^0 = 0,\ n = 0_2,\ c = 1$

$x^1 = x^0 \oplus v_1 = 0.0 \oplus 0.1 = 0.1_2 = 0.5,$

$n = 01_2,\ c = 2$

$x^2 = x^1 \oplus v_2 = 0.1 \oplus 0.01 = 0.11_2 = 0.75,$

$n = 10_2,\ c = 1$

$x^3 = x^2 \oplus v_1 = 0.11 \oplus 0.1 = 0.01_2 = 0.25,$

$n = 11_2,\ c = 3$

$x^4 = x^3 \oplus v_3 = 0.01 \oplus 0.011 = 0.111_2 = 0.875,$

$n = 100_2,\ c = 1$

$x^5 = 0.375$

$x^6 = 0.125$

$x^7 = 0.625$

- 實務上，LDS 數列的主要缺點，
  - 在高維度時會發生群聚的現象，使的亂數分佈產生不均勻的結果。
  - 相對 Halton 數列，Sobol 數列的品質較佳。
    - ✓ 至 256 維皆可均勻分布。

(四)常態分配亂數的產生

◆ Proposition

Let U be a uniform (0,1) random variable. For any continuous distribution function F, the random variable X defined by

$$X = F^{-1}(U)$$

Has distribution F. [$F^{-1}(u)$ is defined to be the value of x such that F(x)=u.]

➢ Proof

Let F_X denote the distribution function of $X=F^{-1}(U)$. Then

$$F_X(x) = P\{X \le x\} = P\{F^{-1}(U) \le x\}$$

Since F(x) is a monotone increasing function.

$$F_X(x) = P\{F(F^{-1}(U)) \le F(x)\} = P\{U \le F(x)\} = F(x)$$

Since $F(F^{-1}(U))=U$.

$$F_X(x) = P\{U \le F(x)\}$$

Since U is a uniform (0, 1).

$$F_X(x) = P\{U \le F(x)\} = F(x)$$

◆ 常態分配 CDF 的反函數，$N^{-1}()$，可以多項式來近似，前面講義有提 Glasserman 的近似多項式。

Input: u between 0 and 1
Output: x, approximation to $\Phi^{-1}(u)$.
$y \leftarrow u - 0.5$
if $|y| < 0.42$
  $r \leftarrow y * y$
  $x \leftarrow y * (((a_3 * r + a_2) * r + a_1) * r + a_0) /$
    $((((b_3 * r + b_2) * r + b_1) * r + b_0) * r + 1)$
else
  $r \leftarrow u;$
  if $(y > 0)$ $r \leftarrow 1 - u$
  $r \leftarrow \log(-\log(r))$
  $x \leftarrow c_0 + r * (c_1 + r * (c_2 + r * (c_3 + r * (c_4 +$
    $r * (c_5 + r * (c_6 + r * (c_7 + r * c_8)))))))$
  if $(y < 0)$ $x \leftarrow -x$
return x

$a_0 =$	2.50662823884	$b_0 =$	-8.47351093090
$a_1 =$	-18.61500062529	$b_1 =$	23.08336743743
$a_2 =$	41.39119773534	$b_2 =$	-21.06224101826
$a_3 =$	-25.44106049637	$b_3 =$	3.13082909833

$c_0 =$	0.3374754822726147	$c_5 =$	0.0003951896511919
$c_1 =$	0.9761690190917186	$c_6 =$	0.0000321767881768
$c_2 =$	0.1607979714918209	$c_7 =$	0.0000002888167364
$c_3 =$	0.0276438810333863	$c_8 =$	0.0000003960315187
$c_4 =$	0.0038405729373609		

