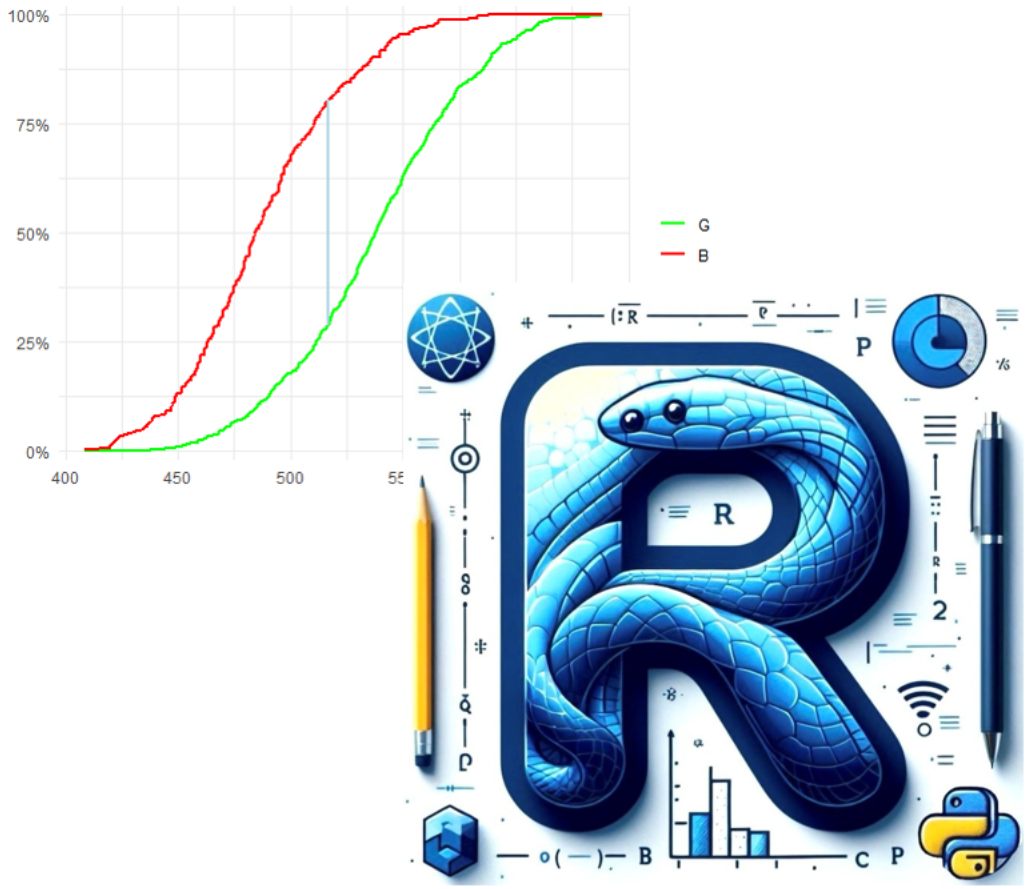
Credit Risk Modeling Working Notes



A Collection of Presentations, Experiments, and Technical Papers

Andrija Djurovic

Contents

Preface

These Working Notes compile material from the author's GitHub repository, accessible at the following link. The notes and the GitHub repository are dedicated to two books - Applied Data Science for Credit Risk and Probability of Default Rating Modeling with R - covering different credit risk modeling topics. Their main goal is to extend the content of the books to specific tasks and challenges practitioners face in their day-to-day work.

The motivation behind writing these books and creating the repository stems from the observed gap between academic literature, industry practices, and the evolving landscape of data science. While the literature on credit risk modeling has grown significantly, discrepancies persist. The evolution of data science has brought considerable automation to many processes but has also introduced the risk of overreliance on pre-programmed procedures, sometimes resulting in the misuse of statistical methods. Additionally, many practitioners entering the field of credit risk modeling often overlook fundamental principles, which hinders their professional development.

These Working Notes aim to serve as a centralized hub for continuous education and consolidating essential concepts in credit risk modeling.

Like the idea behind the GitHub repository, the Working Notes will be regularly updated as new material becomes available.

As the Working Notes compile various types of materials - from presentations to technical papers - if specific links are not directly accessible from the book, readers are encouraged to consult the corresponding document on the GitHub repository.

About the author

Andrija Djurovic is a credit risk professional with over ten years of experience in credit risk modeling. His expertise encompasses modeling Probability of Default, Loss Given Default, Exposure At Default, the development of scoring models, macroeconomic modeling, and portfolio analysis. With comprehensive statistical knowledge spanning academia to industry, his proficiency extends to crafting tailored analytics applications. Notably, Andrija is the author and developer of essential `R` (`monobin`, `monobinShiny`, `PDtoolkit`, `LGDtoolkit`) and `Python` (`monobinpy`) packages tailored for credit risk modeling.
Andrija is also the author of the books Applied Data Science for Credit Risk and Probability of Default Rating Modeling with R.
To learn more, visit his LinkedIn profile at www.linkedin.com/in/andrija-djurovic, github page at https://github.com/andrija-djurovic, or connect directly through email at djandrija@gmail.com.

Citing this book

Please follow the citation format provided below to cite this book for your references, research, or academic work.
When citing within your text, use the author's last name along with the publication year, like this:

```
(Djurovic, 2025).
```

In your bibliography or reference list, use the following citation style:

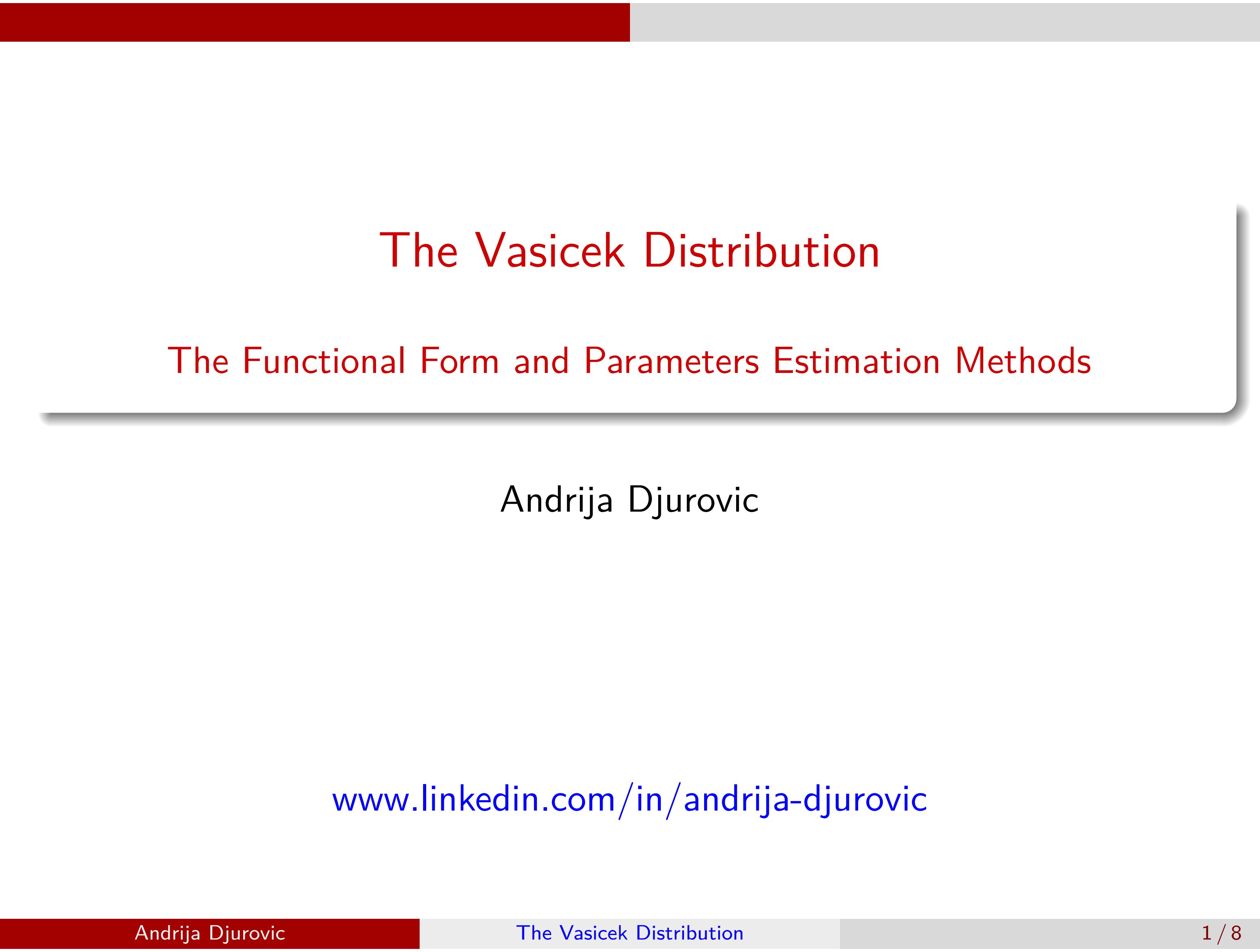
```
Djurovic, Andrija. (2025). Credit Risk Modeling Working Notes:
A Collection of Presentations, Experiments, and Technical Papers.
https://leanpub.com/crmwn
```

