MIE 1622H: Assignment 3 – Credit Risk Modeling and Simulation

Dr. Oleksandr Romanko, Jiaoyang Li

March 9, 2025

Due: Wednesday, March 26, 2025, not later than 11:59 p.m.

Use Python for all MIE 1622H assignments.

You should hand in:

- Your report (pdf file and docx file). Maximum page limit is 6 pages.
- Compress all of your Python code (i.e., ipynb notebook with plots and necessary outputs intact) into a zip file and submit it via Quercus portal no later than 11:59 p.m. on March 26.

Where to hand in: Online via Quercus portal, both your code and report.

Introduction

The purpose of this assignment is to model a credit-risky portfolio of corporate bonds. Consider a structural model for portfolio credit risk described in class. Using the data for 100 counterparties, simulate 1-year losses for each corporate bond. You will need to generate 3 sets of scenarios:

- *Monte Carlo approximation 1*: 5000 in-sample scenarios ($N = 1000 \cdot 5 = 5000$ (1000 systemic scenarios and 5 idiosyncratic scenarios for each systemic), non-Normal distribution of losses);
- *Monte Carlo approximation 2*: 5000 in-sample scenarios ($N = 5000$ (5000 systemic scenarios and 1 idiosyncratic scenario for each systemic), non-Normal distribution of losses);
- *True distribution*: 100000 out-of-sample scenarios ($N = 100000$ (100000 systemic scenarios and 1 idiosyncratic scenario for each systemic), non-Normal distribution of losses).

The out-of-sample scenarios represent true distribution of portfolio losses. Two in-sample non-Normal datasets are used for evaluating sampling error and performing portfolio optimization (portfolio credit-risk optimization is the subject of Assignment 4).

If we incorrectly assume losses for each corporate bond are Normally distribute,

To evaluate model error (if we wrongly assumed that counterparty losses follow Normal distribution), compute mean loss and standard deviation of losses for each corporate bond from the 3 scenario sets (N=1000×5, 5000, 100000). This 3 in-sample sets are referred as in-sample Normal model.

In-sample normal means if incorrectly consider it as Normal Distribution,
this is used for evaluate model error

Evaluate VaR and CVaR at quantile levels 99% and 99.9% for the two portfolios:

(1) one unit invested in each of 100 bonds;

(2) equal value (dollar amount) is invested in each of 100 bonds;

For each portfolio, compute VaR and CVaR at quantile levels 99% and 99.9% for each of the six datasets (non-Normal with N=1000×5, 5000, 100000; and Normal with mean/standard deviation computed from N=1000×5, 5000, 100000). Compare each in-sample VaR and CVaR to out-of-sample VaR and CVaR. Explain the effects of sampling error and model error.

To better evaluate sampling and model errors, perform the experiment 100 times in the following way: generate in-sample datasets with N=1000×5 and 5000 (non-Normal and Normal) one hundred times and compute VaR and CVaR. Keep the out-of-sample scenario set unchanged. Perform analysis of the results for the 100 trials, e.g., compute averages of the results for 100 trials, analyze standard deviation over 100 trials, etc.

Questions

1. **(50 %) Implement portfolio credit risk simulation model in Python:**

   There is a file **credit_risk_simul.ipynb** on the course web-page. You are required to complete the code in the file.

2. **(30 %) Analyze your results:**

   - Produce the following output from your Python code:

     ```
     Portfolio 1:
     Out-of-sample: VaR 99.0% = ..., CVaR 99.0% = ...
     In-sample MC1: VaR 99.0% = ..., CVaR 99.0% = ...
     In-sample MC2: VaR 99.0% = ..., CVaR 99.0% = ...
      In-sample No: VaR 99.0% = ..., CVaR 99.0% = ...
      In-sample N1: VaR 99.0% = ..., CVaR 99.0% = ...
      In-sample N2: VaR 99.0% = ..., CVaR 99.0% = ...
     Out-of-sample: VaR 99.9% = ..., CVaR 99.9% = ...
      ...
     Portfolio 2:
     Out-of-sample: VaR 99.0% = ..., CVaR 99.0% = ...
      ...
     ```

   - Plot loss distributions in Python that illustrate both out-of-sample and in-sample results. Include plots that help illustrating your analysis in the report.
   - Analyze sampling error when comparing non-Normal approximations to the true (out-of-sample) loss distribution. Analyze model error when comparing Normal approximations to the true (out-of-sample) loss distribution. Summarize the two types of errors in the tables.

3. **(20 %) Discuss possible strategies for minimizing impacts of sampling and model errors:**

   - If you report the in-sample VaR and CVaR to decision-makers in your bank, what consequences for the bank capital requirements it may have?
   - Can you suggest techniques for minimizing impacts of sampling and model errors?

