VPINによる市場の考察

VPIN(Volume-Synchronized Probability of Informed Trading)

構成

1. 研究目的
2. PIN
3. VPIN
4. シミュレーション・考察
5. 今後の課題
6. 参考文献
7. プログラム

法政大学大学院

工学研究科システム工学専攻

藤原佑太

西村吉政

1.1 研究目的

フラッシュクラッシュのような大幅な株価変動によるリスクの回避。

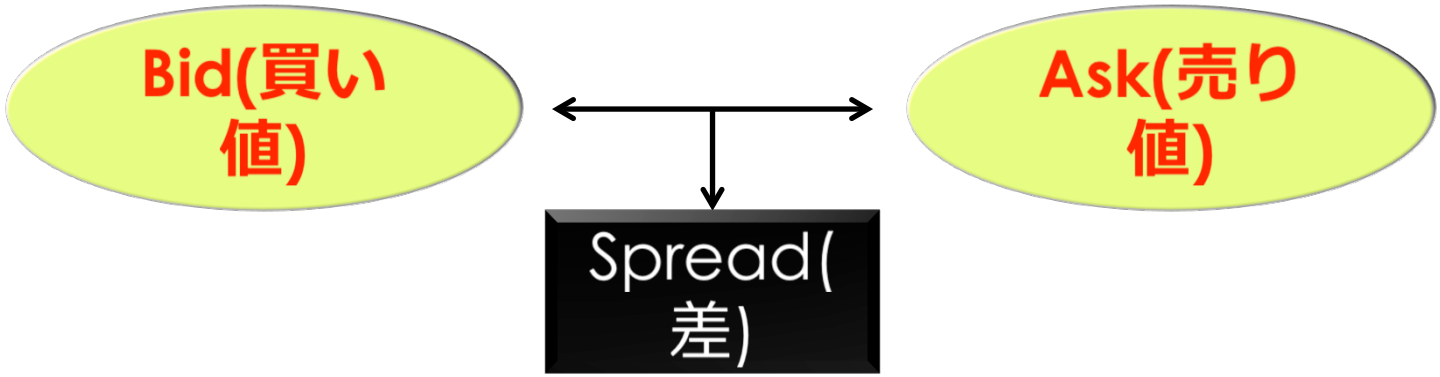
- 2010年5月6日にわずか数分間で株価が大暴落。
- 高速取引を行う市場参加者達が一時的に取引を停止した。

売り手と買い手の数の不均衡が生じたため株価は暴落を続けた。

そこで我々はVPIN (Volume-Synchronized Probability of Informed Trading) を用いてフラッシュクラッシュのようなリスクの回避の方法を考察。

