Financial Engineering Mathematics

財務工程數學

NTUST/First Semester, 2019

昀騰金融科技

Wintom Financial Technology

技術長 CTO

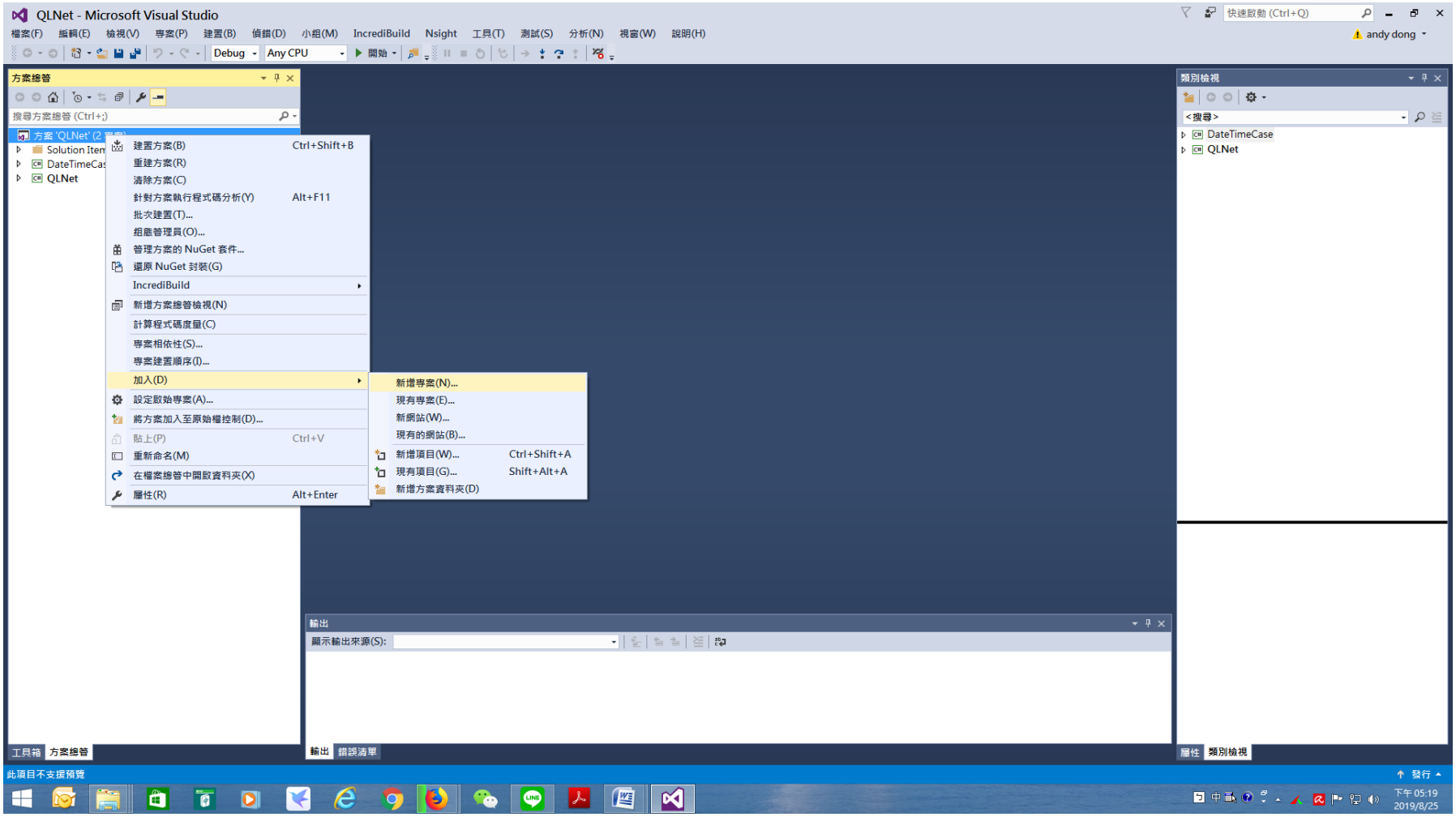
董夢雲 博士 Dr. Andy Dong

dongmy@ms5.hinet.net

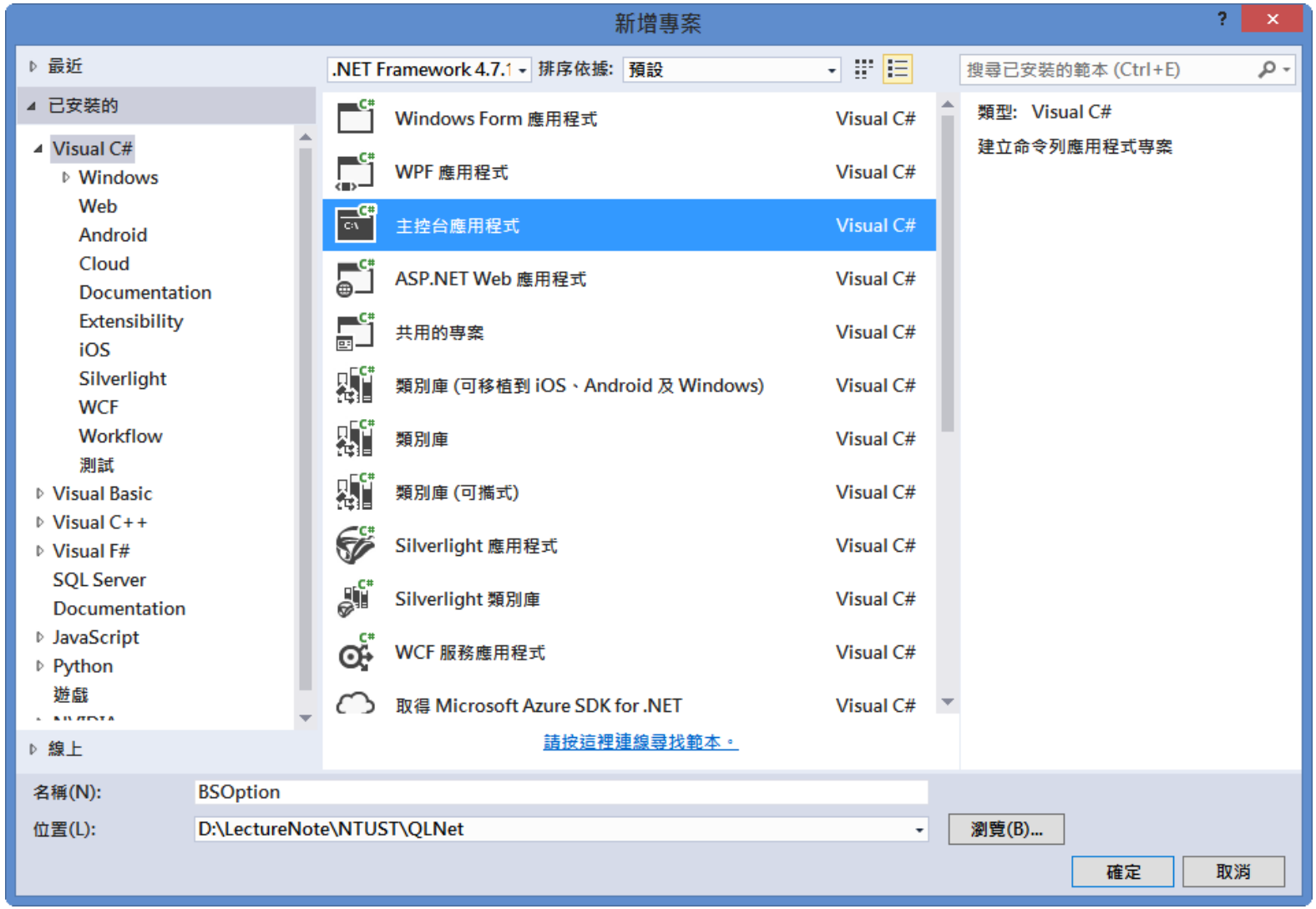
Contents

2.1 Black-Scholes Option Model and Greeks Calculation

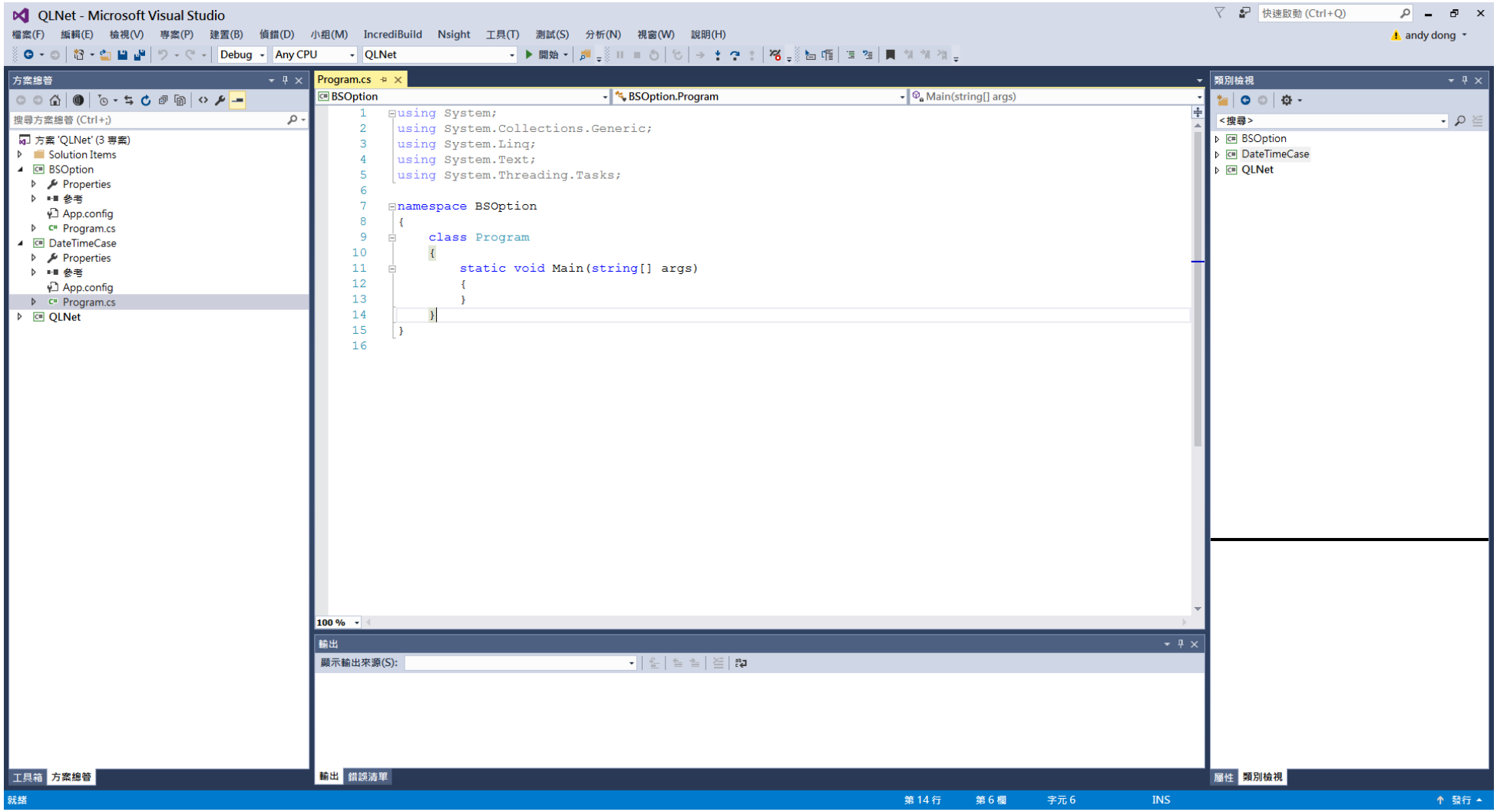
◆ Add New Project



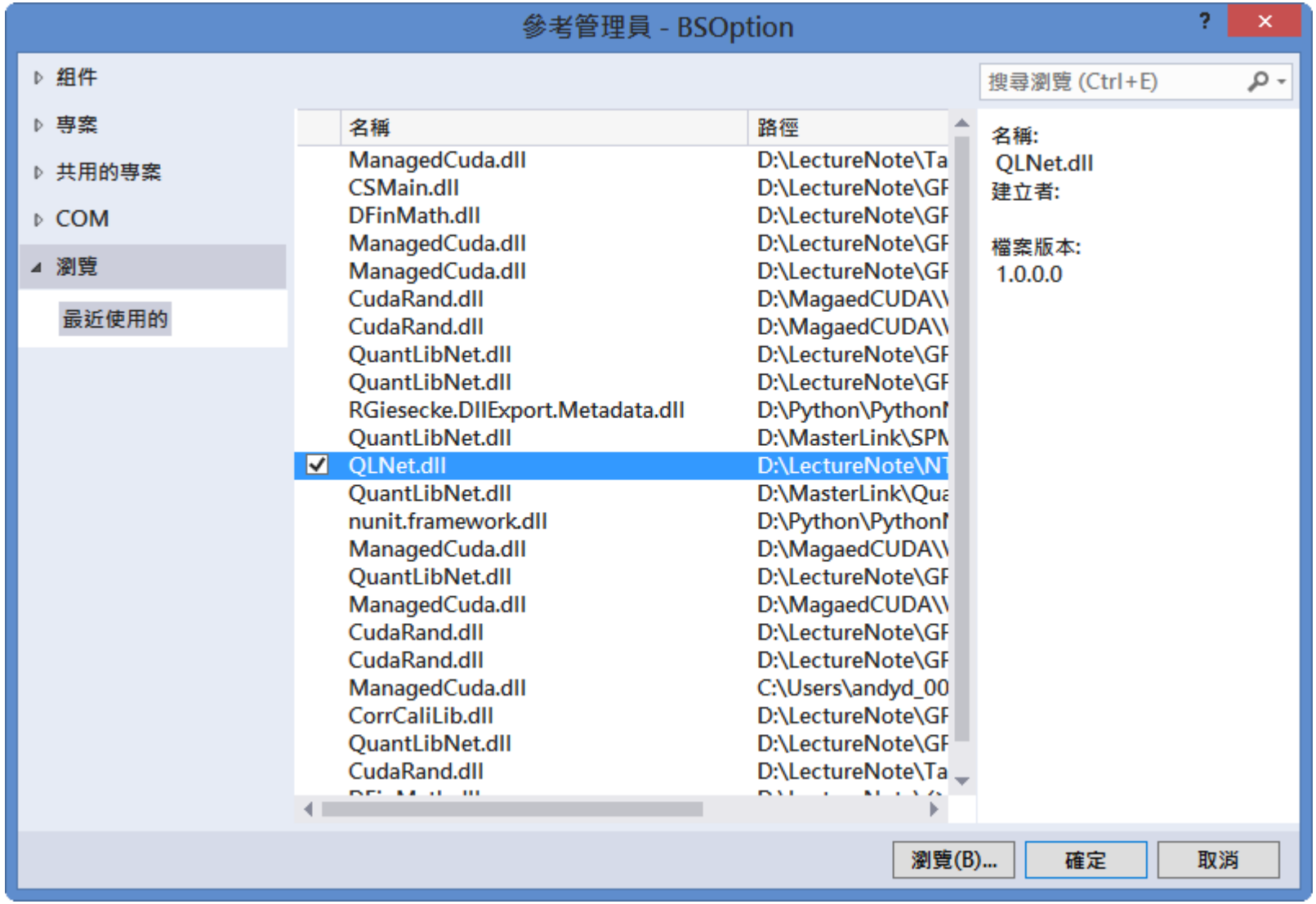
➢ Name：BSOption, Console Application。

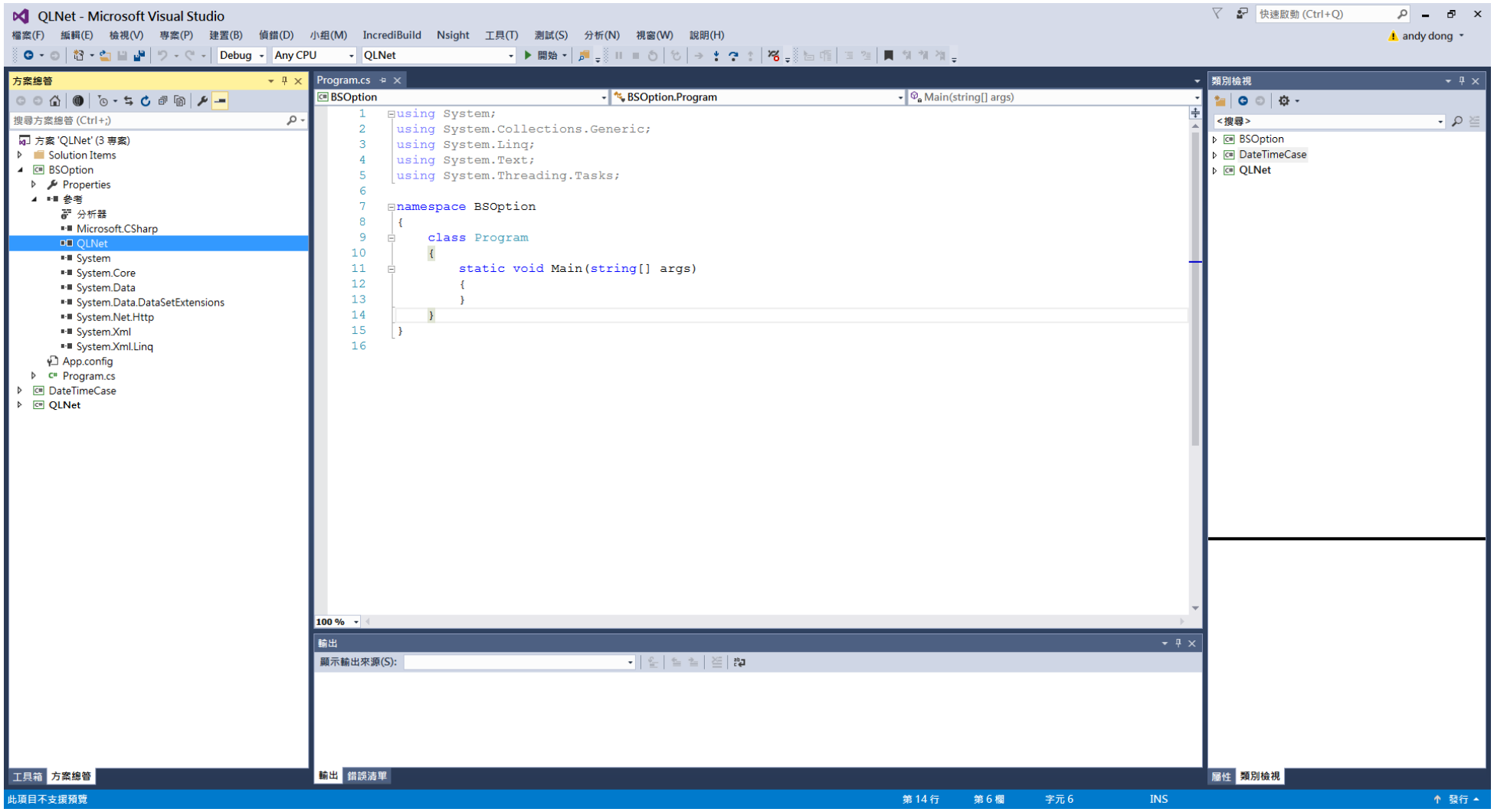


◆ Default Function

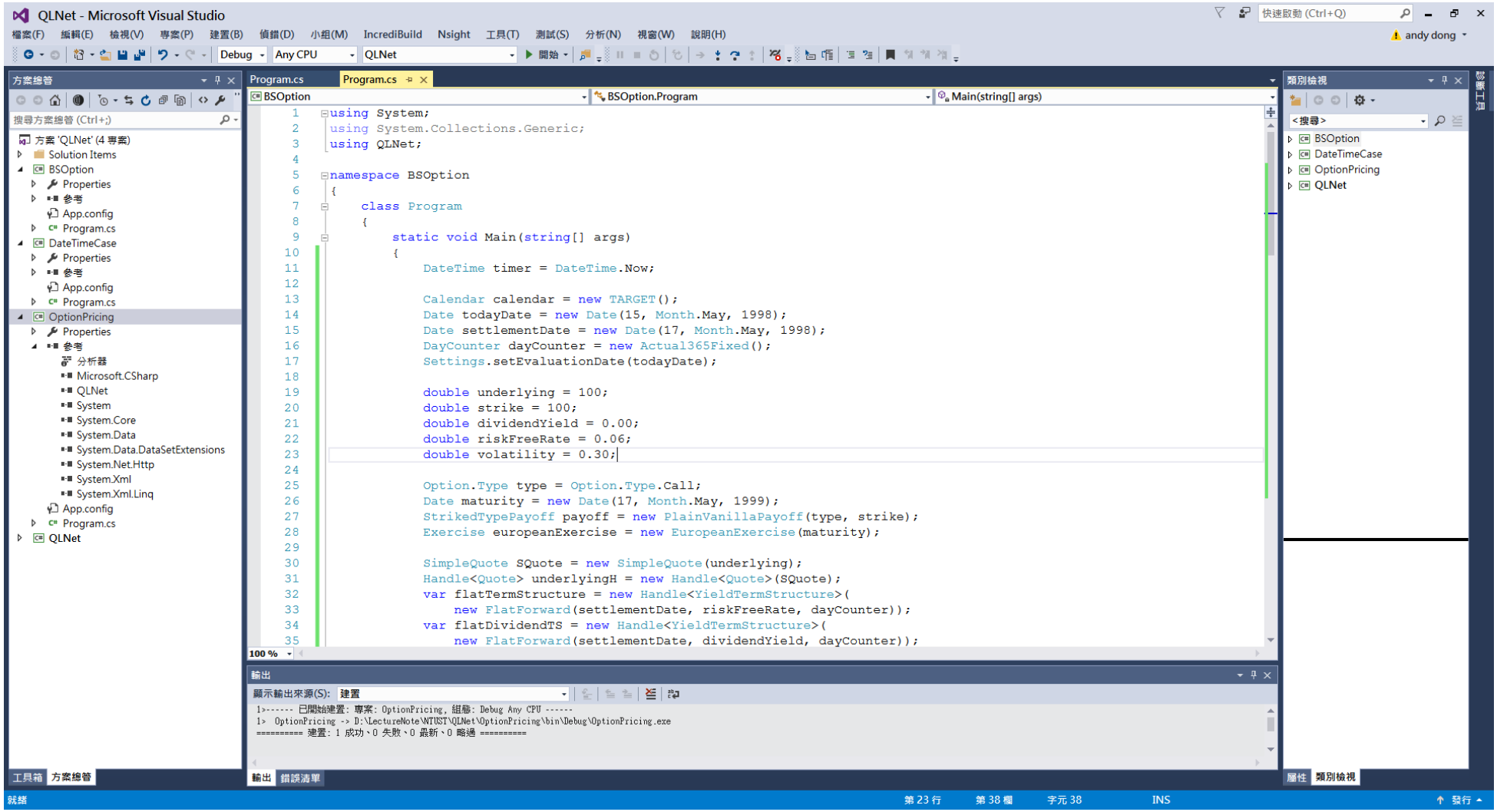


◆ Add Reference




QLNet - Microsoft Visual Studio
檔案(F) 編輯(E) 檢視(V) 專案(P) 建置(B) 偵錯(D) 小組(M) IncrediBuild Nsight 工具(T) 測試(S) 分析(N) 視窗(W) 說明(H)
Debug
Any CPU
QLNet
開始
快速啟動 (Ctrl+Q)
andy dong
方案總管
搜尋方案總管 (Ctrl+;)
方案 'QLNet' (3 專案)
Solution Items
BSOption
Properties
參考
分析器
Microsoft.CSharp
QLNet
System
System.Core
System.Data
System.Data.DataSetExtensions
System.Net.Http
System.Xml
System.Xml.Linq
App.config
Program.cs
DateTimeCase
QLNet
Program.cs
BSOption
BSOption.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace BSOption
{
class Program
{
static void Main(string[] args)
{
}
}
}
100 %
類別檢視
<搜尋>
BSOption
DateTimeCase
QLNet
輸出
顯示輸出來源(S):
工具箱 方案總管
輸出 錯誤清單
屬性 類別檢視
此項目不支援預覽