Or use the following BibTeX entry:

```
@book{djuroviccrmwn,
 title = {Credit Risk Modeling Working Notes},
 author = {Andrija Djurovic},
 publisher = {\url{https://leanpub.com/crmwn}},
 year = {2025},
 subtitle = {A Collection of Presentations, Experiments, and Technical Papers}
}
```

1 The Vasicek Distribution (Probability of Default Models)

1.1 The Functional Form and Parameters Estimation Methods



The Vasicek Distribution

The Functional Form and Parameters Estimation Methods

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | The Vasicek Distribution | 1 / 8

The Functional Form and Parameters

The Vasicek distribution is a two-parameter ($0 < p < 1$ and $0 < \rho < 1$) continuous distribution on the range 0 to 1. If a variable x has a Vasicek distribution, then x can be represented as:

$$x = \phi\left(\frac{\phi^{-1}(p) - \sqrt{\rho}z}{\sqrt{1-\rho}}\right)$$

where:

- p and ρ are the parameters of the distribution, commonly referred to as the average default rate and asset correlation, respectively;
- z represents the systemic factor drawn from the standard normal distribution; and
- ϕ and ϕ^{-1} denote the distribution and quantile function of the standard normal distribution, respectively.

Distribution, Density, and Quantile Functions

Cumulative distribution function:

$$F_{p,\rho}(x) = \phi\left(\frac{\sqrt{1-\rho}\,\phi^{-1}(x) - \phi^{-1}(p)}{\sqrt{\rho}}\right)$$

Probability density function:

$$f_{p,\rho}(x) = \sqrt{\frac{1-\rho}{\rho}}\, e^{\frac{1}{2}\left(\phi^{-1}(x)^2 - \left(\frac{\sqrt{1-\rho}\phi^{-1}(x)-\phi^{-1}(p)}{\sqrt{\rho}}\right)^2\right)}$$

Quantile function:

$$F^{-1}_{p,\rho}(\alpha) = \phi\left(\frac{\phi^{-1}(p) + \sqrt{\rho}\,\phi^{-1}(\alpha)}{\sqrt{1-\rho}}\right)$$

The Parameters Estimation Methods

The parameters of the Vasicek distribution can be estimated using one of the following methods:

1. Direct Moment Matching
2. Indirect Moment Matching
3. Maximizing the Log-Likelihood of the Vasicek Probability Density Function
4. Quantile-Based Estimation

Direct Moment Matching

$$\hat{p} = \frac{\sum_{i=1}^{T} x_i}{T}$$

$$\hat{\sigma}_x^2 = \phi_2(y_1 \leq \phi^{-1}(\hat{p}), y_2 \leq \phi^{-1}(\hat{p}), \rho) - \hat{p}^2$$

where:

- T denotes the number of observations;
- x_i represents the observed default rates;
- $\hat{\sigma}_x^2$ is the variance of the observed default rates;
- ϕ_2 denotes the bivariate standard normal cumulative distribution function with $\mu_{y_i} = 0$ and $\sigma_{y_i} = 1$; and
- ϕ^{-1} stands for the quantile function of the standard normal distribution.

The asset correlation parameter ρ is estimated based on a numerical root-finding procedure.

Indirect Moment Matching

$$\hat{p} = \phi\left(\frac{\hat{\mu}_x}{\sqrt{1+\hat{\sigma}_x^2}}\right)$$

$$\hat{\rho} = \frac{\hat{\sigma}_x^2}{1+\hat{\sigma}_x^2}$$

where:

- $\hat{\mu}_x$ is defined as $\hat{\mu}_x = \frac{\sum_{i=1}^{T} \phi^{-1}(x_i)}{T}$ and ϕ^{-1} denotes the quantile function of the standard normal distribution; and
- $\hat{\sigma}_x^2$ is defined as $\hat{\sigma}_x^2 = \frac{\sum_{i=1}^{T} (\phi^{-1}(x_i) - \hat{\mu}_x)^2}{T-1}$ with ϕ^{-1} being the quantile function of the standard normal distribution.

This method is also known as the analytical solution for the Maximum Likelihood Estimator.

Maximizing the Log-Likelihood of the Vasicek Probability Density Function

The Log-Likelihood of the Vasicek Probability Density Function is given by:

$$\sum_{i=1}^{T} ln(f_{p,\rho}(x_i))$$

with

$$f_{p,\rho}(x) = \sqrt{\frac{1-\rho}{\rho}}e^{\frac{1}{2}\left(\phi^{-1}(x)^2 - \left(\frac{\sqrt{1-\rho}\phi^{-1}(x) - \phi^{-1}(p)}{\sqrt{\rho}}\right)^2\right)}$$

where:

- T represents the number of observations;
- $f_{p,\rho}(x)$ denotes the Vasicek Probability Density Function;
- x represents the observed default rates;
- p denotes the average default rate; and
- ϕ^{-1} is the quantile function of the standard normal distribution.

With p calculated as $\frac{\sum_{i=1}^{T} x_i}{T}$, ρ is derived by maximizing the Log-Likelihood function based on the observed default rates.

Quantile-Based Estimation

After selecting the probabilities (α_1 and α_2), a system of two equations ($\hat{\mu}_x$ and $\hat{\sigma}_x$) is solved to obtain estimates for p and ρ using the formulas derived from the Indirect Moment Matching method.