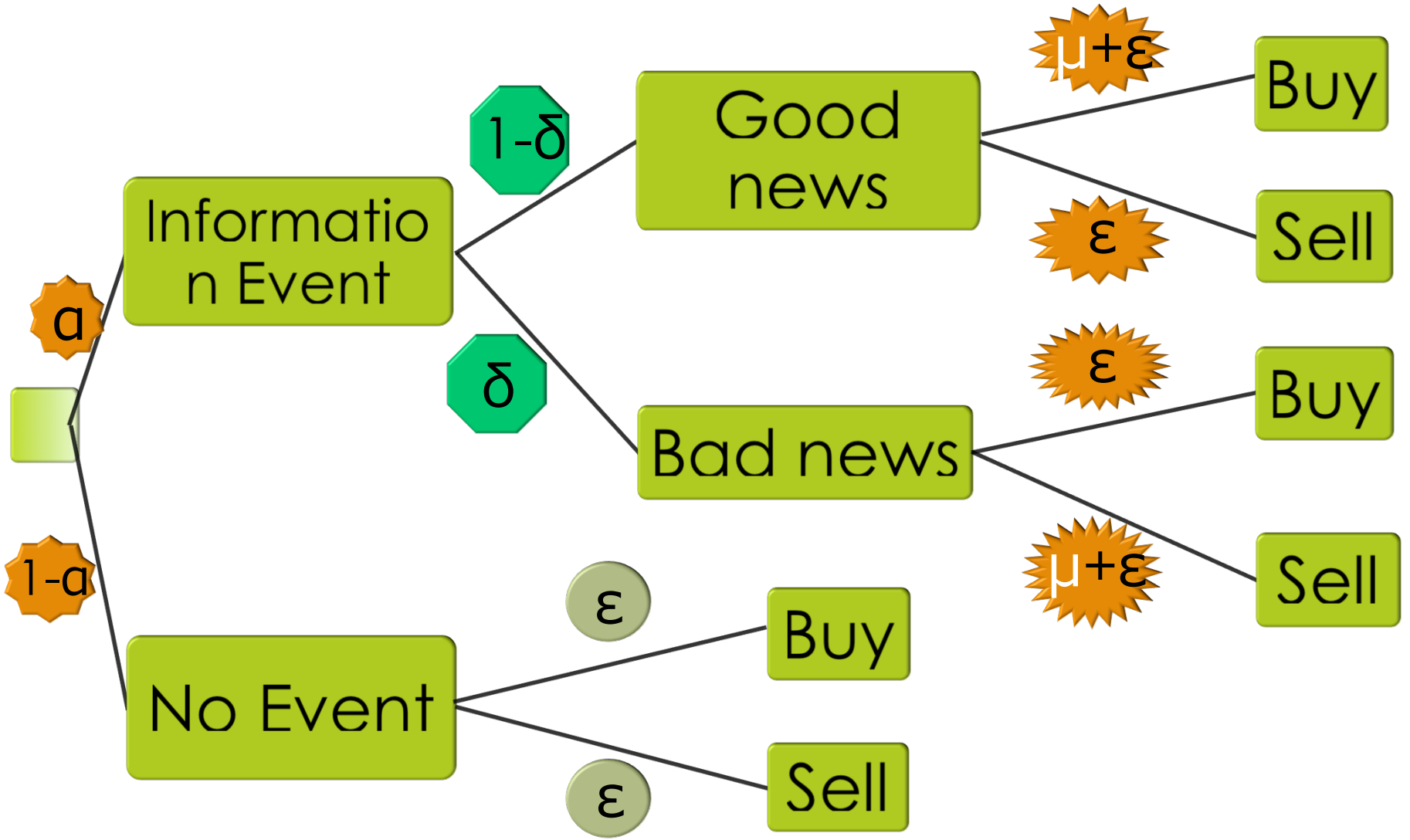
2.1 PIN(probability of information)

PINとは、プライベートインフォメーションを注文数の比率から測定する方法である。



情報の非対称性の度合いを考察することができる。

2.2 情報と売買の関係


α
1-α
Information Event
1-δ
δ
Good news
Bad news
No Event
μ+ε
ε
ε
μ+ε
ε
ε
Buy
Sell
Buy
Sell
Buy
Sell

2.3 PIN(2)

$$PIN = \frac{\alpha\mu}{\alpha\mu + 2\varepsilon}$$

情報を持つトレーダーの注文

全ての注文の合計

PINは全注文に含まれる情報の量を表している

2.4 PIN(3)

しかし、PINは……

- 注文数が莫大でパラメータ計算に時間がかかる。
- リアルタイムでの細かい注文を把握できない。

そこで……

素早く計算ができ、出来高に焦点を置いたVPINを活用。

3.1 VPIN

VPIN (Volume-Synchronized Probability of Informed Trading)とはトレーダーの情報や流動性を膨大な注文数からではなく、出来高から算出する方法。

Easley, Prado, and O'Hara [2012]によると、VPINとPINの誤差は無視できる程の差異である。

VPIN はPIN と同様に情報の非対称性を表現しており、パラメータの推定を必要とせず、処理するデータがPINよりも少ない。つまり、高速取引におけるシミュレーションに適している。

3.2 出来高の分割(1)

出来高を集計し、平等な量の出来高に分割する。分割した出来高をバケットと呼びサイズをVとする。

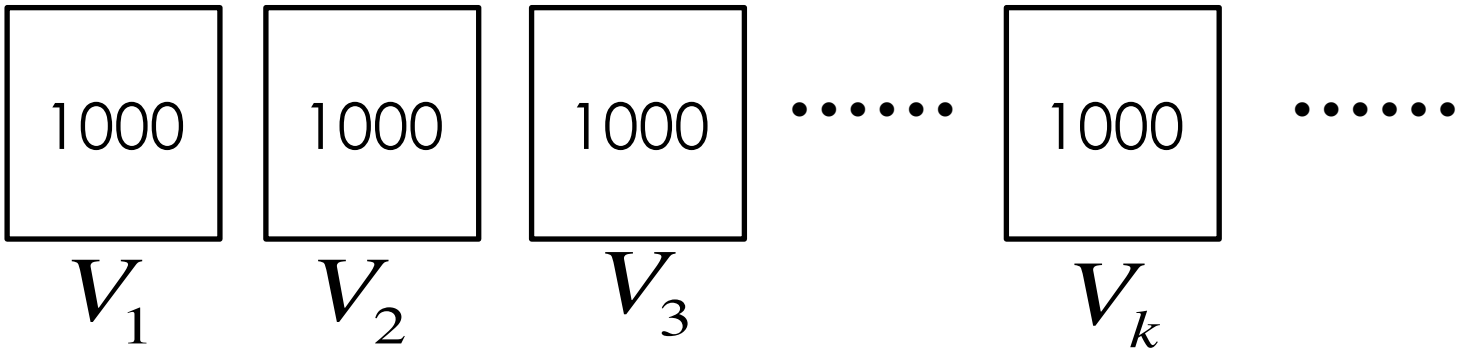
$$V = V_1 = V_2 = V_3 = \cdots\cdots = V_k$$

