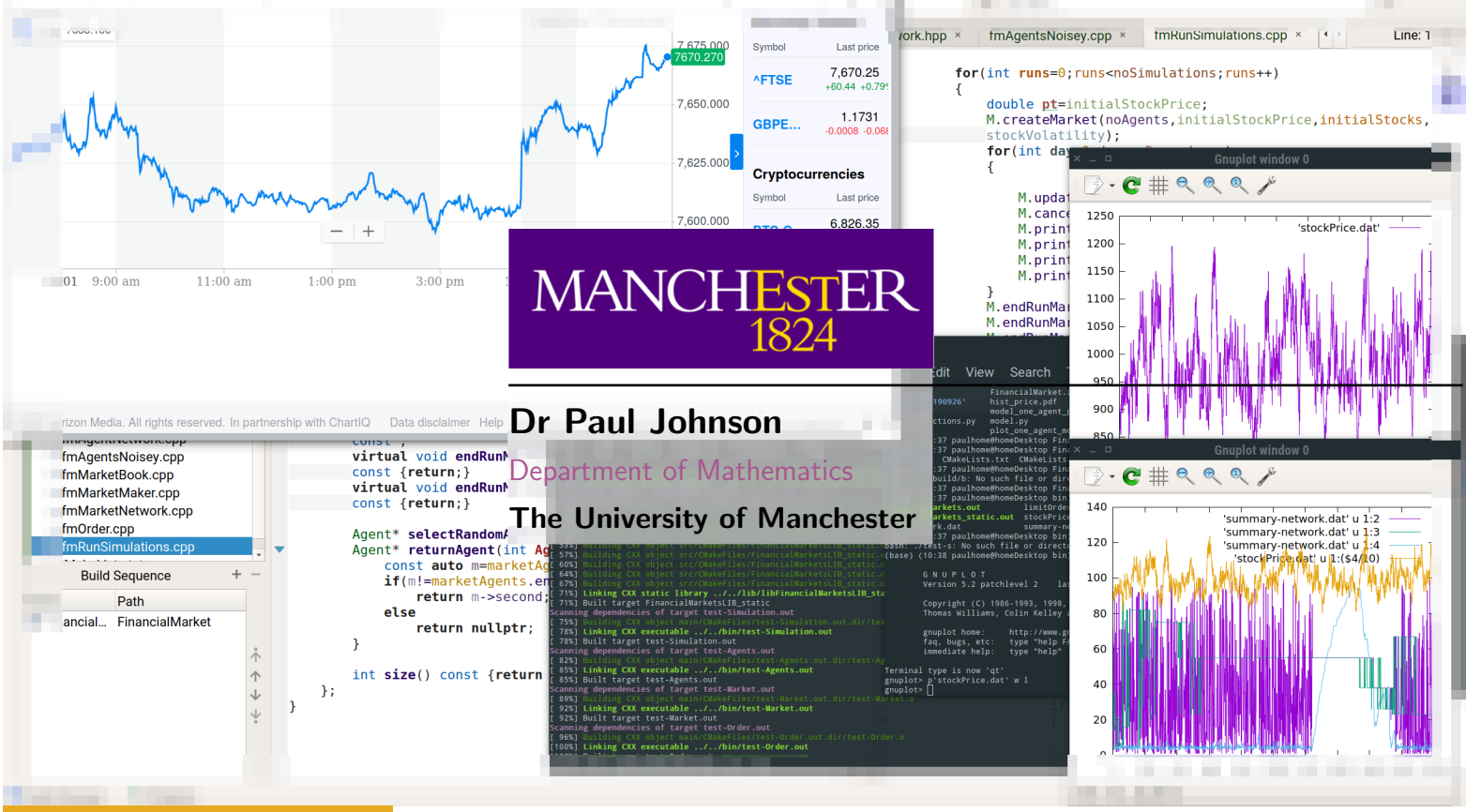

Dr Paul Johnson

Department of Mathematics

The University of Manchester

MATH60082:
Computational Finance

Semester 2 2020

Course Outline

MATH60082 Computational Finance

Dr Paul Johnson

Senior Lecturer in Financial Mathematics
2.107 Alan Turing Building
Department of Mathematics
Office hours: Tuesday 11:30am-12:30
email:
paul.johnson-2@manchester.ac.uk

website: https://personalpages.manchester.ac.uk/staff/paul.johnson-2/pages/math60082.html