第 14 行
第 6 欄
字元 6
INS
發行

◆ Add Code


QLNet - Microsoft Visual Studio
Program.cs
BSOption
BSOption.Program
Main(string[] args)
using System;
using System.Collections.Generic;
using QLNet;
namespace BSOption
{
class Program
{
static void Main(string[] args)
{
DateTime timer = DateTime.Now;
Calendar calendar = new TARGET();
Date todayDate = new Date(15, Month.May, 1998);
Date settlementDate = new Date(17, Month.May, 1998);
DayCounter dayCounter = new Actual365Fixed();
Settings.setEvaluationDate(todayDate);
double underlying = 100;
double strike = 100;
double dividendYield = 0.00;
double riskFreeRate = 0.06;
double volatility = 0.30;
Option.Type type = Option.Type.Call;
Date maturity = new Date(17, Month.May, 1999);
StrikedTypePayoff payoff = new PlainVanillaPayoff(type, strike);
Exercise europeanExercise = new EuropeanExercise(maturity);
SimpleQuote SQuote = new SimpleQuote(underlying);
Handle<Quote> underlyingH = new Handle<Quote>(SQuote);
var flatTermStructure = new Handle<YieldTermStructure>(
new FlatForward(settlementDate, riskFreeRate, dayCounter));
var flatDividendTS = new Handle<YieldTermStructure>(
new FlatForward(settlementDate, dividendYield, dayCounter));

◆ Code

```
class Program
{
    static void Main(string[] args)