- バケットのサイズは任意で決定する。

3.3 出来高の分割(2)

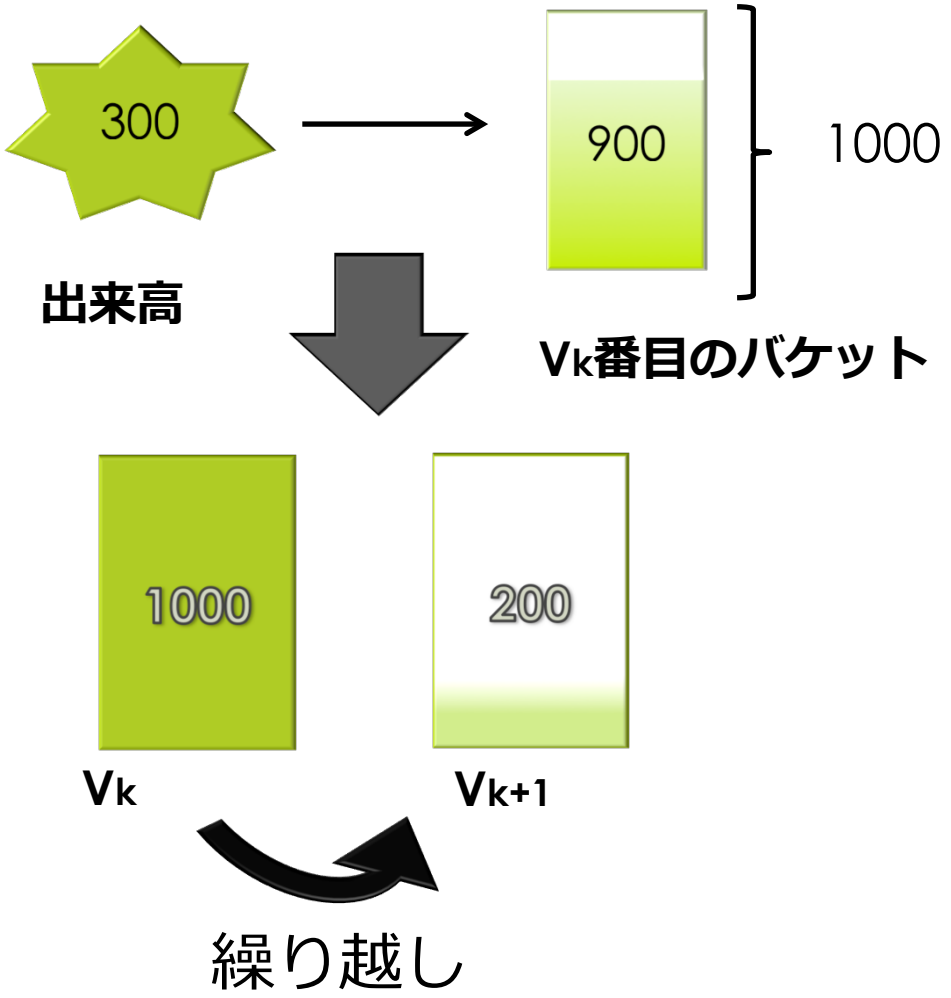
累計出来高数 $\sum V$

大きな価格変動が起きる
可能性がある取引の集合
に分割することができる。



3.4 出来高の分割

バケットサイズを1000とすると…



- Vk番目のバケットに900の出来高が詰まっているとする。
- そこにバケットの容量(1000)を超えてしまう出来高(300)が来る
- 300のうち、100を入れVk番目のバケットを満たす。
- 残りの200は次のVk+1番目のバケットに繰り越される