◆ DFinMath 程式庫中有實作程式碼，請參考之。

```
public static double N_Inv(double x)
{
    //const double SQRT_TWO_PI = 2.506628274631;
    const double e_1 = -39.6968302866538;       const double e_2 = 220.946098424521;
    const double e_3 = -275.928510446969;       const double e_4 = 138.357751867269;
    const double e_5 = -30.6647980661472;       const double e_6 = 2.50662827745924;

    const double f_1 = -54.4760987982241;       const double f_2 = 161.585836858041;
    const double f_3 = -155.698979859887;       const double f_4 = 66.8013118877197;
    const double f_5 = -13.2806815528857;

    const double g_1 = -0.00778489400243029;    const double g_2 = -0.322396458041136;
    const double g_3 = -2.40075827716184;       const double g_4 = -2.54973253934373;
    const double g_5 = 4.37466414146497;        const double g_6 = 2.93816398269878;

    const double h_1 = 0.00778469570904146;     const double h_2 = 0.32246712907004;
    const double h_3 = 2.445134137143;          const double h_4 = 3.75440866190742;

    const double x_l = 0.02425;                 const double x_u = 0.97575;

    double z, r;

    // Lower region: 0 < x < x_l
    if (x < x_l)
    {
        z = Math.Sqrt(-2.0 * Math.Log(x));
        z = (((((g_1 * z + g_2) * z + g_3) * z + g_4) * z + g_5) * z + g_6) / ((((h_1 * z + h_2) * z + h_3) * z + h_4) * z + 1.0);
    }
    // Central region: x_l <= x <= x_u
    else if (x <= x_u)
    {
        z = x - 0.5;
        r = z * z;
        z = (((((e_1 * r + e_2) * r + e_3) * r + e_4) * r + e_5) * r + e_6) * z / (((((f_1 * r + f_2) * r + f_3) * r + f_4) * r + f_5) * r + 1.0);
    }
    // Upper region. ( x_u < x < 1 )
    else
    {
        z = Math.Sqrt(-2.0 * Math.Log(1.0 - x));
        z = -(((((g_1 * z + g_2) * z + g_3) * z + g_4) * z + g_5) * z + g_6) / ((((h_1 * z + h_2) * z + h_3) * z + h_4) * z + 1.0);
    }

    // Now |relative error| < 1.15e-9.  One iteration of Halley's third
    // order zero finder gives full machine precision:
    //r = (N(z) - x) * SQRT_TWO_PI * exp( 0.5 * z * z );  //  f(z)/df(z)
    //z -= r/(1+0.5*z*r);

    return z;
}
```

三、相關性的處理

- 模擬之標的變數可能不只一個，且變數間有相關性

$$\frac{dS_1}{S_1} = \mu_1 dt + \sigma_1 dZ_1$$

$$\frac{dS_2}{S_2} = \mu_2 dt + \sigma_2 dZ_2$$

- 兩變數報酬率之相關性為 ρ
  - Φ_1、Φ_2 為獨立之常態分配隨機變數 $\Phi_i \sim N(0, dt)$

$$dZ_1 = \Phi_1$$

$$dZ_2 = \sqrt{(1-\rho^2)}\Phi_2 + \rho\Phi_1$$

  - 矩陣表示為

$$\begin{bmatrix} dZ_1 \\ dZ_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix} \begin{bmatrix} \Phi_1 \\ \Phi_2 \end{bmatrix}$$

$$\Lambda = \begin{bmatrix} 1 & 0 \\ \rho & \sqrt{1-\rho^2} \end{bmatrix}$$

$$\Lambda \times \Lambda' = \begin{bmatrix} 1 & \rho \\ \rho & 1 \end{bmatrix}$$

◆ 考慮 n 種具相關性資產的情況

- i, j 資產間相關性為 ρ_{ij}
- Φ_i 為獨立之標準常態分配變數，$1 \le i \le n$，要求

$$dZ_i = \sum_{k=1}^{i} \alpha_{ik} \Phi_k$$

$$\sum_{k=1}^{i} \alpha_{ik}^2 = 1 \text{，} 1 \le j \le i$$

$$\sum_{k=1}^{j} \alpha_{ik} \alpha_{jk} = \rho_{ij} \text{，} j < i$$

◆ 上述步驟即為 Cholesky decomposition。

$$\Lambda = \lfloor \alpha_{ij} \rfloor$$

$$\Lambda \times \Lambda' = \begin{bmatrix} 1 & \rho_{12} & \dots & \rho_{1n} \\ \rho_{21} & 1 & \dots & \rho_{2n} \\ \dots & \dots & \dots & \dots \\ \rho_{n1} & \rho_{n2} & \dots & 1 \end{bmatrix}$$