    {
        DateTime timer = DateTime.Now;

        Calendar calendar = new TARGET();
        Date todayDate = new Date(15, Month.May, 1998);
        Date settlementDate = new Date(17, Month.May, 1998);

        DayCounter dayCounter = new Actual365Fixed();
        Settings.setEvaluationDate(todayDate);

        double underlying = 100;
        double strike = 100;
        double dividendYield = 0.00;
        double riskFreeRate = 0.06;
        double volatility = 0.30;

        Option.Type type = Option.Type.Call;
        Date maturity = new Date(17, Month.May, 1999);

        StrikedTypePayoff payoff = new PlainVanillaPayoff(type, strike);
```

```
Exercise europeanExercise = new EuropeanExercise(maturity);

SimpleQuote SQuote = new SimpleQuote(underlying);
Handle<Quote> underlyingH = new Handle<Quote>(SQuote);

var flatTermStructure = new Handle<YieldTermStructure>(
    new FlatForward(settlementDate, riskFreeRate, dayCounter));

var flatDividendTS = new Handle<YieldTermStructure>(
    new FlatForward(settlementDate, dividendYield, dayCounter));

var flatVolTS = new Handle<BlackVolTermStructure>(
    new BlackConstantVol(settlementDate, calendar, volatility, dayCounter));

var bsmProcess = new BlackScholesMertonProcess(