3.5 出来高の分類(1)

- 取引の性質によって出来高の分類を行う。
- バケット内の出来高を買い手主導の出来高と売り手主導の出来高に分類する。

価格が上昇($\Delta P>0$) → 買い手主導の出来高に

価格が下落($\Delta P<0$) → 売り手主導の出来高に

価格の変化なし($\Delta P=0$) → 計算から除外

3.6 出来高の分類

k 番目のバケットの最初から最後までの合計

標準正規分布の累積分布関数

バケット内の出来高の価格差

価格差の標準偏差

$$V_k^B = \sum_{i=t(k-1)+1}^{t(k)} \left(V \times Z\left(\frac{\Delta P_i}{\sigma_{\Delta P}}\right)\right)$$

$$V_k^S = \sum_{i=t(k-1)+1}^{t(k)} \left(V \times \left[1 - Z\left(\frac{\Delta P_i}{\sigma_{\Delta P}}\right)\right]\right)$$

$$V = V_k^S + V_k^B$$

バケットのサイズ(V)は一定なので、どちらかを求めれば良い。

3.7 出来高の不均衡

$$\text{不均衡値} = |V_k^S - V_k^B|$$

- 不均衡値が大きければ大きいほど、取引の偏りが強い。
- この不均衡値はクラッシュが発生した場合、どちらかに偏った値が出る。

3.8 VPINモデル

$$VPIN = \frac{\sum_{k=i}^{i+n-1} |V_k^S - V_k^B|}{nV}$$

