Homework Assignment 3
FE 621: Computational Methods in Finance
Instructor: Ionut Florescu
4/21/2019

Rukmal Weerawarana
rweerawa@stevens.edu | 104-307-27
Department of Financial Engineering
Stevens Institute of Technology

1 Quadratic Volatility Model

1.1 Part (a)

Analyzing Figure 1, it is clear that the transition density increases commensurately with converging values of x and x_0.

Additionally, the transition density appears to increase significantly as the time to maturity, t decreases. This is particularly evident when comparing the maximum values of Figure 1 Panel (a) to Figure 1 Panel (d), whose maximum volatility transition density appears to be barely half of that of Panel (a) at its peak.

1.2 Part (b)

Absolute Difference between PDE and Finite Difference Approximation
7.598788770759247e-27

Verifying that the finite difference approximation of the transition probability density satisfies the initial Partial Differential Equation. The absolute value of the difference between the Finite Difference approximation and the PDE value is displayed above.

1.3 Part (c)

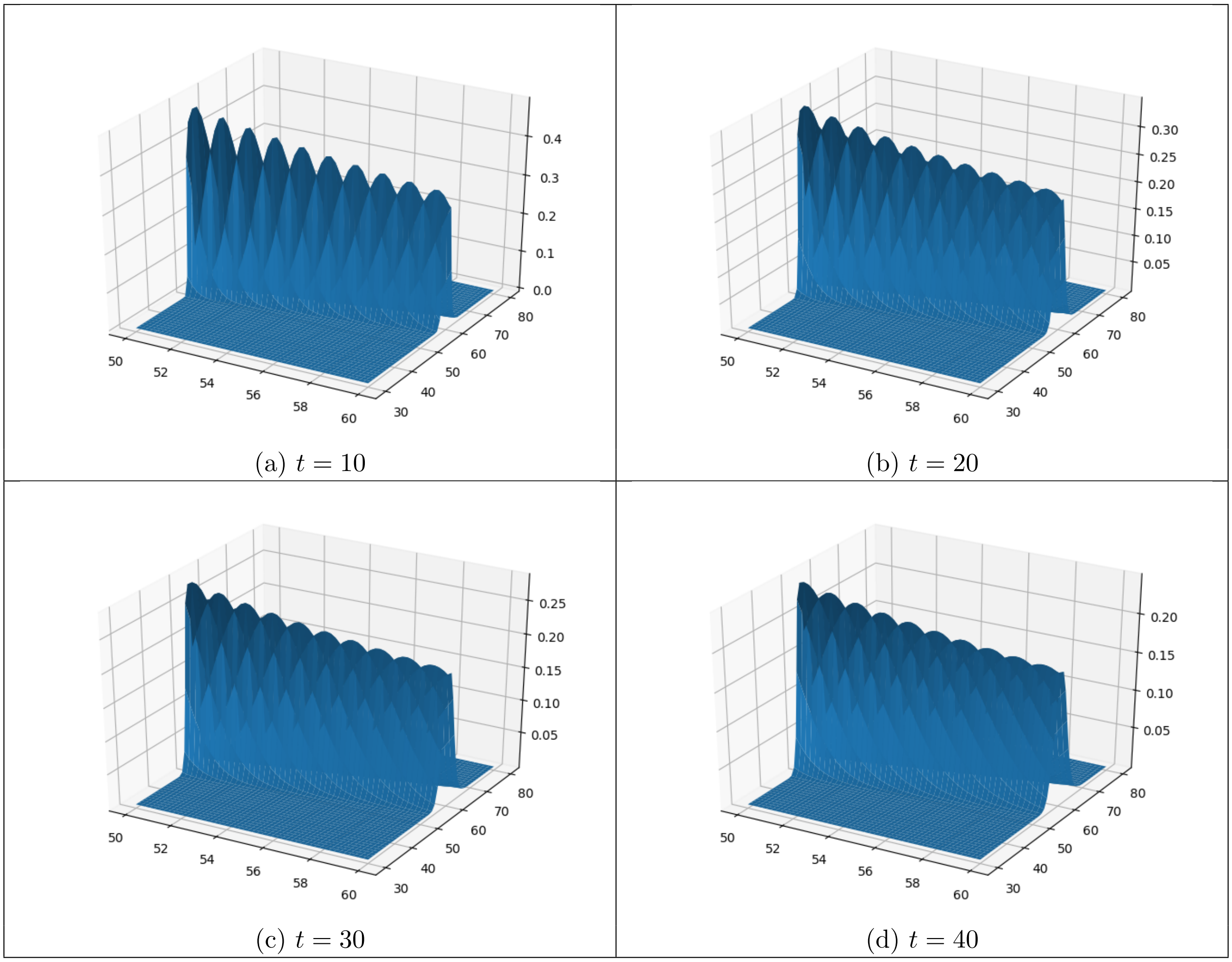
Black Scholes Price	Quadratic Volatility Process Described Price
5.06712184	4.99987481

Table 1: European Call Option priced with the Quadratic Volatility and Black Scholes models.