Getting in Contact

- If you need to see me in person come along during my office hour (Tuesday 11:30-12:30) or catch me in the lab class.
- Preferred method of answering queries is to use the Blackboard forum (so that everyone can see the answer and I don't get asked same questions again and again). I am automatically notified every time someone posts and will try answer queries here within one working day.
- When answering emails I will normally post the query and response on the blackboard forum (anonymised) or point you towards a similar question on the forum.

Assessment

- This course is entirely assessed by project work.
- There will be four assignments in total, with two short mini tasks accounting for 5%, and two main assignments that will account for 40% and 50% of your final mark for this module.
- I will attempt to spread the assessment throughout the semester; and I will give around two weeks (for the main assignments) between giving out project details and the date to be handed in.
- The **DEADLINES MUST BE STRICTLY OBSERVED!!!**

Lectures

- Tuesdays 10am:- G.207, Alan Turing Building
- We will go through some of the background theory in the lectures.
- These notes contain gaps for you to fill in examples we will go through in the class.
- There is much more information than I can cover in the lecture time so you should try and read through these notes as though they were an accompanying textbook.

Examples/Lab Classes

- Thursdays 3pm:- G.105, Alan Turing Building
- There will be examples sheets accompanying the course with coding tasks. Selected problems will have full solutions.
- During the lab classes I will demonstrate coding up some of the example problems, as well as analysing the results.
- You are free to use **any** of the codes given out during lab classes (without reference), but you must code up your solutions yourself. DO NOT UNDER ANY CIRCUMSTANCES COPY/PASTE CODE FROM OTHER STUDENTS IN THE CLASS.

Important Dates

Support Computing Class	14:00pm - 16:00pm Wednesday 29th January 2020 - AT G.105
Mini Task 1	**Sunday 9th February 2020**
Mini Task 2	**Sunday 16th February 2020**
Main Assignment 1	**Sunday 15th March 2020**
Last Lecture	Monday 27th April 2020
Main Assignment 2	**Sunday 3rd May 2020**

Syllabus

1. Introduction to numerical computation. Numerical approximation and different methodologies. Discussion of errors, roundoff, truncation, discretisation.
2. Introduction to numerical solution of ODE's using multi-step methods.
3. Monte Carlo simulations; generation of random numbers. Pricing of European/Vanilla call/put options. Simple path-dependency options.
4. Binomial tree valuation of European/Vanilla call/put options. Application to early-exercise put options.
5. Introduction to solution of PDE's using finite-difference methods. Discussion of stability, consistency and convergence. Solution of European call/put options using Crank-Nicolson method. Solution of early exercise put options.
6. Advanced techniques: quadrature methods, body-fitted (free-boundary) coordinate systems.

ILOs

By the end of the course student should be able to:

(I) translate mathematical problems (well defined systems of mathematical equations) into computational tasks,

(II) to assess the accuracy of any numerical approximations, through numerical experimentation (and, when possible, by comparison with analytic solution),

(III) to process numerical results into a comprehensible form (including the use of standard graphical plotting packages), for presentation in a report,

(IV) to be able to give a critical assessment of the integrity of numerical methods and results.

Recommended Texts:

There are a number of texts on *Computational Finance*; most are fairly awful. Other texts on more general aspects of the subject are good on some areas of Computational Finance, but not on others. Here are some suggestions:

Text books:

- A good basic text for Mathematical Finance (also useful for MATH 39032/60008) is:

  Wilmott, P., Howison, S., Dewynne, J., 1995: The Mathematics of Financial Derivatives, Cambridge U.P. ISBN: 0521497892

- Alternatively, as an introductory text to the area:

  Wilmott, P., 2001: Paul Wilmott Introduces Quantitative Finance, 2nd Edition, Wiley. ISBN: 0471498629.

