MF731 Capital Charge and Allocation under FRTB

Nov 27, 2022

Haoran Chen, Pei Zhu, Xueyi Wang, Xuyang Liu

In [1]:

```
# import packages
import numpy as np
import pandas as pd
```

1. Consider a portfolio of TWO risk positions. Each risk position can be thought as one asset, which has exposure to all different *risk factors* RF_i, $i = 1, \ldots, 5$, and different *liquidity horizons* LH_j, $j = 1, \ldots, 5$:

$$\{RF_i : 1 \leq i \leq 5\} = \{CM, CR, EQ, FX, IR\},$$

$$\{LH_j : 1 \leq j \leq 5\} = \{10, 20, 40, 60, 120 days\}.$$

In [2]:

```
# different liquidity horizons
LH = np.array([10, 20, 40, 60, 120])
```

2. Suppose that the 10 days loss of risk position n, attributed to RF_i and LH_j, is denoted by $\tilde{X}_n(i, j)$, for $n = 1, 2$, $1 \leq i, j \leq 5$.

Assume that $\tilde{X}_1(i, j) \sim N(0.004, 0.04)$ and $\tilde{X}_2(i, j) \sim N(0.006, 0.05)$,

where 0.04 and 0.05 are the standard deviation of these normal distributions. The correlation between any two different $\tilde{X}_1(i, j)$ and $\tilde{X}_1(k, l)$ is assumed to be 0.3. The correlation between any two different $\tilde{X}_2(i, j)$ and $\tilde{X}_2(k, l)$ is assumed to be 0.1. $\tilde{X}_1(k, l)$ and $\tilde{X}_2(k, l)$ are assumed to be independent.

In [3]:

```
# (2) Simulate 10 days loss of risk position
def simulate_xij(mu, sigma, pho):
    z = np.random.normal(0, 1)
    Xij = (np.sqrt(pho)*z + np.sqrt(1-pho)*np.random.normal(0, 1, [5, 5]))*sigma+mu
    return Xij
```

3. The total loss of the risk position n is

$$\tilde{X}_n = \sum_{i,j}^{5} \tilde{X}_n(i, j), \qquad n = 1, 2.$$

The total loss of the portfolio is

$$\tilde{X} = \tilde{X}_1 + \tilde{X}_2.$$

In [4]:

```
def simulate_x(mu,sigma,pho,n):
    X_list = np.array([])
    for i in range(n):
        Xij = simulate_xij(mu,sigma,pho)
        X = np.sum(Xij)
        X_list = np.append(X_list,X)
    return X_list
```

In [5]:

```
n = 10000
X1 = simulate_x(0.004,0.04,0.3,n)
X2 = simulate_x(0.006,0.05,0.1,n)
Xlist = X1 + X2
```

4. **(Standard VaR and ES)** Use the simulation method to calculate the $VaR_{0.99}$ and $ES_{0.975}$ for the 10 days loss of the portfolio. To simulate different $\tilde{X}_1(i,j)$ with pair correlation 0.3, we can set

$$\frac{\tilde{X}_1(i,j) - 0.004}{0.04} = \sqrt{0.3}Z + \sqrt{0.7}Z(i,j)$$

where Z and $\{Z_{i,j}\}_{1\le i,j\le 5}$ are all independent standard normal random variables. Similar method can be used to simulate $\tilde{X}_2(i,j)$.

In [6]:

```
VaR1 = np.quantile(X1,0.99)
VaR2 = np.quantile(X2,0.99)
```

In [7]:

```
def ES(x,alpha=0.975):
    return 1/(1-alpha) * np.sum(x[x>=np.quantile(x,alpha)])/n
```

In [8]:

```
ES1 = ES(X1)
ES2 = ES(X2)
VaR1,VaR2,ES1,ES2
```

Out[8]:

```
(1.4336537005075556,
 1.2316109069196408,
 1.4465213740330123,
 1.2379846917409074)
```

In [9]:

```
# (4) Calculate the VaR_0.99 andES_0.975 for the 10 days loss of the portfolio
print("The $VaR$ 0.99 for the 10 days loss of the portfolio is",np.quantile(Xlist,0.99))
print("The $ES$ 0.975 for the 10 days loss of the portfolio is",ES(Xlist))
```

