CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

METHODS TO SOLVE ASSET BUBBLE IN FINANCE

A thesis submitted in partial fulfillment of the requirements
For the degree of Master of Science in
Applied Mathematics

by

Jaspreet Kaur

August 2013

The thesis of Jaspreet Kaur is approved:

Dr. Stephen Breen	Date
Dr. Vladislav Panferov	Date
Dr. Jorge Balbás, Chair	Date

California State University, Northridge

Dedication

Jas' dedication

Acknowledgements

I would like to thank my advisor Dr. Jorge Balbas

Table of Contents

ABSTRACT

METHODS TO SOLVE ASSET BUBBLE IN FINANCE

By

Jaspreet Kaur

Master of Science in Applied Mathematics

Financial Market is very attracting topic in finance and mathematics world. Recently we have heard a lot about Gold Prices inflations. It is the the hot topic in today's finance market. So how will be combine mathematics with today's asset changes like gold? How can we determine the tale of asset's volatility for future? These are the questions which we will consider in this thesis. We will study non parametric estimator Floren Zmirou in local real time on compact domain with stochastic differential equation which has unknown drift and diffusion coeificents. Once we will have volatility from floren zmirou then we will able to use RKHS to estimates function which will extrapolate the tale of function.

Chapter 1

Implementation of an Asset Bubble Problem

1.1 Implementation

We used sage python to solve this problem. We orgnize our data and process in six classes. Each class is designed to store the information about Stock, Floren Zmirou AssetBubble, AssetBubbleDetection, Approximation, and Run. First we will work with Stock Class. First We will start with time and stock prices for some stock symbol. First question which will come in mind that How to download and use historic stock prices? So to answer that question, we initialize Stock Class which will explain step by step process how to download, open, read and save historical stock prices.

1.1.1 Stock Class

Methods of Stock Class We obtain stock data by two types of methods. First, we will get it through Google Finance and second, we will get it from CSV file which we download and saved from yahoo finance. There are four functions IsNumber,GetGoogleData, GetStockPrice,and init in Stock Class.

1. `IsNumber()`: Input for this function is rowValue. This function determine if rows of stock data is a numerical string or not.

2. `GetGoogleData()`: Input for this function are Ticker,days and period. This function obtains data for any stock from Google finance.

3. `GetStockPrices()`: Input for this function is csv filename. This function obtains stock prices which are stored in csv file by filename. It will read the yahoo API based minute to minute data.

4. `__init__()`: Input for this function is keyward Usage.This function will analyzie wheather csv filename is used or Ticker parameter for Google finance.

1.1.2 Floren Zmirou Class

Process to solve Floren Zmirou Now we have Stock Prices from Stock class. We will obtain list of sigma values from list of Stock Prices.We will explain this class in following algorithms:

T = 60*n where T is the minute to minute time period [0,T].
Now we will derive hn, x-step_size, and x values which will be used in Floren Zmirou Estimator.

5. `(Derive_hn)`: Input for this function is Stock Prices which we obtained from Stock class. This function will derive hn which will be used in Floren Zmirou Estimator.

Algorithm 1 GetGoogleData

INPUT: Ticker, days, period
OUTPUT: Obtain S (Stock Prices) in either NASD or NYSE.

if the length of the Ticker ≤ 3 **then**
    exchange it with New York Stock Exchange (NYSE).
else
    exchange it with Natinal Association of Securities Dealers (NASD)
end if
We initialize current time in integer.
We will open Google Finance link.
DataList = read each line from Google Finance link.
We initialize tickerData to be the list of array of dataList.
We will put stockPrices in list.
We initialize minuteData.

for minuteData in tickerData **do**
    We split minuteData in commas and put in list.
    We append row one in float and store it in S list.
return S

Algorithm 2 GetStockPrices

INPUT: filename
OUTPUT: Obtain S (Stock Prices) from CSV file.