- For a very detailed (and expensive) look at mathematical finance:

  Wilmott, P., 2000: Paul Wilmott on Quantitative Finance, Wiley. ISBN: 0471874388

- For a more financial look at options and derivatives the following is excellent and is the course text for finance students (usually MBA or PhD) studying derivatives (with a decent treatment of binomial trees):

  Hull, J. C., 2002: Options, Futures and other Derivatives, 5th edition, Prentice Hall. ISBN: 0130465925.

- For a readable book on Stochastic Finance (including a good treatment of Monte Carlo methods):

  Higham, D.J. 2004: An introduction to financial option valuation. Cambridge University Press. ISBN 0521 54757 1 for paperback and ISBN 0521 83884 3 for hardback.

- For a book which describes numerous computational routines

  William H. Press, Brian P. Flannery and Saul A. Teukolsky 1992 Numerical Recipes in C: the art of scientific computing Cambridge University Press. ISBN-10: 0521431085 (there are also FORTRAN and C++ versions of this book).

- For a great book on the numerical solution of PDEs (written long before CF was invented!)

  G. D. Smith, 1986 Numerical Solution of Partial Differential Equations Oxford University Press. ISBN10: 0198596502

- For an excellent book for programing using C++

  M. Joshi, 2004 C++ Design Patterns and Derivatives Pricing. Cambridge University Press. ISBN 0 521 83235 7

Lecture 1

Introduction

Important - always remember:

Garbage In, Garbage Out

- this is the golden rule of computing!

1.1 Introduction

In this course we will be studying computational finance (or CF). Whilst basic financial mathematics problems *may* have analytical solutions (which even then may ultimately require an element of computation, such as the evaluation of special functions - see below), most *real* finance problems rely heavily on numerical (i.e. computational) techniques. Indeed, the calculation of the values of early exercise (e.g. American) options is inherently nonlinear, and as a consequence these must be tackled numerically.

The aim of the course is to give a broad outline of the main numerical techniques employed in the finance world. The list, in chronological order is:

- 'Exact' solutions - evaluation of the Black-Scholes formula
- Lattice (tree) methods
- Quadrature methods
- Monte Carlo methods
- Methods for partial differential equations (PDEs) - finite difference methods

Throughout the emphasis will be on a critical appraisal of methods, especially accuracy (i.e. errors).

The different types of errors can be categorised into the following main types:

- Roundoff errors
- Truncation and discretization errors
- Errors in modelling
- AND THE MOST IMPORTANT FOR BEGINNERS: Programming errors

Let us consider these in turn:

1.2 Roundoff errors

These arise when a computer is used for doing numerical calculations. Some typical examples include the inexact representation of (e.g. irrational) numbers such as π, $\sqrt{2}$. Roundoff and chopping errors arise from the way numbers are stored on the computer, and in the way arithmetic operations are carried out.

Example 1.1 (Floating Point Numbers)**.** When storing numbers in a computer, floating point style is used so

$$a = 134.2 = \underbrace{1.342}_{\text{mantissa}} \times \overbrace{10^2}^{\text{exponent}}$$

is stored as (using eight spaces)

$$a \sim \boxed{+}\boxed{1}\boxed{3}\boxed{4}\boxed{2}\boxed{+}\boxed{0}\boxed{2}$$

and $b = 0.09684143$ would be

$$b \sim \boxed{+}\boxed{9}\boxed{6}\boxed{8}\boxed{4}\boxed{-}\boxed{0}\boxed{2}$$

Assuming intermediate values are stored with 8 spaces, calculate the results

$$c = a + b - a$$

$$c = 10 \times b$$

$$c = b + b + b + b + b + b + b + b + b + b$$

Solution 1.1.

