Finite Difference Methods for Financial Partial Differential Equations

Christoffer A. Brevadt (pnk332),

Jannick Pedersen (pvm656),

Victor H. N. Poulsen (nmt831)

Januar 16, 2023

Introduction

In this project, we will work with different finite difference methods to solve financial partial differential equations. We will assume throughout this project that the underlying stock follows a simple Black-Scholes model:

$$\frac{dS_t}{S_t} = \mu dt + \sigma dW_t,$$

where W_t is a Wienerprocess. We have created a method called `kBlack::fdRunner` in the existing Saxo codes. This method calculates the price for either a put-, call- or digital option using finite difference. The method is heavily inspired by the existing `kBachelier::fdRunner` method. The `kBachelier::fdRunner` method has the same functionality as our new method but is assuming the underlying asset follows a Bachelier model. One major difference between the two methods is that the `kBlack::fdRunner` does not use uniform grid spacing. This is due to the exponential behavior of the Black-Scholes dynamics. Furthermore, is the method `kBlack::fdRunner` updated to run for the Black-Scholes model by updating the parameters when using the `fd.rollBwd` method.

Unless otherwise stated we will use the following parameters when running the finite difference code:

Model parameters			
s_0	1	eur amr	0
r	0.04	smooth	0
μ	-0.03	θ	0.5
σ	0.2	wind	0
T	5	num std	5
K	1.025	num t	800
digital	0	num x	98
put call	1		

All codes and Excel files can be accesed via the follwing link:
https://github.com/VNemirov/SaxoPUKCompFin.

Question 1

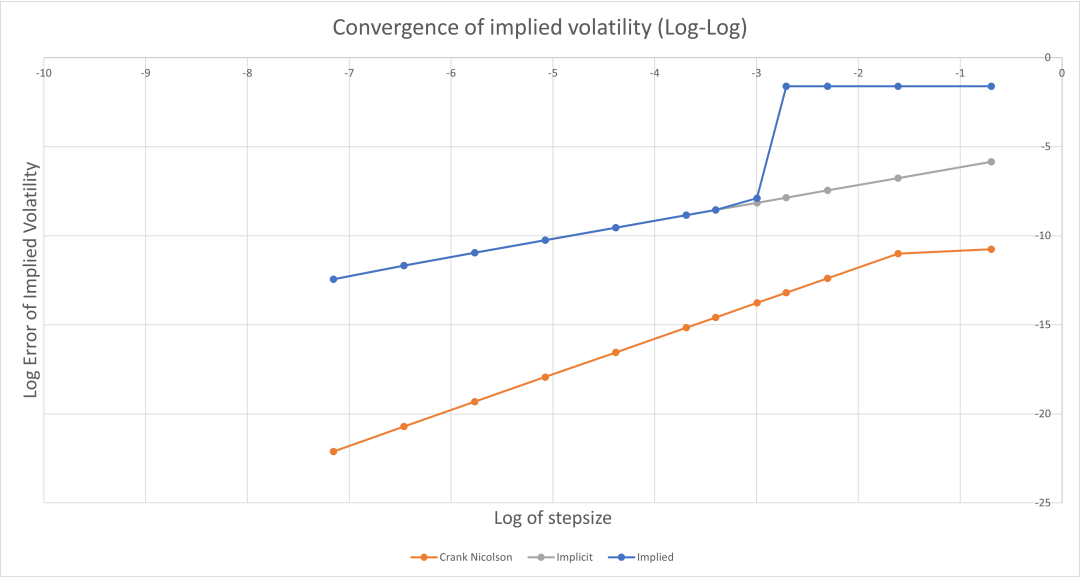
We will in this question find the order of convergence for each of the three finite difference methods: the implicit-, explicit- and Crank-Nicolson method. To do this we have calculated the price for a European call option for several time steps being $m = 10, 25, 50, 75, 100, 150, 200, 400, 800, 1600, 3200, 6400$. After this, we have used the `kBlack::implied` to calculate the implied volatilities given the prices. The implied volatilities are given in the following table.

Implied volatilities				
m	Δt	Crank-Nicolson ($\theta = 0.5$)	Implicit ($\theta = 1$)	Explicit ($\theta = 0$)
10	$\frac{1}{2}$	0.20002534	0.197136166	NaN
25	$\frac{1}{5}$	0.200020781	0.198851659	NaN
50	$\frac{1}{10}$	0.200008305	0.199427066	NaN
75	$\frac{1}{15}$	0.200005994	0.199619243	NaN
100	$\frac{1}{20}$	0.200005185	0.199715401	0.200376137
150	$\frac{1}{30}$	0.200004607	0.199811604	0.200196863
200	$\frac{1}{40}$	0.200004404	0.199859722	0.200148667
400	$\frac{1}{80}$	0.200004209	0.199931921	0.200076393
800	$\frac{1}{160}$	0.200004161	0.199968029	0.200040266
1600	$\frac{1}{320}$	0.200004148	0.199986086	0.200022204
3200	$\frac{1}{640}$	0.200004145	0.199995115	0.200013174
6400	$\frac{1}{1280}$	0.200004145	0.19999963	0.200008659

One can notice that the price/implied volatility for the explicit method is unstable for a small number of time steps. From table 5 in *Fun with Finite Difference* we know that we must require $\Delta t \leq O(\Delta s^2)$ for the explicit method to be stable. This is why we do not give the implied volatilities for these time steps for the explicit method.

It is seen that all of the methods converge to the same implied volatility namely the volatility used in the model. However, the Crank-Nicolson method converges much faster than the other two methods. This means that we can find an approximation to the option price with much fewer time steps needed. We will therefore need less computational power to calculate the price and therefore save time.

To illustrate the convergence of the implied volatilities we make a log-log plot:



Please note that the last 4 points for the explicit method are set to zero due to a lack of data. It can be seen here that the Crank-Nicolson curve lies under the curve of the implicit and explicit methods. We see that Crank-Nicolson is better at all step sizes because its relative error is smaller than the implicit and explicit's relative error.

