E-kursuse

"Computational Finance"

materjalid

Aine maht 6 EAP

Raul Kangro (Tartu Ülikool), 2011

Course introduction

Amount of credits: 6 EAP

Lecturer: Raul Kangro (Assoc. prof., Institute of Mathematical statistics, Tartu University)

Target Group: Master students of financial and actuarial mathematics program

Goals: After successful completion of the course the students derive and implement numerical methods for pricing various financial options. More precisely, after successful completion of the course the students

1. know the basic properties and derivation procedures of the partial differential equations arising in mathematical finance and can apply the procedures in the case of new market models,
2. can derive numerical methods for computing the prices of financial options and are able to implement them in Python programming language,
3. obtain practical skills for computing the prices of common options,
4. Know and can use some popular methods of estimating the parameters of various market models and for computing the quantities that are necessary for constructing hedging portfolios.

Topics of the course: Definition and types of financial options. Arbitrage principle. Black-Scholes market model. Methods for estimating market parameters: maximal likelihood method, least squares minimization. Monte-Carlo method for pricing options. Derivation of Black-Scholes Partial Differential Equation (PDE) for European options. Alternative approaches for deriving PDE. The idea of finite difference methods. Explicit method, implicit method, Crank-Nicolson method. Stability of numerical methods. Computing option prices with given accuracy. Derivation of PDE for Asian options. A finite difference method for pricing Asian options. PDE and an explicit numerical method for options depending on two stocks. The idea of finite element methods. Derivation of the method for pricing Europaen options.

Independent works: There are 3 theoretical homeworks (specified in electonic course materials) and 10 practical homeworks (handed out in labs). The homeworks are due one week after they were handed out. The late submissions are allowed but the maximal score for such submissions is reduced by 50%. The solutions of the practical homeworks have to be submitted throw the Moodle web page of the course as Python files, the solutions of the theoretical homeworks can be submitted either via Moodle or in the lecture. The practical homeworks give up to 3 points each; first two of the theoretical homeworks give up to 3 points and the last one up to 4 points. The maximal total score for homeworks is 40 points.

Requirement to be met for final assessment: At least 20 points (50%) for homeworks is required for qualifying for the final examination.

Composition of the final grade: 60% of the total score is given by the final exam, 40% comes from the homework assignments. The final exam consists of a theoretical part (derivation of a numerical method, derivation of PDE for option pricing) and a computational part (pricing of a concrete option). The final grade is determined by the total score as follows: the score less than 50 gives F, from 50 to 59.9 gives E, from 60 to 69.9 gives D, from 70 to 79.9 gives C, from 80 to 89.9 gives B and a score of 90 or more gives A

E-learning activities: The course materials are divided between 16 study weeks and can be used for independent study or as supporting materials for the lectures and the computer labs. The lecture notes and lab handouts contain all of the theoretical materials that is required for this course. The solutions of the 10 practical homeworks have to be submitted through Moodle. The soutions of the theoretical homeworks can be handed in on paper or submitted electronically through Moodle. The final examination has to be taken in person.

Additional information: Raul Kangro raul.kangro@ut.ee

Creation of the web page of the course was supported by European Union

Computational Finance

Raul Kangro

Fall 2011

Contents

Introduction

In the early 1970s, Fisher Black and Myron Scholes [1] made a major breakthrough by deriving a differential equation that must be satisfied by the price of any derivative security dependent on a non-dividend-paying stock. They used the equation to obtain values for European call and put options on the stock. Their work had a huge impact on how options were viewed in the financial world. Options are now traded on many different exchanges throughout the world and are very popular instruments for both speculating and risk management. Because of the popularity of derivative securities there is a great need for good and reliable ways to compute their prices.

In order to price an option one has to complete several steps:

1. specify a suitable mathematical model describing sufficiently well the behavior of the stock market;
2. calibrate the model to available market data;
3. derive formula or equation for the price of the option of interest;
4. compute the price of the option.

Very often the last step requires the usage of some numerical methods because usually the explicit formulas for the price of the option is not available.

In this course we pay very little attention to the first two steps and concentrate our attention to the last two. More precisely, there are two main approaches to completing those steps, namely probabilistic approach (where option prices are expressed as expected values of some random variables) and Partial Differential Equations (PDE) approach (where option prices are expressed as solutions to certain differential equations). This course is mainly about the PDE approach, although some aspects of the probabilistic approach are also considered.

The lecture notes are self-contained and contain (together with the lab materials) all theoretical knowledge that is required for passing the course. There is a huge number of books where the aspects of the computational finance are discussed. For additional reading I recommend [**?**, WHD]or an alternative introduction to mathematical finance and finite difference methods, [5] for more extensive discussion of the theory and practice of computational finance, [2] for extensive treatment of Monte-Carlo methods in finance and [3] for details of Finite Element methods.

Chapter 1

Options on one underlying

1.1 Definitions and examples

We adopt a nonstandard, but quite general definition of financial options

Definitsioon 1 *An* option *is a contract giving it's holder the right to receive in the future a payment which amount is determined by the behavior of the stock market up to the moment of executing the contract.*

Option contracts are classified according to several characteristics including

- possible execution times (a fixed date vs a time interval),
- the number of underlying assets,
- how the value of option depends on the asset prices (depending on the price at the execution time vs a path dependent value of the asset prices).

In order to clarify the meaning of the definition, let us look at some examples.

Example 2 The right to buy 100 Nokia shares for 400 Euros exactly after 3 months (say, November 30th, 2011) .

This is an *European* (with fixed execution date) *Call* (the right to buy) option, which is equivalent to the right to receive after three months the sum of $100 \cdot \max(S(T) - 4, 0)$ Euros, where $S(T)$ denotes the price of a Nokia share at the specified date.

Example 3 The right to sell one Amazon.com share during next 6 months for $200.

This is *American* (with a free execution time) *Put* (the right to sell) option.

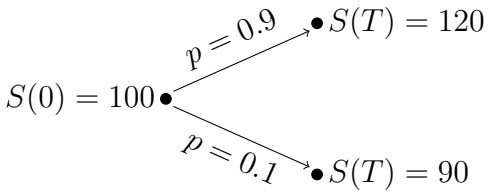
Example 4 The right to exchange after one year 10000 USD for Euros with the rate that is the average of the daily exchange rates in the one year period.

The last an *Asian* option that is an example of path dependent options.

1.2 Strange things about pricing options.

If an investor makes a decision about buying or selling a financial instrument, it is customary to consider the expected return and risk of the investment. Investment decision is usually made based on those quantities and investor's risk tolerance, so there is a different "right price" for each investor. It turns out that usual thinking models do not help to determine the right price for an option contract. In order to clarify this point, let us consider some simple toy models for stock price behavior.

First, suppose that the at a future time T there are only two possible stock prices:



Assume for simplicity that the risk free interest rate is 0 (meaning that it is not possible to ear interest by depositing money in bank and it is possible to borrow money so that you have to pay back exactly the sum you borrowed). Let us consider and option to buy a at time T 10 shares of stock for 99. Now the value of this option at time T is 210 if $S(T) = 120$ (since you can buy 10 shares for 99 when the market price is 120) and it is worthless, if $S(T) = 0$. So buying the option seems to be a good investment possibility, if the price is not too high: the expected value of the option at time T is

$$0.9 \cdot 210 + 0.1 \cdot 0 = 189.$$

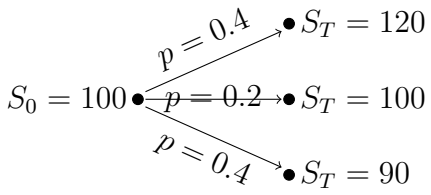
Note also that if the probability of $S(T) = 120$ is 0.1, then the expected value is only 21, so the expected value of the option depends strongly on the market probabilities. So it is natural to think that the fair price of the option should also depend on market probabilities.

But before deciding to buy the option we can consider alternative investment possibilities. It turns out that we can achieve exactly the same outcome by forming an investment portfolio consisting form a loan of 630 and 7 shares of stock: if $S_T = 120$, then the value of the portfolio is $7 \cdot 120 - 630 = 210$ and if $S_T = 90$, then the value is $7 \cdot 90 - 630 = 0$. The cost of forming the portfolio at $t = 0$ is 700-630=70 (we get 630 from the loan and have to add 70 of our own money to buy 7 shares for 100 each). So it is clear that at this market no sensible person pays more than 70 for the option. Moreover, if there exists any person willing to buy the option for more than 70, there is a possibility for anyone to earn money at the market without any risk of losing anything: one should just sell the option for the price and use 70 to set up the portfolio to cover the liabilities at time T. Such opportunities are called arbitrage opportunities and usually it is assumed that there are no arbitrage opportunities at the market.

By similar argument we can argue, that the price of the option can not be less than 70, so the price of the option is completely determined by the market model. Moreover, the

arguments we used did not depend on the probabilities of the up and down movements, so the option price does not depend on expected value and the risk of the option contract.

Let us consider now a similar market model with three different stock prices at time T:



It is easy to check that the price of the same option considered for the previous market model can not be larger than 70 (since the same portfolio as before requires 70 of initial investment and is worth at least as much as the option at time T for all possible values of $S(T)$. It is also possible to show that the price of the option can not be less than 10 (consider setting up the portfolio with one option and -7 shares of stock). Moreover, it is also possible to show that from the arbitrage principle it follows only that the value of the option is between 10 and 70.

Homework exercise 1 *(Deadline September 13, 2011) Show that in the case of the last market model, for prices of option between 10 and 70 it is not possible to form a portfolio from a bank account, stock holding (positive or negative) and by buying or selling the option such that the initial cost of the portfolio is zero, but the value at time T is positive for all values of $S(T)$. (Hint: use the inequalities at time T to show that the value of the corresponding portfolio is strictly greater than 0 at time 0).*

Consider now a second option that pays 60, if $S(T) = 100$ and 0 otherwise. If we consider this option separately from the first one, then by the arbitrage principle it follows only that the price of the option is between 0 and 60. But if it is possible to buy/sell both of those options then from the arbitrage principle it follows that the sum of the prices of the options considered has to be 70 (try to prove it!). So there are strong consistency requirements between the prices of different options.

Based on the two simple models we can make the following conclusions:

- Naive pricing approaches (based on the expected return and risk) do not work.
- In the case of some market models the option price is determined completely by the model (and no arbitrage condition)
- There are market models, for which the option prices are not determined completely but prices of different options have to be consistent with each other.

1.3 A stock market model, no arbitrage condition

In order to use mathematics in option pricing one has to start by specifying a model for stock price evolution and describing the conditions for trading.

1.3.1 Black-Scholes model.

A relatively simple but useful market model is so called Black-Scholes model, which assumes that the stock price changes according to the stochastic differential equation

$$dS(t) = S(t)(\mu(t)\,dt + \sigma(S(t), t)\,dB(t)), \tag{1.1}$$

where $S(t)$ is the stock price at time t, μ is the average growth rate of the stock price, σ is the volatility and B is the standard Brownian motion. Technically correct discussion of the meaning of the equation is out of scope of this course but for intuitively it means for small noninteracting time periods (t_{i-1}, t_i) we have

$$\begin{aligned} S(t_i) &\approx S(t_{i-1}) + S(t_{i-1})(\mu(t_{i-1})h_i + \sigma(S(t_{i-1}), t_{i-1})X_i) \\ &= S(t_{i-1})(1 + \mu(t_{i-1})h + \sigma(S(t_{i-1}), t_{i-1})X_i), \end{aligned}$$

where $h_i = t_i - t_{i-1}$ and $X_j \sim N(0, \sqrt{h_i})$, $j = 1, 2, \ldots, N$ and X_i are independent normally distributed random variables. This relation enables us to simulate sample trajectories according to the market model. The figure 1.1 shows 5 stock price trajectories illustrating the fact that future stock prices are random, so each time we compute a trajectory, we get a different one. In addition to the market model we make several additional simplifying assumptions:

- the risk free interest rate is a known constant r and is the same for lending and borrowing;
- it is possible to trade continuously and with arbitrarily small fractions of a stock;
- there are no transaction costs;
- it is not possible to make riskless profit by trading on the market.

It is clear, that some of the additional assumptions do not hold in practice and that the Black-Scholes model, at least with constant parameters μ and σ, is often not in a very good accordance with real market behavior, but still it is a good starting point for mathematical modeling of the market behavior.

1.3.2 Self-financing investment strategies

We call an *investment strategy* a rule for forming t each t in a period $[t_0, T]$ a portfolio consisting of a deposit $b(t)$ to a riskless bank account (if $b(t)$ is negative, then it corresponds to borrowing money) and of holding $\eta(t)$ shares of the stock. Both $b(t)$ and

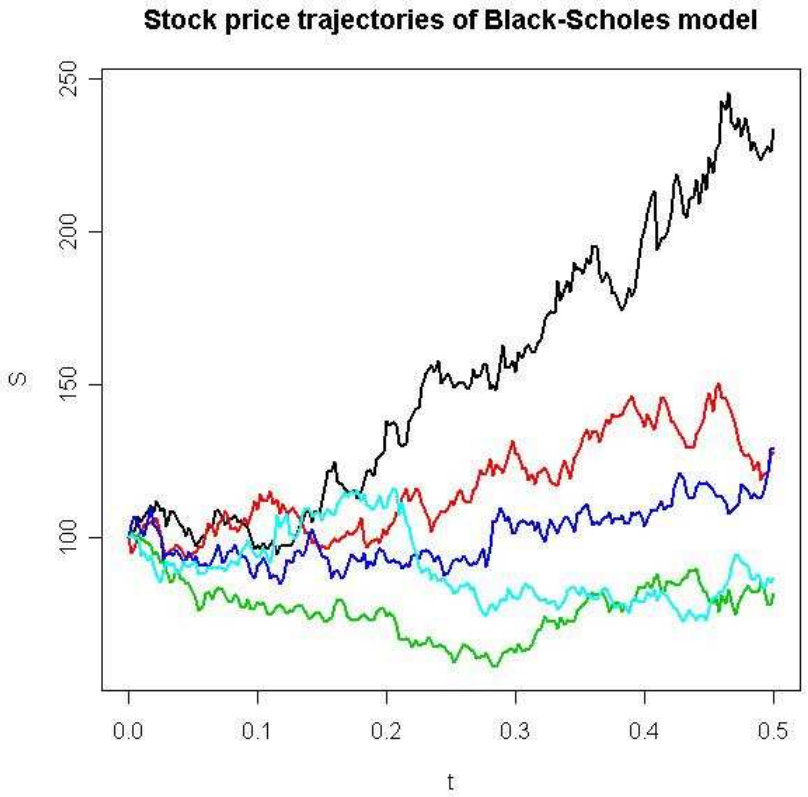


Figure 1.1: Sample trajectories of the stockprice process following Black-Scholes model

$\eta(t)$ may depend on the history up to time t (including the current value) of the stock prices but are not allowed to depend on the future values. An investment strategy is called *self-financing* if the only changes in the bank account after setting up the initial portfolio are the results of accumulation of interests of the same account, cash flows coming from holding the shares of the stock (eg dividend payments), or reflect buying or selling the shares of the stock required by changes of η, and if all cash flows that come from the changes of $\eta(t)$ are reflected in the bank account.

Let $X(t)$ denote the value of a self-financing portfolio at time t. Assume that the stock pays its holders continuously dividends with the rate D percent (realistic if the "stock" is a foreign currency, for usual stocks $D = 0$). Then in an infinitesimally small time interval dt the value of a self-financing portfolio changes according to the equation

$$dX(t) = r \cdot (X(t) - \eta(t)S(t))\, dt + D\eta(t)S(t)\, dt + \eta(t)\, dS(t). \tag{1.2}$$

The first term on the right hand side corresponds to the condition that all money that is not invested in the stock, is deposited to (or borrowed from) a bank account and bears the interest with the risk free rate r, the second term takes into account dividends and the last term reflects the change in the value of the portfolio coming from the change in

the stock price. The value of a self-financing portfolio at any time $t > t_0$ is determined by the initial value $X(t_0) = X_0$ and the process $\eta(t),\ t \in [t_0, T]$.

Since nobody can borrow infinitely large sums of money, only such investment strategies for which the value of the portfolio is almost surely bounded below by a constant, are allowed.

1.3.3 No arbitrage condition.

In general, no arbitrage assumption states that it is not possible to make risk free profits by investing in the market. More precisely, it should not be possible to form a portfolio such that it does not cost any money today, the value of the portfolio is never negative during it's lifetime and has a positive value with nonzero probability at some future date. We need a corollary of the general no arbitrage condition.

Lemma 5 *(No arbitrage condition) If a self-financing portfolio produces exactly the same cash flows as holding an option, then the initial value of the portfolio and the option price have to be equal.*

Proof. If the price of the option is higher then we sell the option, form the self-financing portfolio and some money will be left for us to spend without any risk. If the option price is lower, then we buy the option and use the opposite investment strategy (having $-\eta(t)$ shares at time t). Again some money will be left over and we can spend it without any risk. Since such possibilities should not exist on a real market (at least for long), the option price and the initial value of the portfolio have to be the same.□

1.4 Itō's formula and Monte-Carlo method for pricing European options

We have specified a stochastic differential equation for the stock price evolution but it is not enough. We want also to consider functions of the stock price and differentiate them with respect to time. It turns out, that in the case of stochastic variables the usual rules of calculus do not hold and we need new differentiation rules (stochastic calculus).

1.4.1 Itō's Formula.

The following result proved by Japanese mathematician Kiyosi Itō in 1942, is of great importance in the theory of mathematical finance.

Lemma 6 Itō 's formula *Assume that $f(y,t)$ is a twice differentiable function of two variables and that a stochastic process Y satisfies the stochastic differential equation*

$$dY(t) = \alpha(t)\,dt + \beta(t)\,dB(t),$$

where α and β are continuous processes and B is the Brownian motion. Then

$$df(Y(t),t) = \left(\frac{\partial f}{\partial t}(Y(t),t) + \frac{\beta(t)^2}{2}\frac{\partial^2 f}{\partial y^2}(Y(t),t)\right)\,dt + \frac{\partial f}{\partial y}(Y(t),t)\,dY(t).$$

Example 7 Let us show that if μ and σ are constant then the process

$$S(t) = S(0)\cdot e^{(\mu-\frac{\sigma^2}{2})t+\sigma\,B(t)},\ t\in[0,T] \tag{1.3}$$

is a solution to the equation (1.1).

Denote

$$f(y,t) = e^{(\mu-\frac{\sigma^2}{2})t+\sigma\,y},\ Y(t) = B(t),$$

then $S(t) = f(Y(t),t)$. Since

$$\begin{aligned}
\frac{\partial f}{\partial t}(y,t) &= (\mu-\frac{\sigma^2}{2})f(y,t),\\
\frac{\partial f}{\partial y}(y,t) &= \sigma\,f(y,t),\\
\frac{\partial^2 f}{\partial y^2}(y,t) &= \sigma^2 f(y,t),
\end{aligned}$$

then, according to Itō's formula, we have

$$\begin{aligned}
dS(t) &= \left((\mu-\frac{\sigma^2}{2})S(t) + \frac{1}{2}\sigma^2 S(t)\right)dt + \sigma\,S(t)\,dB(t)\Big)\\
&= S(t)(\mu\,dt + \sigma\,dB(t)).\square
\end{aligned}$$

Exercise 1 *Compute $df(B(t))$ for $f(y) = y^2$.*

Exercise 2 *Let $Y(t) = e^t\cos(B(t))$. Compute $dY(t)$.*

1.4.2 Estimating the parameters of BS model

There are two different approaches for estimating the parameters of the market model.

1. Fitting the market model to historical data.
2. Fitting the option prices derived from a market model to the actual prices of theoretical options.

Practitioners usually prefer the second approach since, according to the efficient market hypothesis, the traded options should have correct prices and it is highly desirable for an option pricing framework to produce correct prices to traded options. One has to use the first approach if the prices of traded options are not available or if we want to check the validity of our market model for a concrete stock. Let us discuss briefly both approaches.

Fitting the historical data

For simplicity, we assume that we have available n historical observation S_i, $i = 1, 2, \ldots, n$ of stock prices at equally spaced timesteps (eg closing prices).

In the case of constant parameters μ and σ we get, using the Black-Scholes model (1.1) and Itō's formula, that

$$d(\ln S(t)) = (\mu - \frac{\sigma^2}{2})\, dt + \sigma\, dB(t),$$

hence for any time moments t_1 and $t_2 > t_1$ we have

$$\ln \frac{S(t_2)}{S(t_1)} = (\mu - \frac{\sigma^2}{2})\,(t_2 - t_1) + \sigma(B(t_2) - B(t_1)).$$

Thus $x_i = \ln \frac{S_{i+1}}{S_i}$ are (if our assumption about the market model is correct) values of normally distributed iid random variables, $x_i \sim N((\mu - \frac{\sigma^2}{2})\Delta t, \sigma\sqrt{\Delta t})$, where Δt is the time interval between observations (usually measured in years). Therefore we can find estimates for μ and σ as follows:

$$\bar{\sigma} = \frac{\mathrm{std}(x)}{\sqrt{\Delta t}},\ \bar{\mu} = \frac{\mathrm{mean}(x)}{\Delta t} + \frac{\bar{\sigma}^2}{2}.$$

Unfortunately, if we test the normality of the logarithms of the quotients of the stock prices by some well-known statistical test, then it usually turns out that we have to reject the normality hypothesis.

As an example, let us conside the closing prices of a Cisco share. The price trajectory is given in the Figure 1.2. From the formulas above we get (assuming 255 working days per year)

$$\bar{\sigma} = 0.3506234,\ \bar{\mu} = -0.1887979.$$

Unfortunately it is not safe to use the Black-Scholes market model with constant parameters for pricing options on Cisco stock since the statistical test tell us that we should use the normality assumptions. For example, Shapiro-Wilk normality test (see Wikipedia!) gives for logarithmic returns the following result:

```
        Shapiro-Wilk normality test

data:  returns
    W = 0.8232, p-value = 4.712e-11
```

Hence, the probability to get stock prices similar to the actual ones when Black-Scholes market model with constant coefficients holds, is extremely small (less than 0.000000000047), so it is not reasonable to believe in the validity of this simple market model. If we do not want to assume that the parameters are constant, we may start with approximating the market model:

$$\frac{S(t_i) - S(t_{i-1})}{S(t_{i-1})} \approx \mu(t_{i-1})\,(t_i - t_{i-1}) + \sigma(S(t_{i-1}), t_{i-1})(B(t_i) - B(t_{i-1})).$$

Figure 1.2: Closing prices of Cisco share, March 10-September 9, 2011. Source: http://finance.yahoo.com

Next, we introduce a finite number of unknown parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$ and make an assumption how the functions $\mu = \mu_\theta$ and $\sigma = \sigma_\theta$ depend on those parameters. One way to find those parameters is to maximize the log-likelihood function: if Y_i are random variables with (conditional) probability density functions f_i, then the log-likelihood function of the values y_i is

$$\sum_i \ln f_i(y_i).$$

Since in our case the random variables $Y_i = \frac{S_i - S_{i-1}}{S_{i-1}}$ are according to the approximate market model normally distributed with mean $\mu_\theta(t_{i-1})\Delta t$ and standard deviation $\sigma_\theta(S_{i-1}, t_{i-1})\sqrt{\Delta t}$ we have to maximize the function

$$f(\theta) = -\sum_i \left(\frac{(Y_i - \mu_\theta(t_{i-1})\Delta t)^2}{2\sigma_\theta(S_{i-1}, t_{i-1})^2 \Delta t} + \ln \sigma_\theta(S_{i-1}, t_{i-1}) \right).$$

or minimize the negative of the function. Since f is usually a quite complicated function of the parameter vector θ, it may have several local extremum points, so one has to be careful in accepting an output of an optimization procedure as the solution of our parameter estimation problem.

Fitting the data of traded options

Starting from a market model we derive prices of various options. In the simplest cases we have explicit formulas, in more complicated cases we have to solve certain equations

to get the option prices, but always we may thing that there is a function depending on market parameters that gives us the option prices. Suppose that we know the current prices $V_1, V_2, \ldots, V_m$ of m different options and that $f_i(\theta)$ are the functions that give the option prices for (unknown) market parameters θ. Then we have m equations:

$$f_i(\theta) = V_i,\ i = 1, \ldots, m.$$

Usually the number of unknown market parameters is much smaller than the number of available option prices, so the system of equations is solved in the least squares sense by minimizing the function

$$F(\theta) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\theta) - V_i)^2.$$

Again there may be several local minima, so one should check carefully a possible candidate for the optimal solution.

Let us consider again the example of Black-Scholes market model with constant coefficients. It is known that in the case of this model the prices of European put and call options with exercise date T and strike price E at time t can be computed by Black-Scholes formulas as follows:

$$C(S,t,T) = Se^{-D(T-t)}\Phi(d_1) - Ee^{-r(T-t)}\Phi(d_2),$$
$$P(S,t,T) = -Se^{-D(T-t)}\Phi(-d_1) + Ee^{-r(T-t)}\Phi(-d_2),$$

where

$$d_1 = \frac{\ln(\frac{S}{E}) + (r - D + \frac{\sigma^2}{2})(T-t)}{\sigma\sqrt{T-t}},\ d_2 = d_1 - \sigma\sqrt{T-t}$$

and Φ is the cumulative distribution function of the standard normal distribution. Here D is the rate of proportional dividend payments, r is the risk free interest rate and σ is the volatility of the stock. So if we assume that r, D, t, T are fixed then we have functions that for any given volatility and exercise price give us the values of corresponding options. Since options are traded on the market, the prices of standard call and put options are available for several exercise prices. So we can try to pick the value of σ so that we get the observed prices from Black-Scholes formula. Moreover, if the Black-Scholes market model with constant coefficients holds, we should be able to find a value of σ that gives the observed prices for all strike prices for which we have data. Usually this is not the case: for each strike price we get a different value of σ (so called volatility smile effect). If this is the case, then we can be sure that Black-Scholes market model with constant volatility does not hold.

As a concrete example, let us consider finding the volatility from the market prices of call options for Cisco shares. Part of the data available for 4 month options expiring on January 20, 2012 was on September 19, 2011 as follows (source: `http:/finance.yahoo.com`)

E	11	12.5	14	15	16	17.5	19	20	21	22.5
Price	5.6	3.95	2.76	2.07	1.48	0.75	0.34	0.20	0.12	0.06

The share price was at that moment \$16.26 and the share does not pay proportional dividends, so $D = 0$. Since 4 months corresponds to one third of a year, we take $T = \frac{1}{3}$ and $t = 0$ in the Black-Scholes formula. For the risk free interest rate we use $r = 0.02$. Let us consider first the strike price $E = 11$, the graph of theoretical prices as a function of the volatility is given at figure 1.3. the horizontal line indicates the observed price

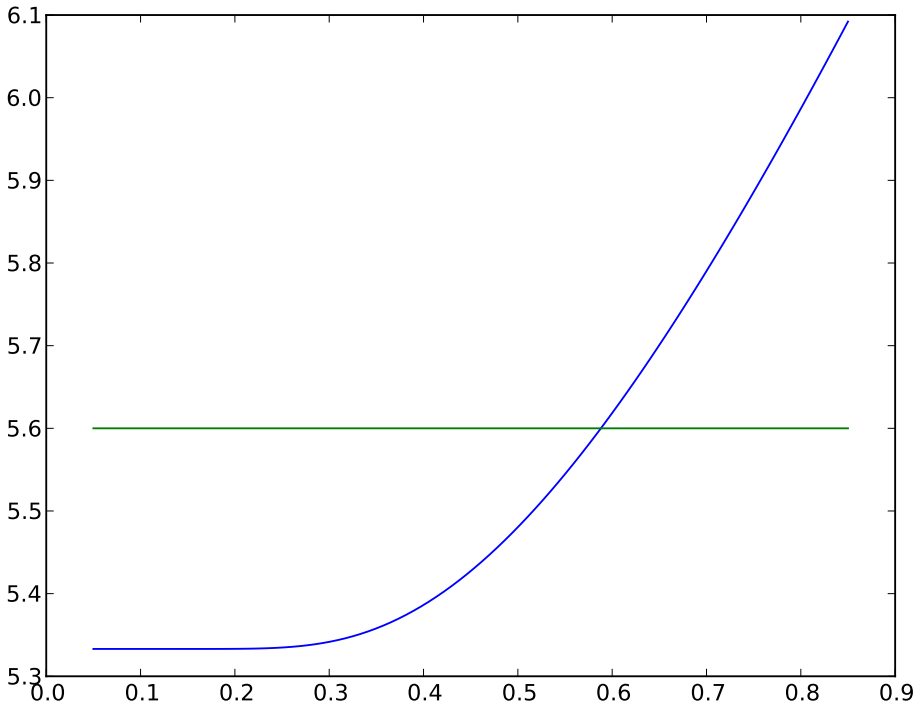


Figure 1.3: Call option price as a function of volatility for strike price $E = \$11$

\$5.6. From the graph we see that there is a single volatility that gives us the observed price, the approximate value of the volatility is 0.59. By using a numerical solver we get that the volatility giving us the observed price is 0.58804108.