The VaR 0.99 for the 10 days loss of the portfolio is 1.9575005801431595
The ES 0.975 for the 10 days loss of the portfolio is 1.9543025075812748

5. **(VaR and ES allocations)** For each simulation of $\tilde{X}_n(i,j)$, we have a simulation of $\tilde{X}_1$,$\tilde{X}_2$, and $\tilde{X}$. Using these simulations to compute Euler allocations for VaR and ES:

• Among all simulations, find those with $\tilde{X} \in (VaR_{0.99} - \epsilon, VaR_{0.99} + \epsilon)$, for a small ϵ. Take average of $\tilde{X}_1$ and $\tilde{X}_2$ for these simulations. They are Euler allocations of $VaR_{0.99}$ for the risk position 1 and 2, respectively.

• Among all simulation, find those with $\tilde{X} \geq VaR_{0.975}$. Take average of $\tilde{X}_1$ and $\tilde{X}_2$ for these simulations. They are Euler allocations of $ES_{0.975}$ for the risk position 1 and 2, respectively.

In [10]:

```
# (5) Set error as 0.001
def var_alloc(X,error,VaR):
    C1 = np.where(Xlist >= VaR - error )
    C2 = np.where(Xlist <= VaR + error)
    C = np.intersect1d(C1,C2)
    return np.mean(X[C])

def es_alloc(X,VaR):
    C = np.where(Xlist >= VaR)
    return np.mean(X[C])
```

In [11]:

```
# Then the VaR and ES allocation shoule be:
error = 0.001
print("Euler allocations for $VaR$ 0.99 of $\tilde{X}1$ is", var_alloc(X1,error,np.quantile(Xlist,0.99)))
print("Euler allocations for $VaR$ 0.99 of $\tilde{X}2$ is", var_alloc(X2,error,np.quantile(Xlist,0.99)))
print("Euler allocations for $ES$ 0.975 of $\tilde{X}1$ is", es_alloc(X1,np.quantile(Xlist,0.975)))
print("Euler allocations for $ES$ 0.975 of $\tilde{X}2$ is", es_alloc(X2,np.quantile(Xlist,0.975)))
```

Euler allocations for VaR 0.99 of $\tilde{X}1$ is 1.4029856783987258
Euler allocations for VaR 0.99 of $\tilde{X}2$ is 0.5544034434406837
Euler allocations for ES 0.975 of $\tilde{X}1$ is 1.1240955766513272
Euler allocations for ES 0.975 of $\tilde{X}2$ is 0.8302069309299489

6. In FRTB, the liquidity horizon adjusted loss for risk position n is

$$X_n(i,j) = \sqrt{\frac{LH_j - LH_{j-1}}{10}} \sum_{k=j}^{5} \tilde{X}_n(i,k), \quad 1 \leq i,j \leq 5.$$

The liquidity horizon adjusted loss for the portfolio is

$$X(i,j) = X_1(i,j) + X_2(i,j), \quad 1 \leq i,j \leq 5.$$

In [12]:

```
# (6) FRTB liquidity horizon adjusted loss for risk position
def adj_loss(mu,sigma,pho):
    Xij = simulate_xij(mu,sigma,pho)
    Xn = np.zeros((5,5))
    lh = np.append(0,LH)
    for i in range(5):
        for j in range(5):
            Xn[i,j] = np.sqrt((lh[j+1]-lh[j])/10)*np.sum(Xij[i,j:])
    return Xn

# Re-simulate n times adjusted X
def simulate_nX(n):
    Xlist = []
    X1 = []
    X2 = []
    for i in range(n):
        X1n = adj_loss(0.004,0.04,0.3)
        X2n = adj_loss(0.006,0.05,0.1)
        Xn = X1n+X2n
        X1.append(X1n)
        X2.append(X2n)
        Xlist.append(Xn)
    return Xlist,X1,X2
```

7. **(FRTB ES)** For each $i = 1, \ldots, 5$, the FRTB expected shortfall for portfolio loss attributed to RF_i is

$$ES(X(i)) = \sqrt{\sum_{j=1}^{5} ES_{0.975}(X(i,j))^2}.$$

Denote the previous expected shortfall as $ES^{F,C}(X(i))$.

