Efficient, Consistent and Flexible Credit Default Simulation with TRNG and RcppParallel

Riccardo Porreca
Roland Schmid
Integrated market and default risk model

- Portfolio of defaultable securities issued by a set of counterparties
- State of the credit environment driving the default of counterparty \( j \) in \([1,J]\)

\[
Y_j = \beta_j Z_j + \sigma_j \varepsilon_j, \text{ i.i.d. } \varepsilon_j \sim N(0, 1);
\]

systemic specific component

\( Y_j = \phi_j(1 + r_j); \)

market value

\[
D_j = \begin{cases} 
1, & Y_j < \Phi^{-1}(P^D_j) \\
0, & Y_j \geq \Phi^{-1}(P^D_j) 
\end{cases}
\]

default indicator

\[
V_j = V_j^0 (1 + r_j);
\]

market value

\[
L_j = V_j^0 - [(1 - D_j)V_j + D_j R_j]
\]

integrated model loss
Integrated market and default risk model

- Portfolio of defaultable securities issued by a set of counterparties

- State of the credit environment driving the default of counterparty $j$ in $[1,J]$

$$Y_j = \beta_j Z_j + \sigma_j \varepsilon_j , \text{ i.i.d. } \varepsilon_j \sim N(0,1);$$

systemic component (specific component component)

$$V_j = V_j^0 (1 + r_j);$$

market value

$$D_j = \begin{cases} 1, & Y_j < \Phi^{-1}(P_j^D) \\ 0, & Y_j \geq \Phi^{-1}(P_j^D) \end{cases}$$

default indicator

$$L_j = V_j^0 - [(1 - D_j)V_j + D_j R_j]$$

integrated model loss

- Assumption: $M$ scenarios of the state of the world are available

$$\{Z_j^{(m)}, r_j^{(m)}\}_{m=1}^M$$

- Simulation problem: for each scenario $m$ in $[1,M]$, generate $K$ samples of $\varepsilon_j$ to obtain $M*K$ realizations of the credit environment return $Y_j$

  - combined simulation size $M*K$ high enough to capture the rare nature of default events
Efficient, consistent, flexible MC simulation

Exact reproducibility of full-simulation results (fair playing)

- Parallel execution (block splitting)

- Sub-portfolio simulation

- Insight for given scenarios of interest

- Limitation: (Pseudo)Random Number Generators (RNGs) are intrinsically sequential

\[ \epsilon_j \] idiosyncratic returns
Efficient, consistent, flexible MC simulation

Exact reproducibility of full-simulation results (*fair playing*)

- **Parallel execution** (*block splitting*)
  - jump to the beginning of a given chunk of simulations
- **Sub-portfolio simulation**
  - split and simulate only the relevant counterparties
- **Insight for given scenarios of interest**
  - jump to individual simulations
- **Limitation**: (Pseudo)Random Number Generators (RNGs) are intrinsically sequential
- **TRNG**: “*state of the art C++ pseudo-random number generator library for sequential and parallel Monte Carlo simulations*”
  - RNGs with powerful *jump* and *split* capabilities

![Diagram](image)
Efficient, consistent, flexible MC simulation

Exact reproducibility of full-simulation results (*fair playing*)

- **Parallel execution** (*block splitting*)
  - jump to the beginning of a given chunk of simulations
- **Sub-portfolio simulation**
  - split and simulate only the relevant counterparties
- **Insight for given scenarios of interest**
  - jump to individual simulations
- **Limitation**: (Pseudo)Random Number Generators (RNGs) are intrinsically sequential
- **TRNG**: “*state of the art C++ pseudo-random number generator library for sequential and parallel Monte Carlo simulations*”
  - RNGs with powerful jump and split capabilities

\begin{itemize}
  \item J counterparties
  \item M*K simulations
  \item idiosyncratic returns $\varepsilon_j$
\end{itemize}
Efficient, consistent, flexible MC simulation

Exact reproducibility of full-simulation results (*fair playing*)

- **Parallel execution** (*block splitting*)
  - jump to the beginning of a given chunk of simulations
- **Sub-portfolio** simulation
  - split and simulate only the relevant counterparties
- **Insight** for given scenarios of interest
  - jump to individual simulations
- **Limitation**: (Pseudo)Random Number Generators (RNGs) are intrinsically sequential
- **TRNG**: "*state of the art C++ pseudo-random number generator library for sequential and parallel Monte Carlo simulations*" [H. Bauke, http://numbercrunch.de/trng]
  - RNGs with powerful jump and split capabilities
Efficient, consistent, flexible MC simulation