Similarly we can find the volatilities that correspond to the other observed option prices for different strike prices. The results obtained are given in the figure 1.4. As we see, the volatilities that correspond to different strike prices are not equal. Thus either Black-Scholes market model with constant volatility does not hold, or there are arbitrage possibilities at the market. It is safer to assume that the model does not hold, so a better model is needed for pricing real options.

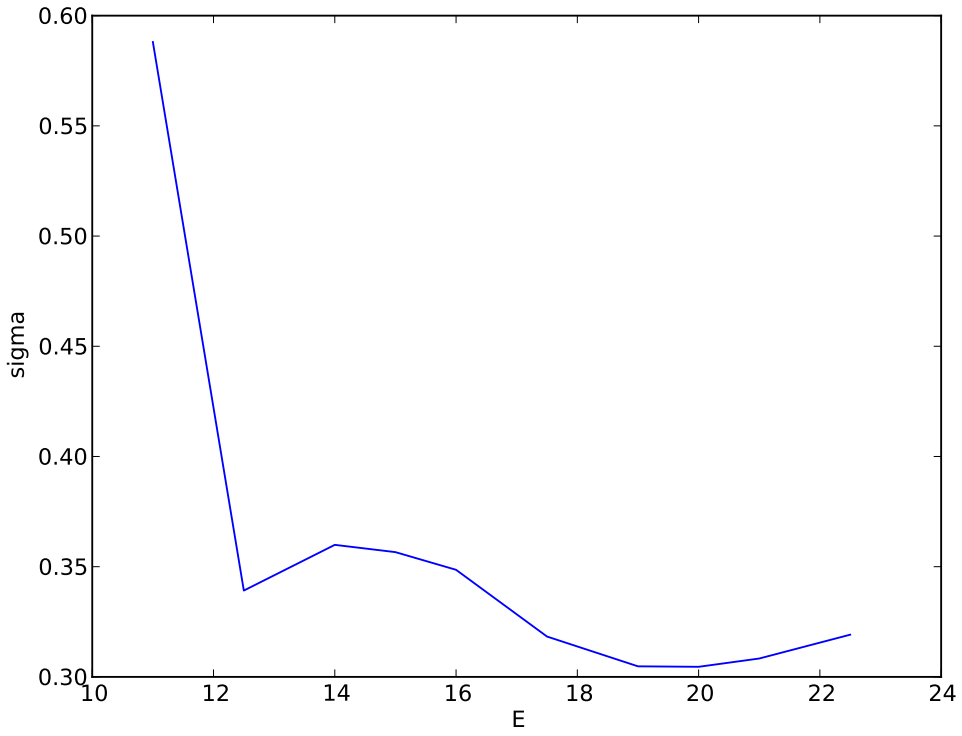


Figure 1.4: Implied volatilities for different exercise prices

1.4.3 Monte-Carlo method for computing the prices of European options.

Suppose we know that an European option can be replicated (exactly the same outcome can be achieved by) a self-financing trading strategy. Let us recall, that the value of a portfolio corresponding to a self-financing trading strategy, satisfies the equation

$$dX(t) = r \cdot (X(t) - \eta(t)S(t))\,dt + D\eta(t)S(t)\,dt + \eta(t)\,dS(t).$$

Note that we can rewrite the equation in the form

$$d(e^{-rt}X(t)) = \eta(t)e^{-rt}S(t)((\mu(t) - r + D)\,dt + \sigma(S(t), t)\,dB(t)).$$

Consider the case $\mu(t) \equiv r - D$. Then we have on the right hand side only the term with $dB(t)$ and, according to the theory of stochastic processes, the expected value of $e^{-rt}X(t)$ is the same for any t, ie $E(e^{-rt}X(t)) = X(0)$. Therefore, if an investment strategy replicates an option with payoff $p(S(T))$, then $X(T) = p(S(T))$ and hence the price of the option at time $t = 0$ is

$$X(0) = E(e^{-rT}p(S(T)),$$

hence the option price can be found by computing numerically (or analytically) the expected value in this case.

On the other hand, we prove later that under the assumptions we made about the stock market behavior every option can be replicated and the replication strategy does not depend on μ. Thus we can find the correct price by taking $\mu = r - D$ in the market model 1.1 and evaluating the expected value of the discounted payoff. Moreover, it can be shown that even when exact replication is not possible, option prices can still be expressed as expected values of some random variables.

One way to compute an expected value of a stochastic variable numerically is to generate n values of the variable and compute the average of the result. This is called Monte-Carlo method.

Lemma 8 (MC error) *Assume that $Y_1, Y_2, \ldots$ is a sequence of iid random variables with $EY_i = a$ and $DY_i = \sigma^2 < \infty$. Denote $H_n = \frac{\sum_{i=1}^n Y_i}{n}$. Then, for sufficiently large values of n we have*

$$P(|H_n - a| \geq \varepsilon) \approx 2\,\Phi(-\frac{\varepsilon\sqrt{n}}{\sigma})$$

and hence with probability $1 - \alpha$ we have

$$|H_n - a| \leq \frac{-\Phi^{-1}(\frac{\alpha}{2})\sigma}{\sqrt{n}} \tag{1.4}$$

where Φ is the cumulative distribution function of the standard normal distribution.

Exercise 3 *Derive the estimates of Lemma 8 from Central Limit Theorem.*

As we see, the error behaves like $\frac{1}{\sqrt{n}}$, so the convergence of the method is quite slow. We saw earlier (see formula (1.3)) that if the Black-Scholes model with a constant volatility σ holds then the stock price $S(T)$ corresponding to the trend $\mu = r = D$ is given by

$$S(T) = S(0)e^{(r-D-\frac{\sigma^2}{2})\cdot T + \sigma B(T)}.$$

Generating the prices is easy: since $B(T)$ is normally distributed with variance T, we can just generate values of a random variable distributed according to the standard normal distribution, multiply those values with $\sqrt{T}$ and use the results for $B(T)$ in the formula above. Thus, in this case we can use MC method to compute the prices of any European options: we just generate the values of stock prices, compute the average of the discounted pay-off values and estimate the error of the result by (1.4). If we want to compute with a given accuracy, then we just have to keep generating the values of stock prices $S(T)$ until the error estimate is less than the desired accuracy.

Very often it is not possible to generate $S(T)$ values that correspond exactly to the stochastic differential equation; then it is necessary to use some approximation methods. One such method is the Euler method, where we divide the interval $[0, T]$ into m

equal subintervals and use the approximations (int the case of Black-Scholes market model)

$$S_{i+1} = S_i(1 + (r - D)\,\Delta t + \sigma(S_i, t_i)\sqrt{\Delta t}X_i), i = 0, \ldots, m-1,$$

where S_i are approximations to $S(i\Delta t)$, $\Delta t = \frac{T}{m}$ and $X_i \sim N(0,1)$. Instead of $S(T)$ we use S_m, thus we use Monte-Carlo method to compute an approximate value of $\hat{V}_m$, where

$$\hat{V}_m = E[e^{-rT}p(S_m)].$$

It is known that if p is continuous and has bounded first derivative (ie it is Lipshitz continuous), then

$$|V - \hat{V}_m| = \frac{C}{m} + o(\frac{1}{m}),$$

where C is a constant that does not depend on m and $m \cdot o(\frac{1}{m}) \to 0$ as $m \to \infty$. Thus, if we use S_m instead of $S(T)$ and use Monte-Carlo method, then the total error is

$$|V - \bar{V}_{m,n}| \leq |V - \hat{V}_m| + |\hat{V}_m - \bar{V}_{m,n}| \leq \frac{C}{m} + o(\frac{1}{m}) + |\hat{V}_m - \bar{V}_{m,n}|,$$

where $\bar{V}_{m,n}$ is computed by generating n different final stock prices S_m. The last term is the error of the Monte-Carlo method and can be estimated easily. So, in order to compute the option price V with a given error ε, we should choose large enough m (so that the term $\frac{C}{m}$ is small enough, for example less than $\frac{\varepsilon}{2}$) and then use MC method with large enough n so that the MC error estimate is also small enough (less than $\frac{\varepsilon}{2}$). There is one trouble: we do not know C. There are several methods for determining approximately it's value:

1. Fix a value of n and choose several values of m: $m_1, m_2, m_3, \ldots, m_k$ (very often one chooses $m_{i+1} = 2\,m_i$). Then if the values of m are large enough (meaning that we can ignore the $o(\frac{1}{m})$ term), we have

   $$V_{m_i,n} \approx V + \frac{C}{m_i} + \varepsilon_i,\ i = 1, 2, \ldots, k,$$

   where ε_i are independent and approximately correspond to the same normal distribution. So we have a linear regression model for determining the values of C and the true option price V. Unfortunately the 95% confidence interval for V is usually too wide for practical purposes, but we can use the largest absolute value of the limits of the 95% confidence interval of C as an estimate $\bar{C}$ for the true value of $|C|$.

2. We use a value of m_1, define $m_2 = 2m_1$ and compute $V_{m_1,n}$ and $V_{m_2,n}$. Their difference satisfies

   $$V_{m_1,n} - V_{m_2,n} \approx \frac{C}{2m} + \varepsilon_1 - \varepsilon_2,$$

where ε_1 and ε_2 are Monte-Carlo errors for computing $\hat{V}_{m_1}$ and $\hat{V}_{m_2}$, respectively. From here we get (how?) that with probability $1-\alpha$ the estimate

$$|C| \leq \bar{C} = 2m_1(|V_{m_1,n} - V_{m_2,n}| + e_1 + e_2),$$

where e_i are the probability $1-\alpha$ MC error estimates of the corresponding computations.

Exercise 4 *Prove the previous error estimate for $|C|$.*

After we have estimated $|C|$, we can choose m large enough so that the term $\frac{C}{m}$ is sufficiently small (for example $\frac{1}{2}$ of the desired accuracy) and then choose n large enough so that the MC error is also sufficiently small.

1.5 Partial differential equation for European options

One way to price options is to derive a partial differential equation (PDE) for the price of the options and then solve the equations either explicitly or numerically.

1.5.1 Derivation of Black-Scholes PDE.

Consider an European option with the payoff $p(S(T))$. Our procedure is as follows:

1. we'll make an assumption about what variables the option price depends on;
2. assume that the option can be replicated by a self-financing investment strategy and derive a PDE for the option price;
3. we'll show that the assumption was justified by using a solution to the PDE for constructing a self-financing portfolio that replicates the option.

It is clear that the option price depends on time (or on how much is left until the expiration date) and on the current stock price. So the first thing to try is to assume that the option price is a function of those two variables, ie the price at time t is $v(S(t), t)$.

Assume that the function v is sufficiently smooth (meaning differentiable) for using Itō's lemma. Assume also that there exists a self-financing investment strategy that replicates the option, then the price of the option at any time should be equal to the value of the portfolio at that time, $v(S(t), t) = X(t)$. Let $\eta(t)$ be the number of shares at time t that determines (with the initial value X(0)) the self-financing strategy.

We know that (see 1.2)

$$dX(t) = (r\,X(t) - (r-D)\,\eta(t)S(t))\;dt + \eta(t)\,dS(t)$$

and according to Itō's formula we have

$$d(v(S(t),t) = \left(\frac{\partial v}{\partial t}(S(t),t) + S(t)^2\frac{\sigma(S(t),t)^2}{2}\frac{\partial^2 v}{\partial s^2}(S(t),t)\right)\,dt + \frac{\partial v}{\partial s}(S(t),t)\,dS(t).$$

As, according to our assumptions we have $v(S(t),t) = X(t)$, the expressions for $dX(t)$ and $d(v(S(t),t)$ should also be equal. Thus, we should have

$$\eta(t) = \frac{\partial v}{\partial s}(S(t),t)$$

and

$$\frac{\partial v}{\partial t}(S(t),t) + \frac{S(t)^2\sigma(S(t),t)}{2}\frac{\partial^2 v}{\partial s^2}(S(t),t) = r\,v(S(t),t) - (r-D)\,S(t)\frac{\partial v}{\partial s}(S(t),t).$$

The last equality is satisfied for all values of t and $S(t)$, if v is a function of two variables satisfying the partial differential equation

$$\frac{\partial v}{\partial t}(s,t) + \frac{s^2\sigma^2(s,t)}{2}\frac{\partial^2 v}{\partial s^2}(s,t) + (r-D)s\frac{\partial v}{\partial s}(s,t) - r\,v(s,t) = 0.$$

Now we have derived a partial differential equations for the option price. It remains to show that we can indeed construct a replicating self-financing investment strategy for European options.

Theorem 9 *Let $p : (0,\infty) \to [0,\infty)$ be a locally integrable function, r the risk-free interest rate, D the rate of continuous dividend payment of the underlying stock and let v be the solution of the partial differential equation*

$$\frac{\partial v}{\partial t} + \frac{s^2\sigma^2(s,t)}{2}\frac{\partial^2 v}{\partial s^2} + (r-D)s\frac{\partial v}{\partial s} - rv = 0,\ 0 \le t < T,\ 0 < s < \infty \tag{1.5}$$

satisfying the final condition

$$v(s,T) = p(s), \quad 0 < s < \infty.$$

Assume that v is twice differentiable in the region $(0,\infty)\times[0,T)$ and is bounded from below. Then the price of the European option with the exercise date T and payoff $p(S(T))$ at any time $0 \le t \le T$ is $v(S(t),t)$ and the option can be replicated with a self-financing investment strategy with the initial value $X(0) = v(S(0),0)$ and the stock holding $\eta(t) = \frac{\partial v}{\partial s}(S(t),t)$.

Proof. Let X be the value of the portfolio corresponding to the self-financing investment strategy with the initial value $X(0) = v(S(0),0)$ and the stock holding of $\eta(t) = \frac{\partial v}{\partial s}(S(t),t)$. Then, according to Itō's Lemma we have

$$\begin{aligned} d(X(t) - v(S(t),t)) &= (rX(t) - r\eta(t)S(t) + D\eta(t)S(t))\,dt \\ &\quad \left(-\frac{\partial v}{\partial t}(S(t),t) - \frac{S^2(t)\sigma^2(S(t),t)}{2}\frac{\partial^2 v}{\partial s^2}(S(t),t)\right)\,dt \\ &= r(X(t) - v(S(t),t))\,dt. \end{aligned}$$

Thus the difference $X(t) - v(S(t), t)$ satisfies an ordinary linear homogeneous differential equation with the zero initial condition and hence $X(t) = v(S(t), t)\ \forall t \in [0, T]$. In particular, we have $X(T) = v(S(T), T) = p(S(T))$, so the investment strategy replicates the option. This proves the lemma. □

The equation (1.5) is called Black-Scholes equation.

1.5.2 An alternative approach to option pricing

There are many market models for which it is not possible to replicate all options by self-financing portfolios, then the previous procedure for deriving a partial differential equation for the option pricing function does not work. A popular alternative is as follows:

1. It is postulated that the option price can be expressed as an expected value, for example
$$V = E[e^{-rT} p(S(T))],$$
where $S(T)$ follows a suitable stochastic differential equation.

2. It is shown that the expected value can be computed as a value of a function that satisfies certain partial differential (or partial integro-differential) equation.

One result, that enables us to relate expected values with solutions of partial differential equations is Feynman-Kac theorem.

Theorem 10 *(Feynman-Kac) Assume that $X(\tau)$ is a process that satisfies*
$$dX(\tau) = \alpha(X(\tau), \tau)\, d\tau + \beta(X(\tau), \tau)\, dB(\tau),\ \ t \le \tau \le T$$
together with the initial condition $X(t) = x$. Let $q(t, x)$ and $p(x)$ be sufficiently well-behaved functions (so that the the expectations below exist). Then
$$E[\exp(-\int_t^T q(X(\tau), \tau)\, d\tau) p(X(T))] = v(x, t),$$
where v is the solution of the partial differential equation
$$\frac{\partial v}{\partial t} + \alpha(x, t)\frac{\partial v}{\partial x} + \frac{\beta(x, t)^2}{2}\frac{\partial^2 v}{\partial x^2} - q(x, t)v = 0$$
satisfying the final condition $v(x, T) = p(x)$.

Proof. Exercise for those who have taken the Martingales course. (Hint: show that if v satisfies the equation, then $\exp(-\int_t^s q(X(\tau), \tau)\, d\tau) v(X(s), s)$ is a martingale).□

The previous theorem has generalizations to multidimensional processes X and for different type of stochastic differential equations for X.

Using this result it is easy to show that if an option price is computed according to the assumption above in the case $dS(t) = S(t)((r - D)\, dt + \sigma(S(t), t)\, dB(t))$, then the option pricing function satisfies the Black-Scholes equation.

1.5.3 Classification and properties of partial differential equations

Definitsioon 11 *A partial differential equation with respect to an unknown function u is* linear*, if all it's terms are products of some function (or constant) not depending on u, and u or some partial derivative of u.*

Black-Scholes equation is a linear PDE.

In the case of linear equations a linear combination of any number of solutions is also a solution.

Definitsioon 12 *The* order *of a PDE is the highest order of derivative of the unknown function appearing in PDE.*

The order of Black-Scholes equation is 2.

A non-complete classification of second order equations of two variables is as follows:

1. If for each independent variable there is a second order term that contains a derivatives with respect to that variable and if the highest order terms

$$a(x,t)\frac{\partial^2 u}{\partial t^2} + b(x,t)\frac{\partial^2 u}{\partial x \partial t} + c(x,t)\frac{\partial^2 u}{\partial x^2}$$

   are such that at a point (x,t) we have

$$b^2(x,t) - 4a(x,t)c(x,t) > 0,$$

   then the PDE is *hyperbolic* at that point. An example is the wave equation

$$\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0.$$

   An example of a solution of the wave equation is $u(x,t) = \sin(x-t)$ or, more generally, $u(x,t) = f(x-t)$, where f is an arbitrary twice differentiable function.

2. If for each independent variable there is a second order term that contains a derivatives with respect to that variable and if the highest order terms

$$a(x,t)\frac{\partial^2 u}{\partial t^2} + b(x,t)\frac{\partial^2 u}{\partial x \partial t} + c(x,t)\frac{\partial^2 u}{\partial x^2}$$

   are such that at a point (x,t) we have

$$b^2(x,t) - 4a(x,t)c(x,t) < 0,$$

   then the PDE is *elliptic* at that point. An example is the Laplace equation

$$\frac{\partial^2 u}{\partial t^2} + \frac{\partial^2 u}{\partial x^2} = 0.$$

3. If the equation is of the form

$$a(x,t)\frac{\partial u}{\partial t} + b(x,t)\frac{\partial^2 u}{\partial x^2} + \text{lower order terms},$$

then at the points where $a(x,t) \neq 0,\ b(x,t) \neq 0$ the equation is a parabolic equation at that point. An example is the heat equation

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0.$$

An example of a solution of the heat equation is $u(x,t) = e^{-t}\sin(x)$.

If an equation is of the same type at every point, then we say that the equation is hyperbolic, elliptic or parabolic. Black-Scholes equation is a parabolic equation.

Some properties of parabolic equations

1. Parabolic equations are well-posed in one direction of time only: if the value of solution at $t = t_0$ is given, then the value of corresponding solution can be found only for $t > t_0$ (in the case when the coefficients of $\frac{\partial u}{\partial t}$ and $\frac{\partial^2 u}{\partial x^2}$ have the same sign) or for $t < t_0$ (in the case when the coefficients of $\frac{\partial u}{\partial t}$ and $\frac{\partial^2 u}{\partial x^2}$ have different signs).

2. Parabolic equations are smoothing equation, the smoothness (differentiability) of solution does not depend on the smoothness of the given initial (or final) condition but on the smoothness of the coefficients only.

3. Parabolic equations have infinite propagation speed: if the value of the given initial (or final) condition is changed in a neighborhood of one point only, then the change has some effect of the solution at all other times at every point.

1.5.4 Transformation of Black-Scholes equation to the heat equation

Using the change of variables $v(s,t) = u(x,t)$, where $x = \ln s$, we can transform the equation (1.5) to the form

$$\frac{\partial u}{\partial t}(x,t) + \alpha(x,t)\frac{\partial^2 u}{\partial x^2}(x,t) + \beta(x,t)\frac{\partial u}{\partial x}(x,t) - r\,u(x,t) = 0, \tag{1.6}$$

where

$$\alpha(x,t) = \frac{\sigma^2(e^x,t)}{2},\ \beta(x,t) = r - D - \frac{\sigma^2(e^x,t)}{2}.$$

The corresponding final condition for the function u is

$$u(x,T) = p(e^x),\ -\infty < x < \infty. \tag{1.7}$$

The equation (1.6) is a backward parabolic partial differential equation. It turns out that if σ is a constant, then we can further transform the equation to the standard heat equation. First, note that by defining $u(x,t) = e^{-r\tau}\tilde{u}(x,\tau)$, where $\tau = T - t$ we get a usual parabolic PDE which does not have a term without derivatives:

$$\frac{\partial \tilde{u}}{\partial \tau}(x,\tau) = \alpha \frac{\partial^2 \tilde{u}}{\partial x^2}(x,\tau) + \beta \frac{\partial \tilde{u}}{\partial x}(x,\tau).$$

Now the change of variables

$$\tilde{u}(x,\tau) = w(y,\eta),\ \eta = \alpha\tau,\ y = x + \beta\tau$$

gives us the equation

$$\frac{\partial w}{\partial \eta}(y,\eta) = \frac{\partial^2 w}{\partial y^2}(y,\eta).$$

This is the heat equation. It is known that the solution of the heat equation has a representation

$$w(y,\eta) = \frac{1}{2\sqrt{\pi\eta}} \int_{-\infty}^{\infty} e^{-\frac{(y-\xi)^2}{4\eta}} w(\xi,0)\, d\xi.$$

Taking into account that

$$w(y,0) = u(y,T) = v(e^y,T),$$

$$v(s,t) = u(\ln s, t) = e^{-r(T-t)}\tilde{u}(\ln s, T-t) = e^{-r(T-t)} w(\ln s + (r - D - \frac{\sigma^2}{2})(T-t), \frac{\sigma^2}{2}(T-t))$$

we can now express the solution of the original black-scholes equation in an integral form:

$$v(s,t) = \frac{e^{-r(T-t)}}{\sqrt{2\pi(T-t)}\sigma} \int_{-\infty}^{\infty} e^{-\frac{(\ln s + (r - D - \frac{\sigma^2}{2})(T-t) - \xi)^2}{2\sigma^2(T-t)}} p(e^{\xi})\, d\xi \tag{1.8}$$

Using this form it is possible to derive explicit formulas for several options.

Exercise 5 *The transformations used above are not the only possible ones. Assume that the volatility σ is constant. Find a, b such that the function w defined by $u(x,t) = e^{ax+b\tau}w(\tau,x)$, where $\tau = T - t$, satisfies the partial differential equation*

$$\frac{\partial w}{\partial \tau}(\tau,x) = \frac{\sigma^2}{2}\frac{\partial^2 w}{\partial x^2}(\tau,x).$$

1.5.5 Special solutions of Black-Scholes equation

It turns out that for both constant and non-constant volatility case the functions of the form

$$v(s,t) = c_1 e^{-r\,(T-t)} + c_2 e^{-D\,(T-t)} s,\ c_1, c_2 \in \mathbf{R} \tag{1.9}$$

are solutions of the equation 1.5. One consequence of this is so called Put-Call parity.

Lemma 13 *(Put-Call parity) Let $P(S,t,T)$ and $C(S,t,T)$ denote the values of the European put and call options with the exercise price E and expiration time T at time t if the stock price is $S(t) = S$. Then*

$$C(S,t,T) = P(S,t,T) + e^{-D\,(T-t)}S - E\,e^{-r\,(T-t)}.$$

Exercise 6 *Prove the previous lemma by using the uniqueness of the solution of the final value problem of BS equation.*

Remark. *If $D = 0$, then Put-Call parity relation follows directly from an arbitrage argument even without the assumption of no transaction costs.*

The special solutions are important for constructing effective numerical methods.

1.6 Finite difference methods for Black-Scholes equation

A popular class of numerical methods for solving partial differential equations is finite difference methods, where approximate values of solutions at certain rectangular mesh points are found by replacing partial derivatives in the PDE by finite difference approximations (using only the values at the mesh points) and solving the resulting system of equations.

1.6.1 The idea of finite difference methods

Let u be the solution of the problem (1.6), (1.7). Since in a numerical computation we can find only finitely many numbers, we may try to compute a table of the approximate values of u. For this we fix the minimal and maximal values of x we are interested in (say x_{min} and x_{max}), the number m of subintervals of the time period $[0,T]$ (this is the number of time steps we use to get from 0 to T) and the number of subintervals n we use in the x direction.

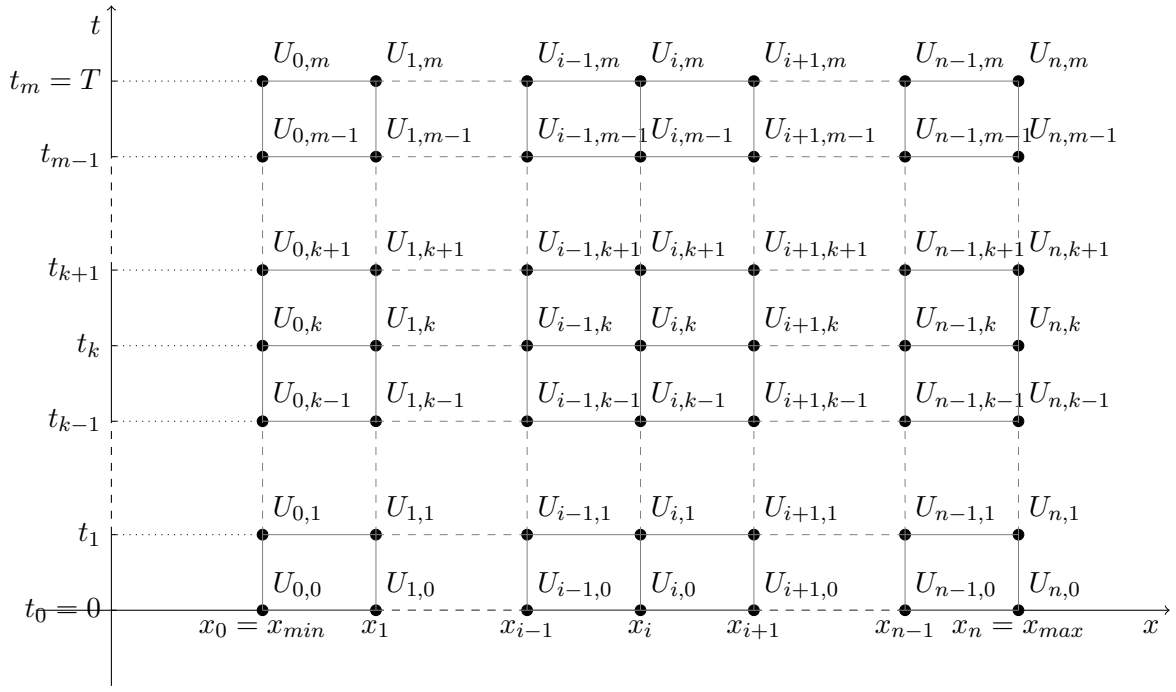
Denote

$$\Delta t = \frac{T}{m}, \quad \Delta x = \frac{x_{max} - x_{min}}{n}$$

and define

$$t_k = k\,\Delta t,\ k = 0,\dots,m;\ x_i = x_{min} + i\,\Delta x,\ i = 0,\dots,n.$$

Our aim is to find approximately the values $u_{ik} = u(x_i, t_k)$, ie we want to form a $(m+1)\times(n+1)$ table of approximate values. Let U_{ik} be the approximate values we write in the table. The notations are illustrated below.



The values of u at $t = T$ are given by the final condition (1.7). Therefore we have

$$U_{im} = p(e^{x_i}), \quad i = 0, 1, \ldots, n. \tag{1.10}$$

The values corresponding to $x = x_{min}$ and $x = x_{max}$ will be given by some boundary conditions discussed later.

In order to find the other values, we have to make use of the equation (1.6), where derivatives are replaced by numerical differentiation formulas.