cr = open and read the csv filename.
We skip the header.
c1 = the empty list.
for row in cr **do**
    if the second row is numberical string **then**
        we add second row to c1.
return c1

Algorithm 3 –init–

INPUT: Keyword Usage
OUTPUT:

if filename is in keyward Usage **then**
    We will obtain Stock Prices from GetStockPrices in list of filename Ticker is used in keyward Usage
    We will obtain Stock Prices from GetGoogleData in a list of Ticker
elseWe will give exception of bad parameters

Algorithm 4 Derive hn

INPUT: Stock Prices S
OUTPUT: hn

1: n = length of Stock Prices S.
2: hn = 1.0/n**(1.0/3.0)
3: return hn =0

6. `(x_step_size)` : Input for this function is Stock Prices which we obtained from Stock class. This function will derive x_step_size which will be used to create step size to generate x.

Algorithm 5 x_hn = x step size

INPUT: Stock Prices S
OUTPUT: $\frac{x}{hn}$

1: We obtained hn from Derive hn function.
2: Double hn = 2 **hn
3: Difference = max S-min S
4: x_hn = Difference * Double hn
5: Return x_hn

7. `(Derive_x_values)` : Input for this function is Stock Prices which we obtained from Stock class. This function will derive x_values which will be used in Floren Zmirou Estimator. Now we have all the ingridents which will be usefull to solve flo-

Algorithm 6 Derive x Values

INPUT: Stock Prices S
OUTPUT: x

1: halfh_n = x_hn /2.0
2: x = empty list.
3: Append x with min S + halfh_n
4: We initilaize ex to be first element of x's list.
5: **while** ex < max S **do**
6: ex = ex+x_hn
7: **end while**
8: We will append ex into x's list
9: Return x

ren zmirou estimator. We will implement floren Zmirou on the following functions. Floren Zmirou has Sublocal time, Local time, volatility Estimator and Indicator function.

8. `(Sublocal_Time)`: Input for this function are T,S,x,n,hn Output for this function will be $L_T^n(x) = (\frac{T}{2nhn}) \sum_{i=1}^{n} 1_{|S\_t(i)-x)|<hn}$

Algorithm 7 Sublocal Time

INPUT: T=Time,S= Stock Prices,x= grid points,n= number of total Stock Prices,hn=Step Size
OUTPUT: $L_T^n(x) = (\frac{T}{2nhn}) \sum_{i=1}^{n} 1_{|S\_t(i)-x)|}$

```
sum = 0.0
scalar = T/(2.0*n*hn)
for i in range of the length of Stock Prices do
    We initialize Sti to be the ith element of the list of Stock prices
    absoluteValue = abs(Sti-x)
     We initilize indicatorValue to pass through indicator function
    if absoluteValue is less than hn then
        sum = sum+indicatorValue return scalar*sum
```

9. `(Local_Time)`: Input for this function are T,S,x,n,hn. Output for this function will be $L_T^n(x) = (\frac{T}{2nhn}) \sum_{i=1}^{n} 1_{|S\_t(i)-x)|<hn} * n(S(t(i+1)) - S(t(i))^2$

Algorithm 8 Local Time

Inputs: T=Time,S= Stock Prices,x= grid points,n= number of total Stock Prices,hn=Step Size
Steps

```
sum = 0.0
scalar = T/(2.0*n*hn)
for for i in range of the length of Stock prices - 1 do
    We initialize Sti to be the ith element of the list of Stock prices
    We initialize Stj to be the jth element of the list of Stock prices
    absoluteValue = abs(Sti-x)
    Difference = (Stj-Sti)**2
     We initilize indicatorValue to pass through indicator function
    if  absoluteValue is less than hn then
        sum = sum+indicatorValue*n*Difference return scalar*sum
```

10. `(Volatility_Estimator Sn(x))`: Volatility Estimator is $\sigma^2(x)$.Input for this function are T,S,x,n,hn. Output for this function will be Local Time/Sublocal Time. We have $\sigma^2(x)$ for Stock Prices. Now we want to see how many stock prices are in each grid point. If there are less than 0 or 1 percent of stock prices in grid then we will exclude that grid point from our calculation process.

