**UNIVERSITY OF PORTSMOUTH**

DEPARTMENT OF MATHEMATICS

COURSEWORK — 2013 / 2014

SECOND YEAR (FHEQ LEVEL 5)

UNIT M261 / U20740

# MATHEMATICS FOR FINANCE

Instructions: HAND-IN DATE: FRIDAY, 28 FEB, 2014 [VIA MOODLE]

Answer ALL FOUR questions.
Each question is worth 25% of this coursework.
Write your code as clearly as possible.
Include comments in the code to explain what you are doing.
This is an INDIVIDUAL coursework.
This must be all your own work.
All files should be uploaded to Moodle (details to follow).

Student ID
625015

## Read this first!

1. Ensure that you have the correct questions — check that the question file matches your Student Reference Number. **Answering the wrong set of questions will gain you zero marks.**
2. Each question requires you to write one or more Python functions to perform certain calculations. **The names and inputs for these functions, and the names of the Python files in which they are written must be *exactly as specified in the question***, but the instructions you put inside them are up to you.
3. Each function should **return the results of the calculations** required by the question.
4. The majority of the marks available for this part of the course work will be given for writing functions which **correctly perform the required calculations.**
5. The remaining marks available for the programs is for the **clarity of the code, and the comments that you write inside it.** To help explain what you are doing inside the program to someone who is reading the code, you should include comment lines which start with the # symbol. You do *not* need copious numbers of these lines; aim to give the reader a clear idea about what your code is doing in as concise a way as you can.
6. **Aim to make your python code easily readable** - for example try to give variables meaningful or recognisable names.
7. Think about how tasks can best be divided up into separate functions - if there is some part of a calculation that may be needed multiple times in different places then perhaps it is worth defining a function which does just this one calculation. **You are encouraged to write your own additional small functions in order to split a large calculation into smaller pieces.**
8. **Each file that you create should, as usual, have a single `main()` function that demonstrates the code. Don't forget to run the main function at the end of each file.**
9. Your answers will be compared, so ensure that you do the final work yourself and write meaningful comments. **Plagiarism is an assessment offence and disciplinary action will be taken against students who have cheated.**
10. **Coursework submission will be electronic via Moodle.**

# Useful formulae

## Compound interest

The future value, $F$, of a present amount, $P$, invested for $n$ time-periods with compound interest at rate $r$ per time-period, is given by

$$F = M^n P, \tag{1}$$

where $M$ is the multiplier for one time-period, given by

$$M = \begin{cases} (1+r) & \text{(single compounding)}, \\ (1+r/m)^m & (m\text{-times multiple compounding per time-period}), \\ e^r & \text{(continuous compounding)}. \end{cases} \tag{2}$$

## The binomial model

Suppose we take the limit $\delta t \to 0$ in the Binomial model, with:

$$u = e^{r\delta t + \sigma\sqrt{\delta t}} \tag{3}$$

$$d = e^{r\delta t - \sigma\sqrt{\delta t}} \tag{4}$$

where $r$ is the continuous compounding interest rate and the parameter $\sigma$ is called the volatility of the stock. We find that the value of a path independent European option with payoff function $f(S_T)$ where $S_T$ is the stock price at expiry is given by:

$$V_0 = \frac{e^{-rT}}{\sqrt{2\pi\sigma^2 T}} \int_{-\infty}^{\infty} \exp\left(-\frac{(z+\frac{1}{2}\sigma^2 T)^2}{2\sigma^2 T}\right) f(S_0 e^{rT+z}) dz \tag{5}$$

## Normal probability

The normal probability density function (pdf) is given by

$$n(x) = \frac{e^{-(x^2)/2}}{\sqrt{2\pi}}. \tag{6}$$

Its integral is the normal cumulative density function (cdf) given by

$$\Phi(d) \equiv N(d) = \int_{x=-\infty}^{d} n(x)\, dx = \frac{1}{2}\left(1 + \operatorname{erf}\left(\frac{d}{\sqrt{2}}\right)\right). \tag{7}$$

## The bond price formula

A bond with face value $F$ that makes $m$ coupon payments of $C/m$ each per year, and has $n$ coupon payments remaining has value (where $r$ is the annual interest rate with compounding $m$ times per year).

$$P = \left(\sum_{k=1}^{n} \frac{C/m}{(1+r/m)^k}\right) + \frac{F}{(1+r/m)^n}. \tag{8}$$

**Q1** Put your answers to this question in a file named `question1.py`

This question concerns the basic compound interest schemes.

(a) Write a Python function, of the form

```
def single_compound(P, r, n):
    #Function body here.
    return #Don't forget to return the answer.
```

which calculates the future value of a present investment, $P$, after $n$ years in an interest-bearing bank account, with interest rate $r$ per year, with single compounding.

(b) Write a Python function, of the form

```
def multiple_compound(P, r, m, n):
    #Function body here.
    return #Don't forget to return the answer.
```

which calculates the future value of a present investment, $P$, after $n$ years in an interest-bearing bank account, with interest rate $r$ per year, compounded $m$ times per year.

