EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Assignment #1:: The Explicit Finite Difference Method

B473 Numerical Methods in Finance
Prof. Dr.-Ing. Rainer Schöbel

Winter Term 2016-2017

Konstantin Smirnov
Student-ID: 3980253
Economics and Finance
Tübingen, December 22, 2016

List of Figures

Contents

1 Introduction

In this paper we examine how the Explicit Finite Difference Method is used to solve numerically for the Black-Scholes Merton partial differential equation (PDE) with the focus on an European put and a Backspread with Calls. Generally speaking the Finite-Difference methods uses a Taylor series expansions in order to approximate the partial derivatives terms of the PDE (Brandimarte, 2006). Primarily this paper discusses the theoretical framework and which problems can occur using this method. After that we will discuss stability and convergence of an European put in more detail. Each chapter also contains a technical part where the implementation of the algorithms in *MATLAB* are discussed.

2 Explicit Finite Difference Method for an European Put

Our Black-Scholes-Merton-PDE is defined as:

$$\frac{1}{2}\eta^2 S^2 F_{SS}(S,T) + (r-q)SF_S(S,t) - rF(S,t) + F_t(S,T) = 0. \tag{1}$$

where η^2 is the volatility, S is the stock price, r is the risk-free rate, q is the continuous payed dividend and F(S,T) is the payoff function of the option depended of the stock price S and time to maturity T. Since we are interested in the values of an European put option, we use the following payoff function (terminal condition):

$$F(S,T) = max(E - S(T), 0) \tag{2}$$

where S is the stock price at time-to-maturity T and E as the strike price of the option.

2.1 Theoretical Framework

2.1.1 Defining a Grid

To solve the PDE with the Finite Difference method, we have to set up a grid first by discretizing our input variables S and T (Hirsa, 2013). Hence, we define our input variables as

$$x = S = ih, \text{ where } i \in [0, M]$$

and

$$\tau = T - t = nk, \text{ where } n \in [0, N]$$

where i and n are index numbers, and h and k are step sizes with respect to the stock price and time. Hence one can define the rectangular grid as

$$x \in [0, x_{max}], \text{ with } x_{max} = M \cdot h$$

and

$$\tau \in [0, T], \text{ with } T = N \cdot k$$

Discretizing the PDE of the European put for (1) at the points x and τ leads to the following equation:

$$a(i,n)u_{xx}(i,n) + b(i,n)u_x(i,n) + c(i,n)u(i,n) - u_\tau(i,n) = 0 \tag{3}$$

with

$$\begin{aligned} a(i,n) &= \frac{1}{2}\eta i^2 h^2 \\ b(i,n) &= (r-q)ih \\ c(i,n) &= -r \end{aligned}$$

and

$$u[x,0] = max(E - ih, 0)$$

2.1.2 Difference Approximation: Explicit Scheme

In order to approximate the discretized differential equation (3) with the explicit scheme, we use following Taylor series expansions in order to approximate the partial derivatives (Brennan and Schwartz, 1978):
The central approximation of the gamma term Γ:

$$[U_{xx}]_i^n = \frac{1}{h^2}(U_{i+1}^n - 2U_i^n + U_{i-1}^n) + \mathcal{O}(h^2) \tag{4}$$

The central approximation of the delta term Δ:

$$[U_x]_i^n = \frac{1}{2h}(U_{i+1}^n - U_{i-1}^n) + \mathcal{O}(h^2) \tag{5}$$

and the forward approximation for the theta term Θ:

$$[U_\tau]_i^n = \frac{1}{k}(U_i^{n+1} - U_i^n n) + \mathcal{O}(k) \tag{6}$$

Note that the $\mathcal{O}$ terms are converging with the speed of h^2 or k to zero. Plugging these approximations (4),(5), and (6) into our discretized PDE (3), we get following equation:

$$\frac{a}{h^2}(U_{i+1}^n - 2U_i^n + U_{i-1}^n) + \frac{b}{2h}(U_{i+1}^n - U_{i-1}^n) + cU_i^{n+1} - \frac{1}{k}(U_i^{n+1} - U_i^n n) = 0$$

Note that compared to (3), the approximation of *u(i,n)* does not have to be taken into account because the error of this adjustment tends to zero with increasing grid points (Brandimarte, 2006).
Because we need U_i^{n+1} we rearrange this equation to get the explicit solution[1] of the explicit difference approximation at the grid points $x = ih$ and $\tau = nk$:

$$U_i^{n+1} = d_1(i,n)U_{i-1}^n + d_2(i,n)U_i^n + d_3(i,n)U_{i+1}^n \tag{7}$$

with

$$\begin{aligned} d_1(i,n) = (\frac{1}{1-ck})(a\frac{k}{h^2} - b\frac{k}{2h}) &= (\frac{1}{1+rk})(a\frac{1}{2}\eta^2 i^2 k - \frac{1}{2}(r-q)ik) \\ d_2(i,n) = (\frac{1}{1-ck})(-a\frac{2k}{h^2} + 1) &= (\frac{1}{1+rk})(-\eta^2 i^2 k + 1) \\ d_3(i,n) = (\frac{1}{1-ck})(a\frac{k}{h^2} + b\frac{k}{2h}) &= (\frac{1}{1+rk})(a\frac{1}{2}\eta^2 i^2 k + \frac{1}{2}(r-q)ik) \end{aligned}$$

[1] Note that these approximations are still independent of the time index n.

