Rare-event simulation: Assignment 2

Patrick Laub

March 10, 2020

Due: Wednesday, March 24, 1:30 pm

Submission by email, send completed Jupyter notebook.

Consider the portfolio loss incurred from defaults,

$$L(\boldsymbol{X}) = c_1 1\{X_1 > x_1\} + \dots c_n 1\{X_n > x_n\}$$

where the c_i's are the size of the outstanding loan to obliger i, and $1\{X_i > x_i\}$ represents the random indicator for whether or not obliger i will default on their loan.

Here X_i somewhat represents the level of financial strain for obliger i. This reflects the individual ("idiosyncratic") situation for each obliger, but all obligers are equally affected by broad economic swings. We model these separately, so

$$X_i = \sqrt{(1-\rho)}\eta_i + \sqrt{\rho}Z$$

where $\rho \in (-1, 1)$ specifies the $\mathbb{C}\text{orr}(X_i, X_j)$, $Z \sim \mathsf{Normal}(0, 1)$ are shared between all obligers, and $\eta_i \overset{\text{i.i.d.}}{\sim} \mathsf{Normal}(0, 1)$ is the idiosyncratic variable.

This problem is inspired by Section 5 of Chan and Kroese (2011), though I have simplified it a bit. This paper is extremely well written, I'd recommend taking a look for more explanation & context.

The main goal is to use crude Monte Carlo, cross-entropy method, and the improved cross-entropy method (with MCMC samples) to estimate

$$\ell = \mathbb{P}(L(\boldsymbol{X}) > \gamma)\,.$$

We start by importing some packages and defining the constants for our particular problem.

[ ]:
```
import arviz as az
import matplotlib.pyplot as plt
import numpy as np
import numpy.random as rnd
from scipy import stats
from tqdm.notebook import trange, tqdm
%config InlineBackend.figure_format = 'retina'
```

[ ]:
```
n = 10
ρ = 0.5
cs = np.exp(0.2 * np.arange(n))
xs = np.exp(0.2 * np.arange(n))
γ = 0.75 * np.sum(cs)
print("c:", list(cs))
print("x:", list(xs))
print("γ:", γ)
```

I'll supply the code for a (very) crude Monte Carlo run using a small number of R replications.

[ ]:
```
%%time

rng = rnd.default_rng(1)
R = 10**6

ηs = rng.normal(size=(R, n))
Zs = rng.normal(size=(R, 1))
Xs = np.sqrt(1-ρ) * ηs + np.sqrt(ρ) * Zs

defaults = Xs > xs
losses = np.dot(defaults, cs)

ests = losses > γ

ℓHat = ests.mean()
σHat = ests.std()
widthCI = 1.96 * σHat / np.sqrt(R)
print(f"CMC estimate:\t {ℓHat} (+/- {widthCI})")
print(f"CMC low bound:\t {np.maximum(ℓHat-widthCI, 0)}")
print(f"CMC upp bound:\t {ℓHat+widthCI}")
```

Big crude Monte Carlo

1) Run repeated crude Monte Carlo tests, as in the code demonstrations, so that in total you have a combined CMC test with $R = 10^9$ iterations.

[ ]:
```
%%time

rng = rnd.default_rng(2)
R = 10**9

# ADD CODE HERE


print(f"CMC estimate:\t {ℓHat} (+/- {widthCI})")
print(f"CMC low bound:\t {np.maximum(ℓHat-widthCI, 0)}")
print(f"CMC upp bound:\t {ℓHat+widthCI}")
```

Improved cross-entropy method

2) Use MCMC to sample $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$ and Z conditionally on the event that $L(\boldsymbol{X}) > \gamma$ where $\boldsymbol{X} \equiv \boldsymbol{X}(\boldsymbol{\eta}, Z)$. Use the random walk sampler where each jump is an $n+1$ dimensional vector of i.i.d. normal random variables.

