Alternating Direction Implicit methods for Black-Scholes Equations

Ray Wu

1 Introduction

In this project we consider numerically solving two- and three-dimensional Black-Scholes parabolic partial differential equations (PDEs) using finite difference (FD) methods. The main purpose is to contrast the performance of Alternating Direction Implicit (ADI) methods and Crank-Nicolson (CN) methods in multiple dimensions as measured by time complexity. A secondary purpose is to present an efficient implementation of the ADI method.

We will discuss the problem formulation in Section 2, describe the algorithms in Section 3, present numerical results of our algorithms in Section 4, discuss the order of convergence and the runtime in Section 5, and make our conclusions in Section 6.

2 Problem Formulation

Although the focus of the project is on multidimensional Black-Scholes PDEs, we will start by introducing the one-dimensional Black-Scholes PDE.

$$V(\tau, S)_\tau = \mathcal{L}(V) = \frac{1}{2}\sigma^2 S^2 V_{SS} + (r-q)SV_S - rV, \tag{1}$$

where V is the price of the option, S is the price of the underlying asset, τ is backwards time, σ is the volatility, r is the interest rate, and q is the dividend rate. We start at some time $\tau = 0$ until the end time $\tau = T$, given some initial conditions $V^*(S) = V(\tau = 0, S)$ known as the payoff. The payoffs most useful to us will be the call and put, given by

$$V^*_{\text{call}}(S) = \max(S - K, 0) \tag{2}$$
$$V^*_{\text{put}}(S) = \max(K - S, 0) \tag{3}$$

The solution of Equation (1) at $\tau = T$ gives the Black-Scholes price of the option.

Next, we will consider the d-dimensional European options with no dividend, which are given by

$$V(\tau, S_i)_\tau = \mathcal{L}^{(d)}(V) = \frac{1}{2}\sum_{i,j=1}^{d} \rho_{i,j}\sigma_i\sigma_j V_{S_i}V_{S_j} + \sum_{i=1}^{d} rS_iV_{S_i} - rV, \tag{4}$$

where S_i are the stock prices, σ_i, $\rho_{i,j}$ denote volatility of stock prices and correlation between stock prices (with $\rho_{j,j} = 1$).

For the problem to be well-posed, we need to have initial conditions, which correspond to the type of payoff at the option's maturity. The test problems we use to illustrate our methods are geometric put and call which are convenient because they can be simplified to one-dimensional option pricing problems that either have exact solutions or can be solved with efficient algorithms [6]. The payoff functions for d-dimensional geometric average call and put are given by:

$$V^*_{\text{call}}(S_i) = \max((\Pi S_i)^{1/d} - K, 0) \tag{5}$$
$$V^*_{\text{put}}(S_i) = \max(K - (\Pi S_i)^{1/d}, 0) \tag{6}$$

Note that when $d = 1$, the above definitions simplify to the one-dimensional call and put payoffs.

Geometric average option pricing problems are special because that they are one of the few option pricing problems that have exact solutions no matter what number of underlying assets (dimensions) we have. Therefore it is a popular and convenient test problem for many algorithms designed for high-dimensional PDE problems.

With a geometric average call/put initial condition, the problem (4) can be transformed to a one-dimensional option pricing problem

$$V(\tau, S)_\tau = \frac{\sigma^2 S^2}{2} V_{SS} + (r - q)S - rS \tag{7}$$

where

$$\sigma = \frac{1}{n} \sqrt{\sum_{i,j=1}^{d} \rho_{i,j} \sigma_i \sigma_j} \tag{8}$$

$$q = \frac{1}{2n} \sum_{i=1}^{d} \sigma_i^2 - \frac{1}{2}\sigma^2 \tag{9}$$

and we can either use the Black-Scholes formula to compute the solution in the case of European options or use known algorithms such as the penalty method [6] in the case of American options.

3 PDE discretization

Consider the stock prices S_i which belong to the semi-infinite domain $[0, \infty)$. For convenience, we will use x, y, and z to represent the (up to) three stock prices S_1, S_2, and S_3 in our problem. For computation purposes, we truncate the domain to $[0, x_{\max}]$. Here, we pick $S_{1\max} = 10K$. As is convention, let τ^k, $k = 0, \ldots, N_\tau$ denote a (possibly non-uniform) partition of the time interval $[0, T]$ in ascending order, and $\Delta\tau^k = \tau^k - \tau^{k-1}$. Let $x_0, x_1, \ldots, x_{N_x}$ denote a (possibly non-uniform) partition of the interval $[0, x_{\max}]$, and y_i, z_i defined likewise. In our numerical experiments, we use a uniform timestepping scheme for European options and a variable timestepping scheme presented in [6] for American options. For spatial discretization, we use a nonuniform grid in [1] with a more dense grid (more gridpoints are allocated near the region of interest) for two dimensions and a slightly less dense grid for three dimensions. Finally, let $v^k_{h,i,j}$ denote a computed solution of $u(\tau^k, x_h, y_i, z_j)$ and $v^k_{i,j}$ denote a computed solution of $u(\tau^k, x_i, y_j)$. For the purposes of algorithm analysis, we assume that $N_x = N_y = N_z$ which we denote as N.

3.1 Space discretization

We use standard second-order FD for the spatial derivatives, and denote the d-dimensional discretized operator as a square $(N + 1)^d$ matrix $L^{(d)}$ which encompasses the boundary conditions.