2.1.3 Boundary Conditions

One major aspect of the finite difference method is the definition of the boundaries. These points are the only values which we assume to be known. In our BSM-PDE grid we basically define the initial condition (lower bound) at maturity (n=0), and additionally the 'left-side' and 'right-side' boundaries at S_{min} (i=0) and S_{max} (i=M) respectively (Hirsa, 2013).
Since our grid starts at maturity we define our grid backwards, starting at maturity. Our initial condition is therefore simply the payoff function of the European put (2) since we know its value at maturity:

$$U_i^0 = max(E - ih, 0)$$

To establish the left-side and right-side boundaries, our assumptions have to be more precise. Therefore we use either a *Dirichlet* or a *von-Neumann* boundary condition.
For the left-side boundary at i=0 we establish our assumption with a Dirichlet condition which specifies the values at the border of the grid points directly. Assuming no-arbitrage in the market one can show that the maximum value of an European put can be described as the discounted exercise price at a certain time level, which we use as our left-side boundary of our grid (Hirsa, 2013):

$$U_0^{n+1} = Ee^{-r(n+1)k}$$

For the right side boundary at i=M we use a von-Neumann condition which specifies a partial derivative at the boundary. In our case we assume that at the maximum stock price level M, the partial derivative with respect to the stock price Δ is 0. Hence:

$$[U_x]_M^n = \frac{1}{2h}(U_{M+1}^n - U_{M-1}^n) + \mathcal{O}(h^2) = 0$$

As you can see at this point, the central approximation of our Δ contains a so called "ghost-point" M+1 since it lays outside the grid. But since we assume that at the boundary our derivative equals to 0, the point next to our boundary point should als have approximately a slope of 0. Mathematically speaking, assuming $\mathcal{O}(h^2)$ converging to 0, we consequently get:

$$U_{M+1}^n = U_{M-1}^n$$

Plugging this into our adjusted PDE (7), we get our right-side boundary condition at i=M defined as

$$U_M^{n+1} = (d_1 + d_3)U_{M-1}^n + d_2 U_M^n$$

Taking everything together we hence get following boundary conditions of our grid:

$$\begin{aligned} &U_i^0 = max(E - ih, 0) \qquad\qquad for \quad all \quad i \in [0, M] \\ &U_0^{n+1} = Ee^{-r(n+1)k} \\ &U_M^{n+1} = (d_1 + d_3)U_{M-1}^n + d_2 U_M^n \end{aligned}$$

2.2 Discussion of Results

In our computation we used following parameters:

$$\begin{aligned} S_{min} &= 200 \qquad & M &= 60 \\ S_{max} &= 1200 \qquad & N &= 400 \\ E &= 200 \\ T - t &= 1.00 \\ r &= 0.04 \\ q &= 0.06 \\ \eta &= 0.25 \end{aligned}$$

In order to validate the explicit-scheme method we need a benchmark. In our case we use the closed-form solution of an European put:

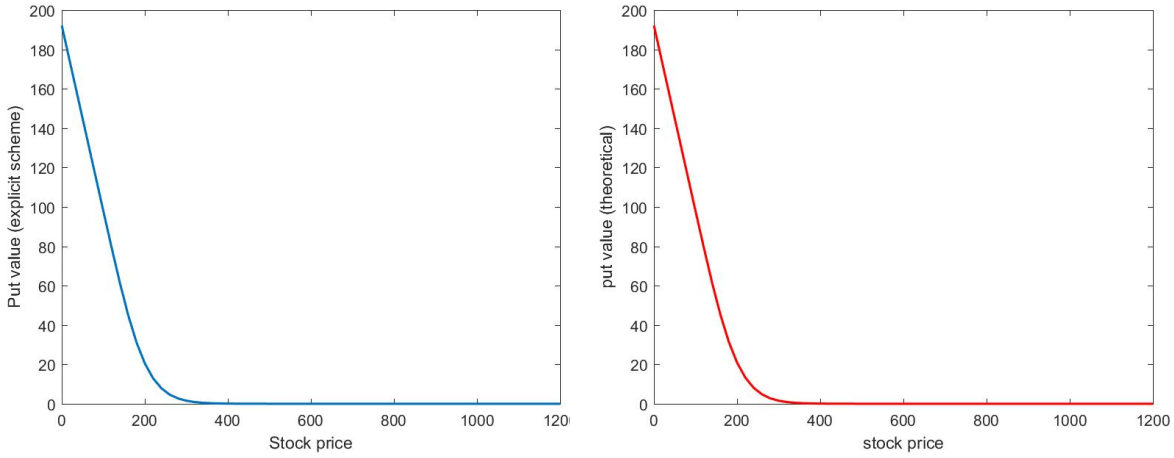
$$BSMP(S, T-t) = Ke^{-r(T-t)}N(-d_2) - Se^{(h-r)(T-t)}N(-d_1)$$

with

$$d_1 = \frac{ln(\frac{S}{X}) + (h + 0.5\eta^2)(T-t)}{\eta\sqrt{T-t}}$$

and

$$d_2 = d_1 - \eta\sqrt{T-t}$$



Explicit Scheme Theoretical

Figure 1: European put value for different S: Explicit Scheme vs. Theoretical

The left plot of figure 1 shows the value of an European put for different stock price levels with a time to maturity of 1 with respect to the other parameters and step-sizes calculated with the explicit scheme algorithm. The right plot depicts the theoretical computation (benchmark). As it can clearly be seen, the values only diverge minimally in those illustrations. For a stock price of 200 we get an explicit finite difference value of 20.5116 and for the closed-form solution we get an value of 20.8886.
In order to get a closer examination on the divergence of the explicit scheme method, the

approximated values will directly be compared with the theoretical values. We define this divergence from now on as our error term:

$$Error = u_0(M, N) - BSMP$$

where u_0 and the BSMP are vectors which contain all calculated put values solved with the explicit scheme or respectively with the closed form solution, for different combinations of S=ih and (T-t)=nk.