(c) Write a Python function, of the form

```
def continuous_compound(P, r, n):
    #Function body here.
    return #Don't forget to return the answer.
```

which calculates the future value of a present investment, $P$, after $n$ years in an interest-bearing bank account, with interest rate $r$ per year, continuously compounded. Hint: you will need to import the exponential function from the math library, at the top of your file, by means of the following code:

```
import math
```

which will enable you to use the exponential function under the name `math.exp`

**Q2** Put your answers to this question in a file named `question2.py`

(a) Write a *recursive* function for calculating the $n$-th triangle number,

$$T_n = 1 + 2 + 3 + \cdots + (n-1) + n = n + T_{n-1} \text{ if } n \geq 1, \text{ otherwise } 0.$$

You function should take the form:

```
def triangle(n):
    #Function body here.
```

(b) Write a *recursive* function that calculates the $n$-th term in the sequence

$$a_n = 2a_{n-1} + 5a_{n-2},$$

where $a_0 = 1$ and $a_1 = 2$. Your function should take the form

```
def a(n):
    #Function body here.
```

(c) Write a *recursive* function that calculates the $n$-th term in the sequence

$$b_n = Cb_{n-1} + Db_{n-2},$$

where $b_0 = E$ and $b_1 = F$ and $C$, $D$, $E$, and $F$, are constants which should be passed to the function as inputs. Your function should take the form

```
def b(n, C, D, E, F):
    #Function body here.
```

(d) Define a Python function of the following form

```
def straddle_payoff(ST, K):
    #Function body here.
    return #Don't forget to return the answer.
```

that calculates the payoff of a "straddle option", given by:

$$f(S_T, K) = \begin{cases} K - S_T & \text{if } S_T < K \\ S_T - K & \text{if } S_T \geq K \end{cases}$$

(e) Write a *recursive* function which calculates and returns the value of a straddle option within the $n$ step binomial model. Rather than taking $u$ and $d$ as inputs, these two quantities should be calculated from $r$ and $\sigma$ using equations (3) and (4). The function definition should have the form:

```
def straddle_price(S0, K, T, sigma, r, n):
    #Function body here.
    return #Don't forget to return the answer.
```

**Q3** Put your answers to this question in a file named `question3.py`

(a) Write a function which approximately calculates definite integrals of the form:

$$\int_a^b f(x)dx,$$

where $f(x)$ is an arbitrary function. The Python function should have the following form:

```
def integrate(f, a, b, n):
    #Function body here.
    return #Don't forget to return the answer.
```

where $f$ is a Python function representing the mathematical function to be integrated, $a$ and $b$ are the integration limits and $n$ is the number of intervals used in the calculation. Extra marks will be awarded for using Simpson's rule rather than the Trapezium rule.

(b) Using numerical integration, together with equation (5), write a function which calculates the price of a straddle option in the limit $n \to \infty$ or equivalently $\delta t \to 0$. The function should have the form:

```
def straddle_bs(S0, K, T, sigma, r):
    #Function body
    return #Return the value of the option
```

(c) For the case of a straddle option, evaluate (by hand) the integral in equation (5) in terms of the cumulative normal function $\Phi(x)$. Then write a python function to evaluate the resulting expression. The Python function should have the form:

```
def straddle_analytic(S0, K, T, sigma, r):
    #Function body
    return #Return the value of the option
```

**Q4** Put your answers to this question in a file named `question4.py`

(a) Consider a bond with face value $\mathrm{F}$ that makes $\mathrm{m}$ coupon payments of $\mathrm{C}/\mathrm{m}$ each per year, and has $\mathrm{n}$ coupon payments remaining. If $\mathrm{r}$ is the interest rate for compounding $\mathrm{m}$ times per year, then equation (8) gives the value, $\mathrm{P}$, of the bond. Write a Python function which evaluates the bond price formula by direct summation using a for loop. The function should have the form:

```
def bond_price_loop(C, F, m, n, r):
    #Function body
    return #Return the value of the bond
```

Derive the bond price formula by evaluating the sum in equation (8) as a geometric series. Write a Python function to implement the formula, of the form:

```
def bond_price(C, F, m, n, r):
    #Function body
    return #Return the value of the bond
```

This function should give exactly the same output as the loop version.

(b) Write a Python function which uses Newton's method to solve the equation $\mathrm{f}(\mathrm{x}) = 0$ where $\mathrm{f}$ is a mathematical function of one variable. The Python function should have the form:

```
def newton(f, x0, dx, tol, max_steps):
    #Function body
    return #Return the approximate root
```

where $\mathrm{f}$ is a Python function of one variable, $\mathrm{x0}$ is the initial guess at its root, $\mathrm{dx}$ is the increment used to compute approximate derivatives of $\mathrm{f}$, $\mathrm{tol}$ is largest value of $|\,\mathrm{f}(\mathrm{x})\,|$ for which we will call $\mathrm{x}$ a root, and `max_steps` is the maximum allowed number of iterations. Make sure you test your root finder using some simple functions.

(c) Given the bond parameters $\mathrm{P}, \mathrm{C}, \mathrm{F}, \mathrm{m}, \mathrm{n}$, write a function that calculates the redemption yield $\mathrm{r}$:

```
def redemption(P, C, F, m, n):
    #Function body here.
    return #Don't forget to return something.
```

(d) Recall that the *term structure of interest rates* involves associating different interest rates with loans of different lengths. In what follows, consider only bonds that pay one coupon per year and have the same annual coupon value, $\mathrm{C}$, and the same face value, $\mathrm{F}$.

Write a function that, given a list of $\mathrm{N}$ bond prices, $[\mathrm{P}_0, \mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_{\mathrm{N}-1}]$, and a list of $\mathrm{N}$ maturities (number of payments remaining) for those bonds, $[\mathrm{n}_0, \mathrm{n}_1, \mathrm{n}_2, \ldots, \mathrm{n}_{\mathrm{N}-1}]$, calculates the corresponding list of bond yields $[\mathrm{r}_0, \mathrm{r}_1, \mathrm{r}_2, \ldots, \mathrm{r}_{\mathrm{N}-1}]$:

```
def term_structure(prices, maturities, C, F):
    #Function body here.
    return #Don't forget to return the list of yields.
```