|  |
| --- |
|  |
| QF4102 Assignment 3 Report |
|  |
| Financial Modeling |

**Prepared by: Meng Xiang A0133998L**

2016-11-13

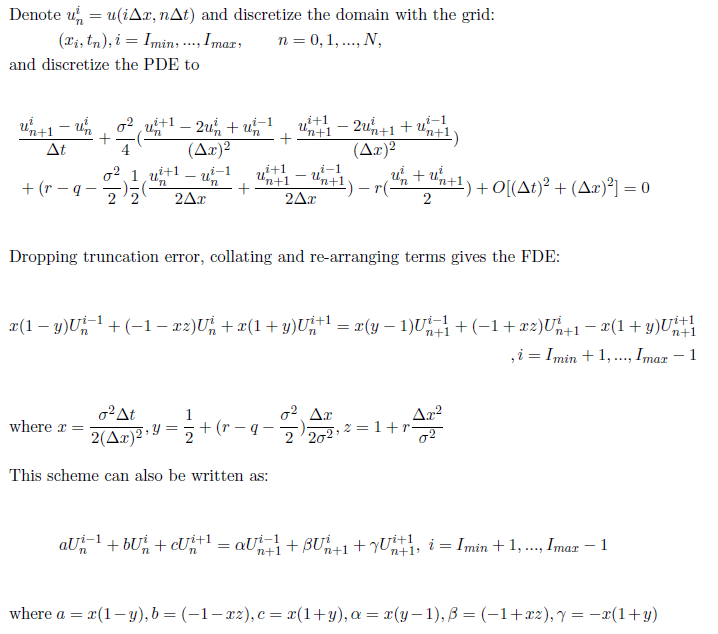
QF4102 Assignment 3 Report

Financial Modeling

All scripts/codes can be found in the appendix and the rar file.

# Section 1

1. In this section, the Crank-Nicolson finite difference scheme for the transformed PDE was derived. The derivation is shown as follows:

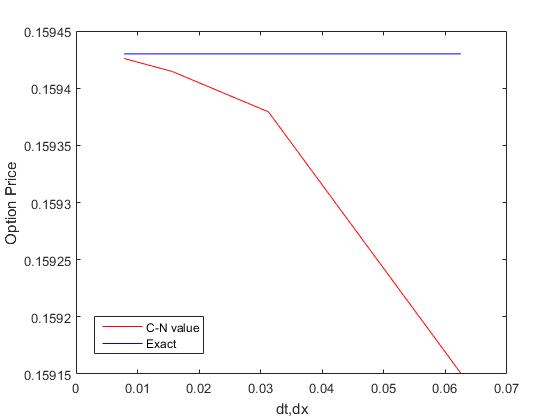


In the derivation above, we followed the same manner as the example in the lecture notes (Page 32-34, Chapter 3) to simplify the equation. A “.tex” file with the name “C-N\_Transformed.tex” can be found in the “tex” folder.

1. Testing the implementation with a European put option which has a strike price of $1, time to maturity of 1 year, and the current asset price is $0.95, volatility of asset return is 40%, dividend yield is 2% and risk free rate is 5%, the following results are obtained (x lies in [-5,2]):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1/16 | 1/32 | 1/64 | 1/128 |
| **Value** | 0.15915 | 0.15938 | 0.15941 | 0.15943 |

In order to see the relationship clearly, we also plotted the values for different versus the exact value 0.15943:



Note that we shrunk Δt,Δx each time, i.e., the trend is opposite to the x-direction. Hence, from the table and the plot above, we conclude that the approximated value is getting closer to the true value when the mesh (Δt,Δx) is refined. Intuitively, a finer grid gives a more accurate result.

Please check the appendix or the code “CN\_put\_transformed.m” for the detail code, and please refer to “A3\_1\_ii.m” for the implementation and plot.

1. Since we did not have a proxy, the second method given in the lecture notes (Page78-79, Chapter 3) was applied. Note that we then modified function in (ii) to export vectors at time 0.

Set , and other parameters to be the same as those in part (ii). The result is tabulated as follows:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | **Ratio** |
| 1/16 | 1/16 |  | 4.0151 |
| 1/32 | 1/32 |  | 4.0034 |
| 1/64 | 1/64 |  | 4.0007 |
| 1/128 | 1/128 |  |  |
| 1/256 | 1/256 |  |  |

Note that the ratio converges to 4 when the mesh (Δt,Δx) is refined, which confirms the expected result.