四、Heston 模型的模擬

(一)資產價格行為

- Steven Heston(1993)提出下面模型，

$$dS_t = \mu S_t dt + \sqrt{V_t} S_t dW_t^1 \quad \text{.....(4.1)}$$

$$dV_t = \kappa(\theta - V_t)dt + \sigma\sqrt{V_t} dW_t^2 \quad \text{.....(4.2)}$$

$$dW_t^1 dW_t^2 = \rho \cdot dt \quad \text{.....(4.3)}$$

  - 其中 $\{S_t\}_{t\geq 0}$ 表價格過程，$\{V_t\}_{t\geq 0}$ 表波動性過程。
  - 以 P 測度表示此真實世界下的機率測量。
  - $\{W_t^1\}_{t\geq 0}$ 與 $\{W_t^2\}_{t\geq 0}$ 表真實世界中兩相關的布朗運動過程，相關係數為 ρ。
  - $\{V_t\}_{t\geq 0}$ 為一平方根均數回覆過程，長期平均為 θ，回覆速率為 κ，σ 稱之為波動性之波動性。
  - μ、ρ、θ、κ、σ 均為常數。

◆ 在 Q 測度下，(4.1)、(4.2)、(4.3)式成為，

$$dS_t = rS_t dt + \sqrt{V_t} S_t dZ_t^1 \quad \text{(4.4)}$$

$$dV_t = \kappa^*(\theta^* - V_t)dt + \sigma\sqrt{V_t} dZ_t^2 \quad \text{(4.5)}$$

$$dZ_t^1 dZ_t^2 = \rho \cdot dt \quad \text{(4.6)}$$

- 其中，$\kappa^* = \kappa + \lambda$，$\theta^* = \dfrac{\kappa\theta}{\kappa + \lambda}$。
- 由於我們所在意的為評價問題，因此所處理的測度為 Q 測度。
  - ✓ 後面的市場校準也是求得 Q 測度下的參數。
  - ✓ 參數 λ_t 的數值並不是重要的，因為已經吸收在 κ^*與 θ^*中，沒有明白的出現在(4.4)、(4.5)、(4.6)。
- 使用非線性最適化方法，校準出五個模型參數，V_0、κ^*、θ^*、ρ、σ。
  - ✓ QunatLib、Intel MKL、IMSL、Centerspace NMath 程式庫皆有內建最適化模組。
  - ✓ Nelder-Mead 與 Levenberg-Marquardt 演算法是較為被採用的方法。
  - ✓ 此部分因只要執行一次，CPU 端程式執行即可。

- 在 Heston 93 模型下，大部份的異種選擇權並沒有解析解。
  - 我們需要根據(4.4)~(4.6)的隨機過程，配合估計的參數，使用蒙地卡羅模擬法來計算權利金。
  - 此模擬乃在 Q 測度下進行的，下面重述這些隨機過程。

(二)Euler Scheme

- 令中間時點 $t_i = i*h$，$0 = t_0 < t_1 < \ldots < t_m = T$，(4.4)、(4.5)在 Euler 法下的近似式可表示為，
  - 以 S 為模擬對象，

$$S_{i+1} = S_i + rS_i[t_{i+1} - t_i] + \sqrt{V_i} S_i \sqrt{t_{i+1} - t_i} Z_{i+1}^1$$

$$V_{i+1} = V_i + \kappa^*(\theta^* - V_i)[t_{i+1} - t_i] + \sigma\sqrt{V_i}\sqrt{t_{i+1} - t_i} Z_{i+1}^2$$

  - 可以表示為，

$$S_{i+1} = S_i + rS_i h + \sqrt{V_i} S_i \sqrt{h} Z_{i+1}^1 \quad \text{.........(4.7)}$$

$$V_{i+1} = V_i + \kappa^*(\theta^* - V_i)h + \sigma\sqrt{V_i}\sqrt{h} Z_{i+1}^2 \quad \text{.........(4.8)}$$

◆ 以 ln(S)為模擬對象，

$$\ln S_{i+1} = \ln S_i + rh - \frac{1}{2} V_i h + \sqrt{V_i}\sqrt{h} Z_{i+1}^1 \quad \text{..........(4.9)}$$

$$V_{i+1} = V_i + \kappa^*(\theta^* - V_i)h + \sigma\sqrt{V_i}\sqrt{h} Z_{i+1}^2 \quad \text{..........(4.10)}$$