From table 5 in *Fun with Finite Difference* we expect the order of convergence of the explicit and implicit methods to be around 1 and we expect the order of convergence for the Crank-Nicolson method to be around 2. If we calculate the order of convergence for our implied volatilities we find that:

Method	$O(\Delta t^a)$
Crank-Nicolson	$a \approx 2.00$
Implicit method	$a \approx 1.01$
Explicit method	$a \approx 1.09$

As expected we see that the Crank-Nicolson method has a higher order of convergence, namely around 2, whereas the implicit and explicit methods have an order of convergence around 1.

Question 2

We will in this question find the order of convergence for the early exercise premium i.e. the difference between an American option and a European option:

$$EEP = c^{Am} - c^{E}.$$

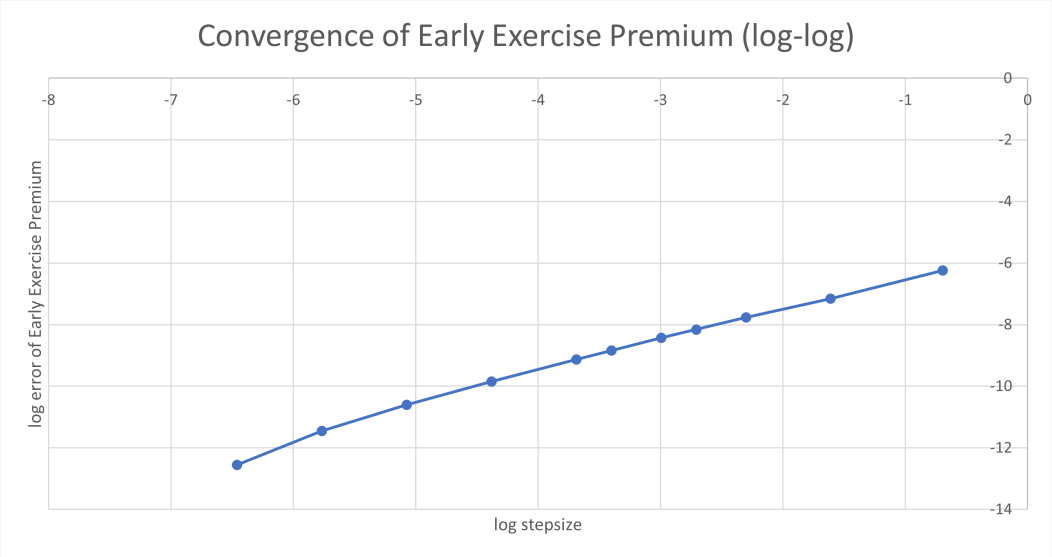
Before calculating the prices for the American option we note that the returned prices are actually for a Bermudan option and not an American option due to the discrete time steps. The American option is exercisable in every continuous-time step and is therefore worth more than the Bermudan:

$$c^{Am} \geq c^{B}(\Delta t), \quad \forall \Delta t.$$

The early exercise premiums for the different numbers of steps/stepsizes are given in the table below.

m	Δt	European Price	Bermudan Price	Early Exercise Premium
10	$\frac{1}{2}$	0.07943311	0.10100643	0.02157332
25	$\frac{1}{5}$	0.07943029	0.10216770	0.02273741
50	$\frac{1}{10}$	0.07942255	0.10252048	0.02309793
75	$\frac{1}{15}$	0.07942112	0.10265394	0.02323282
100	$\frac{1}{20}$	0.07942062	0.10272191	0.02330130
150	$\frac{1}{30}$	0.07942026	0.10279431	0.02337405
200	$\frac{1}{40}$	0.07942014	0.10283130	0.02341116
400	$\frac{1}{80}$	0.07942001	0.10288665	0.02346664
800	$\frac{1}{160}$	0.07941998	0.10291472	0.02349474
1600	$\frac{1}{320}$	0.07941998	0.10292883	0.02350885
3200	$\frac{1}{640}$	0.07941997	0.10293589	0.02351592
6400	$\frac{1}{1280}$	0.07941997	0.10293942	0.02351945

The below log-log plot of stepsize against the log error of the early exercise premium shows the order of convergence.



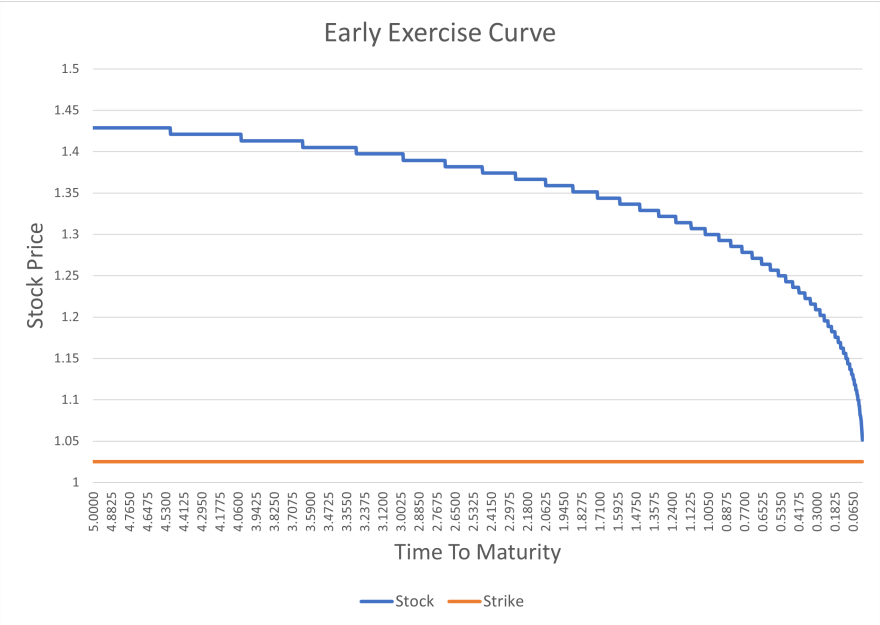
When looking at our data we see an order of convergence of $a \approx 1.06$. Therefore, we have lost one order of convergence despite using the Crank-Nicolson method.