From the textbooks of numerical methods we can find the following approximate differentiation rules for a sufficiently smooth (meaning enough times continuously differentiable) function f:

$$f'(z) = \frac{f(z+h) - f(z)}{h} + O(h), \tag{1.11}$$

$$f'(z) = \frac{f(z) - f(z-h)}{h} + O(h), \tag{1.12}$$

$$f'(z) = \frac{f(z+h) - f(z-h)}{2h} + O(h^2), \tag{1.13}$$

$$f''(z) = \frac{f(z-h) - 2f(z) + f(z+h)}{h^2} + O(h^2), \tag{1.14}$$

where $O(h^q)$ denotes some function (which may be different in different formulas) that may depend on f, z satisfying the inequality $|O(h^q)| \leq const.h^q$ for all sufficiently small values of h. The first formula is called forward difference approximation, the

second is backward difference approximation and the third is the central difference approximation of the derivative. The name of finite difference methods comes from replacing the derivatives (that are the limit of those formulas when h goes to 0) with approximations corresponding to some finite (small) values of h.

The same formulas can be used for approximating partial derivatives. For example, if we want to approximate $\frac{\partial u}{\partial x}(x,t)$, we consider the variable t fixed and use x in the role of z and Δx in the role of h in the above formulas, so applying for example (1.13) gives us

$$\frac{\partial u}{\partial x}(x,t) = \frac{u(x+\Delta x,t) - u(x-\Delta x,t)}{2\Delta x} + O(\Delta x^2).$$

We start by deriving an explicit finite difference method (meaning that the solution of the system of equations can be written out in an explicit form) for solving Black-Scholes PDE.

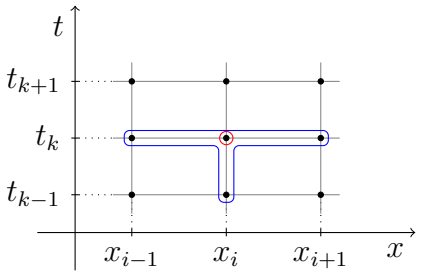
1.6.2 Explicit finite difference method

After taking into account the final condition we still have $(n+1)\cdot m$ empty spaces in our table of approximate values. We get the values corresponding to $x = x_{min}$ and $x = x_{max}$) (or some additional equations corresponding to the values) from the boundary conditions. This means we have to use the PDE to derive $(n-1)\cdot m$ additional equations for the unknown values. The procedure for deriving those equations is the same for all finite difference methods. Namely we write down the equation (1.6) at $(n-1)\cdot m$ points and then approximate the derivatives at the chosen points by finite difference formulas using the values of the unknown function u only at the points that correspond to our table. We get different methods by choosing different points for writing out equations and by using different finite difference approximations for the derivatives.

In order to get an explicit method for backward parabolic equation we start by writing out the equation (1.6) at the points $(x_i,t_k),\ i=1,\dots,(n-1),\ k=1,\dots,m$:

$$\frac{\partial u}{\partial t}(x_i,t_k) + \alpha(x_i,t_k)\frac{\partial^2 u}{\partial x^2}(x_i,t_k) + \beta(x_i,t_k)\frac{\partial u}{\partial x}(x_i,t_k) - r\,u(x_i,t_k) = 0.$$

To approximate the partial derivatives of u in the previous equation we use its values at the following grid points (surrounded by a green curve, the red circle denotes the point where we wrote down the equation).



Using the approximations (1.12), (1.13) and (1.14) for the time derivative, for the first derivative with respect to x and for the second derivative with respect to x, respectively, we get

$$\begin{aligned}
\frac{\partial u}{\partial t}(x_i,t_k) &= \frac{u(x_i,t_k)-u(x_i,t_{k-1})}{\Delta t}+O(\Delta t)=\frac{u_{ik}-u_{i,k-1}}{\Delta t}+O(\Delta t),\\
\frac{\partial u}{\partial x}(x_i,t_k) &= \frac{u_{i+1,k}-u_{i-1,k}}{2\Delta x}+O(\Delta x^2),\\
\frac{\partial^2 u}{\partial x^2}(x_i,t_k) &= \frac{u_{i-1,k}-2u_{ik}+u_{i+1,k}}{\Delta x^2}+O(\Delta x^2).
\end{aligned}$$

Thus the values u_{ki} of the exact solution u satisfy the relations

$$\frac{u_{i,k}-u_{i,k-1}}{\Delta t}+\alpha_{ik}\frac{u_{i-1,k}-2u_{ik}+u_{i+1,k}}{\Delta x^2}+\beta_{ik}\frac{u_{i+1,k}-u_{i-1,k}}{2\Delta x}-ru_{ik}+O(\Delta t+\Delta x^2)=0, \tag{1.15}$$

where

$$\alpha_{ik}=\alpha(x_i,t_k),\ \beta_{ik}=\beta(x_i,t_k).$$

The idea of the finite difference methods is that throwing away the small error term $O(\Delta t+\Delta x^2)$ in (1.15) should cause only small errors in the results. Therefore we find the approximate values U_{ik} from the equations

$$\frac{U_{i,k}-U_{i,k-1}}{\Delta t}+\alpha_{ik}\frac{U_{i-1,k}-2U_{ik}+U_{i+1,k}}{\Delta x^2}+\beta_{ik}\frac{U_{i+1,k}-U_{i-1,k}}{2\Delta x}-r\,U_{ik}=0. \tag{1.16}$$

The algorithm of the explicit finite difference method.

Solving the equations (1.16) for $U_{i,k-1}$ we get

$$U_{i,k-1}=a_{ik}U_{i-1,k}+b_{ik}U_{ik}+c_{ik}U_{i+1,k},\ i=1,2,\dots,n-1,\ k=1,\dots,m, \tag{1.17}$$

where

$$\begin{aligned}
a_{ik} &= \frac{\Delta t}{\Delta x^2}\left(\alpha_{ik}-\frac{\beta_{ik}}{2}\Delta x\right),\\
b_{ik} &= 1-2\frac{\Delta t}{\Delta x^2}\alpha_{ik}-r\Delta t,\\
c_{ik} &= \frac{\Delta t}{\Delta x^2}\left(\alpha_{ik}+\frac{\beta_{ik}}{2}\Delta x\right).
\end{aligned}$$

The equations (1.17) are in a very convenient form: if we know the values corresponding to k-th column of the matrix U, then using those equations we can simply compute the values $U_{i,k-1},\ i=1,\dots,n-1$. In order to be able to compute all values of the table we should additionally specify how the values of the zeroth and n-th rows should be computed. One way to do this is to specify some functions $\phi_1(t)$ and $\phi_2(t)$ and define $U_{0k}=\phi_1(t_k),\ U_{nk}=\phi_2(t_k),\ k=0,1,\dots,m-1$.

Unfortunately we do not know what are the right functions ϕ_1 and ϕ_2 (ideally, they should be the values of the unknowns solution u at those boundaries) but we should try to specify some functions that are not too far from the right values. If the choice of the functions is not very good, then our approximate solution may have relatively large errors close to $x = x_{min}$ and $x = x_{max}$. The simplest reasonable choice is to require that the values of the approximate solution U remain constant at the boundaries, ie

$$U_{0k} = p(e^{x_{min}}),\ U_{nk} = p(e^{x_{max}}),\ k = 0, 1, \ldots, m-1. \tag{1.18}$$

Of cause this choice introduces some errors close to the boundary, therefore we should choose x_{min} and x_{max} so that the region of values of x we are interested in is quite far from both of those values (but between the values). We'll come back to the question of choosing good boundary conditions later.

To summarize, we have derived the following *explicit finite difference method* for solving the equation (1.6):

1. Fill according to (1.10) the m-th column of the table.
2. For each time step $k = m, m-1, \ldots, 1$
   (a) Fill according to (1.17) the $(k-1)$-th column, except the 0-th and the n-th value.
   (b) Compute according to (1.18) the 0-th and the n-th value of the column.

Stability of the explicit method

It turns out that the error of the approximate solution obtained by the explicit finite difference method does not always go to zero when m and n tend to infinity. Namely, if a certain relation between m and n values does not hold the difference between the exact values and the approximate solutions may grow by a factor that is bigger than one at each timestep, resulting in huge errors at the final time. If this happens, the method is called *instable*.

In order to understand better the phenomenon of instability, let us consider a situation where we have two different sets of the values $U_{ik},\ i = 0, \ldots, n$ and $\tilde{U}_{ik},\ i = 0, \ldots, n$ of the approximate solution at the k-th column. Then the values of the $(k-1)$-th column, computed according to the explicit finite difference method, satisfy the equations

$$U_{i,k-1} - \tilde{U}_{i,k-1} = a_{ik}(U_{i-1,k} - \tilde{U}_{i-1,k}) + b_{ik}(U_{ik} - \tilde{U}_{ik}) + c_{ik}(U_{i+1,k} - \tilde{U}_{i+1,k}),\ i = 1, 2, \ldots, n-1.$$

Let ε be the maximal difference of the values of U and $\tilde{U}$ at the k-th column, then

$$|U_{i,k-1} - \tilde{U}_{i,k-1}| \le (|a_{ik}| + |b_{ik}| + |c_{ik}|)\varepsilon,\ i = 1, 2, \ldots, n-1.$$

If all coefficients a_{ik}, b_{ik} and c_{ik} are non-negative then, taking into account the equality

$$a_{ik} + b_{ik} + c_{ik} = 1 - r\,\Delta t,$$

we get that the maximal error in the $(k+1)$-th row is bounded by $(1 - r\,\Delta t)\varepsilon$. That means that in this case the errors are not increasing and the mathod is *stable*. But if any of the coefficients is negative, then the sum of the absolute values of the coefficients may be larger than 1 and the errors in one timestep may be multiplied by a factor that is larger than one in each of the subsequent timesteps, resulting in huge errors at the final time. There are always some errors in numerical computations (the real numbers are not computed exactly, the approximate solution is not exactly equal to the theoretical one). Therefore, when implementing the method, it is important to choose m and n so that the coefficients are all positive since otherwise the answers may be totally inaccurate.

1.7 Basic implicit finite difference method. Crank-Nicolson method

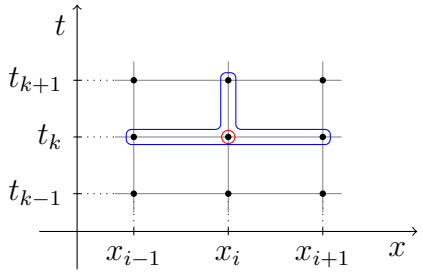
Explicit finite difference method is very convenient for implementation but it turned out to have a uncomfortable feature of being unstable if one does not choose the values of the discretization parameters m and n carefully. Next we consider some methods that are always stable but require the solution of a system of equations at each timestep.

1.7.1 Derivation of the basic implicit method

When deriving the basic implicit finite difference method we use the equation (1.6) at the points (x_i, t_k), $k = 0, 1, \ldots, m-1$, $i = 1, \ldots, n-1$ (in comparison with the explicit method, we use the points with $t = 0$ instad of $t = T$) and the forward difference approximation for the time derivative:

$$\frac{\partial u}{\partial t}(x_i, t_k) = \frac{u(x_i, t_{k+1}) - u(x_i, t_k)}{\Delta t} + O(\Delta t).$$

The derivatives with respect to x are approximated as before. This means that we use the following points (surrounded by the blue curve) for approsimating the partial derivatives of the equation at the point (x_i, t_k).



After substituting in the the approximations for the derivatives and throwing away the error terms we get

$$\frac{U_{i,k+1}-U_{i,k}}{\Delta t}+\alpha_{ik}\frac{U_{i-1,k}-2U_{ik}+U_{i+1,k}}{\Delta x^2}+\beta_{ik}\frac{U_{i+1,k}-U_{i-1,k}}{2\Delta x}-r\,U_{ik}=0,\ k=1,\ldots,m,\ i=1,\ldots,n-1. \tag{1.19}$$

After simplifications we get the following system of equations for finding the values U_{ik}:

$$a_{ik}U_{i-1,k}+b_{ik}U_{ik}+c_{ik}U_{i+1,k}=U_{i,k+1}, k=0,1,\ldots,m-1,\ i=1,\ldots,n-1, \tag{1.20}$$

where

$$\begin{aligned} a_{ik} &= -\frac{\Delta t}{\Delta x^2}\left(\alpha_{ik}-\frac{\beta_{ik}}{2}\Delta x\right), \\ b_{ik} &= 1+2\frac{\Delta t}{\Delta x^2}\alpha_{ik}+r\Delta\tau, \\ c_{ik} &= -\frac{\Delta t}{\Delta x^2}\left(\alpha_{ik}+\frac{\beta_{ik}}{2}\Delta x\right). \end{aligned}$$

In order to find the values U_{ik}, we have to fix suitable boundary conditions at $x=x_{min}$, $x=x_{max}$ and solve step-by-step the systems of equations for $U_{i,m-1},\ i=0\ldots,n$, $U_{i,m-2},\ i=0\ldots,n,\ \ldots,\ U_{i0},\ i=0\ldots,n$. The errors of the finite difference approximation is $O(\Delta t+\Delta x^2)$ (there is an additional error coming from specifying the boundary conditions).

1.7.2 The stability of the basic implicit method

We show that the basic implicit method is stable under quite general assumptions about the coefficients.

Lemma 14 *If $b_{ik}\geq 0,\ a_{ik}\leq 0$ and $c_{ik}\leq 0$, $k=0,\ldots,m-1$, $i=1,\ldots,(n-1)$, then the basic implicit method is stable.*

Proof. Suppose U_{ik} and $\tilde{U}_{ik}$ both satisfy the same boundary conditions and the equation (1.20)for some $k\in\{0,1,\ldots,m-1\}$ and that $|U_{i,k+1}-\tilde{U}_{i,k+1}|\leq\varepsilon\ \forall i$. Denote

$$E_i=U_{ik}-\tilde{U}_{ik},\ i=0,\ldots,n.$$

Denote by M the maximal value of $|E_i|,\ i=0,\ldots,n$. We want to show that $M\leq\varepsilon$; this shows the stability of the system. Since both U_{ik} and $\tilde{U}_{ik}$ satisfy (1.20), their difference also satisfies the system. We write the equation for the difference in the form

$$b_{ik}E_i=U_{i,k+1}-\tilde{U}_{i,k+1}-a_{ik}E_{i-1}-c_{ik}E_{i+1}.$$

By taking absolute values of both sides and using properties of the absolute value, we get

$$b_{ik}|E_i|\leq\varepsilon-a_{ik}|E_{i-1}|-c_{ik}|E_{i+1}|.$$

Here we used all of the assumptions of the lemma. We can make the right hand side larger, by replacing the absolute values of E_{i-1} and $|E_{i+1}|$ with the maximal value M:

$$b_{ik}|E_i| \leq \varepsilon - a_{ik}M - c_{ik}M.$$

The last inequality holds for all $i = 1, \ldots, n-1$. Choose the value of $i \in \{1, \ldots, n-1\}$ such that $|E_i| = M$. In the case of that i we have

$$b_{ik}M \leq \varepsilon - a_{ik}M - c_{ik}M,$$

hence

$$(a_{ik} + b_{ik} + c_{ik})M \leq \varepsilon.$$

But $a_{ik} + b_{ik} + c_{ik} = 1 + r\Delta t$, hence we have shown that

$$M \leq \frac{\varepsilon}{1 + r\Delta t} < \varepsilon.$$

This proves the lemma. □ From the formulas of the coefficients it is easy to see, that the validity of the stability conditions does not depend on m and that under quite general assumptions about α and β the conditions hold for sufficiently large values of n.

1.7.3 Derivation of the Crank-Nicolson method

One problem with the numerical methods considered so far is that their accuracy with respect to time (first order accuracy, $O(\Delta t)$) is lower than with respect to the x variable (second order accuracy, $O(\Delta x^2)$). This means that if we want to reduce the error four times, we have to increase the value of m four times and the value of n two times, resulting in 8 times longer computation time. It would be much nice to have second order accuracy with respect to the t variable, too.

The low accuracy of the explicit and basic implicit methods with respect to time comes from the fact that both forward and backward difference (used for approximating the derivative with respect to t) have the first order accuracy at the points (x_i, t_k) where we wrote down our partial differential equation. But, taking into account that the central difference approximates a derivative with the second order accuracy, the finite difference approximation $\frac{\partial u}{\partial t} \approx \frac{u_{k+1,i} - u_{k,i}}{\Delta t}$ is of the order $O(\Delta t^2)$ at the point $(x_i, t_k + \frac{1}{2}\Delta t)$. This gives the idea to try to get a better approximation of the partial differential equation by writing the equation out at those points before approximating the derivatives.

Denote $t_{k+\frac{1}{2}} = t_k + \frac{\Delta t}{2}$. Let us use the following steps for deriving a finite difference method for our equation:

1. Write the equation (1.6) out at the points

$$(x_i, t_{k+\frac{1}{2}}),\ k = 0, 1, \ldots, m-1,\ i = 1, \ldots, n-1)$$

and use the approximation

$$\frac{\partial u}{\partial t}(x_i, t_{k+\frac{1}{2}}) = \frac{u_{k+1,i} - u_{k,i}}{\Delta t} + O(\Delta t^2)$$

for the time derivative.

2. Approximate u and it's partial derivatives with respect to x at $(x_i, t_{k+\frac{1}{2}})$ with the average values of those quantities at the points (x_i, t_{k+1}) and (x_i, t_k), ie use the approximations

$$\begin{aligned}
u(x_i, t_{k+\frac{1}{2}}) &= \frac{1}{2}(u(x_i, t_{k+1}) + u(x_i, t_k)) + O(\Delta t^2), \\
\frac{\partial u}{\partial x}(x_i, t_{k+\frac{1}{2}}) &= \frac{1}{2}(\frac{\partial u}{\partial x}(x_i, t_{k+1}) + \frac{\partial u}{\partial x}(x_i, t_k)) + O(\Delta t^2), \\
\frac{\partial^2 u}{\partial x^2}(x_i, t_{k+\frac{1}{2}}) &= \frac{1}{2}(\frac{\partial^2 u}{\partial x^2}(x_i, t_{k+1}) + \frac{\partial^2 u}{\partial x^2}(x_i, t_k)) + O(\Delta t^2)
\end{aligned}$$

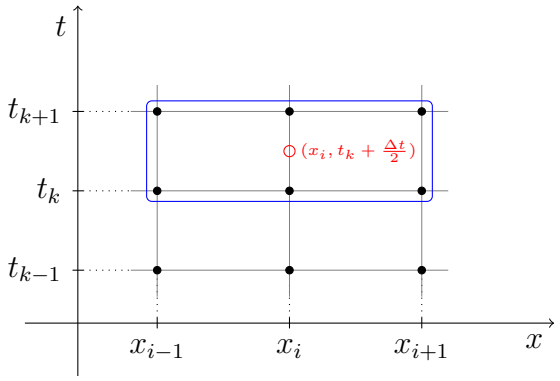
and after that, replace the derivatives with the usual finite difference approximations.

3. Throw away the error terms and reorganize the equations so that the terms corresponding to $t = t_k$ are to the left of the equality and the terms corresponding to $t = t_{k+1}$ are on the right-hand-side of the equation.

After carrying through those steps we get a finite difference method of the form.

$$a_{ik}U_{i-1,k} + b_{ik}U_{ik} + c_{ik}U_{i+1,k} = d_{ik}U_{i-1,k+1} + e_{ik}U_{i,k+1} + f_{ik}U_{i+1,k+1}.$$

Schematically each equation involves the following grid points (red circle denotes the point where equation is written down, blue curve is around the grid points used in the corresponding discretized equation).



The error of the method (called Crank-Nicolson method) is of the order $O(\Delta t^2 + \Delta x^2)$

Homework exercise 2 *(Deadline November 8, 2011) Find the formulas for the coefficients $a_{ik}, b_{ik}, \ldots, f_{ik}$ of the Crank-Nicolson method. Present full derivation (starting from formulating the differential equation and definition of gridpoints).*

This method is used similarly to the basic implicit method: the values of U_{ik} are found step-by-step starting from $k = m - 1$, solving for each k a three-diagonal system of equations.

Extra reading: solving three-diagonal systems of equations

Although we use in computer labs predefined functions for solving systems of equations, it is quite easy to implement solvers for three-diagonal systems of equations arising in finite difference equations for Black-Scholes equation.

Consider a system of equations

$$\begin{aligned} B_1 y_1 + C_1 y_2 &= F_1, \\ A_2 y_1 + B_2 y_2 + C_2 y_3 &= F_2, \\ A_3 y_2 + B_3 y_3 + C_3 y_4 &= F_3, \\ &\cdots\cdots \\ A_{n-1} y_{n-2} + B_{n-1} y_{n-1} &= F_{n-1}, \end{aligned}$$

where A_i, B_i, C_i and F_i are some numbers and $y_1, \ldots, y_{n-1}$ are unknowns. The process of solution of such systems is quite simple: first we eliminate from all equations (starting from the second one) the terms with coefficients A_i. This gives us a system where in each equation except the last one are two unknowns; the last one has only one unknown. After that we can compute the values of y_i starting from the last one (from the last equation), then the one before the last and so on, finishing with the value of y_1. More precisely, the equations after the elimination of the A_i terms are of the form $\bar{B}_i y_i + C_i y_{i+1} = \bar{F}_i$, $i = 1, \ldots, n-2$, $\bar{B}_{n-1} y_{n-1} = \bar{F}_{n-1}$, where

$$\begin{aligned} \bar{B}_1 &= B_1, \ \bar{F}_1 = F_1, \\ \bar{B}_i &= B_i - \frac{A_i}{\bar{B}_{i-1}} C_{i-1}, \ \bar{F}_i = F_i - \frac{A_i}{\bar{B}_{i-1}} \bar{F}_{i-1}, \ i = 2, \ldots, n-1. \end{aligned}$$

The computation of the values of the solution of the system goes then as follows:

$$\begin{aligned} y_{n-1} &= \frac{\bar{F}_{n-1}}{\bar{B}_{n-1}}, \\ y_i &= \frac{\bar{F}_i - C_i y_{i+1}}{\bar{B}_i}, \ i = n-2, \ldots, 1 \end{aligned}$$

In implementation it is useful to notice that if we want to solve the same system of equations with many different right-hand-side values then we need only the values of $\bar{B}_i$ that have to be computed only once.

1.7.4 Solving untransformed Black-Scholes equation

There are several problems with using the logarithmic transformation $x = \ln s$ before solving the Black-Scholes equation numerically. First, we get the values of the solution for stock prices that are unevenly spaced (at places $S_i = e^{x_i}$) and this is not very good

if we want to form a table of option prices corresponding to evenly spaced intervals in the stock price; computing approximations for the derivatives is also more difficult. Second, this transformation makes our solution region doubly infinite while before the transformation we had a boundary at $S = 0$. This means that we have to introduce two artificial boundaries while for untransformed equation it would be enough to specify only one artificial boundary $s = S_{max}$. Therefore it makes sense to try to solve the Black-Scholes equation without the logarithmic change of variables.

Recall that the option price satisfies the Black-Scholes partial differential equation

$$\frac{\partial v}{\partial t} + \alpha(s,t)\frac{\partial^2 v}{\partial s^2} + \beta(s)\frac{\partial v}{\partial s} - r\,u,\ 0 < t \leq T,\ s > 0 \tag{1.21}$$

together with the final condition

$$v(s,T) = p(s),\ s > 0.$$

Here

$$\alpha(s,t) = \frac{s^2\sigma^2(s,t)}{2},$$
$$\beta(s) = (r - D)s.$$

Notice that the equation (1.21) is of the same form as (1.6), only instead of x we have s and the final condition and the values of the coefficients are computed differently. This means that if we consider finite difference methods for finding the values of the function v at the points (s_i, t_k), where $s_i = i \cdot \frac{S_{max}}{n}),\ t_k = k \cdot \frac{T}{m},$, we can use the formulas for the coefficients we derived for (1.6) by changing x_i with s_i and Δx with $\Delta s = \frac{S_{max}}{n}$. Thus, we can use any of the methods derived so far without any additional effort for finding formulas for the coefficients.

Notice that if we take $s = 0$ in the equation (1.21) then we get an ordinary differential equation $\frac{\partial v}{\partial t}(0,t) = r\,v(0,t)$ which has the solution

$$v(0,t) = p(0)e^{-r(T-t)}.$$

Therefore we have an exact boundary condition at $s = 0$ so we have to specify only a condition for the artificial boundary $s = S_{max}$. The usual boundary conditions are:

- $v(S_{max}, t) = p(S_{max}),\ 0 \leq t < T$. This condition can always be used but is relatively crude (meaning it introduces relatively large errors close to the boundary).
- If the payoff function p is linear from some point to infinity, $p(s) = k_1 + k_2 s,\ s \geq S_{max}$, then the boundary condition

  $$v(S_{max}, t) = k_1 e^{-r(T-t)} + k_2 e^{-D(T-t)} S_{max}$$

  corresponding to the special solution with the same final values gives usually much better results.

1.7.5 Computing the option prices with a given accuracy

In practical situations when one wants to compute option prices numerically, it is not enough just to get a value of the approximate solution. It is very important to know how to compute the value with a given accuracy. There are three sources of errors in using finite difference methods for computing option prices:

- the placement of artificial boundaries;
- the form of artificial boundary conditions used;
- the discretization error controlled by the parameters m and n.

There are known some theoretical estimates for the error caused by the artificial boundary conditions that allow one to choose the placement of the artificial boundary (for a given boundary condition) so that this component of error is less that a given number before starting numerical computations. Then one has to estimate only the discretization error when computing the option prices. Unfortunately those estimates are quite complicated, therefore we adopt a simple (although more time-consuming) approach in this course.

Suppose we want to find a table of the prices of an European option so that the maximal error for the stock prices in the interval $s \in [s_1, s_2]$ was less that ε. When using a finite difference method for the untransformed equation, we have to fix only one artificial boundary; a good starting point is to take $s_{max} = 2 \cdot s_2$ (if σ is large or the time period is long, it may make sense to take larger value for s_{max}). Our procedure is as follows:

1. Solve the problem with a finite difference method and estimate the error by Runge's method, until the (estimated) finite difference discretization error is less than $\frac{\varepsilon}{2}$.
2. Increase the value of s_{max} two times (multiply it by 2) and solve the problem with finite difference method with the same Δt and Δs as before. If the solution changes (in the region of interest) by more than $\frac{\varepsilon}{4}$ then go back to step one. Otherwise we have obtained the solution with the desired accuracy.

In order to follow the instructions, one has to know how to estimate the discretization error by Runge's method.

Runge's error estimate

Usually a numerical procedure gives us the result with some error that we do not know:

$$result_1 = exact + error_1.$$

Very often we can rerun the numerical procedure with some other input parameters so that the error is (approximately) reduced by a certain factor $q > 1$: we get

$$result_2 = exact + error_2 \approx exact + \frac{error_1}{q}.$$

Then, by subtracting the second equation from the first one and reorganizing the terms, we get an estimate for the error of the second computation:

$$error_2 \approx \frac{result_1 - result_2}{q - 1}.$$

This is called Runge's error estimate.

In the case of finite difference methods we have considered so far, the (formal) error estimate is $O(\Delta t + \Delta s^2)$ (for the basic implicit method) or $O(\Delta t^2 + \Delta s^2)$ in the case of Crank-Nicolson method. Assuming that the actual error behaves according to the estimate (the leading, meaning the most slowly decreasing term in the error expansion is shown in the estimate), the error is reduced four times, if we reduce Δs two times and $\Delta\tau$ either four times (in the case of the basic implicit method) or two times (for Crank-Nicolson method). So, computing the numerical results first with $n = n_0,\ m = m_0$ (giving us $result_1$) and then with $n = 2n_0,\ m = 4m_0$ in the case of the basic implicit method or $m = 2m_0$ for Crank-Nicolson (giving us $result_2$), we can use the Runge's estimate with $q = 4$ to estimate the error at the common points of the two computation. This means that if the computed option prices corresponding to $result_1$ are $U_{ik},\ k = 0, \ldots, m_0,\ i = 0, \ldots, n_0$ and the option prices corresponding to $result_2$ are $W_{ik},\ k = 0, \ldots, m_1,\ i = 0, \ldots, 2n_0$ (where $m_1 = 4m_0$ for basic implicit and $m_1 = 2m_0$ for Crank-Nicolson method), we estimate the discretization error as the maximum of the quantities

$$\frac{|U_{ik} - W_{\frac{m1}{m_0}k,2i}|}{3}.$$

If we are interested only in option values corresponding to $t = 0$, then we can estimate the discretization error by those quantities corresponding to $k = 0$.

Remark. Actually, the formal error estimates we have been using are correct only if the payoff function is sufficiently many times (at least two times) continuously differentiable. In the case of usual financial payoff functions (which have discontinuous derivatives) the error is not reduced by four but by a number between 2 and 4. Therefore, if we want to be more confident that the actual error is smaller than the estimated error, we should divide the difference of U and W by 2 or by 1 instead of three.