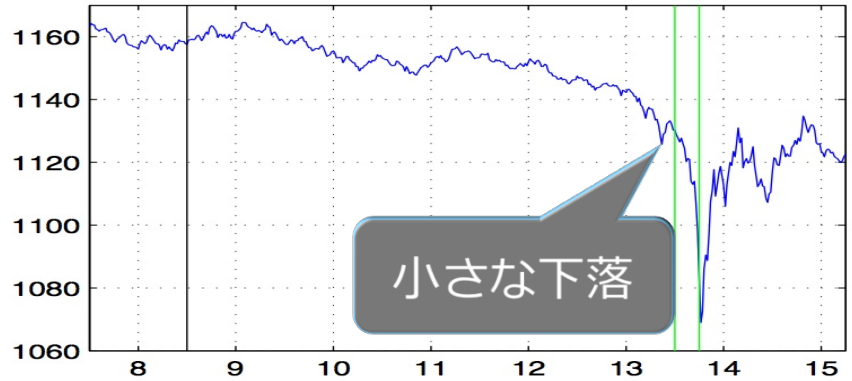
VPINを計算するのに使ったバケット数

累計出来高数

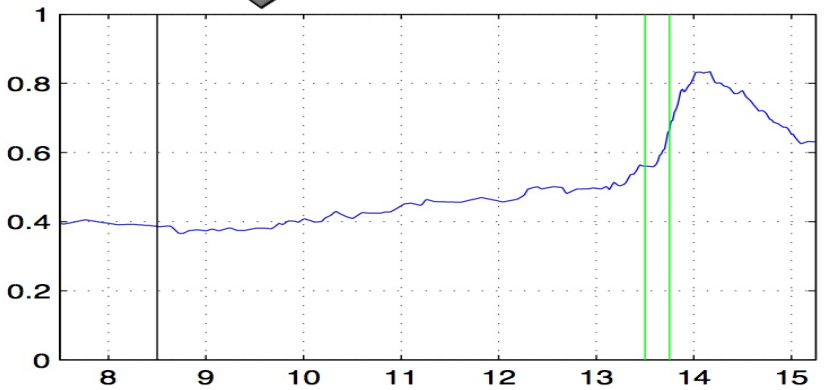
VPINはn個のバケットを用いて算出する

4.1 シミュレーション

E-mini S&P 500のフラッシュクラッシュ時のPrice



E-mini S&P 500のフラッシュクラッシュ時のVPIN



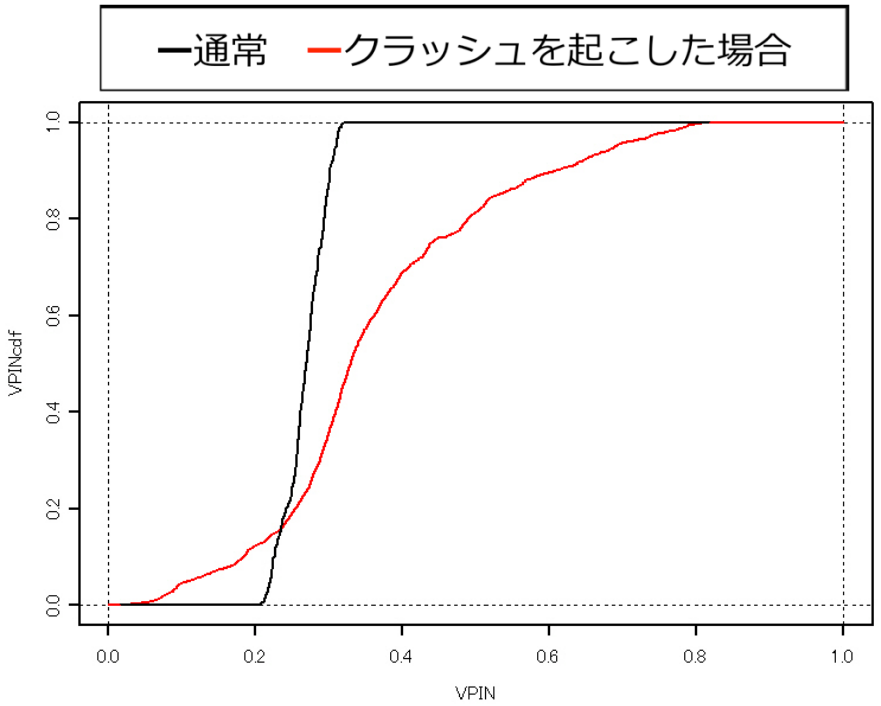
出典:「VPIN and the Flash Crash」Torben G.Andersen,Oleg Bondarenko October 2011

これはフラッシュクラッシュ発生時のＶＰＩＮのグラフである。
価格が下落した時にＶＰＩＮの値が上昇しているのが分かる。
今回は、安定した市場のティックデータにクラッシュを発生させ、次の２通りのシミュレーションを行った。データは2011/01/11のE-mini S&P500を使用。

①何の前触れもなくクラッシュが発生した場合。
②上図の様に、クラッシュが発生する前に小さな価格変動がある場合。

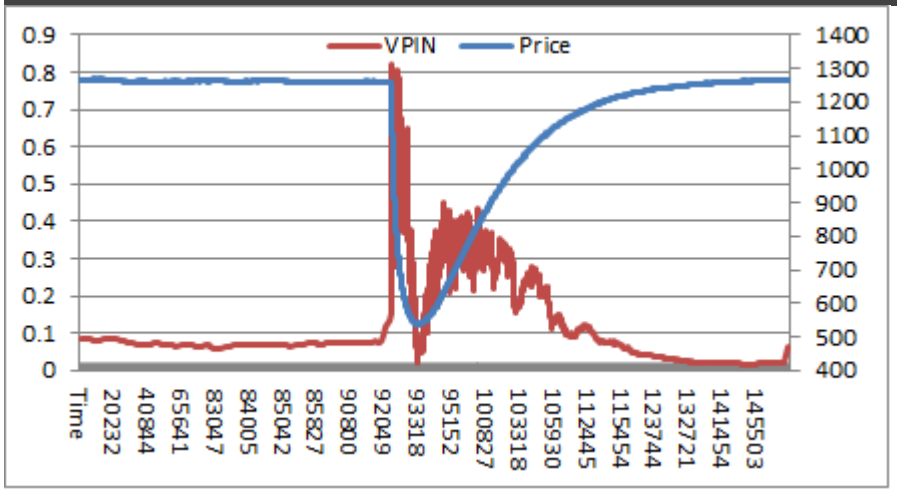
4.2 VPINの累積確率密度(CDF)

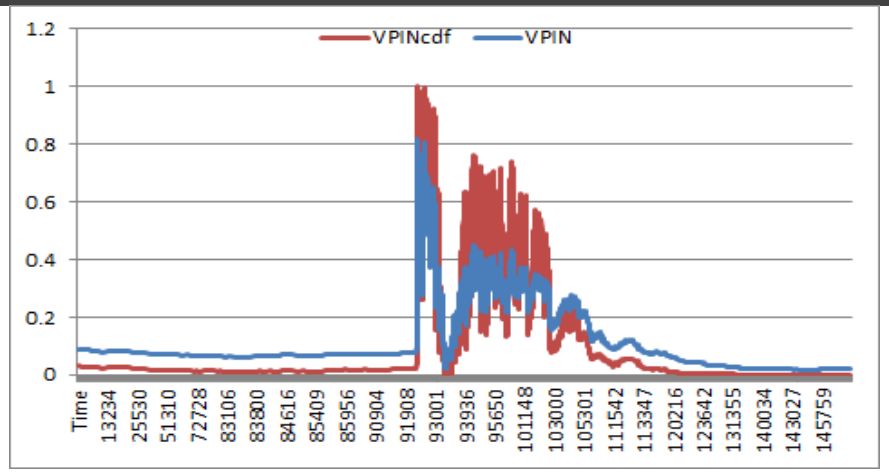
- 取得したVPINが過去のVPINと比較する時の指標となる、CDFを作成した。
- CDFの値が 1 に近い値だと、過去に収集したVPINの中で高い値だということが分かる。