2 Fast Fourier Transform

	Value
Fast Fourier Transform Price	12.732485315787473
Black Scholes Price	12.82158139269142
Difference	0.08909607690394772
% Difference compared to BS	8.91%

Table 2: European Call Option priced with the Fast Fourier Transform and Black Scholes models.



(a) $t = 10$ (b) $t = 20$

(c) $t = 30$ (d) $t = 40$

Figure 1: Surface plots of Quadratic Volatility Models with varying times, t.

3 Solution Source Code

3.1 Question 1 Solution

3.1.1 Quadratic Volatility Plots

```python
from context import fe621

from mpl_toolkits.mplot3d import Axes3D
from typing import Callable
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd


# Setting parameters
alpha = 1e-4
beta = 1e-4
gamma = -1e-4
N = 100000

# Computing Q(t)
def Q(alpha: float=alpha, beta: float=beta, gamma: float=gamma) -> float:
    return ((alpha * gamma) / 2) - (np.power(beta, 2) / 8)

# Sigma(x)
def sigma(x: float, alpha: float=alpha, beta: float=beta, gamma: float=gamma) \
    -> float:
    return alpha * np.power(x, 2) + (beta * x) + gamma


# s(x) space-domain transformation
def s_integrand(x: float) -> float:
    return 1 / sigma(x)


# Note: Not using package function here as it is not compatible with
#       numpy meshgrid objects.
def trapezoidalRule(f: Callable, a: np.array, b: np.array, n: int) -> np.array:
    h = (b - a) / n
    integral = (0.5 * f(a)) + (0.5 * f(b))
    for i in range(1, n):
        integral += f(a + (i * h))
    integral *= h
    return integral


# Defining transformed CDF
def probability(x: float, x0: float, t: float) -> float:
    return 1 / (sigma(x) * np.sqrt(2 * np.pi * t)) * (sigma(x0) / sigma(x)) \
        * np.exp((-0.5 / t * np.power(
        trapezoidalRule(s_integrand, x0, x, N), 2)) + Q() * t)

# Defining points
x = np.linspace(50, 60)
x0 = np.linspace(30, 80)

# Building meshgrid of points for evaluation
x, x0 = np.meshgrid(x, x0)

```

```
# Defining vector of t's for plotting
t_vec = np.arange(10, 41, 10)

# Part (a) Quadratic Vol Plots

for t in t_vec:
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    transition_prob = probability(x=x, x0=x0, t=t)
    ax.plot_surface(x, x0, transition_prob)
    plt.tight_layout()
    plt.savefig(fname='Homework 3/bin/q1_quadvol_t_{0}.png'.format(t))
    plt.close()


# Part (b)

# Verifying that the finite difference approxiumations of the transition
# probability density satisfies the PDE

# Partial of density w.r.t. time
def partialT(x, x0, t, delT):
    return (probability(x, x0, t + delT) - probability(x, x0, t)) / delT

# Partial of density w.r.t. price
def partialX(x, x0, t, delX):
    return (probability(x + delX, x0, t) - probability(x, x0, t)) / delX


delX = delT = 1e-3
x = 50
x0 = 40
t = 20

# Computing difference
diff = np.abs(partialT(x, x0, t, delT) - (np.power(sigma(x), 2) * 0.5 *
    partialX(x, x0, t, delX)))

# Saving to CSV file
pd.DataFrame({
    'Absolute Difference between PDE and Finite Difference Approximation': \
        [diff]
}).to_csv('Homework 3/bin/q1_finite_diff_approx_verification.csv', index=False)
```