This is because the Bermudan option price converges to the American option price with $O(\Delta t)$. This is seen when looking at the difference between the intrinsic values at the optimal stopping time for the American option, τ, and the next exercisable time for the Bermudan option, $\tau + \Delta t$, is enforced:

$$
\begin{aligned}
0 &\leq c^{Am} - c^{B} \\
&\leq \mathbb{E}\left[e^{-r\tau}\mathbb{1}_{\tau<T}\mathbb{E}_{\tau}[I(\tau) - e^{-r\Delta t}I(\tau+\Delta t)]\right] \\
&= \mathbb{E}\left[e^{-r\tau}\mathbb{1}_{\tau<T}\left[(S(\tau)-K) - e^{-r\Delta t}\mathbb{E}_{\tau}\left[S(\tau+\Delta t)-K\right]\right]\right] \\
&= \mathbb{E}\left[e^{-r\tau}\mathbb{1}_{\tau<T}[S(\tau)(1-e^{(\mu-r)\Delta t}) - K(1-e^{-r\Delta t})]\right] \\
&= \mathbb{E}\left[e^{-r\tau}\mathbb{1}_{\tau<T}S(\tau)\left(1-e^{-q\Delta t}\right) - K\left(1-e^{-r\Delta t}\right)\right] \\
&= \mathbb{E}\left[e^{-r\tau}\mathbb{1}_{\tau<T}S(\tau)(1-1+O(\Delta t)) - K(1-1+O(\Delta t))\right] \\
&= O(\Delta t)\mathbb{E}\left[e^{-r\tau}\mathbb{1}_{\tau<T}(S(\tau)-K)\right] \\
&= O(\Delta t).
\end{aligned}
$$

Because of this, the result is around one order of convergence worse.

Next, we look at the early exercise curve, which is defined by the points where the difference between the payoff and the current option price is larger than the corresponding payoff at expiry:

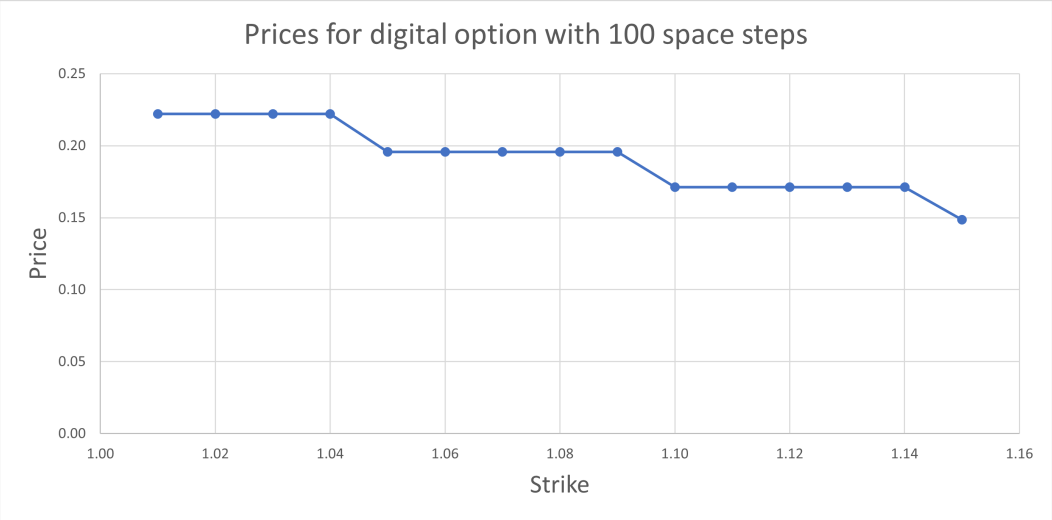


It is seen that the early exercise curve is decreasing as the time to maturity goes to zero. This is simply because the value of holding the option is decreasing as the option has less time to end up in the money. Therefore the holder of the option will choose to exercise earlier if it is profitable.

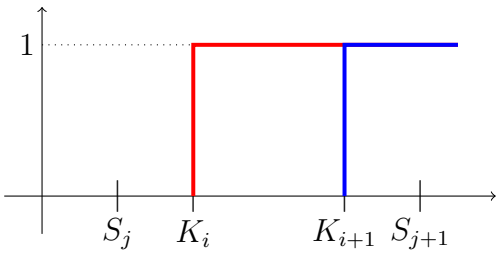
Question 3

In this question, we will fix the grid and find different prices for digital options given different strikes. In this question, we use the Crank-Nicolson method due to the discussion about convergence from question 1. We use 800 time steps since the price when using Crank-Nicolson, does not change a lot when using more time steps than this. We want the best price but also the fastest price, so this is the middle ground between getting the right price and finding it fast.

We then make the finite difference grid for $K = 1.01, 1.02, ..., 1.14, 1.15$. Doing this gives us the following prices:



Here we see that multiple different strikes have the same price. In theory, this does not make a lot of sense. The reason the prices have this stairwell shape is due to the grid and the payoff function. The payoff of a digital option is either 1 or 0. This means that at $S_i = K$ the payoff makes a jump. If the spatial steps are too large and the strikes are not separated between the spatial steps, then the payoff of different strikes will return the same payoff for different stock prices.

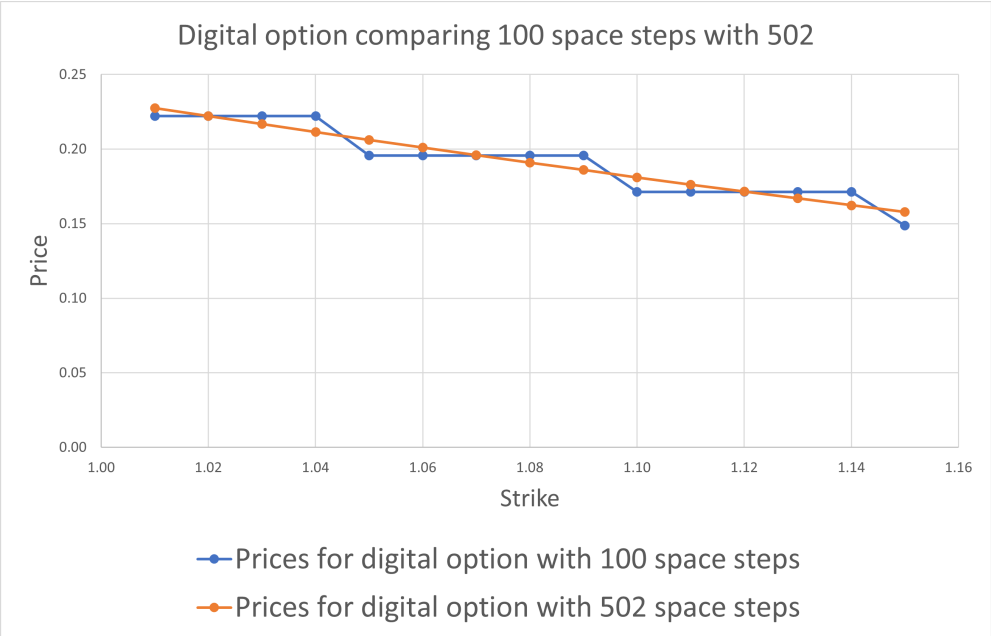


Here we see that the jumps at K_i and K_{i+1} both happens between S_j and S_{j+1}. It does however catch the different jumps at K_{i-1} meaning that this will get another price.

If we look at our constructed grid we see that multiple strikes are captured between our spatial grid points:

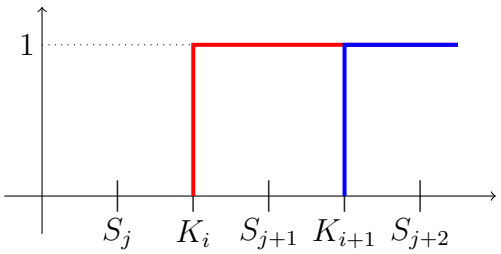
$s(i)$	$s(i+1)$	$s(i+2)$	$s(i+3)$
1	1.045736435	1.093564691	1.143580441

If we increase the number of steps to 502 we can compare the prices of the digital options that we find with the digital options from before:



Here we do not see the stairwell shape of the prices given different strikes for the increased number of steps. This is because we have decreased the step size, so that, when making

the grid, we now have a stock value between the different strikes. This can be illustrated as:

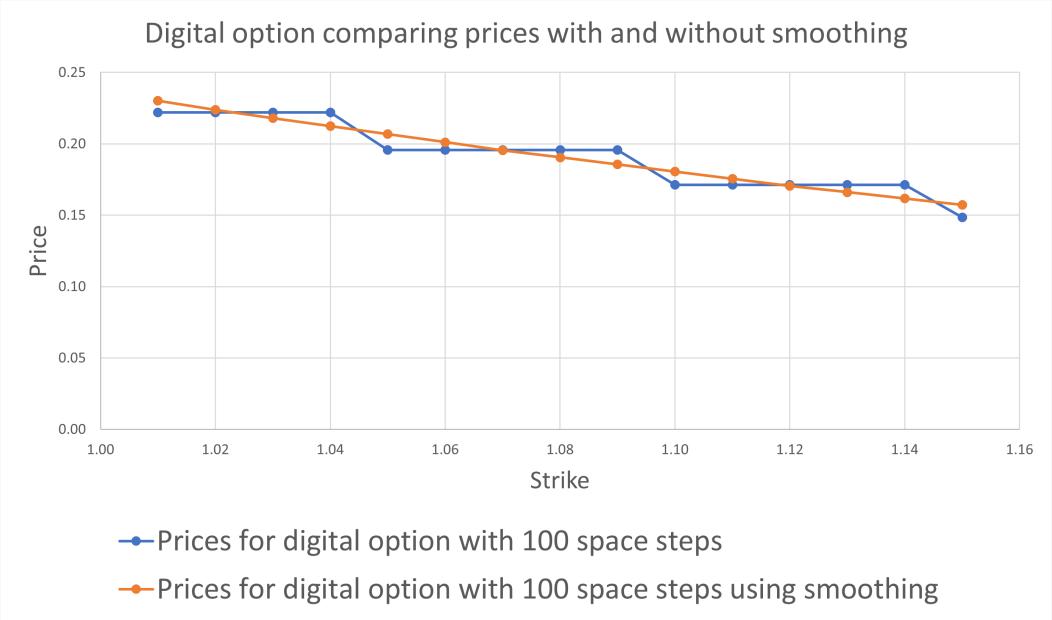


This means that there are not two jumps happening within the same interval of the stock prices. This leads to all the prices of the digital options being different. It is good to note however that when increasing the number of steps the computing time increases significantly. This means that this method gives better prices but it also increases the time it takes to calculate the prices, which is not optimal.

Another way one can improve the prices is by using smoothing. When we use smoothing we change the payoff function. The true payoff function of a digital option is $\Phi = \mathbb{1}_{\{S_T \geq K\}}$. When using the smoothed payoff we want to smooth the jump, because of this we introduce a lower and upper bound. The lower and upper bound is given as $X_L = \frac{1}{2}(S_{i-1} + S_i)$ and $X_U = \frac{1}{2}(S_i + S_{i+1})$ respectively. The smoothed payoff function is then given as:

$$\Phi = \begin{cases} 0, & X_U \leq K \\ 1, & X_L \geq K \\ \frac{X_U - K}{X_U - X_L}, & \text{otherwise.} \end{cases}$$

We can then compare the prices of the smoothed payoff function with the original prices of the digital option, where both have 100 space steps:



Here we can see that, using a smoothed payoff function, we get a nicer-looking curve over the prices of the digital options given the different strikes. This gives us different prices for all the different strikes as we hoped. Using this method does not increase the computing time significantly. However, it is worth noting that when we use smoothing we use an approximated payoff function and not the true payoff function. This can lead to a small difference in the price found using smoothing and the true price of a digital option.