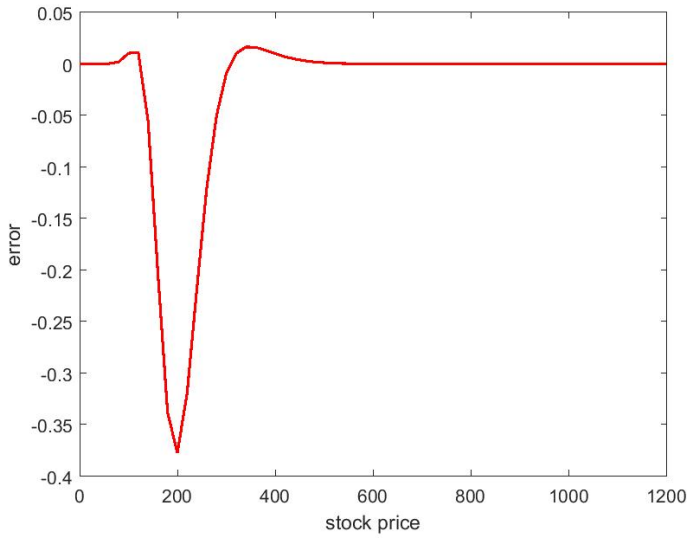


Figure 2: European Put: Error vs. Stock Price

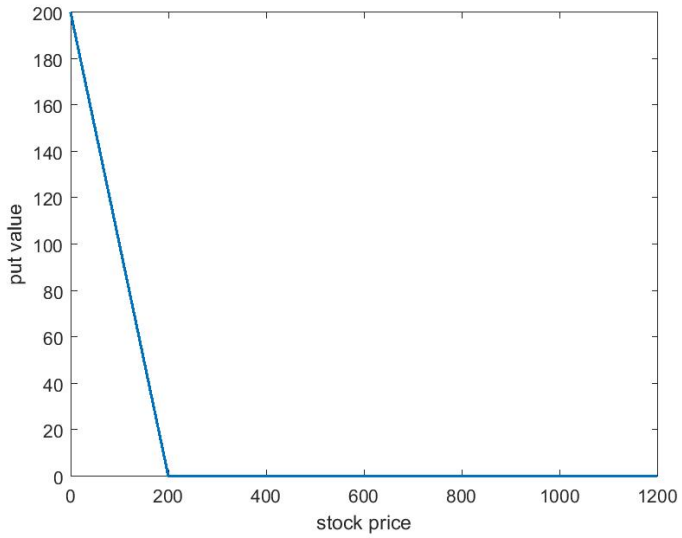


Figure 3: Payoff function of an European put

In figure 2, the errors for different levels of S (from 0 to 1200) with a time to maturity of 1 are illustrated. One can see that the biggest deviation of 0.3770 can be found at a stock price level of S = 200 of which is equal to our strike price of E = 200.
The reason for this is illustrated in the payoff function of the European put in figure 3. When

the option is at-the-money, discontinuity can be observed. At this point it is not possible to take the derivative and hence we get poor approximations at these grid points. As we have seen in the derivations in the theoretical part, the error vanishes with a speed of the step-size h^2 at each following approximation.

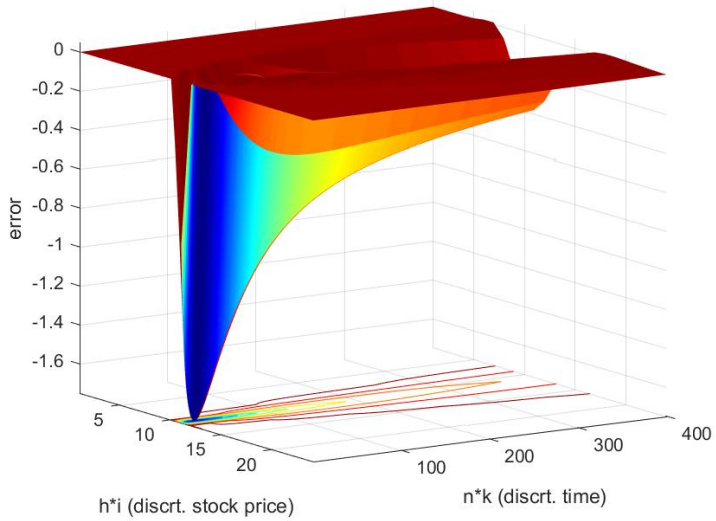


Figure 4: European Put: Error vs. Stock Price vs. Time to Maturity

In figure 4 we extend our analysis to a 3-dimensional case where we add time-to-maturity as another varying factor. Note that we plot in a discretized framework, where we have $Mx = 60$ steps in total on the x-axis, where each stock price step i has the size of $h = 20$. On the time level (y-axis) we have $Nt = 400$ steps where each time step k has a size of $n = 0.0025$.[2] Also here we can clearly see that the biggest deviations can be found at the step i = 11[3] when the option is at the money for the same reason as just discussed.
Also relatively big deviations can be observed when the option has a short time-to-maturity level of n. The reason for this can be explained from an economic perspective. An option at the money with a very short-time to-maturity has still a value, even tough it is really small. Note that due to our terminal condition, at maturity n=0 no errors occur since the value of the option is certain in both cases. Hence for the next time level n=1 at S=200 we use the values at time level n=0 at S = 180, S = 200 and S = 220 which are 20, 0 and 0 respectively. Our approximation is hence not precise enough to approximate the upside probability of the option at that certain time step. The error itself vanishes with the speed of the step-size k and takes longer to vanish compared to the error term for the stock price approximations h^2. Generally it can be said that the explicit scheme underestimates the values compared to the closed-form solution.

2.3 Technical Implementation

The main script *Task1_EuroPut.m* contains all necessary computations and plots for task 1. The script basically calls the function *Explicit_EuroPut.m* which consists of the main

[2] $h = \frac{1200}{60}$ and $n = \frac{1}{400}$.

