CQF Final Project Report

Rustam Guseynov

July 9, 2013

Abstract

When I implemented this project, my primary goal was to better understand the models and find answers to questions I couldn't answer before. Hence, I tried to avoid reproducing all formulas already provided and proven in textbooks or going into great details of well-known mathematical methods I used. At the same time I try to show which choice I had during project implementations, what decisions I made, and why.

1 Introduction

1.1 Choice of assignments

After thorough examination of the tasks available, I have chosen HJM Model and Static Hedge. The reason for HJM Model was to obtain hands-on experience with interest rate models, which I find essential to understand quantitative finance. Interest rate models are more complex than equity model because of incompleteness of interest rate market, and at the same time, they are good starting point for more advanced models, for example, credit risk models.

Static Hedge is was chosen for two reasons: first, I did FDM when studied in university, and second is that uncertain parameters models look very impressive, they are simple and able to solve very advanced problems. Uncertain parameters approach can easily incorporate uncertain interest rate and uncertain dividends, which together with uncertain volatility covers most substantial uncertainties in pricing of contingent claims. Static hedging in this setting is essential to narrow bid-ask spread on securities, priced with uncertain parameters model.

1.2 Computational environment

I have chosen Matlab as a primary programming environment. The choice was determined by the unique features of Matlab: easy vector operations, lots of built-in linear algebra algorithms, and convenience of programming language itself. It supports advanced programming techniques like anonymous functions and structures with dynamic fields. It also supports OOP and code structuring via packages. Matlab allows to concentrate on the task first, and then go to details as deep as it's necessary.

Matlab also has a very convenient feature of storing data in its native ".mat" files, which were like Excel spreadsheets or database tables. I have once downloaded, structured and imported source data, and then stored them into .mat file, which is reloaded each time script runs.

My code can be also run with Octave.

2 HJM Model

2.1 Comparing to short-rate models

HJM model was the first interest rate model which took advantage of modelling whole yield curve instead of modelling only one point of that curve like it was in short-rate models.

Why do I say "advantage"? Generally speaking, there's nothing wrong with short-rate models, but there's something unnatural in the way they treat bond pricing and yield curve modelling. Here's what I mean.
Let's take spot yield curve r(t). Instantaneous forward yield curve is then

$$f(t) = \frac{d}{dt}(rt) = r(t) + r'(t)t \tag{1}$$

and, conversely,

$$r(t) = \frac{1}{t}\int_0^t f(s)ds \tag{2}$$

Price of zero-coupon bond is computed very naturally, by simple discounting bond future (face) value by rate, corresponding to the term of the bond:

$$Z(0,T) = e^{-r(T)T} = \exp\left(-T\frac{1}{T}\int_0^T f(s)ds\right) = \exp\left(-\int_0^T f(s)ds\right) \tag{3}$$

These formulas are very intuitive and could be used directly should we have behaviour of spot or forward yield curve modelled. But in case of short-rate models what we model is short rate $r(0) = r_0$ and thus it is impossible to get $r(T)$ directly. Without understanding of yield curve movements, one has to consider ZCB to be a derivative on a short rate and look for solution of bond pricing equation (Black-Scholes equation for a bond) in form

$$Z(t,T) = e^{A(t,T)r + B(t,T)} \tag{4}$$

Comparing this equation to "true" bond pricing formula (3), I conclude that short-rate models approximate longer interest rate with linear function of r_0 in form $A(t,T)r_0 + B(t,T)$. One could call it a "first-order approximation". Later on one could estimate full yield curve $r(t,T)$ and forward curve $f(t,T)$ from these bond prices

$$r(t,T) = -\frac{\ln Z(t,T)}{T-t}, \tag{5}$$

$$f(t,T) = -\frac{d\ln Z(t,T)}{dT} \tag{6}$$

Thus short rate models are a "double trouble": a headache to implement and deduce bond prices, and only a first-order approximation of true dynamics of yield curve. The latter shortcoming can be overcome by using multidimensional model, but for the cost of significant increase of model's complexity.

2.2 HJM setting

As I have already told before, HJM is a model of a whole yield curve. Or, more specifically, a model of an instantaneous forward yield curve.

[1, ch. 37] starts with premise that ZCB price follows some general lognormal random walk

$$dZ(t,T) = \mu(t,T)Z(t,T)dt + \sigma(t,T)Z(t,T)dW_t \qquad (7)$$
$$\forall t \; Z(t,t) = 1$$

Integrating (7) into

$$lnZ(t,T) = \left(\mu(t,T) - \frac{1}{2}\sigma^2(t,T)\right)t + \sigma(t,T)W_t \qquad (8)$$

and substituting into (6) we obtain

$$f(t,T) = \frac{\partial}{\partial T}\left(\frac{1}{2}\sigma^2(t,T) - \mu(t,T)\right)t - \frac{\partial}{\partial T}\sigma(t,T)W_t \qquad (9)$$