All the prices are given in the table below:

K	$\pi^D_{n=100}$	$\pi^D_{n=502}$	$\pi^{D,\text{ smoothing}}_{n=100}$
1.01	0.2220648384486190	0.2275780210450110	0.2301120958943020
1.02	0.2220648384486190	0.2221393200594940	0.2238585044421760
1.03	0.2220648384486190	0.2167661814669550	0.2180520594800670
1.04	0.2220648384486190	0.2114599214014600	0.2124254452036330
1.05	0.1957422170992960	0.2062217868668810	0.2067988309271990
1.06	0.1957422170992960	0.2010529551354700	0.2011722166507650
1.07	0.1957422170992960	0.1959545332641390	0.1955670840762880
1.08	0.1957422170992960	0.1909275577278530	0.1905552215735490
1.09	0.1957422170992960	0.1859729941692990	0.1855433590708110
1.10	0.1712231817224060	0.1810917372637080	0.1805314965680720
1.11	0.1712231817224060	0.1762846106974700	0.1755196340653340
1.12	0.1712231817224060	0.1715523672589160	0.1705923084268730
1.13	0.1712231817224060	0.1668956890394130	0.1661726756183360
1.14	0.1712231817224060	0.1623151877426670	0.1617530428097990
1.15	0.1486125518465550	0.1578114050999400	0.1573334100012610

Prices for a digital option for varying strikes with $n = 100$, $n = 502$ and using smoothing.

When comparing the prices from our data, we see that using 502 steps and smoothing are fairly close. This means that both methods can be used to improve the result satisfactorily. What method is then favored depends on the user's preference about the trade-off between a more exact price or a faster computation time. Either way, these two suggested methods will always be better than the original computation using 100 timesteps without smoothing.

Question 4

We will in this question increase the width of the grid and the number of spatial steps such that $\Delta \log S$ is kept constant. This means that we will slowly increase the width of the grid but we will keep the same spatial points. We notice that:

$$\Delta \log S = \log \left(\frac{2 \cdot \text{num std} \cdot \text{std}}{\text{num s}} \right).$$

By changing the number of standard deviations with a fraction of k and the number of spatial steps with a fraction of c we would obtain the following equation:

$$\Delta \log S = \log \left(\frac{2 \cdot k \cdot \text{num std} \cdot \text{std}}{c \cdot \text{num s}} \right).$$

It is easy to see that for $\Delta \log S$ to be constant we must require that $c = k$. This means that if we want to have a double as wide a grid we must require double as many spatial time steps.

By choosing the number of standard deviation to be equal to $1, \cdots, 10$ we obtain the following prices:

Num std	FD price
1	0.072421476105058
2	0.079411383721769
3	0.079419983564961
4	0.079419983960401
5	0.079419983960402
6	0.079419983960402
7	0.079419983960402
8	0.079419983960402
9	0.079419983960402
10	0.079419983960402

We observe that a lower number of standard deviations leads to a wrong price. But setting the number of standard deviations to 5 gives the same prices as using more standard deviations. Setting too wide of a grid would therefore lead to a lot of unnecessary calculations and would be a waste of time. Looking at the standard normal distribution quantiles we see that the probability mass is increasing, but the marginal increases are decreasing fast:

Num std	Normal Cdf
1	0.682689492137086
2	0.954499736103642
3	0.997300203936740
4	0.999936657516334
5	0.999999426696856
6	0.999999998026825
7	0.999999999997440
8	0.999999999999999
9	1.000000000000000
10	1.000000000000000

Already at 5 standard deviations, we capture most of the sample space, with a very small marginal difference when increasing to 6 standard deviations. This means that the sweet spot is 5 and is therefore a good rule of thumb.

Question 5

We have in this question programmed a new method called `kBlack::fdFwdRunner()`. This method constructs the finite difference grid by rolling forward instead of backward as the name suggests. The method returns a full finite difference grid of initial European call option prices where the first dimension is different expiries and the second dimension is different strike values. First, the method calculates the probability of each option with strike K_i ending up in the money before it takes advantage of the hint regarding that

$$c(t_h, s_i) = \sum_{j=i}^{n-1}(s_j - s_i)p(t_h, s_j) = \sum_{j=0}^{n-1}(s_j - s_i)\mathbb{1}_{\{s_j > s_i\}}p(t_h, s_j),$$

to calculate the call option prices. We know that a European call option always has a positive gamma in the Black-Scholes model. We are ensured to have this property as well if we use the implicit method. This is because:

$$\begin{aligned}
\partial_{ss}c &= \partial_s\partial_s c \\
&= \partial_s \frac{1}{s_{i+1} - s_i}\left[\sum_{j=0}^{n-1}(s_j - s_{i+1})\mathbb{1}_{\{s_j > s_{i+1}\}}p(t_h, s_j) - \sum_{j=0}^{n-1}(s_j - s_i)\mathbb{1}_{\{s_j > s_i\}}p(t_h, s_j)\right] \\
&= \partial_s \frac{1}{s_{i+1} - s_i}\left[\sum_{j=0}^{n-1}\left\{(s_j - s_{i+1})\mathbb{1}_{\{s_j > s_{i+1}\}} - (s_j - s_i)\mathbb{1}_{\{s_j > s_i\}}\right\}p(t_h, s_j)\right] \\
&= \partial_s \frac{1}{s_{i+1} - s_i}\left[-\sum_{j=i+2}^{n-1}(s_{i+1} - s_i)p(t_h, s_j) - (s_{i+1} - s_i)p(s_{i+1})\right] \\
&= \partial_s\left[-\sum_{j=i+1}^{n-1}p(t_h, s_j)\right] \\
&= -\frac{1}{s_{i+1} - s_i}\left[-\partial_s\sum_{j=i+2}^{n-1}p(t_h, s_j) + \partial_s\sum_{j=i+1}^{n-1}p(t_h, s_j)\right] \\
&= \frac{p(t_h, s_{i+1})}{s_{i+1} - s_i}
\end{aligned}$$

Since it always holds that $\frac{1}{s_{i+1}-s_i} > 0$ we must look at the probabilities. From table 5 in *Fun with Finite Difference* we know that for a drift $\mu \neq 0$ the probabilities are non-negative for the implicit method when using winding. For the other two methods, it only holds if $\Delta t \leq O(\Delta s^2)$. If however, the drift $\mu = 0$ then we would always obtain

non-negative probabilities for the implicit method but are still not ensured to obtain that using the explicit or Crank-Nicolson method. As a consequence of this, we are only ensured to see non-negative gammas when we are using the implicit method.

