Objected Oriented Programming—Final Project

**Convertible Bond Calculator**

**Ke Gao**

Title of system:

Convertible Bond Calculator

Why you choose this application:

We need solution for convertible bond value calculation and plot.

How is the system impact the user:

It will generate key ratio and plot of convertible bond.

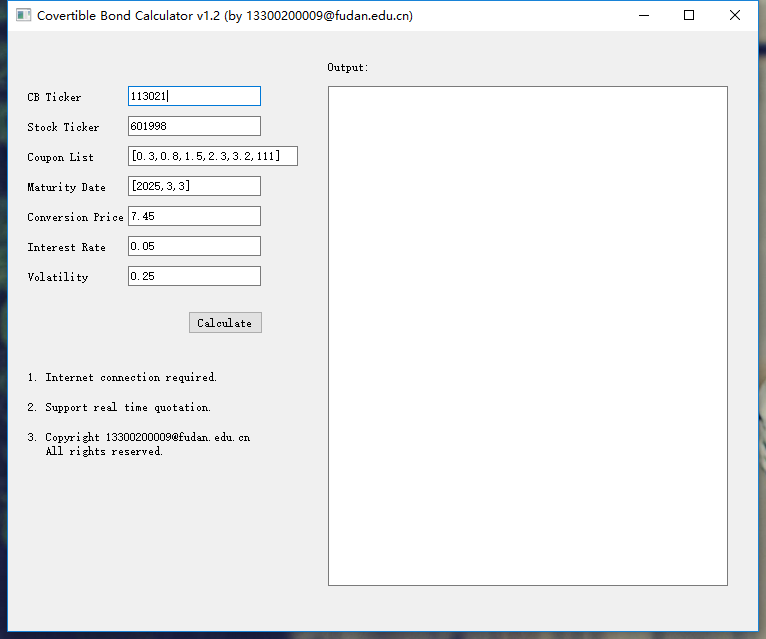
What is the merits of this system:

It has intuitive GUI, and support real time quotation update

Who will be benefited：

Traders and analysts of convertible bond.

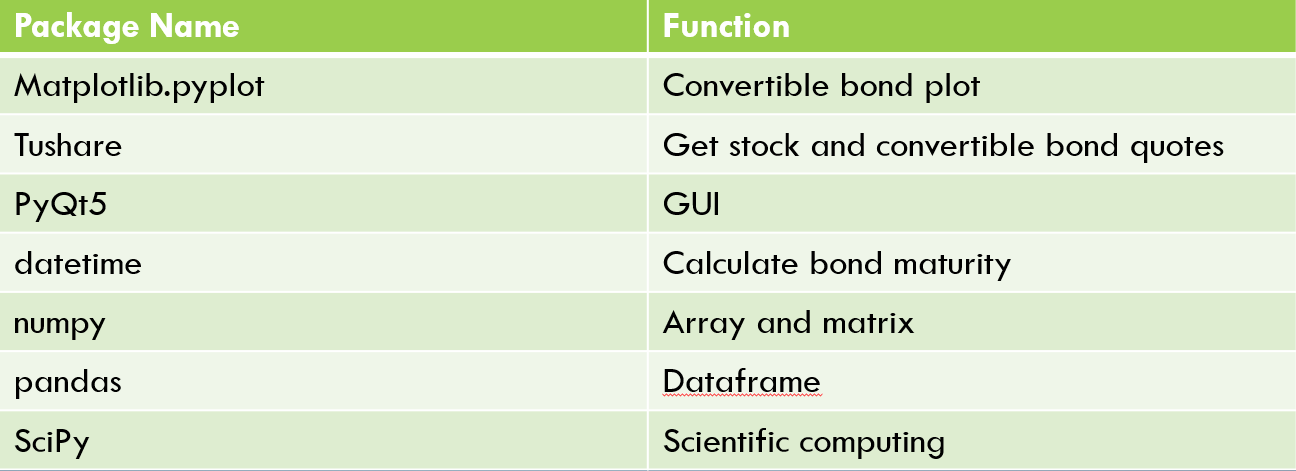
User interface：（Example: China CITIC bank convertible bond）