It can be shown ([2, par. 10.3.2] for example) that in risk-free case $\mu(t,T) = r(t)$. This means that $\frac{\partial}{\partial T}\mu(t,T) = 0$ and thus

$$f(t,T) = \sigma(t,T)\frac{\partial\sigma(t,T)}{\partial T}t - \frac{\partial}{\partial T}\sigma(t,T)W_t \qquad (10)$$

Introducing $\nu(t,T) = -\frac{\partial}{\partial T}\sigma(t,T)$ and differentiating we obtain almost final version of forward rate SDE

$$df(t,T) = \nu(t,T)\left(\int_t^T \nu(t,s)ds\right)dt + \nu(t,T)dW_t$$

Finally, applying Musiela parametrization $\nu(t,T) = \bar{\nu}(t,T-t)$ and SDE becomes

$$d\bar{f}(t,\tau) = \left[\bar{\nu}(t,\tau)\int_0^\tau \bar{\nu}(t,s)ds + \frac{\partial}{\partial\tau}\bar{f}(t,\tau)\right]dt + \bar{\nu}(t,\tau)dW_t \qquad (11)$$

where $\tau = T - t$.

2.3 Dimensionality and dimension reduction

The main input into equation (11) is $\bar{\nu}(t,\tau)$. In this model we do not simulate or forecast volatility, hence we use constant volatility model. This means that $\bar{\nu}(t,\tau) = \nu(\tau)$, which is known at time zero. That volatility function is an only source of randomness in SDE. Term $\bar{\nu}(t,\tau)dW_t$ means that on each step the whole yield curve will experience a shift proportional to value of $\bar{\nu}(t,\tau)$ at particular term τ. At the same time the model can easily be extended so that to introduce several random walks. It takes form

$$d\bar{f}(t,\tau) = \left[\sum_{k=1}^{N} \bar{\nu_k}(\tau) \int_0^\tau \bar{\nu_k}(s)ds + \frac{\partial}{\partial\tau}\bar{f}(t,\tau)\right] dt + \sum_{k=1}^{N} \bar{\nu_k}(\tau)dW_{t,k} \qquad (12)$$
$$\forall k = \overline{1:N}$$

where N is number of components.

We calibrate the model to the historical data, so we can easily estimate interest rates covariance matrix Ω. Let us suppose we have history on N interest rates of different terms. In this case covariance matrix components are determined by $(\Omega)_{ij} = \rho_{ij}\sigma_i\sigma_j$ and $\Omega \in R^{N\times N}$, where $\rho_{ij} = corr(r_i, r_j)$ - correlation of i^{th} and j^{th} interest rates and σ_i is standard deviation of i^{th} rate.

In this case we have simulate N factors, one for each term. This means we need N correlated random walks. In appendix A it is shown that there's an easy way to reduce number of random walks by extracting principal components.

2.4 Implementation

2.4.1 Data

I use two datasets. First is Bank of England yield curves, as suggested in project description and another is a set of MICEX [1] OFZ[2] curves. Both datasets are stored in Matlab files: hjm_boe_forward.mat and hjm_micex_nss.mat. Bank of England yield curves are ready to use, and after loading hjm_boe_forward.mat file I obtain main variables: terms (1D array of terms of interest rates) and rates(2D array of interest rates history).

MICEX data are a bit more contrived. The data consists of a list of daily parameters of extended Nelson-Siegel-Svennson[3] approximation of validated and bootstrapped OFZ curves. I store this list in hjm_micex_nss.mat. After loading it I create my own grid of terms and evaluate rate curves at these points. Then I recalculate them into forward rates.

Choice between data sources depends on SELECTED_MODEL variable. It can take two values, BANK_ENGLAND_FWD and MICEX_NSS.

[1] Major Russian exchange
[2] Russian domestic treasury bonds
[3] More detailed description is available on MICEX website

2.4.2 Sampling

In order to simulate equation (12), we have to replace it with its discrete version. Let's start from t dimension. Time t starts at zero and has step of arbitrary fixed length δt.

$$\bar{f}_i(\tau) - \bar{f}_{i-1}(\tau) = \left[\sum_{k=1}^{N} \bar{\nu_k}(\tau) \int_0^\tau \bar{\nu_k}(s)ds + \frac{\partial}{\partial \tau} \bar{f}_{i-1}(\tau) \right] \delta t + \sum_{k=1}^{N} \bar{\nu_k}(\tau)\delta W_k \tag{13}$$

Here N is number of principal components, and i ranges from 1 to arbitrary chosen period of time.

Next step is terms dimension τ. In my case I either already have data taken at discrete points (Bank of England data are presented as rates for certain terms) or I could take some continuous data and retrieve samples at specific terms. This means I have a system of M equations, there M is number of points on term grid.

$$\bar{f}_i^j - \bar{f}_{i-1}^j = \left[\sum_{k=1}^{N} \bar{\nu_k}(\tau_j) \int_0^{\tau_j} \bar{\nu_k}(s)ds + \frac{\bar{f}_{i-1}^j - \bar{f}_{i-1}^{j-1}}{\delta \tau_j} \right] \delta t + \sum_{k=1}^{N} \bar{\nu_k}(\tau_j)\delta W_k \tag{14}$$

where $j = \overline{1:M}$ and $\delta\tau_j = \tau_j - \tau_{j-1}$ is a length of j^{th} term grid step.