    underlyingH, flatDividendTS, flatTermStructure, flatVolTS);

VanillaOption europeanOption = new VanillaOption(payoff, europeanExercise);
europeanOption.setPricingEngine(new AnalyticEuropeanEngine(bsmProcess));
```

```
string method = "Black-Scholes";
double premium = europeanOption.NPV();
double delta = europeanOption.delta();
double gamma = europeanOption.gamma();
double vega = europeanOption.vega();
double theta = europeanOption.theta();
double rho = europeanOption.rho();

int[] widths = new int[] { 15, 15, 15, 15, 15 };
Console.Write("{0, -" + widths[0] + "}", "Method");
Console.Write("{0, -" + widths[1] + "}", "European");
Console.WriteLine();
Console.Write("{0, -" + widths[0] + "}", method);
Console.Write("{0, -" + widths[1] + ":0.000000}", premium);
Console.WriteLine();
Console.Write("{0, -" + widths[0] + "}", "Delta");
Console.Write("{0, -" + widths[1] + "}", "Gamma");
Console.Write("{0, -" + widths[2] + "}", "Vega");
Console.Write("{0, -" + widths[3] + "}", "Theta");
Console.WriteLine("{0, -" + widths[4] + "}", "Rho");
Console.Write("{0, -" + widths[0] + ":0.000000}", delta);
Console.Write("{0, -" + widths[1] + ":0.000000}", gamma);
Console.Write("{0, -" + widths[2] + ":0.000000}", vega);