In [13]:

```
# (7) Calaclate ES of FRTB
def FRTB_ES(X):
    newX = np.zeros((5,5))
    for i in range(5):
        for j in range(5):
            s = np.array([])
            for k in range(len(X)):
                s = np.append(s,X[k][i,j])
            #newX[i,j] = ES(s) # Same results as the line below
            newX[i,j] = np.mean(s[s[:]>=np.quantile(s,0.975)])
    return np.sqrt(np.sum(newX**2,1)),newX
```

In [14]:

```
adj_X,adj_X1,adj_X2 = simulate_nX(n)
FC_ES,tol_ES = FRTB_ES(adj_X)
print("The previous expected shortfall is",FC_ES)
```

```
The previous expected shortfall is [0.96487921 0.9522721  0.94392673 0.9562566  0.96
226636]
```

8. **(FRTB ES capital charge)** Assume that $ES^{R,S}(X(i))/ES^{R,C}(X(i)) = 2$ for all $1 \leq i \leq 5$. (In practice, this ratio is calculated using the loss data in the stress period and in the current 12 months.) The FRTB ES capital charge for RF_i is

$$IMCC(X(i)) = \frac{ES^{R,S}(X(i))}{ES^{R,C}(X(i))} ES^{F,C}(X(i)), 1 \leq i \leq 5.$$

In [15]:

```
# (8) FRTB ES capital charge
IMCC_Xi = 2 * FC_ES
print("The FRTB ES capital charge is",IMCC_Xi)
```

```
The FRTB ES capital charge is [1.92975842 1.9045442  1.88785346 1.9125132  1.9245327
2]
```

9. For the risk position n, the unconstrained portfolio with LH_j is

$$X_n(6, j) = \sum_{i=1}^{5} X_n(i, j), \quad 1 \leq j \leq 5.$$

For the portfolio, the unconstrained portfolio with LH_j is

$$X(6, j) = X_1(6, j) + X_2(6, j).$$

$IMCC(X(6))$ is calculated similarly as in item 7 and 8 with $i = 6$.

In [16]:

```
# (9) Unconstrained portfolio with LH
def unconst_ES(X,n):
    unX = np.zeros((n,5))
    newES = np.zeros(5)
    for k in range(n):
        unX[k,:] = np.sum(X[k],0)
    for j in range(5):
        s = np.array([])
        for k in range(len(unX)):
            s = np.append(s,unX[k][j])
        newES[j] = np.mean(s[s[:]>=np.quantile(s,0.975)]) #ES(s)
    return np.sqrt(np.sum(newES**2)),newES

ES_X6,arrEuler = unconst_ES(adj_X,n)
IMCC_X6 = 2*ES_X6
print("The $IMCC(X(6))$ is",IMCC_X6)
```

The $IMCC(X(6))$ is 7.0309539776713565

10. **(FRTB capital charge for modellable risk factors)** For the portfolio loss X, its aggregate capital charge for modellable risk factors is

$$IMCC(X) = 0.5IMCC(X(6)) + 0.5\sum_{i=1}^{5} IMCC(X(i)).$$

In [17]:

```
# (10) FRTB capital charge for modellable risk factors
IMCC = 0.5*IMCC_X6 + 0.5*np.sum(IMCC_Xi)
print("The aggregate capital charge for modellable risk factors is",IMCC)
```

The aggregate capital charge for modellable risk factors is 8.295077988405822

12. **(FRTB Euler allocation of $X(i,j)$)** For each i,j, use the simulations of $\tilde{X}_n(i,j)$ in item 2 to simulate $X_1(i,j)$, $X_2(i,j)$, and $X(i,j)$ in item 6. Calculate $VaR_{0.975}(X(i,j))$. Find all simulations with $X(i,j) \geq VaR_{0.975}(X(i,j))$. Among all these simulations, calculate the average of $X_1(i,j)$ and $X_2(i,j)$. They are Euler allocation of $X(i,j)$. We denote them as $ES(X_1(i,j)|X(i,j))$ and $ES(X_2(i,j)|X(i,j))$.