Exact reproducibility of full-simulation results (*fair playing*)

- **Parallel execution** (*block splitting*)
  - jump to the beginning of a given chunk of simulations

- **Sub-portfolio simulation**
  - split and simulate only the relevant counterparties

- **Insight for given scenarios of interest**
  - jump to individual simulations

- **Limitation**: (Pseudo)Random Number Generators (RNGs) are intrinsically sequential

- **TRNG**: “*state of the art C++ pseudo-random number generator library for sequential and parallel Monte Carlo simulations*”
  - [H. Bauke, http://numbercrunch.de/trng](http://numbercrunch.de/trng)
  - RNGs with powerful *jump* and *split* capabilities

- Available to the R community via *rTRNG* package
  - `install_github("miraisolutions/rTRNG")`
R/cpp simulation kernel and data  
https://github.com/miraisolutions/PortfolioRiskMC

\[
Y_j = \beta_j Z_j + \sqrt{1 - \beta_j^2} \varepsilon_j \\
V_j = V_j^0 (1 + r_j) \\
D_j = \begin{cases} 
1, & Y_j < \Phi^{-1}(P_j^D) \\
0, & Y_j \geq \Phi^{-1}(P_j^D) 
\end{cases} \\
L_j = V_j^0 - [(1 - D_j)V_j + D_j R_j]
\]

```R
simulationKernel(pf, Z, r, 
J, 
K, mk = seq_len(K * nrow(Z)), 
agg = factor(rep("PF", nrow(pf))), 
seed)
```
# R(cpp) simulation kernel and data

https://github.com/miraisolutions/PortfolioRiskMC

<table>
<thead>
<tr>
<th>j</th>
<th>V0</th>
<th>R</th>
<th>beta</th>
<th>PD</th>
<th>rtng</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>32606970</td>
<td>910000</td>
<td>0.707</td>
<td>BBB</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>8932752</td>
<td>92800</td>
<td>0.707</td>
<td>A</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>564931</td>
<td>335675</td>
<td>0.707</td>
<td>CCC</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3494502</td>
<td>82000</td>
<td>0.707</td>
<td>B</td>
<td>.</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6679886</td>
<td>1000500</td>
<td>0.707</td>
<td>BB</td>
<td>.</td>
<td></td>
</tr>
</tbody>
</table>

**Usage**

```
simulationKernel(pf, Z, r, J, K, mk = seq_len(K * nrow(Z)),
agg = factor(rep("PF", nrow(pf))), seed)
```

**Value**

<table>
<thead>
<tr>
<th>mk</th>
<th>agg</th>
<th>1</th>
<th>.</th>
<th>.</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2942986</td>
<td>2144142</td>
<td>2551616</td>
<td>2128115</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>3269994</td>
<td>1886979</td>
<td>2881280</td>
<td>1447524</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>3874471</td>
<td>1711273</td>
<td>2544507</td>
<td>3696855</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>2809653</td>
<td>3757694</td>
<td>1816909</td>
<td>3071337</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>4100697</td>
<td>2123775</td>
<td>2544716</td>
<td>1878057</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>2106241</td>
<td>4758459</td>
<td>4014828</td>
<td>3573142</td>
<td></td>
</tr>
<tr>
<td>M*K</td>
<td>2045032</td>
<td>2990315</td>
<td>4636829</td>
<td>2588612</td>
<td></td>
</tr>
</tbody>
</table>

**total nr. of counterparties**

```
1  0.5241 -0.484 -0.402 -0.774 -0.702
. -2.2608 -0.666 -1.003  0.423  0.683
. -0.0197 -0.174 -0.178 -0.607 -0.858
.  0.1831 -1.011 -0.488  0.209  0.368
. -0.3614  0.740  0.928 -0.777 -1.430
```