1.8 Pricing American options

American options, which give the holder the right to exercise the option at any time before the expiration time, are very popular and attractive for both buyers of the

options and writers of the options. Buyers like the additional freedom compared to European options and the writers (sellers) like the possibility to earn extra money if the owner of the option does not choose the optimal time for exercising it.

1.8.1 An inequality for American options

It is clear that the price of an American option is never less than the price of the corresponding European option, so we have a lower bound on the option value. The following lemma allows us to obtain upper bounds.

Lemma 15 *If continuous and in the region $(s,t) \in (0,\infty) \times [0,T)$ two times continuously differentiable function $w(s,t)$ satisfies the inequalities*

$$\frac{\partial w}{\partial t} + \frac{s^2\sigma^2(s,t)}{2}\frac{\partial^2 w}{\partial s^2} + (r-D)s\frac{\partial w}{\partial s} - rw \leq 0,\ 0 \leq t < T,\ 0 < s < \infty$$

and

$$w(s,t) \geq p(s),$$

then the price $v(s,t)$ of the american option with the expiration date T and payoff function p satisfies the inequality $v(s,t) \leq w(s,t)\ \forall (s,t) \in 0 \leq t < T,\ 0 < s < \infty$.

Proof. Fix $t_0 \in [0,T)$ and let $s_0 = S(t_0)$. Using the investment strategy $\eta(t) = \frac{\partial w}{\partial s}(S(t),t)$ with the initial wealth $X(t_0) = w(s_0,t_0)$ we get a portfolio which value $X(t)$ satisfies the inequality

$$\begin{aligned} d(X(t) - w(S(t),t)) &= \Big(r\,X(t) - r\,S(t)\frac{\partial w}{\partial s}(S(t),t) + D\,S(t)\frac{\partial w}{\partial s}(S(t),t) \\ &\quad - \frac{\partial w}{\partial t}(S(t),t) - \frac{S(t)^2\sigma^2(S(t),t)}{2}\frac{\partial^2 w}{\partial s^2}(S(t),t)\Big)\,dt \\ &\geq r(X(t) - w(S(t),t))\,dt. \end{aligned}$$

Hence

$$d[e^{-rt}(X(t) - w(S(t),t))] \geq 0,$$

therefore also

$$\int_{t_0}^{t} d[e^{-r\tau}(X(\tau) - w(S(\tau),\tau))] = e^{-rt}(X(t) - w(S(t),t)) \geq 0\ \forall t \in [t_0,T].$$

This means that the value of the portfolio at any time t satisfies the inequality $X(t) \geq w(S(t),t) \geq p(S(t))$. Thus, at any time t_0, using the sum $w(S(t_0),t_0)$ we can form a self-financing portfolio which at any future time is at least as valuable as the option, therefore the option price can not be larger than $w(S(t_0),t_0)$.□

A corollary of the result is that the price of the European call option on a non-dividend paying stock is equal to the price of the same American option.

It can be shown that the price of an American option at each point is either equal to the payoff function or satisfies the Black-Scholes differential equation.

Lemma 16 *The price of an american option with the payoff function p is a function of two variables t and s satisfying the following complementarity problem*

$$Lv(t,s) := \frac{\partial v}{\partial t} + \frac{s^2\sigma^2}{2}\frac{\partial^2 v}{\partial s^2} + (r-D)s\frac{\partial v}{\partial s} - rv \le 0 \quad 0 \le s < T,\ s \ge 0, \tag{1.22}$$

$$v(s,t) \ge p(s), \quad 0 \le s \le T,\ s \ge 0, \tag{1.23}$$

$$Lv(s,t)\,(v(s,t) - p(s)) = 0, \quad 0 \le s < T,\ s \ge 0. \tag{1.24}$$

1.8.2 Using finite difference methods for pricing American options

Let us consider american options with payoffs that depend only on the stock price at the time of exercising the option. Then the simplest possibility to compute the prices of american options is to modify a finite difference code of computing the values of European options so that at the end of each timestep we take the maximum of the computed values of U and the payoff function:

$$U_{ik} := \max(U_{ik}, p(s_i)),\ i = 1, \ldots, n.$$

This introduces additional error of the order $O(\Delta t)$, therefore modified Crank-Nicolson method does not converge faster than the basic implicit method.

1.9 Pricing Asian options

Asian options are options that depend on the the average stock price. There are two basic types of averages - arithmetic and geometric average, and the average can be compounded discretely (eg once a day) or continuously. We consider continuously compounded averages. Continuously compounded arithmetic average of the stock price over the period $[0, T]$ is

$$A = \frac{1}{T}\int_0^T S(t)\,dt,$$

the geometric average over the same period is

$$G = e^{\frac{1}{T}\int_0^T \ln(S(t))\,dt}.$$

Both of the averages give a value that is between the minimal and maximal stock prices over the period and it can be shown that the geometric average is never larger than the arithmetic average. There are four basic types of Asian options derived from the european put and call options. If the strike price E of european options is replaced in the payoff function by an average stock price, we get average strike put/call options. If the stock price itself is replaced by an average stock price, we get average price put/call options. Other types of payoff functions depending on the final stock price and the average stock price can be considered.

In order to derive partial differential equations for the prices of Asian options, we need a more general version of $Ito\bar{o}'s$ lemma.

Lemma 17 *Let $Y_1(t)$ and $Y_2(t)$ be two stochastic processes satisfying stochastic differential equations*

$$dY_i(t) = \alpha_i(Y_1(t), Y_2(t), t)\, dt + \beta_i(Y_1(t), Y_2(t), t)\, dB_i(t),\ i = 1, 2,$$

where B_1 and B_2 are brownian motions with correlation ρ (meaning that $(B_1(t_2) - B_1(t_1), B_2(t_2) - B_2(t_1)) \sim N(0, (t_2 - t_1)\begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix})$ for all $t_1, t_2,\ t_1 < t_2$). Then

$$\begin{aligned} df(t, Y_1(t), Y_2(t)) =& \frac{\partial f}{\partial t}\, dt + \frac{\partial f}{\partial y_1}\, dY_1(t) + \frac{\partial f}{\partial y_2}\, dY_2(t) + \\ & \left(\frac{\beta_1^2}{2} \frac{\partial^2 f}{\partial {y_1}^2} + \rho\beta_1\beta_2 \frac{\partial^2 f}{\partial y_1\, \partial y_2} + \frac{\beta_2^2}{2} \frac{\partial^2 f}{\partial {y_2}^2} \right) dt \end{aligned}$$

for all sufficiently smooth (having all needed partial derivatives) functions f.

Since arithmetic average can be expressed in terms of the integral of the stock price, let us introduce a new variable $I(t)$:

$$I(t) = \int_0^t S(\tau)\, d\tau.$$

Let us derive now the partial differential equation for the price of Asian options in the cas of arithmetic average. The scheme is as before:

1. Make an assumption about what the option price depends on;
2. Assume that the option can be replicated by a self-financing portfolio;
3. use Ito's lemma for deriving the partial differential equation by requiring that the differential of the stock price and the differential of the value of the replicating portfolio to be equal.

It is clear that in the case of arithmetic average the option price depends on the current stock price, time and the integral of the stock price (since the average can be expressed in terms of the integral). Therefore assume that the Asian option price is a function of t, s and I, $v = v(t, s, I)$. In order to find differential of $v(t, S(t), I(t))$ we note that $I(t)$ satisfies

$$dI(t) = S(t)\, dt + 0\, dB_2(t),$$

so, according to Lemma 17, we have

$$\begin{aligned} dv(S(t),I(t),t) &= \frac{\partial v}{\partial t}(S(t),I(t),t)\,dt + \frac{\partial v}{\partial s}(S(t),I(t),t)\,dS(t) + \frac{\partial v}{\partial I}(S(t),I(t),t)\,dI(t) \\ &\quad + \frac{S(t)^2\sigma^2}{2}\frac{\partial^2 v}{\partial s^2}(S(t),I(t),t)\,dt \\ &= \left(\frac{\partial v}{\partial t}(S(t),I(t),t) + S(t)\frac{\partial v}{\partial I}(S(t),I(t),t) + \frac{S(t)^2\sigma^2}{2}\frac{\partial^2 v}{\partial s^2}(S(t),I(t),t)\right)\,dt \\ &\quad \frac{\partial v}{\partial s}(S(t),I(t),t)\,dS(t). \end{aligned}$$

Recall, that the value of the self-financing portfolio corresponding to holding $\eta(t)$ stocks at any time t satisfies the equation

$$dX(t) = r(X(t) - \eta(t)S(t))\,dt + D\eta(t)S(t)\,dt + \eta(t)\,dS(t).$$

If the asian option can be replicated, then the value of the corresponding self-financing portfolio is equal to the option price and the differentials of the option price and the value of the portfolio have to be the same. From the equality of $dv(t,S(t),I(t))$ and $dX(t)$ we get that $\eta(t) = \frac{\partial v}{\partial s}(t,S(t),I(t))$ and that the optio price has to satisfy the differential equation

$$\frac{\partial v}{\partial t} + \frac{s^2\sigma^2}{2}\frac{\partial^2 v}{\partial s^2} + (r-D)s\frac{\partial v}{\partial s} + s\frac{\partial v}{\partial I} - rv = 0.$$

It is actually possible to reverse the derivation of the partial differential equation and to prove the following result.

Theorem 18 *Under the assumption of the validity of the Black-Scholes market model the price v of an Asian option depending on continuously compounded arithmetic average satisfies the equation*

$$\frac{\partial v}{\partial t} + \frac{s^2\sigma^2}{2}\frac{\partial^2 v}{\partial s^2} + (r-D)s\frac{\partial v}{\partial s} + s\frac{\partial v}{\partial I} - rv = 0,\ 0 \le t < T,\ s, I \ge 0 \tag{1.25}$$

and the final condition

$$v(s,I,T) = p(s,\frac{I}{T}),$$

where p is the payoff function (the owner receives at the final time the payoff $p(S(T),A)$, where A is the arithetical average of the stock price over the period [0,T]). The price of the option at any time $t \in [0,T]$ is given by $v(S(t),\int_0^t S(\tau)\,d\tau,t)$ and the option is replicated with the self-financing strategy holding $\eta(t) = \frac{\partial v}{\partial s}(S(t),\int_0^t S(\tau)\,d\tau,t)$ stocks at any time $t \in [0,T]$.

Proof. Exercise for the reader.□

The derivation of the equation for the Asian option depending on the geometric average is similar (with I denoting the integral of logarithm of the stock price); the resulting equation has, as the only difference, the coefficient $\ln s$ in front of the term with $\frac{\partial v}{\partial I}$. Since the options depending on the geometric average are less common, we do not consider them further in the course. We'll end our discussion of Asian options depending on the geometric average with the remark that in the case of constant volatility there exist exact formulas for several types of options.

Exercise 7 *Derive (with explanations) the partial differential equation and the final condition for pricing Asian options depending on geometric average.*

As in the case of European options, it is useful to know some special solutions of the equation 1.25. Let us look for special solutions that are linear with respect to the variables s and I. So we look at solutions of the form

$$v(s, I, t) = \phi_1(t) + \phi_2(t)\, s + \phi_3(t) I.$$

Substituting this guess into the equation (1.25) we get

$$\phi_1'(t) + \phi_2'(t) s + \phi_3'(t) I + (r - D) s \phi_2(t) + s\phi_3(t) - r(\phi_1(t) + \phi_2(t)\, s + \phi_3(t) I) = 0.$$

Since this equality has to hold for all values of s and I the constant term, the coefficient of s and the coefficient of I have to be equal to 0. Thus we get a system of ordinary differential equations:

$$\begin{cases} \phi_1'(t) & = r\, \phi_1(t), \\ \phi_2'(t) & = D\phi_2(t) - \phi_3(t), \\ \phi_3'(t) & = r\, \phi_3(t). \end{cases}$$

Solving this system of equations we get in the case $r \neq D$ a family of solutions of the form

$$v(s, I, t) = C_1 e^{-r(T-t)} + e^{-D(T-t)} \left(C_2 - C_3 \frac{e^{-(r-D)(T-t)}}{r - D} \right) s + C_3\, e^{-r(T-t)} I$$

and in the case $r = D$ a family of solutions of the form

$$v(s, I, t) = C_1 e^{-r(T-t)} + e^{-r(T-t)} \left(C_2 + C_3 (T - t) \right) s + C_3\, e^{-r(T-t)} I.$$

This special solutions allow us to obtain put-call parities for Asian options.

Recall that there are two kinds of put/call options of Asian type: average strike options and average price options. Let us derive a put/call relationship for average strike options.

Since the payoff of the average strike call is $\max(s - A, 0)$ and the payoff of the average strike put is $\max(A - s, 0)$, the portfolio corresponding to buying one average strike call and selling (writing) one average strike put gives the holder at the final time the income $S(T) - A$, where A is the continuously computed arithmetic average of the

stock price. This means that if $C(s,I,t)$ is the function giving the value of the call option and $P(s,I,t)$ corresponds to the put option, then

$$C(s,I,T) - P(s,I,T) = s - \frac{I}{T}.$$

Since the equation (1.25) is a linear equation and both C and P are solutions of this equation, the difference is also a solution of the equation satisfying a linear final condition. Since linear final conditions correspond to special solutions, the difference of the call and put value is given by the formula of the special solution with the constants $C_1 = 0,\ C_3 = -\frac{1}{T},\ C_2 = 1 - \frac{1}{T(r-D)}$ (if $r \neq 0$) or $C2 = 1$, if $r = D$. Thus, in the case $r \neq D$ we have

$$C(s,I,t) = P(s,I,t) + e^{-D(T-t)}\left(1 + \frac{e^{-(r-D)(T-t)} - 1}{T(r-D)}\right) s - \frac{e^{-r(T-t)}}{T} I$$

and in the case $r = D$ we have

$$C(s,I,t) = P(s,I,t) + e^{-r(T-t)}\left(1 - \frac{(T-t)}{T}\right) s - \frac{e^{-r(T-t)}}{T} I.$$

1.9.1 A finite difference method for pricing Asian options depending on arithmetic average

In order to solve the equation (1.25) numerically, we introduce artificial boundaries $I = I_{max}$ and $s = S_{max}$ and derive equations for determining approximate values of the option price at the points (s_i, I_j, τ_k), where

$$\tau_k = k \cdot \frac{T}{m},\ s_i = i \cdot \frac{S_{max}}{n_s} \text{ and } I_j = j \cdot \frac{I_{max}}{n_I}$$

for some natural numbers m, n_s, n_I. Here τ denotes the remaining lifetime of the option and the equation (1.25) is, after the change of the variable $\tau = T - t$, in the following form:

$$\frac{\partial u}{\partial \tau} = \frac{s^2\sigma(s,I,T-\tau)^2}{2}\frac{\partial^2 u}{\partial s^2} + (r-D)s\frac{\partial u}{\partial s} + s\frac{\partial u}{\partial I} - ru.$$

Denote by U_{ij}^k the approximate values of the option price at the point (s_i, I_j, τ_k). In order to derive equations for determining the approximate values, let us use the PDE at the points $s_i, I_{j+\frac{1}{2}}, (\tau_{k-\frac{1}{2}}) = (s_i, I_j + \frac{1}{2}\Delta I, \tau_{k-1} + \frac{1}{2}\Delta\tau)$, where $\Delta\tau = \frac{T}{m}$ and $\Delta I = \frac{I_{max}}{n_I}$.

Instead of the partial derivatives, we use the following approximations:

$$\begin{aligned}
\frac{\partial u}{\partial \tau}(s_i, I_{j+\frac{1}{2}}, \tau_{k-\frac{1}{2}}) &= \frac{1}{2}\left(\frac{\partial u}{\partial \tau}(s_i, I_j, \tau_{k-\frac{1}{2}}) + \frac{\partial u}{\partial \tau}(s_i, I_{j+1}, \tau_{k-\frac{1}{2}})\right) + O(\Delta I^2) \\
&= \frac{u^k_{i,j} - u^{k-1}_{i,j}}{2\Delta\tau} + \frac{u^k_{i,j+1} - u^{k-1}_{i,j+1}}{2\Delta\tau} + O(\Delta\tau^2 + \Delta I^2), \\
\frac{\partial u}{\partial s}(s_i, I_{j+\frac{1}{2}}, \tau_{k-\frac{1}{2}}) &= \frac{1}{2}\left(\frac{\partial u}{\partial s}(s_i, I_j, \tau_k) + \frac{\partial u}{\partial s}(s_i, I_{j+1}, \tau_{k-1})\right) + O(\Delta\tau^2 + \Delta I^2) \\
&= \frac{u^k_{i+1,j} - u^k_{i-1,j}}{4\Delta s} + \frac{u^{k-1}_{i+1,j+1} - u^{k-1}_{i-1,j+1}}{4\Delta s} + O(\Delta\tau^2 + \Delta s^2 + \Delta I^2), \\
\frac{\partial^2 u}{\partial s^2}(s_i, I_{j+\frac{1}{2}}, \tau_{k-\frac{1}{2}}) &= \frac{1}{2}\left(\frac{\partial^2 u}{\partial s^2}(s_i, I_j, \tau_k) + \frac{\partial^2 u}{\partial s^2}(s_i, I_{j+1}, \tau_{k-1})\right) + O(\Delta\tau^2 + \Delta I^2) \\
&= \frac{u^k_{i-1,j} - 2u^k_{ij} + u^k_{i+1,j}}{2\Delta s^2} + \frac{u^{k-1}_{i-1,j+1} - 2u^{k-1}_{i,j+1} + u^{k-1}_{i+1,j+1}}{2\Delta s^2}, \\
u(s_i, I_{j+\frac{1}{2}}, \tau_{k-\frac{1}{2}}) &= \frac{1}{2}(u^k_{ij} + u^{k-1}_{i,j+1}) + O(\Delta\tau^2 + \Delta I^2), \\
\frac{\partial u}{\partial I}(s_i, I_{j+\frac{1}{2}}, \tau_{k-\frac{1}{2}}) &= \frac{1}{2}\left(\frac{\partial u}{\partial I}(s_i, I_{j+\frac{1}{2}}, \tau_{k-1}) + \frac{\partial u}{\partial I}(s_i, I_{j+\frac{1}{2}}, \tau_k)\right) + O(\Delta\tau^2) \\
&= \frac{u^k_{i,j+1} - u^k_{i,j}}{2\Delta I} + \frac{u^{k-1}_{i,j+1} - u^{k-1}_{i,j}}{2\Delta I} + O(\Delta\tau^2 + \Delta I^2).
\end{aligned}$$

After substituting those approximations to the PDE and throwing away the error terms, we get the following equations for the approximate values of the option prices:

$$\begin{aligned}
a_{kij}U^k_{i-1,j} + b_{kij}U^k_{ij} + c_{kij}U^k_{i+1,j} = d_{kij}U^{k-1}_{i-1,j+1} + e_{kij}U^{k-1}_{i,j+1} + f_{kij}U^{k-1}_{i+1,j+1} \\
+ g_{kij}(U^k_{i,j+1} - U^{k-1}_{i,j}).
\end{aligned}$$

where (using the notation $\rho = \frac{\Delta\tau}{\Delta s^2}$)

$$\begin{aligned}
a_{kij} &= \frac{\rho}{4}(-s_i^2\sigma^2(s_i, I_{j+\frac{1}{2}}, T - \tau_{k-\frac{1}{2}}) + (r - D)s_i\Delta s), \\
b_{kij} &= \frac{1}{2}\left(1 + \rho s_i^2\sigma^2(s_i, I_{j+\frac{1}{2}}, T - \tau_{k-\frac{1}{2}}) + \frac{s_i\Delta\tau}{\Delta I} + r\Delta\tau\right), \\
c_{kij} &= -\frac{\rho}{4}(s_i^2\sigma^2(s_i, I_{j+\frac{1}{2}}, T - \tau_{k-\frac{1}{2}}) + (r - D)s_i\Delta s), \\
d_{kij} &= -a_{kij} \\
e_{kij} &= \frac{1}{2}\left(1 - \rho s_i^2\sigma^2(s_i, I_{j+\frac{1}{2}}, T - \tau_{k-\frac{1}{2}}) + \frac{s_i\Delta\tau}{\Delta I} - r\Delta\tau\right), \\
f_{kij} &= -c_{kij} \\
g_{kij} &= \frac{1}{2}\left(-1 + \frac{s_i\Delta\tau}{\Delta I}\right).
\end{aligned}$$

If σ does not depend on t and I, then the coefficients $a, b, \ldots, g$ depend only on the index i.

The equations for the approximate values can be viewed as a three-diagonal system for finding the values of U^k_{ij}, if the values corresponding to the level $\tau = \tau_{k-1}$ and the values $U^k_{i,j+1},\ i = 0, \ldots, n_s$ have been found earlier.
The values of $U^{0,i,j}$ can be found from the initial condition:

$$U^0_{i,j} = p(s_i, \frac{I_j}{T}),\ i = 0, \ldots, n_s,\ j = 0, \ldots, n_I.$$

At the boundary $S = 0$ we have the exact boundary condition $u(0, I, \tau) = p(0, \frac{I}{T})e^{-r\tau}$, hence

$$U^k_{0,j} = p(0, \frac{I_j}{T})e^{-r\tau_k},\ k = 1, \ldots, m.$$

In order to solve the system of equation for U^k_{ij} we have to specify boundary conditions at the boundaries $I = I_{max}$ and $S = S_{max}$. There are a few possible boundary conditions one can specify at those boundaries:

- If the payoff function is piecewise linear, then we may take maximum of the corresponding special solution and 0 at the boundaries.
- Use the boundary condition that corresponds to S remaining constant at the boundary; in that case the value at the boundary corresponds to the function $e^{-r\tau}p(s, \frac{I+\tau s}{T})$.

In the case of average price call and put options it can be shown that we can choose $I_{max} = E \cdot T$ and the exact boundary condition at $I = I_{max}$ is (assuming $r \neq D$) 0 in the case of average price put and is given by

$$u(\tau, s, I_{max}) = \frac{e^{-D\tau}}{(r - D)T}(1 - e^{-(r-D)\tau})\, s$$

for average price call option.
The procedure for solving the option pricing equation is as follows.

1. Fill in the values for $U^0_{ij},\ i = 0, \ldots, n_s,\ j = 0, \ldots, n_I$, using the payoff function.
2. For each $k = 1, \ldots, m$
   (a) Apply the boundary conditions.
   (b) For each $j = n_I - 1, \ldots, 0$ solve the three-diagonal system for values $U^k_{ij},\ i = 1, \ldots, n_s - 1$.

If we need only the option price at $t = 0$, then it is not necessary to store the full matrix U of approximate option prices; we need only two levels, $Uold$ corresponding to $\tau = \tau_{k-1}$ that is known and $Unew$ corresponding to the current level $\tau = \tau_k$. At the beginning $Uold$ is computed usng the initial condition and at the end of each timestep the values of $Unew$ are copied to $Uold$.

Chapter 2

Options depending on two underlying stocks

Sometimes it is of interest to consider options whose payoff functions depend on more than one stock prices. A few examples of popular so called rainbow options:

- Dual Put option, where the owner has the right to sell one of the two underlying stocks at the exercise time T: $p(s_1, s_2) = \max(E_1 - s_1, E_2 - s_2, 0)$;
- Dual Strike option, where the owner has the right to buy one of the two underlying stocks at the exercise time T: $p(s_1, s_2) = \max(s_1 - E_1, s_2 - E_2, 0)$.

Let us assume that the two stocks satisfy the system of stochastic differential equations

$$\begin{aligned} dS_1(t) &= S_1(t)(\mu_1\, dt + \sigma_1\, dB_1(t)), \\ dS_2(t) &= S_2(t)(\mu_2\, dt + \sigma_2\, dB_2(t)), \end{aligned}$$

where $\mu_1, \mu_2, \sigma_1, \sigma_2$ may depend on time and on both stock prices and $(B_1(t), B_2(t))$ is a two-dimensional Brownian motion with correlation ρ (the increments of B_1 and B_2 over any time interval Δt are jointly normal with standard deviations $\sqrt{\Delta t}$ and correlation ρ).

Then, applying Itô's lemma and the arbitrage principle it is possible to show that the price v of european option with a payoff function $p(s_1, s_2)$ satisfies a two-dimensional Black-Scholes Equation

$$\begin{aligned} \frac{\partial v}{\partial t} + \frac{\sigma_1^2 s_1^2}{2}\frac{\partial^2 v}{\partial s_1^2} + \rho\sigma_1\sigma_2 s_1 s_2 \frac{\partial^2 v}{\partial s_1 \partial s_2} + \frac{\sigma_2^2 s_2^2}{2}\frac{\partial^2 v}{\partial s_2^2} \\ + (r - D_1)s_1\frac{\partial v}{\partial s_1} + (r - D_2)s_2\frac{\partial v}{\partial s_2} - rv = 0,\ \ s_1, s_2 \geq 0 \end{aligned}$$

with the final condition

$$v(s_1, s_2, T) = p(s_1, s_2),\ \ s_1, s_2 \geq 0.$$

Using the change of variables $v(s_1, s_2, t) = u(\frac{\ln(s_1)}{\sigma_1}, -\rho\frac{\ln(s_1)}{\sigma_1} + \frac{\ln(s_2)}{\sigma_2}, T-t)$ the equation is transformed to the equation

$$\frac{\partial u}{\partial \tau} = \frac{1}{2}\frac{\partial^2 u}{\partial x_1^2} + \frac{1-\rho^2}{2}\frac{\partial^2 u}{\partial x_2^2} + \frac{r - D_1 - \frac{1}{2}\sigma_1^2}{\sigma_1}\frac{\partial u}{\partial x_1} + \left(\frac{r - D_2 - \frac{1}{2}\sigma_2^2}{\sigma_2} - \rho\frac{r - D_1 - \frac{1}{2}\sigma_1^2}{\sigma_1}\right)\frac{\partial u}{\partial x_2} - ru.$$

From the final condition for v it follows that u satisfies the initial condition

$$u(x_1, x_2, 0) = p(e^{\sigma_1 x_1}, e^{\sigma_2 x_2 - \rho\sigma_1 x_1}).$$

In order to solve the transformed equation numerically we introduce artificial boundaries $x_{1,min}, x_{1,max}, x_{2,min}, x_{2,max}$ and look for approximate solution at the points $(\tau_k, x_{1,i}, x_{2,j})$, where

$$\tau_k = k\,\delta\tau,\ x_{1,i} = x_{1,min} + i\,\delta x_1,\ x_{2,j} = x_{2,min} + j\,\delta x_2$$

and

$$\tau = \frac{T}{m},\ \delta x_1 = \frac{x_{1,max} - x_{1,min}}{n_1},\ \delta x_2 = \frac{x_{2,max} - x_{2,min}}{n_2}$$

for some natural numbers m, n_1, n_2.

We use the usual notation $U_{k,i,j} \approx u(x_{1,i}, x_{2,j}, \tau_k)$. Replacing the derivatives with the suitable finite difference approximations we get an explicit finite difference method

$$U_{k+1,i,j} = a_{-1,0}U_{k,i-1,j} + a_{0,0}U_{k,i,j} + a_{1,0}U_{k,i+1,j} + a_{0,1}U_{k,i,j+1} + a_{0,-1}U_{k,i,j-1},$$

where

$$\begin{aligned}
a_{-1,0} &= \frac{\Delta\tau}{2}\left(\frac{1}{\Delta x_1^2} - \frac{r - D_1 - \frac{1}{2}\sigma_1^2}{\sigma_1\Delta x_1}\right),\\
a_{1,0} &= \frac{\Delta\tau}{2}\left(\frac{1}{\Delta x_1^2} + \frac{r - D_1 - \frac{1}{2}\sigma_1^2}{\sigma_1\Delta x_1}\right),\\
a_{0,-1} &= \frac{\Delta\tau}{2}\left(\frac{1-\rho^2}{\Delta x_2^2} - \frac{r - D_2 - \frac{1}{2}\sigma_2^2}{\sigma_2\Delta x_2} + \rho\frac{r - D_1 - \frac{1}{2}\sigma_1^2}{\sigma_1\Delta x_2}\right),\\
a_{0,1} &= \frac{\Delta\tau}{2}\left(\frac{1-\rho^2}{\Delta x_2^2} + \frac{r - D_2 - \frac{1}{2}\sigma_2^2}{\sigma_2\Delta x_2} - \rho\frac{r - D_1 - \frac{1}{2}\sigma_1^2}{\sigma_1\Delta x_2}\right),\\
a_{0,0} &= 1 - r\Delta\tau - \frac{\Delta\tau}{\Delta x_1^2} - \frac{\Delta\tau(1-\rho^2)}{\Delta x_2^2}.
\end{aligned}$$

This method is stable in the case of sufficiently large values of n_1 and n_2, if the coefficient $\delta\tau$ (or the number of timesteps m) is chosen so that the coefficient $a_{0,0}$ is nonnegative.