Console.Write("{0, -" + widths[3] + ":0.000000}", theta);
Console.WriteLine("{0, -" + widths[4] + ":0.000000}\n\n", rho);
```

```
SQuote.setValue(101);

premium = europeanOption.NPV();

delta = europeanOption.delta();
gamma = europeanOption.gamma();
vega = europeanOption.vega();
theta = europeanOption.theta();
rho = europeanOption.rho();

Console.Write("{0, -" + widths[0] + "}", "Method");
Console.Write("{0, -" + widths[1] + "}", "European");
Console.WriteLine();

Console.Write("{0, -" + widths[0] + "}", method);
Console.Write("{0, -" + widths[1] + ":0.000000}", premium);
Console.WriteLine();

Console.Write("{0, -" + widths[0] + "}", "Delta");
Console.Write("{0, -" + widths[1] + "}", "Gamma");
Console.Write("{0, -" + widths[2] + "}", "Vega");
Console.Write("{0, -" + widths[3] + "}", "Theta");
Console.WriteLine("{0, -" + widths[4] + "}", "Rho");
```

```
        Console.Write("{0, -" + widths[0] + ":0.000000}", delta);
        Console.Write("{0, -" + widths[1] + ":0.000000}", gamma);
        Console.Write("{0, -" + widths[2] + ":0.000000}", vega);
        Console.Write("{0, -" + widths[3] + ":0.000000}", theta);
        Console.WriteLine("{0, -" + widths[4] + ":0.000000}", rho);

        Console.ReadKey();
    }
}
```