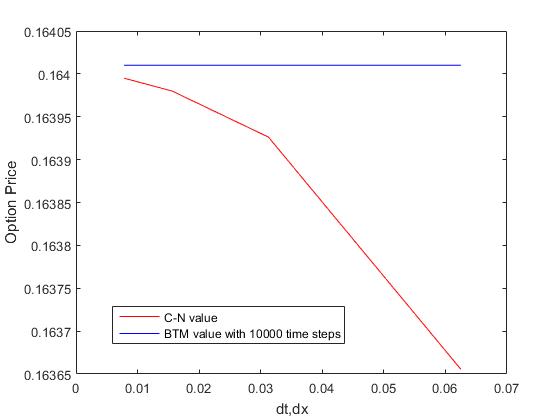
Please refer to “A3\_1\_iii.m” for the implementation.

1. In this section, we still applied the Crank-Nicolson finite difference scheme, but to price an **American** vanilla put option. The **PSOR algorithm** is involved here.

Using the same parameters as part (ii), the following result is obtained:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 1/16 | 1/32 | 1/64 | 1/128 |
| **Value** | 0.16366 | 0.16393 | 0.16398 | 0.16399 |

In order to see the relationship clearly, we also plot the values for different against the exact value:



From the table and the plot above, a similar conclusion can be drawn that the approximated value is getting closer to the (nearly) true value when the mesh (Δt,Δx) is refined. Again, this is because we discretize time and price more smoothly.

Please check the appendix or the code “CN\_put\_transformed\_PSOR.m” for the detail code, and please refer to “A3\_1\_iv.m” for the implementation and plot.

# Section 2

In this section, we price a 3-asset option using the Monte-Carlo simulation.

(i)&(ii) For each strike price, we used a total of 30 simulation runs with 100, 1000, 10000, 100000 price-path bundles to generate 30 mean option prices. Then we use the mean of these 30 prices to approximate the option price. The prices and standard errors are summarized as follows:

X=$12:

|  |  |  |
| --- | --- | --- |
| **N (No. of path-bundles)** | **Estimated Price** | **Standard Error** |
| 100 | 3.4513 | 0.2044 |
| 1000 | 3.4633 | 0.0660 |
| 10000 | 3.4529 | 0.0213 |
| 100000 | 3.4525 | 0.0064 |

X=$13:

|  |  |  |
| --- | --- | --- |
| **N (No. of path-bundles)** | **Estimated Price** | **Standard Error** |
| 100 | 4.4034 | 0.2067 |
| 1000 | 4.3709 | 0.0632 |
| 10000 | 4.3576 | 0.0221 |
| 100000 | 4.3666 | 0.0072 |

X=$14:

|  |  |  |
| --- | --- | --- |
| **N (No. of path-bundles)** | **Estimated Price** | **Standard Error** |
| 100 | 5.2945 | 0.2554 |
| 1000 | 5.2944 | 0.0756 |
| 10000 | 5.3099 | 0.0242 |
| 100000 | 5.3075 | 0.0072 |

From the table above, we observe that:

1. Unlike other models, there is no monotonicity for prices when N increases. This is a result of randomness of Monte-Carlo simulation.
2. However, when N increases, the price should converge to the correct value. At the same time, the standard error is reduced.

Please check the appendix or the code “MC\_3put\_min.m” for the detail code, and please refer to “A3\_2\_i.m” for the implementation.

(iii) In order to reduce the variance, we chose a combination of 3 European vanilla put with equal weight to be our control variate. The payoff for the portfolio is

The prices and standard errors are shown as follows:

X=$12:

|  |  |  |  |
| --- | --- | --- | --- |
| **N (No. of path-bundles)** | **Estimated Price** | **Standard Error** | **Standard Error (w/o control variate)** |
| 100 | 3.4392 | 0.1366 | 0.2044 |
| 1000 | 3.4576 | 0.0478 | 0.0660 |
| 10000 | 3.4505 | 0.0157 | 0.0213 |
| 100000 | 3.4536 | 0.0053 | 0.0064 |

X=$13:

|  |  |  |  |
| --- | --- | --- | --- |
| **N (No. of path-bundles)** | **Estimated Price** | **Standard Error** | **Standard Error (w/o control variate)** |
| 100 | 4.3536 | 0.1888 | 0.2067 |
| 1000 | 4.3661 | 0.0555 | 0.0632 |
| 10000 | 4.3693 | 0.0129 | 0.0221 |
| 100000 | 4.3667 | 0.0052 | 0.0072 |

X=$14:

|  |  |  |  |
| --- | --- | --- | --- |
| **N (No. of path-bundles)** | **Estimated Price** | **Standard Error** | **Standard Error (w/o control variate)** |
| 100 | 5.2800 | 0.1416 | 0.2554 |
| 1000 | 5.3009 | 0.0494 | 0.0756 |
| 10000 | 5.2914 | 0.0135 | 0.0242 |
| 100000 | 5.3086 | 0.0055 | 0.0072 |

Comparing with the result in part (i)&(ii), where the control variate was not implemented, it is clear that adding a control variate does help us to reduce the standard error. In addition, the estimated prices are quite close to the ones we obtained without the control variate. Above results verifies proofs in lecture notes.