**aggregation criterion**

```
K, mk = seq_len(K * nrow(Z)),
```

**initial RNG state**

```
agg = factor(rep("PF", nrow(pf))), seed)
```

**simulations of interest**
Examples and results
https://github.com/miraisolutions/PortfolioRiskMC

- **Full simulation** (multi-threaded): $J = 6’000$, $M = 10’000$, $K = 100$
  - Aggregation criterion: rating (credit quality)

$$L_{rtng} \leftarrow \text{simulationKernel} (pf, Z, r, J, K, \text{agg} = pf(rtng), \text{seed} = s)$$

- **ES99**: average loss in the 1% worst scenarios

$$\text{ES99}_{rtng} \leftarrow \text{ES99} (L_{rtng})$$

```
##  Builder   AA     AAA     A     BB ...
## 9878920788 8620051762 7468245838 4796596354 1190520347 ...
```

- Consistent simulation for the sub-portfolio with BBB rating

$$L_{BBB} \leftarrow \text{simulationKernel} (pf \%>\% \text{filter}(rtng == "BBB"), Z, r, J, K, \text{seed} = s)$$

```r
all.equal(c(L_{BBB}), L_{rtng}[ , "BBB"], \text{check.attributes} = \text{FALSE})
## [1] TRUE
ES99_{BBB} \leftarrow \text{ES99} (L_{BBB})
## 9878920788
```
Examples and results
https://github.com/miraisolutions/PortfolioRiskMC

- **Risk insight** for BBB

- **Contribution** of individual counterparties to the BBB total **ES99**

```r
pfBBB <- pf %>% filter(rtg == "BBB")
L_jBBBtail <- simulationKernel(pfBBB, Z, r, J, K, agg = pfBBB$j,
                           mk = tail99(L_BBB), seed = s)
ContrES99_jBBB <- colMeans(L_jBBBtail)
all.equal(sum(ContrES99_jBBB), ES99_BBB, check.attributes = FALSE)
## [1] TRUE
```

- Focus on the **top 3 counterparties** (highest contribution)

```r
pftop3BBB <- pfBBB %>% filter(j %in% top3jBBB)
L_top3BBB <- simulationKernel(pftop3BBB, Z, r, J, K,
                              agg = pftop3BBB$j, seed = s)
ES99_top3BBB <- ES99(L_top3BBB)
## j VO R ES99 ContrES99 Div Contr/V0
## 2 9444041000 278250000 4720889738 4277383374 0.9060545 0.4529188
## 8 3260697000 91000000 1672364832 999636420 0.5977382 0.3065714
## 70 298111300 13483307 963482756 436598485 0.4531461 1.4645486
```
Examples and results
https://github.com/miraisolutions/PortfolioRiskMC

• What-if scenario

• Top 3 BBB counterparties downgraded => PD from 0.0025 to 0.01

```r
pfBBBwi <- pf %>% filter(rtg == "BBB") %>%
  mutate(PD = replace(PD, j %in% top3jBBB, 0.01))
pftop3BBBwi <- pfBBBwi %>% filter(j %in% top3jBBB)
L_top3BBBwi <- simulationKernel(pftop3BBBwi, Z, r, J, K,
  agg = pftop3BBBwi$j, seed = s)
```

• Effect on the BBB total

```r
L_BBBwi <- L_BBB + (rowSums(L_top3BBBwi) - rowSums(L_top3BBB))
ES99_BBBwi <- ES99(L_BBBwi)
##  ES99_BBBwi   ES99_BBB
##     11943875892 9878920788
```

• New contribution for the full BBB sub-portfolio

```r
L_jBBBtailwi <- simulationKernel(pfBBBwi, Z, r, J, K, agg = pfBBBwi$j,
  mk = tail99(L_BBBwi), seed = s)
```

All this achieved without re-simulating the whole BBB portfolio!
Summary

- Monte Carlo simulation of an integrated market and default risk model
  - Flexible, consistent, slim, multi-purpose simulation kernel
- TRNG state-of-the-art parallel random number generators
  - rTRNG for prototyping in R and broader usage in R/C++ projects
- Flexible and fast ad-hoc assessments on sub-portfolios, simulations of interest, what-if scenarios
- Incremental simulations and updates possible
- Can also be used for driver or change analysis, isolating away the MC variability

=> Achieve fast re-simulation instead of
  - storing GBs or TBs of granular results
  - using complex analytic approximation models that are hard to explain and understand
Appendix
Integrated market and default risk model