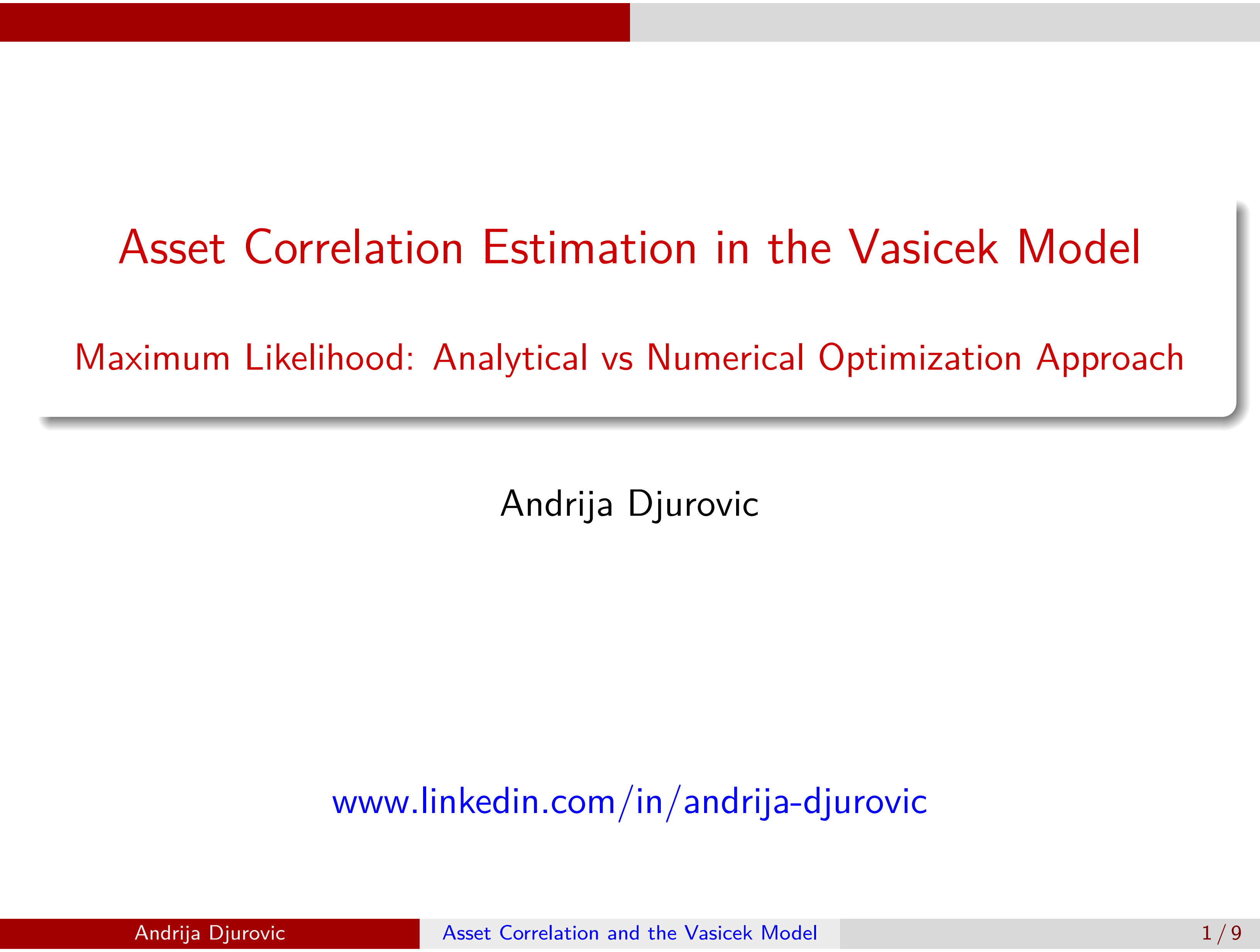
$$\hat{q}(\alpha_1) = \hat{\mu}_x + \hat{\sigma}_x \phi^{-1}(\alpha_1)$$

$$\hat{q}(\alpha_2) = \hat{\mu}_x + \hat{\sigma}_x \phi^{-1}(\alpha_2)$$

where:

- $\hat{q}$ represents the observed quantiles for the selected probabilities α_1 and α_1; and
- ϕ^{-1} is the quantile function of the standard normal distribution.

1.2 Asset Correlation Estimation in the Vasicek Model - Maximum Likelihood: Analytical vs Numerical Optimization Approach



Asset Correlation Estimation in the Vasicek Model

Maximum Likelihood: Analytical vs Numerical Optimization Approach

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

The Functional Form and Parameters of the Vasicek-Distributed Variable

The Vasicek distribution is a two-parameter ($0 < p < 1$ and $0 < \rho < 1$) continuous distribution on the range 0 to 1. If a variable x has a Vasicek distribution, then x can be represented as:

$$x = \phi\left(\frac{\phi^{-1}(p) - \sqrt{\rho}z}{\sqrt{1-\rho}}\right)$$

where:

- p and ρ are the parameters of the distribution, commonly referred to as the average default rate and asset correlation, respectively;
- z represents the systemic factor drawn from the standard normal distribution; and
- ϕ and ϕ^{-1} denote the distribution and quantile function of the standard normal distribution, respectively.

The Parameters Estimation Methods

The parameters of the Vasicek distribution can be estimated using one of the following methods:

1. Direct Moment Matching
2. Indirect Moment Matching
3. Maximizing the Log-Likelihood of the Vasicek Probability Density Function
4. Quantile-Based Estimation

While each method produces nearly unbiased estimators for parameter p, this is not the case with parameter ρ.

The most frequently used method for estimating ρ in practice is Indirect Moment Matching. This method is also known as the analytical solution for the Maximum Likelihood Estimator, but how does it compare with the method of maximizing the Log-Likelihood of the Vasicek Probability Density Function?

The following slides provide a brief description of these two approaches along with simulation results for different sample sizes and selected elements of the variable with the Vasicek distribution.

Indirect Moment Matching

$$\hat{p} = \phi\left(\frac{\hat{\mu}_x}{\sqrt{1+\hat{\sigma}_x^2}}\right)$$

$$\hat{\rho} = \frac{\hat{\sigma}_x^2}{1+\hat{\sigma}_x^2}$$

where:

- $\hat{\mu}_x$ is defined as $\hat{\mu}_x = \frac{\sum_{i=1}^{T}\phi^{-1}(x_i)}{T}$ and ϕ^{-1} denotes the quantile function of the standard normal distribution; and
- $\hat{\sigma}_x^2$ is defined as $\hat{\sigma}_x^2 = \frac{\sum_{i=1}^{T}(\phi^{-1}(x_i)-\hat{\mu}_x)^2}{T-1}$ with ϕ^{-1} being the quantile function of the standard normal distribution.

Maximizing the Log-Likelihood of the Vasicek Probability Density Function

The Log-Likelihood of the Vasicek Probability Density Function is given by:

$$\sum_{i=1}^{T} ln(f_{p,\rho}(x_i))$$

with

$$f_{p,\rho}(x) = \sqrt{\frac{1-\rho}{\rho}} e^{\frac{1}{2}\left(\phi^{-1}(x)^2 - \left(\frac{\sqrt{1-\rho}\phi^{-1}(x) - \phi^{-1}(p)}{\sqrt{\rho}}\right)^2\right)}$$

where:

- T represents the number of observations;
- $f_{p,\rho}(x)$ denotes the Vasicek Probability Density Function;
- x represents the observed default rates;
- p denotes the average default rate; and
- ϕ^{-1} is the quantile function of the standard normal distribution.