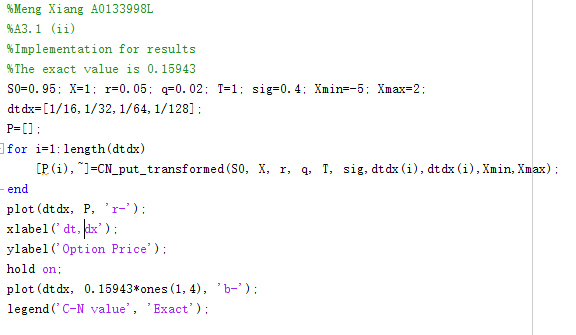
Please check the appendix or the code “MC\_3put\_min\_CV.m” for the detail code, and please refer to “A3\_2\_iii.m” for the implementation.

# Appendix: Screen shot of scripts

## A3.1(ii):

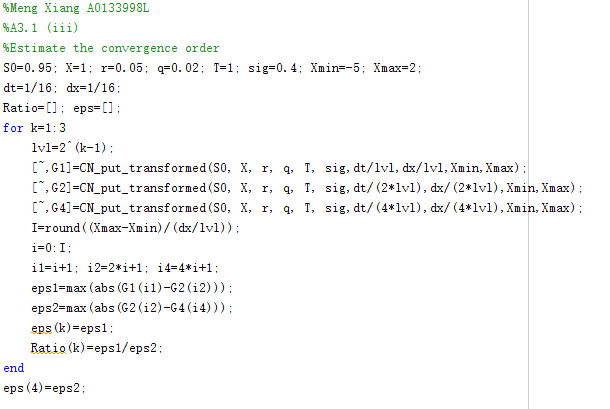
##### Function:

##### Implementation:



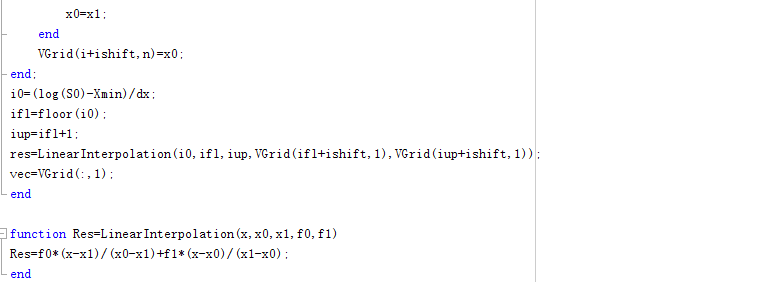
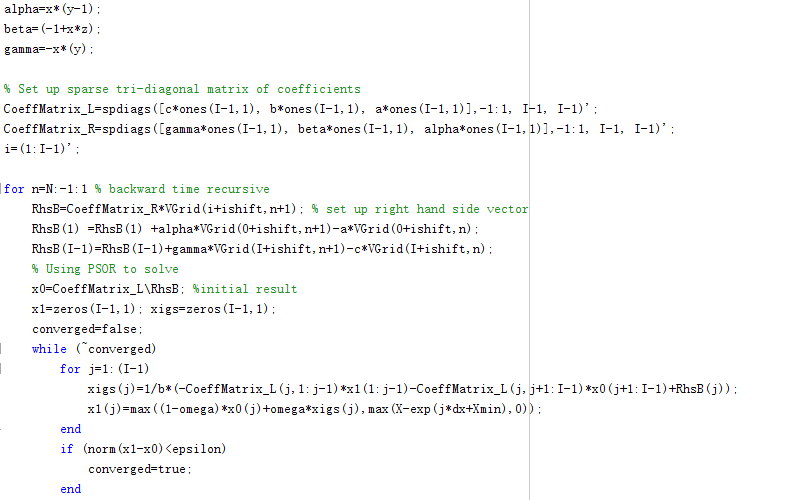
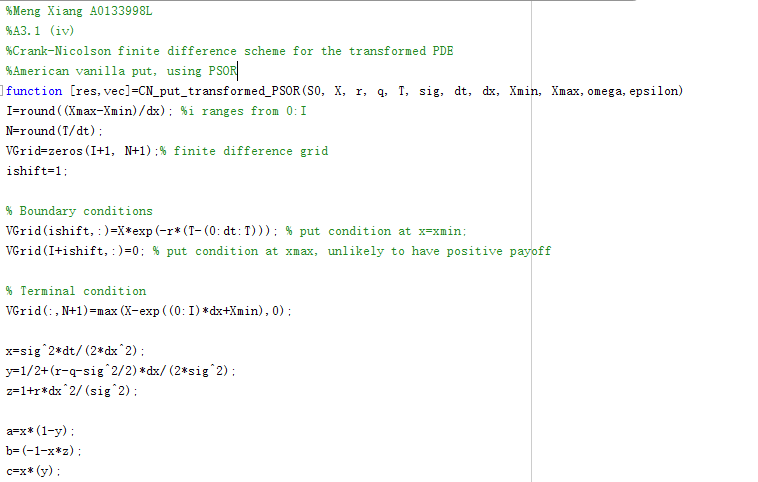
## A3.1(iii):