- Portfolio of **defaultable securities** issued by a set of **counterparties**

- Model market risk in correlation with credit default risk using an **integrated** approach
  - Market and default risk are intrinsically related
  - Dependency must be properly taken into account

- Simplifying assumptions
  - Default occurrence determined at **counterparty** level
  - Exactly **one security** per each counterparty in the portfolio
  - We ignore **non-defaultable** securities subject to market risk only
Appendix
Integrated market and default risk model

- State of the credit environment driving the default of counterparty \(j\) in \([1,J]\)
  \[
  Y_j = \beta_j Z_j + \sigma_j \varepsilon_j, \text{ i.i.d. } \varepsilon_j \sim N(0, 1)
  \]
  systemic component \quad specific component
  (reflects the state of the world) \quad (idiosyncratic return)

- Return \(r_j\) drives the market value at horizon (based on the state of the world)
  \[
  V_j = V_j^0 (1 + r_j)
  \]

- Default indicator
  \[
  D_j = \begin{cases} 1, & Y_j < \theta_j \\ 0, & Y_j \geq \theta_j \end{cases}, \quad \theta_j : P(Y_j < \theta_j) = P_j^D
  \]

- Loss (including occurrence of defaults)
  \[
  L_j = V_j^0 - [(1 - D_j)V_j + D_j R_j], \quad 0 \leq R_j \leq V_j^0
  \]

- Default events \(D_j\) and losses \(L_j\) inherit the correlation structure of \(r_j\) and \(Z_j\)
  with other counterparties
Appendix
Integrated market and default risk model

• **Assumption**: M scenarios of the state of the world are available
  \[ \{Z^{(m)}_j, r^{(m)}_j\}_{m=1}^M \]

• Monte Carlo realizations from a given market risk model, which we extend by the occurrence of defaults

• we also assume (WLOG): \( Z_j \sim N(0, 1), \sigma_j = \sqrt{1 - \beta_j^2} \Rightarrow \theta_j = \Phi^{-1}(P^D_j) \)

\[
Y_j = \beta_j Z_j + \sqrt{1 - \beta_j^2} \varepsilon_j \\
V_j = V^0_j (1 + r_j) \\
D_j = \begin{cases} 
1, & Y_j < \Phi^{-1}(P^D_j) \\
0, & Y_j \geq \Phi^{-1}(P^D_j) 
\end{cases} \\
L_j = V^0_j - [(1 - D_j)V_j + D_j R_j]
\]
Appendix

Integrated market and default risk model

• **Assumption**: M scenarios of the state of the world are available
  \[ \left\{ Z_j^{(m)}, r_j^{(m)} \right\}_{m=1}^M \]

• Monte Carlo realizations from a given market risk model, which we extend by the occurrence of defaults

• we also assume (WLOG): \( Z_j \sim N(0, 1), \ \sigma_j = \sqrt{1 - \beta_j^2} \Rightarrow \theta_j = \Phi^{-1}(P_j^D) \)

• Monte Carlo approach for simulating the integrated model:
  • combine \( V_j \) and \( Z_j \) for the available scenarios with independent realizations of the idiosyncratic returns \( \varepsilon_j \)
  • for each scenario \( m \) in \([1, M]\), generate \( K \) samples of \( \varepsilon_j \) to obtain \( M*K \) realizations of the credit environment return \( Y_j \)
  • combined simulation size \( M*K \) high enough to capture the rare nature of default events
microbenchmark results \((M = 1000, K = 10)\)

number of parallel threads

![Graph showing microbenchmark results for different numbers of parallel threads. The x-axis represents the number of parallel threads (1, 2, 4), and the y-axis represents time in seconds. The graph shows the distribution of time taken for different thread counts.]
Appendix
Performance benchmark

microbenchmark results ($M = 1000$, $K = 10$)

size of the sub-portfolio

number of sub-simulations
Contact

Riccardo Porreca
Senior Solutions Consultant
E-Mail: riccardo.porreca@mirai-solutions.com
Mobile: +41 (0)76 786 10 28

Roland Schmid
Partner, Applications Lead
E-Mail: roland.schmid@mirai-solutions.com
Mobile: +41 (0)79 478 31 82

Mirai Solutions GmbH
Tödistrasse 48
CH-8002 Zurich
Switzerland

info@mirai-solutions.com
www.mirai-solutions.com