可以進一步表示為，

$$\ln S_{i+1} = \ln S_i + rh - \frac{1}{2} V_i h + \sqrt{V_i}\sqrt{h} Z_{i+1}^1 \quad \text{..........(4.11)}$$

$$V_{i+1} = V_i + \kappa^*(\theta^* - V_i)h + \sigma\sqrt{V_i}\sqrt{h} Z_{i+1}^2 \quad \text{..........(4.12)}$$

◆ 問題：(4.12)的變異數並不保證一定為正數。

- 即使 Feller Condition，$2\kappa\theta > \sigma^2$，滿足，也不一定確保變異數為正?
  - ✓ (4.5)為一 CIR Process，CIR 指出只要此條件滿足，變異數不會為負。
  - ✓ 實際模擬會產生負值，Why?
- 處理方法有二，
  - ✓ Full Truncation Scheme：若變異數為負，以零代之。
  - ✓ Reflection Scheme：若變異數為負，取絕對值代之。
- QuantLib 程式庫有詳細說明文件。
  - ✓ `\ql\processes\hestonprocess.cpp`
  - ✓ 提出四種方法，請回去自行研究。

(二)Milstein Scheme

◆ 針對(4.11)、(4.12)式隨機項進一步修正，可得如下式，

$$\ln S_{i+1} = \ln S_i + rh - \frac{1}{2}V_i h + \sqrt{V_i}\sqrt{h}Z_{i+1}^1 + \frac{1}{2}\sqrt{V_i}\cdot\sqrt{V_i}\cdot h\left(\left[Z_{i+1}^1\right]^2 - 1\right) \quad \text{...(4.13)}$$

$$V_{i+1} = V_i + \kappa^*(\theta^* - V_i)h + \sigma\sqrt{V_i}\sqrt{h}Z_{i+1}^2 + \frac{1}{2}\sigma\sqrt{V_i}\cdot\sigma\sqrt{V_i}\cdot h\left(\left[Z_{i+1}^2\right]^2 - 1\right) \quad \text{....................................(4.14)}$$

(三)Second-Order Method

- 將 Heston 模型改寫如下式，

$$dS_t = rS_t dt + \sqrt{V_t} S_t dZ_1$$

$$dV_t = \kappa(\theta - V_t)dt + \sqrt{V_t}(\sigma_1 dZ_1 + \sigma_2 dZ_2)$$

$$dZ_1 dZ_2 = 0$$

- 針對(4.7)、(4.8)的二階近似方法，可以下式估計，

$$S_{i+1} = S_i(1 + rh + \sqrt{V_i}\Delta Z_1) + \frac{1}{2}r^2 S_i h^2 + \left(\left[r + \frac{\sigma_1 - \kappa}{4}\right]S_i\sqrt{V_i} + \left[\frac{\kappa\theta}{4} - \frac{\sigma^2}{16}\right]\frac{S_i}{\sqrt{V_i}}\right)\Delta Z_1 h$$

$$+\frac{1}{2}S_i\left(V_i + \frac{\sigma_1}{2}\right)(\Delta Z_1^2 - h) + \frac{1}{4}\sigma_2 S_i(\Delta Z_2 \Delta Z_1 + \xi)$$

$$V_{i+1} = \kappa\theta h + (1-\kappa h)V_i + \sqrt{V_i}(\sigma_1 \Delta Z_1 + \sigma_2 \Delta Z_2) - \frac{1}{2}\kappa^2(\theta - V_i)h^2$$

$$+\left(\left[\frac{\kappa\theta}{4} - \frac{\sigma^2}{16}\right]\frac{1}{\sqrt{V_i}} - \frac{3\kappa}{2}\sqrt{V_i}\right)(\sigma_1 \Delta Z_1 + \sigma_2 \Delta Z_2)h$$

$$+\frac{1}{4}\sigma_1^2(\Delta Z_1^2 - h) + \frac{1}{4}\sigma_2^2(\Delta Z_2^2 - h) + \frac{1}{2}\sigma_1\sigma_2\Delta Z_2\Delta Z_1$$