In order to find the option prices with the explicit method one has to specify also boundary conditions at the artificial boundaries. The simplest choice is to fix the value at the boundaries to be equal to the value of the payoff function at the corresponding points.

Bibliography

[1] Black F, Scholes M, "The pricing of options and corporate liabilities," *Journal of Political Economy 81* (1973), 637-659.

[2] Glassermann, P. Monte Carlo Methods in Financial Engineering. Springer 2004.

[3] Strang G, Fix G. An Analysis of The Finite Element Method. Prentice Hall 1973.

[4] Wilmott P, Howison S, Dewynne. J, The Mathematics of Financial Derivatives. A Student Introduction. Cambridge, 1995.

[5] Wilmott P. Paul Wilmott on quantitative finance (Volumes 1,2). Jon Wiley and Sons, 2000.

Python essentials for the Computational Finance course

The aim of the document is to discuss a minimal number of Python commands that are needed for this course. Reading Python and Scipy tutorials is highly encouraged. This document is based on Python 2.7.

1 Some warnings

NB! if you divide integers in Python 2.x, the result is an integer; ie `1/4` gives 0. If this is not what you want, then use decimal numbers (`1.0/4.0`)

In Python, **numbering of elements of an object (array, string, list, sequence) starts from 0**; index of the element is given in square brackets

Example:

```
a=(1,2,3); print a[1]
```

outputs the second element of `a`, that is 2. (Remark: in python 3.x the syntax of the print command changes, the arguments have to be given in parantheses)

2 Programming constructs

It is important to remember, that Phython groups commands by indentation.

The format of `for` cycle:

```
for i in Something:
  first command
  second command
  ....
  last command of the cycle
next commands (outside for)
```

"`Something`" is usually an object that has many elements; `i` takes the value of each element of the object `Something`

Example:

```
for x in (1,10,"blue"):
    print x
```

The format of the `while` cycle is as follows:

```
while condition:
   first command
   second command
   ...
   last command
commands outside while
```

Example:

```
i=1
while i<5:
   print i,'square is',i*i
   i=i+1
```

The format of the `if` statement is as follows:

```
if condition:
  first command
  second command
  etc
elif condition:
  first command
  etc
else:
  first command
  etc
other commands
```

There may be many `elif` (else if) parts.

Example:

```
x=input('x=')
if x==0:
   print 'zero'
elif x==1:
   print 'one'
elif x==2:
   print 'two'
else:
   print 'a large number'
```

3 Using modules an packages of Python. Defining new functions

Using modules and packages. Python functions are organized in modules or packages (collections of modules). There are two ways to use those functions. 1) import the module with the command

```
import module_name
```

Then it is possible to use a function `func()` from the module in the form `module_name.func()`.
2) import the needed functions from the module using the command

```
from module_name import fun1,fun2,etc
```

or, to import all functions,

```
from module_name import *
```

then it is possible to use function names directly. In the case of a package the command "`from package import *`" does not import all functions from all subpackages of the package; they should be imported separately.

Examples:

```
import time
time.ctime()
from scipy import sin
sin(0.5)
from scipy import *
cos(pi)
```

Defining your own functions: the `def` command. Example:

```
def f(x,y=1,z=0):
tmp=x*y+z
return(tmp)
```

If default value for a variable is given, then it is not necessary to specify it's value: valid uses of the function are for example

```
f(5)
f(2,3)
f(2,3,4)
f(1,z=2)
f(z=2,x=0,y=1)
```

invalid uses are: `f()`- `x` does not have a default value, `f(z=2,3)` - unnamed arguments have to be before named arguments.

4 Numerical computations in Python: the package SciPy

For numerical computations in Python there is the package **scipy** (which hast to be installed together with the package **numpy**). The functions in the packages can operate on arrays, that speeds up computations a lot.

4.1 Creating arrays.

arange(start,stop,step=1) - creates array of the elements start, start+step, start+2*step, ... which are less than stop (stop is not included)

linspace(start,stop, num=50) - divides the interval [start,stop] into num-1 equal subintervals (ie returns num equally spaced points including start and stop)

zeros(shape) returns an array filled with zeros; the dimensions of the matrix are in the variable shape Examples:
`zeros(10)` - one-dimensional array with 10 elements;
`zeros(shape=(3,4))` - two-dimensional array with 3*4 elements

ones(shape) - array filled with ones

empty(shape) - an array with given dimension with arbitrary values

array([[1,2],[3,4],[5,6]]) - 3*2 matrix

4.2 Accessing array elements

Elements are numbered starting from 0.
For one-dimensional array `A`:
`A[1]` gives the second element of `A`
`A[1:3]` gives second and third elements (ie `A[1]`,`A[2]`, but not `A[3]`)
`A[2:]` gives all elements starting from the third (ie `A[2]`,`A[3]` etc)
`A[:3]` gives first three elements, ie `A[0]`,`A[1]`,`A[2]`.
`A[2:-1]` gives all elements except two first and 1 last; negative index after colon indicates how many elements to leave out from the end
If `b` is an array of integers, then `A[b]` returns the elements of A which have indeces in the array `b` in the same order as they are listed in `b`
for two-dimensional array:
`A[i,j]` gives the single element,
`A[i]` gives the (i+1)th row,
`A[:,j]` gives the (j+1)th column,
`A[A>0]` gives all elements that are greater than 0

Examples:

```
b=arange(1,11)
A=empty(shape=(2,10))
A[0]=b
A[1,0:3]=2
A[1,3:]=5
print(A)
print A[:,3]
A[A<3]=0
print A
z=array([5,1,4,1])
print A[1,z]
```

WARNING: assignments like `B=A` or `b=A[:,1]` DO NOT COPY values of `A` to new matrices; in this case `B` is just another name for the entries of `A` (ie they use exactly the same values, modifying one modifies the other, too) and `b` is just a name to use the second column of `A`; `b[0]=10` sets the element `A[0,1]` to be equal to 10. If copying of values is needed, then the commands of the form `B=A.copy()` and `b=A[:,1].copy()` should be used.
Arrays can be added or subtracted elementwise, also multiplication and division works elementwise. In order to multiply arrays as matrices, one has to use the command `dot(A,B)`

4.3 Other useful array functions:

sum(A) - the sum of elements of an array;

mean(A) - the average of the elements of `A`;

std(A) - standard deviation of elements of `A`;

amin(A) - minimal value of elements of `A`;

amax(A) - maximal value of elements of `A`;

minimum(A,B) - elementwise minimum of two arrays (or an array and a number)

maximum(A,B) - elementwise maximum of two arrays

log(A) - natural logarithm of elements of `A`;

5 Graphics and file input

For graphics the package `matplotlib` should be installed. For simple plots the following commands work:

```
from pylab import plot, show
plot(x,y)
show()
```

The `show()` command should be the last command in a script, then all results of previous plot commands are shown together. Arguments `x,y` can be 1D arrays or 2d arrays. In the last case the plots corresponding to the columns of the arrays are created. If `x` has only one column or is 1D array, then it is used with every column of `y`.

Example:

```
from scipy import *
from pylab import plot,show
n=101
x=linspace(0,1,num=n)
y=empty(shape=(n,2))
y[:,0]=cos(x)
y[:,1]=exp(-x)
plot(x,y)
show()
```

Importing data from a csv file (assuming the decimal separator is . and field separator is , and that from the package `scipy` everything is imported)

```
x=loadtxt(filename,delimiter=',', \
  usecols=seq_of_columns, skiprows=n)
```

Sequence of columns numbers (starting from 0!) in the sequence `usecols` is of the form `(a,b,c,etc)`. If the argument `usecols=` is not given, all columns will be read in; otherwise only columns indicated in the sequence are read. The parameter `skiprows` specifies the number of rows to ignore at the beginning of the file.

Examples: create a file called data.csv with the content

```
"january",2.0,-2.0,3
"february",1,0,2
"march",5,1,3
```

The command

```
x=loadtxt("data.csv", delimiter=',', \
        usecols=(3,))
print x
```

reads in only the 4th column; the command

```
x=loadtxt("data.csv",separator=',', \
    usecols=(1,2),skiprows=1)
print x
```

reads in columns number 1,2 (ie the second and the third column) and skips the first line of the file.

6 Functions related to the standard normal distribution

Here it is assumed, that the commands `from scipy import *` and `from scipy import stats` has been entered previously.

randn(d1,d2,...,dn) - creates a n-dimensional array filled with normally distributed random numbers

Phi=stats.norm.cdf - defines `Phi` as cumulative distribution function of the standard normal distribution

invPhi=stats.norm.ppf - defines `invPhi` to be the inverse of the cumulative distribution function of the standard normal distribution

Computational Finance, Fall 2011
Computer Lab 1

The aim of the Lab is to get acquinted with the Python programming language.
Exercises:

1. Define a function `f1(x)=x*sin(x)`. Plot the graph of the function for $0 \leq x \leq 10$.

2. Recall that the midpoint rule for computing integrals is as follows:

$$\int_a^b f(x)\,dx \approx h \cdot \sum_{i=1}^{n} f(x_i),$$

   where n is a given natural number, $h = \frac{b-a}{n}$ and $x_i = a + (i - \frac{1}{2}) \cdot h,\ i = 1, 2, \ldots, n$. Write a function `midpoint(f, a,b,n)`, which uses the midpoint rule to compute approximately the value of the integra of f over the interval $[a, b]$. Use the function to compute approximately

$$\int_0^1 x \cdot \sin(x)\,dx,$$

   using $n = 100$ subintervals. Compare the answer to the exact answer (that you hopefully can compute by yourself)

3. Write a script, that defines the variables m and n, defines x to be an array with the elements $x_i = \frac{i}{m},\ i = 0, \ldots, m$ and then creates a $(m + 1) \times n$ array Y as follows:
   The first column is filled acoording to $Y_{i0} = \sin(4\pi\, x_i),\ i = 0, \ldots, m$
   For each $j = 1, \ldots, n - 1$ we compute

$$\begin{aligned} Y_{0,j} &= Y_{m,j} = 0 \\ Y_{ij} &= \frac{Y_{i-1,j-1} + Y_{i,j-1} + Y_{i+1,j-1}}{3},\ i = 1, \ldots, m - 1. \end{aligned}$$

   Plot the graphs of the last column of Y using the array x for x-coordinates.

Computational Finance, Fall 2011
Computer Lab 2

The aim of the Lab is to define some useful functions given by Black-Scholes option pricing formulas and to learn to simulate the paths of solutions of stochastic differential equations corresponding to common stock market models.

Two important functions in mathematical finance are Black-Scholes formulas for call and put option prices under the assumption, that the Black-Scholes market model with constant parameters holds. The formulas are as follows:

$$Call(S,E,T,r,\sigma,D) = Se^{-D\,T}\Phi(d_1) - Ee^{-r\,T}\Phi(d_2),$$

$$Put(S,E,T,r,\sigma,D) = -Se^{-D\,T}\Phi(-d_1) + Ee^{-r\,T}\Phi(-d_2),$$

where

$$d_1 = \frac{\ln(\frac{S}{E}) + (r - D + \frac{\sigma^2}{2})\cdot T}{\sigma\sqrt{T}},\ d_2 = d_1 - \sigma\sqrt{T},$$

S is the current stock price, T is the time to expiry of the option and Φ is the cumulative distribution function of the standard normal distribution.

Exercise 1. Define in Python Black-Scholes Call an Put price functions (including a suitable description). Save them in a file named BSformulas.py. Test the correctness of the functions:

$$Call(S=100, E=100, T=0.5, sigma=0.5, r=0.05, D=0.01) = 14.830417641356284,$$

$$Put(S=100, E=100, T=0.5, sigma=0.5, r=0.05, D=0.01) = 12.86016092492131.$$

Sometimes (especially for applying Monte-Carlo methods) it is important to know how to simulate the stock price trajectories corresponding to a market model. A simple and quite universal (but often not the best) way to generate the trajectories of solutions of stochastic differential equations is Euler's method, where differentials are replaced by differences over small time intervals. For Black-Scholes market model

$$dS(t) = S(t)\big(\,\mu(t)\,dt + \sigma(S(t),t)\,dB(t)\,\big)$$

this leads to an approximation

$$S(t_i) - S(t_{i-1}) \approx S(t_{i-1})\big(\,\mu(t_{i-1})\,h_i + \sigma(S(t_{i-1}),t_{i-1})\,(B(t_i) - B(t_{i-1}))\,\big),$$

where $0 = t_0 < t_1 < \ldots < t_m = T$ is a partition of the interval $[0,T]$ into (usually equal) subintervals and $h_i = t_i - t_{i-1}$. Using this approximation, the knowledge that $B(t_i) - B(t_{i-1}) \sim N(0,\sqrt{h_i})$ and a given value of $S_0 = S(0)$ we can compute approximate values $S_1, S_2, \ldots, S_m$ of $S(t_1), S(t_2), \ldots, S(t_m)$ by

$$S_i = S_{i-1}\big(1 + \mu(t_{i-1})\,h_i + \sigma(S_{i-1},t_{i-1})\sqrt{h_i}X_i\big),\ i = 1,\ldots,m,$$

where X_i are independent random variables with the standard normal distribution.

Exercise 2. Write a function `BSgraph(S0,n,m,mu,sigma,T)` that plots the graph of n trajectories of the stock price on the interval $[0,T]$, corresponding to the Black-Scholes market model with constant parameters μ and σ. For computing the values of the stock prices divide the interval $[0,T]$ into m equal subintervals (ie. use the time points $t_i = \frac{i\cdot T}{m},\ i = 0,1,\ldots,m$) and use the Euler's method.

It is well known that the Black-Scholes model is not perfect and that, if a similar model holds, then the volatility can not be constant. A possible alternative is to allow the volatility to be stochastic, too. One such model is the GARCH volatility model:

$$dS(t) = S(t)((r - D)\,dt + \sqrt{V(t)}\,dB_1(t)),$$
$$dV(t) = \theta \cdot (\omega - V(t))\,dt + \xi V(t)\,dB_2(t),$$

where $r, D, \theta, \omega, \xi$ are known parameters and B_1, B_2 are independent Brownian motions.

Homework problem 1. (Deadline September 14, 2011) Write a function GARCH, that for given m and n and for $S(0) = 50, V(0) = 0.3, r = 0.1, D = 0, \omega = 0.25, \theta = 0.5, \xi = 0.5, T = 0.5$ draws the graph with n trajectories of the stock price S on the interval $[0, T]$, using m equal timesteps and the Euler's method.

Computational Finance, Fall 2011
Computer Lab 3

The aim of the Lab is to learn some possibilities for identifying market model parameters from historical data.

Consider the Black-Scholes market model

$$dS(t) = S(t)(\mu(t)\,dt + \sigma(S(t),t)\,dB(t).$$

We have shown in the lecture that in the case of constant parameters μ and σ we have

$$d(\ln S(t)) = (\mu - \frac{\sigma^2}{2})\,dt + \sigma\,dB(t)$$

and hence

$$\ln \frac{S(t_2)}{S(t_1)} = (\mu - \frac{\sigma^2}{2})\,(t_2 - t_1) + \sigma(B(t_2) - B(t_1)).$$

Consequently

$$\ln \frac{S(t+\Delta t)}{S(t)} \sim N((\mu - \frac{\sigma^2}{2})\Delta t, \sigma\sqrt{\Delta t}).$$

So, if $S_i,\ i = 0, 1, 2, \ldots$ are closing prices of a stock at consecutive trading days, then $x_i = \ln \frac{S_{i+1}}{S_i}$ are values of normally distributed iid random variables. Usually time is measured in years, thus $\Delta t = \frac{1}{\text{trading days in a year}}$. By computing the mean and standard deviation of x_i we can find estimates for the constants μ and σ

Exercise 1. Download from http://finance.google.com/ historical prices of the stock of Amazon.com corporation for one year as a *.csv file. Write a script, that reads the closing prices from the file and uses them to estimate μ and σ (assuming the BS model with constant parameters holds). NB! pay attention to the order the data is in the file!

Unfortunately stock prices usually do not behave according to the model with constant coefficients. In order to see if the model suits for a particular stocks we can use various tests to verify the hypothesis of x_i to have normal distributions. Two such tests are Anderson-Darling test (the function `stats.anderson()`) in Python) and Shapiro-Wilk test (the function `stats.shapiro()` in Python).

Exercise 2. Read the help information of the normality tests. Check if we can assume that the Amazon.com stock prices follow the Black-Scholes model with constant coefficients.

If we do not want to assume that the parameters are constant, we may start with approximating the market model:

$$\frac{S(t_{i+1}) - S(t_i)}{S(t_i)} \approx \mu(t_i)\,(t_{i+1} - t_i) + \sigma(S(t_i), t_i)(B(t_{i+1}) - B(t_i)).$$

Next, we introduce a finite number of unknown parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_k)$ and make an assumption how the functions $\mu = \mu_\theta$ and $\sigma = \sigma_\theta$ depend on those parameters. One way to find those parameters is to maximize the log-likelihood function: if Y_i are random variables with probability density functions f_i, then the log-likelihood function of the values y_i is

$$\sum_i \ln f_i(y_i).$$

Since in our case the random variables $Y_i = \frac{S_{i+1}-S_i}{S_i}$ are according to the approximate market model normally distributed with mean $\mu_\theta(t_i)\Delta t$ and standard deviation $\sigma_\theta(S_i, t_i)\sqrt{\Delta t}$ we have to maximize the function

$$\text{loglike}(\theta) = -\sum_i \left(\frac{(Y_i - \mu_\theta(t_i)\Delta t)^2}{2\sigma_\theta(S_i, t_i)^2 \Delta t} + \ln \sigma_\theta(S_i, t_i) \right).$$

Often there are no procedures for maximizing a function in software packages but minimization procedures are available. Fortunately this does not pose any problems, since maximization of a funciton $f(\theta)$ is equivalet to minimization of the function $g(\theta) = -f(\theta)$. Therefore, in Python, we are going to minimize the function

$$f(\theta) = \sum_i \left(\frac{(Y_i - \mu_\theta(t_i)\Delta t)^2}{2\sigma_\theta(S_i, t_i)^2 \Delta t} + \ln \sigma_\theta(S_i, t_i) \right).$$

In Python there are various commands for finding the minimum of a function; we try `optimize.fmin()` and `optimize.fmin_cg()`.

Exercise 3. Assume that

$$\mu_\theta(t_i) = \theta_0 + \theta_1 \cdot \frac{S_i - S_{i-1}}{\Delta t}$$

and

$$\sigma_\theta(s, t) = \theta_2.$$

Use the maximal likelihood method to determine the optimal values of θ.

Exercise 4. We can again test the validity of our assumptions. Namely, if our assumptions are correct, then $\frac{Y[i]-\mu(t_i)}{\sigma t_i}$ shold be independent random variables from a normal distribution. So we can again use the tests for normality. Please test the validity of the model fitted in the previous exercise.

Computational Finance, Fall 2011
Computer Lab 4

The aim of the Lab is to learn to learn to determine market parameters from the prices of traded options.

If we make assumptions about market behavior or about the methods of option pricing, we get functions that for a given set of market (or option pricing) parameters gives theoretical values of every concrete option. Black-Scholes formulas are examples of such functions that for given value of the volatility σ and for given put or call option parameters (exercise price, duration) give the price of the option. Whenever we have such functions (which can be explicit formulas or some computer programs that compute the prices) and there are available prices of some traded options we can try to determine the unknown parameters from the known option prices. More precisely, suppose that we know the current prices $V_1, V_2, \ldots, V_m$ of m different options and that $f_i(\theta)$ are the functions that give the option prices for (unknown) market parameters θ. Then we have m equations:

$$f_i(\theta) = V_i,\ i = 1, \ldots, m.$$

Usually the number of unknown market parameters is much smaller than the number of available option prices, so the system of equations may be solved in the least squares sense by minimizing the function

$$F(\theta) = \frac{1}{2}\sum_{i=1}^{m}(f_i(\theta) - V_i)^2.$$

Let us use the current information about prices of 3-months call options for Cisco stock available from `http://finance.yahoo.com/`.

Exercise 1. (Implied volatility) Let us assume that the Black-Scholes market model with constant volatility holds, then call option prices can be computed by Black-Scholes formula. Assume $r = 0.02$ and $D = 0$, then the only unknown parameter is σ. Since we have only one unknown parameter, only one equation is needed to determine the value of σ and if the assumption about the market model is correct, then every known option price should give the same value of σ. Use the equation solver `fsolve` of `scipy.optimize` to find a value of σ for each of 6 actual option price corresponding to 6 strike prices closest to the current share price. Show the dependence of found σ values on the exercise price on a graph. In order to do this, for each value or the exercise price define a function of one argument sigma that computes the difference of the corresponding theoretical call option price and the observed price of the option and use this function as an input for the command `optimize.fsolve` together with a suitable initial guess for the parameter σ.

Exercise 2. Define a function that for a given value of σ computes the sum of squares of differences of the theoretical and observed option prices and use a minimizer from `scipy.optimize` to find the least squares estimate of σ. For this σ, find the largest difference between the theoretical and observed option prices.

Exercise 3. Consider Black-Scholes market model with the non-constant volatility

$$\sigma(s,t) = |\theta_0 + \theta_1 \arctan(0.3(s-16))|.$$

From the course web page you can download a module `lab4solver` that contains a function `lab4solver(theta,E,S0)` that for a given values of E and $S0$ and for a given parameter vector θ computes the theoretical price of the corresponding call option, for which we have the market price. Use the function and the market data to find suitable values of θ_0 and θ_1.

Homework 2 (deadline Sept. 28, 2011) Consider IBM stock.

1. Assume Black-Scholes market model with constant coefficients. Use historical stock prices of one year to determine μ and σ. Comment on suitability of the model.

2. Consider Black-Scholes model with constant trend and non-constant volatility

$$\sigma(s,t) = |\theta_0 + \theta_1 \arctan(0.2 \cdot (s - 170))|.$$

   Find maximal likelihood estimates for the trend and volatility parameters.

3. Use observed values of 4 month put options (10 values) to determine implied volatilities for each observation. Comment on the validity of the BS model with constant volatility.

4. Find least squares estimate of the volatility and compute corresponding theoretical option prices. Plot the theoretical and observed option prices on a graph.

Computational Finance, Fall 2011
Computer Lab 5

The aim of the Lab is to learn to apply Monte-Carlo method for computing option prices.

Often it can be shown (or it is assumed in the case of certain market model) that the price of an European option can be expressed as the expected value

$$V = E[e^{-rT}p(S(T))],$$

where $S(T)$ is generated according to a certain stochastic differential equation. In such case we can compute V approximately by generating n values of the random variable $S(T)$: $S(T)_1, S(T)_2, \ldots, S(T)_n$ and computing the arithmetic average of the function under the expectation:

$$V \approx \bar{V}_n = \frac{e^{-rT}}{n}\sum_{i=1}^{n} p(S(T)_i).$$

From Central Limit Theorem it follows that

$$P\left(|V - \bar{V}_n| \leq \frac{-\Phi^{-1}(\frac{\alpha}{2})\mathrm{std}(Y)}{\sqrt{n}}\right) \approx 1 - \alpha$$

for large values of n. Here Φ is the cumulative distribution function of the standard normal distribution and $Y = e^{-rT}p(S(T))$

Exercise 1. If we assume that the Black-Scholes market model with constant volatility holds, then we have to generate $S(T)$ according to the stochastic differential equation

$$dS(t) = S(t)((r - D)\,dt + \sigma\,dB(t).$$

From the lecture we know that the solution of the equation is

$$S(t) = S(0)e^{(r-D-\frac{\sigma^2}{2})t+\sigma B(t)},$$

so $S(T) = S(0)e^{(r-D-\frac{\sigma^2}{2})T+\sigma X}$, where $X \sim N(0, \sqrt{T})$. Write a function MC1, that for given values of $S(0), r, D, \sigma, T, \alpha$ and n and for given payoff function p computes an approximate option value and its error estimate holding with the probability $(1 - \alpha)$ by Monte-Carlo method, using n generated stock prices. Verify the correctness of the function by Black-Scholes formulas for put and call options in the case $S(0) = 100$, $E = 97$, $\sigma = 0.4$, $T = 0.5$, $r = 0.02$, $D = 0.03$, $\alpha = 0.05$. How often the actual error is larger than the error estimate if you use MC1 100 times?

Very often it is not possible to generate $S(T)$ values that correspond exactly to the stochastic differential equation; then it is necessary to use some approximation methods. One such method is the Euler's method, where we divide the interval $[0, T]$ into m equal subintervals and use the approximations (int the case of Black-Scholes market model)

$$S_{i+1} = S_i(1 + (r - D)\,\Delta t + \sigma(S_i, t_i)X_i), i = 0, \ldots, m - 1,$$

where S_i are approximations to $S(i\Delta t)$, $\Delta t = \frac{T}{m}$ and $X_i \sim N(0, \sqrt{\Delta t})$. Instead of $S(T)$ we use S_m, thus we use Monte-Carlo method to compute an approximate value of $\hat{V}$, where

$$\hat{V}_m = E[e^{-rT}p(S_m)].$$

Since S_m for a fixed m does not have exactly the same distribution as $S(T)$, we have in general $\hat{V}_m \neq V$ and therefore Monte-Carlo method converges to a value that is different from the option price.

Exercise 2. Write a function MC2 that computes approximate option prices so that the stock prices are generated according to Euler's method. Determine how large is the difference between $\hat{V}_m$ and the correct option price in the case of European call option, using the same parameters as in the previous exercise for $m = 2, 4, 6, 8$. In order to see the difference, large enough value for n should be used (so that corresponding MC error is at least 5 times smaller than the computed difference).

It is known that if p is continuous and has bounded first derivative (ie it is Lipshitz continuous), then

$$|V - \hat{V}_m| = \frac{C}{m} + o(\frac{1}{m}),$$

where C is a constant that does not depend on m and $m \cdot o(\frac{1}{m}) \to 0$ as $m \to \infty$. Thus, if we use S_m instead of $S(T)$ and use Monte-Carlo method, then the total error is

$$|V - \bar{V}_{m,n}| \leq |V - \hat{V}_m| + |\hat{V}_m - \bar{V}_{m,n}| \leq \frac{C}{m} + o(\frac{1}{m}) + |\hat{V}_m - \bar{V}_{m,n}|.$$

The last term is the error of the Monte-Carlo method and can be estimated easily. So, in order to compute the option price V with a given error ε, we should choose large enough m (so that the term $\frac{C}{m}$ is small enough, for example less than $\frac{\varepsilon}{2}$) and then use MC method with large enough n so that the MC error estimate is also small enough (less than $\frac{\varepsilon}{2}$). There is one trouble: we do not know C. One possibility to estimate C is as follow:

1. Choose some values for m_0, n_0 for m and n. They should not be too small, but very large values take too much computation time.
2. Use MC method twice to compute $\bar{V}_{m_0,n_0}$ and $\bar{V}_{2m_0,n_0}$
3. Estimate the value of C: if m_0 is large enough, then

   $$|C| \leq \bar{C} = 2m_0 \cdot (|\bar{V}_{m_0,n_0} - \bar{V}_{2m_0,n_0}| + |\hat{V}_{m_0} - \bar{V}_{m_0,n_0}| + |\hat{V}_{2m_0} - \bar{V}_{2m_0,n_0}|).$$

   The last two terms are errors of the MC method.
4. Choose m_1 such that $\frac{\bar{C}}{m_1} \leq \frac{\varepsilon}{2}$ and n_1 such that MC error of $\bar{V}_{m_1,n_1}$ is less than $\frac{\varepsilon}{2}$. Then $\bar{V}_{m_1,n_1}$ is an approximation of the true option price which satisfies the desired error estimate.

In order to get reasonable estimates for the right value of m, the value of n_0 should be such that the difference of the values of the two first computations is larger than the sum of the corresponding MC errors.

Homework 3. (Deadline October 5, 2011). Assume that the Black-Scholes market model with the volatility

$$\sigma(s,t) = 0.4 + \frac{e^t}{\cdot(S-100)^2 + 10}$$

holds. Consider an option with duration $T = 0.5$ and payoff function

$$p(s) = \min(|s - 100|, 20).$$

Assume that $S(0) = 99, r = 0.03$ and $D = 0$. Find the price of the option with accuracy $\varepsilon = 0.04$ (using $\alpha = 0.05$ for estimating MC errors). Explain how you got the final answer.

Computational Finance, Fall 2011
Computer Lab 6

The aim of the Lab is to learn to use finite difference approximations of derivatives of a function and to derive finite difference methods for boundary value problems of ordinary differential equations.