◆ Output

VS2015 x64

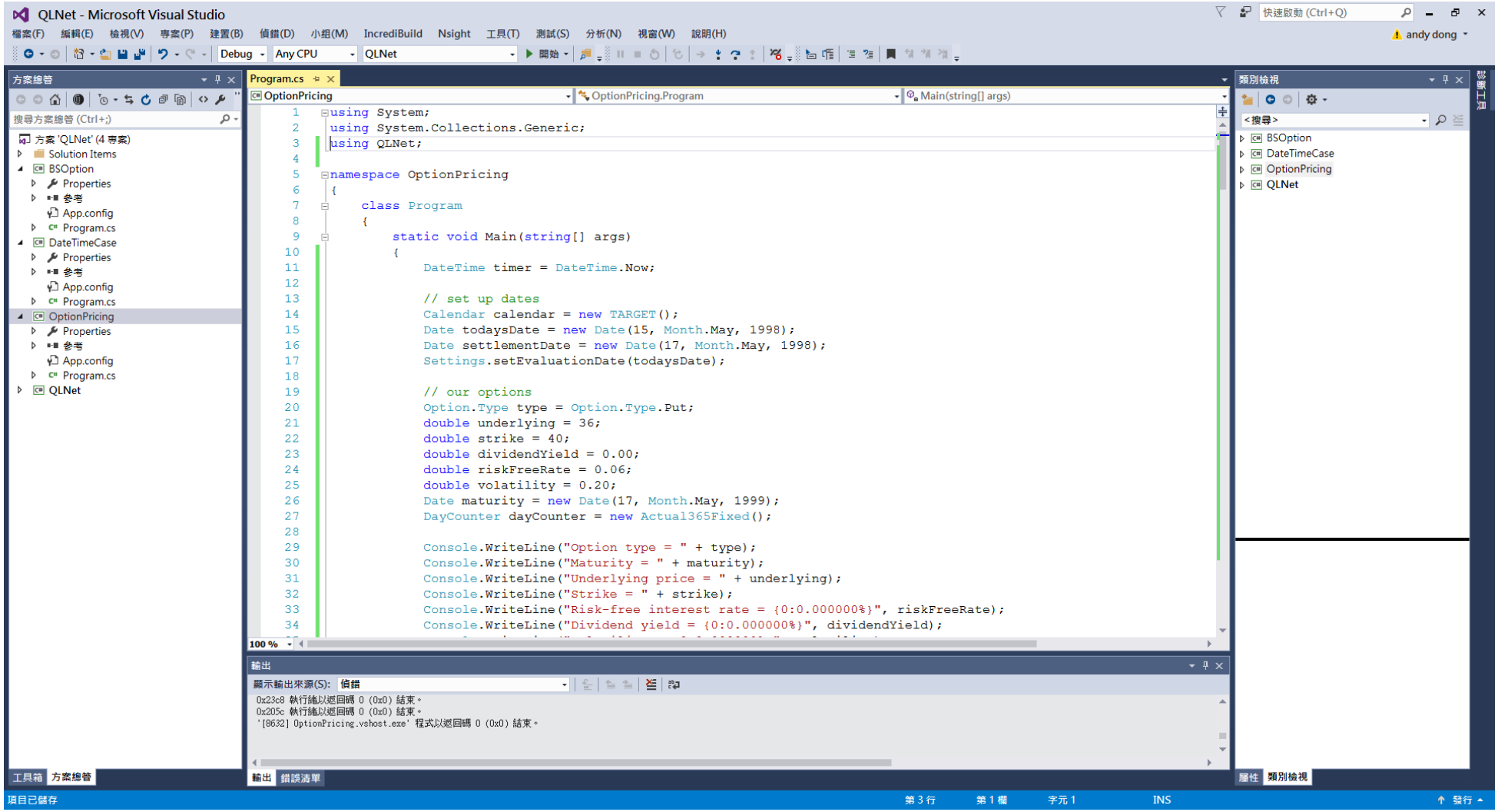
```
D:\LectureNote\NTUST\QLNet\BSOption\bin\Debug>BSOption
Method          European
Black-Scholes   14.717072
Delta           Gamma           Vega            Theta           Rho
0.636831        0.012508        37.524035       -8.566565       48.965993


Method          European
Black-Scholes   15.360112
Delta           Gamma           Vega            Theta           Rho
0.649202        0.012235        37.441257       -8.628748       50.209322

D:\LectureNote\NTUST\QLNet\BSOption\bin\Debug>
```