[3] Due to reasons which will be discussed in the technical implementation, our grid has to start at n=1. So the relationship that 10*20=200 still holds.

algorithm for the explicit scheme.
To preserve the outline only the maximum stock price value *Smax*, time-to-maturity TT and the step-size parameters i and n serve as input variables. To change other input parameters one can adjust it in the algorithm function *Explicit_EuroPut.m* itself.
The algorithm basically follows all steps which I described in the theoretical framework of this chapter. First one has to define the grid by discretizing our parameters S and T-t with respect to the step-sizes h and k. Our starting point of our grid is simply the payoff-function at maturity since these values are certain (terminal condition). Keep in mind the algorithm follows a backward-induction since only the payoff-function in the future is certain.
In the next steps we first rearrange the approximations to d_1, d_2 and d_3 before approximating the grid points with the main-loop. The main-loop is a built as for-for-loop. The outer for-loop varies the time-level *nn* while the inner grid points are approximated with the inner for-loop at each stock price step i up to the right of the grid until the bound *Mx+1* is reached. After the inner for-loop is engaged, the outer for-loop jumps to the next time-level *nn+1* ('going up' on the grid) and the inner for-loop again approximates the values until *Mx+1* and so forth until the border of the upper bound *Nt+1* is reached. Now all inner grid points are filled. All values of each combination of MxN will be stored in the matrix *u1* or *u0*. It should be mentioned that in our *u0* matrix we overwrite all former approximations up to *Nt+1* since we are not interested in *u0*-matrix of different time-steps n.
The boundaries derived at the left and right-side at i=0 and i=Mx in subsection 1.1.3. can simply be included inside the outer loop since they are only depended on the time-level n. Furthermore it should be made clear that the index notation for Nt+1 and Mx+1 at the boundaries does not represent the 'ghost-points' as discussed in the theoretical framework of this chapter. Since *MATLAB* starts computations only with positive integers, the boundaries had to be adjusted to a level of Mx+1 or Nt+1 respectively.
Since we are only interested in the errors at each n and k, the values are stored in an own matrix called *Error3D*. As discussed in the theoretical framework, our benchmark function is the theoretical value of the European put, which is calculated with an own function called *BSMP.m*.[4] The final vector of our main-loop *u0* represents the final European put value for a time-to-maturity level of 1. Additionally the functions computes the maximum errors further computations an illustrations.
Taking everything into account, as an output of our main algorithm we get the error vectors *Error2D_EuroPut*, *Error3D_EuroPut*, as well as the errors-vector *ME_Put_2D* for time-to-maturity of 1 and *ME_Put_3D* for the whole grid.
In order to plot, the output additionally contains the discretized stock price vector S, the final u0-vector ('upper row of the grid') and the theoretical European put vector *TheoPut*. All illustrations are simply plotted with the *plot* function with adjusted settings.

3 Explicit Finite Difference Method for a Backspread with Calls

In this section we analyze how the value of a Backspread with Calls (BSC) with the Explicit Finite Difference Method can be calculated. A Backspread with Calls is a strategy consisting of several call options with the same underlying, where the long position of the call with the higher strike is twice as high as the short position of the call with the lower strike. The

[4]Note that in order to use the BSMP.m function we additionally have to calculate our normal-distribution value with the *N.m function*.

BSM-PDE of chapter 1 also holds for the case of a BSC:

$$U_i^{n+1} = d_1(i,n)U_{i-1}^n + d_2(i,n)U_i^n + d_3(i,n)U_{i+1}^n \tag{1}$$

As our terminal condition we now use the payoff function of a Backspread with Calls:

$$F(S,T) = 2 * max(x - E_2, 0) - max(x - E_1, 0) \tag{2}$$

We use the same parameters as in chapter one, except of the exercises prices which are assumed to be $E_1 = 160$ and $E_2 = 220$ now . Also we assume in this task that no dividends are being payed (q=0).

3.1 Theoretical Framework

The theoretical framework is established in the same way, we just have to establish feasible and sufficient boundary conditions for our BSC payoff function. Again we have to derive boundaries for the initial condition (lower boundary) at n=0, as well as for the left-side i=0 and right-side i=M of our grid.
As in the case of the European put, our terminal condition is simply the payoff function at maturity. Hence:

$$U_i^0 = 2 * max(x - E_2, 0) - max(x - E_1, 0) \tag{3}$$

To establish the left-side and right-side boundaries we use von-Neumann conditions since they are more precise.
For the left-side boundary we assume that at i=0 our Δ is 0:

$$\frac{\delta u(0,\tau)}{\delta x} = 0$$

Applying the central approximation of the delta term Δ at the grid point i=0, we get:

$$[U_x]_0^n = \frac{1}{2h}(U_{-1}^n - U_1^n) + \mathcal{O}(h^2)$$

As in chapter 1, one can show that our 'ghost point' at $U_{-1}^n = U_1^n$. Plugging this into (1) and rearrange with respect to U_0^{n+1} we hence get for our left-side boundary following equation:

$$U_0^{n+1} = (d_1 + d_3)U_1^n + d_2 U_0^n \tag{4}$$

For the right-side boundary we assume that our Γ is 0:

$$\lim_{x \to \infty} \frac{\delta^2 u(x,\tau)}{\delta^2 x} = 0$$

Using the central approximation for Γ at i=M we get :

$$[U_{xx}]_M^n = \frac{1}{h^2}(U_{M+1}^n - 2U_M^n + U_{M-1}^n) + \mathcal{O}(h^2)$$