Python Code to be Completed

```
# Import libraries
import numpy as np
import pandas as pd
import scipy
from scipy import special
from pathlib import Path


Nout = 100000  # number of out-of-sample scenarios
Nin = 5000     # number of in-sample scenarios
Ns = 5         # number of idiosyncratic scenarios for each systemic

C = 8          # number of credit states

# Read and parse instrument data
instr_data = np.array(pd.read_csv('instrum_data.csv', header=None))
instr_id = instr_data[:, 0]     # ID
driver = instr_data[:, 1]       # credit driver
beta = instr_data[:, 2]         # beta (sensitivity to credit driver)
recov_rate = instr_data[:, 3]   # expected recovery rate
value = instr_data[:, 4]        # value
prob = instr_data[:, 5:(5 + C)] # credit-state migration probabilities (default to AAA)
exposure = instr_data[:, 5 + C:5 + 2 * C]  # credit-state migration exposures (default to AAA)
retn = instr_data[:, 5 + 2 * C] # market returns

K = instr_data.shape[0]         # number of CPs

# Read matrix of correlations for credit drivers
rho = np.array(pd.read_csv('credit_driver_corr.csv', sep='\t', header=None))
# Cholesky decomp of rho (for generating correlated Normal random numbers)
sqrt_rho = np.linalg.cholesky(rho)

print('======= Credit Risk Model with Credit-State Migrations =======')
print('============== Monte Carlo Scenario Generation ===============')
print(' ')
print(' ')
print(' Number of out-of-sample Monte Carlo scenarios = ' + str(Nout))
print(' Number of in-sample Monte Carlo scenarios = ' + str(Nin))
print(' Number of counterparties = ' + str(K))
print(' ')

# Find credit-state for each counterparty
# 8 = AAA, 7 = AA, 6 = A, 5 = BBB, 4 = BB, 3 = B, 2 = CCC, 1 = default
CS = np.argmax(prob, axis=1) + 1

# Account for default recoveries
exposure[:, 0] = (1 - recov_rate) * exposure[:, 0]

# Compute credit-state boundaries
CS_Bdry = scipy.special.ndtri((np.cumsum(prob[:, 0:C - 1], 1)))

# -------- Insert your code here -------- #
if Path(filename_save_out+'.npz').is_file():
    Losses_out = scipy.sparse.load_npz(filename_save_out + '.npz')
else:
    # Generating Scenarios

    # -------- Insert your code here -------- #

    for s in range(1, Nout + 1):
        # -------- Insert your code here -------- #

        # Calculated out-of-sample losses (100000 x 100)
        # Losses_out (sparse matrix)
        Losses_out = #...
```

```
# Normal approximation computed from out-of-sample scenarios
mu_l = np.mean(Losses_out, axis=0).reshape((K))
var_l = np.cov(Losses_out.toarray(), rowvar=False) # Losses_out as a sparse matrix

# Compute portfolio weights
portf_v = sum(value)  # portfolio value
w0 = []
w0.append(value / portf_v)   # asset weights (portfolio 1)
w0.append(np.ones((K)) / K)  # asset weights (portfolio 2)
x0 = []
x0.append((portf_v / value) * w0[0])  # asset units (portfolio 1)
x0.append((portf_v / value) * w0[1])  # asset units (portfolio 2)

# Quantile levels (99%, 99.9%)
alphas = np.array([0.99, 0.999])

VaRout = np.zeros((2, alphas.size))
VaRinN = np.zeros((2, alphas.size))
CVaRout = np.zeros((2, alphas.size))
CVaRinN = np.zeros((2, alphas.size))

for portN in range(2):
    # Compute VaR and CVaR
    for q in range(alphas.size):
        alf = alphas[q]
        # -------- Insert your code here -------- #

        VaRout[portN, q] = #...
        VaRinN[portN, q] = #...
        CVaRout[portN, q] = #...
        CVaRinN[portN, q] = #...

# Perform 100 trials
N_trials = 100

VaRinMC1 = {}
VaRinMC2 = {}
VaRinN1 = {}
VaRinN2 = {}
CVaRinMC1 = {}
CVaRinMC2 = {}
CVaRinN1 = {}
CVaRinN2 = {}

for portN in range(2):
    for q in range(alphas.size):
        VaRinMC1[portN, q] = np.zeros(N_trials)
        VaRinMC2[portN, q] = np.zeros(N_trials)
        VaRinN1[portN, q] = np.zeros(N_trials)
        VaRinN2[portN, q] = np.zeros(N_trials)
        CVaRinMC1[portN, q] = np.zeros(N_trials)
        CVaRinMC2[portN, q] = np.zeros(N_trials)
        CVaRinN1[portN, q] = np.zeros(N_trials)
        CVaRinN2[portN, q] = np.zeros(N_trials)


for tr in range(1, N_trials + 1):
    # Monte Carlo approximation 1

    # -------- Insert your code here -------- #


    for s in range(1, np.int32(np.ceil(Nin / Ns) + 1)): # systemic scenarios
        # -------- Insert your code here -------- #

        for si in range(1, Ns + 1): # idiosyncratic scenarios for each systemic
            # -------- Insert your code here -------- #

    # Calculate losses for MC1 approximation (5000 x 100)
```

```
    # Losses_inMC1


    # Monte Carlo approximation 2

    # -------- Insert your code here -------- #


    for s in range(1, Nin + 1): # systemic scenarios (1 idiosyncratic scenario for each systemic)
        # -------- Insert your code here -------- #

    # Calculated losses for MC2 approximation (5000 x 100)
    # Losses_inMC2


    # Compute VaR and CVaR

    for portN in range(2):
        for q in range(alphas.size):
            alf = alphas[q]
            # -------- Insert your code here -------- #
            # Compute portfolio loss
            portf_loss_inMC1 = #...
            portf_loss_inMC2 = #...
            mu_MC1 = np.mean(Losses_inMC1, axis=0).reshape((K))
            var_MC1 = np.cov(Losses_inMC1.toarray(), rowvar=False)
            mu_MC2 = np.mean(Losses_inMC2, axis=0).reshape((K))
            var_MC2 = np.cov(Losses_inMC2.toarray(), rowvar=False)
            # Compute portfolio mean loss mu_p_MC1 and portfolio standard deviation of losses sigma_p_MC1
            # Compute portfolio mean loss mu_p_MC2 and portfolio standard deviation of losses sigma_p_MC2
            # Compute VaR and CVaR for the current trial
            mu_p_MC1 = #...
            sigma_p_MC1 = #...
            mu_p_MC2 = #...
            sigma_p_MC2 = #...
            VaRinMC1[portN, q][tr - 1] = #...
            VaRinMC2[portN, q][tr - 1] = #...
            VaRinN1[portN, q][tr - 1] =  #...
            VaRinN2[portN, q][tr - 1] =  #...
            CVaRinMC1[portN, q][tr - 1] = #...
            CVaRinMC2[portN, q][tr - 1] = #...
            CVaRinN1[portN, q][tr - 1] =  #...
            CVaRinN2[portN, q][tr - 1] =  #...

# Display VaR and CVaR

for portN in range(2):
    print('\nPortfolio {}:\n'.format(portN + 1))
    for q in range(alphas.size):
        alf = alphas[q]
        print('Out-of-sample: VaR %4.1f%% = $%6.2f, CVaR %4.1f%% = $%6.2f' % (
        100 * alf, VaRout[portN, q], 100 * alf, CVaRout[portN, q]))
        print('In-sample MC1: VaR %4.1f%% = $%6.2f, CVaR %4.1f%% = $%6.2f' % (
        100 * alf, np.mean(VaRinMC1[portN, q]), 100 * alf, np.mean(CVaRinMC1[portN, q])))
        print('In-sample MC2: VaR %4.1f%% = $%6.2f, CVaR %4.1f%% = $%6.2f' % (
        100 * alf, np.mean(VaRinMC2[portN, q]), 100 * alf, np.mean(CVaRinMC2[portN, q])))
        print('In-sample No: VaR %4.1f%% = $%6.2f, CVaR %4.1f%% = $%6.2f' % (
        100 * alf, VaRinN[portN, q], 100 * alf, CVaRinN[portN, q]))
        print('In-sample N1: VaR %4.1f%% = $%6.2f, CVaR %4.1f%% = $%6.2f' % (
        100 * alf, np.mean(VaRinN1[portN, q]), 100 * alf, np.mean(CVaRinN1[portN, q])))
        print('In-sample N2: VaR %4.1f%% = $%6.2f, CVaR %4.1f%% = $%6.2f\n' % (
        100 * alf, np.mean(VaRinN2[portN, q]), 100 * alf, np.mean(CVaRinN2[portN, q])))

# Plot results
# Figure (1):
# -------- Insert your code here -------- #
# Figure (2):
# -------- Insert your code here -------- #
#
```

```
# Build tables for errors
# MC  approximations (16 rows)
df_mc = pd.DataFrame({})
# -------- Insert your code here -------- #
# Normal approximations (16 rows)
df_N = pd.DataFrame({})
# -------- Insert your code here -------- #
```