In [18]:

```
# (12) FRTB Euler allocation of X(i,j)
def Euler_ES(X,tX):
    newES = np.zeros((5,5))
    for i in range(5):
        for j in range(5):
            s = np.array([])
            ts = np.array([])
            for k in range(len(X)):
                s = np.append(s,X[k][i,j])
                ts = np.append(ts,tX[k][i,j])
            var = np.quantile(s,0.975)
            newES[i,j] = np.mean(ts[s[:]>=var])
    return newES
```

In [19]:

```
Euler_ES1 = Euler_ES(adj_X,adj_X1)
Euler_ES2 = Euler_ES(adj_X,adj_X2)
```

$ES(X_1(i,j)|X(i,j))$:

In [20]:

```
Euler_ES1
```

Out[20]:

```
array([[0.23654151, 0.19869192, 0.2162014 , 0.15656567, 0.16702301],
       [0.2364705 , 0.19763232, 0.21273252, 0.15398708, 0.15694212],
       [0.22688353, 0.19221149, 0.20840142, 0.15301951, 0.16012753],
       [0.23375018, 0.19291366, 0.21034905, 0.14536061, 0.1495991 ],
       [0.24918584, 0.20193303, 0.21846726, 0.1518283 , 0.14907566]])
```

$ES(X_2(i,j)|X(i,j))$:

In [21]:

```
Euler_ES2
```

Out[21]:

```
array([[0.25706179, 0.22067556, 0.26396481, 0.20137376, 0.22387129],
       [0.25696171, 0.21392934, 0.25543738, 0.197455  , 0.2319913 ],
       [0.25072536, 0.21118575, 0.25522982, 0.20193434, 0.23888618],
       [0.25758103, 0.22059549, 0.25477045, 0.20949027, 0.2501198 ],
       [0.24554108, 0.21552513, 0.25920615, 0.20512217, 0.23995516]])
```

13. For each $1 \leq i \leq 5$, use the results in item 7 and 12 to calculate

$$ES(X_n(i,j)|X(i)) = \frac{ES(X(i,j))}{ES(X(i))} ES(X_n(i,j)|X(i,j)), \quad n = 1,2, j = 1,\ldots,5.$$

In [22]:

```
# (13) Use given formula to calculate
Euler_FC_ES1 = Euler_ES1*tol_ES/FC_ES.reshape(5,1)
Euler_FC_ES2 = Euler_ES2*tol_ES/FC_ES.reshape(5,1)
```

$ES(X_1(i,j)|X(i))$:

In [23]:

```
Euler_FC_ES1
```

Out[23]:

```
array([[0.12100755, 0.08635789, 0.1075913 , 0.05808087, 0.06766478],
       [0.12253027, 0.08541454, 0.10458667, 0.05682992, 0.06409936],
       [0.11479873, 0.08214365, 0.10236113, 0.0575414 , 0.0676886 ],
       [0.12010245, 0.08342067, 0.10231296, 0.0539409 , 0.062533  ],
       [0.12811312, 0.08760422, 0.10844815, 0.05632036, 0.0602692 ]])
```

$ES(X_2(i,j)|X(i))$:

In [24]:

```
Euler_FC_ES2
```

Out[24]:

```
array([[0.13150511, 0.09591268, 0.13136047, 0.07470325, 0.0906953 ],
       [0.13314806, 0.09245794, 0.12558185, 0.07287203, 0.09475146],
       [0.12686224, 0.0902525 , 0.12536197, 0.07593531, 0.10098121],
       [0.1323469 , 0.09539098, 0.12391936, 0.07773835, 0.10455103],
       [0.12623925, 0.09350085, 0.12867111, 0.07608959, 0.09701051]])
```

Check that

$$\sum_{n=1}^{2}\sum_{j=1}^{5} ES(X_n(i,j)|X(i)) = ES(X(i)).$$

Denote $ES(X_n(i,j)|X(i))$ as $ES^{F,C}(X_n(i,j)|X(i))$.