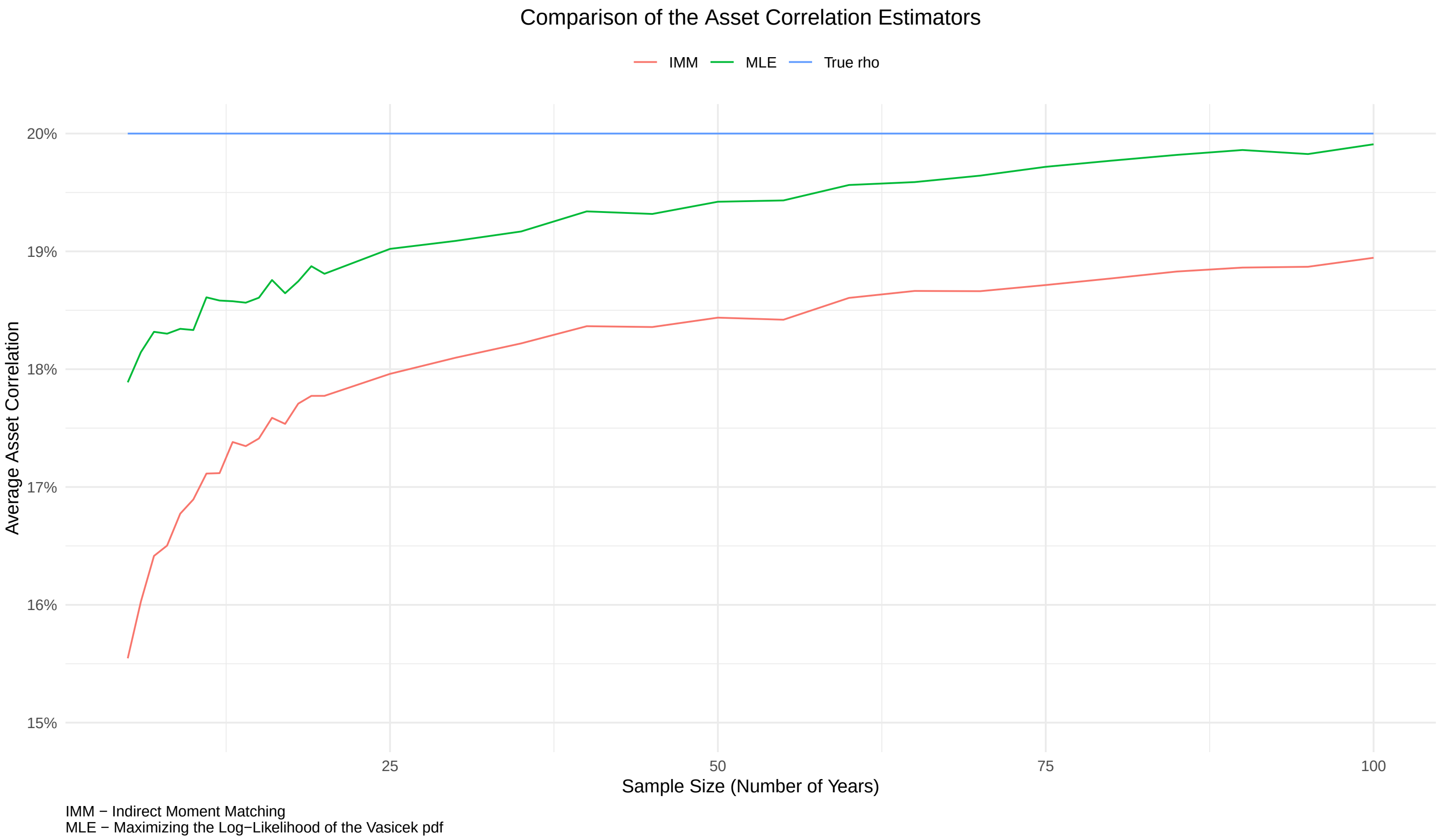
With p calculated as $\frac{\sum_{i=1}^{T} x_i}{T}$, ρ is derived by maximizing the Log-Likelihood function based on the observed default rates.

Simulation Setup

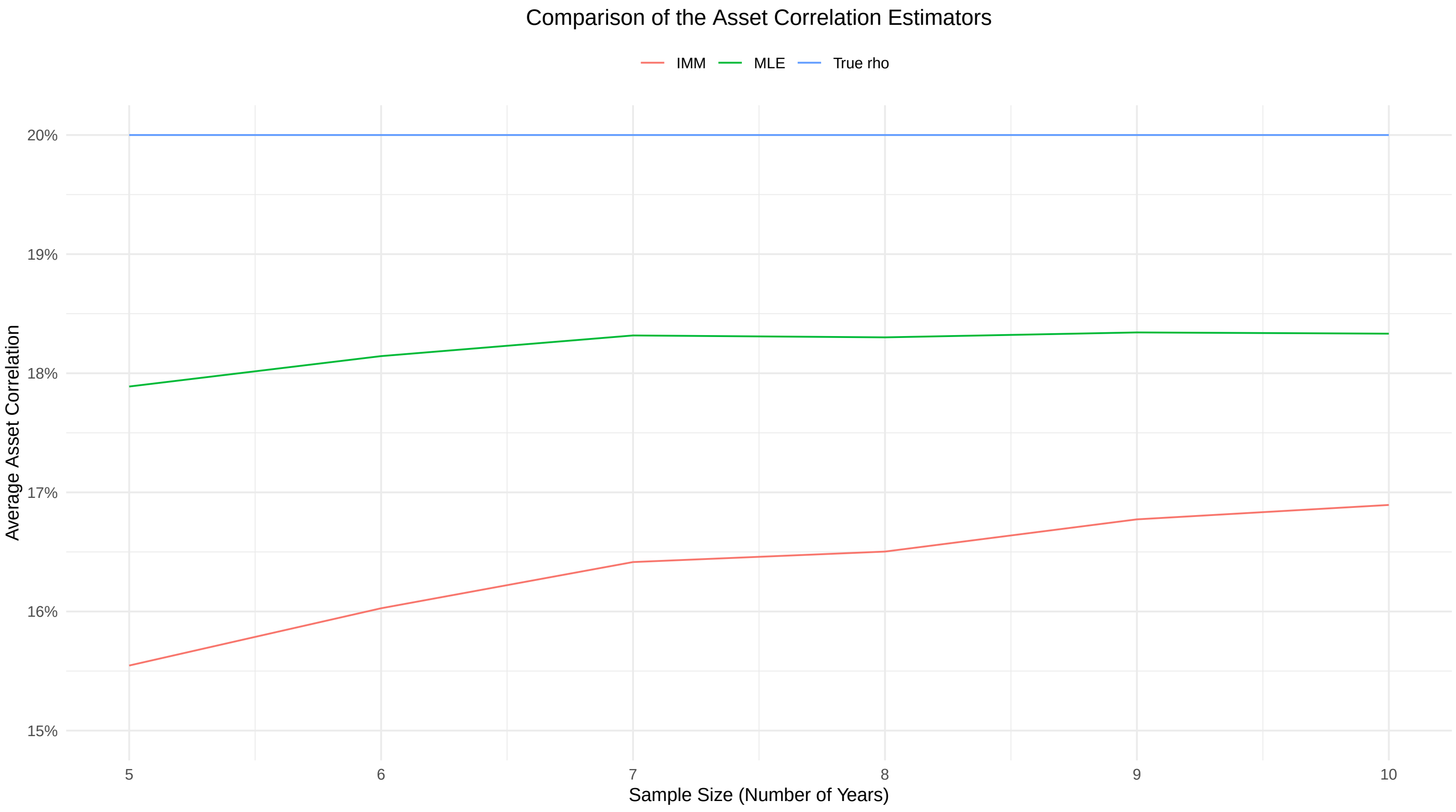
1. Select the inputs for simulating random data from the Vasicek distribution: sample size ($T = 5$), unconditional PD ($pd = 0.05$), asset correlation ($\rho = 0.20$), time-dependent systemic factor (z) from the standard normal distribution with an autoregression coefficient $\theta = 0.30$.
2. Using the simulated data, estimate the asset correlation by employing Indirect Moment Matching and maximizing the Log-Likelihood of the Vasicek Probability Density Function.
3. Store the results and repeat step 2 a total of N times ($N = 10,000$).
4. Calculate the average asset correlation for each estimation method, then compare these averages to the true ρ.
5. Change the sample size T from step 1, and repeat steps 2 through 4.

Practitioners are encouraged to test and simulate the bias of the estimators with other parameters of the Vasicek-distributed variable.