- クラッシュが起こらない日に注目すると、CDFの値が極端な値が出やすいということが分かる。
- 通常の日でもバケットの容量を増やすとVPINの値にばらつきが見られたが、バケットの数が増え、計算が遅くなってしまう。
- クラッシュを起こした過去のデータも含めて蓄積していくと、VPINの値の信頼性が増すと考えられる。

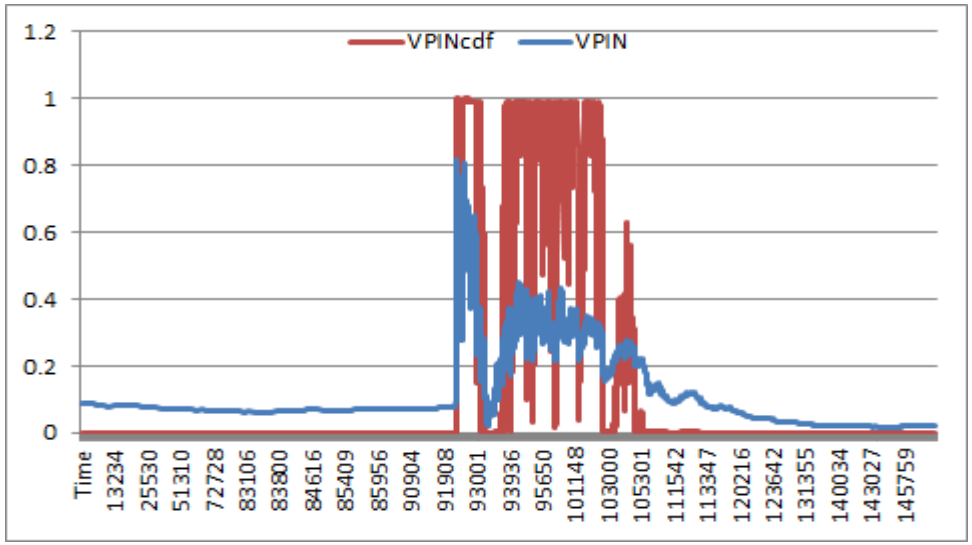
4.3 シミュレーション①





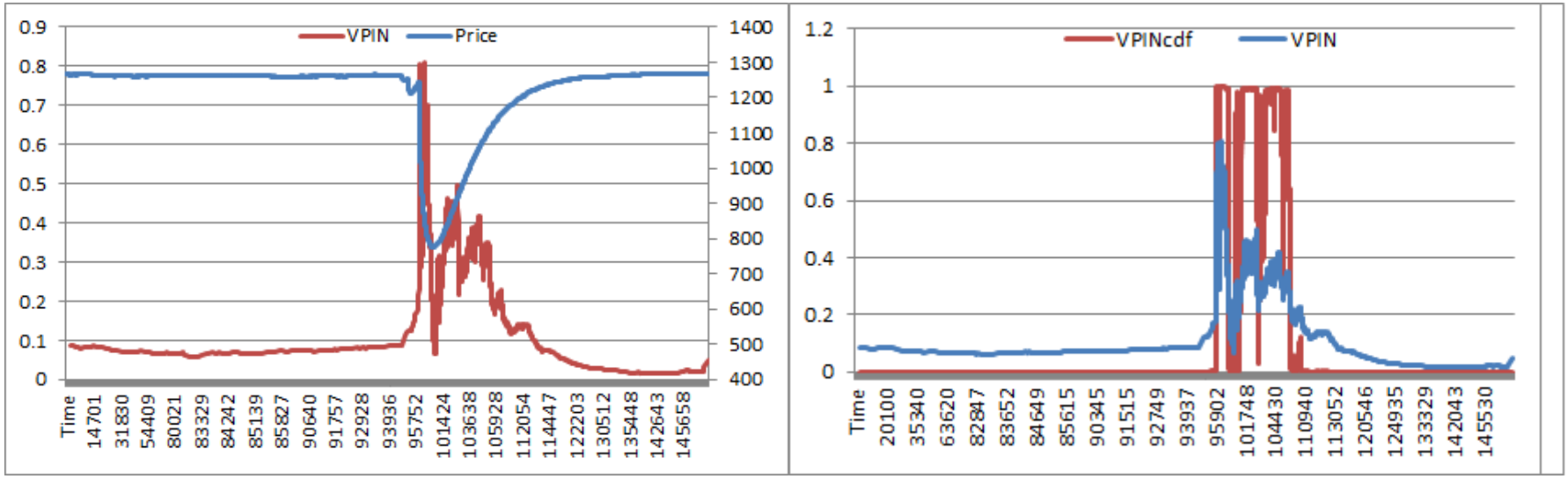
何の前触れもなく突然クラッシュさせ、VPINとCDFの反応を観察した。
左の図では、VPINの値は価格の下落が始まってから反応している。
右の図では、VPINとCDFをグラフに表わしたものである。
このシミュレーションが第一回目なので、右のグラフではVPINとCDFのグラフが似た形になっている。