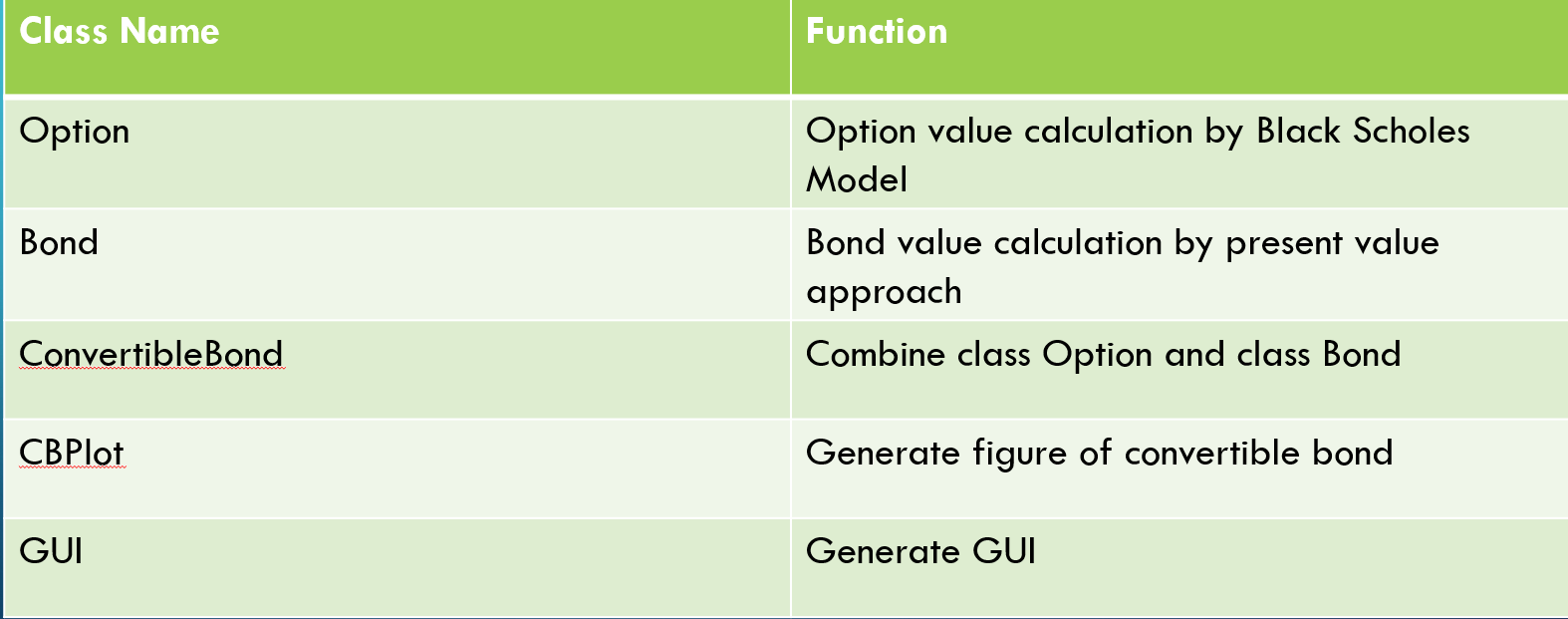
|  |  |  |
| --- | --- | --- |
| **Input** | **Type** | **Example:** |
| CB Ticker | str | 113021 |
| Stock Ticker | str | 601998 |
| Coupon list | list | [0.3,0.8,1.5,2.3,3.2,111] |
| Maturity Date | list | [2025,3,3] |
| Conversion Price | float | 7.45 |
| Interest rate | float | 0.05 |
| Volatility | float | 0.25 |

Programming design:

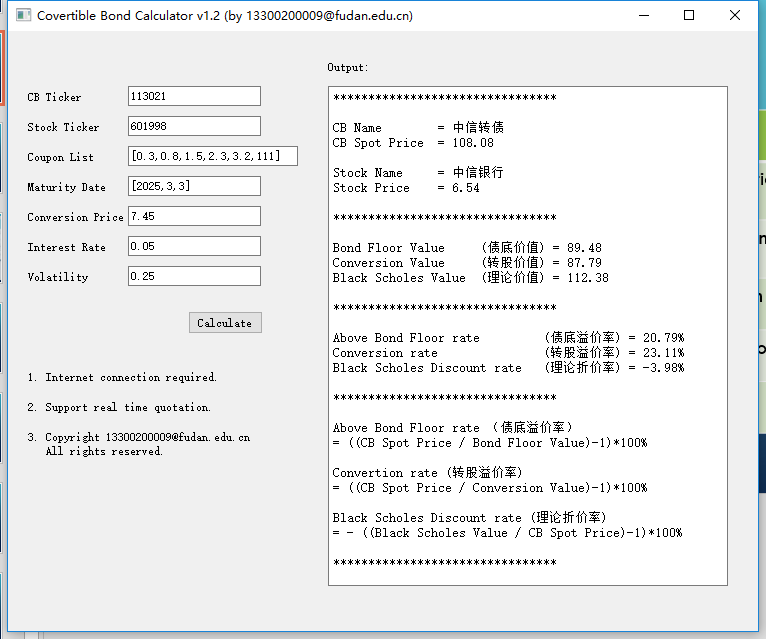
We use these packages:

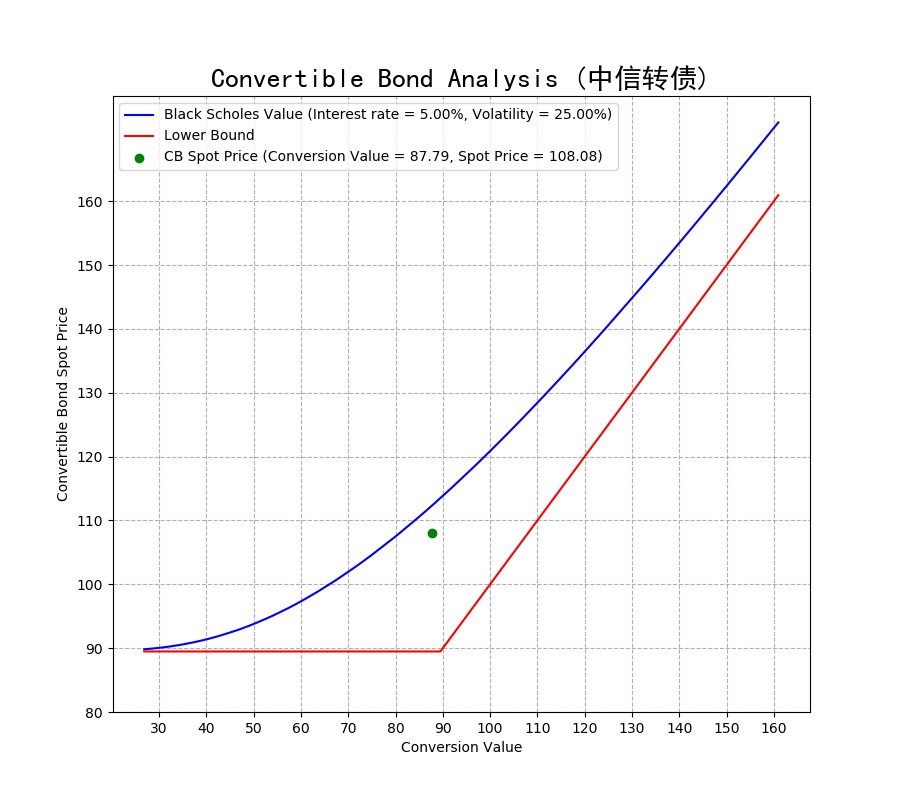


We use these classes:

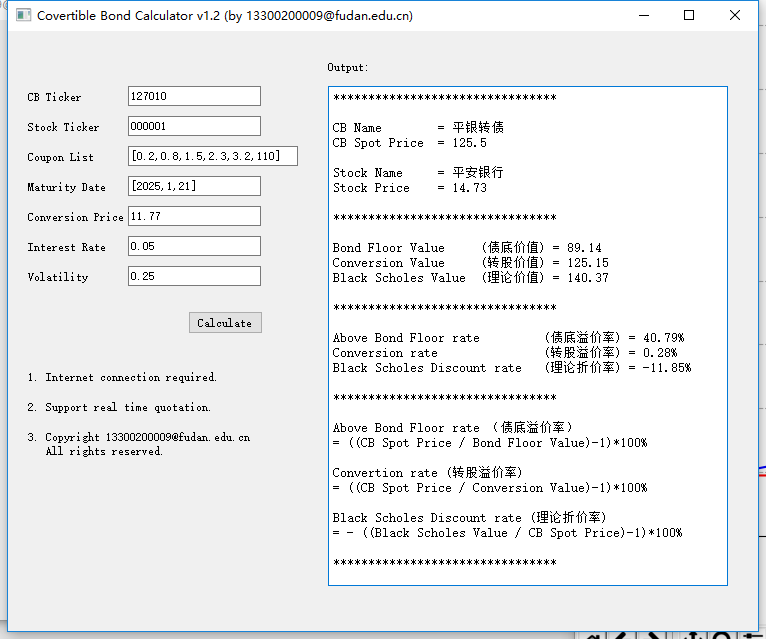


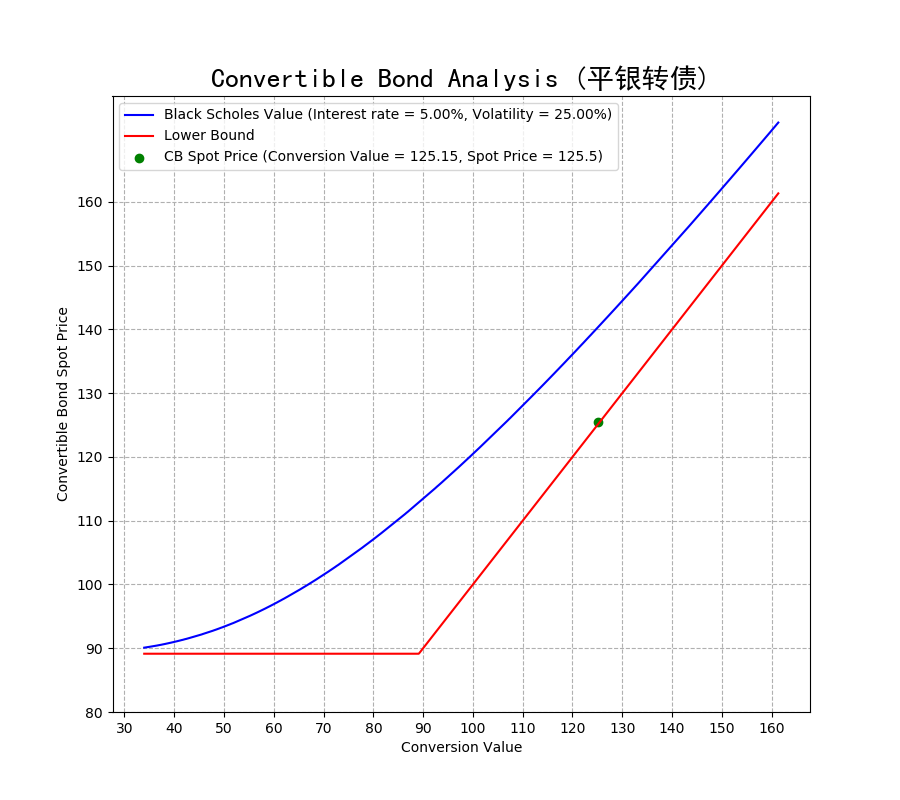
Demo: （Example 1: China CITIC bank convertible bond）





（Example 2: China Ping An bank convertible bond）





Source code:

import ast

import datetime

import sys

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import scipy.stats as sps

from PyQt5.QtCore import QCoreApplication

from PyQt5.QtWidgets import (QApplication, QLabel, QLineEdit, QMessageBox,

QPushButton, QTextBrowser, QWidget)

import tushare as ts

def getQuote(ticker):

return float(ts.get\_realtime\_quotes(str(ticker))['price'][0])

def getName(ticker):

return ts.get\_realtime\_quotes(str(ticker))['name'][0]

def Outputdf(df,name='output'):

df.to\_csv(str(name)+'.csv')

def percent(float):