question_solutions/q1_qvol_plots.py

3.2 Call Option Pricing

```
from context import fe621

from scipy.stats import norm
from typing import Callable
import numpy as np
import pandas as pd


# Setting parameters
alpha = 1e-4
beta = 1e-4
```

```
gamma = -1e-4
N = 100000

# Computing Q(t)
def Q(alpha: float=alpha, beta: float=beta, gamma: float=gamma) -> float:
    return ((alpha * gamma) / 2) - (np.power(beta, 2) / 8)

# Sigma(x)
def sigma(x: float, alpha: float=alpha, beta: float=beta, gamma: float=gamma) \
    -> float:
    return alpha * np.power(x, 2) + (beta * x) + gamma

# s(x) space-domain transformation
def s_integrand(x: float) -> float:
    return 1 / sigma(x)


# Note: Not using package function here as it is not compatible with
#       numpy meshgrid objects.
def trapezoidalRule(f: Callable, a: np.array, b: np.array, n: int) -> np.array:
    h = (b - a) / n
    integral = (0.5 * f(a)) + (0.5 * f(b))
    for i in range(1, n):
        integral += f(a + (i * h))
    integral *= h
    return integral

def qvolCall(T: float, K: float, x0: float):
    s = np.abs(trapezoidalRule(s_integrand, x0, K, N))
    return np.maximum(x0 - K, 0) + ((sigma(K) * sigma(x0)) / (2 * np.sqrt(-2 *
        Q())) * ((np.exp(s * np.sqrt(-1 * Q())) * norm.cdf((-1 * s / np.sqrt(2 *
        T)) - np.sqrt(-2 * Q() * T))) - (np.exp(-1 * s * np.sqrt(-1 * Q())) *
        norm.cdf((-1 * s / np.sqrt(2 * T)) + np.sqrt(-2 * Q() * T)))))


# Let the candidate option have the following characteristics:
S = 105
K = 100
vol = 0.03
T = 1.
rf = 0

bs_price = fe621.black_scholes.call(
    current=S,
    volatility=vol,
    ttm=T,
    strike=K,
    rf=rf
)

qvol_price = qvolCall(T=T, K=K, x0=S)


pd.DataFrame({
    'Black Scholes Price': [bs_price],
    'Quadratic Volatility Process Described Price': [qvol_price]
}).round(decimals=8).to_csv(
    'Homework 3/bin/q1_call_option_prices.csv', index=False)
```

question_solutions/q1_call_option.py

3.3 Question 2 Solution

```
from context import fe621

import numpy as np
import pandas as pd


# Option characteristics
S = 100
K = 100
vol = 0.3
T = 1.
rf = 0.02

# FFT parameters
alpha = 1.1
N = 4096
k = np.log(K)
b = np.ceil(k)
lmbda = 2 * b / N
eta = 2 * np.pi / (N * lmbda)

# Values
x_j = np.zeros(N)
X_j = np.zeros(N)
k_u = np.array([-b + (lmbda * i) for i in range(0, N)])


# Phi
def phi(v, i):
    return np.exp(np.complex(0, np.complex(v, -(alpha + 1))) * (np.log(S) +
        (rf - 0.5 * vol) * T * i / N) - (0.5 * np.power(vol, 2) *
        np.power(np.complex(v, -(alpha + 1)), 2)))

# Psi
def psi(v, i):
    return (np.exp(-rf * T * i / N) * phi(v, i)) / np.complex(np.power(alpha, 2) + alpha -
        np.power(v, 2), ((2 * alpha) + 1) * v)

# Computing adjusted values
for j in range(0, N):
    x_j[j] = np.exp(np.complex(0, b * eta * j)) * psi(j * eta, j) * eta

# Performing Fast Fourier Transform
X_j = np.fft.fft(x_j)

# Computing call option prices
C_k = np.exp(-alpha * k_u) / np.pi * X_j

# Isolating most accurate estimate
# for i in range(N):
#     if (np.abs(k_u[i] - np.log(K)) < 0.01):
#         print(i)
#         print(C_k[i].real)


# Isolting most accurate estimate
minarg = np.argmin(np.abs(k_u - k))
fft_price = C_k[minarg].real

```

```
# Computing traditional black-scholes price
bs_price = fe621.black_scholes.call(
    current=S,
    volatility=vol,
    ttm=T,
    strike=K,
    rf=rf
)

diff = np.abs(bs_price - fft_price)

# Building output dataframe, saving to CSV
pd.DataFrame({
    'Fast Fourier Transform Price': [fft_price],
    'Black Scholes Price': [bs_price],
    'Difference': [np.abs(diff)],
    '% Difference compared to BS': [str(round(diff * 100, 2)) + '%']
}, index=['Value']).T.round(decimals=7).to_csv(
    'Homework 3/bin/q2_price_comparison.csv')
```

question_solutions/q2_fft.py

References

Carr, Peter, and Dilip B Madan. n.d. *Option valuation using the fast Fourier transform.* Technical report. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.348.4044%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

Florescu, Ionut. 2019. "1.11.4 Simpson's Rule." Chap. 1 in *Computational Methods in Finance,* 25–26. Hoboken, NJ.

Shreve, Steven E. 2004. *Stochastic Calculus for Finance II.* 153–164. April. Pittsburgh, PA: Springer Finance. ISBN: 0-387-40101-6.

Stefanica, Dan. 2011. *A Primer for the Mathematics of Financial Engineering.* First Edit. 89–96. New York, NY: FE Press. ISBN: 0-9797576-2-2.

Weerawarana, Rukmal. 2016. *Homework 3 - CFRM 460 (Mathematical Methods for Computational Finance) - University of Washington - rukmal - GitHub.* Accessed February 12, 2019. https://github.com/rukmal/CFRM-460-Homework/blob/master/Homework%203/Homework%203%20Solutions.pdf.