4.4 シミュレーション①



では、次にCDFをクラッシュの起こっていない通常のデータを使い、CDFを観察。
今回は、通常では観測しなかったVPINの値が出たため、CDFは大きな値を示している。

4.5 シミュレーション②



次に、クラッシュを起こす前に小さな変動を発生させた。
また、クラッシュの価格変動の関数を変更し、シミュレーションを行う。
今回のCDFは通常のデータを５００日間と、突然クラッシュが発生したときのデータ１日を使用した。
通常のデータを多く使用することによってCDFの値を安定させる事が出来たが、小さな変動に反応しなかった。

5.1 今後の課題

- クラッシュを発生させたシミュレーションではVPINの累積分布のデータを集めることができたが、同じバケットの容量で通常のデータを処理するとVPINの累積分布データが極端な値になってしまった。今後は適正なバケットの値を推定することが課題である。
- クラッシュが発生したデータをリアルタイムで蓄積し、CDFの信頼性を上げる必要がある。今回はサンプルデータが１日分しか入手出来なかった。実際にクラッシュが発生したデータを用いて、検証を行いたい。
- CDFのデータベースにどのような株価の組み合わせを入れるのか。

6.1 参考文献

- 「VPIN and the Flash Crash」 Torben G.Andersen,Oleg Bondarenko [2011]
- 「Flow Toxicity and Liquidity in a High Frequency World」 David Easley,Marcos M.Lopez de prado,Maureen O'Hara[2012]

7.1 プログラム（１）

1

```
#データの読み込み
t<-scan("ファイルの場所")#時間
volume<-scan("ファイルの場所")#出来高
price<-scan("ファイルの場所")#価格

#先に注文数0を除く
P=V=Time=NULL
i<-0
while(i<length(volume)){
        i<-i+1
        if(volume[i]>0){
                V<-append(V,volume[i])
                P<-append(P,price[i])
                Time<-append(Time,t[i])
        }
}
##クラッシュ発生のシミュレーション
ctime<-100000  ##時間
cnumber<-which(abs(Ti-ctime)==min(abs(Ti-ctime))) ##場所
cnumber1<-cnumber[1]
width<-length(Time)-cnumber1 ##クラッシュさせる幅
x<-seq(0,15,length=width)
C<-dchisq(x,df=3)
revC<-rev(C)
volume2<-replace(V,cnumber1:
(cnumber1+width-1),V[cnumber1:(cnumber1+width-1)]
+(10000*C))
price2<-replace(P,cnumber1:
(cnumber1+width-1),P[cnumber1:(cnumber1+width-1)]-
(3000*C))
#前日を追加（データ取り）
price2=volume2=NULL
volume2<-append(volume2,V,after=0)
price2<-append(price2,P,after=0)
time<-append(Time,Time,after=0)
tt<-Time
```