Simulation Results

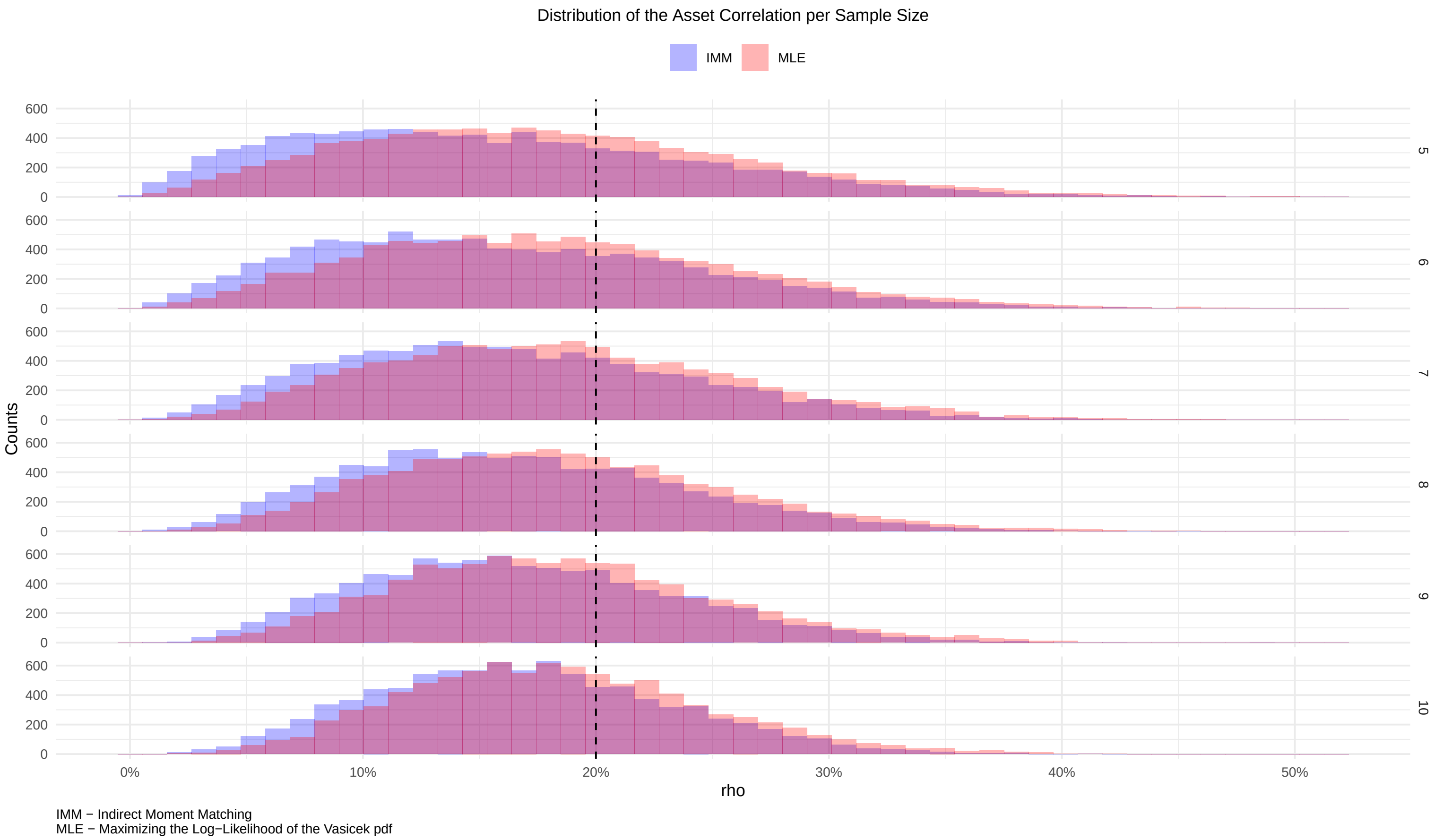


Simulation Results cont.

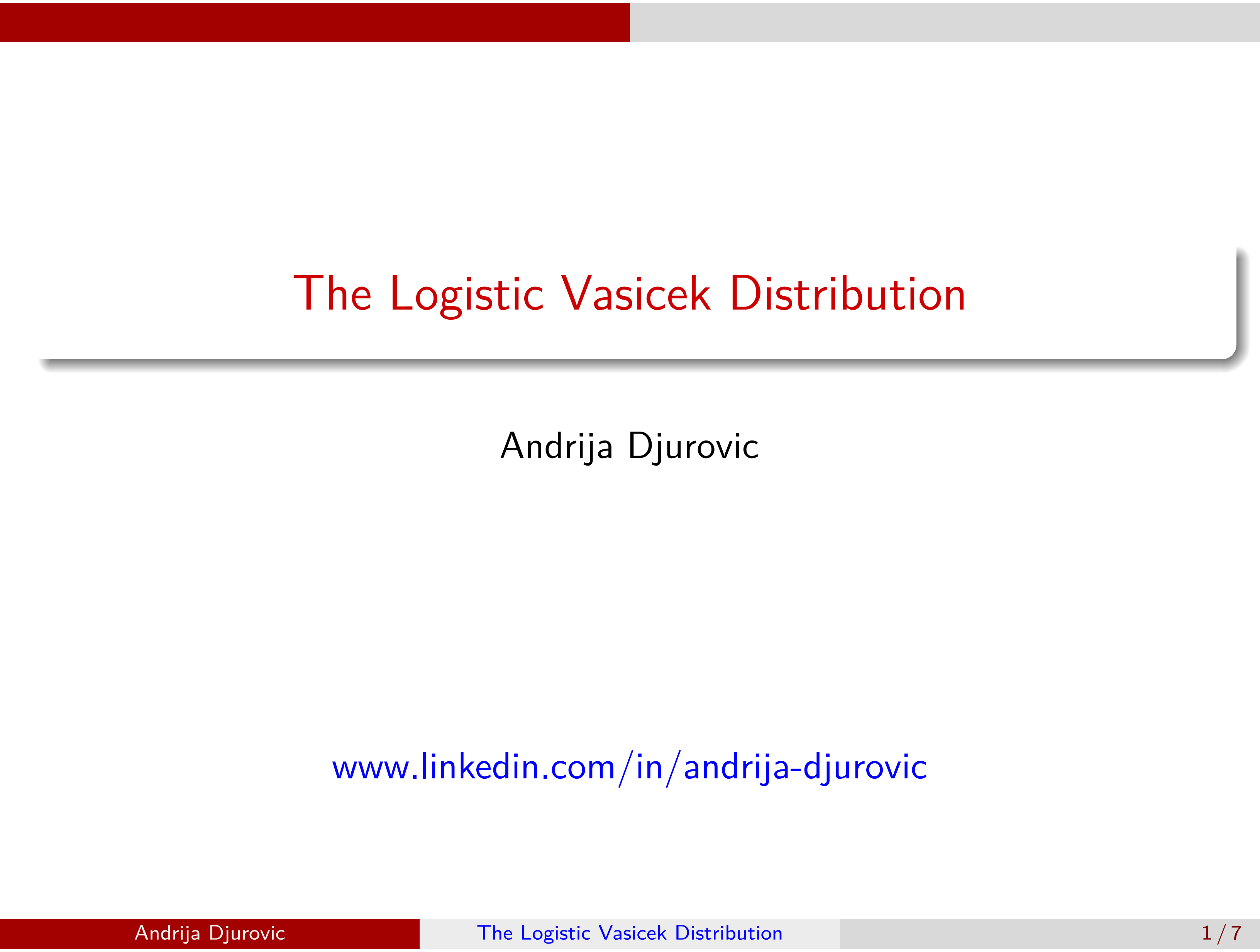


IMM − Indirect Moment Matching
MLE − Maximizing the Log−Likelihood of the Vasicek pdf

Simulation Results cont.



1.3 The Logistic Vasicek Distribution



The Logistic Vasicek Distribution

Andrija Djurovic

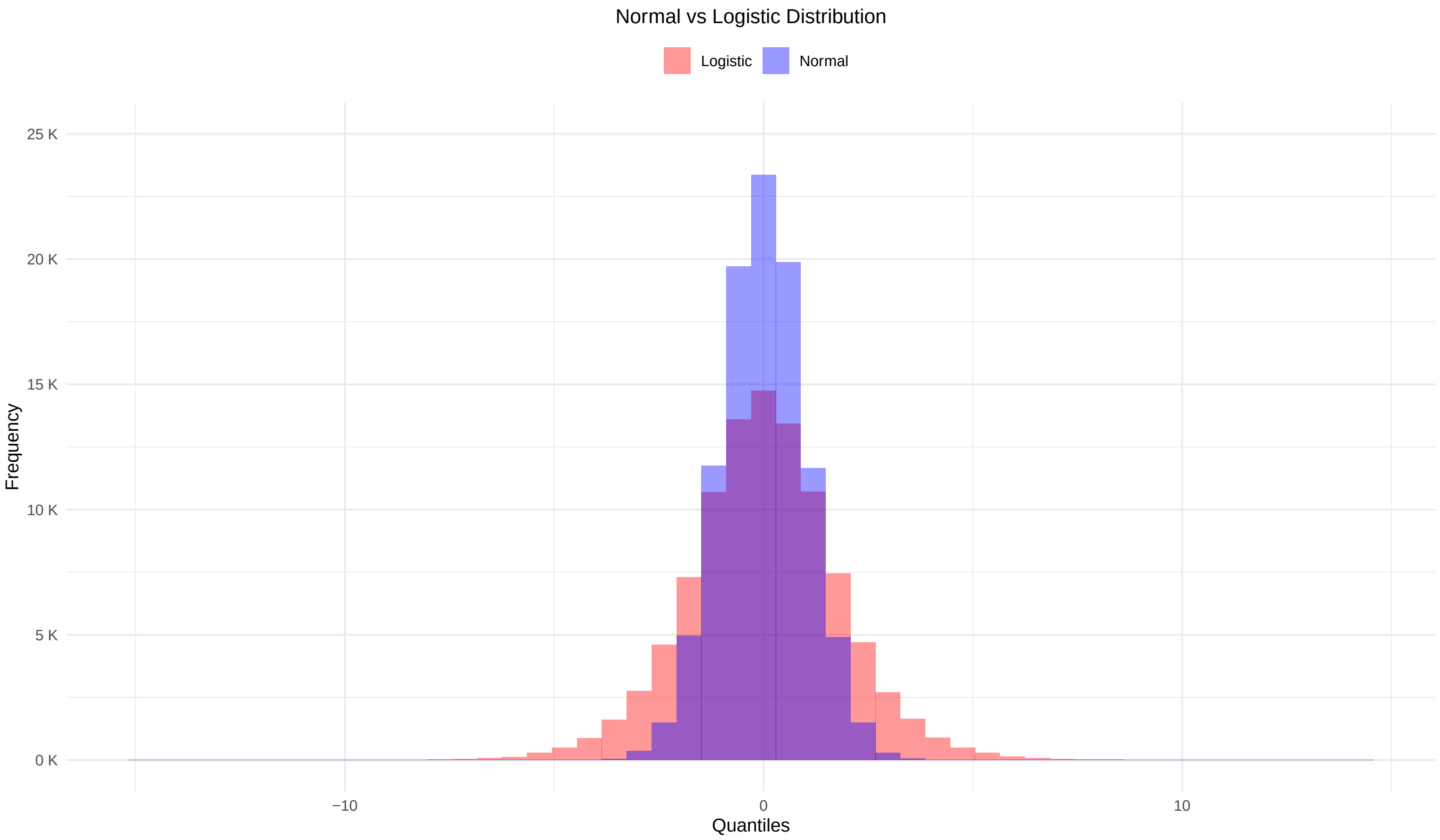
www.linkedin.com/in/andrija-djurovic

Andrija Djurovic The Logistic Vasicek Distribution 1 / 7

Adjusting the Vasicek Model Using the Logistic Distribution

- Witzany suggests replacing the standard normal distribution function with the logistic distribution in the Vasicek model, which could be a natural approach for calculating regulatory capital requirements, aligning with the typical use of logistic regression in Probability of Default (PD) modeling.
- In addition to calculating regulatory capital requirements, the Logistic Vasicek model could be explored for other applications, such as developing IFRS9 ECL challenger models or measuring concentration risk. These areas also often rely heavily on the Vasicek model.
- Another interesting area of investigation would be to examine the bias in estimating the asset correlation parameter under normal and logistic distributions in the Vasicek model for smaller sample sizes.

Normal vs Logistic Distribution



The Logistic Vasicek Distribution

The Logistic Vasicek distribution is derived similarly to the classical Vasicek distribution from the asset return formula:

$$y_i = \sqrt{\rho}z + \sqrt{1-\rho}\epsilon_i$$

with the key difference being that the systemic (z) and idiosyncratic (ϵ) factors are assumed to come from a logistic distribution rather than a normal distribution.

A technical disadvantage of this new assumption is that the sum of two independent logistically distributed variables is not necessarily another logistically distributed variable. However, this approximation can be considered valid for asset correlation (ρ) values up to 30%. For a more detailed comparison, refer to slide 6.

The Logistic Vasicek Distribution cont.

The Logistic Vasicek distribution is still a two-parameter ($0 < p < 1$ and $0 < \rho < 1$) continuous distribution on the range 0 to 1. If a variable x has a Logistic Vasicek distribution, then x can be represented as:

$$x = \lambda\left(\frac{\lambda^{-1}(p) - \sqrt{\rho}z}{\sqrt{1-\rho}}\right)$$

where:

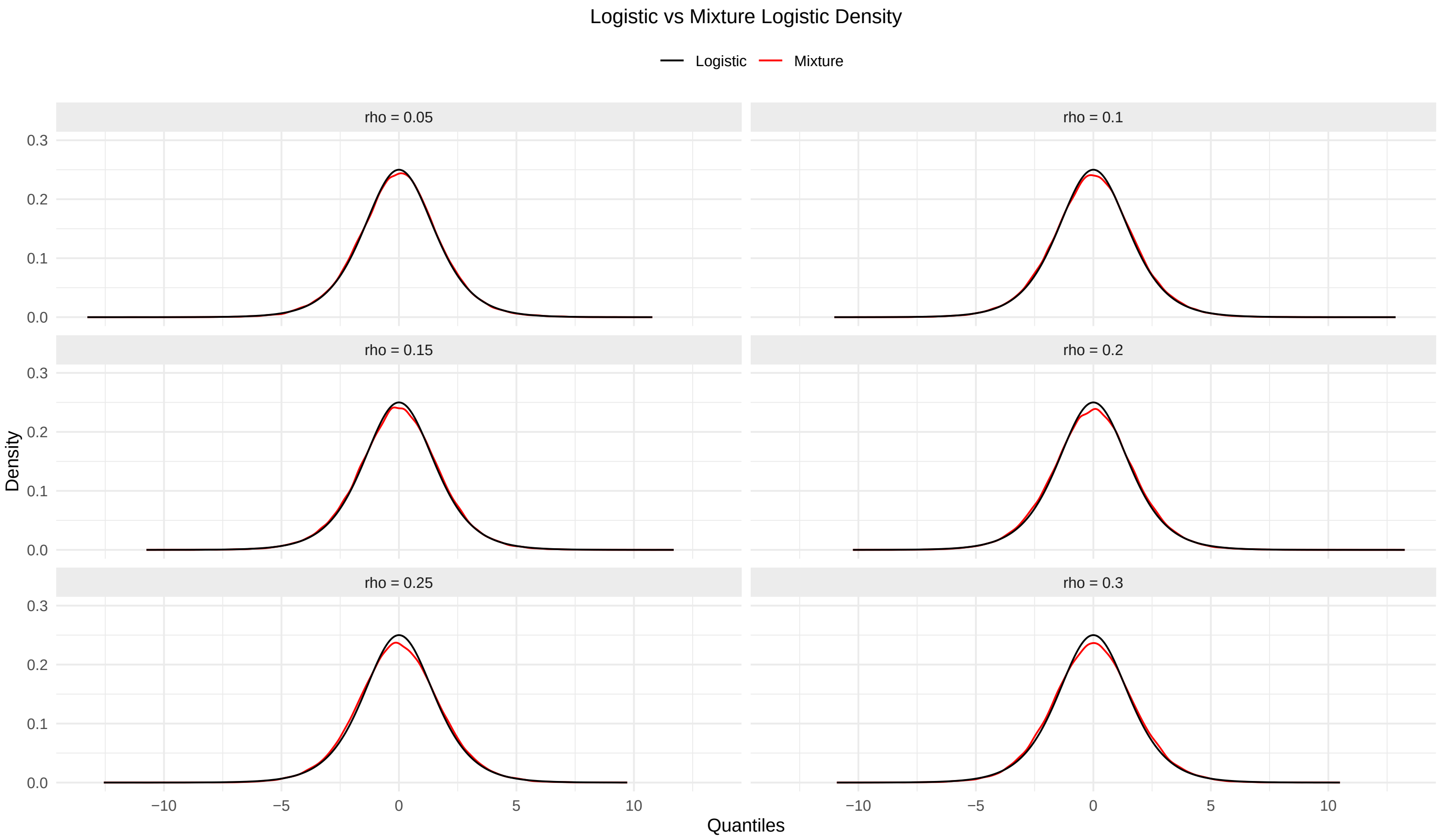
- p and ρ are the parameters of the distribution, commonly referred to as the average default rate and asset correlation, respectively;
- z represents the systemic factor drawn from the logistic distribution with mean 0 and standard deviation of 1; and
- λ and λ^{-1} denote the distribution and quantile function of the logistic distribution with mean 0 and standard deviation of 1, respectively.