Note that $L^{(d)}$ can be decomposed into $L^{(d)} = L_0 + L_1 + L_2 + \cdots + L_d$, where L_j ($j > 0$) are the matrices resulting from just the FD with respect to the j-th spatial variable, sharing equally the $-ru$ (no-derivative) term and L_0 is the matrix resulting from only the cross-derivative discretizations.

3.2 Crank-Nicolson Method

Let $\theta \in [0,1]$. To solve the solution at time τ^k given the solution at τ^{k-1}, the θ-timestepping discretization solves the linear system

$$(I - \theta\Delta\tau^k L^{(d)})v^k = (I + (1-\theta)\Delta\tau^k L^{(d)})v^{k-1} \tag{10}$$

where I denotes the identity matrix of size $(N+1)^d$. When $\theta = 1$, we have the Fully Implicit/Backward Euler (BE) method, and when $\theta = 1/2$, we have the Crank-Nicolson (CN) method. Although CN is second order convergent, it may produce spurious oscillations when applied to non-smooth initial conditions. On the other hand, BE is more stable, but only first order convergent. To obtain both the accuracy of CN and the stability of BE, we use the Rannacher smoothing [9] technique which applies BE to the first two timesteps partitioned into four smaller timesteps, and in the subsequent timesteps uses CN and the usual stepsize. This results in smooth data for the Crank-Nicolson method, and the use of a few iterations of the first-order BE method does not affect the overall second order convergence of the CN method.

Solving Equation (10) requires the solution of a $N^d \times N^d$ sparse, nonsymmetric linear system with bandwidth $\mathcal{O}(N^{d-1})$ each iteration. Recall that banded LU factorization requires $\mathcal{O}(nlu)$ where n is the size of the matrix, l is the lower bandwidth and u is the upper bandwidth. Banded LU will take $\mathcal{O}(N^{d(d-1)^2})$; in both the two-dimensional or three-dimensional case MATLAB's backslash operator eventually resorts to using a general sparse LU solver with full pivoting [3] which is designed to reduce fill-in.

3.3 ADI and Craig-Sneyd

ADI methods such as [2, 4, 8] minimize the bandwidth of the matrices being solved; If we solve a tridiagonal system at each iteration, then the cost reduces from $\mathcal{O}(N^{d(d-1)^2})$ to $\mathcal{O}(N^d)$, which is optimal since we have $\mathcal{O}(N^d)$ unknowns.

The original ADI method [5] does not consider mixed derivatives (i.e. v_{xy}). The Craig-Sneyd [2] and more general ADI methods [8] address this and we will focus on the Modified Craig-Sneyd (MCS) method in this study.

In the MCS method, we solve the following set of equations/update the following vectors for a d-dimensional problem, where the restrictions $1/3 \leq \theta$, $\sigma = \theta$, $\mu = 1/2 - \theta$ are imposed upon the parameters to ensure second order convergence.

Note that for solving the systems of equations, we re-order the entries of the right-hand-side vector so that the matrix $I - \theta\Delta\tau^k L_j$ is always tridiagonal. This ensures $\mathcal{O}(N^d)$ performance per iteration.

3.4 Matricized Craig-Sneyd

This only applies to the two-dimensional case, although a similar process can be used for the three-dimensional case.

Instead of writing the previous algorithm in vector form, we can take advantage of the special structure of the matrices L_1 and L_2. Note that in the default ordering,

$$L_0 = \rho D_x \otimes D_y \tag{11}$$
$$L_1 = L_x \otimes I \tag{12}$$
$$L_2 = I \otimes L_y \tag{13}$$

Algorithm 1 MCS ADI timestepping in d dimensions

Take an explicit step:
$Y_0 = v^{k-1} + \Delta\tau^k(Lv^{k-1})$
Take d **implicit corrector steps, one in each dimension**:
for $j = 1, \ldots, d$ **do**
  compute $g = Y_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
  reorder the entries in g such that L_j is tridiagonal
  Solve the linear system $(I - \theta\Delta\tau^k L_j)Y_j = Y_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
end for
Take two more explicit updates:
$\tilde{Y_0} = Y_0 + \sigma\Delta\tau^k L_0(Y_2 - v^{k-1})$
$\hat{Y_0} = \tilde{Y_0} + \mu\Delta\tau^k L(Y_2 - v^{k-1})$
Take d **implicit corrector steps in each dimension**:
for $j = 1, \ldots, d$ **do**
  compute $g = \tilde{Y}_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
  reorder the entries in g such that L_j is tridiagonal
  Solve the linear system $(I - \theta\Delta\tau^k L_j)\hat{Y}_j = g$
end for
reorder $\hat{Y}_d$ to the original ordering
assign v^k to $\hat{Y}_d$

where D_x and D_y are matrices resulting from the discretization of the $\sigma_x x v_x$ and $\sigma_y y v_y$ terms, I is the identity matrix of size $N+1$, and L_x, L_y are matrices resulting from the discretization of just the x and just the y terms, with the no-derivative term $(-ru)$ shared equally between L_x and L_y.

Let V^k be an $N+1 \times N+1$ matrix containing the computed solution at time τ^k; so $u = \text{vec}(U)$ where vec denotes the vectorization operator. Then we will use the following useful identity:

$$(A \otimes B)v = \text{vec}(BVA^T), \tag{14}$$

where $V = \text{vec}^{-1}(v)$ is the inverse of the vectorization operator applied to v.