We have implemented a new function `kFd1d::rollFwdDupire`. It rolls initial option prices using the relation given in the question:

$$c(t_{h+1}) = \left[I + (1-\theta)\Delta t\bar{A}\right]\left[I - \theta\Delta t\bar{A}\right]^{-1} c(t_h)$$

When we have zero drift and rates, we see that the option prices using the transition probabilities and using Dupire's formula above are exactly the same. This can be seen in the following tables:

Forward method

k	0.0		1.0		2.0		3.0		4.0	
0.107	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614
0.131	0.869030272523864	0.869030272524989	0.869030272544188	0.869030272705262	0.869030273595523	0.869030277262422	0.869030289337421	0.869030322640275	0.869030402193169	0.869030571035275
0.16	0.839507836541676	0.839507836554560	0.839507836753685	0.839507838264910	0.839507845822017	0.839507874028290	0.839507958441137	0.839508170895751	0.839508636370474	0.839509547605905
0.197	0.803330624351900	0.803330624498088	0.803330626517360	0.803330640190996	0.803330701199037	0.803330904737920	0.803331451186719	0.803332691307192	0.803335156342851	0.803339563988532
0.241	0.758998554917870	0.758998556576381	0.758998576761254	0.758998696945949	0.758999168631144	0.759000557070275	0.759003863907360	0.759010570091353	0.759022581742272	0.759042103312943
0.295	0.704673407640240	0.704673426456112	0.704673624551766	0.704674642368231	0.704678093710281	0.704686917207834	0.704705325878069	0.704738378977338	0.704791397382942	0.704869430384653
0.362	0.638102600898907	0.638102814365674	0.638104711178374	0.638112912348565	0.638136387310937	0.638187521826317	0.638279684994901	0.638424908948450	0.638632525232612	0.638908736953458
0.443	0.556525789196147	0.556528210984618	0.556545752939934	0.556607367502416	0.556751709886955	0.557013595031590	0.557415833400022	0.557968443356433	0.558671309577484	0.559517542567863
0.543	0.456560405969753	0.456587881250673	0.456741771005630	0.457158349076039	0.457924676407208	0.459055587936644	0.460522155816071	0.462277197758777	0.464270245080706	0.466454631863728
0.666	0.334061406131276	0.334373114222551	0.335607061416436	0.337969634186772	0.341214492367958	0.345058229955719	0.349278448325841	0.353719495293467	0.358277248179783	0.362883339286752
0.816	0.183949392581120	0.187485732553313	0.195676955710122	0.205204966332011	0.214847672145498	0.224181366029456	0.233084353883585	0.241546188490305	0.249594368834123	0.257266497483167
1	0.000000000000000	0.040119909456988	0.067158187524546	0.087480706050489	0.104007912089252	0.118173139696689	0.130735997742267	0.142132622815667	0.152633311136772	0.162417248758882
1.225	0.000000000000000	0.004333481208204	0.014371122357345	0.026046881844891	0.037863190448578	0.049300831446702	0.060210679161034	0.070579931422829	0.080442285878101	0.089843821247657
1.502	0.000000000000000	0.000468073324093	0.002321017732551	0.005868751400622	0.010741360093168	0.016513270030735	0.022850518553306	0.029519372121031	0.036363475959129	0.043280166400380
1.84	0.000000000000000	0.000050558113951	0.000333735410281	0.001100293598139	0.002510436214882	0.004591461487707	0.007290138388525	0.010519645332548	0.014187113333254	0.018206670990205
2.255	0.000000000000000	0.000005460945443	0.000045016696125	0.000183952767211	0.000509433661074	0.001099964380152	0.002006980748725	0.003253073401054	0.004837982288298	0.006746172121493
2.763	0.000000000000000	0.000000589854383	0.000005831154000	0.000028492743186	0.000093359090353	0.000234654704089	0.000489321268481	0.000890606131696	0.001464294327198	0.002227526297075
3.386	0.000000000000000	0.000000063712080	0.000000734480159	0.000004180889823	0.000015867415231	0.000045744493409	0.000108077727852	0.000219998116193	0.000399522417037	0.000663747518826
4.149	0.000000000000000	0.000000006881742	0.000000090635771	0.000000589323056	0.000002546501255	0.000008307571406	0.000022028577575	0.000049854096536	0.000099692228929	0.000180687568160
5.085	0.000000000000000	0.000000000743252	0.000000011009238	0.000000080521622	0.000000390637659	0.000001425123364	0.000004201896798	0.000010501803979	0.000023019624416	0.000045391030201
6.231	0.000000000000000	0.000000000079524	0.000000001306072	0.000000010591818	0.000000056887161	0.000000229056515	0.000000742102280	0.000002026926226	0.000004825842804	0.000010270015793
7.635	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000