In [25]:

```
Euler_FC_ES = np.sum(Euler_FC_ES1,1) + np.sum(Euler_FC_ES2,1)
Euler_FC_ES
```

Out[25]:

```
array([0.96487921, 0.9522721 , 0.94392673, 0.9562566 , 0.96226636])
```

In [26]:

```
# Check if the results same
FC_ES # Same as Euler_FC_ES
```

Out[26]:

```
array([0.96487921, 0.9522721 , 0.94392673, 0.9562566 , 0.96226636])
```

14. Follow the same method as in item 13, calculate $ES^{F,C}(X_n(6,j)|X(6))$

In [27]:

```
# (14) Use the same method metioned in 13 to calculate ES_X6
def const_ES(tX,X,n):
    cX = np.zeros((n,5))
    ctX = np.zeros((n,5))
    newES = np.zeros(5)
    for k in range(n):
        cX[k,:] = np.sum(X[k],0)
        ctX[k,:] = np.sum(tX[k],0)
    for j in range(5):
        s = np.array([])
        ts = np.array([])
        for k in range(len(cX)):
            s = np.append(s,cX[k][j])
            ts = np.append(ts,ctX[k][j])
        var = np.quantile(s,0.975)
        newES[j] = np.mean(ts[s[:]>=var])
    return np.sqrt(np.sum(newES**2)),newES
```

In [28]:

```
Euler1,arrEuler1 = const_ES(adj_X1,adj_X,n)
Euler2,arrEuler2 = const_ES(adj_X2,adj_X,n)
```

In [29]:

```
IMCC1_6 = arrEuler1*arrEuler/ES_X6
IMCC2_6 = arrEuler2*arrEuler/ES_X6
```

$ES^{F,C}(X_1(6,j)|X(6))$:

In [30]:

```
IMCC1_6
```

Out[30]:

```
array([0.64249509, 0.42411612, 0.48821025, 0.23342908, 0.20415393])
```

$ES^{F,C}(X_2(6,j)|X(6))$:

In [31]:

```
IMCC2_6
```

Out[31]:

```
array([0.44607335, 0.30573963, 0.37378004, 0.18831205, 0.20916744])
```

In [32]:

```
# Check
np.sum(IMCC1_6) + np.sum(IMCC2_6)
```

Out[32]:

```
3.5154769888356787
```

In [33]:

```
ES_X6 # Same
```

Out[33]:

```
3.5154769888356783
```

15. **(Euler allocation of IMCC)** Use the assumption in item 8, calculate

$$IMCC(X_n(i,j)|X(i)) = 0.5\frac{ES^{R,S}(X(i))}{ES^{R,C}(X(i))}ES^{F,C}(Xn(i,j)|X(i)),$$

for $i = 1, \ldots, 6$ and $j = 1, \ldots, 5$.

In [34]:

```
# (15) Euler allocation of IMCC
Euler_IMCC1 = np.append(Euler_FC_ES1, IMCC1_6).reshape(6, 5)
Euler_IMCC2 = np.append(Euler_FC_ES2, IMCC2_6).reshape(6, 5)
```

$IMCC(X_1(i,j)|X(i))$:

In [35]:

```
Euler_IMCC1
```

Out[35]:

```
array([[0.12100755, 0.08635789, 0.1075913 , 0.05808087, 0.06766478],
       [0.12253027, 0.08541454, 0.10458667, 0.05682992, 0.06409936],
       [0.11479873, 0.08214365, 0.10236113, 0.0575414 , 0.0676886 ],
       [0.12010245, 0.08342067, 0.10231296, 0.0539409 , 0.062533  ],
       [0.12811312, 0.08760422, 0.10844815, 0.05632036, 0.0602692 ],
       [0.64249509, 0.42411612, 0.48821025, 0.23342908, 0.20415393]])
```

$IMCC(X_2(i,j)|X(i))$:

In [36]:

```
Euler_IMCC2
```

Out[36]:

```
array([[0.13150511, 0.09591268, 0.13136047, 0.07470325, 0.0906953 ],
       [0.13314806, 0.09245794, 0.12558185, 0.07287203, 0.09475146],
       [0.12686224, 0.0902525 , 0.12536197, 0.07593531, 0.10098121],
       [0.1323469 , 0.09539098, 0.12391936, 0.07773835, 0.10455103],
       [0.12623925, 0.09350085, 0.12867111, 0.07608959, 0.09701051],
       [0.44607335, 0.30573963, 0.37378004, 0.18831205, 0.20916744]])
```