2.2 Option Pricing Models

- Create Project by the same way



◆ Add Code ◦

```
static void Main(string[] args)
{
    DateTime timer = DateTime.Now;
    // set up dates
    Calendar calendar = new TARGET();
    Date todaysDate = new Date(15, Month.May, 1998);
    Date settlementDate = new Date(17, Month.May, 1998);
    Settings.setEvaluationDate(todaysDate);

    // our options input data
    Option.Type type = Option.Type.Put;
    double underlying = 36;
    double strike = 40;
    double dividendYield = 0.00;
    double riskFreeRate = 0.06;
    double volatility = 0.20;
    Date maturity = new Date(17, Month.May, 1999);
    DayCounter dayCounter = new Actual365Fixed();
```

```
Console.WriteLine("Option type = " + type);
Console.WriteLine("Maturity = " + maturity);
Console.WriteLine("Underlying price = " + underlying);
Console.WriteLine("Strike = " + strike);
Console.WriteLine("Risk-free interest rate = {0:0.000000%}", riskFreeRate);
Console.WriteLine("Dividend yield = {0:0.000000%}", dividendYield);
Console.WriteLine("Volatility = {0:0.000000%}", volatility);
Console.Write("\n");

string method;
Console.Write("\n");

// write column headings
int[] widths = new int[] { 35, 14, 14, 14 };
Console.Write("{0,-" + widths[0] + "}", "Method");
Console.Write("{0,-" + widths[1] + "}", "European");
Console.Write("{0,-" + widths[2] + "}", "Bermudan");
Console.WriteLine("{0,-" + widths[3] + "}", "American");
```

```
// ---------------------------------------------------------------------------
// maturity dates
List<Date> exerciseDates = new List<Date>(); ;

for (int i = 1; i <= 4; i++)
{
    exerciseDates.Add(settlementDate + new Period(3 * i, TimeUnit.Months));
}

Exercise europeanExercise = new EuropeanExercise(maturity);
Exercise bermudanExercise = new BermudanExercise(exerciseDates);
Exercise americanExercise = new AmericanExercise(settlementDate, maturity);

// underlying asset
Handle<Quote> underlyingH = new Handle<Quote>(new SimpleQuote(underlying));
```

```
// bootstrap the yield/dividend/vol curves
var flatTermStructure = new Handle<YieldTermStructure>(new FlatForward(settlementDate,
    riskFreeRate, dayCounter));

var flatDividendTS = new Handle<YieldTermStructure>(new FlatForward(settlementDate,
    dividendYield, dayCounter));

var flatVolTS = new Handle<BlackVolTermStructure>(new BlackConstantVol(settlementDate,
    calendar, volatility, dayCounter));

StrikedTypePayoff payoff = new PlainVanillaPayoff(type, strike);

var bsmProcess = new BlackScholesMertonProcess(underlyingH, flatDividendTS,
    flatTermStructure, flatVolTS);

// options
VanillaOption europeanOption = new VanillaOption(payoff, europeanExercise);
VanillaOption bermudanOption = new VanillaOption(payoff, bermudanExercise);
VanillaOption americanOption = new VanillaOption(payoff, americanExercise);
```

```
// ******************************************************************************
// Analytic formulas:
// Black-Scholes for European
method = "Black-Scholes";
europeanOption.setPricingEngine(new AnalyticEuropeanEngine(bsmProcess));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + "}", "N/A");
Console.WriteLine("{0,-" + widths[3] + "}", "N/A");

// Barone-Adesi and Whaley approximation for American
method = "Barone-Adesi/Whaley";
americanOption.setPricingEngine(new BaroneAdesiWhaleyApproximationEngine(bsmProcess));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + "}", "N/A");
Console.Write("{0,-" + widths[2] + "}", "N/A");
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());
```

```
// Bjerksund and Stensland approximation for American
method = "Bjerksund/Stensland";
americanOption.setPricingEngine(new BjerksundStenslandApproximationEngine(bsmProcess));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + "}", "N/A");
Console.Write("{0,-" + widths[2] + "}", "N/A");
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());
```

```
// Integral
method = "Integral";
europeanOption.setPricingEngine(new IntegralEngine(bsmProcess));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + "}", "N/A");
Console.WriteLine("{0,-" + widths[3] + "}", "N/A");

// Finite differences
int timeSteps = 801;
method = "Finite differences";
europeanOption.setPricingEngine(new FDEuropeanEngine(bsmProcess, timeSteps, timeSteps - 1));
bermudanOption.setPricingEngine(new FDBermudanEngine(bsmProcess, timeSteps, timeSteps - 1));
americanOption.setPricingEngine(new FDAmericanEngine(bsmProcess, timeSteps, timeSteps - 1));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());
```

```
// Binomial method
// Jarrow-Rudd
method = "Binomial Jarrow-Rudd";
europeanOption.setPricingEngine(new BinomialVanillaEngine<JarrowRudd>(bsmProcess, timeSteps));
bermudanOption.setPricingEngine(new BinomialVanillaEngine<JarrowRudd>(bsmProcess, timeSteps));
americanOption.setPricingEngine(new BinomialVanillaEngine<JarrowRudd>(bsmProcess, timeSteps));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());

// Cox-Ross-Rubinstein
method = "Binomial Cox-Ross-Rubinstein";
europeanOption.setPricingEngine(new BinomialVanillaEngine<CoxRossRubinstein>(bsmProcess, timeSteps));
bermudanOption.setPricingEngine(new BinomialVanillaEngine<CoxRossRubinstein>(bsmProcess, timeSteps));
americanOption.setPricingEngine(new BinomialVanillaEngine<CoxRossRubinstein>(bsmProcess, timeSteps));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());
```

```
// Additive equiprobabilities
method = "Additive equiprobabilities";
europeanOption.setPricingEngine(new BinomialVanillaEngine<AdditiveEQPBinomialTree>(bsmProcess, timeSteps));
bermudanOption.setPricingEngine(new BinomialVanillaEngine<AdditiveEQPBinomialTree>(bsmProcess, timeSteps));
americanOption.setPricingEngine(new BinomialVanillaEngine<AdditiveEQPBinomialTree>(bsmProcess, timeSteps));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());

// Binomial Trigeorgis
method = "Binomial Trigeorgis";
europeanOption.setPricingEngine(new BinomialVanillaEngine<Trigeorgis>(bsmProcess, timeSteps));
bermudanOption.setPricingEngine(new BinomialVanillaEngine<Trigeorgis>(bsmProcess, timeSteps));
americanOption.setPricingEngine(new BinomialVanillaEngine<Trigeorgis>(bsmProcess, timeSteps));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());
```

```
// Binomial Tian
method = "Binomial Tian";
europeanOption.setPricingEngine(new BinomialVanillaEngine<Tian>(bsmProcess, timeSteps));
bermudanOption.setPricingEngine(new BinomialVanillaEngine<Tian>(bsmProcess, timeSteps));
americanOption.setPricingEngine(new BinomialVanillaEngine<Tian>(bsmProcess, timeSteps));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());

// Binomial Leisen-Reimer
method = "Binomial Leisen-Reimer";
europeanOption.setPricingEngine(new BinomialVanillaEngine<LeisenReimer>(bsmProcess, timeSteps));
bermudanOption.setPricingEngine(new BinomialVanillaEngine<LeisenReimer>(bsmProcess, timeSteps));
americanOption.setPricingEngine(new BinomialVanillaEngine<LeisenReimer>(bsmProcess, timeSteps));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());
```

```
// Binomial Joshi
method = "Binomial Joshi";
europeanOption.setPricingEngine(new BinomialVanillaEngine<Joshi4>(bsmProcess, timeSteps));
bermudanOption.setPricingEngine(new BinomialVanillaEngine<Joshi4>(bsmProcess, timeSteps));
americanOption.setPricingEngine(new BinomialVanillaEngine<Joshi4>(bsmProcess, timeSteps));
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", bermudanOption.NPV());
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());
```

```
// ********************************************************************************
// Monte Carlo Method
// MC (crude)
timeSteps = 1;
method = "MC (crude)";
ulong mcSeed = 42;
IPricingEngine mcengine1 = new MakeMCEuropeanEngine<PseudoRandom>(bsmProcess)
                                    .withSteps(timeSteps)
                                    .withAbsoluteTolerance(0.02)
                                    .withSeed(mcSeed)
                                    .value();
europeanOption.setPricingEngine(mcengine1);
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", "N/A");
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", "N/A");
```

```
// QMC (Sobol)
method = "QMC (Sobol)";
int nSamples = 32768;  // 2^15
IPricingEngine mcengine2 = new MakeMCEuropeanEngine<LowDiscrepancy>(bsmProcess)
                                    .withSteps(timeSteps)
                                    .withSamples(nSamples)
                                    .value();
europeanOption.setPricingEngine(mcengine2);
Console.Write("{0,-" + widths[0] + "}", method);
Console.Write("{0,-" + widths[1] + ":0.000000}", europeanOption.NPV());
Console.Write("{0,-" + widths[2] + ":0.000000}", "N/A");
Console.WriteLine("{0,-" + widths[3] + ":0.000000}", "N/A");
```

```
    // MC (Longstaff Schwartz)
    method = "MC (Longstaff Schwartz)";
    IPricingEngine mcengine3 = new MakeMCAmericanEngine<PseudoRandom>(bsmProcess)
                                    .withSteps(100)
                                    .withAntitheticVariate()
                                    .withCalibrationSamples(4096)
                                    .withAbsoluteTolerance(0.02)
                                    .withSeed(mcSeed)
                                    .value();
    americanOption.setPricingEngine(mcengine3);
    Console.Write("{0,-" + widths[0] + "}", method);
    Console.Write("{0,-" + widths[1] + ":0.000000}", "N/A");
    Console.Write("{0,-" + widths[2] + ":0.000000}", "N/A");
    Console.WriteLine("{0,-" + widths[3] + ":0.000000}", americanOption.NPV());

    // End test
    Console.WriteLine(" \nRun completed in {0}", DateTime.Now - timer);
    Console.WriteLine();
    Console.ReadKey();
}
```