Again one can show that our ghost-point at M+1 can be described as:

$$U_{M+1}^n = 2 * U_M^n - U_{M-1}^n)$$

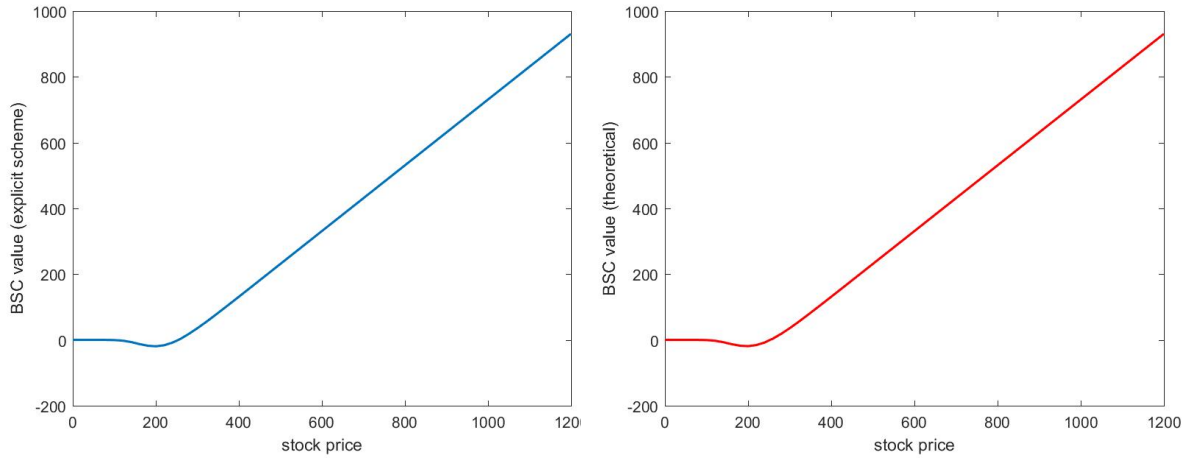
Plugging this into (1) and rearrange with respect to U_M^{n+1} we get for our right-side boundary condition i=M following equation:

$$U_M^{n+1} = (d_1 - d_3)U_{M-1}^n + (d_2 + 2d_3)U_M^n \tag{5}$$

Summarized our boundaries (3), (4), (5) are:

$$\begin{aligned} U_i^0 &= 2 * max(x - E_2, 0) - max(x - E_1, 0) \\ U_0^{n+1} &= (d_1 + d_3)U_1^n + d_2 U_0^n \\ U_M^{n+1} &= (d_1 - d_3)U_{M-1}^n + (d_2 + 2d_3)U_M^n \end{aligned}$$

3.2 Discussion of Results



Explicit Scheme Theoretical Payoff-Function

Figure 5: European BSC value for different S and T-t=1: Explicit Scheme vs. Theoretical

As in chapter one we use the same analysis in order to check how valid our approximations are. In figure 5 the explicit finite difference BSC values are compared with the theoretical BSC values. Again divergence is not directly observable. To get a more detail view on our analysis we calculate the error where the theoretical BSC price is our benchmark:

$$Error = u_0(M, N) - (2 * BSMC_2 - BSMC_1) \tag{6}$$

where $u_0(M, N)$ contains the BSC prices for different stock levels calculated with the explicit scheme, and $BSMC_1$ $BSMC_2$ contains the theoretical values of the BSC with strike at X_1 = 160 and X_1 = 220.
For a stock price of S=200 the explicit FD BSC value is -19.4825, while the theoretical value is -19.0178. It should be made clear that negative values are in our case possible because we use a combination of calls as a strategy with a relatively wide strike price range between both calls.

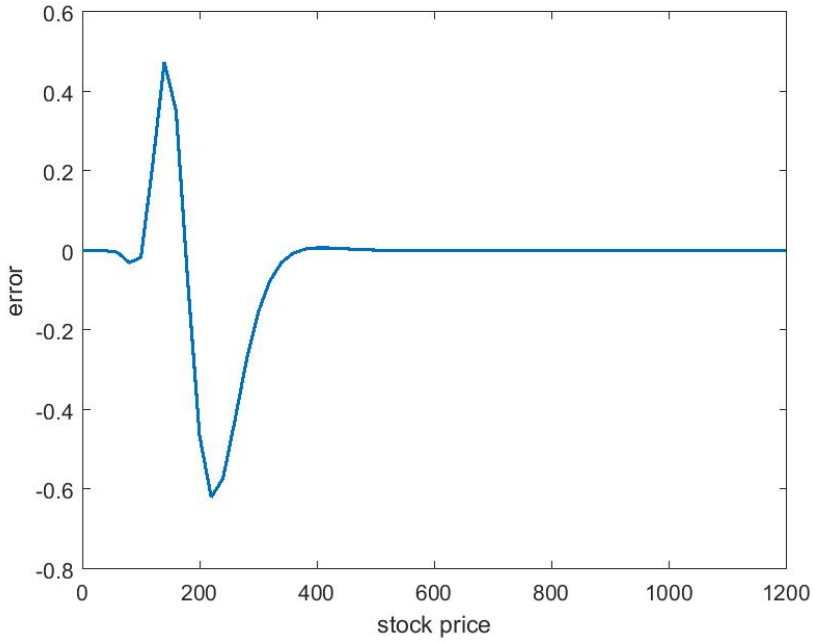


Figure 6: Backspread with Calls (European): Error vs. Stock Price

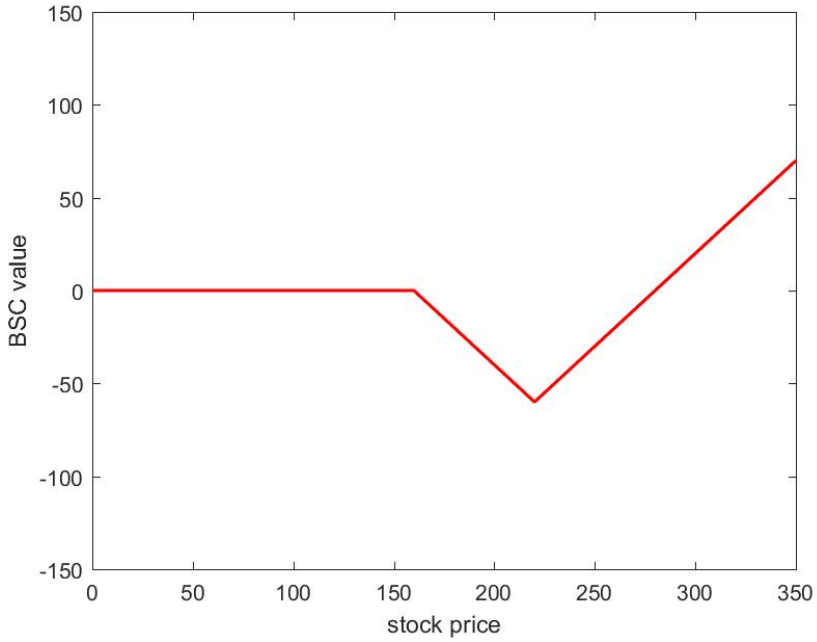


Figure 7: Payoff-Function of a Backspread with Calls with $X_1 = 160$ and $X_2 = 220$ (European)