16. For each $i = 1, \ldots, 6$, calculate

$$IMCC(\tilde{X}_n(i,k)|X_n(i,j)) = \frac{1}{5-j+1} IMCC(X_n(i,j)|X(i)), \quad k \geq j.$$

In [37]:

```
# (16) Follow the instruction
def Euler_IMCC(X):
    IMCC = np.zeros((6,5))
    for i in range(6):
        for j in range(5):
            for k in range(j,5):
                IMCC[i,k] = 1/(6-j)*X[i,j]
    return IMCC
```

$IMCC(\tilde{X}_1(i,k)|X_1(i,j))$:

In [38]:

```
Euler_IMCC(Euler_IMCC1)
```

Out[38]:

```
array([[0.02016793, 0.01727158, 0.02689783, 0.01936029, 0.03383239],
       [0.02042171, 0.01708291, 0.02614667, 0.01894331, 0.03204968],
       [0.01913312, 0.01642873, 0.02559028, 0.01918047, 0.0338443 ],
       [0.02001707, 0.01668413, 0.02557824, 0.0179803 , 0.0312665 ],
       [0.02135219, 0.01752084, 0.02711204, 0.01877345, 0.0301346 ],
       [0.10708251, 0.08482322, 0.12205256, 0.07780969, 0.10207696]])
```

$IMCC(\tilde{X}_2(i,k)|X_2(i,j))$:

In [39]:

```
Euler_IMCC(Euler_IMCC2)
```

Out[39]:

```
array([[0.07934928, 0.07333902, 0.12740439, 0.10078796, 0.19660875],
       [0.081682  , 0.07436436, 0.11979765, 0.10148679, 0.19405976],
       [0.07990829, 0.07072363, 0.12531938, 0.10060962, 0.19406844],
       [0.08070722, 0.07392311, 0.12510372, 0.10311966, 0.19617826],
       [0.07886085, 0.06900546, 0.11809178, 0.10103233, 0.21361261],
       [0.28540192, 0.23294651, 0.34554281, 0.24521777, 0.38427236]])
```

Finally, the Euler allocation of IMCC is

$$IMCC(\tilde{X}_n(i,k)|X(i)) = \sum_{j=1}^{k} IMCC(\tilde{X}_n(i,k)|X_n(i,j))$$

17. Report $IMCC(\tilde{X}_n(i,k)|X(i))$, $i = 1, \ldots, 6$ and $k = 1, \ldots, 5$.

In [39]:

```
# (17)
def allocation(X):
    new_IMCC = np.zeros((6,5))
    for i in range(6):
        s = 0
        for j in range(5):
            s += X[i,j]
            new_IMCC[i,j] = s
    return new_IMCC
```

$IMCC(\tilde{X}_1(i,k)|X(i))$:

In [41]:

```
allocation(Euler_IMCC(Euler_IMCC1))
```

Out[41]:

```
array([[0.10625681, 0.1904126 , 0.32367246, 0.41621327, 0.57835408],
       [0.10757009, 0.1948961 , 0.33432495, 0.42798695, 0.5977668 ],
       [0.09697216, 0.18058799, 0.31362563, 0.41227352, 0.5852871 ],
       [0.10479894, 0.19127595, 0.32701216, 0.42057814, 0.59326539],
       [0.10434982, 0.19178785, 0.32646955, 0.4188397 , 0.55958825],
       [0.52534257, 0.93805291, 1.56758525, 1.96212579, 2.47700336]])
```

$IMCC(\tilde{X}_2(i,k)|X(i))$:

In [40]:

```
allocation(Euler_IMCC(Euler_IMCC2))
```

Out[40]:

```
array([[0.02191752, 0.04110006, 0.07394017, 0.09884125, 0.14418891],
       [0.02219134, 0.04068293, 0.07207839, 0.09636907, 0.1437448 ],
       [0.02114371, 0.03919421, 0.0705347 , 0.09584647, 0.14633707],
       [0.02205782, 0.04113601, 0.07211585, 0.09802864, 0.15030415],
       [0.02103988, 0.03974005, 0.07190782, 0.09727102, 0.14577628],
       [0.07434556, 0.13549348, 0.2289385 , 0.29170918, 0.3962929 ]])
```

In [ ]:

```

```