The only thing we have to estimate here is integral $\int_0^{\tau_j} \bar{\nu_k}(s)ds$. $\bar{\nu}(\tau)$, which is a function with known values on grid points. Hence we can easily do a numerical integration using, for example, standard trapezoid rule:

$$\int_0^{\tau_j} \bar{\nu_k}(s)ds = \sum_{i=1}^{j-1} \frac{\bar{\nu_k}(\tau_i) + \bar{\nu_k}(\tau_{i+1})}{2} \delta\tau_i \tag{15}$$

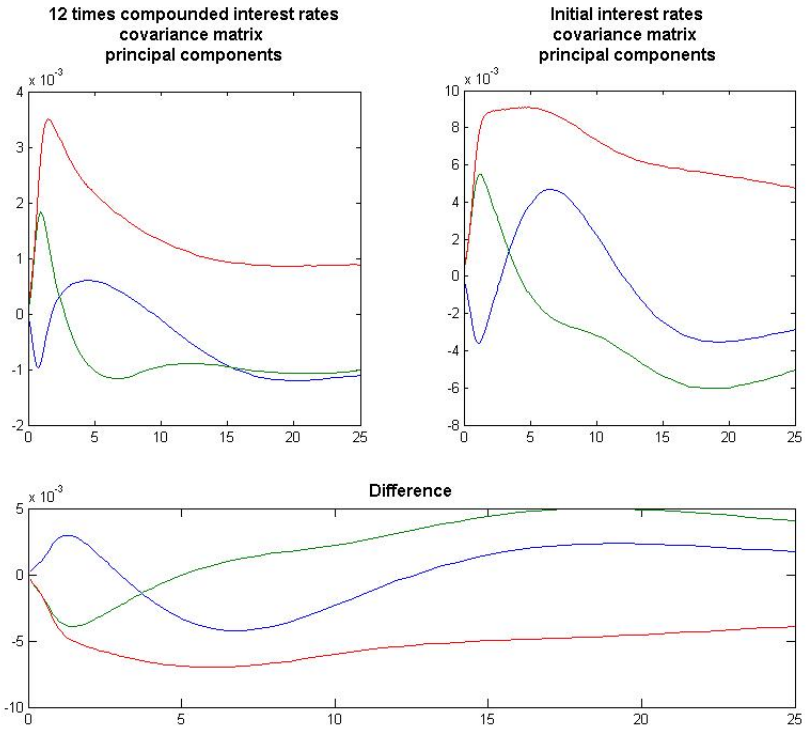
Another approach is to use approximated principal components. In this case integration is performed as described in appendix B.

2.4.3 Few more words on volatility

The last issue I wanted to discuss is volatility structure. In [1, par. 37.15] it is told that standard HJM model might give negative interest rates. That was exactly what I have gotten, and it was a bit irritating.

To avoid it I implemented a non-infinitesimal short rate model as it is explained in there. In order to switch from standard to non-infinitesimal model it is necessary to set variable LOGNORMAL from 0 to 1.

In non-infinitesimal model we work with m times compounded interest rates. In this case it might be correct to recalculate our interest rates into corresponding compounding and calculate covariance matrix for such rates. Recalculation is done using formula $r_m = m\left((1 + r_1)^{1/m} - 1\right)$. There's significant difference in principal components:



In this case no negative interest rates occur.

2.4.4 Fitting eigenvectors with polynomials

In the lectures it was told that it is necessary to substitute eigenvectors with their polynomial approximation. The reason was that polynomial function is smooth and with these functions simulation would give better results.

I tried both approximated and raw vectors and I must say that I found no significant difference in model behaviour.

2.5 Pricing financial instruments

2.5.1 Zero-coupon bond

In order to price a zero-coupon bond we need to use equation 3, namely

$$Z(0,T) = \exp\left(-\int_0^T f(s)ds\right)$$

where $f(s)$ is an instantaneous forward rate at time s. In terms of the model, instantaneous forward rate at time s is $\bar{f}_i^0$ where $i = \frac{s}{\delta t}$. This means that integral in question can be written as $\int_0^T f(s)ds = \sum_{i=0}^N \bar{f}_i^0 \delta t = \delta t \sum_{i=0}^N \bar{f}_i^0$. This allows us to calculate bond price for one simulation.