As it can be seen in these illustrations, the error terms of the BSC follow a similar behavior which was already discussed in the previous chapter.
Figure 6 shows that the errors occur now in two areas at these points where our options are at-the-money. As you can see, the payoff function is again discontinuous at two points for the strike rates of $X_1 = 160$ and $X_2 = 220$ (see 7). The error converges to zero again with a speed of the stock step-size h^2.

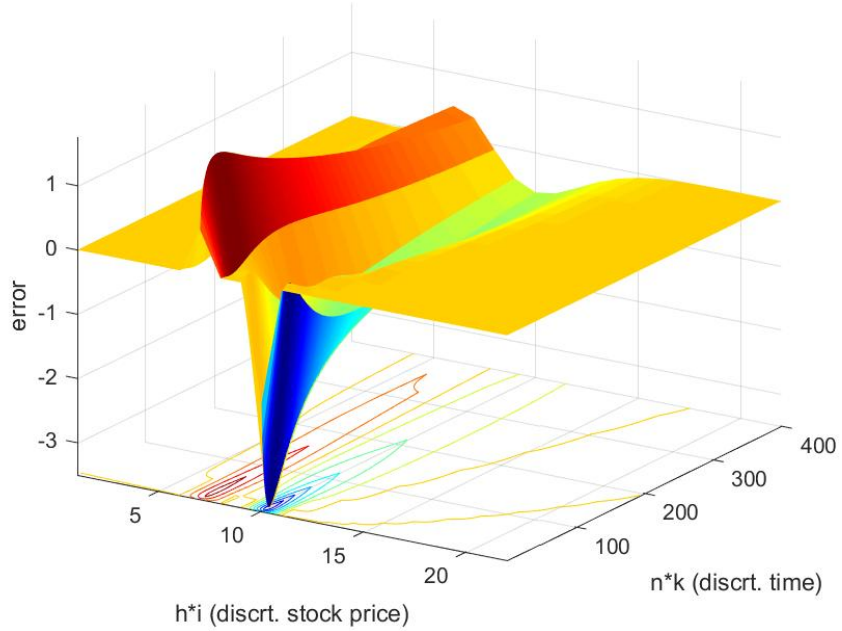


Figure 8: Backspread with Calls (European): Error vs. Stock Price vs. Time to Maturity

The errors also tend to increase if time-to-maturity decreases as it is illustrated in figure 8. The approximations are not precise enough due to the same reasons as discussed in chapter one. Also here the error converges to zero again with a speed of the time step-size k and takes longer to vanish compared to h^2.
Even tough it is not directly observable because we use combination of calls with relatively large deviation between the strike rates, in general it can be said that the explicit finite difference framework tends to overrate European calls.

3.3 Technical Implementation

All computations and plots regarding this task can be called in the *Task2_EuroBSC.m* script in which the main algorithm *Explicit_EuroBSC.m* is called. The technical implementation of this algorithm is basically equal to the European put case in chapter 1. Only the payoff function and the boundaries had to be adjusted.
The output is equal to the European Put case (*u0*-matrix, error vectors etc.). Note that to compute the benchmark value we again use a self-written function *BSMC.m* to calculate the theoretical call price.

4 Stability and Convergence

In this section we analyse the stability of the explicit finite difference framework in the case of an European put option. In order to do that we check for the eigenvalues and the norm of the matrices and check how the error converges similar to the previous chapters but now for increasing step-size combinations MxN.

4.1 Theoretical Framework

In order to investigate the stability we first have to derive an amplification matrix **A** for the specific MxN combination, consisting of the grid-points d_1, d_2 and d_3 for different stock-prices S and time levels $T-t$.[5] For every MxN combination we get one matrix which looks like:[6]

$$\mathbf{A} = \begin{bmatrix} d_2 & d_3 & & & & \\ d_1 & d_2 & d_3 & & & \\ & d_1 & d_2 & d_3 & & \\ & & & \ddots & & \\ & & & d_1 & d_2 & d_3 \\ & & & & d_1^* & d_2 \end{bmatrix}$$

In order to check the stability of the system we have to calculate the the eigenvalues λ of each **A**. Stability is given if (Fusai, Gianluca and Roncoroni (2008)):

$$max|\lambda_i| < 1 \tag{1}$$

Note here that in (1) we are only interested in the biggest maximum eigenvalue λ_i.
Furthermore we have to check if the spectral radius conditions[7] holds by measuring the infinity-norm of **A**:

$$max|\lambda_i| \leq ||\mathbf{A}||_\infty \tag{2}$$

Furthermore we examine stability by computing the true error for different size-combinations of MxN to see how these error develop. Again our benchmark is simply the theoretical European put price as discussed in chapter 1:

$$MaxError = max(|u_0(M,N) - BSMP(S,T-t)|)$$

We are just interested in 'feasible frontier' combinations where the system starts to be feasible. Hence we only need the errors of MxN combinations with respect to the analysis of the eigenvalues and the norm in (1) and (2). Note again that $u_0(M,N)$ and BSMP(S,T-t) are vectors, where every row represents a different stock price for a fixed time level. But since we are interested in the maximum error of the difference we again get a scalar as a result.