The cumulative distribution function is defined as follows:

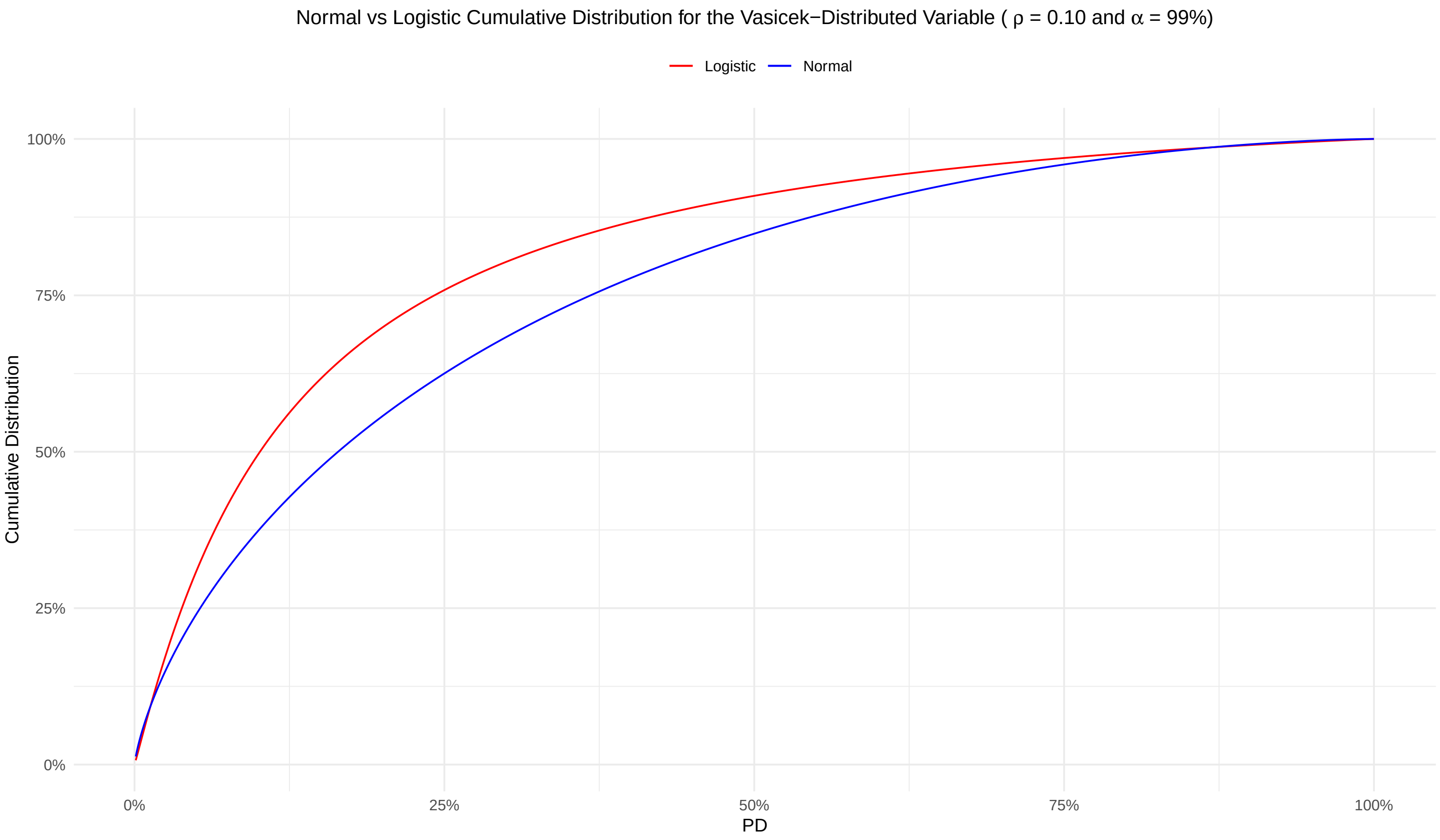
$$F_{p,\rho}(x) = \lambda\left(\frac{\sqrt{1-\rho}\;\lambda^{-1}(x) - \lambda^{-1}(p)}{\sqrt{\rho}}\right)$$

where the probability density function is calculated as the first derivative of the aforementioned cumulative probability function.

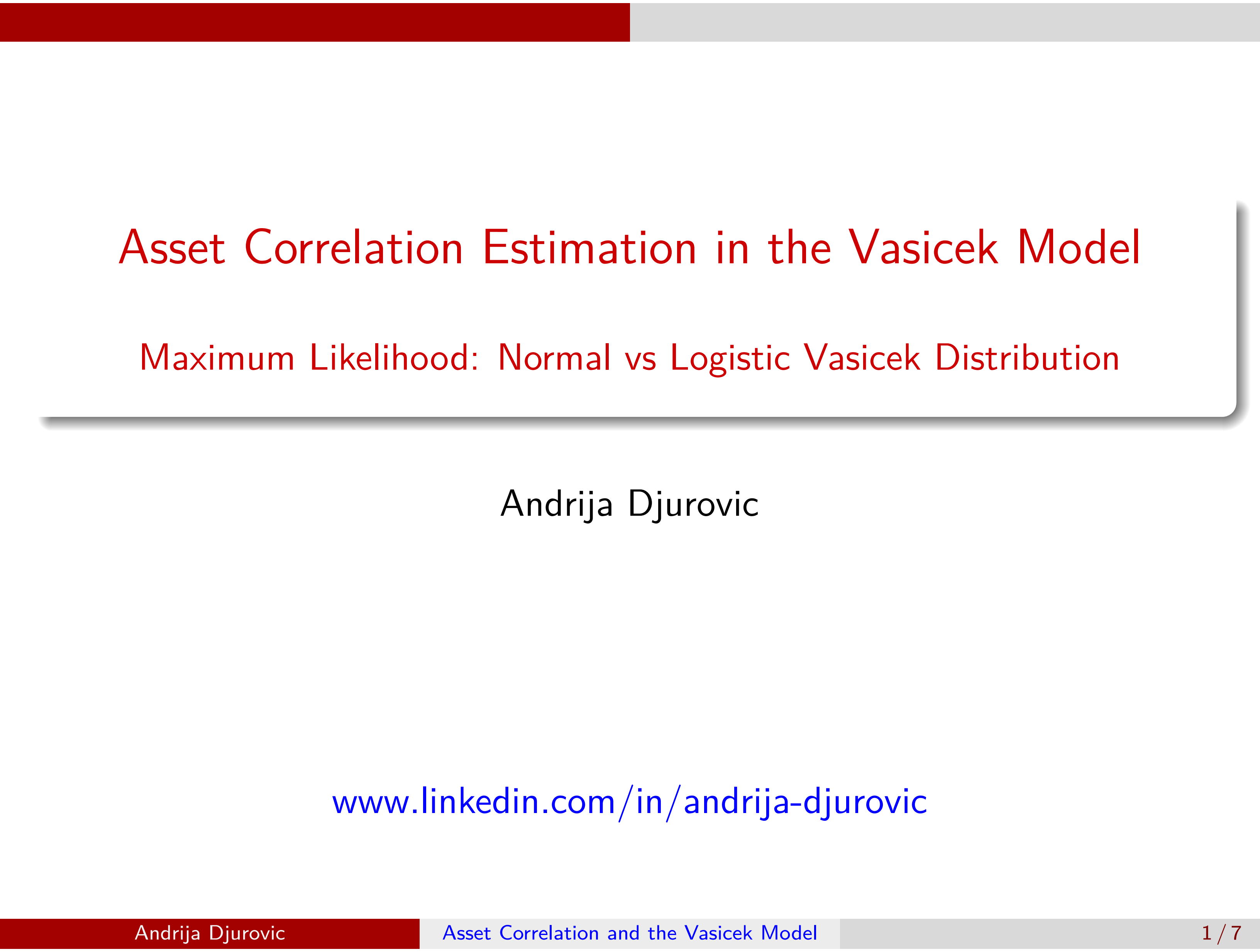
Logistic vs Mixture-Logistic Distribution