Then,

$$L_0 u = \rho D_y U D_x^T \tag{15}$$
$$L_1 u = U L_x^T \tag{16}$$
$$L_2 u = L_y U \tag{17}$$

In addition (after reordering the entries in v appropriately),

$$I - \theta\Delta\tau L_1 = I \otimes I - I \otimes \theta\Delta\tau L_x = I \otimes (I - \theta\Delta\tau L_x) \tag{18}$$
$$I - \theta\Delta\tau L_2 = I \otimes I - I \otimes \theta\Delta\tau L_y = I \otimes (I - \theta\Delta\tau L_y) \tag{19}$$

Therefore,

$$(I - \theta\Delta\tau L_1)^{-1} = (I \otimes (I - \theta\Delta\tau L_x))^{-1} = I^{-1} \otimes (I - \theta\Delta\tau L_x)^{-1} = I \otimes (I - \theta\Delta\tau L_x)^{-1} \tag{20}$$

and we can apply identity (11) to (17) as well. Note that we never actually compute the inverse of any matrix, but (17) is used to illustrate how we can avoid solving an $\mathcal{O}(N^2)$ system and instead solve an $\mathcal{O}(N)$ system. Note that in Algorithm 2, all the update equations only involve multiplying with or solving linear systems of size N.

Algorithm 2 Matrix-form of Craig-Sneyd ADI timestepping in two dimensions

1: **Take an explicit step**:
2: $Y_0^T = U^T + \Delta\tau^k(\rho D_y U D_x^T + L_x U^T + (UL_y)^T)$
3: **Take two implicit corrector steps, one in each dimension**
4: Solve the linear system $(I - \theta\Delta\tau^k L_x)Y_1^T = Y_0 - \theta\Delta\tau^k L_x U^T$
5: Solve the linear system $(I - \theta\Delta\tau^k L_y)Y_2 = Y_1 - \theta\Delta\tau^k L_y U$
6: **Take two more explicit updates**:
7: $\tilde{Y}_0 = Y_0 + \sigma\rho\Delta\tau^k D_y U D_x^T$
8: $\hat{Y}_0 = \tilde{Y}_0 + \mu\Delta\tau^k[\rho D_y(Y_2 - U)D_x^T + L_x(Y_2 - U)^T + ((Y_2 - U)L_y)^T]$
9: **Take two implicit corrector steps, one in each dimension**
10: Solve the linear system $(I - \theta\Delta\tau^k L_x)\tilde{Y}_1^T = \tilde{Y}_0^T - \theta\Delta\tau^k L_x U^T$
11: Solve the linear system $(I - \theta\Delta\tau^k L_y)U^k = \tilde{Y}_1 - \theta\Delta\tau^k L_y U$

For three and higher dimensions, we cannot use the matrix identity (14) in the same way, however, we can still take advantage of the special structure of the matrix. The tridiagonal matrix that we solve in lines 7 and 16 in Algorithm 1 can be decomposed as follows:

$$I - \theta\Delta\tau^k L_j = I \otimes I \otimes \cdots \otimes I \otimes (I - \theta\Delta\tau^k L_x) \tag{21}$$

Therefore, when solving the linear system

$$(I - \theta\Delta\tau^k L_j)v = g \tag{22}$$

for u, we can rearrange the $(N+1)^d$ length vector g into an $N+1 \times (N+1)^{d-1}$ matrix G in column major order. Then instead of solving Equation (22) we only need to solve the $N+1$ by $N+1$ linear system

$$(I - \theta\Delta\tau^k L_x)V = G. \tag{23}$$

After solving, V can be rearranged into a vector again for explicit updates.

3.5 American options

In contrast to European options, American options can be exercised at any point prior to the expiry date. Therefore, the Black-Scholes model (4) becomes more complicated and can be written in the form of a linear complementarity problem (LCP)

$$((v_\tau - Lv > 0) \wedge (v = V^*)) \vee ((v_\tau = Lv) \wedge (v - V^* > 0)) \tag{24}$$

which leads to the PDE

$$v_\tau = \mathcal{L}v + p\max(V^* - v, 0) \tag{25}$$

for some large value p, typically the reciprocal of any accuracy required.

The PDE (25) can be solved by the penalty method introduced in [6]. For ADI methods, [7] incorporate the penalty method into the Craig-Sneyd methods, leading to efficient algorithms for two-dimensional problems.

The penalty term $p\max(V^* - v, 0)$ is discretized as $P^k(V^* - v^k)$. The matrix P^k is a diagonal matrix defined as

$$P^k_{i,i} = \begin{cases} p & \text{if } v^k_i < V^*_i \\ 0 & \text{otherwise.} \end{cases} \tag{26}$$

For discretization, the other terms in the PDE (25) are discretized in the same way as the previous section.

The penalty iteration applied to θ-timestespping is given by Algorithm 3

Algorithm 3 Penalty Iteration for θ-timestepping

1: let $v^{k,0} = v^{k-1}$ and compute $P^{k,0} = P(v^{k-1})$.
2: **for** m = 1, ... **do**
3: solve $(I - \theta\Delta\tau^k L^{(d)} + P^{k,m-1}v^{k,m} = (I + (1-\theta)\Delta\tau^k L^{(d)})v^{k-1} + P^{k,m-1}V^*$.
4: compute $P^{k,m} = P(v^{k,m})$
5: **if** $\max_j\{\frac{|v_j^{k,m} - v_j^{k,m-1}|}{\max\{1,|v_j^{k,m}|\}}\}$ or $P^{k,m-1} = P^{k,m}$ **then**
6: break
7: **end if**
8: **end for**
9: $v^k, = v^{k,m}$

For ADI methods, the MCS method remains largely unchanged, except Algorithm 3 is incorporated into the computation of the last term:

Unfortunately, since the penalty matrix is not the result of a kronecker (tensor) product, it is not straightforward to efficiently solve the linear system in Algorithm 2. However, we can still speed up the computation by using our more efficient algorithm for the other solves in each timestep.

Finally, we have the matrix form of MCS ADI timestepping for American options in d dimensions:

4 Numerical results

We show numerical results for three algorithms (Crank-Nicolson, Vectorized MCS, and Matrcized MCS) applied to four problems: European and American Geometric Average Put in two and three dimensions. This section will first show the European problems in two and three dimensions in Tables 1 - 8, then show the computed reference values for American problems (using a one-dimensional penalty iteration) in Tables 9 and 10, and then show the American problems in Tables 11-16.

The tables for European options have 6 columns, showing the number of subintervals, the number of timesteps taken, the computed value, the signed error, the computed convergence rate, and the time recorded.

Algorithm 4 MCS ADI timestepping in d dimensions

Take an explicit step:
$Y_0 = v^{k-1} + \Delta\tau^k(Lv^{k-1})$
Take d **implicit corrector steps, one in each dimension**:
for $j = 1, \ldots, d$ **do**
  compute $g = Y_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
  reorder the entries in g such that L_j is tridiagonal
  Solve the linear system $(I - \theta\Delta\tau^k L_j)Y_j = Y_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
end for
Take two more explicit updates:
$\tilde{Y}_0 = Y_0 + \sigma\Delta\tau^k L_0(Y_2 - v^{k-1})$
$\hat{Y}_0 = \tilde{Y}_0 + \mu\Delta\tau^k L(Y_2 - v^{k-1})$
Take d **implicit corrector steps in each dimension**:
for $j = 1, \ldots, d-1$ **do**
  compute $g = \tilde{Y}_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
  reorder the entries in g such that L_j is tridiagonal
  Solve the linear system $(I - \theta\Delta\tau^k L_j)\hat{Y}_j = g$
end for
compute $g = \tilde{Y}_{d-1} - \theta\Delta\tau^k L_d v^{k-1}$
reorder the entries in g such that L_d is tridiagonal
Apply Algorithm 3 to the system $(I - \theta\Delta\tau^k L_d)P^k v^k = g + P^k V^*$
reorder v^k to the original ordering

Algorithm 5 Matrix-form of MCS ADI timestepping for American options in two dimensions

Take an explicit step:
$Y_0^T = U^T + \Delta\tau^k(\rho D_y U D_x^T + L_x U^T + (UL_y)^T)$
Take two implicit corrector steps, one in each dimension
Solve the linear system $(I - \theta\Delta\tau^k L_x)Y_1^T = Y_0 - \theta\Delta\tau^k L_x U^T$
Solve the linear system $(I - \theta\Delta\tau^k L_y)Y_2 = Y_1 - \theta\Delta\tau^k L_y U$
Take two more explicit updates:
$\tilde{Y}_0 = Y_0 + \sigma\rho\Delta\tau^k D_y U D_x^T$
$\hat{Y}_0 = \tilde{Y}_0 + \mu\Delta\tau^k[\rho D_y(Y_2 - U)D_x^T + L_x(Y_2 - U)^T + ((Y_2 - U)L_y)^T]$
Take two implicit corrector steps, one in each dimension
Solve the linear system $(I - \theta\Delta\tau^k L_x)\tilde{Y}_1^T = \tilde{Y}_0^T - \theta\Delta\tau^k L_x U^T$
compute $B = \tilde{Y}_1 - \theta\Delta\tau^k L_y U$
Apply the penalty iteration (i.e. Algorithm 3) to the system
$(I - \theta\Delta\tau^k L_2) + P^k v^k = b + P^k V^*$

Algorithm 6 MCS ADI timestepping in d dimensions

Take an explicit step:
$Y_0 = v^{k-1} + \Delta\tau^k(Lv^{k-1})$
Take d implicit corrector steps, one in each dimension:
for $j = 1, \ldots, d$ **do**
  compute $g = Y_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
  compute G according to Equation (23)
  Solve the linear system $(I - \theta\Delta\tau^k L_j)Y_j = G$
end for
Take two more explicit updates:
$\tilde{Y_0} = Y_0 + \sigma\Delta\tau^k L_0(Y_2 - v^{k-1})$
$\hat{Y_0} = \tilde{Y_0} + \mu\Delta\tau^k L(Y_2 - v^{k-1})$
Take d implicit corrector steps in each dimension:
for $j = 1, \ldots, d-1$ **do**
  compute $g = \tilde{Y}_{j-1} - \theta\Delta\tau^k L_j v^{k-1}$
  compute G according to Equation (23)
  Solve the linear system $(I - \theta\Delta\tau^k L_j)\hat{Y}_j = G$
end for
compute $g = \tilde{Y}_{d-1} - \theta\Delta\tau^k L_d v^{k-1}$
reorder the entries in g such that L_d is tridiagonal
Apply Algorithm 3 to the system $(I - \theta\Delta\tau^k L_d)P^k v^k = g + P^k V^*$
reorder v^k to the original ordering

The tables for American options have 7 columns, showing the number of subintervals, the number of timesteps taken, the number of penalty iterations taken, the computed value, an estimation of the error using successive differences, an estimation of the rate of convergence using the ratio of the differences, and the time recorded.

The error and convergence rate is computed point-wise at the expiry date and the strike price, which is typically near the region of interest in financial applications, since stock prices decreasing to zero or increasing many multiple times are unlikely occurances.