return str('%.2f%%' % (float\*100))

class Option:

def \_\_init\_\_(self, type, s0, k, t, r, sigma,dv=0.0):

if type=='c':

self.type='Call'

self.cp\_sign=1.0

elif type=='p':

self.type='Put'

self.cp\_sign=-1.0

self.s0=s0

self.k=k

self.t=t

self.r=r

self.sigma=sigma

self.dv=dv

self.d\_1 = (np.log(self.s0 / self.k) + (self.r - self.dv + .5 \* self.sigma \*\* 2) \* self.t) / self.sigma / np.sqrt(self.t)

self.d\_2 = self.d\_1 - self.sigma \* np.sqrt(self.t)

def price(self):

return self.cp\_sign \* self.s0 \* np.exp(-self.dv \* self.t) \* sps.norm.cdf(self.cp\_sign \* self.d\_1) \

- self.cp\_sign \* self.k \* np.exp(-self.r \* self.t) \* sps.norm.cdf(self.cp\_sign \* self.d\_2)

def \_\_repr\_\_(self):

string="\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

string+="Option type: "+str(self.type)+'\n'

string+="Stock price: "+str(self.s0)+'\n'

string+="Strike price: "+str(self.k)+'\n'

string+="Maturity: "+str(self.t)+'\n'

string+="Interest rate: "+str(self.r)+'\n'

string+="Volatility: "+str(self.sigma)+'\n'

string+="Dividend: "+str(self.dv)+'\n'

string+="\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

string+=self.type+' price = '+str(self.price())+'\n'

string+="\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

return string

class Bond:

def \_\_init\_\_(self, coupon\_list,t):

self.coupon\_list=coupon\_list

self.t=t

def pv(self, r):

length=len(self.coupon\_list)

pv=0.0

for i, coupon in enumerate(self.coupon\_list):

if i>=length-round(self.t)-1:

pv+=coupon\*np.exp(-r\*(self.t-(length-i-1)))

return pv

def fv(self, r):

fv=self.pv(r)\*np.exp(r\*self.t)

return fv

class ConvertibleBond:

def \_\_init\_\_(self,coupon\_list,t, convert\_price, s0, r, sigma,dv=0.0, face\_value=100, redemption\_ratio=1.3):

self.s0=s0

self.redemption\_ratio=redemption\_ratio

self.bond=Bond(coupon\_list, t)

self.t=t

self.sigma=sigma

self.k=convert\_price\*coupon\_list[-1]/face\_value

self.r=r

self.face\_value=face\_value

self.multiplier=face\_value/convert\_price

self.option=Option("c", s0, self.k, t, r, sigma)

def BondValue(self):

return round(self.bond.pv(self.r),2)

def OptionValue(self):

return self.multiplier\*self.option.price()

def TotalValue(self):

return round(self.BondValue()+self.OptionValue(),2)

def ConvertValue(self,stock\_price):

return round(self.multiplier\*stock\_price,2)

def TotalValue\_ConvertValue\_Ratio(self,stock\_price):

return (self.TotalValue()/self.ConvertValue(stock\_price))

def \_\_repr\_\_(self):

string="\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

string+="Maturity: "+str(self.t)+'\n'

string+="Interest rate: "+str(self.r)+'\n'

string+="Volatility: "+str(self.sigma)+'\n'

string+="\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

string+='Bond Value= '+str(self.BondValue())+'\n'

string+='Option Value= '+str(self.OptionValue())+'\n'

string+='Total Value= '+str(self.TotalValue())+'\n'

string+="\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

return string

class CBPlot:

def \_\_init\_\_(self,cb\_ticker,stock\_ticker, coupon\_list, maturity\_date, convert\_price, interest\_rate, volatility):

self.cb\_ticker=cb\_ticker

self.stock\_ticker=stock\_ticker

self.cb\_name=getName(cb\_ticker)

self.stock\_name=getName(stock\_ticker)

self.cb\_spot\_price=getQuote(cb\_ticker)

self.stock\_price=getQuote(stock\_ticker)

self.coupon\_list=coupon\_list

self.maturity=((datetime.date(maturity\_date[0],maturity\_date[1],maturity\_date[2])-datetime.date.today()).days)/365