Whereas most of the time the numbers are stored on a computer is not under our control, the way certain expressions are computed definitely is. A simple example will illustrate the point. Consider the computation of the roots of a quadratic equation $ax^2 + bx + c = 0$ with the expressions

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

Let us take $a = c = 1$, $b = -28$. Then $x_1 = 14 + \sqrt{195}$, $x_2 = 14 - \sqrt{195}$. Now to 5 significant

figures we have $\sqrt{195} = 13.964$. Hence $x_1 = 27.964$, $x_2 = 0.036$. So what can we say about the error? We need some way to quantify errors. There are two useful measures for this.

Absolute error

Suppose that ϕ^* is an approximation to a quantity ϕ. Then the absolute error is defined by $|\phi^* - \phi|$.

Relative error

Another measure is the relative error and this is defined by $|\phi^* - \phi|/|\phi|$ if $\phi \neq 0$.

For our example above we see that $|x_1^* - x_1| = 2.4 * 10^{-4}$ and $|x_2^* - x_2| = 2.4 * 10^{-4}$ which look small. On the other hand the relative errors are $|x_1^* - x_1|/|x_1| = 8.6 * 10^{-6}$ and $|x_2^* - x_2|/|x_2| = 6.7 * 10^{-3}$. Thus the accuracy in computing x_2 is far less than in computing x_1. On the other hand if we compute x_2 via

$$x_2 = 14 - \sqrt{195} = \frac{1}{14 + \sqrt{195}}$$

we obtain $x_2 = 0.03576$ with a much smaller absolute error of $3.4 * 10^{-7}$ and a relative error of $9.6 * 10^{-6}$.

Note that roundoff error can be reduced by performing arithmetic operations at higher precision (i.e. more significant figures).

Definition 1.1 (Significant figures or *s.f.*). The number of digits that are used to express a number to the required degree of accuracy, starting from the first non-zero digit. So $x = 0.01524927$ is $x = 0.01525$ to $4s.f.$.

Example 1.2 (Nested Formula)**.** Evaluate the polynomial

$$f(x) = x^3 - 6.1x^2 + 3.2x + 1.5$$

Using $3s.f.$ at $x = 4.71$

Now try using a nested formula $f(x) = (((x - 6.1)x + 3.2)x + 1.5)$. Why are the answers different? What has changed between the examples?

Solution 1.2.

1.3 Truncation and discretization errors

These errors arise when we take the continuum model and replace it with a discrete approximation. For example suppose we wish to solve

$$\frac{d^2U}{dx^2} = f(x),$$

using Taylor series (more details later) we can approximate the second derivative term by

$$\frac{U(x_{i+1}) - 2U(x_i) + U(x_{i-1})}{h^2}$$

where we consider a set of x points $x_i = x_0 + i*h$, $i = 1, 2, \ldots$, where we have taken a uniform grid with spacing h say and node points x_i. As far the approximation of the equation is concerned we will have a *truncation error* given by

$$\tau(x_i) = \frac{d^2U(x_i)}{dx^2} - f(x_i) = \frac{h^2}{12}\frac{d^4U}{dx^4} + \ldots$$

Even though the discrete equations may be solved to high accuracy, there will be still an error of $O(h^2)$ arising from the discretization of the equations. Of course with more points, we would expect this error to diminish.

1.4 Errors in modelling

These arise for example when the equations being solved are not the proper equations for the problem. For example the Black Scholes equation has a number (many) underlying assumptions (some of which can be relaxed, others of which are still open to question). No matter how accurately the solution has been computed, it may not be close to the real solution because other factors have been neglected in the computation. This class of error is not dealt with in this course, but you should always be aware of the limitations of the model you are using.

1.5 Programming errors (bugs)

These are errors entirely under the control of the programmer. To eliminate these requires careful testing of the code and logic, as well as comparison with previous work. It is always useful to have benchmarks - for example an exact solution may exist (sometimes), or previous work with which it is possible to compare your numerical results. However for your problem for which there may not be previous work to compare with, one has to do numerous self-consistency checks with further analysis as necessary.

Example 1.3 (Spotting Bugs)**.** Assume we are calculating

$$I = \sum_{i=0}^{10} f(x_i), \qquad x_i = i \times 0.1$$

so x varies in steps of 0.1 over the interval $x \in [0, 1]$ Suppose have written the psuedo code

```
x=0 sum=f(0)
while(x<=1)
{
        x = x + 0.1
        sum = sum + f(x)
}
```

What is the problem in the above code?