4.2 Discussion of Results

In figure 9 the stability of the explicit scheme of an European put is illustrated for different grid sizes MxN with M_{max}= 400 and N_{max} =2500. The green area shows the combinations where the explicit scheme is stable with eigenvalues smaller then 1 while the red area represents combinations of eigenvalues ≥ 1. The feasible frontier is represented as the black line. Every combination on that line shows the combination of MxN with an eigenvalue as close as possible to 1. Every combination above would be feasible, but inefficient regarding computational time while all MxN combinations below would be unstable and hence unfeasible.
In figure 10 one can see semi-log and log-scaled plots of maximum absolute errors for the grid sizes MxN at T-1=1 on the feasible frontier. As it can clearly be seen especially in the semi-log case, with increasing MxN the error, in the limit, tends towards 0 since a larger grid

[5] **A** is derived from the matrix notation of the grid, see Fusai, Gianluca and Roncoroni (2008)
[6] Note that due to the von-Neumann boundary condition we get for $d_1^* = d_1 + d_3$
[7] This condition is sufficient in our analysis.

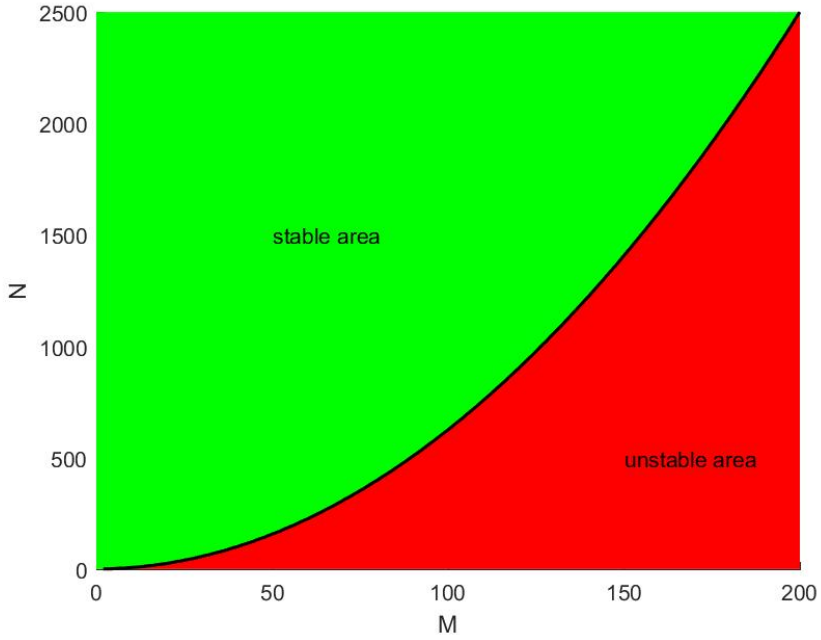
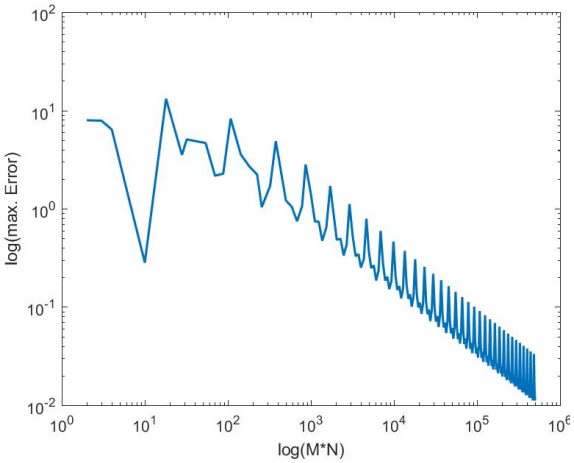
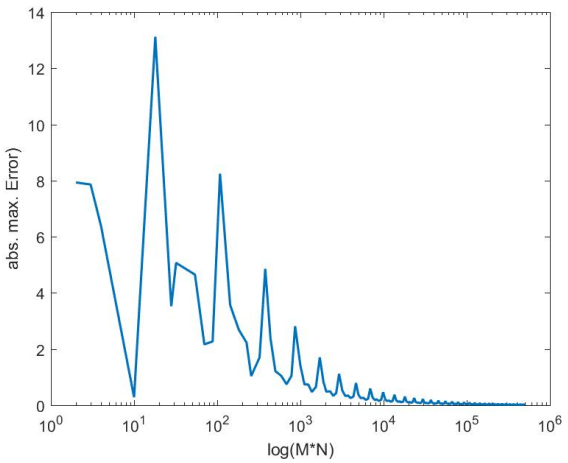


Figure 9: Combination of MxN with stable and unstable eigenvalues

size implies more grid points. Hence the approximations are more precise. Same holds for the absolute maximum errors if you look at the whole grid as it clearly can be seen in figure 11. The error just does not converge as fast to 0 as the errors at T-t=1 due to the reasons we discussed in the previous chapters.
Additionally it approves our theoretical derivation and observation of chapter 1. For the approximations terms of the stock price Δ and Γ, the errors converge with a speed of h^2 to 0. On the contrast, for the approximation of the Θ term, the error converges only with a speed of k to 0. Hence more grid points are required for the time space N relative to the grid points of the stock price space M to increase precision. Keep in mind that bigger grids can increase computational time significantly.