##### Implementation:

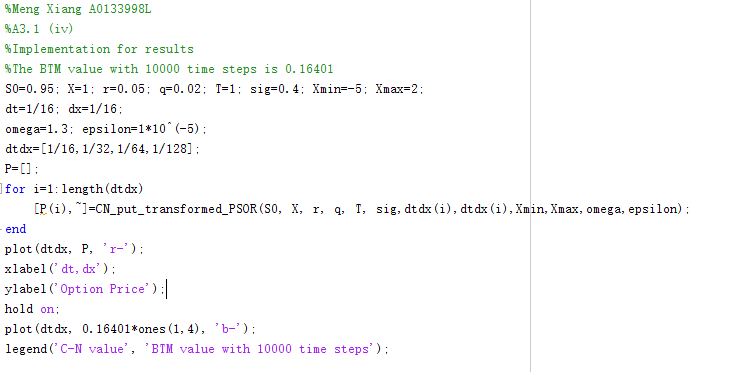


## A3.1(iv):

##### Function:

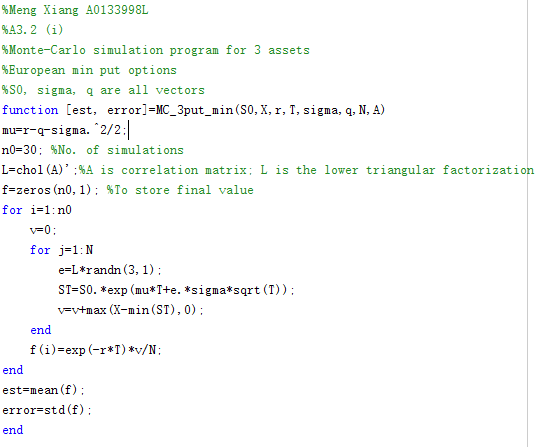


##### Implementation:

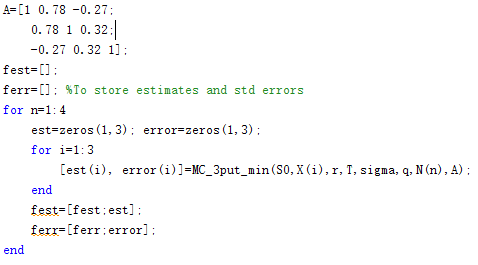
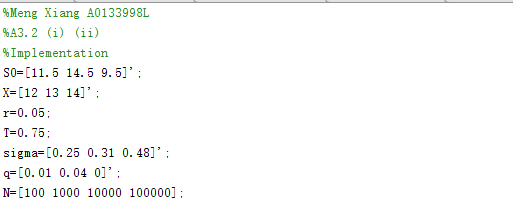


## A3.2(i)&(ii):

##### Function:

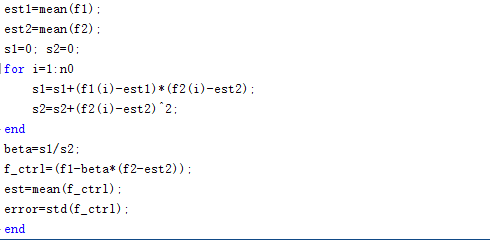
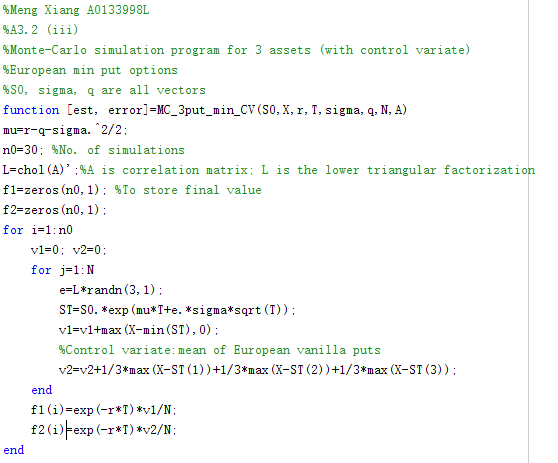


##### Implementation:



## A3.2(iii):

##### Function:



##### Implementation:

