UCD Smurfit Graduate School of Business

Numerical Analytics and Software

Programming Assignment 2

Authors:
Eoghan Aherne
Louis Carnec
Conor Reid

Lecturer:
Dr. Sean McGarraghy

November 20, 2016

Contents

1 Overview

1.1 Mathematical Issues

1.1.1 4.0

1.1.2 4.1

1.2 Applications

1.3 Questions

useful to check for convergence of the residual to zero?
what matrix norms might be useful and why?
scaling be helpful?
which matrices do you get expect convergence and which do not? why?
effect of different values of tolerance?
effect of an ill-conditioned matrix?

2 Design

2.1 4.0

2.2 4.1

The implementation of the Black-Scholes-Merton pricing problem found in the module `tridiagM.py` takes advantage of the fact that the Black-Scholes differential equation as a system of linear equations ($\mathbf{Af} = \mathbf{b}$) can be written as a tridiagonal $(N-1) \times (N-1)$ matrix. This tridiagonal matrix, $\mathbf{A}$, which by definition is a sparse matrix, can be solved using the Sparse SOR algorithm which solves for vector $\mathbf{x}$ the system of linear equations $\mathbf{Ax} = \mathbf{b}$ by successive iteration.
We are able to approximate all option prices for a given stock price for all $m \in M$ by approximating the previous time step using a system of linear equations. There are two approximation in finding option prices for all time steps m.
The first of these entails approximating put option prices at time T given Stock prices and a known put option strike price, X. Using this approximation for $f_{n,M}$, we can solve the system of linear equation for option prices at previous time periods $f_{n,m}$.
Given that the matrix $\mathbf{A}$ represents the finite difference constants relating $f_{n,m-1}$ to $f_{n,m-1}$ and that we can approximate the N European put option prices at $f_{n,M}$, where $M = T$ and time at which the option is at maturity, given the fact that the portfolio must
$P = -f + \frac{\partial f}{\partial S} S \left(-\sigma S \frac{\partial f}{\partial S} + \sigma S \frac{\partial f}{\partial S} \right) dz = 0$

2.2.1 Creating the Grid

To solve for option prices at all time steps we need create a grid of stock prices (S, the stock price vector) and of M time steps until the option reaches maturity (T). Options have expiration dates or maturity. For example, an option expires in 12 days (T=12) and since our time steps is days ($k = T/M = 1$) we have that M=12. If our time step was half a day (k=0.5) for the same expiry date in 12 days, the grid would have 24 time steps ($M = 12 \times 0.5$). Stock prices can be any non-negative real number. However the grid must start at 0 since the put option price is the strike price when the stock price is zero ($f_{0,m} = X$).
A function (`createSandT`) was created to return two vectors for stock prices and times.

2.3 Implementation issues

2.4 Design Decisions

3 Testing

Appendix

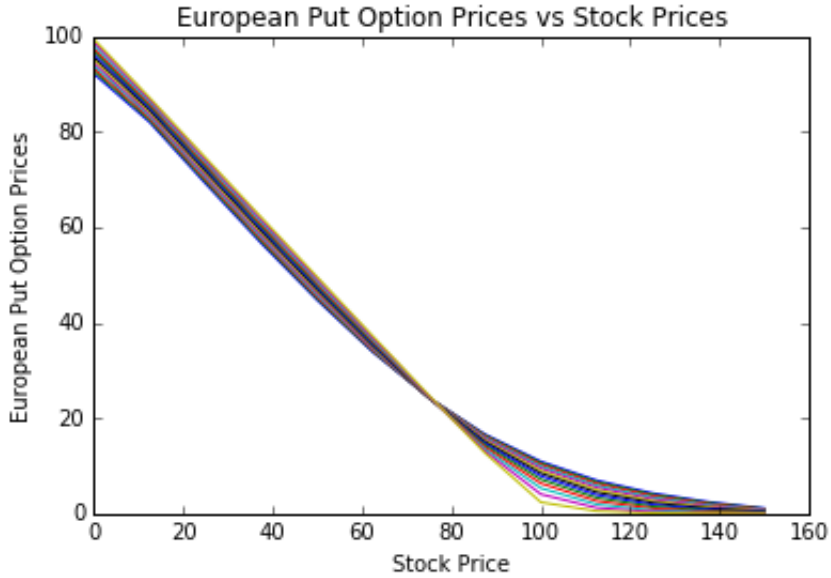

Figure 1: Distribution of put option prices against stock prices for m $T = 12$, $M = 12$, $k = 1$

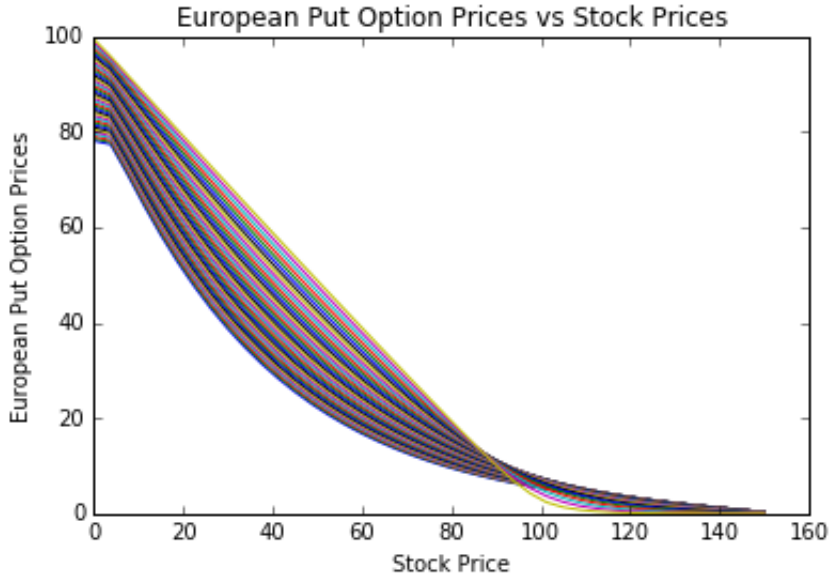

Figure 2: Distribution of put option prices against stock prices for m $T = 12$, $M = 40$, $k = 0.3$

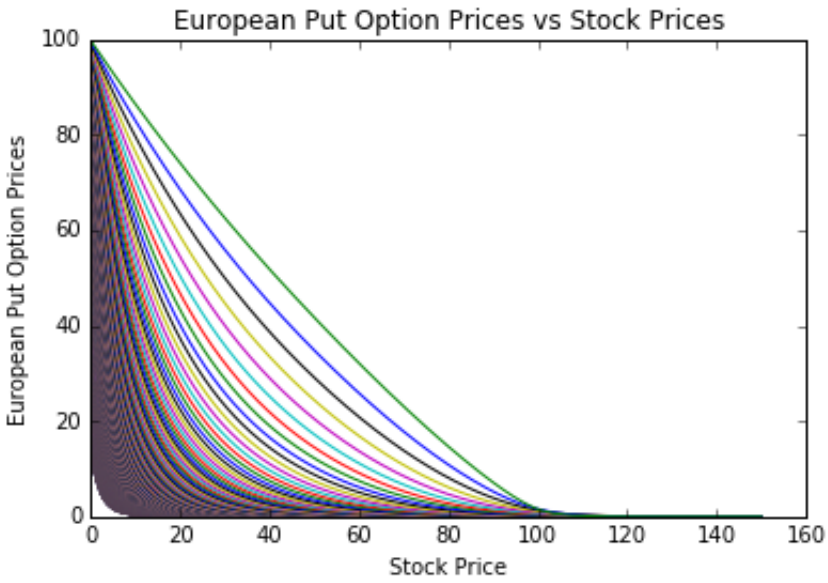

Figure 3: Distribution of put option prices against stock prices for m $T = 12$, $M = 365$, $k = 0.033$

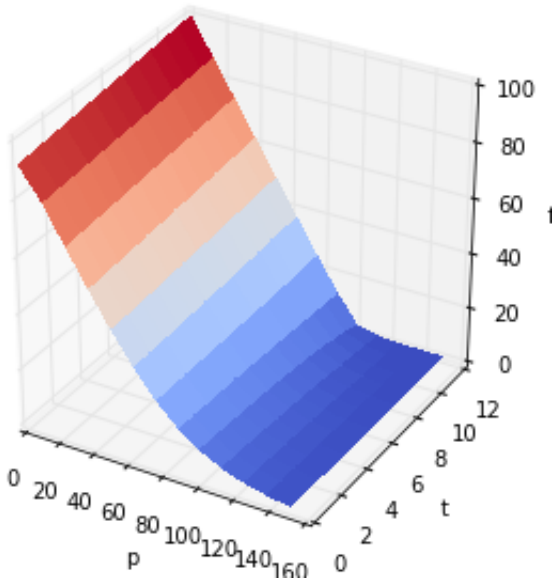

Figure 4: 3D Plot of European Put Option Prices for $T = 12$, $M = 12$, $k = 1$

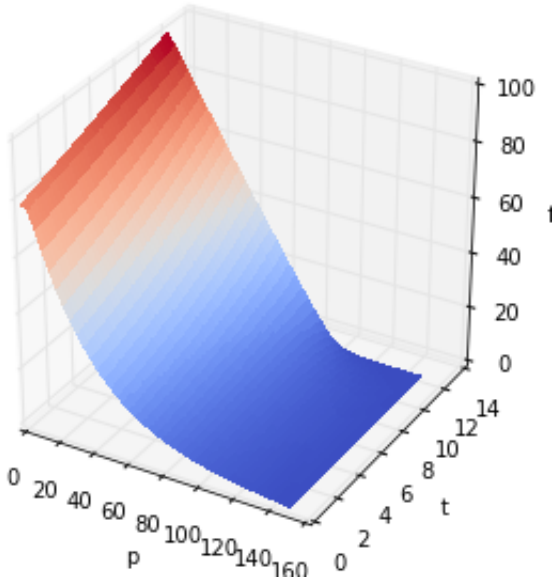

Figure 5: 3D Plot of European Put Option Prices for $T = 12$, $M = 40$, $k = 0.3$

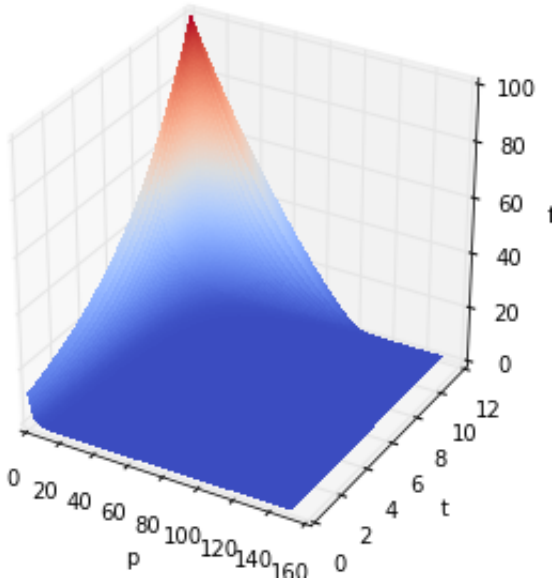

Figure 6: 3D Plot of European Put Option Prices for $T = 12$, $M = 365$, $k = 0.033$

References

[1] Some Reference