In case of M simulations the final bond pricing formula becomes:

$$Z(0,T) = \frac{1}{M}\sum_{k=1}^{M}\exp\left(-\delta t\sum_{i=0}^{N}\bar{f}_{i,k}^{0}\right) \tag{16}$$

2.5.2 Cap pricing

Zero coupon bond price is a discount factor, which can be used when pricing more complex derivatives. Hence, bond pricing formula can be easily aggregated to price any contingent claim with payoff V_T

$$V_0 = \frac{1}{M}\sum_{k=1}^{M}\left[\exp\left(-\delta t\sum_{i=0}^{N}\bar{f}_{i,k}^{0}\right)V_T\right] \tag{17}$$

Cap payoff is $(Y_t^\tau - R)^+$, where Y_t^τ is a spot rate at time t with term τ. If $t = t_i$ and $\tau = t_n$, then

$$Y_{t_i}^{t_n} = \frac{\exp\left(\delta t\sum_{k=0}^{n-1}\bar{f}_{i,k}^{0}\right) - 1}{n\delta t} \tag{18}$$

3 Uncertain volatility and static hedge

3.1 Task description

Uncertain volatility model assumes volatility to be not just random, but uncertain, i.e. we do not know anything about its distribution and can rely only on its expected range. Contingent claim price is still determined by Black-Scholes equation, but instead of constant volatility this equation now uses the most undesirable variable of volatility from that range. This means that instead of constant σ equation contains $\sigma = \sigma(\Gamma)$, which definition depends on whether option is long or short. For more details I refer to [1, ch. 52 and 60].

So the task basically consist of two major parts. First is to solve non-linear bond pricing equation (as described above) and the second is to solve optimization problem. The goal of the second part is to find combination of our option being priced with other traded instruments, which (combination) will have least residual cost.

Optimization doesn't make sense for linear pricing models, because in linear pack of options cost is exactly the same as the sum of their prices. That's not the case for non-linear models. One can create a pack, which will contain some non-traded exotic option and number of traded vanilla options. Then he or she will try to solve optimization problem:

$$[\lambda_1, ..., \lambda_N] = \underset{[\lambda_1,...,\lambda_N]\in\mathbf{R}^N}{\arg\min}\left[\text{Price}\left(\text{Exotic} + \textstyle\sum_{i=1}^{N}\lambda_i\text{Vanilla}_i\right) - \sum_{i=1}^{N}\lambda_i\text{Price}\left(\text{Vanilla}_i\right)\right].$$

Application of static hedge allows to narrow bid-ask spread on exotic product, which price is calculated using uncertain parameters model.

So, this task consists of two major parts, implementation of finite-difference scheme, and optimization. I will address these issues in the sections below.

3.2 Finite-difference method for single option

Price of derivatives in question can be found as a solution of Black-Scholes PDE, which takes form of:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0 \tag{19}$$

This means that in the first part of this task we have to develop an FDM which solves BS equation with necessary boundary conditions.

Finite difference methods approach suggest substitution of partial derivatives with their approximation with finite differences. The problem that arises is accuracy of such approximation and convergence of a numerical method.

First off, let's introduce grids on time and asset price axes. If time $t \in [0, T]$, then grid ω_t is $\omega_t = \{t_k = \frac{k}{\tau}, k = 0...N_t, \delta t = \frac{T}{K+1}\}$. K is arbitrary chosen number of points on time axis. Similarly, asset price grid is $\omega_s = \{S_i = \frac{i}{h}, i = 0...N, h = \frac{S_{max}}{N+1}\}$.

I will use the following symbols for difference operations:

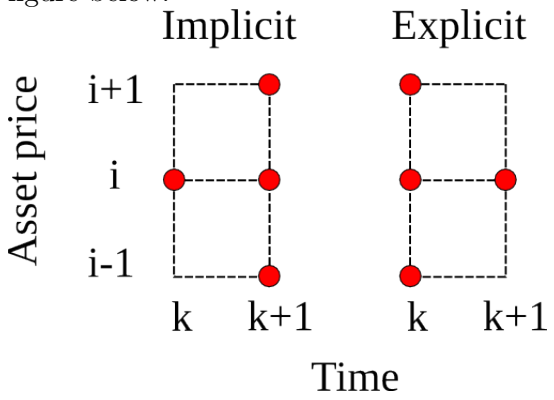
- Right first-order derivative: $V_s = \frac{V_{i+1} - V_i}{h} = V'(s) + O(h)$
- Left first-order derivative: $V_{\bar{s}} = \frac{V_i - V_{i-1}}{h} = V'(s) + O(h)$
- Central first-order derivative: $V_{\mathring{s}} = \frac{V_{i+1} - V_{i-1}}{2h} = V'(s) + O(h^2)$
- Second order derivative $V_{s\bar{s}} = \frac{1}{h}\left(\frac{V_{i+1} - V_i}{h} - \frac{V_i - V_{i-1}}{h}\right) = \frac{V_{i+1} - 2V_i + V_{i-1}}{h^2} = V''(s) + O(h^2)$

Finite difference approximation of BS equation at point i is the following:

$$V_t + \sigma^2 S_i^2 V_{\bar{s}s} + rS_i V_{\mathring{s}} + rV_i = 0 \tag{20}$$

This scheme has second order of accuracy w.r.t s and first order w.r.t t, or $O(\delta t, \delta s^2)$.

If last three terms of previous equation are taken at time step k, the scheme is explicit, and when $k+1$ is used, the scheme is implicit. These schemes are plotted on the figure below.