4.1 Two-Dimensional European Geometric Average Put

N	timesteps	Value	Error	Rate	time(s)
10	12	8.490342	-1.32e-01	—	5.56e-02
20	22	8.590928	-3.17e-02	2.06	1.86e-01
40	42	8.614176	-8.49e-03	1.90	8.70e-01
80	82	8.620456	-2.21e-03	1.94	5.39e+00
160	162	8.622131	-5.35e-04	2.05	4.55e+01
320	322	8.622530	-1.36e-04	1.98	4.02e+02
640	642	8.622632	-3.37e-05	2.01	4.17e+03

Table 1: Two dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i_{\max}} = 1000$, $r = 0.1$. Exact solution is 8.622665388263. Solved with Crank-Nicolson.

N	timesteps	Value	Error	Rate	time(s)
10	12	8.502136	-1.21e-01	—	3.30e-02
20	22	8.585842	-3.68e-02	1.71	5.19e-02
40	42	8.613079	-9.59e-03	1.94	1.51e-01
80	82	8.620397	-2.27e-03	2.08	6.82e-01
160	162	8.622116	-5.49e-04	2.05	4.44e+00
320	322	8.622524	-1.41e-04	1.96	3.54e+01
640	642	8.622631	-3.48e-05	2.02	3.06e+02

Table 2: Two dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Exact solution is 8.622665388263. Solved with vectorized Craig-Sneyd.

N	timesteps	Value	Error	Rate	time(s)
10	12	8.502133	-1.21e-01	—	9.81e-01
20	22	8.585842	-3.68e-02	1.71	8.22e-02
40	42	8.613079	-9.59e-03	1.94	1.19e-01
80	82	8.620397	-2.27e-03	2.08	2.67e-01
160	162	8.622116	-5.49e-04	2.05	1.37e+00
320	322	8.622524	-1.41e-04	1.96	5.07e+00
640	642	8.622631	-3.48e-05	2.02	2.33e+01
1280	1282	8.622657	-8.50e-06	2.03	2.05e+02
2560	2562	8.622663	-2.15e-06	1.99	2.57e+03

Table 3: Two dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Exact solution is 8.622665388263. Solved with matricized Craig-Sneyd.

4.2 Three-Dimensional European Geometric Average Put

N	timesteps	Value	Error	Rate	time(s)
8	10	7.344038	-3.30e-01	—	8.77e-02
16	18	7.635836	-3.84e-02	3.10	2.48e+00
32	34	7.669053	-5.16e-03	2.89	2.44e+02
64	66	7.673314	-9.00e-04	2.52	1.76e+04

Table 4: Three dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Exact solution is 7.674214. Solved with Crank-Nicolson.

N	timesteps	Value	Error	Rate	time(s)
8	10	7.477913	-1.96e-01	—	1.65e-01
16	18	7.644642	-2.96e-02	2.73	2.97e-01
32	34	7.670165	-4.05e-03	2.87	2.22e+00
64	66	7.673582	-6.33e-04	2.68	3.11e+01
128	130	-47.315418	-5.50e+01	-16.41	5.48e+02

Table 5: Three dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Exact solution is 7.674214. Solved with vectorized Craig-Sneyd. $\theta = 1/2$.

N	timesteps	Value	Error	Rate	time(s)
8	10	7.477653	-1.97e-01	—	1.46e-01
16	18	7.644627	-2.96e-02	2.73	3.16e-01
32	34	7.670165	-4.05e-03	2.87	1.27e+00
64	66	7.673582	-6.33e-04	2.68	1.13e+01
128	130	-43.979294	-5.17e+01	-16.32	1.90e+02

Table 6: Three dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Exact solution is 7.674214. Solved with matricized Craig-Sneyd. $\theta = 1/2$.

We see that when $\theta = 1/2$, we do not have stability in convergence since for $N = 128$, the value is clearly wrong, indicating that the method is not stable. However, this contradicts the [8] which states that the MCS ADI method is stable in three dimensions if $\theta > 6/13$.

Since larger values of θ result in more stability, we attempt to remedy this by increasing θ to $2/3$, which achieves stability of the Craig-Sneyd method. We will use $\theta = 2/3$ for American options in the subsequent section.

N	timesteps	Value	Error	Rate	time(s)
8	10	7.485199	-1.89e-01	—	1.88e-01
16	18	7.644765	-2.94e-02	2.68	3.12e-01
32	34	7.670106	-4.11e-03	2.84	2.29e+00
64	66	7.673565	-6.49e-04	2.66	2.99e+01
128	130	7.674071	-1.44e-04	2.17	6.16e+02

Table 7: Three dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Exact solution is 7.674214. Solved with vectorized Craig-Sneyd. $\theta = 2/3$.

N	timesteps	Value	Error	Rate	time(s)
8	10	7.484872	-1.89e-01	—	1.89e-01
16	18	7.644748	-2.95e-02	2.68	3.22e-01
32	34	7.670106	-4.11e-03	2.84	1.12e+00
64	66	7.673565	-6.49e-04	2.66	9.87e+00
128	130	7.674071	-1.44e-04	2.17	2.69e+02
256	258	7.674168	-4.67e-05	1.62	4.90e+03

Table 8: Three dimensional European Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Exact solution is 7.674214. Solved with matricized Craig-Sneyd. $\theta = 2/3$.

4.3 Reference values for American option pricing problems

N	timesteps	Iterations	Value	Error	Rate	time(s)
100	39	59	9.4647541515	—	—	9.32e-03
200	73	114	9.4684135388	3.66e-03	—	1.60e-02
400	141	226	9.4692758928	8.62e-04	2.09	4.46e-02
800	277	456	9.4694864416	2.11e-04	2.03	1.42e-01
1600	547	912	9.4695394777	5.30e-05	1.99	5.05e-01
3200	1087	1778	9.4695525711	1.31e-05	2.02	1.94e+00
6400	2167	3465	9.4695558064	3.24e-06	2.02	7.06e+00
12800	4326	6943	9.4695566300	8.24e-07	1.97	2.76e+01
25600	8643	14199	9.4695568442	2.14e-07	1.94	1.12e+02

Table 9: One-dimensional American Put problem, with parameters chosen to match the two-dimensional problem.

N	timesteps	Iterations	Value	Error	Rate	time(s)
100	37	56	8.4046980424	—	—	7.55e-03
200	68	107	8.4076796836	2.98e-03	—	1.51e-02
400	132	213	8.4084829388	8.03e-04	1.89	4.17e-02
800	257	426	8.4086725800	1.90e-04	2.08	1.33e-01
1600	507	851	8.4087203268	4.77e-05	1.99	4.71e-01
3200	1006	1665	8.4087321856	1.19e-05	2.01	1.83e+00
6400	2004	3230	8.4087351562	2.97e-06	2.00	6.58e+00
12800	3999	6482	8.4087359061	7.50e-07	1.99	2.60e+01
25600	7989	13173	8.4087360994	1.93e-07	1.96	1.04e+02

Table 10: One-dimensional American Put problem, with parameters chosen to match the three-dimensional problem.

4.4 Two-Dimensional American Geometric Average Put

N	timesteps	Iterations	Value	Difference	Rate	time(s)
10	27	30	9.255044	—	—	3.50e-02
20	62	75	9.415944	-1.61e-01	—	1.45e-01
40	134	191	9.455322	-3.94e-02	2.03	1.38e+00
80	274	460	9.465989	-1.07e-02	1.88	1.54e+01
160	552	1053	9.468645	-2.66e-03	2.01	2.84e+02

Table 11: Two-Dimensional American Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Solved with Crank-Nicolson.

N	timesteps	Iterations	Value	Difference	Rate	time(s)
10	27	34	9.229663	—	—	1.44e-02
20	62	84	9.404770	-1.75e-01	—	4.64e-02
40	134	218	9.449318	-4.45e-02	1.97	3.01e-01
80	274	500	9.462960	-1.36e-02	1.71	2.32e+00
160	552	1105	9.467121	-4.16e-03	1.71	1.90e+01
320	1102	2221	9.468552	-1.43e-03	1.54	1.57e+02

Table 12: Two-Dimensional American Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Solved with Craig-Sneyd.

N	timesteps	Iterations	Value	Difference	Rate	time(s)
10	27	34	9.229664	—	—	2.28e-02
20	62	84	9.404770	-1.75e-01	—	5.07e-02
40	134	218	9.449318	-4.45e-02	1.97	1.93e-01
80	274	501	9.462960	-1.36e-02	1.71	1.34e+00
160	552	1107	9.467121	-4.16e-03	1.71	1.10e+01
320	1102	2222	9.468552	-1.43e-03	1.54	8.48e+01

Table 13: Two-Dimensional American Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Solved with Matricized Craig-Sneyd.

4.5 Three-Dimensional American Geometric Average Put

N_x	timesteps	Iterations	Value	Error	Rate	time(s)
8	10	13	8.229706	—	—	8.66e-02
16	18	30	8.369964	-1.40e-01	—	2.69e+00
32	34	66	8.395212	-2.52e-02	2.47	3.42e+02
64	66	131	8.403383	-8.17e-03	1.63	3.81e+04

Table 14: Three dimensional American Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Solved with Crank-Nicolson.

N	timesteps	Iterations	Value	Error	Rate	time(s)
8	10	27	8.230826	—	—	5.96e-02
16	18	57	8.355286	-1.24e-01	—	1.83e-01
32	34	122	8.387251	-3.20e-02	1.96	2.37e+00
64	66	226	8.399088	-1.18e-02	1.43	3.59e+01
128	130	448	8.404125	-5.04e-03	1.23	7.71e+02

Table 15: Three dimensional American Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Solved with vectorized Craig-Sneyd. $\theta = 2/3$.

N	timesteps	Iterations	Value	Error	Rate	time(s)
8	10	35	8.231054	—	—	7.24e-02
16	18	70	8.355287	-1.24e-01	—	1.30e-01
32	34	143	8.387251	-3.20e-02	1.96	1.56e+00
64	66	274	8.399088	-1.18e-02	1.43	2.29e+01
128	130	539	8.404125	-5.04e-03	1.23	5.69e+02

Table 16: Three dimensional American Geometric Average Put problem, $\sigma_i = 0.4$, $\rho = 0.2$, $K_i = 100$, $S_{i\max} = 1000$, $r = 0.1$. Solved with matricized Craig-Sneyd. $\theta = 2/3$.

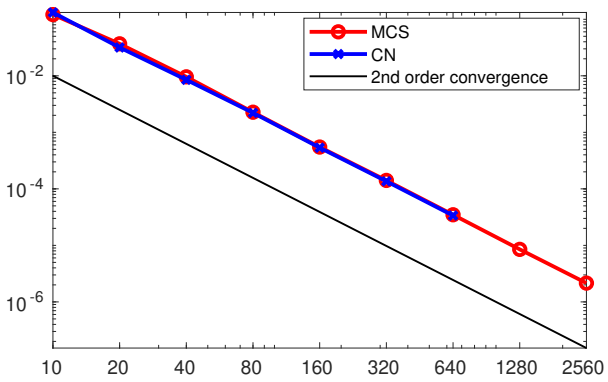
5 Convergence and runtime analysis

We will now analyze the convergence and runtime of our algorithms. Recall that both MCS and CN are supposed to be second order accurate, and this is shown in Figure 1. For two and three dimensional European options, it is clear that the convergence is second order. For American options, it is known from [6] that the penalty method for the one-dimensional problem does not exhibit second order convergence unless variable timesteps are used. We adapted the variable timestep method to the two-dimensional problem to help fix the rate of convergence. However, more work is required for it to work with three dimensions so we left that as future work.

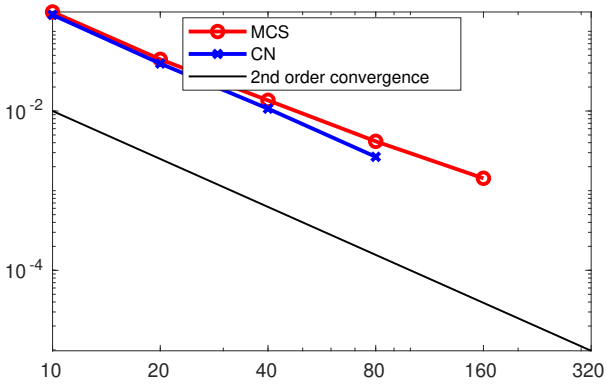
In Figure 2 we show the time complexity of our methods. Note that we expect that the runtime is $\mathcal{O}(N)$ for the solving of a linear system in each timestep in the MCS method for two dimensions, however, we still have to operate on $\mathcal{O}(N^2)$ number of inputs so the runtime is expected to be $\mathcal{O}(N^3)$ since we are taking $\mathcal{O}(N)$ timesteps. For the CN method, the solving of a banded linear system is $\mathcal{O}(N^4)$, however matlab's backslash operator uses LU with full pivoting which reduces fill-in and therefore reduces the runtime of the solving of the linear system. However, we still see that CN is significantly slower than the ADI methods for a two-dimensional European option.

For the two-dimensional American option, we see that the runtime of the CN seems to be $\mathcal{O}(N^4)$ and the ADI method is $\mathcal{O}(N^3)$ as expected.

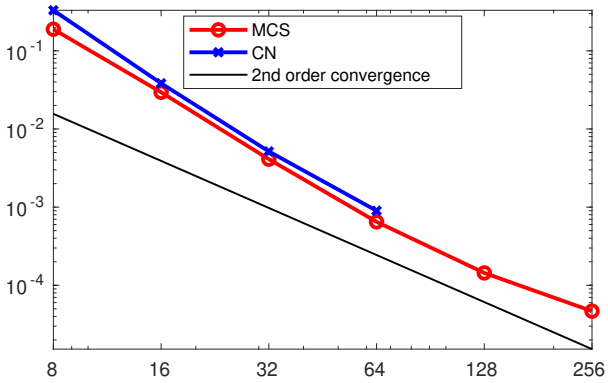
For three-dimensional problems, we expect the runtime of CN to be $\mathcal{O}(N^7)$ per iteration for the banded solve, therefore the total runtime is $\mathcal{O}(N^8)$. We also expect the runtime of the ADI method to be $\mathcal{O}(N^3)$ per solve, which leads to $\mathcal{O}(N^4)$ total runtime for the ADI method. From the bottom half of Figure 2 it seems that both the European and the American problems have approximately the expected runtime. As mentioned before, the use of an ADI method improves the time complexity per iteration from $\mathcal{O}(N^{d(d-1)^2})$