11. `(DoGridAnalysis)`:Input for this function are T,S,x,n,hn. Output for this function give us the list of usable grid points and for each usable grid point, the list of Stock prices.

Algorithm 9 Volatility Estimator

Inputs: T=Time,S= Stock Prices,x= grid points,n= number of total Stock Prices,hn=Step Size

Steps

We will initialize ratio to be Local Time / Sublocal Time.
Return ratio. =0

Algorithm 10 DoGridAnalysis

INPUT: T=Time,S= Stock Prices,x= grid points,n= number of total Stock Prices,hn=Step Size,Y = Y percent of total data points.

OUTPUT:

x = useableGridPoints.
d = empty dictionary
Si = length of stock prices.
for grid Points in x **do**
    create a dictionary where the keys are the grid points and the value is a list for each grid point
    for stockPrice in S **do**
        if |grid Points - stockPriceCount|$\leq$ hn **then**
            Add x value to corresponding Si
            for grid Points in x **do**
                if if the list of data points corresponding to x values than Y percent of total grid points **then**
                    add it to the list of usable grid points
                    L = dictionary with key values of grid points.
                    N = Length of L in float.
                    Percent of Stock Prices = N/S
                    if Percent of Stock prices < Y **then**
                        Remove grid points from x
                        Delete L
return x = usable grid points and d

12. `(GetGridVariance`:Input for this function is self.

Algorithm 11 GetGridVariance

INPUT:
OUTPUT: Standard Deviation of usable grid points

```
Points = Empty list
for x in usable grid points do
    y =Sn(x))
    Put (x,y) in Points's list
return Points
```

13. `(GetGridInverseStandardDeviation)` :Input for this function is self.

Algorithm 12 GetGridInverseStandardDeviation

INPUT:
OUTPUT: Standard Deviation of usable grid points

```
Points = Empty list
for x in usable grid points do
    y = 1/sqrt(Sn(x))
    Put (x,y) in Points's list
return Points
```

Natural Cubic Spline- To construct the cubic spline interpolant S for the function f, defined at the numbers $x_0 < x_1 < \ldots < x_n$ satisfying $S''(x_0) = S''(x_n) = 0$:

14. `(Natural Cubic Spline)`

Algorithm 13 Natural Cubic Spline

INPUT: $n, x_0, x_1, \ldots, x_n; a_0 = f(x_0), a_1 = f(x1), \ldots, a_n = f(x_n)$
OUTPUT: a_j, b_j, c_j, d_j for $j = 0, 1, \ldots, n-1$.
(note: $S_(x) = S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$ for $x_j \leq x \leq x_{j+1}$)

for $i = 0, 1, \ldots, n-1$ **do**
    $h_i = x_{i+1} - x_i$
end for
for $i = 1, 2, \ldots, n-1$ **do**:
    $\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$
end for
Set $l_0 = 1$;
$\mu = 0$;
$z_0 = 0$.
for $i = 1, 2, \ldots, n-1$ **do**
    $l_1 = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$;
$\mu_1 = \frac{h_1}{l_1}$;
$z_1 = \frac{(\alpha_1 - h_{i-1}z_{i-1})}{l_1}$.
Set $l_n = 1$;
$z_n = 0$;
$c_n = 0$.
    for $j = n-1, n-2, \ldots, 0$ **do**
        $c_j = z_j - \mu_j c_{j+1}$;
        $b_j = (a_{j+1} - a_j)/h_j - h_j(c_{j+1} + 2c_j)/3$;
        $d_j = (c_{j+1} - c_j)/(3h_j)$
Return (a$_j, b_j, c_j, d_j for j = 0, 1, \ldots, n-1$);