◆ Output -- 5.88 Secs, Very fast.

```
VS2015 x64

D:\LectureNote\NTUST\QLNet\OptionPricing\bin\Debug>OptionPricing
Option type = Put
Maturity = 1999/5/17
Underlying price = 36
Strike = 40
Risk-free interest rate = 6.000000%
Dividend yield = 0.000000%
Volatility = 20.000000%


Method                              European      Bermudan      American
Black-Scholes                       3.844308      N/A           N/A
Barone-Adesi/Whaley                 N/A           N/A           4.459628
Bjerksund/Stensland                 N/A           N/A           4.453064
Integral                            3.844309      N/A           N/A
Finite differences                  3.844342      4.360807      4.486118
Binomial Jarrow-Rudd                3.844132      4.361174      4.486552
Binomial Cox-Ross-Rubinstein        3.843504      4.360861      4.486415
Additive equiprobabilities          3.836911      4.354455      4.480097
Binomial Trigeorgis                 3.843557      4.360909      4.486461
Binomial Tian                       3.844171      4.361176      4.486413
Binomial Leisen-Reimer              3.844308      4.360713      4.486076
Binomial Joshi                      3.844308      4.360713      4.486076
MC (crude)                          3.834522      N/A           N/A
QMC (Sobol)                         3.844613      N/A           N/A
MC (Longstaff Schwartz)             N/A           N/A           4.481675

Run completed in 00:00:05.8832057


D:\LectureNote\NTUST\QLNet\OptionPricing\bin\Debug>
```