Let us start from the numerical differentiation by finite difference approximations. Well-known finite difference approximations are as follows:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \qquad (\text{error} \leq ch^2),$$
$$f''(x) \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \qquad (\text{error} \leq ch^2).$$

Exercise 1. Write a function my_derivative that takes four arguments: a name of a function, the value of x, the value of h and the order of the derivative (1 or 2) to compute and computes the value of the specified derivative at x using the finite difference approximation given above. Using this function, verify the accuracy of error estimates in the case of several concrete functions and several values of x: compute the derivatives for $h = 1, \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{2^{10}}$ and find the quotient of the error to h^2. The quotients should approach a constant value.

Finite difference approximations of derivatives can be used for deriving numerical methods for solving differential equations.

Let us consider the following problem: find y such that

$$y''(x) = f(x),\ x \in [a, b] \tag{1}$$
$$y(a) = 1,\ y(b) = -1, \tag{2}$$

where f is a given functions. The procedure for deriving a finite difference approximation for the problem above consists of the following steps.

1. Choose a set of points at which we want to find approximate values of the unknown function. Usually this set of points is chosen by dividing the interval $[a, b]$ into n equal subintervals: we get points $x_i = a + ih,\ i = 0, \ldots, n$ where $h = \frac{b-a}{n}$. In order to determine the approximate values of the solution y we need $n + 1$ equations for $n + 1$ unknown values.
2. In order to determine the values for the $n + 1$ unknowns, we need $n + 1$ equations. We get those equations by using boundary conditions (two equations) and by writing down the differential equation at $n - 1$ points $\bar{x}_i,\ i = 1, \ldots, n - 1$ and then replacing the derivatives by approximations that use only the function values at points $x_i,\ i = 0, \ldots, n$. The points $\bar{x}_i,\ i = 1, \ldots, n - 1$ do not have to be the same as the points $x_i, i = 1, \ldots, n - 1$, but in the case of the current problem let us use $\bar{x}_i = x_i,\ i = 1, \ldots, n - 1$.

If $x_i,\ i = 0 \ldots, n$ are equally spaced (with stepsize $h = \frac{c}{n}$), then we can use the finite difference approximation discussed above:

$$y''(x_i) \approx \frac{y(x_{i-1}) - 2y(x_i) + y(x_{i+1})}{h^2} \qquad (\text{error} \leq ch^2).$$

Hence, writing out the differential equation at points $x_i,\ i = 1, \ldots, n-1$ and replacing derivatives with finite difference approximations we get a system of equations

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} = f(x_i),\ i = 1, \ldots, n-1.$$

We can view the boundary conditions as two additional equations

$$y_0 = 1,\ y_n = -1.$$

Since i-th equation contains only 3 unknowns y_{i-1}, y_i, y_{i+1} the resulting matrix of the system of equations has non-zero entries only on three diagonals, so we obtain three diagonal system of equations.

Exercise 2. One possibility to solve a linear system of equations is to use the Python command `linalg.solve(M,z)`, which returns the solution y of the system of equations $My = z$. Use this command to find the values of the approximate solution in the case $n = 10$, $a = 0$, $b = 1$, $f(x) = -24x^2$. Find the errors between the approximate solution and the exact solution $y(x) = 1 - 2x^4$.

Exercise 3. If the system matrix has only some nonzero diagonals then it is actually a waste of computer memory to store the full matrix. For solving such system it is actually possible to use the command `linalg.solve_banded((l,u),Dgs,z)`, where the rows of the matrix Dgs contain the diagonals of M starting from the highest one, l is the number of diagonals below the main diagonal and u is the number of diagonals above the main diagonal (in our case $l = u = 1$).NB! In the matrix Dgs for the diagonals that are above the main diagonal the first elements are actually not used (for the diagonal directly above the main diagonal the first element is not used, for the next diagonal two steps above the main diagonal the first two elements are not used etc; for diagonals below the main diagonal last elements are not used. Write a function that for a given n solves the problem of the previous exercise with the command `linalg.solve_banded((l,u),Dgs,z)` and returns the maximal error at the points $x_i,\ i = 1, \ldots, n-1$. Determine how many times the error is reduced if we increase the number of points two times.

Exercise 4. Consider the problem

$$y''(x) + y'(x) - x\,y(x) = \sin(x),\ x \in [a, b]$$
$$y(a) = 1,\ y'(b) = 0.$$

Derive a finite difference approximation for the problem and write a function that for given n returns the values of the approximate solution at $x_i, i = 0, \ldots, n$. Use approximation $y'(b) \approx \frac{y(b)-y(b-h)}{h}$ for approximating the second boundary condition.

Computational Finance, Fall 2011
Computer Lab 7

The aim of the Lab is to derive an explicit finite difference method for solving an initial value problem of the heat equation in a bounded domain.

Let us consider the following problem: find u such that

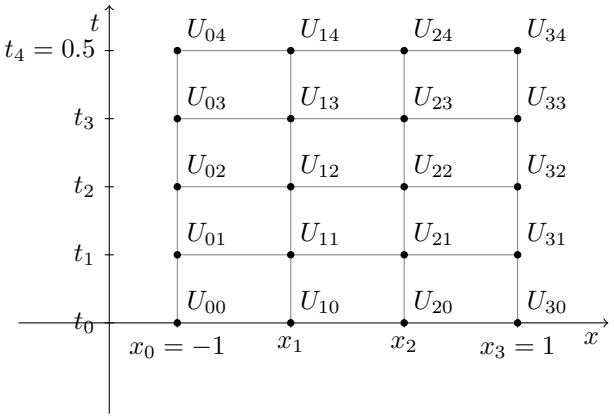
$$\frac{\partial u}{\partial t}(x,t) = \frac{1}{4}\frac{\partial^2 u}{\partial x^2}(x,t),\ x \in [-1,1],\ t \in (0,0.5] \tag{1}$$

$$u(-1,t) = 1, u(1,t) = 0,\ t \in (0,0.5] \tag{2}$$

$$u(x,0) = u_0(x),\ x \in [-1,1] \tag{3}$$

where u_0 is a given function. The procedure for deriving a finite difference approximation for the problem above consists of the following steps.

1. Choose a set of points at which we want to find approximate values of the unknown function. We define this set of points by dividing the interval $[-1,1]$ in x direction into n equal subintervals and the time interval $[0,0.5]$ into m subintervals: we get points (x_i,t_k), where $x_i = -1+i\frac{2}{n},\ i = 0,\dots,n$, $t_k = k\frac{0.5}{m}$. Since we know the values of the unknown function for $x=-1$, $x=1$ and for $t=0$, we have to determine approximate values $U_{ik} \approx u(x_i,t_k),\ i = 1,\dots,n-1,\ k = 1,\dots,m$, thus we have $m\cdot(n-1)$ unknowns (see the picture below for $n=3, m=4$).



2. In order to determine the values for $m\cdot(n-1)$ unknowns, we need $m\cdot(n-1)$ equations. We get those equations by writing down the differential equation at $m\cdot(n-1)$ points and then replacing the derivatives by approximations that use only the function values at points $(x_i,t_k),\ i = 0,\dots,n,\ k = 0,\dots,m$.

3. In order to get an explicit finite difference method we use the equation at the points $(x_i,t_k),\ i = 1,\dots,n-1,\ k = 0,\dots,m-1$ and use the approximations

$$\frac{\partial u}{\partial t}(x_i,t_k) \approx \frac{U_{i,k+1}-U_{ik}}{\Delta t}\ (\text{error} \le const.\Delta t),$$
$$\frac{\partial^2 u}{\partial x^2}(x_i,t_k) \approx \frac{U_{i-1,k}-2U_{ik}+U_{i+1,k}}{\Delta x^2}\ (\text{error} \le const.\Delta x^2).$$

Using the procedure outlined above, we get a system of equations of the form

$$U_{i,k+1} = a\,U_{i-1,k} + b\,U_{ik} + c\,U_{i+1,k},\ k = 0,\dots,m-1,\ i = 1,\dots,n-1,$$

where a, b and c are certain coefficients. Fortunately it is very easy to solve the system of equations: since the values of $U_{i0},\ i = 0, \ldots, n$ are known, we can just compute $U_{i1},\ i = 1, \ldots, n-1$ from the equations, after that we can compute U_{i2} etc. Since we do not have to solve any systems of equations but can just compute the values of the approximate solutions, the method is called explicit method.

Exercise 1. Write a function that for given values of m and n and for given function u_0 returns the values $U_{im},\ i = 0, \ldots, n$ of the approximate solution obtained by explicit finite difference method. Test the correctness of your function in the case $m = 100, n = 10$ and $u_0(x) = \sin(\pi x) + \frac{1-x}{2}$, when the exact solution is $u(x,t) = e^{-\pi^2 t/4}\sin(\pi x) + \frac{1-x}{2}$.

Exercise 2. The total error caused by replacing exact derivatives with finite difference approximations is $O(\Delta t + \Delta x^2)$, which usually implies that the error of the approximate solution is of the same order. This means, that if we increase m four times and n two times, then the total error should be reduced approximately four times. Verify the convergence rate by computing the errors in the settings of the previous exercise for $m = 4, 16, 64, 256$ and $n = 2, 4, 8, 16$.

Exercise 3. It turns out that explicit methods may be unstable for certain choices of parameters m and n. This means, that if m and n do not satisfy certain condition, the approximate solution may have arbitrarily large errors even when we let m and n to go to infinity. The sufficient condition of stability is that the coefficients a, b and c are all nonnegative. Repeat the computations of the previous exercise for $m = 2, 8, 32, 128$ and $n = 10, 20, 40, 80$ and compute the errors.

Homework 4. (Deadline October 19, 2011) Write a function that for given values of m and n and for given function u_0 returns the values $U_{im},\ i = 0, \ldots, n$ of the approximate solution of the problem

$$\frac{\partial u}{\partial t}(x,t) = (2+e^t)\frac{\partial^2 u}{\partial x^2}(x,t) + 3\,x\,u(x,t),\ x \in [-1,2],\ t \in (0,1.5] \tag{4}$$

$$u(-1,t) = 3, u(2,t) = -3t,\ t \in (0,1.5] \tag{5}$$

$$u(x,0) = x^2 - 2x,\ x \in [-1,2] \tag{6}$$

obtained by explicit finite difference method and also prints a warning, if the choices of m and n are such that the method may be unstable (if any of the coefficients a_{ik}, b_{ik}, c_{ik} is negative).

Computational Finance, Fall 2011
Computer Lab 8

If we consider an European option with exercise time T and payoff function p and assume the validity of Black-Scholes market model, then the option price at time t is given by $v(S(t),t) = u(\ln(S(t)),t)$, where u is the solution of the problem

$$\frac{\partial u}{\partial t}(x,t) + \alpha(x,t)\frac{\partial^2 u}{\partial x^2}(x,t) + \beta(x,t)\frac{\partial u}{\partial x}(x,t) - r\,u(x,t) = 0,\ x \in \mathbf{R}, 0 \le t < T \tag{1}$$

satisfying the final condition

$$u(x,T) = p(e^x),\ x \in \mathbf{R}.$$

Here

$$\alpha(x,t) = \frac{\sigma^2(e^x,t)}{2},$$
$$\beta(x,t) = r - D - \frac{\sigma^2(e^x,t)}{2}.$$

For solving the equation for u numerically, we introduce two boundaries x_{min} and x_{max} and specify boundary conditions $u(x_{min},t) = \phi_1(t),\ u(x_{max},t) = \phi_2(t)$ at those points. Next, we introduce the points $x_i = x_{min} + i\Delta x,\ i = 0,\dots,n$ and $t_k = k\Delta t,\ k = 0,\dots,m$ and denote by U_{ik} approximate values of $u(x_i,t_k)$. Here $\Delta x = \frac{x_{max}-x_{min}}{n}$ and $\Delta t = \frac{T}{m}$. In the case of the explicit finite difference method we compute the values U_{ik} as follows:

$$\begin{aligned}
U_{im} &= p(e^{x_i}),\ i = 0,\dots,n \\
U_{0,k-1} &= \phi_1(t_{k-1}),\ U_{n,k-1} = \phi_2(t_{k-1}),\ k = m, m-1,\dots,1, \\
U_{i,k-1} &= a_{ik}U_{i-1,k} + b_{ik}U_{ik} + c_{ik}U_{i+1,k},\ i = 1,\dots,n-1,\ k = m, m-1,\dots,1,
\end{aligned}$$

where

$$a_{ik} = \frac{\Delta t}{\Delta x^2}\left(\alpha_{ik} - \frac{\beta_{ik}}{2}\Delta x\right),$$
$$b_{ik} = 1 - 2\frac{\Delta t}{\Delta x^2}\alpha_{ik} - r\Delta t,$$
$$c_{ik} = \frac{\Delta t}{\Delta x^2}\left(\alpha_{ik} + \frac{\beta_{ik}}{2}\Delta x\right).$$

If σ is a constant, then the coefficients a, b and c are also constants and the numerical scheme simplifies to

$$U_{i,k-1} = a\,U_{i-1,k} + b\,U_{ik} + c\,U_{i+1,k},\ i = 1,\dots,n-1,\ k = m, m-1,\dots,1.$$

The stability condition is in this case $b \ge 0$.

Exercise 1. Write a function that for given values of n, $\rho > 1$, r, D, $S0$, T, σ and for given functions p, ϕ_1 and ϕ_2 takes m to be equal to the smallest number satisfying the stability constraint and returns the values $U_{i0},\ i = 0,\dots,n$ of the approximate solution (option prices) obtained by explicit finite difference method and the corresponding stock prices $S_i = e^{x_i}$ in the case $x_{min} = \ln\frac{S0}{\rho},\ x_{max} = \ln(\rho\,S0)$. Test the correctness of your code by comparing the results to the exact values obtained by Black-Scholes formula in the case $r = 0.03$, $\sigma = 0.5$, $D = 0.05$, $T = 0.5$, $E = 97$, $S0 = 100$, $p(s) = \max(s-E,0)$, $\phi_1(t) = p(e^{x_{min}}),\ \phi_2(t) = p(e^{x_{max}})$.

Practical Homework 5. (Deadline October 26, 2011) Let $r = 0.02$, $\sigma = 0.6$, $D = 0.03$, $T = 0.5$, $E = 99$, $S0 = 100$, $p(s) = \max(s-E,0)$. If we use the explicit method of previous exercise, then even if we let m and n go to infinity there is going to be a finite error between the exact option price

at $t = 0, S(0) = S0$ and the corresponding approximate value. This error is caused by introducing artificial boundaries x_{min} and x_{max} and the boundary conditions specified at those boundaries. Use the boundary conditions $\phi_1(t) = p(e^{x_{min}})$, $\phi_2(t) = p(e^{x_{max}})$ and determine the value of the resulting error for $\rho = 1.5, 2, 2.5$. In order to see the resulting error you should do several computations with fixed ρ and increasing values of n (assuming m is determined from the stability condition, n should be increased by multiplying it by 2 each time). Use the knowledge that for large enough n the part of the error depending on the choice of n behaves approximately like $\frac{const.}{n^2}$ (so the difference of the last two computations divided by 3 is an estimate of this part of the error for the last computation) for determining how far your last computation is from the limiting value.

Computational Finance, Fall 2011
Computer Lab 9

The aim of the lab is to learn to use the explicit finite difference method in the case of variable coefficients and to learn to use special solutions of the Black-Scholes equation for constructing boundary conditions.

When constructing boundary conditions ϕ_1 and ϕ_2 in the case of option pricing it is often a good idea to use the fact, that the solution of the Black-Scholes equation satisfying the final condition $v(s,T) = c_1 s + c_2$ is

$$v(s,t) = c_1 e^{-D(T-t)} s + c_2 e^{-r(T-t)}.$$

Hence, if the payoff function is a linear function starting from some value $s = s_1$, then for large values of s the solution is practically equal to the special solution corresponding to this linear function. And if the payoff function is linear for $s < s_2$, than for small values of s the solution is practically equal to the special solution corresponding to that linear function. This gives us a possibility to define boundary values so that they are not very different from actual option prices at those boundaries and hence to reduce significantly the error caused by introducing x_{min} and x_{max} in option pricing equations. For example, when finding the value of a call option price by solving untransformed equation, we have that for large values of s the payoff is $p(s) = s - E$. Since for transformed equation $u(x,t) = v(e^x, t)$, a suitable boundary condition for $x = x_{max}$ is

$$\phi_2(t) = e^{-D(T-t)} e^{x_{max}} - E e^{-r(T-t)}.$$

In this Lab we price an European option with $T = 0.5$ and payoff function

$$p(s) = \begin{cases} 90 - s, & s < 80, \\ \frac{(s-100)^2}{40}, & 80 \le s \le 120, \\ s - 110, & s > 120. \end{cases}$$

Additionally, we assume $r = 0.05$, $D = 0.02$, $S(0) = 100$.

Exercise 1. Derive suitable boundary conditions for both transformed and untransformed equation.

Exercise 2. Implement the explicit finite difference method described in the previous lab so that it works with nonconstant α and β. Using this implementation, write solver for untransformed Black-Scholes equation. Test your solver in the case $n = 20$, $\rho = 1.5$ and

$$\sigma(s,t) = 0.5 + e^{-t} \arctan(0.1\, s - 10).$$

Exercise 3. Try to compute the exact option price with as small error as possible. Compare the errors of the results of the previous exercise with the errors with those obtained by using the simple (constant) boundary conditions.

Computational Finance, Fall 2011
Computer Lab 10

The aim of the lab is to implement the basic implicit method for computing European option prices. For this we consider the problem

$$\begin{aligned}
&\frac{\partial u}{\partial t}(x,t) + \alpha(x,t)\frac{\partial^2 u}{\partial x^2}(x,t) + \beta(x,t)\frac{\partial u}{\partial x}(x,t) - r\,u(x,t) = 0,\ x \in (x_{min}, x_{max}), 0 \le t < T,\\
&u(x_{min},t) = \phi_1(t), 0 \le t < T,\\
&u(x_{max},t) = \phi_2(t), 0 \le t < T,\\
&u(x,T) = p(e^x),\ x \in (x_{min}, x_{max}).
\end{aligned}$$

We introduce the points $x_i = x_{min} + i\Delta x,\ i = 0,\dots,n$ and $t_k = k\Delta t,\ k = 0,\dots,m$ and denote by $U_{i,k}$ the approximate values of $u(x_i,t_k)$. Here $\Delta x = \frac{x_{max}-x_{min}}{n}$ and $\Delta t = \frac{T}{m}$. In the case of the basic implicit finite difference method we compute the values U_{ik} as follows:

$$\begin{aligned}
&U_{im} = p(e^{x_i}),\ i = 0,\dots,n\\
&U_{0k} = \phi_1(t_k),\ U_{nk} = \phi_2(t_k),\ k = m-1, m-2,\dots,0
\end{aligned}$$

and for determining the values of $U_{ik},\ i = 1,\dots,n-1,\ k = m-1,\dots,0$ we solve for each value of k (starting with $k = m-1$) a three-diagonal system

$$a_{ik}U_{i-1,k} + b_{ik}U_{ik} + c_{ik}U_{i+1,k} = U_{i,k+1},\ i = 1,\dots,n-1$$

for the unknown values of $U_{ik},\ i = 1,\dots,n-1$. Here

$$\begin{aligned}
a_{ik} &= -\frac{\alpha(x_i,\tau_k)\Delta t}{\Delta x^2} + \frac{\beta(x_i,t_k)\Delta t}{2\Delta x},\\
b_{ik} &= 1 + \frac{2\alpha(x_i,t_k)\Delta t}{\Delta x^2} + r\Delta t,\\
c_{ik} &= -\frac{\alpha(x_i,t_k)\Delta t}{\Delta x^2} - \frac{\beta(x_i,t_k)\Delta t}{2\Delta x}.
\end{aligned}$$

Exercise 1. Write a function that for given values of m, n, x_{min}, x_{max}, T,γ and for given functions u_0,α,β, ϕ_1 and ϕ_2 returns the values $U_{i0},\ i = 0,\dots,n$ of the approximate solution (option prices) obtained by the implicit finite difference method. Use this method for computing approximate values of the put option price by solving the transformed Black-Scholes equation in the case $r = 0.05$, $\sigma = 0.5$, $D = 0.05$, $T = 0.5$, $E = 100$, $S0 = 98$, $p(s) = \max(E-s,0)$, $\rho = 2$, $x_{min} = \ln\frac{S0}{\rho}$, $x_{max} = \ln(\rho\,S0)$, $n = 20$, $m = 100$. Use $\phi_1(t) = u_0(x_{min})$, $\phi_2(t) = u_0(x_{max})$

Practical Homework 6. (Deadline 09.11.2011) Find approximate option prices for the option considered in the previous exercise in the case $S0 = 10, 20, \dots, 200$ and their errors by solving the untransformed Black-Scholes equation with the basic implicit method in the case $x_{min} = 0$, $x_{max} = 200$, $\phi_1(t) = p(0)e^{-r(T-t)}$, $\phi_2(t) = 0$, $n = 40$, $m = 100$.

Computational Finance, Fall 2011
Computer Lab 11

If we want to compute an option price corresponding to the current price $S(0) = S0$ with a given accuracy ε, then a possible procedure is as follows:

1. Let $\rho = 2$, choose the boundaries $x_{min} = 0$, $x_{max} = \rho \cdot S0$ if you use a solver for the untransformed equation and $x_{min} = \ln(\frac{S0}{\rho})$, $x_{max} = \ln(\rho \cdot S0)$ if you use a solver of the transformed equation.
2. Solve the problem with a finite difference method and estimate the error by Runge's method, until the (estimated) finite difference discretization error is less than $\frac{\varepsilon}{4}$.
3. Increase the value of ρ two times (multiply it by 2), define corresponding x_{min} and x_{max} and solve the problem with the same method again until the (estimated) finite difference discretization error is less than $\frac{\varepsilon}{4}$. If the answer changes by more than $\frac{\varepsilon}{2}$ then repeat the step (multiply ρ by two and compare answers etc). Otherwise we have obtained the solution with the desired accuracy.

It is good idea to choose some starting values m_0 and n_0 of discretization parameters and to define a multiplier z of n_0 that is 1 for $\rho = 2$ and when ρ is multiplied by two, the factor is also multiplied by two if the untrasnformed equation is solver and increased by one, if the transformed equation is solved. So when starting computations with new value of ρ, the starting value of m should be taken m_0 and the starting value of n should be taken $z \cdot n_0$; this procedure keeps the stepsizes Δx the same for all values of ρ and makes it easier to compare the results obtained for different ρ. If this recommendation is followed, then in the procedure above we do not have to solve the equation for fixed ρ with accuracy $\varepsilon/4$, it is enough to solve it with the accuracy $\varepsilon/2$.

Practical Homework 7. (Deadline November 16, 2011) Find the call option price with maximal error 0.01 for current stock price $S0 = 99$ in the case $E = 100$, $r = 0.05$, $D = 0$, $T = 0.4$ and nonconstant volatility

$$\sigma(s,t) = 0.4 + \frac{0.2}{1 + 0.04(s-100)^2}.$$

Use Crank-Nicolson method for untransformed equation to obtain the answer.

Computational Finance, Fall 2011
Computer Lab 12

The aim of the lab is to learn to compute prices of American options.

If we have reduced the Black-Scholes option pricing equation to a problem of the form

$$\begin{aligned}
\frac{\partial u}{\partial t}(x,t) + \alpha(x,t)\frac{\partial^2 u}{\partial x^2}(x,t) + \beta(x,t)\frac{\partial u}{\partial x}(x,t) - r\,u(x,t),\ x \in (x_{min}, x_{max}), 0 \le t < T,\\
u(x_{min},t) = \phi_1(t), 0 \le t < T,\\
u(x_{max},t) = \phi_2(t), 0 \le t < T,\\
u(x,T) = u_0(x),\ x \in (x_{min}, x_{max}).
\end{aligned}$$

and have derived a finite difference method for computing option prices, then approximate prices of the corresponding American option can be computed by taking maximum of found approximate prices U_{ik} and the values of $u_0(x_i)$ at each timestep. The convergence rate of the resulting method is $O(\Delta t + \Delta x^2)$ even in the case of Crank-Nicolson method.

Exercise 1. Modify explicit, implicit and Crank-Nicolson methods for computing American options. Use the methods for computing approximate prices of the American put option in the case $r = 0.1$, $\sigma = 0.5$, $D = 0$, $T = 0.5$, $E = 100$, $S0 = 100$, $p(s) = \max(E - s, 0)$, $n = 20$, $m = 100$.

Exercise 2. Find the price of the American put option with maximal error 0.01 for current stock price $S0 = 100$ for the option considered in the previous exercise.

Exercise 3. Find the value $\frac{\partial v}{\partial s}(100, 0)$ of the price v of American put option with maximal error 0.0001 for the option considered in the previous exercise.

Computational Finance, Fall 2011
Computer Lab 13

The aim of the lab is to check how well an American options can be replicated by trading.

If the Black-Sholes market model holds, then by arguments used in the lecture we know that every European and American option can be replicated by a self-financing tranding strategy that uses the same sum of money as the option price as starting capital and holds at each time moment $\frac{\partial v}{\partial s}(S(t), t)$ shares of the stock. We assume that short selling (holding negative number of stocks) is possible and that stocks are infinitely divisible, so that any fraction of a stock can be held in a portfolio.

We shall simulate the trading by using the historical data of cisco stock for one year.

1. Read the data for cisco stock for the last year into python. We'll assume that the Black-Scholes market model holds with constant volatility. Estimate the volatility by using the first half of the data.
2. Consider an American put option with exercise price equal to the last stock price used for parameter estimation and exercise time half a year. Compute the price of the option (using $r = 0.03$) and it's derivative with respect to the stock price.
3. Let us set up a self-financing portfolio. It consists of an bank account and a stock holding; the initial stock holding is equal to te derivative found in the previous step and the bank account is initially option price minus the money under the stock (ie stock holding times the stock price).
4. For each day until expiry of the option we now simulate the change in the portfolio: we'll find the number of stocks we should have on that day (by finding the derivative of the option price at the current time and stock price) and modify the bank account by the interest earned and the money coming from the change of the stock holdings.
5. If the theory is good enough the total value of our portfolio should never go (much) below the value of the payoff function during the lifetime of the option but the minimal difference of the portfolio and the payoff function should be practically 0.

Practical homework 8 (deadline 30.11.2011) Perform computer simulations to determine how well can American Put options be replicated by trading once a day. For this each one of you will be given different stock. Download the historical prices for last 5 years. For each 3 months period in the last 4.5 years determine the historic volatility (assuming BS model with constant volatility) by using the data of 6 months prior the current period and then try to replicate 3 months American put option with exercise price equal to the stock price at the beginning of the period. Record for each period the smallest value of the difference between the portfolio and the payoff value. Analyze your findings.

Computational Finance, Fall 2011
Computer Lab 14

The aim of the lab is to learn to compute prices of Asian options.

Let us assume that the volatility σ in the Black-Scholes market model depends only on the current stock price S.

Let $v(s,I,t)$ be the function giving the price of an Asian option (depending on arithmetic average) with exercise time T and payoff $p(s,A_T)$; denote $u(s,I,\tau)=v(s,I,T-\tau)$. For finding approximate option prices we introduce artificial boundaries I_{max}, S_{max}, choose natural numbers n_s, n_I, m and look for approximate values of u at points (s_i, I_j, τ_k), where

$$s_i = i\Delta s = i\frac{S_{max}}{n_s},\ I_j = j\Delta I = j\frac{I_{max}}{n_I},\ \tau_k = k\Delta\tau = k\frac{T}{m}.$$

Denote those approximate values by U_{kij}. A finite difference approximation gives the following equations for the unknown values:

$$a_iU_{k,i-1,j}+b_iU_{kij}+c_iU_{k,i+1,j} = d_iU_{k-1,i-1,j+1}+e_iU_{k-1,i,j+1}+f_iU_{k-1,i+1,j+1}+g_i(U_{k,i,j+1}-U_{k-1,i,j}),$$

where $i=1,2,\dots,n_s-1,\ j=0,1,\dots,n_I-1,\ k=1,\dots,m$ and (using the notation $\rho=\frac{\Delta\tau}{\Delta s^2}$)

$$\begin{aligned}
a_i &= \frac{\rho}{4}(-s_i^2\sigma^2(s_i)+(r-D)s_i\Delta s),\\
b_i &= \frac{1}{2}\left(1+\rho s_i^2\sigma^2(s_i)+\frac{s_i\Delta\tau}{\Delta I}+r\Delta\tau\right),\\
c_i &= -\frac{\rho}{4}(s_i^2\sigma^2(s_i)+(r-D)s_i\Delta s),\\
d_i &= -a_i\\
e_i &= \frac{1}{2}\left(1-\rho s_i^2\sigma^2(s_i)+\frac{s_i\Delta\tau}{\Delta I}-r\Delta\tau\right),\\
f_i &= -c_i\\
g_i &= \frac{1}{2}\left(-1+\frac{s_i\Delta\tau}{\Delta I}\right).
\end{aligned}$$

The equations for the approximate values can be viewed as a three-diagonal system for finding the values of U_{kij}, if the values corresponding to the level $\tau=\tau_{k-1}$ and the values $U_{k,i,j+1},\ i=0,\dots,n_s$ have been found earlier.