Dupire method

k	0.00	0.50	1.00	1.50	2.00	2.50	3.00	3.50	4.00	4.50
0.107	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614	0.893122074339614
0.131	0.869030272523864	0.869030272525	0.869030272544188	0.869030272705262	0.869030273595523	0.869030277262423	0.869030289337421	0.869030322640275	0.869030402193169	0.869030571035276
0.16	0.839507836541676	0.839507836554561	0.839507836753685	0.839507838264910	0.839507845822017	0.839507874028290	0.839507958441138	0.839508170895751	0.839508636370474	0.839509547605905
0.197	0.803330624351900	0.803330624498088	0.803330626517360	0.803330640190996	0.803330701199038	0.803330904737921	0.803331451186720	0.803332691307192	0.803335156342852	0.803339563988533
0.241	0.758998554917870	0.758998556576381	0.758998576761254	0.758998696945949	0.758999168631144	0.759000557070275	0.759003863907361	0.759010570091354	0.759022581742273	0.759042103312943
0.295	0.704673407640240	0.704673426456112	0.704673624551767	0.704674642368231	0.704678093710281	0.704686917207834	0.704705325878069	0.704738378977339	0.704791397382943	0.704869430384654
0.362	0.638102600898907	0.638102814365675	0.638104711178374	0.638112912348565	0.638136387310937	0.638187521826317	0.638279684994901	0.638424908948451	0.638632525232612	0.638908736953459
0.443	0.556525789196147	0.556528210984618	0.556545752939934	0.556607367502416	0.556751709886955	0.557013595031591	0.557415833400022	0.557968443356433	0.558671309577485	0.559517542567864
0.543	0.456560405969753	0.456587881250673	0.456741771005630	0.457158349076040	0.457924676407208	0.459055587936644	0.460522155816071	0.462277197758778	0.464270245080707	0.466454631863728
0.666	0.334061406131276	0.334373114222551	0.335607061416436	0.337969634186772	0.341214492367958	0.345058229955719	0.349278448325841	0.353719495293467	0.358277248179784	0.362883339286752
0.816	0.183949392581120	0.187485732553313	0.195676955710122	0.205204966332011	0.214847672145498	0.224181366029457	0.233084353883585	0.241546188490306	0.249594368834123	0.257266497483167
1	0.000000000000000	0.040119909456988	0.067158187524546	0.087480706050489	0.104007912089252	0.118173139696689	0.130735997742267	0.142132622815667	0.152633311136772	0.162417248758882
1.225	0.000000000000000	0.004333481208204	0.014371122357345	0.026046881844891	0.037863190448578	0.049300831446702	0.060210679161034	0.070579931422829	0.080442285878101	0.089843821247657
1.502	0.000000000000000	0.000468073324093	0.002321017732551	0.005868751400622	0.010741360093168	0.016513270030735	0.022850518553306	0.029519372121031	0.036363475959129	0.043280166400381
1.84	0.000000000000000	0.000050558113951	0.000333735410281	0.001100293598139	0.002510436214882	0.004591461487707	0.007290138388525	0.010519645332548	0.014187113333254	0.018206670990205
2.255	0.000000000000000	0.000005460945443	0.000045016696125	0.000183952767211	0.000509433661074	0.001099964380152	0.002006980748725	0.003253073401054	0.004837982288298	0.006746172121493
2.763	0.000000000000000	0.000000589854383	0.000005831154000	0.000028492743186	0.000093359090353	0.000234654704089	0.000489321268481	0.000890606131696	0.001464294327198	0.002227526297075
3.386	0.000000000000000	0.000000063712080	0.000000734480159	0.000004180889823	0.000015867415231	0.000045744493409	0.000108077727852	0.000219998116193	0.000399522417037	0.000663747518826
4.149	0.000000000000000	0.000000006881742	0.000000090635771	0.000000589323056	0.000002546501255	0.000008307571406	0.000022028577575	0.000049854096536	0.000099692228929	0.000180687568160
5.085	0.000000000000000	0.000000000743252	0.000000011009238	0.000000080521622	0.000000390637659	0.000001425123364	0.000004201896798	0.000010501803979	0.000023019624416	0.000045391030201
6.231	0.000000000000000	0.000000000079524	0.000000001306072	0.000000010591818	0.000000056887161	0.000000229056515	0.000000742102280	0.000002026926226	0.000004825842804	0.000010270015793
7.635	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000	0.000000000000000

Numerical difference

k	0.00	0.50	1.00	1.50	2.00	2.50	3.00	3.50	4.00	4.50
0.107	0.00E+00	2.22E-16	1.11E-16	0.00E+00	1.11E-16	1.11E-16	1.11E-16	2.22E-16	1.11E-16	0.00E+00
0.131	0.00E+00	0.00E+00	1.11E-16	0.00E+00	1.11E-16	2.22E-16	2.22E-16	3.33E-16	3.33E-16	1.11E-16
0.16	0.00E+00	1.11E-16	3.33E-16	2.22E-16	2.22E-16	2.22E-16	4.44E-16	4.44E-16	3.33E-16	2.22E-16
0.197	0.00E+00	1.11E-16	0.00E+00	1.11E-16	3.33E-16	4.44E-16	6.66E-16	5.55E-16	7.77E-16	7.77E-16
0.241	0.00E+00	3.33E-16	4.44E-16	3.33E-16	3.33E-16	2.22E-16	5.55E-16	5.55E-16	6.66E-16	6.66E-16
0.295	0.00E+00	3.33E-16	3.33E-16	0.00E+00	0.00E+00	3.33E-16	4.44E-16	6.66E-16	7.77E-16	6.66E-16
0.362	0.00E+00	2.22E-16	0.00E+00	1.11E-16	2.22E-16	3.33E-16	3.33E-16	5.55E-16	5.55E-16	4.44E-16
0.443	0.00E+00	1.11E-16	0.00E+00	3.33E-16	0.00E+00	4.44E-16	2.22E-16	3.33E-16	4.44E-16	2.22E-16
0.543	0.00E+00	5.55E-17	0.00E+00	1.67E-16	1.11E-16	1.67E-16	2.78E-16	2.22E-16	3.33E-16	2.78E-16
0.666	0.00E+00	5.55E-17	0.00E+00	5.55E-17	5.55E-17	0.00E+00	1.11E-16	1.11E-16	1.67E-16	1.11E-16
0.816	0.00E+00	0.00E+00	2.78E-17	5.55E-17	5.55E-17	8.33E-17	8.33E-17	8.33E-17	1.39E-16	5.55E-17
1	0.00E+00	0.00E+00	1.39E-17	1.39E-17	1.39E-17	4.16E-17	2.78E-17	1.11E-16	8.33E-17	8.33E-17
1.225	0.00E+00	1.73E-18	6.94E-18	0.00E+00	6.94E-18	6.94E-18	6.94E-18	1.39E-17	4.16E-17	4.16E-17
1.502	0.00E+00	5.42E-20	0.00E+00	8.67E-19	0.00E+00	0.00E+00	6.94E-18	1.73E-17	2.08E-17	4.16E-17
1.84	0.00E+00	6.78E-21	5.42E-20	2.17E-19	8.67E-19	0.00E+00	8.67E-19	3.47E-18	5.20E-18	1.04E-17
2.255	0.00E+00	1.69E-21	6.78E-21	1.36E-19	2.17E-19	2.17E-19	4.34E-19	0.00E+00	8.67E-19	3.47E-18
2.763	0.00E+00	2.12E-22	2.54E-21	1.36E-20	5.42E-20	1.63E-19	0.00E+00	2.17E-19	2.17E-19	4.34E-19
3.386	0.00E+00	1.32E-23	2.12E-22	2.54E-21	1.36E-20	2.03E-20	6.78E-20	1.63E-19	2.17E-19	1.08E-19
4.149	0.00E+00	4.96E-24	3.97E-23	3.18E-22	2.12E-21	1.02E-20	1.69E-20	5.42E-20	8.13E-20	1.08E-19
5.085	0.00E+00	9.31E-25	9.93E-24	1.06E-22	4.76E-22	2.12E-21	6.78E-21	1.52E-20	2.37E-20	4.74E-20
6.231	0.00E+00	6.46E-26	1.24E-24	1.32E-23	6.62E-23	3.71E-22	1.06E-21	3.39E-21	5.08E-21	1.02E-20
7.635	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