3.2.1 Boundary conditions

Payoff condition $(t = T, \forall s)$ depends on the option in question. In either case it is simply a value of payoff function, calculated in grid points.

Minimum asset price $(s = 0, \forall t)$. One possible way of treating lower boundary is to analyse option price behaviour at the lower bound depending on the type of the option. For example, call option price tends to zero when asset price tends to zero. However vanilla put price in same circumstances tends to strike price. This means, I'd have to build boundary conditions manually depending on type of the option.

In book [1, ch. ????] it was proposed a better boundary condition: take value at end point equal to linear projection of the values in two previous points, i.e. $V_0 = 2V_1 - V_2$. First, it makes sense from financial point of view, because conventional options exhibit linear behaviour on far bounds. Second, this approximation is of second order of accuracy by s:

$V_1 = V_0 + hV_0' + \frac{h^2}{2}V_0'' + O(h^3)$ and
$V_2 = V_0 + 2hV_0' + \frac{(2h)^2}{2}V_0'' + O(h^3)$ and hence
$2V_1 - V_2 = 2V_0 + 2hV_0' + h^2V_0'' - V_0 - 2hV_0' - 2h^2V_0'' + O(h^3) = V_0 + O(h^2)$.

Moreover, such approximation is very convenient when it comes to pricing of a pack of the options.

Maximum asset price $(s \to +\infty, \forall t)$. In this case everything told about lower boundary condition is still correct, and thus, similar condition can be used, which would look like $V_N = 2V_{N-1} - V_{N-2}$. The only thing to mention is that we are, of course, unable to stretch the grid up to positive infinity and hence we will have to use some large value of asset price instead. In literature it is told that $S_{max} = 2 \cdot \text{Strike}$ would be enough.

3.2.2 Scheme formulas

In case of explicit scheme FDM formula writes as follows:
$\frac{V_i^{k+1} - V_i^k}{\tau} + \frac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1}^{k+1} - 2V_i^{k+1} + V_{i-1}^{k+1}}{h^2} + rS_i \frac{V_{i+1}^{k+1} - V_{i-1}^{k+1}}{2h} - rV_i^{k+1} = 0$, or
$V_i^{k+1} + \frac{1}{2}\sigma^2 S_i^2 \frac{\tau}{h^2}\left(V_{i+1}^{k+1} - 2V_i^{k+1} + V_{i-1}^{k+1}\right) + rS_i \frac{\tau}{2h}\left(V_{i+1}^{k+1} - V_{i-1}^{k+1}\right) - r\tau V_i^{k+1} = V_i^k$.

Denoting $a_i = \frac{1}{2}\sigma^2 S_i^2 \frac{\tau}{h^2}$, $b_i = rS_i \frac{\tau}{2h}$ and $c_i = -r\tau$, obtain:
$V_i^{k+1} + a_i\left(V_{i+1}^{k+1} - 2V_i^{k+1} + V_{i-1}^{k+1}\right) + b_i\left(V_{i+1}^{k+1} - V_{i-1}^{k+1}\right) - c_i V_i^{k+1} = V_i^k$

We can also remember that $S_i = ih$ and hence $\frac{S_i}{h} = i$ and formulas for coefficients become: $a_i = \frac{1}{2}\sigma^2 i^2 \tau$, $b_i = \frac{1}{2}ri\tau$ and $c_i = -r\tau$.

Now, gathering coefficients at different V_i^k we have:
$(a_i - b_i)V_{i+1}^{k+1} + (1 - 2a_i + c_i)V_i^{k+1} + (a_i + b_i)V_{i-1}^{k+1} = V_i^k$.

Denoting $A_i = a_i + b_i$, $B_i = 1 - 2a_i + c_i$ and $C_i = a_i + b_i$ we have:

$$A_i V_{i+1}^{k+1} + B_i V_i^{k+1} + C_i V_{i-1}^{k+1} = V_i^k \tag{21}$$

This means I can obtain $V^k = [V_1^k, V_2^k, ..., V_N^k]^T$ - vector of function value on layer k, from equation $V^k = ZV^{k+1}$, where Z is a tridiagonal matrix with A, B and C coefficients above, on, and below main diagonal accordingly. In order to obtain completed form of this matrix Z we have to consider boundary conditions.