The values of $U_{0,i,j}$ can be found from the initial condition:

$$U_{0,i,j} = p(s_i,\frac{I_j}{T}),\ i=0,\dots,n_s,\ j=0,\dots,n_I.$$

At the boundary $S=0$ we have the exact boundary condition $u(0,I,\tau)=p(0,\frac{I}{T})e^{-r\tau}$, hence

$$U_{k,0,j} = p(0,\frac{I_j}{T})e^{-r\tau_k},\ k=1,\dots,m.$$

In order to determine all values of U_{kij} uniquely, we have to specify boundary conditions for the boundary $I=I_{max}$ and the boundary $s=S_{max}$. Denote by $\phi_1(s,\tau)$ and $\phi_2(I,\tau)$ the functions describing the boundary conditions, then

$$U_{k,i,n_I} = \phi_1(s_i,\tau_k),\ U_{k,n_s,j} = \phi_2(I_j,\tau_k).$$

Exercise Let us consider an average price put option (payoff function $p(s, A_T) = \max(E - A_T, 0)$, where E is the exercise price specified in the option contract). Assume $r = 0.05$, $D = 0$, $\sigma = 0.5$, $T = 0.5$, $E = 100$, $S(0) = 95$. Define $I_{max} = E\,T$, $S_{max} = 190$, $\phi_1(s, \tau) = 0$,

$$\phi_2(I, \tau) = \max\{Ee^{-r\tau} + \frac{e^{-D\tau}}{(r - D)T}(e^{-(r-D)\tau} - 1)S_{max} - \frac{e^{-r\tau}}{T}I, 0\}.$$

Write a function, that for given values n_s, n_I, m finds an approximate option price at $t = 0$ using the finite difference method described above.

Let us discuss one possibility to specify suitable artificial boundary conditions ϕ_1 and ϕ_2. Recall that the functions of the form

$$v(s, I, t) = C_1 e^{-r(T-t)} + e^{-D(T-t)}\left(C_2 - C_3\frac{e^{-(r-D)(T-t)}}{r - D}\right) s + C_3\, e^{-r(T-t)} I$$

are solutions of the Asian option pricing PDE in the case $r \neq D$; if $r = D$ then corresponding special solutions are

$$v(s, I, t) = C_1 e^{-r(T-t)} + e^{-r(T-t)}\left(C_2 + C_3(T - t)\right) s + C_3\, e^{-r(T-t)} I.$$

Consider payoff functions of the form

$$p(s, a) = \max\{k_1 + k_2 s + k_3 a, 0\}.$$

Let v_{spec} be the special solution satisfying $v_{spec}(s, I, T) = k_1 + k_2 s + k_3 I/T$ then we define

$$\phi_1(s, \tau)) = \max\{v_{spec}(s, I_{max}, T - \tau), 0\}$$

and

$$\phi_2(I, \tau)) = \max\{v_{spec}(S_{max}, I, T - \tau), 0\}.$$

Practical Homework 9 (Deadline December 7, 2011) Let us consider an average price put option, an average price call option, an average strike put option and an average strike call options. Assume $r = 0.06$, $D = 0.06$, $\sigma = 0.5$, $T = 0.5$, $E = 100$, $S(0) = 100$. Define $I_{max} = ET$, $S_{max} = 200$. Write functions that for given values n_s, n_I, m find approximate option prices at $t = 0$ using the finite difference method described in this Lab.

Computational Finance, Fall 2011
Computer Lab 15

The aim of the lab is to start learning finite element methods.

Let us consider the following problem: find y such that

$$y''(x) + a(x)y'(x) + b(x)y(x) = f(x),\ x \in [0,1] \tag{1}$$
$$y(0) = y0,\ y(1) = y1, \tag{2}$$

where $y0, y1$ are given numbers and a, b, f are given functions. The procedure for deriving a finite element approximation for the problem above consists of the following steps.

1. Derive a weak form of the differential equations. For deriving a weak form we multiply the equation (**??**) by an arbitrary differentiable function ϕ such that $\phi(0) = \phi(1) = 0$ and integrate the equation. By integrating the term with the highest derivative by parts we get

   $$-\int_0^1 y'(x)\phi'(x)\,dx + \int_0^1 a(x)y'(x)\phi(x)\,dx + \int_0^1 b(x)y(x)\phi(x)\,dx = \int_0^1 f(x)\phi(x)\,dx.$$

   It is quite easy to convince oneself that if the integrated equation holds for every ϕ and y satisfies the boundary conditions (**??**), then y is the solution of the original problem.

2. In order to determine an approximate solution we look for y in the form of a linear combination of $n+1$ basis functions $\phi_j, j = 0, \ldots, n$:

   $$y(x) \approx \sum_{j=0}^{n} \xi_j \phi_j(x),$$

   and require that the boundary conditions hold and that the weak form holds in the cases $\phi = \phi_i,\ i = 1, \ldots, n-1$. This gives us $n+1$ equations for $n+1$ unknowns; by solving the system of equations we get an approximate solution.

   The method is called a finite element method if the basis functions ϕ_i are such that they are non-zero only on a small subinterval of $[0,1]$.

In this lab we define a collection of basis functions by introducing a grid $0 = x_0 < x_1 < x_2 < \ldots < x_n = 1$ and defining $h_i = x_{i+1} - x_i$ and

$$\phi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h_{i-1}}, & x \in [x_{i-1}, x_i), \\ \frac{x_{i+1} - x}{h_i}, & x \in [x_i, x_{i+1}], \\ 0 & \text{elsewhere.} \end{cases}$$

Exercise Find the values of the approximate solution in the case of uniform grid, $n = 10$, $a(x) = 0$, $b(x) = -1$, $f(x) = 0$, $y0 = 1$, $y1 = 0$ by the piecewise linear finite element method. Plot the graph of the error between approximate solution and the exact solution $y(x) = \frac{1}{1-e^2}(e^x - e^{2-x})$.

Let us now consider the following problem: find u such that

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2} + \beta \frac{\partial u}{\partial x} + \gamma u,\ x \in [x_{min}, x_{max}] \tag{3}$$
$$u(x,0) = u_0(x),\ x \in [x_{min}, x_{max}], \tag{4}$$
$$u(x_{min}, t) = \psi_1(t),\ u(x_{max}, t) = \psi_2(t),\ t \in [0, T]. \tag{5}$$

where $x_{min}, x_{max}, T, \alpha, \beta, \gamma$ are given numbers and u_0, ϕ_1, ϕ_2 are given functions. The procedure for deriving a finite element approximation for the problem above consists of the following steps.

1. Derive a weak form of the differential equations. For deriving a weak form we multiply the equation (**??**) by an arbitrary differentiable function ϕ of one variable such that $\phi(x_{min}) = \phi(x_{max}) = 0$ and integrate the equation over the x variable. By integrating the term with the highest derivative by parts we get

$$\int_{x_{min}}^{x_{max}} \frac{\partial u}{\partial t}(x,t)\phi(x)\,ds = -\int_{x_{min}}^{x_{max}} \alpha\frac{\partial u}{\partial x}(x,t)\phi'(x)\,dx + \int_{x_{min}}^{x_{max}} (\beta\frac{\partial u}{\partial x}(x,t) + \gamma u(x,t))\phi(x)\,dx$$

2. In order to determine an approximate solution we look for u in the form of a linear combination of $n+1$ basis functions $\phi_i, i = 0, \ldots, n$ with time-dependent coefficients:

$$u(x,t) \approx \sum_{i=0}^{n} \xi_i(t)\phi_i(x),$$

and require that the boundary conditions hold and that the weak form holds in the cases $\phi = \phi_i,\ i = 1, \ldots, n-1$. This gives us $n+1$ ordinary differential equations for $n+1$ unknown functions; by solving the system of equations we get an approximate solution.

The method is called a finite element method if the basis functions ϕ_i are such that they are non-zero only on a small subinterval of $[x_{min}, x_{max}]$.

We consider the same basis functions as before.

For solving the resulting system of ordinary differential equations there are many methods available. We'll use Euler's method by dividing the time interval into m equal subintervals and replacing the time derivative by approximation

$$\xi_i'(t_k) \approx \frac{\xi_i(t_{k+1}) - \xi_i(t_k)}{\Delta t}.$$

Homeworks (Deadline December 19, 2011) Derive equations for determining the approximate values of $\xi_i(t_k),\ k = 1, \ldots, m$. Implement the corresponding method (so that it works with any given grid) and use it to compute approximately the price of the put option in the case $T = 0.5$, $D = 0$, $\sigma = 0.5$, $r = 0.05$, $E = 100$, $S(0) = 95$, by solving the transformed Black-Scholes equation with $m = 3500$, $n = 20$, $x_{min} = \ln 50$, $x_{max} = \ln 200$ in the case of the grid $x_i = \frac{x_{min}+x_{max}}{2} - \frac{x_{max}-x_{min}}{2}\left(\frac{n-2i}{n}\right)^2,\ i = 0, 1, \ldots, \frac{n}{2}$; $x_i = \frac{x_{min}+x_{max}}{2} + \frac{x_{max}-x_{min}}{2}\left(\frac{2i-n}{n}\right)^2,\ i = \frac{n}{2}+1, \ldots, n$. Submit both the detailed derivation (This is Theoretical Homework 3, worth max 4 points) and the Python code (Practical Homework 10, max 3 points).

Computational Finance
Final Examination
January 14, 2010

Problem 1. Consider the partial differential equation for Asian option:

$$\frac{\partial v}{\partial t}(s,I,t)+\frac{\sigma s^2}{2}\frac{\partial^2 v}{\partial s^2}(s,I,t)+(r-D)s\frac{\partial v}{\partial s}(s,I,t)+s\frac{\partial v}{\partial I}(s,I,t)-rv(s,I,t)=0.$$

Let us look for a special solution of the form $v(s,I,t)=Iw(x,t)$, where $x=\frac{s}{I}$. Find the equation that must be satisfied by the function $w(x,t)$.

Problem 2. Derive an Explicit finite difference method for solving the problem

$$\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+(x+t)\frac{\partial^2 u}{\partial y^2},\ 1<x<4,\ 1<y<3,\ 0<t<0.5,$$

$$u(x,y,0)=u_0(x,y),\ 1<x<4,\ 1<y<3,$$

$$u(x,1,t)=u(x,3,t)=1,\ u(1,y,t)=u(4,y,t)=(y-2)^2,$$

showing also how the initial and boundary conditions are used.

Problem 3. Consider a barrier option which gives it's holder at $T=0.5$ the payment $(S(T)-100)^2/100$, if $S(t)<120$, $0\le t\le T$. If the stock price achieves the value 120 first time at $t=t_0$, then the holder of the option receives at $t=t_0$ the payment $4e^{-r(T-t_0)}$ (where r is the risk free interest rate). It is known that the price of the option at $t=0$ is given by $v(S(0),0)$, where $v(s,t)$ satisfies the Black-Scholes equation in the region $0\le s<120,\ t\in[0,0.5]$, the boundary condition

$$v(120,t)=4e^{-r(T-t)},0<t\le T$$

and the final condition

$$v(s,T)=(s-100)^2/100,0\le s\le 120.$$

Assume that $S(0)=94.7237$ and that the Black-Scholes market model holds with $\sigma(s,t)=0.5+0.1\sin(s\,t)$. Assume also that $r=0.1$ and $D=0$. Find the price of the option with the maximal error 0.01 (Hint: if u_1 is the price of the option at the $t=0$ corresponding to $S(0)=s_1$ and u_2 is the price of the option at the $t=0$ corresponding to $S(0)=s_2$, then the the price corresponding to $S(0)=s_0\in[s_1,s_2]$ is approximately given by $\frac{s_2-s_0}{s_2-s_1}u_1+\frac{s_0-s_1}{s_2-s_1}u_2$ with the error $O((s_2-s_1)^2)$). Partial credit can be obtained for solving the problem for $S(0)=90$.

Problem 4. Consider an option with the exercise date $T=0.5$ and the payoff function $p(s)=\max(10-|s-100|,0)$. Assume that the stock pays at $t=0.3$ a proportional dividend $0.05\,S(0.3)$ per share. Assume that the stock does not pay any continuously compounded dividends. Assume also that the stock price satisfies the Black-Scholes market model with the constant volatility $\sigma=0.5$ for $t\neq 0.3$ (it can be shown that the stock price is reduced by the amount of $0.05S(0.3)$ at $t=0.3$. Let $v_1(s,t)$ be the pricing function for the corresponding usual European option for $t\in[0.3,0.5]$ and let $v_2(s,t)$ be the pricing function of the European option with the exercise date $T_2=0.3$ and the payoff $p(s)=v_1((1-0.05)s,0.3)$. Then it can be shown that the price v of the European option is given by

$$v(s,t)=\begin{cases}v_1(s,t) & \text{for } t\in(0.3,0.4],\\ v_2(s,t) & \text{for } t\in[0,0.3].\end{cases}$$