Note, if we say $\boldsymbol{X} \equiv \boldsymbol{X}(\boldsymbol{\eta}, Z)$, then the target density in terms of $\boldsymbol{\eta}$ and Z is

$$\pi(\boldsymbol{\eta}, Z) = \frac{1}{\ell} 1\big\{L(\boldsymbol{X}(\boldsymbol{\eta}, Z)) > \gamma\big\}\, \phi(Z) \prod_{i=1}^{n} \phi(\eta_i)$$

where ϕ is the p.d.f. of a standard normal distribution. Since we don't need proportionality constants, we can instead use

$$\pi(\eta, Z) = 1\big\{L(\boldsymbol{X}(\boldsymbol{\eta}, Z)) > \gamma\big\} \exp\{-Z^2\} \prod_{i=1}^{n} \exp\{-\eta_i^2\}$$

[ ]:
```
%%time

rng = rnd.default_rng(3)
R = 10**6

# ADD CODE HERE
```

3) Print out the traceplots for η_1 and Z. Throw away some burn in samples if you decide it is necessary.

[ ]:

[ ]:

3) Calculate the effective sample size (ESS) for your Z samples and make sure that your R is large enough so that this ESS is at least 1000. If it is too small, go back and update the previous cells until this constraint is reached. If this takes a long time, try playing with the scale parameter for the MCMC jumps.

[ ]:

4) Use the arviz `plot_posterior` function to visualise the η_0 samples, again for the η_n samples, and again for the Z samples.

[ ]:

[ ]:

[ ]:

5) Calculate the overall sample mean $\overline{\eta}$ of the $\boldsymbol{\eta}$ samples (i.e. just one number for the mean of the $R \times n$ matrix of samples), and the sample mean $\overline{Z}$ of the Z samples.

[ ]:

6) Run importance sampling with $R = 10^6$ samples where we sample each η_i from a $\mathsf{Normal}(\overline{\eta}, 1)$ distribution and each Z from a $\mathsf{Normal}(\overline{Z}, 1)$ distribution. This is the improved cross-entropy estimate. Print the result and the confidence interval.

[ ]:
```
%%time

rng = rnd.default_rng(5)
R = 10**6

# ADD CODE HERE


print(f"ICE estimate:\t {ℓHat} (+/- {widthCI})")
print(f"ICE low bound:\t {ℓHat-widthCI}")
print(f"ICE upp bound:\t {ℓHat+widthCI}")
```

Cross entropy

The problem above is particularly hard for the traditional CE method (this is probably why the authors chose it to compare their 'improved' version against it). Let's consider the same problem except the loss will instead be

$$L(\boldsymbol{X}) = X_1 1\{X_1 > x_1\} + \ldots X_n 1\{X_n > x_n\}\,.$$

Here are the constants we'll use for this question.

[ ]:
```
n = 3
ρ = 0.5
xs = np.exp(0.15 * np.arange(n))
γ = 1.5 * np.sum(xs)
print("x:", list(xs))
print("γ:", γ)
```

7) Use the original cross-entropy algorithm to find a good proposal distribution. Look inside the family of distributions where $\eta_n \sim \mathsf{Normal}(\overline{\eta}_n, 1)$ (note, the other η_i's are unchanged) and where $Z \sim \mathsf{Normal}(\overline{Z}, 1)$.

[ ]:
```
%%time

rng = rnd.default_rng(6)
R = 10**6

maxIter = 20
ρ = 0.05
```

```
v = (0, 0)

for iterNum in range(maxIter):
    print(v)

    # ADD CODE HERE
```

8) Run importance sampling with this proposal to get the cross-entropy estimate.

[ ]:
```
%%time
rng = rnd.default_rng(1234)
R = 10**6

# ADD CODE HERE

print(f"CE estimate:\t {ℓHatCE} (+/- {widthCICE})")
print(f"CE low bound:\t {ℓHatCE-widthCICE}")
print(f"CE upp bound:\t {ℓHatCE+widthCICE}")
```