Lower boundary condition writes as $V_0 = 2V_1 - V_2$. This means that when $i = 1$, equation 21 becomes $A_1 V_2^{k+1} + B_1 V_1^{k+1} + C_1 V_0^{k+1} = V_1^k$, or $A_1 V_2^{k+1} + B_1 V_1^{k+1} + C_1(2V_1^{k+1} - V_2^{k+1}) = V_1^k$. This is equivalent to:

$$V_1^{k+1}(B_1 + 2C_1) + V_2^{k+1}(A_1 - C_1) = V_1^k \tag{22}$$

Similarly, upper boundary condition is now:

$$V_{N-2}^{k+1}(A_{N-1} - C_{N-1}) + V_{N-1}^{k+1}(B_{N-1} + 2C_{N-1}) = V_{N-1}^k \tag{23}$$

And the matrix Z is now:

$$Z = \begin{pmatrix} B_1 + 2C_1 & A_1 - C_1 & & & & & \\ A_2 & B_2 & C_2 & & & & \\ & A_3 & B_3 & C_3 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & A_{N-2} & B_{N-2} & C_{N-2} & \\ & & & & A_{N-1} - C_{N-1} & B_{N-1} + 2C_{N-1} \end{pmatrix} \tag{24}$$

Note that this matrix is $(N-1) \times (N-1)$, while the grid has $N+1$ points by S axis. This is because values in top and bottom row are to be calculated from boundary conditions.

In implicit scheme equation is:
$\frac{V_i^{k+1} - V_i^k}{\tau} + \frac{1}{2}\sigma^2 S_i^2 \frac{V_{i+1}^k - 2V_i^k + V_{i-1}^k}{h^2} + rS_i \frac{V_{i+1}^k - V_{i-1}^k}{2h} - rV_i^k = 0$, or after similar simplifications:

$$A_i V_{i+1}^k + B_i V_i^k + C_i V_{i-1}^k = V_i^{k+1} \tag{25}$$

where $A_i = -(a_i + b_i)$, $B_i = 1 + 2a_i - c_i$ and $C_i = -a_i + b_i$

3.2.3 Non-linear gamma case

In this case we have to substitute simple σ by $\sigma(\Gamma) = \begin{cases} \sigma_{min}, \Gamma < 0 \\ \sigma_{max}, \Gamma \geq 0 \end{cases}$. During the computations it is done according to the following algorithm:

1. Before FDM calculation cycle

- Calculate Z_{min} and Z_{max} matrices using σ_{min} and σ_{max} respectively, according to description above.

2. In FDM calculation cycle

   - In the calculation cycle.
   - Calculate Γ at current time step. We use option payoff as a starting point.
   - Take $Z = Z_{max}$. Replace rows of Z to those of Z_{min}, where Γ is negative. In Matlab code it looks as $Z(\Gamma < 0, :) = Z_{min}(\Gamma < 0, :)$. This, in essence, replaces equations with maximum Γ to those with minimum.

3.3 Finite-difference method for pack of options

3.4 Stability and convergence condition

In [1, ch. ????] it has been shown that in order for explicit scheme to converge, it is necessary that the following condition was held: $\tau \leq \frac{1}{\sigma^2 N^2}$. The implicit scheme is unconditionally convergent.

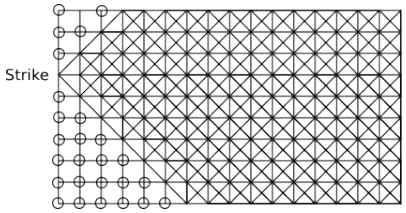
There are interesting analogy between explicit scheme and trinomial tree model in [4, ch. ???]. Since value at particular point on next time step is calculated as a weighted average of values of three sibling points on previous time step, the weight could be considered to be the probabilities of up, down and straight asset price movements. This argument gives two interesting consequences: first, it seems to be rightful to require these probabilities to be non-negative and add up to one. Or, more precisely, to discounted one, because on each time step of the tree discounting takes place. It is easy to see that it is right: $A_i + B_i + C_i = (a_i + b_i) + (1 - 2a_i + c_i) + (a_i - b_i) = 1 + c_i = 1 - r\tau$. And the non-negativity condition yields:

$$\begin{cases} a_i + b_i > 0 \\ 1 - 2a_i + c_i > 0 \\ a_i - b_i > 0 \end{cases}$$

The second equation yields $\sigma^2 i^2 \tau + r\tau - 1 < 0$ or $\tau(\sigma^2 i^2 + r) < 1$, and hence $\tau < \frac{1}{\sigma^2 N^2}$, which is stability condition.

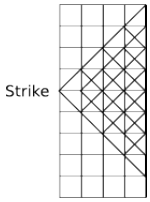
Another interesting consequence of trinomial tree analogy is that it is not really necessary[4] to do the calculations in each and every point of the grid! It would be enough to calculate along the trinomial tree (cone), that converges to the point with coordinates $(S_0, 0)$, i.e. the one that gives the value of the option at time zero and at asset price equal to current spot.

[4]While it is not really necessary indeed, I still did the calculations on the whole grid. First, we do FDM here, not the tree. And second thing was that I wanted to see whole price surface, not the value in a single point



In the figure above circled points are those where calculations are not necessary. Number of points on time axis is enough for scheme to converge to option value at current strike.

And, finally, this gives another great interpretation and justification of stability condition for explicit scheme: time step must be small enough to ensure that there convergent cone could fit into the grid. As an example here's a figure below where there's too few points on time axis for scheme to converge.



3.5 Optimization

Another task was to develop an optimization procedure so that to find optimal hedge. According to recommendations I implemented downhill simplex method following the [7, §10.5].