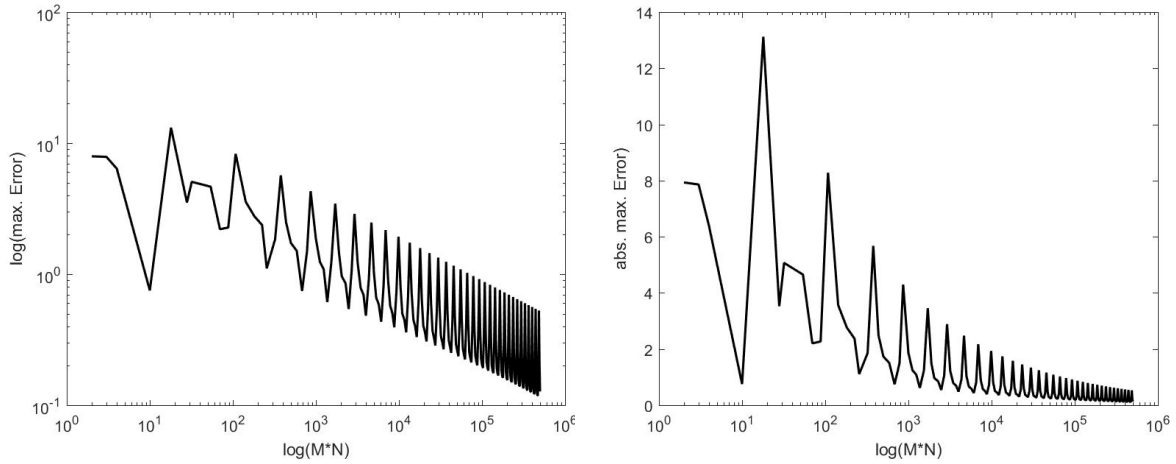


log. MxN combinations vs. log. absolute errors



log. MxN combinations vs. absolute errors

Figure 10: Feasible Frontier: MxN combinations vs. absolute errors at T-t=1



log. MxN combinations vs. log. absolute errors log. MxN combinations vs. absolute errors

Figure 11: Feasible Frontier: MxN combinations vs. absolute errors for the whole grid

4.3 Technical Implementation

In order to check for stability and consistency we setup a script *Task3_Stability* in which the *Explicit_stability.m* function,the main algorithm of this task, is called. Since we are interested in the stability of different MxN combinations or to be more precise, in the stability of different sizes of the grid, we use start and end points of as well as the step-sizes for the stock price and the time levels as an input of our stability function. Note that only positive integers are valid as an input for the size-parameters since we need the location of the rows and columns which are represented as positive integers in *MATLAB*.
To analyse the behavior of different MxN combinations our algorithm contains a for-for-loop, the inner for-loop takes the varying time level N into account, before the outer for-loop increases the stock price S. For each combination or loop step, our d_1, d_2 and d_3 have to be calculated as described in the theoretical framework for each specific $\mathbf{A}$ by using the function *d_calc.m*. Since we are interested in the stability of an European put, the algorithm of task 1 *Explicit_EuroPut* was adjusted in a sense that as an output we get the vectors for d_1, d_2 and d_3 for each combination.
Since $\mathbf{A}$ is tridiagonal we call another function *tridiag.m* which transforms the d_1, d_2 and d_3 into the desired structure.
Now the eigenvalues and the infinity-norm can be calculated with respect to our conditions described in the theoretical framework of this chapter. The eigenvalue vectors are calculated simply with the integrated *MATLAB* command *eig* where we just store the maxmium eigenvalue, in which we are interested. The 2-infinity norm is also simply calculated with an integrated *MATLAB* command *norm*.
Now all mandatory values are achieved to check if our conditions from the theoretical part are fulfilled. To get the specific MxN combinations all values are stored in the matrix called *stability*, which is also defined as the output. First we just store the stock price length inside the first column of the *stability* matrix.
In order to to check if the conditions are fulfilled for this specific MxN combination, we use an if-statement inside the loop with a predefined count-variable which counts for the n (time level). Remarkably the if-statement checks if the eigenvalues are just smaller as 1 (and not

bigger then 1, as one might suggest since this our actual condition). Since the algorithm starts with calculation for small time levels n in the first row, the eigenvalues are relatively large in the first rows. If the condition gets satisfied for this specific MxN combination, the loop 'breaks' just at the step where the eigenvalue just dropped below 1. Our count variable N will now be stored in the second column of the *stability* matrix. Note here that since we are only interested in the 'feasible' frontier we just store the values of N where this condition just gets fulfilled. To improve performance, we jump to the next level *nn* of our time loop since the dynamic of eigenvalues follows a linear relationship in every column.
In short, the algorithm searches for the specific time level N for given S and stores the count variable n into the second column of the *stability* matrix. The first two columns now represent the feasible frontier combinations. Additionally we multiply these combinations (element wise) and store them in the third column for the purpose of plotting later on.
Since we are also interested in the maximum absolute errors of the computed feasible frontier combinations MxN, the algorithm function additionally calls the function of task 1 *Explicit_EuroPut* in order to compute the absolute maximum errors for the feasible frontier which is stored in the the fourth column and fifth column of the *stability* matrix.
Summarized, the *stability* matrix now consists of five columns which contains all values of the feasible stability frontier: the first contains the stock price S, the second the time level N , the third the MxN multiplication, the fourth the maximum absolute error at T-t=1 and the fifth for the maximum absolute error of the whole grid.
Back in the main-script *Task3_Stability* we plot our results. First we want a stability plot of the MxN combinations (with M on the X-axis and N on the Y-axis). To fill the stable and unstable area with colors we used the *patch* command. Therefor one has to define polygon grids to specify the area. As recommended in this task of the assignment we plot the logarithmic feasible frontier (log(MxN) vs. log(Absolute Error)) with the help of the *loglog* command, as well as the 'semi-logarithmic' (log(MxN) vs. Absolute Error) with help of the *semilog* command.

5 References

Brennan, Michael, and Eduardo Schwartz. Finite Difference Methods and Jump Processes and the Pricing of Contingent Claims: A Synthesis. *Journal of Financial and Quantitative Analysis 13*, 1978, 461-474.

Brandimarte, Paolo. *Numerical Methods in Finance and Economics - A MATLAB-Based Introduction*, 2^{nd} ed., John Wiley & Sons, Hoboken NJ, 2006.

Fusai, Gianluca, and Andrea Roncoroni. *Implementing Models in Quantitative Finance: Methods and Cases.* Springer-Verlag, Berlin Heidelberg New York 2008.

Hirsa, Ali. *Computational Methods in Finance.* Chapman & Hall/ CRC Press, Boca Raton FL, 2013.