self.convert\_price=convert\_price

self.interest\_rate=interest\_rate

self.volatility=volatility

self.cb=ConvertibleBond(self.coupon\_list,self.maturity,self.convert\_price,self.stock\_price,self.interest\_rate,self.volatility)

def generateStockPriceSeries(self,lower=0.3,upper=2.0):

simulated\_stock\_price=[]

for i in range(100\*round(self.convert\_price\*lower),100\*round(self.convert\_price\*upper)):

price=i/100

simulated\_stock\_price.append(price)

return simulated\_stock\_price

def plot\_cb(self):

simulated\_price\_series=self.generateStockPriceSeries(upper=max(1.6,self.cb.ConvertValue(self.stock\_price)/self.cb.face\_value))

x\_convert\_value=[]

y\_cb\_total\_value=[]

lower\_bound=[]

for simulated\_price in simulated\_price\_series:

x\_convert\_value.append(self.cb.ConvertValue(simulated\_price))

y\_cb\_total\_value.append(ConvertibleBond(self.coupon\_list,self.maturity,self.convert\_price,simulated\_price,self.interest\_rate,self.volatility).TotalValue())

lower\_bound.append(max(ConvertibleBond(self.coupon\_list,self.maturity,self.convert\_price,simulated\_price,self.interest\_rate,self.volatility).BondValue(),self.cb.ConvertValue(simulated\_price)))

plt.figure(figsize=(9, 8))

curve\_1,=plt.plot(x\_convert\_value,y\_cb\_total\_value,color='blue')

curve\_2,=plt.plot(x\_convert\_value,lower\_bound, color='red')

point\_1 = plt.scatter(self.cb.ConvertValue(self.stock\_price), self.cb\_spot\_price,c='green')

plt.grid(ls='--')

plt.xticks(np.arange(30,max(170,self.cb.ConvertValue(self.stock\_price)),10))

plt.yticks(np.arange(80,max(170,max(170,self.cb\_spot\_price)),10))

plt.title('Convertible Bond Analysis ('+str(self.cb\_name)+')',fontproperties='SimHei',fontsize=20)

plt.xlabel('Conversion Value')

plt.ylabel('Convertible Bond Spot Price')

plt.legend([curve\_1,curve\_2,point\_1],['Black Scholes Value (Interest rate = '+percent(self.interest\_rate)+", Volatility = "+percent(self.volatility)+")",' Lower Bound',"CB Spot Price (Conversion Value = "+str(self.cb.ConvertValue(self.stock\_price))+',Spot Price = '+str(self.cb\_spot\_price)+')'],loc='upper left')

plt.show()

def df\_cb(self):

simulated\_price\_series=self.generateStockPriceSeries(upper=1.6)

x\_convert\_value=[]

y\_cb\_total\_value=[]

lower\_bound=[]

for simulated\_price in simulated\_price\_series:

x\_convert\_value.append(self.cb.ConvertValue(simulated\_price))

y\_cb\_total\_value.append(ConvertibleBond(self.coupon\_list,self.maturity,self.convert\_price,simulated\_price,self.interest\_rate,self.volatility).TotalValue())

lower\_bound.append(max(ConvertibleBond(self.coupon\_list,self.maturity,self.convert\_price,simulated\_price,self.interest\_rate,self.volatility).BondValue(),self.cb.ConvertValue(simulated\_price)))

df=pd.DataFrame(index=simulated\_price\_series)

df['Lower Bound']=lower\_bound

df['Black Scholes Value']=y\_cb\_total\_value

return df

def \_\_repr\_\_(self):

self.cb\_spot\_price=getQuote(self.cb\_ticker)

self.stock\_price=getQuote(self.stock\_ticker)

self.cb=ConvertibleBond(self.coupon\_list,self.maturity,self.convert\_price,self.stock\_price,self.interest\_rate,self.volatility)

string="\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"

string+='CB Name = '+self.cb\_name+'\n'

string+='CB Spot Price = '+str(self.cb\_spot\_price)+'\n\n'

string+='Stock Name = '+self.stock\_name+'\n'

string+='Stock Price = '+str(self.stock\_price)+'\n'

string+="\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"