Assume $r=0.06$ and $S(0)=100$. Write a function that computes an approximate price of the option at $t=0$ using the implicit finite difference method.

```
from scipy import *

#exercise 1
def f1(x):
    result=x*sin(x)
    return(result)
from pylab import plot,show
x=linspace(0,10,100)
plot(x,f1(x))
show()

#exercise 2
def midpoint(f,a,b,n):
    h=(b-a)/float(n) # to avoid wrong answer if a and b are integers, we change n to a
real (floating point) number
    i=arange(1,n+1)
    x=a+(i-1.0/2.0)*h #here again 1/2 would return 0, that gives incorrect answers
    result=h*sum(f(x))
    return(result)
print midpoint(sin,0,1,100)
print midpoint(f1,0,1,100)
#exercise 3
m=100
n=5
Y=zeros(shape=(m+1,n))
i=arange(0,m+1)
x=i/float(m) #again there is a need to avoid dividing integers
Y[:,0]=sin(4*pi*x)
for j in arange(1,n):
    Y[0,j]=Y[m,j]=0 #actually unnecessary, since the matrix Y was filled with zeros at
the time of definition
    i=arange(1,m) #since this vector does not change when j changes, it is more
efficient to define it before the for cycle starts
    Y[i,j]=(Y[i-1,j-1]+Y[i,j-1]+Y[i+1,j-1])/3
plot(x,Y[:,n-1]) #since the column nubers start from 0, the last column corresponds to
n-1
show()
```

```
#-------------------------------------------------------------------------------
# Name:        lab2.py
# Purpose:
#
# Author:      Raul Kangro
#
# Created:     07.09.2011
#-------------------------------------------------------------------------------
from scipy import *

#Exercise 1
from BSformulas import Put, Call
print Call(r=0.05,D=0.01,sigma=0.5,S=100,E=100,T=0.5)
print Put(r=0.05,D=0.01,sigma=0.5,S=100,E=100,T=0.5)

#Exercise 2
from pylab import plot,show
def BSgraph(S0,m,mu,sigma,T,n):
    S=empty(shape=(m+1,n))
    t=linspace(0,T,m+1)
    h=T/float(m) #step size in time
    S[0]=S0 #one index in a matrix gives a row
    for i in arange(1,m+1):
        S[i]=S[i-1]*(1+mu*h+sigma*sqrt(h)*randn(n))
    plot(t,S)
    show()
BSgraph(S0=100,m=500,mu=0.1,sigma=0.5,T=1,n=5)
```

```
#-------------------------------------------------------------------------------
# Name:        lab3.py
# Purpose:
#
# Author:      Raul Kangro
#
# Created:     14.09.2011
#-------------------------------------------------------------------------------
from scipy import *
#read in the data that is located in the 5th row
S=loadtxt("h:/finmat/2011sygis/amazon.csv",delimiter=",",skiprows=1,usecols=(4,))
#compute the number of observations
n=size(S)
#the observations are in wrong order (the newest is the first), so we want to reorder
#make the sequence of indeces starting from the largest
i=arange(n-1,-1,-1)
#form a new sequence by takend the values of S in the order given by i
newS=S[i]
#after checking that newS is correct, we can rename it to be S again
S=newS #actually, we could write S=S[i] to achieve the same thing

#want to compute the logarithms of the quotionts of the consequtive closing prices
#to compute all logarithms with one command, let us make a sequence of integers for
which the computation is possible
i=arange(0,n-1) #one less than the number of S values
x=log(S[i+1]/S[i])
dt=1.0/n
#compute the volatility and trend according to formulas derived in class
sigma=std(x)/sqrt(dt)
mu=mean(x)/dt+sigma**2/2
#these are market parameters, if we assume that
#the Black-Scholes model with constant mu and sigma are valid
print "sigma=", sigma, ", mu=", mu
#check if the assumption is satisfied
#if satisfied, then ui are values of normally distributed random variables

#Exercise 2: test the assumptions of BS model
from scipy import stats
#info about the tests
help(stats.shapiro)
help(stats.anderson)
#check the normality of the data
print stats.shapiro(x)
#the result means that the probability to get the value 0.965.. by computations
performed
#by this test in the case of normally distributed data is only 0.0000079..., so very
small
#Thus it is unlikely that the stock prices follow the BS model with constant
coefficients
print stats.anderson(x)
#the conclusions are the same: the computed value was 2.09, but the probability of
getting value larger than 1.075 is 0.01
#if the data corresponds to a normal distribution
# hence one should look for more complicated models

#esercise 3
def f1(theta):
    #according to our assumptions we can compute the terms appearing in the likelyhood
function starting from the second
    #observation (for the first one we can not compute the trend since it needs
previous stock price)
```

```
    #and ending by one observation before the last (since Y[i] needs the next stock
price)
    i=arange(1,n-1)
    #compute all Y values
    Y=(S[i+1]-S[i])/S[i]
    #and all mu values
    mu=theta[0]+theta[1]*(S[i]-S[i-1])/dt
    #and all volatilities (but those are the same)
    sigma=theta[2]
    #compute the value of the function
    return(sum((Y-mu*dt)**2/(2*sigma**2*dt)+log(sigma)))
from scipy import optimize
#reasonable starting values correspond to the constant coefficient case
#we get constant coefficients when theta[1]=0; theta[0] corresponds then to the
constant trend and theta[2] is the volatility
theta=optimize.fmin(f1,[mu,0,sigma])

print theta

#exercise 4: is the new model with nonconstant mu better?
#theta is now known vector
#compute the quantities that should be iid and normally distributed
i=arange(1,n-1)
Y=(S[i+1]-S[i])/S[i]
mu=theta[0]+theta[1]*(S[i]-S[i-1])/dt
sigma=theta[2]
u1=(Y-mu)/sigma
#now check the normality
print(stats.shapiro(u1))
print stats.anderson(u1)
#No, the more complicated model is not good enough, so one should continue the search
for a good model ...
```

```
#-------------------------------------------------------------------------------
# Name:        lab4.py
# Purpose:
#
# Author:      Raul Kangro
#
# Created:     21.09.2011
#-------------------------------------------------------------------------------
from scipy import *
#three month call option prices for Cisco on Sept. 21,2011
E=[14.0,15.0,16.0,17.0,18.0,19.0]
Prices=[2.89,2.14,1.45,0.91,0.52,0.27]

#assume BS model with constant volatility. Then prices can be computed by BS formulas
#load the BS call option pricing function
from BSformulas import Call

#data about the option
r=0.02
D=0
S=16.53
T=1.0/4

n=size(E)
sigmas=empty(n) #here we store the computed volatilities
from scipy import optimize #solving and minimization functions in Python
for i in arange(n):
    #for each strike price define a function that computes the difference
    #between theoretical price and observed price
    def f(sigma):
        return(Call(S=S,E=E[i],T=T,D=0,r=r,sigma=sigma)-Prices[i])
    sigmas[i]=optimize.fsolve(f,0.5) #optimize.fsolve finds value of argument of the
function which makes it equal to 0.
    #the second parameter 0.5 is the starting value for sigma, around which to look
for the solution

#produce the graph of implied volatilities versus exercise prices
from pylab import plot,show
plot(E,sigmas)
show()
#exercise 2
def f2(sigma):
    result=0 #stores the current value of the sum of squared values
    for i in arange(n): #for each value of i we add to the result the squared error
for the strike price E[i]
        result=result+(Call(S=S,E=E[i],T=T,D=0,r=r,sigma=sigma)-Prices[i])**2
    return(1.0/2*result) #1.0/2 is actually not needed. I included it to make the
function equal to the one specified on handout
print optimize.fmin(f2,0.34)
sigma=optimize.fmin(f2,0.34)

#compute the differences for the best value of sigma
differences=empty(n)
for i in arange(n):
        differences[i]=(Call(S=S,E=E[i],T=T,D=0,r=r,sigma=sigma)-Prices[i])
print differences
#the largest one corresponds to E=15


#exercise 3
```

```
#import the pricing function for the more complicated model
from lab4solver import lab4solver
def f3(theta):
    result=0
    for i in arange(n):
        result=result+(lab4solver(theta,E[i],S)-Prices[i])**2
    return(1.0/2*result)
print optimize.fmin(f3,[0.33,0])
differences=empty(n)
for i in arange(n):
        differences[i]=(lab4solver([0.28460728,-0.0718296],E[i],S)-Prices[i])
print(differences)
#the errors are much smaller, so the more complicated market model with stock price
dependent volatility fits
#better the market data.
```

```
#-------------------------------------------------------------------------------
# Name:        lab5.py
# Purpose:     Sample solutions of lab exercises
#
# Author:      Raul Kangro
#
# Created:     28.09.2011
#-------------------------------------------------------------------------------
from scipy import *
from scipy import stats
InvPhi=stats.norm.ppf #computes the values of the inverse function of the
#cumulative distribution functin of the normal distribution

#data for the exercises
S0=100
E=97
sigma=0.4
T=0.5
r=0.02
D=0.03
alpha=0.05
#exercise 1
def MC1(n,p): #n is the number of generated stock prices, p is the name of the payoff
function that should be used for the current option
    S=S0*exp((r-D-sigma**2/2.0)*T+sigma*sqrt(T)*randn(n)) #generate N stock prices
according to the exact formula
    Y=exp(-r*T)*p(S) #compute the discounted payoff values
    price=mean(Y) #appriximate price is the average of the discounted payoff values
    error=-InvPhi(alpha/2.0)*std(Y)/sqrt(n) #MC error estimate, valid with probability
1-alpha
    return([price,error])
def p_call(s): #the payoff funciton of the call option
    return(maximum(s-E,0))
def p_put(s): #the payoff function of the put option
    return(maximum(E-s,0))
from BSformulas import Call #if sigma is a constant, then the exact price of the Call
and Put options are known. We need the exact price of the call option
exact=Call(S=S0,E=E,T=T,r=r,sigma=sigma,D=D)
#try out MC1
n=100000
V=MC1(n,p_call) #V is a pair of numbers, the first one is the approximate price of the
call option (since we use p_call as the payoff function), the second is the error
estimate.
print "exact price=",exact,"MC price =",V[0],"error estimate=",V[1]

#The error estimate of Monte-Carlo estimate is such that with probability 1-alpha we
get an approximate answer with actual error less than the estimate
#so if alpha=0.05 and we use MC method many times, then in average 1 time in 20 the
actual error is larger than the estimate
#Let us test this property: compute the price of the Call option 100 times and find
the actual error
#Check, how many times the actual error is larger than the estimate
for i in arange(100):
    V=MC1(n,p_call)
    actual_error=abs(exact-V[0])
    if(actual_error>V[1]):
        print i,actual_error,V[1]
#by running this code many times, you should see 5 lines of printout in the average
```

```
#exercise 2
#Use Euler's method for computing the approximate values of S(T)
def MC2(n,p,m):
    S=empty(shape=(m+1,n)) #define a matrix to store the trajectories of the stock
prices
    h=T/float(m) #step size in time
    S[0]=S0 #one index in a matrix gives a row
    mu=r-D
    for i in arange(1,m+1):
        S[i]=S[i-1]*(1+mu*h+sigma*sqrt(h)*randn(n))
    Y=exp(-r*T)*p(S[m]) #the final stock prices are in the last row with index m; we
use those values in the payoff function
    price=mean(Y)
    error=-InvPhi(alpha/2.0)*std(Y)/sqrt(n)
    return([price,error])
#we want to see how quickly the discretization error (the error that depends on the
number of time steps) goes to zero
#for this n has to be large enough so that the MC error is significantly smaller than
the discretization error
n=5000000
print MC2(n=n,m=2,p=p_call)
error_m=empty(4)
V=MC2(n=n,m=2,p=p_call)
error_m[0]=exact-V[0]
V=MC2(n=n,m=4,p=p_call)
error_m[1]=exact-V[0]
V=MC2(n=n,m=6,p=p_call)
error_m[2]=exact-V[0]
V=MC2(n=n,m=8,p=p_call)
error_m[3]=exact-V[0]
print error_m
#If the discretization error behaves like c/m, then it should get 2 times smaller
whenever m is increased 2 times. So
#error_m[1] corresponding to m=4 should be approximately half of the error_m[0] that
corresponds to m=2
#and #error_m[3] corresponding to m=8 should be approximately half of the error_m[1]
that corresponds to m=4


#more efficient version of the Euler's method: Since we need just the final values of
the stock prices, we do not store the matrix
#we just keep in the vector S the current values of the stock prices of all
trajectories
def MC2_2(n,p,m):
    h=T/float(m) #step size in time
    S=S0 #all trajectories start from S0
    mu=r-D
    for i in arange(1,m+1):
        S=S*(1+mu*h+sigma*sqrt(h)*randn(n)) #tha values corresponding to t[i] are
computed from the values of the previous time moment; results are stored in the same
vector S
    #now, after the cycle, the vector S contains the final values of the trajectories
    Y=exp(-r*T)*p(S)
    price=mean(Y)
    error=-InvPhi(alpha/2.0)*std(Y)/sqrt(n)
    return([price,error])
```

```
#-------------------------------------------------------------------------------
# Name:        Lab6.py
# Purpose:
#
# Author:      Raul Kangro
#
# Created:     05.10.2011
#-------------------------------------------------------------------------------
from scipy import *
def my_derivative(f,x,h,order):
    if order==1:
        return((f(x+h)-f(x-h))/(2.0*h))
    elif order==2:
        return((f(x+h)-2*f(x)+f(x-h))/h**2)
    else:
        print("order should be 1 or 2")
print my_derivative(sin,0,0.1,1)
def g(x):
    return(x*exp(2*x))
my_derivative(g,1,0.1,1)
h=2.0**(-arange(10))
x=0
results=my_derivative(sin,x,h,1)
exact=cos(x)
errors=exact-results
errors/h**2
x=0.5
results=my_derivative(sin,x,h,1)
exact=cos(x)
errors=exact-results
errors/h**2
x=0
results=my_derivative(exp,x,h,2)
exact=exp(x)
errors=exact-results
errors/h**2

#exercise 2
def f(x):
    return(-24*x**2)
a=0
b=1
from scipy import linalg
def solve(n,a,b):
    h=(b-a)/float(n)
    M=zeros(shape=(n+1,n+1))
    F=zeros(n+1)
    x=linspace(a,b,n+1)
    M[0,0]=1
    F[0]=1
    for i in arange(1,n):
        M[i,i-1]=1/h**2 #below the main diagonal
        M[i,i]=-2/h**2 #on the main diagonal
        M[i,i+1]=1/h**2 #above the main diagonal
        F[i]=f(x[i])# right hand side
    M[n,n]=1
    F[n]=-1
    y=linalg.solve(M,F)
    return(y)
n=10
```

```
#from pylab import plot,show
#plot(linspace(a,b,n+1),solve(n,a,b))
#show()
def y(x):
    return(1-2*x**4)
approximate=solve(n,a,b)
exact=y(linspace(a,b,n+1))
print exact-approximate
n=100
approximate=solve(n,a,b)
exact=y(linspace(a,b,n+1))
print max(abs(exact-approximate))

#Exercise 3: using special solver
def solve2(n,a,b):
    h=(b-a)/float(n)
    Dgs=zeros(shape=(3,n+1)) #three diagonals
    F=zeros(n+1)
    x=linspace(a,b,n+1)
    #M[0,0]=1
    Dgs[1,0]=1 #the main diagonal
    F[0]=1
    for i in arange(1,n):
        #M[i,i-1]=1/h**2 #below the main diagonal
        Dgs[2,i-1]=1/h**2
        #M[i,i]=-2/h**2 #on the main diagonal
        Dgs[1,i]=-2/h**2
        #M[i,i+1]=1/h**2 #above the main diagonal
        Dgs[0,i+1]=1/h**2
        F[i]=f(x[i])# right hand side
    #M[n,n]=1
    Dgs[1,n]=1
    F[n]=-1
    y=linalg.solve_banded((1,1),Dgs,F)
    return(y)
#exercise 4
def solve4(n,a,b):
    h=(b-a)/float(n)
    Dgs=zeros(shape=(3,n+1)) #three diagonals
    F=zeros(n+1)
    x=linspace(a,b,n+1)
    Dgs[1,0]=1 #the main diagonal
    F[0]=1
    i=arange(1,n) #instead of using for cycle, we can compute the elements in parallel
    Dgs[2,i-1]=1/h**2-1/(2*h)
    #M[i,i]=-2/h**2 #on the main diagonal
    Dgs[1,i]=-2/h**2-x[i]
    #M[i,i+1]=1/h**2 #above the main diagonal
    Dgs[0,i+1]=1/h**2+1/(2*h)
    F[i]=sin(x[i])#The right hand side
    #M[n,n]=1
    Dgs[1,n]=1/h #the boundary condition at x=b is (y[n]-y[n-1])/h=0
    Dgs[2,n-1]=-1/h  #the boundary condition at x=b
    F[n]=0
    y=linalg.solve_banded((1,1),Dgs,F)
    return(y)
n=10
print solve4(n,a,b)
##as we see, the code works and gives some reasonably looking answers
## but how do we know it works correctly?
```

```
##in order to verify the code it is usually a good idea to try to find a similar
problem for which the
##the exact solution is known
##We can construct a problem starting from the solution:
##pick a function that satisfies boundary conditions, for example y(x)=(x-1)**2
## if we substitute the solution to the left hand side of the equation we get
## 2+2*(x-1)-x*(x-1)**2
## So we can test our code so that we change sin(x) to 2-2*(x-1)-x*(x-1)**2 in the code
## and compare the results to the exact answer y(x)=(x-1)**2
```

```
#-------------------------------------------------------------------------------
# Name:        lab7.py
# Purpose:
#
# Author:      Raul Kangro
#
# Created:     12.10.2011
#-------------------------------------------------------------------------------
from scipy import *
def u0(x):
    return(sin(pi*x)+(1-x)/2.0)
def Ex1(m,n,u0):
    xmin=-1.0
    xmax=1.0
    x=linspace(xmin,xmax,n+1)
    T=0.5
    dx=(xmax-xmin)/n
    dt=T/m
    U=zeros(shape=(n+1,m+1))
    #use initial condition
    U[:,0]=u0(x)
    i=arange(1,n)
    a=dt/(4.0*dx**2)
    b=1-dt/(2.0*dx**2)
    c=a
    for k in arange(m):
        #use boundary conditions
        U[0,k+1]=1
        U[n,k+1]=0
        #use the numerical method for other values
        U[i,k+1]=a*U[i-1,k]+b*U[i,k]+c*U[i+1,k]
    return(U[:,m])
m=100
n=10
T=0.5
approximate=Ex1(m=100,n=10,u0=u0)
x=linspace(-1,1,n+1)
exact=exp(-pi**2*T/4)*sin(pi*x)+(1-x)/2.0
print exact-approximate
#exercise 2
for i in arange(4):
    n=2**(i+1)
    m=n**2
    approximate=Ex1(m=m,n=n,u0=u0)
    x=linspace(-1,1,n+1)
    exact=exp(-pi**2*T/4)*sin(pi*x)+(1-x)/2.0
    print "n=",n,"m=",m,"error=",max(abs(exact-approximate))
#we see that the error is reduced approximately 4 times each time we multiply
#n by 2 and m by 4 (except after the first computation). This is consistent with
#the error estimate error<=const.(dt+dx^2)

#exercise 3
m=2
n=10
for i in arange(4):
    approximate=Ex1(m=m,n=n,u0=u0)
    x=linspace(-1,1,n+1)
    exact=exp(-pi**2*T/4)*sin(pi*x)+(1-x)/2.0
    print "n=",n,"m=",m,"error=",max(abs(exact-approximate))
    n=n*2
```

```
    m=m*4
#with different starting values for m and n we see a completely different behaviour:
#the errors are increasing very rapidly when we increase m and n
#the reason is that m and n have to satisfy certain stability relations in order for
#the explicit method to give reasonable answer, and this condition is violated in this
case
```

```
#-------------------------------------------------------------------------------
# Name:        lab8.py
# Purpose:
#
# Author:      Raul Kangro
#
# Created:     12.10.2011
#-------------------------------------------------------------------------------
from scipy import *
def p_call(s):
    return(maximum(s-E,0))
def phi1_call(t,xmin):
    return(p_call(exp(xmin)))
def phi2_call(t,xmax):
    return(p_call(exp(xmax)))
def explicit(n,rho,r,D,S0,T,sigma,phi1,phi2,p):
    xmin=log(S0/float(rho))
    xmax=log(S0*rho)
    dx=(xmax-xmin)/n
    m=ceil(T*(sigma**2/dx**2+r))
    x=linspace(xmin,xmax,n+1)
    dt=T/float(m)
    U=zeros(shape=(n+1,m+1))
    alpha=sigma**2/2
    beta=r-D-alpha
    a=dt/dx**2*(alpha-beta/2*dx)
    b=1-2*dt/dx**2*alpha-r*dt
    c=dt/dx**2*(alpha+beta/2*dx)
    U[:,m]=p(exp(x)) #final condition
    t=linspace(0,T,m+1)
    i=arange(1,n)
    for k in arange(m,0,-1):
        #boundary conditions
        U[0,k-1]=phi1(t[k-1],xmin)
        U[n,k-1]=phi2(t[k-1],xmax)
        U[i,k-1]=a*U[i-1,k]+b*U[i,k]+c*U[i+1,k]
    return([U[:,0],exp(x)])
#data
E=97
T=0.5
S0=100
r=0.03
D=0.05
rho=3
sigma=0.5
n=100
answer=explicit(n,rho,r,D,S0,T,sigma,phi1_call,phi2_call,p_call)
from BSformulas import Call
approximate=answer[0]
S=answer[1]
exact=Call(S,E,T,r,sigma,D)
print approximate - exact
#price corresponding to S0:
print approximate[n/2]
#error or the approximate price
print "n=", n,"rho=",rho,"error=", approximate[n/2]-exact[n/2]
```

```
from scipy import *


#solvers in the case sigma=sigma(s)
#then alpha and beta are functions of x only
def explicit1(n,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r):
    x=linspace(xmin,xmax,n+1)
    dx=(xmax-xmin)/float(n)
    #compute m from the stability constraint
    m=int(amax(T*(2*alpha(x)/dx**2+r)))+1
    print(m)
    dt=T/float(m)
    t=linspace(0,T,m+1)
    #to save memory use only one row of the matrix U
    U=zeros(n+1)
    #final condition
    U[:]=u0(x)
    #coefficients a, b, c do not depend on k (time index)
    i=arange(1,n)
    a_ki=dt/dx**2*(alpha(x[i])-beta(x[i])/2*dx)
    b_ki=1-2*dt/dx**2*alpha(x[i])-r*dt
    c_ki=dt/dx**2*(alpha(x[i])+beta(x[i])/2*dx)
    for k in arange(m,0,-1):
        U[i]=a_ki*U[i-1]+b_ki*U[i]+c_ki*U[i+1]
        #left boundary
        U[0]=phi1(t[k-1])
        U[n]=phi2(t[k-1])
    return(U)
def transformedBSsolver1(n,r,D,S0,sigma,rho,T):
    def p(s):
        #return((90-s)*(s<80)+(s-100)**2/40.0*((s>=80)&(s<=120))+(s-110)*(s>120))
        return(maximum(s-100,0)) #simple call option
    #boundary conditions depend on payoff!
    xmin=log(S0/float(rho))
    xmax=log(S0*rho)
    def phi1(t):
        #return(-exp(xmin)*exp(-D*t)+90*exp(-r*t))
        return(0) #for call option
    def phi2(t):
        #return(exp(xmax)*exp(-D*t)-110*exp(-r*t))
        return(exp(xmax)*exp(-D*(T-t))-100*exp(-r*(T-t))) #for call option
    def alpha(x):
        return(sigma(exp(x))**2/2.0)
    def beta(x):
        return(r-D-sigma(exp(x))**2/2.0)
    def u0(x):
        return(p(exp(x)))
    #return the value for S0. assume n is even
    return(explicit1(n,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r)[n/2])

# try the code in the case of constant volatility
def sigma(s):
    return(0.5*ones(size(s)))
r=0.05
D=0.02
S0=100
rho=2.5
n=40
T=0.5
print(transformedBSsolver1(n,r,D,S0,sigma,rho,T))
```

```
from BSformulas import Call
print Call(S0,100,T,r,0.5,D)


############################################################
#volatility depends on time (and so do alpha, beta)
def explicit2(n,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r):
    x=linspace(xmin,xmax,n+1)
    dx=(xmax-xmin)/float(n)
    #compute m from the stability constraint
    #stability constraint is satisfied if m>T*(2*alpha(x[i],t)/dx**2r) for all i and
0<=t<=T
    #can not compute it for all t, therefore take only some t values
    m0=20
    t=linspace(0,T,m0+1)
    m=0
    for k in arange(m0+1):
      m=max(m,int(amax(T*(2*alpha(x,t[k])/dx**2+r)))+1)
    print(m)
    #to be safe, add something
    m=m+10
    dt=T/float(m)
    t=linspace(0,T,m+1)
    #to save memory use only one row of the matrix U
    U=zeros(n+1)
    #initial condition
    U[:]=u0(x)
    i=arange(1,n)
    for k in arange(m,0,-1):
        a_ki=dt/dx**2*(alpha(x[i],t[k])-beta(x[i],t[k])/2*dx)
        b_ki=1-2*dt/dx**2*alpha(x[i],t[k])-r*dt
        c_ki=dt/dx**2*(alpha(x[i],t[k])+beta(x[i],t[k])/2*dx)
        U[i]=a_ki*U[i-1]+b_ki*U[i]+c_ki*U[i+1]
        #left boundary
        U[0]=phi1(t[k-1])
        U[n]=phi2(t[k-1])
    return(U)
```

```
from scipy import *
from scipy import linalg

def implicit(n,m,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r):
    x=linspace(xmin,xmax,n+1)
    dx=(xmax-xmin)/float(n)
    dt=T/float(m)
    t=linspace(0,T,m+1)
    #to save memory use only one row of the matrix U
    U=zeros(n+1)
    #initial condition
    U[:]=u0(x)
    i=arange(1,n)
    i1=arange(n-1)
    i2=arange(n-2)
    M=zeros(shape=(n-1,n-1))
    F=zeros(n-1)
    for k in arange(1,m+1):
        a_ki=-(dt/dx**2*(alpha(x[i],t[k])-beta(x[i],t[k])/2*dx))
        b_ki=1+2*dt/dx**2*alpha(x[i],t[k])+r*dt
        c_ki=-(dt/dx**2*(alpha(x[i],t[k])+beta(x[i],t[k])/2*dx))
        M[i1,i1]=b_ki
        M[i2,i2+1]=c_ki[i2]
        M[i2+1,i2]=a_ki[i2+1]
        F[i1]=U[i]
        #boundary values
        U[0]=phi1(t[k])
        U[n]=phi2(t[k])
        #modify right hand side
        F[0]=F[0]-a_ki[0]*U[0]
        F[n-2]=F[n-2]-c_ki[n-2]*U[n]
        U[i]=linalg.solve(M,F)
    return(U)
def transformedBSsolver_implicit(n,m,r,D,S0,sigma,rho,T):
    def p(s):
        #return((90-s)*(s<80)+(s-100)**2/40.0*((s>=80)&(s<=120))+(s-110)*(s>120))
        return(maximum(100-s,0)) #simple put option
    #boundary conditions depend on payoff!
    xmin=log(S0/float(rho))
    xmax=log(S0*rho)
    def u0(x):
        return(p(exp(x)))
    def phi1(t):
        #return(-exp(xmin)*exp(-D*(T-t))+90*exp(-r*(T-t))
        return(u0(xmin)) #simple boundary condition
    def phi2(t):
        #return(exp(xmax)*exp(-D*(T-t))-110*exp(-r*(T-t)))
        #return(exp(xmax)*exp(-D*(T-t))-100*exp(-r*(T-t))) #for call option
        return(u0(xmax)) #simple boundary condition
    def alpha(x,t):
        return(sigma(exp(x),t)**2/2.0)
    def beta(x,t):
        return(r-D-sigma(exp(x),t)**2/2.0)

    #return the value for S0. assume n is even
    return(implicit(n,m,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r)[n/2])
```

```
from scipy import *
from scipy import linalg

def implicit(n,m,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r):
    x=linspace(xmin,xmax,n+1)
    dx=(xmax-xmin)/float(n)
    dt=T/float(m)
    t=linspace(0,T,m+1)
    #to save memory use only one row of the matrix U
    U=zeros(n+1)
    #initial condition
    U[:]=u0(x)
    i=arange(1,n)
    i1=arange(n-1)
    i2=arange(n-2)
    M=zeros(shape=(n-1,n-1))
    F=zeros(n-1)
    for k in arange(1,m+1):
        a_ki=-(dt/dx**2*(alpha(x[i],t[k])-beta(x[i],t[k])/2*dx))
        b_ki=1+2*dt/dx**2*alpha(x[i],t[k])+r*dt
        c_ki=-(dt/dx**2*(alpha(x[i],t[k])+beta(x[i],t[k])/2*dx))
        M[i1,i1]=b_ki
        M[i2,i2+1]=c_ki[i2]
        M[i2+1,i2]=a_ki[i2+1]
        F[i1]=U[i]
        #boundary values
        U[0]=phi1(t[k])
        U[n]=phi2(t[k])
        #modify right hand side
        F[0]=F[0]-a_ki[0]*U[0]
        F[n-2]=F[n-2]-c_ki[n-2]*U[n]
        U[i]=linalg.solve(M,F)
    return(U)
def transformedBSsolver_implicit(n,m,r,D,S0,sigma,rho,T):
    def p(s):
        #return((90-s)*(s<80)+(s-100)**2/40.0*((s>=80)&(s<=120))+(s-110)*(s>120))
        return(maximum(s-100,0)) #simple call option
    #boundary conditions depend on payoff!
    xmin=log(S0/float(rho))
    xmax=log(S0*rho)
    def u0(x):
        return(p(exp(x)))
    def phi1(t):
        return(u0(xmin)) #simple boundary condition
    def phi2(t):
        return(u0(xmax)) #simple boundary condition
    def alpha(x,t):
        return(sigma(exp(x),t)**2/2.0)
    def beta(x,t):
        return(r-D-sigma(exp(x),t)**2/2.0)

    #return the value for S0. assume n is even
    return(implicit(n,m,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r)[n/2])

def sigma(s,t):
    return(0.4+0.2/(1+0.04*(s-100)**2))
r=0.05
D=0.00
S0=99
rho=2
```

```
n=20
m=10
T=0.4
epsilon=0.01

#rho=2
V1=transformedBSsolver_implicit(n,m,r,D,S0,sigma,rho,T)
m=m*4
n=n*2
V2=transformedBSsolver_implicit(n,m,r,D,S0,sigma,rho,T)
error=abs(V1-V2)/3
```

```
from scipy import *
from scipy import linalg

def BS_implicit_solver_am(m,n,xmin,xmax,T,r,u0,alpha,beta,phi1,phi2):
    dx=(xmax-xmin)/float(n)
    dt=T/float(m)
    x=linspace(xmin,xmax,n+1)
    t=linspace(0,T,m+1)
    U0=u0(x)
    U=U0.copy()
    diagonals=zeros(shape=(3,n-1))
    for k in range(1,m+1):
        F=U[1:n]
        U[0]=phi1(t[k],xmin)
        U[n]=phi2(t[k],xmax)
        a=-alpha(x[1:n],t[k])*dt/dx**2+beta(x[1:n],t[k])*dt/(2*dx)
        b=1+2*alpha(x[1:n],t[k])*dt/dx**2+r*dt
        c=-alpha(x[1:n],t[k])*dt/dx**2-beta(x[1:n],t[k])*dt/(2*dx)
        diagonals[0,1:]=c[:-1]
        diagonals[1]=b
        diagonals[2,:-1]=a[1:]
        F[0]=F[0]-a[0]*U[0]
        F[n-2]=F[n-2]-c[n-2]*U[n]
        U[1:n]=linalg.solve_banded((1,1),diagonals,F)
        U=maximum(U,U0)
    return U
def BS_CN_solver_am(m,n,xmin,xmax,T,r,u0,alpha,beta,phi1,phi2):
    dx=(xmax-xmin)/float(n)
    dt=T/float(m)
    x=linspace(xmin,xmax,n+1)
    t=linspace(0,T,m+1)
    U0=u0(x)
    U=U0.copy()
    diagonals=zeros(shape=(3,n-1))
    for k in range(1,m+1):
        a=-alpha(x[1:n],t[k])*dt/dx**2/2+beta(x[1:n],t[k])*dt/(4*dx)
        b=1+alpha(x[1:n],t[k])*dt/dx**2+r*dt/2
        c=-alpha(x[1:n],t[k])*dt/dx**2/2-beta(x[1:n],t[k])*dt/(4*dx)
        e=1-alpha(x[1:n],t[k])*dt/dx**2-r*dt/2
        F=-a*U[:-2]+e*U[1:-1]-c*U[2:]
        U[0]=phi1(t[k],xmin)
        U[n]=phi2(t[k],xmax)
        diagonals[0,1:]=c[:-1]
        diagonals[1]=b
        diagonals[2,:-1]=a[1:]
        F[0]=F[0]-a[0]*U[0]
        F[n-2]=F[n-2]-c[n-2]*U[n]
        U[1:n]=linalg.solve_banded((1,1),diagonals,F)
        U=maximum(U,U0)
    return U
r=0.1
sigma=0.5
D=0.0
T=0.5
E=100
S0=100
def p(s):
    return maximum(E-s,0)
def u0(x):
    return p(exp(x))
```

```
gamma=-r
def alpha(x,t):
    return sigma*sigma/2*ones(size(x))
def beta(x,t):
    return (r-D-sigma*sigma/2)*ones(size(x))
def phi1(t,xmin):
    return p(exp(xmin))
def phi2(t,xmax):
    return p(exp(xmax))
rho=2
xmin=log(S0/float(rho))
xmax=log(S0*float(rho))
m0=10
n0=20
m=m0
n=n0
print BS_implicit_solver_am(m,n,xmin,xmax,T,r,u0,alpha,beta,phi1,phi2)[n/2]

xmin=0
xmax=400
def phi1(t,xmin):
    return(p(0))
def phi2(t,xmax):
    return p(xmax)
def alpha(x,t):
    return sigma*sigma/2*x**2
def beta(x,t):
    return (r-D)*x
for i in range(7):
    m=4**i*m0
    n=2**i*n0
    approx=BS_implicit_solver_am(m,n,xmin,xmax,T,r,p,alpha,beta,phi1,phi2)
    print approx[n/4]
    ds=(xmax-xmin)/(float(n))
    print (approx[n/4+1]-approx[n/4-1])/(2*ds)
```

```
#Sample solutions of Lab 13

from scipy import *
S=loadtxt("cisco_lab13.csv",delimiter=",",skiprows=1,usecols=(4,))
n=size(S)
S=S[arange(n-1,-1,-1)]#S=S[::-1]
#i=arange(0,n-1)
#use the first half of closing prices to estimate sigma
i=arange(126)
u=log(S[i+1]/S[i])
dt=1.0/n
sigma1=std(u)/sqrt(dt)
#mu=mean(u)/dt+sigma**2/2
#these are market parameters, if we assume that
#the Black-Scholes model with constant mu and sigma are valid

#check if the assumption is satisfied
#if satisfied, then ui are values of normally distributed random variables
from scipy import linalg
def implicit_am(n,m,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r):
    x=linspace(xmin,xmax,n+1)
    dx=(xmax-xmin)/float(n)
    dt=T/float(m)
    t=linspace(0,T,m+1)
    #to save memory use only one row of the matrix U
    U=zeros(n+1)
    #initial condition
    U0=u0(x)
    U[:]=u0(x)#or U[:]=U0.copy()
    i=arange(1,n)
    i1=arange(n-1)
    i2=arange(n-2)
    M=zeros(shape=(n-1,n-1))
    F=zeros(n-1)
    for k in arange(1,m+1):
        a_ki=-(dt/dx**2*(alpha(x[i],t[k])-beta(x[i],t[k])/2*dx))
        b_ki=1+2*dt/dx**2*alpha(x[i],t[k])+r*dt
        c_ki=-(dt/dx**2*(alpha(x[i],t[k])+beta(x[i],t[k])/2*dx))
        M[i1,i1]=b_ki
        M[i2,i2+1]=c_ki[i2]
        M[i2+1,i2]=a_ki[i2+1]
        F[i1]=U[i]
        #boundary values
        U[0]=phi1(t[k])
        U[n]=phi2(t[k])
        #modify right hand side
        F[0]=F[0]-a_ki[0]*U[0]
        F[n-2]=F[n-2]-c_ki[n-2]*U[n]
        U[i]=linalg.solve(M,F)
        U=maximum(U,U0)
    return(U)
def transformedBSsolver_implicit_am(n,m,r,D,S0,sigma,rho,T):
    def p(s):
        #return((90-s)*(s<80)+(s-100)**2/40.0*((s>=80)&(s<=120))+(s-110)*(s>120))
        return(maximum(23.16-s,0)) #simple put option
    #boundary conditions depend on payoff!
    xmin=log(S0/float(rho))
    xmax=log(S0*rho)
    def u0(x):
        return(p(exp(x)))
```

```
    def phi1(t):
        return(u0(xmin)) #simple boundary condition
    def phi2(t):
        return(u0(xmax)) #simple boundary condition
    def alpha(x,t):
        return(sigma(exp(x),t)**2/2.0)
    def beta(x,t):
        return(r-D-sigma(exp(x),t)**2/2.0)
    #return the value for S0. assume n is even
    return(implicit_am(n,m,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r)[n/2])

# try the code in the case of constant volatility
def sigma(s,t):
    return(sigma1*ones(size(s)))
r=0.03
D=0.00
S0=S[125]
rho=2
n=200
m=100
T=0.5

def transformedBSsolver_implicit_am_derivative(n,m,r,D,S0,sigma,rho,T):
    #compute dv/dS at S=S0, t=0
    def p(s):
        #return((90-s)*(s<80)+(s-100)**2/40.0*((s>=80)&(s<=120))+(s-110)*(s>120))
        return(maximum(23.16-s,0)) #simple put option
    #boundary conditions depend on payoff!
    xmin=log(S0/float(rho))
    xmax=log(S0*rho)
    def u0(x):
        return(p(exp(x)))
    def phi1(t):
        return(u0(xmin)) #simple boundary condition
    def phi2(t):
        return(u0(xmax)) #simple boundary condition
    def alpha(x,t):
        return(sigma(exp(x),t)**2/2.0)
    def beta(x,t):
        return(r-D-sigma(exp(x),t)**2/2.0)
    values=implicit_am(n,m,xmin,xmax,T,u0,phi1,phi2,alpha,beta,r)
    dx=(xmax-xmin)/n
    derivative=1.0/S0*(values[n/2+1]-values[n/2-1])/(2*dx)
    #return the value for S0. assume n is even
    return(derivative)
price=transformedBSsolver_implicit_am(n,m,r,D,S0,sigma,rho,T)
derivative=transformedBSsolver_implicit_am_derivative(n,m,r,D,S0,sigma,rho,T)
i=125
bank=price-derivative*S[i]
dt=1/250.0
eta=derivative
print(bank)
print(eta)
results=zeros(shape=(126,4))
results[0,0]=bank
results[0,1]=eta
results[0,2]=price #value of portfolio
results[0,3]=0 #payoff
#trade simulation
for i in arange(126,251):
```

```
    T=T-dt
    S0=S[i]
    derivative=transformedBSsolver_implicit_am_derivative(n,m,r,D,S0,sigma,rho,T)
    bank=bank*(1+r*dt)-(derivative-eta)*S0
    eta=derivative
    value=bank+eta*S0
    payoff=maximum(23.16-S0,0)
    results[i-125,0]=bank
    results[i-125,1]=eta
    results[i-125,2]=value #value of portfolio
    results[i-125,3]=payoff

#next day
```

```
from scipy import *
from scipy import linalg

def Asian(ns,nI,m,Smax,Imax,sigma,r,D,T,p,phi1,phi2):
    s=linspace(0,Smax,ns+1)
    I=linspace(0,Imax,nI+1)
    ds=Smax/float_(ns)
    dI=Imax/float_(nI)
    dtau=T/float_(m)
    Uk=zeros(shape=(ns+1,nI+1))
    Uk_1=zeros(shape=(ns+1,nI+1))
    i=arange(1,ns)
    rho=dtau/ds**2
    a=rho/4*(-s[i]**2*sigma(s[i])**2+(r-D)*s[i]*ds)
    b=0.5*(1+rho*s[i]**2*sigma(s[i])**2+s[i]*dtau/dI+r*dtau)
    c=-rho/4*(s[i]**2*sigma(s[i])**2+(r-D)*s[i]*ds)
    d=-a
    e=0.5*(1-rho*s[i]**2*sigma(s[i])**2+s[i]*dtau/dI-r*dtau)
    f=-c
    g=0.5*(-1+s[i]*dtau/dI)
    #use initial conditions
    for i in arange(ns+1):
        Uk_1[i,:]=p(s[i],I/T)

    #define the matrix of the system to be solved
    i1=arange(ns-1)
    i2=arange(ns-2)
    M=zeros(shape=(ns-1,ns-1))
    M[i1,i1]=b
    M[i2,i2+1]=c[i2]
    M[i2+1,i2]=a[i2+1]
    i=arange(1,ns)
    tau=linspace(0,T,m+1)
    #start time stepping
    for k in arange(1,m+1):
        #use boundary conditions
        #boundary I=Imax
        Uk[:,nI]=phi1(s,tau[k])
        #boundary S=0
        Uk[0,:nI]=p(0,I[:nI]/T)*exp(-r*tau[k])
        #boundary S=Smax
        Uk[ns,:nI]=phi2(I[:nI],tau[k])

        #start backward stepping in j
        for j in arange(nI-1,-1,-1):
            #compute the right hand side
            F=d*Uk_1[i-1,j+1]+e*Uk_1[i,j+1]+f*Uk_1[i+1,j+1]+g*(Uk[i,j+1]-Uk_1[i,j])
            F[0]=F[0]-a[0]*Uk[0,j]
            F[ns-2]=F[ns-2]-c[ns-2]*Uk[ns,j]
            #solve the system
            Uk[i,j]=linalg.solve(M,F)
        #new level is computed, it becomes old one
        Uk_1=Uk.copy()
    #return the solution
    return(Uk)
r=0.05
D=0
def sigma(s):
    return(ones(size(s))*0.5)
T=0.5
```

```
E=100
S0=95
Imax=E*T
Smax=2*S0
def phi1(s,tau):
    return(zeros(size(s)))
def phi2(I,tau):
    return(maximum(E*exp(-r*tau)+exp(-D*tau)/((r-D)*T)*(exp(-(r-D)*tau)-1)*Smax-exp(-
r*tau)/T*I,0))
def p(s,A):
    return(maximum(E-A,0))
m=20
ns=200
nI=100
```

```
#lab 15 sample code
from scipy import *
from scipy import linalg

def solver(x,y0,y1):#x-gridpoints
    n=size(x)-1 #the number of intervals
    c=zeros(n+1)
    c[0]=y0
    c[n]=y1
    h=x[1:]-x[:n] #lengths of intervals
    M=zeros(shape=(n-1,n-1))
    #diagonal of M
    i=arange(n-1)
    M[i,i]=-1/h[i]-1/h[i+1]+(h[i]+h[i+1])/3

    #above diagonal
    i=arange(n-2)
    M[i,i+1]=1/h[i+1]+h[i+1]/6
    #below diagonal
    M[i+1,i]=1/h[i+1]+h[i+1]/6
    F=zeros(n-1)
    #modify first and last elements of F
    F[0]=F[0]-c[0]*(1/h[0]+h[0]/6)
    F[n-2]=F[n-2]-c[n]*(1/h[n-1]+h[n-1]/6)
    c[1:n]=linalg.solve(M,F)
    return(c)
x=linspace(0,1,10)
result=solver(x,1,0)
```

Additional resources:

Video: Using Moodle https://moodle.ut.ee/mod/resource/view.php?id=48624

Video: Getting Portable Python https://moodle.ut.ee/mod/resource/view.php?id=45175

Video: Finding mistakes in a code that runs without error messages
https://moodle.ut.ee/mod/resource/view.php?id=46643