2

```
 #値段が変わる１つ前の注文に注目
deltaP1<-diff(price2,lag=1)
deltaP1<-deltaP1[!is.na(deltaP1)]
number<-NULL
i<-0
while(i<=length(deltaP1)-1){
        if(i==length(deltaP1)){break}
        i<-i+1
        #今回は無理やりクラッシュを発生させた。
        #本来、価格は0.25刻みで変化するので、小さい変動は無視
        if(deltaP1[i] !=0 && abs(deltaP1[i])>=0.25){
                number<-append(number,i)
                }
}
#⊿Piの作成
V<-volume2[number]
P<-price2[number]
Time<-time[number]
nagasaV<-length(V)
deltaP<-diff(P,lag=1)
i<-0
deltaPi<-NULL
yu<-NULL
u=NULL
nagasax<-length(V)
while(i<(nagasax-1)){
        i=i+1
        y<-rep(deltaP[i],V[i])
        ny<-rep(Time[i],V[i])
        deltaPi<-append(deltaPi,y)
        u<-append(u,ny)
        }
```

7.2 プログラム（２）

3

```
#バケットに詰め込む作業
I<-sum(V)
sigmadeltaP<-sqrt(var(deltaPi))
u<-u[!is.na(u)]
B<-1000 ##バケットの容量
VBtau<-NULL
counter<-NULL
tau<-0
while(I>tau*B){
        tau<-tau+1
        if (I<tau*B){ break}
        r<-deltaPi/sigmadeltaP
        q<-r[((tau-1)*B+1):(tau*B)]
        Z<-pnorm(q,mean=0,sd=1)
        VBtau<-append(VBtau,sum(Z,na.rm=TRUE))
        counter<-append(counter,u[(tau-1)*B+1])
}
counter<-counter[!is.na(counter)]
Vstau<-B-VBtau

#ここからVPINを計算。
L<-tau-1
n<-49  ##バケット数。今回は５０
i<-0
VPIN<-NULL
while((i+n)<L){
        i<-i+1
        s<-VStau[i:(i+n)]
        b<-VBtau[i:(i+n)]

        upper<-sum(abs(s-b))
        lower<-(n+1)*B
        VPIN<-append(VPIN,upper/lower)
        }
```

4

```
##VPIN累積分布関数のデータ##
sor<-sort(VPIN)
per<-sor/sum(VPIN)
VPINcdf<-cumsum(per)
i<-0
Vcdf<-NULL
while(i<length(VPIN)){
        i<-i+1
        num<-which(abs(sor-VPIN[i])==min(abs(sor-VPIN[i])))
        Vcdf<-append(Vcdf,VPINcdf[num[1]])
}

#累積分布のグラフ作成
graphsheet(format="JPG",file="ファイル名.jpg")
plot(sor,VPINcdf,xlim=c(0,1),type="l",xlab="VPIN",col=1,lwd=3)
par(new=T)
plot(sornormal,normalcdf,xlim=c(0,1),ylim=c(0,1),type="l",xlab="VPIN"
,ylab="VPINcdf",lwd=3,col=2)
abline(h=0,lty=8,col=2)
abline(h=1,lty=8,col=2)
abline(v=0,lty=8,col=2)
abline(v=1,lty=8,col=2)
lines(x=c(sornormal[length(sornormal)],1),y=c(1,1),lwd=3,col=2)
lines(x=c(sornormal[1],0),y=c(0,0),lwd=3,col=2)
lines(x=c(sor[length(sor)],1),y=c(1,1),lwd=3,col=1)
lines(x=c(sor[1],0),y=c(0,0),lwd=3,col=1)
dev.off()
delete<-length(tt)
price2<-price2[-c(1:delete)]
volume2<-volume2[-c(1:delete)]
time<-time[-c(1:delete)]
count<-counter[-c(1:237)]
VPINcount<-VPIN[-c(1:(237-n))]
Vcdfcount<-Vcdf[-c(1:(237-n))]
```

7.3 プログラム（３）

5

```
#データの出力
VPINdata<-NULL
Vcdfdata<-NULL
i<-0
c<-1
while(i<=length(time)){
        i<-i+1
        if(i>length(time)){break}
        if(c==length(count)){c<-c-1}
        if(time[i]>count[c] &&count[c]==count[c-1]){c<-c+1}
        if(time[i]>count[c] &&count[c-2]==count[c-1]){c<-c+1}
        if(time[i] == count[c]){
                VPINdata<-append(VPINdata,VPINcount[c])
                Vcdfdata<-append(Vcdfdata,Vcdfcount[c])
                c<-c+1
                }
        else{
                VPINdata<-append(VPINdata,NA)
                Vcdfdata<-append(Vcdfdata,NA)
        }
}
data<-
data.frame(Time=time,Price=price2,VPIN=VPINdata,VPINcd
f=Vcdfdata)
write.table(data,"ファイルの場所")
```