string+='Bond Floor Value (债底价值) = '+str(self.cb.BondValue())+'\n'

string+='Conversion Value (转股价值) = '+str(self.cb.ConvertValue(self.stock\_price))+'\n'

string+='Black Scholes Value (理论价值) = '+str(self.cb.TotalValue())+'\n'

string+="\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"

string+="Above Bond Floor rate (债底溢价率) = "+percent(self.cb\_spot\_price/self.cb.BondValue()-1)+'\n'

string+="Conversion rate (转股溢价率) = "+percent(self.cb\_spot\_price/self.cb.ConvertValue(self.stock\_price)-1)+'\n'

string+="Black Scholes Discount rate (理论折价率) = "+percent(-(self.cb.TotalValue()/self.cb\_spot\_price-1))+'\n'

string+="\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n"

string+="Above Bond Floor rate （债底溢价率）\n= ((CB Spot Price / Bond Floor Value)-1)\*100%\n"

string+="\nConvertion rate (转股溢价率) \n= ((CB Spot Price / Conversion Value)-1)\*100%\n\n"

string+="Black Scholes Discount rate (理论折价率) \n= - ((Black Scholes Value / CB Spot Price)-1)\*100%\n"

string+="\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

return string

class GUI(QWidget):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.initialize\_UI()

def initialize\_UI(self):

self.setGeometry(300, 300, 750, 600)

self.setWindowTitle("Covertible Bond Calculator v1.2 (by 13300200009@fudan.edu.cn)")

self.label1=QLabel("CB Ticker",self)

self.label1.move(20,60)

self.text1=QLineEdit("113021",self)

self.text1.move(120,55)

self.label2=QLabel("Stock Ticker",self)

self.label2.move(20,90)

self.text2=QLineEdit("601998",self)

self.text2.move(120,85)

self.label3=QLabel("Coupon List",self)

self.label3.move(20,120)

self.text3=QLineEdit("[0.3,0.8,1.5,2.3,3.2,111]",self)

self.text3.move(120,115)

self.text3.setFixedWidth(170)

self.label4=QLabel("Maturity Date",self)

self.label4.move(20,150)

self.text4=QLineEdit("[2025,3,3]",self)

self.text4.move(120,145)

self.label5=QLabel("Conversion Price",self)

self.label5.move(20,180)

self.text5=QLineEdit("7.45",self)

self.text5.move(120,175)

self.label6=QLabel("Interest Rate",self)

self.label6.move(20,210)

self.text6=QLineEdit("0.05",self)

self.text6.move(120,205)

self.label7=QLabel("Volatility",self)

self.label7.move(20,240)

self.text7=QLineEdit("0.25",self)

self.text7.move(120,235)

self.label8=QLabel("1. Internet connection required.",self)

self.label8.move(20,340)

self.label9=QLabel("2. Support real time quotation.",self)

self.label9.move(20,370)

self.label9=QLabel("3. Copyright 13300200009@fudan.edu.cn \n All rights reserved.",self)

self.label9.move(20,400)

self.label\_output=QLabel('Output:',self)

self.label\_output.move(320,30)

self.textbrowser=QTextBrowser(self)

self.textbrowser.move(320,55)

self.textbrowser.setFontPointSize(10)

self.textbrowser.resize(400,500)

self.button1=QPushButton('Calculate',self)

self.button1.move(180,280)

self.button1.clicked.connect(self.Calculate)

self.show()

def Calculate(self):

self.cb\_ticker=str(self.text1.text())

self.stock\_ticker=str(self.text2.text())

self.coupon\_list=ast.literal\_eval(self.text3.text())

self.maturity\_date=ast.literal\_eval(self.text4.text())

self.convert\_price=float(self.text5.text())

self.interest\_rate=float(self.text6.text())

self.volatility=float(self.text7.text())

self.conv\_bond=CBPlot(self.cb\_ticker,self.stock\_ticker,self.coupon\_list,self.maturity\_date,self.convert\_price,self.interest\_rate,self.volatility)

self.textbrowser.setText(str(self.conv\_bond))

self.conv\_bond.plot\_cb()

app=QApplication(sys.argv)

a=GUI()

sys.exit(app.exec\_())