The last table shows the numerical difference between the prices we find using the forward and the Dupire method. Here we see that the difference is on the 16th decimal and smaller which is the same as zero since we are working with doubles.

A useful case of Dupire's formula is when we are looking at the local volatilities. Because $\mu = r = 0$ we have a simple form of Dupire's function which makes it easier for us to calculate. The relationship between the call option price and the local volatility is given by[1]:

$$\sigma^2(K, T, S_0) = \frac{\frac{\partial C}{\partial T}}{\frac{1}{2}K^2 \frac{\partial^2 C}{\partial K^2}}.$$

We can therefore estimate volatility by using only market data. This can be very useful if we extend the scope beyond the classical Black-Scholes world to model stochastic volatility.

[1]Gatheral, J. - *The Volatility Surface* 2006

Question 6

In this question, we will modify the code to price a Down-and-Out call option and find the convergence of the price, first by only changing the payoff and next by changing the grid and the dynamics of the underlying. We do this by forcing the barrier onto the grid and then forcing the dynamics in this grid point to be $\mu = \sigma = 0$.

A Down-and-Out call option is an option that terminates if the stock reaches the barrier. The payoff function is therefore the same as the European call option unless the stock price reaches the barrier, then the payoff becomes zero no matter how the stock price changes in the future.

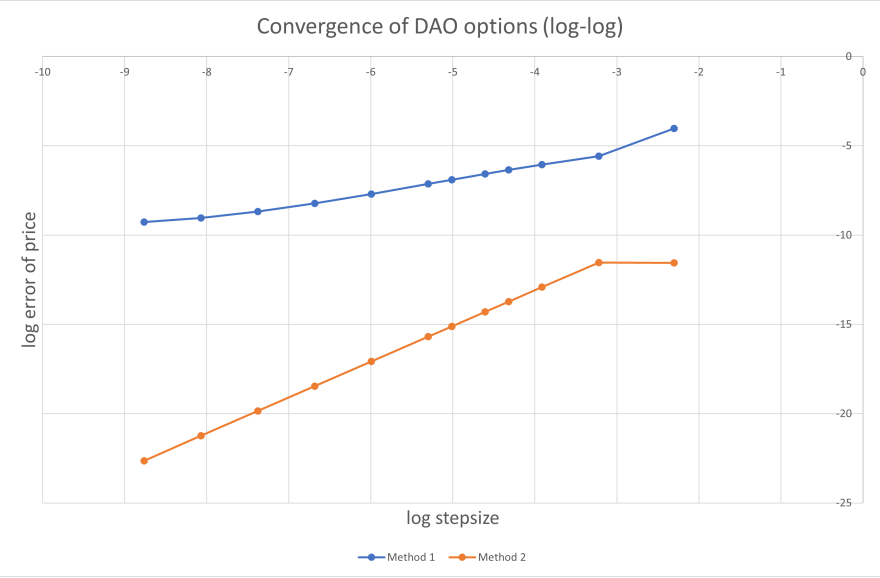
We have modified the `kBlack::fdRunner` function to take a barrier price as input and use this to make the payoff become zero if the stock price hits the barrier. We will later refer to this as method 1. The barrier we use in this question is 0.80.

Since the barrier is continuously monitored we can improve the convergence by placing the barrier level onto the grid and enforcing absorption on the barrier. We do this by taking the grid point closest to the barrier level and replacing this with the barrier. We then set $\mu = \sigma = 0$ exactly in the grid point. We will refer to this as method 2.

This leads to two different methods of finding the Down-and-Out price. These are given in the table below for the different time steps as in question 1:

m	Δt	Method 1	Method 2
10	$\frac{1}{2}$	0.068224226	0.061658317
25	$\frac{1}{5}$	0.065459958	0.061658587
50	$\frac{1}{10}$	0.064010815	0.061651221
75	$\frac{1}{15}$	0.063393629	0.061649855
100	$\frac{1}{20}$	0.063043995	0.061649377
150	$\frac{1}{30}$	0.06265737	0.061649036
200	$\frac{1}{40}$	0.062446679	0.061648916
400	$\frac{1}{80}$	0.062103088	0.061648801
800	$\frac{1}{160}$	0.061916169	0.061648772
1600	$\frac{1}{320}$	0.061818198	0.061648765
3200	$\frac{1}{640}$	0.061767969	0.061648763
6400	$\frac{1}{1280}$	0.061742527	0.061648763

We can illustrate the convergence by the following graph:



We see here that the prices for method 2 look more stable. We can show this by looking at the order of convergence. By doing linear regression we find the order of convergence for method 1 to be $a \approx 0.523$. This matches our expectation of an accuracy loss of $O\left(\Delta t^{1/2}\right)$

(Page 8 in *Fun with Finite Difference*). The reason for the loss in order of convergence is that when pricing without absorption, we are actually pricing the option with the barrier as the first grid point below the barrier, more formally:

$$c^{DAO}(B) \to c^{DAO}(s(i)), \quad \sup_i\{s(i) \leq B\}.$$

For method 2, we expect the order of convergence to be $a = 2$ because we use the Crank-Nicolson method and we expect that the price converges to the price of a Down-and-Out option with the actual barrier. Looking at the convergence order for method 2 of our data we see that $a \approx 2.003$, which matches our expectations.