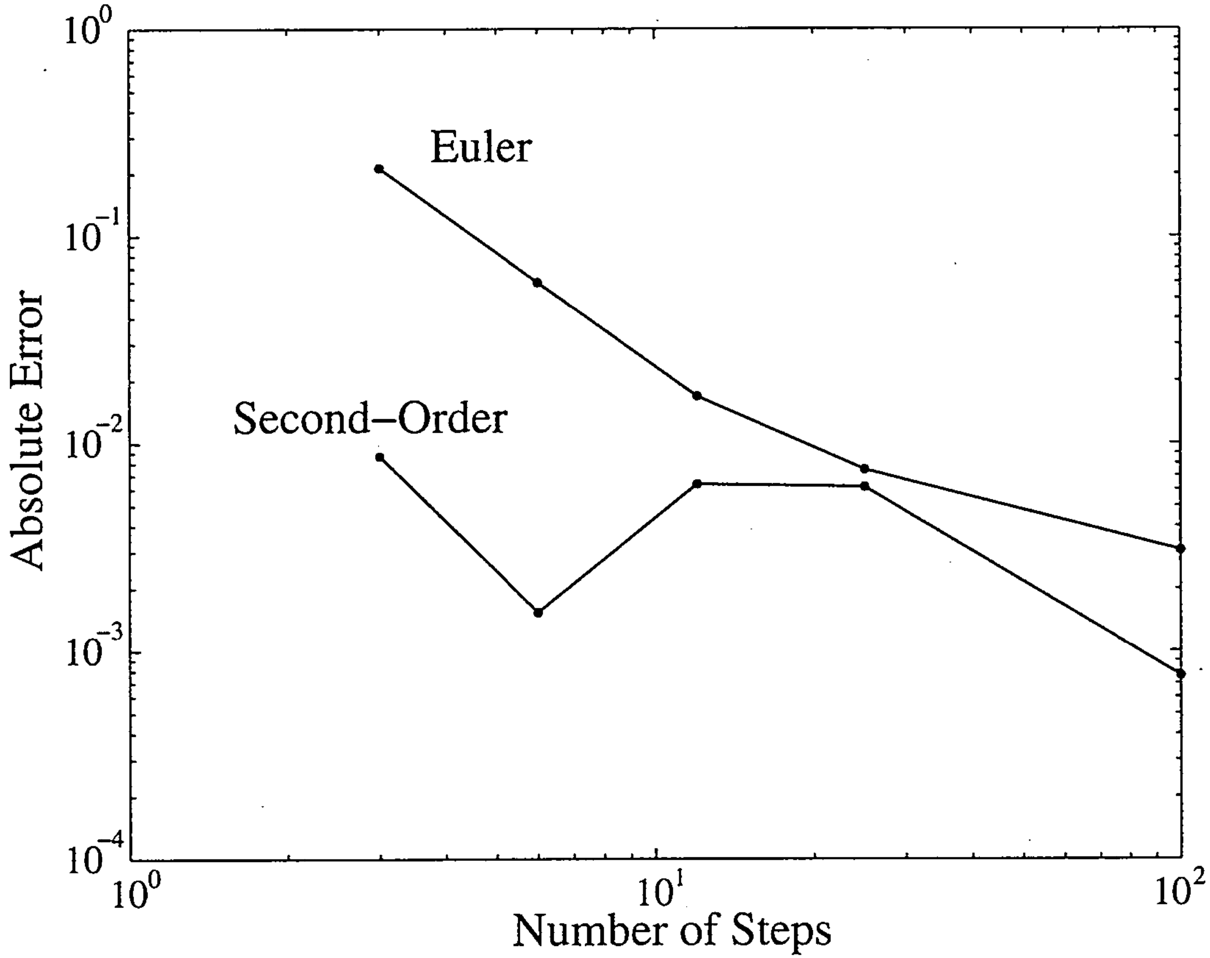
- 其中，ξ為獨立於布朗增量之隨機變數，

$$\xi = \begin{cases} h, & p = 0.5 \\ -h, & 1-p = 0.5 \end{cases}$$

$$\sigma^2 = \sigma_1^2 + \sigma_2^2$$

◆ 下圖為 Euler 法、二階近似法與 Closed-form 的誤差比較圖。

S=100, V=0.04, r=5%, κ=1.2, θ=0.04, σ=0.30, ρ=-0.5, T=1.0, K=100，C=10.2300。



五、實作案例二

以一個一年期的結構商品契約為例，每個月比價一次，償付與路徑價格過程有關。

- 令今日為 2014/6/1，到期日為 2015/6/1。
  - ✓ 百慕達式選擇權契約可以在多個日期執行。
  - ✓ 需要每個月的觀察值。

- 參見實作手冊。