I remember it was suggested that we could employ simulated annealing methods, but in this case it was of no need, because the function in question is smooth and has no local optima.

In theory, we could also try using some kind of gradient descent, since FDM is well suited to calculate option price derivatives (greeks τ and Δ). In practice I found simplex method to be fast enough, and good gradient descent is rather complex to implement.

3.6 Results

3.7 Further improvements

- American options
- Barrier options
- Forward-start Ratchet options
- Uncertain interest rate
- Better schemes with improved convergence
- Irregular grids

A PCA and simulation of correlated random walks

Any matrix is simply a linear transformation in multidimensional vector space. It is well known that any matrix can be decomposed into set of rotations, shifts and scalings.

Let us suppose we have a covariance matrix $\Omega \in R^{N\times N}$ and vector of i.i.d. standard normal random variables $\eta \in R^N$. It is known that for any covariance matrix there exist self-adjoint matrices $U, \Lambda \in R^{N\times N}$ where U contains columns of eigenvectors and $\Lambda = diag(\lambda_1, ..., \lambda_N)$ - eigenvalues of matrix Ω. The eigenvalues are non-negative due to nature of covariance matrix.

A.1 Generating correlated random walks using eigenvalue decomposition

I will start with vector $\eta = (\eta_1, ..., \eta_N)$ of i.i.d standard normal random variables. My goal is to obtain vector ξ of correlated standard normal random variables. Desired correlation structure is described by covariance matrix Ω.

Eigenvalue decomposition of covariance matrix is $\Omega = U\Lambda U^T = \left(U\sqrt{\Lambda}\right)\left(U\sqrt{\Lambda}\right)^T$. Let $\xi = U\sqrt{\Lambda}\eta$. In such case

$$cov(\xi_i, \xi_j) = E\left[(\xi_i - E\xi_i)(\xi_j - E\xi_j)\right] = E\left[\left(U\sqrt{\Lambda}\eta_i - EU\sqrt{\Lambda}\eta_i\right)\left(U\sqrt{\Lambda}\eta_j - EU\sqrt{\Lambda}\eta_j\right)\right]$$

Taking into account that $EU\sqrt{\Lambda}\eta_i = 0$ for any i we obtain

$$\begin{aligned} E\left[\left(\sum u_{ik}\sqrt{\lambda_k}\eta_k\right)\left(\sum u_{jk}\sqrt{\lambda_k}\eta_k\right)\right] = \\ E\sum u_{ik}\sqrt{\lambda_k}\eta_k u_{jm}\sqrt{\lambda_m}\eta_m = \\ \sum u_{ik}\sqrt{\lambda_k}u_{jm}\sqrt{\lambda_m}E(\eta_k\eta_m) = \\ \{\eta\text{'s are standard normal i.i.d}\} = \\ \sum u_{ik}\lambda_k u_{jk} = (\Omega)_{ij}\text{, q.e.d} \end{aligned} \quad (26)$$

A.2 Reducing number of dimensions

Now vector ξ can be written as $\xi = \sum U_k\sqrt{\lambda_k}\eta_k$, where U_k is k^{th} eigenvector. Evidently, ξ is a sum of vectors η weighted by their eigenvalues. This means we can select only K largest eigenvalues for which value

$$R = \frac{\sum_{i=1}^{K}\sqrt{\lambda_i}}{\sum_{i=1}^{N}\sqrt{\lambda_i}} \quad (27)$$

is greater than some threshold (for example, 95%).

B Polynomial approximation

In order to generate smooth and nicely correlated trajectories of interest rates of adjacent terms, we have to smooth the eigenvectors obtained by PCA as described in appendix A. These vector already look smooth, but polynomial approximation has an advantage that it allows easy iteration and differentiation without using any numerical procedures.

B.1 Polynomial function representation

Suppose we have a polynomial $P_N(t) = \sum_{k=0}^{N} a_k t^k$ of power N. Then it can be stored as a vector of its $N+1$ components $V_{N+1} = [a_0, a_1, ..., a_N]$.

Integration of this polynomial on range from 0 to arbitrary value t would give $\int_0^t P_N(s)ds = P_{N+1}(t) = \sum_{k=0}^{N} \frac{a_k}{k+1} t^{k+1}$ which in vector representation is $V_{N+2} = \left[0, \frac{a_0}{1}, \frac{a_1}{2}, ..., \frac{a_N}{N+1}\right]$.

Similarly, differentiation would give $\frac{dP_N(t)}{dt} = P_{N-1}(t) = \sum_{k=1}^{N-1} k a_k t^k$ which in vector representation is $V_N = [a_1, 2a_2..., Na_N]$.

B.2 Solving for polynomial parameters

Now, how should one obtain polynomial approximation of arbitrary function given its values on some grid?

Suppose that we have an interval $x \in [A, B]$ which is divided into not necessarily even parts $A = x_0, x_1, ..., x_{N-1}, x_N = B$ and we have values $y_i = y(x_i)$ for $x = \overline{0:N}$. We want to find the closest approximation of that grid function y_i with polynomials of degree of N in mean-square sense.