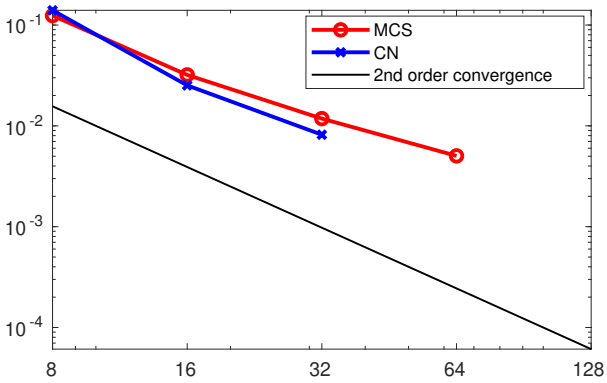
(a) Two-dimensional European option



(b) Two-dimensional American option

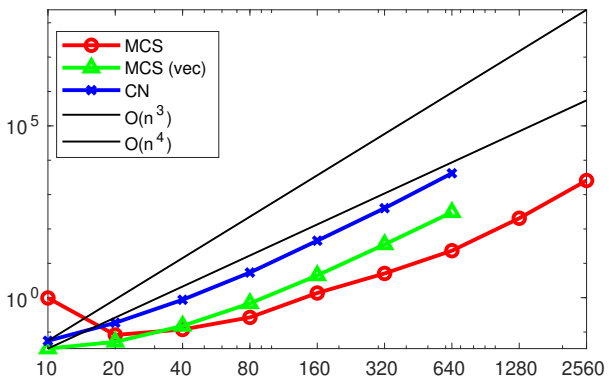


(c) Three-dimensional European option

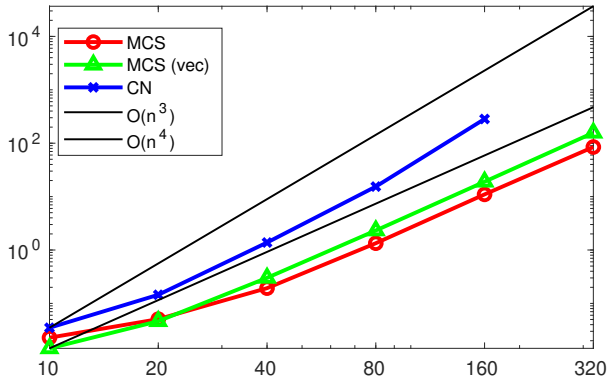


(d) Three-dimensional American option

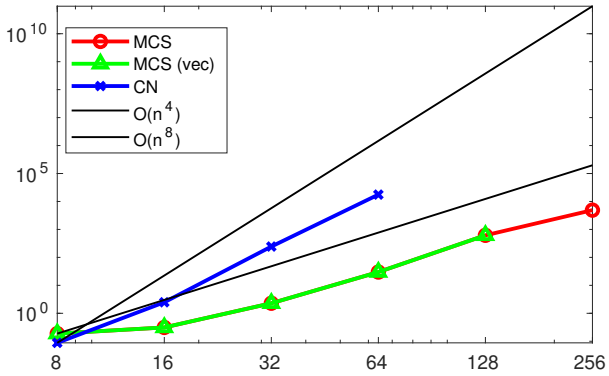
Figure 1: Convergence study for European (left) and American (right) options, in both two (top) and three (bottom) dimensions.



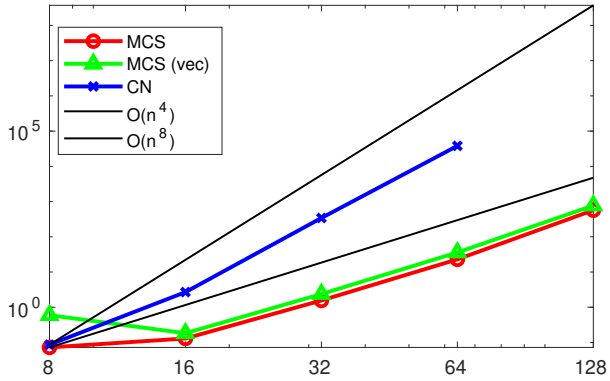
(a) Runtime of two-dimensional European option



(b) Runtime of two-dimensional American option



(c) Runtime of three-dimensional European option



(d) Runtime of three-dimensional American option

Figure 2: Runtime analysis for European (left) and American (right) options, in both two (top) and three (bottom) dimensions.

to $\mathcal{O}(N^d)$ (assuming that a banded solver is used) by reducing the bandwidth to the minimum required. Our experimental results support this theory, although with the use of a general sparse LU solver and other expensive operations such as computing some dense matrix-matrix products the experimental results don't agree entirely with the theory, as is expected.

6 Conclusions

We see that the ADI method is a powerful tool which allows us to minimize the complexity of computing the solution at the next timestep. The time complexity of the MCS ADI method is linear in the number of unknowns that we are required to solve, which is asymptotically optimal and cannot be reduced further. We also see that when solving the linear systems, we can exploit the structure of the matrix (it is block diagonal with identical tridiagonal blocks) to reduce the runtime significantly in the case of European options. In the case of American options, the penalty matrix destroys the block-diagonal structure with identical blocks, so we cannot avoid solving a "full size" matrix. Finally, we see that the variable timestepping method introduced in [6] doesn't extend that well to two-dimensions when using ADI methods and do not work in three-dimensions. Additionally, our understanding of the methods according to [8] is incomplete, since according to the analysis in the papers $\theta \geq 6/13$ is required for stability of the MCS ADI method in three dimensions, however we do not see stability with $\theta = 1/2$ and need to increase θ to $2/3$ to see stability with the three-dimensional problem.

References

[1] N. CLARKE AND K. PARROTT, *Multigrid for American option pricing with stochastic volatility*, Applied Mathematical Finance, 6 (1999), pp. 177–195.

[2] I. J. CRAIG AND A. D. SNEYD, *An alternating-direction implicit scheme for parabolic equations with mixed derivatives*, Computers & Mathematics with Applications, 16 (1988), pp. 341–350.

[3] T. A. DAVIS, *Algorithm 832: UMFPACK v4. 3—an unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software (TOMS), 30 (2004), pp. 196–199.

[4] J. DOUGLAS AND H. H. RACHFORD, *On the numerical solution of heat conduction problems in two and three space variables*, Transactions of the American mathematical Society, 82 (1956), pp. 421–439.

[5] J. DOUGLAS, JR, *On the numerical integration of* $\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 = \partial u/\partial t$ *by implicit methods*, Journal of the society for industrial and applied mathematics, 3 (1955), pp. 42–65.

[6] P. FORSYTH AND K. VETZAL, *Quadratic convergence for valuing American options using a penalty method*, SIAM J. Sci. Comput., 23 (2002), pp. 2095–2122.

[7] V. HEIDARPOUR-DEHKORDI AND C. CHRISTARA, *Spread option pricing using ADI methods*, International J. Numerical Analysis and Modelling, 15 (2018), pp. 353–369.

[8] K. IN'T HOUT AND B. WELFERT, *Stability of ADI schemes applied to convection–diffusion equations with mixed derivative terms*, Applied numerical mathematics, 57 (2007), pp. 19–35.

[9] R. RANNACHER, *Finite element solution of diffusion problems with irregular data*, Numerische Mathematik, 43 (1984), pp. 309–327.