Solution 1.3.

Click here to see my example on binder.

1.6 Benchmarking

As mentioned above, if you are lucky enough to have an exact solution with which to compare your numerical results - use it!

Consider the Black Scholes PDE (this has been/will be derived in other modules)

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.$$

Here V denotes the value of the option, S that of the underlying, t time, σ the volatility and r risk-free interest rate.

For the case of a European call option (final condition $C(t = T) = \max(S - X, 0)$), the price is

$$C(S, t) = SN(d_1) - Xe^{-r(T-t)}N(d_2)$$

whilst for a European put option (final condition $P((t = T) = \max(X - S, 0))$, the price is

$$P(S, t) = Xe^{-r(T-t)}N(-d_2) - SN(-d_1)$$

where

$$d_1 = \frac{\log(S/X) + (r + \frac{1}{2}\sigma^2)(T - t)}{\sigma\sqrt{T - t}},$$
$$d_2 = \frac{\log(S/X) + (r - \frac{1}{2}\sigma^2)(T - t)}{\sigma\sqrt{T - t}}.,$$

and

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}s^2} ds$$

which we recognise as the cumulative distribution function for a Normal distribution. This is an example of the need to undertake a calculation, even when an analytic solution exists.

Example 1.4 (Analysing Results)**.** Assume we try two numerical approximations $\hat{f}(x;n)$ and $\hat{g}(x;n)$ to a function $f(x)$, where n is a numerical parameter. Our mathematical theorems tells us

$$\lim_{n\to\infty} \hat{f}(x;n) = f(x) \text{ and } \lim_{n\to\infty} \hat{g}(x;n) = f(x)$$

Suppose we calculate the following results

n	$\hat{f}(x;n)$	$\hat{g}(x;n)$	n	$\hat{f}(x;n)$	$\hat{g}(x;n)$
100	5.30	5.30	10	4.40	3.00
101	5.30	5.31	20	4.90	3.69
102	5.30	5.32	40	5.15	4.39
103	5.30	5.33	80	5.28	5.08

What is the value to 2 *s.f.*?

Solution 1.4.

1.7 Euler's method - a simple procedure for ODEs

This method is a major component of Monte Carlo simulations!

Consider the initial value problem

$$\frac{dy}{dx} = f(x, y), \quad a \leq x \leq b,$$

subject to the initial condition $y(a) = \alpha$.

Note that this formulation also encompasses nonlinear ODEs.

Using a Taylor series expansion of $y(x + h)$ about $y(x)$ we obtain

$$y(x + h) = y(x) + h\frac{dy}{dx}(x) + \tfrac{1}{2}h^2\frac{d^2y}{dx^2}(x) + \dots$$

Rearranging this, we obtain

$$\frac{dy}{dx}(x) = \frac{y(x + h) - y(x)}{h} - \tfrac{1}{2}h\frac{d^2y}{dx^2}(x) + \dots$$

Substituting this into our ODE, neglecting the $O(h)$ terms leads to

$$\frac{y(x + h) - y(x)}{h} \approx f(x, y(x))$$

or

$$y(x + h) \approx y(x) + hf(x, y(x)).$$

Given $y(x = a) = \alpha$, this gives us a recipe for progressing the solution forwards in x. This is a marching/stepping process, with the solution obtained at $x = a + h$ (from $x = a$), $x = a + 2h$ (from $x = a + h$), ... $x = a + ih$ (from $x = a + (i - 1)h$), etc.

Advantages of the method

- It is simple (and also simple to program)
- It is robust
- h can be changed from step to step

Disadvantages of the method

- It is not very accurate

In general we say that a method is of order h^p if the error is of order h^p. In principle if h decreases, we should be able to achieve greater accuracy, although in practise roundoff error limits the smallest size of h that we can take. Euler's method is first-order accurate. Higher order schemes certainly exist, but Euler's method is certainly the most popular with financial practitioners using Monte Carlo simulations.