This means we have to solve optimization problem

$$J(\theta) = \frac{1}{N+1} \sum_{k=0}^{N} \left(y_k - f(\theta, x_k)\right)^2 \to max \tag{28}$$

where $f(\theta_N, x) = \sum_{k=0}^{N} \theta_k x^k$. Using vector notation. we can rewire it as follows:

$$\theta = \text{argmax} \left[J(\theta) = (\mathbf{y} - \theta \cdot \mathbf{x})^T (\mathbf{y} - \theta \cdot \mathbf{x})\right] \tag{29}$$

Here $y = (y_1, ..., y_N)^T$, $\theta = (\theta_1, ...\theta_M)$ and

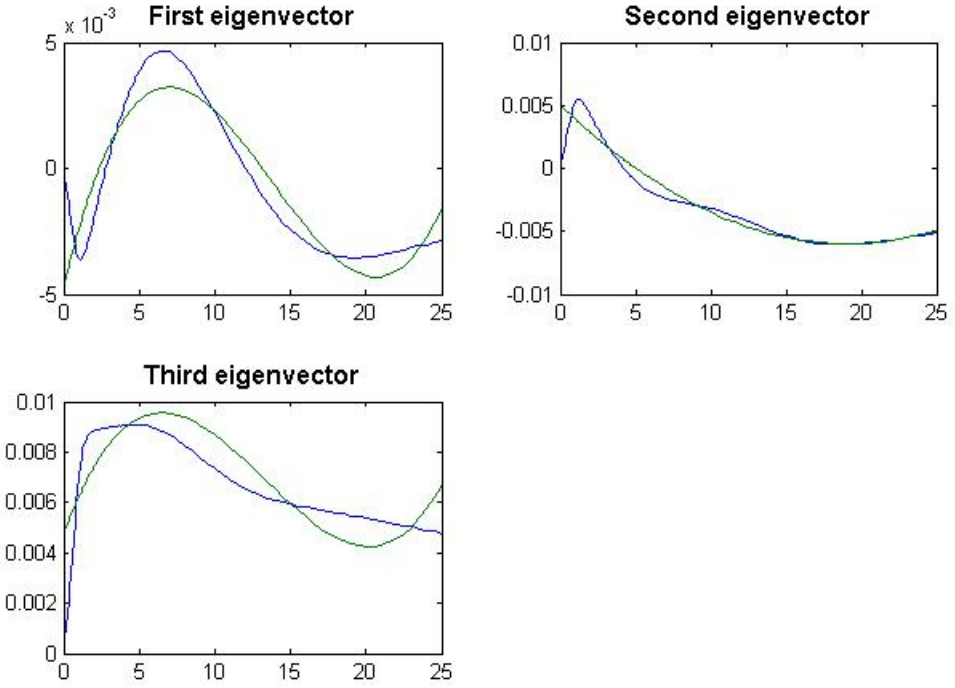
$$X = \begin{pmatrix} 1 & x_1 & \cdots & x_1^M \\ 1 & x_2 & \cdots & x_2^M \\ \vdots & \vdots & \cdots & \vdots \\ 1 & x_N & \cdots & x_N^M \end{pmatrix}$$

Using rules of matrix calculus we can solve this optimization problem by solving equation

$$\frac{dJ(\theta)}{d\theta} = 0 \iff \frac{d}{d\theta}(\mathbf{y} - \mathbf{X}\theta)^T(\mathbf{y} - \mathbf{X}\theta) = 0 \iff$$
$$\mathbf{X^T}(\mathbf{y} - \mathbf{X}\theta) = 0 \iff \theta = \left(\mathbf{X^T X}\right)^{-1}\mathbf{X^T y} \tag{30}$$

The latter is a so-called "normal equation". It is very convenient to use allowing to obtain solution in one step. This contrasts to iterative methods of solving optimization problem, where you would need to normalize independent variables, calculate gradient of cost function $J(\theta)$ and searching for optimal step size.

Example of approximation of eigenvectors with 3^{rd} order polynomials below:



References

[1] Paul Wilmott, *Paul Wilmott on Quantitative Finance*, 2nd Edition, 2006.

[2] Steven E. Shreve, *Stochastic Calculus for Finance*, 2nd Edition, 2008.

[3] Trevor Hastie, *The Elements of Statistical Learning*, 2nd Edition, 2010.

[4] John C. Hull, *Options, Futures and Other Derivatives*, 5th Edition, 2003.

[5] Yuh-Dauh Lyuu, *Financial Engineering and Computation*, 2002.

[6] Damiano Brigo, Fabio Mercurio, *Interest Rate Models - Theory and Practice*, 2nd Edition, 2006.

[7] Mary Jackson, Mike Staunton, *Advanced Modelling in Finance using Excel and VBA*, 2002.

[8] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recepies*, 3rd Edition, 2007.

[9] Daniel J. Duffy, *Finite Difference Methods in Financial Engineering*, 2006.