Normal vs Logistic Cumulative Distribution of the Vasicek-Distributed Variable



1.4 Asset Correlation Estimation in the Vasicek Model - Maximum Likelihood: Normal vs Logistic Vasicek Distribution



Asset Correlation Estimation in the Vasicek Model

Maximum Likelihood: Normal vs Logistic Vasicek Distribution

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

The Vasicek Distribution

If a variable x has a Normal or Logistic Vasicek distribution, then x can be represented as:

$$x = F\left(\frac{F^{-1}(p) - \sqrt{\rho}z}{\sqrt{1-\rho}}\right)$$

where:

- p and ρ are the parameters of the distribution, commonly referred to as the average default rate and asset correlation, respectively;
- z represents the systemic factor drawn from the standard normal or logistic distribution; and
- F and F^{-1} denote the distribution and quantile function of the standard normal (ϕ and ϕ^{-1}) or logistic (λ and λ^{-1}) distribution, respectively.

Further, the cumulative distribution function can be defined as follows:

$$P(X \leq x) = F\left(\frac{\sqrt{1-\rho}\; F^{-1}(x) - F^{-1}(p)}{\sqrt{\rho}}\right)$$

The Asset Correlation Estimator

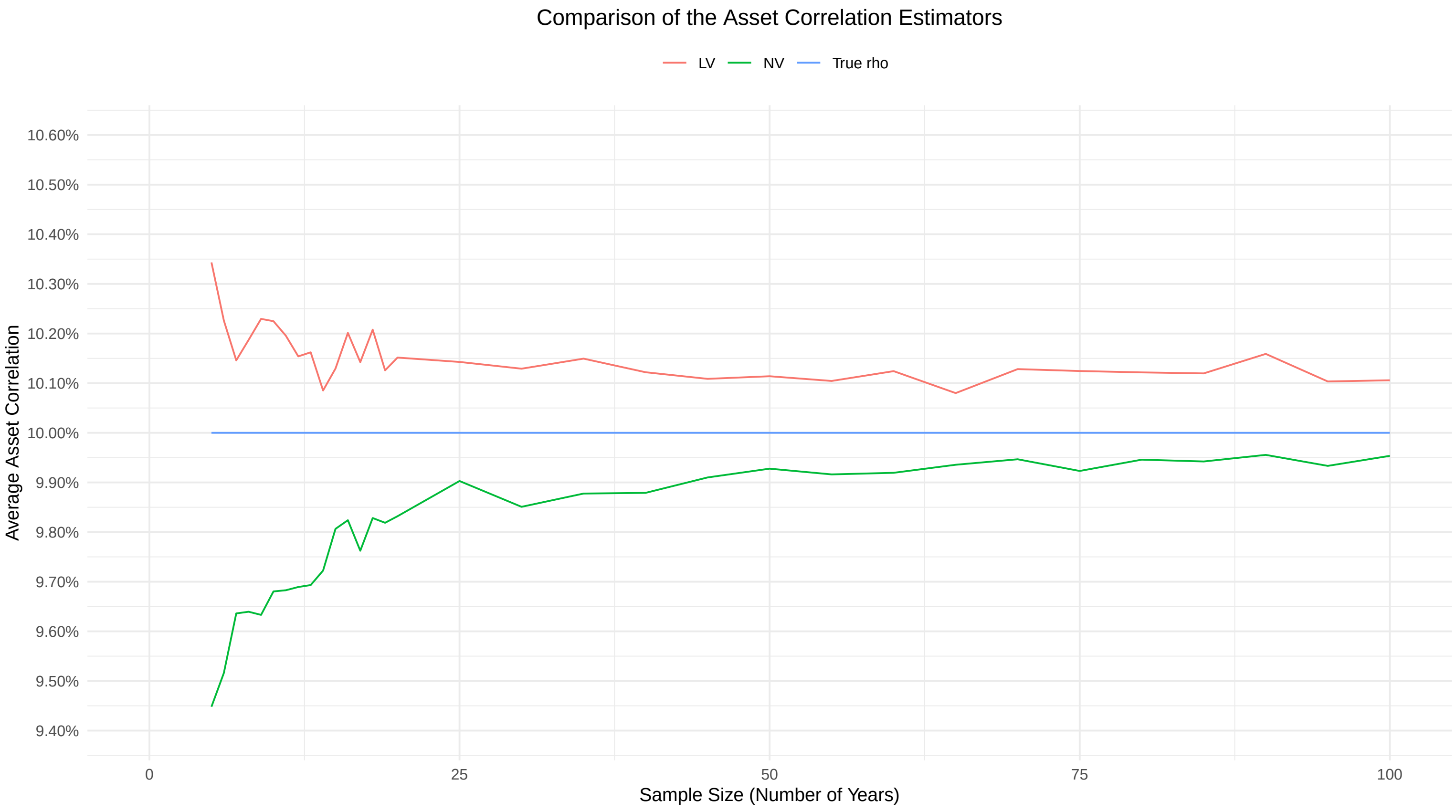
- One of the most commonly used estimation methods for fitting the parameters of the Vasicek distribution to observed data is to maximize the log-likelihood of the probability density function.
- Using the cumulative distribution function presented in slide 2, practitioners can initially derive the probability density function as a derivative of the corresponding cumulative function. Subsequently, they can estimate the asset correlation by maximizing the probability density function's log-likelihood.
- The following slides will utilize this estimation method to compare the bias of the asset correlation estimator between the Normal and Logistic Vasicek models. For both models, the parameter p is estimated as a simple average of the observed default rates.

Simulation Setup

1. Select the inputs for simulating random data from the Vasicek distribution: sample size ($T = 5$), unconditional PD ($pd = 0.05$), asset correlation ($\rho = 0.10$), time-dependent systemic factor (z) from the standard normal and logistic distribution with an autoregression coefficient ($\theta = 0.30$).
2. Using the simulated data, estimate the asset correlation by maximizing the Log-Likelihood of the Vasicek Probability Density Function, considering the different underlying distributions of the Vasicek model (normal and logistic).
3. Store the results and repeat step 2 a total of N times ($N = 10,000$).
4. Calculate the average asset correlation for each estimation model, then compare these averages to the true ρ.
5. Change the sample size T from step 1, and repeat steps 2 through 4.

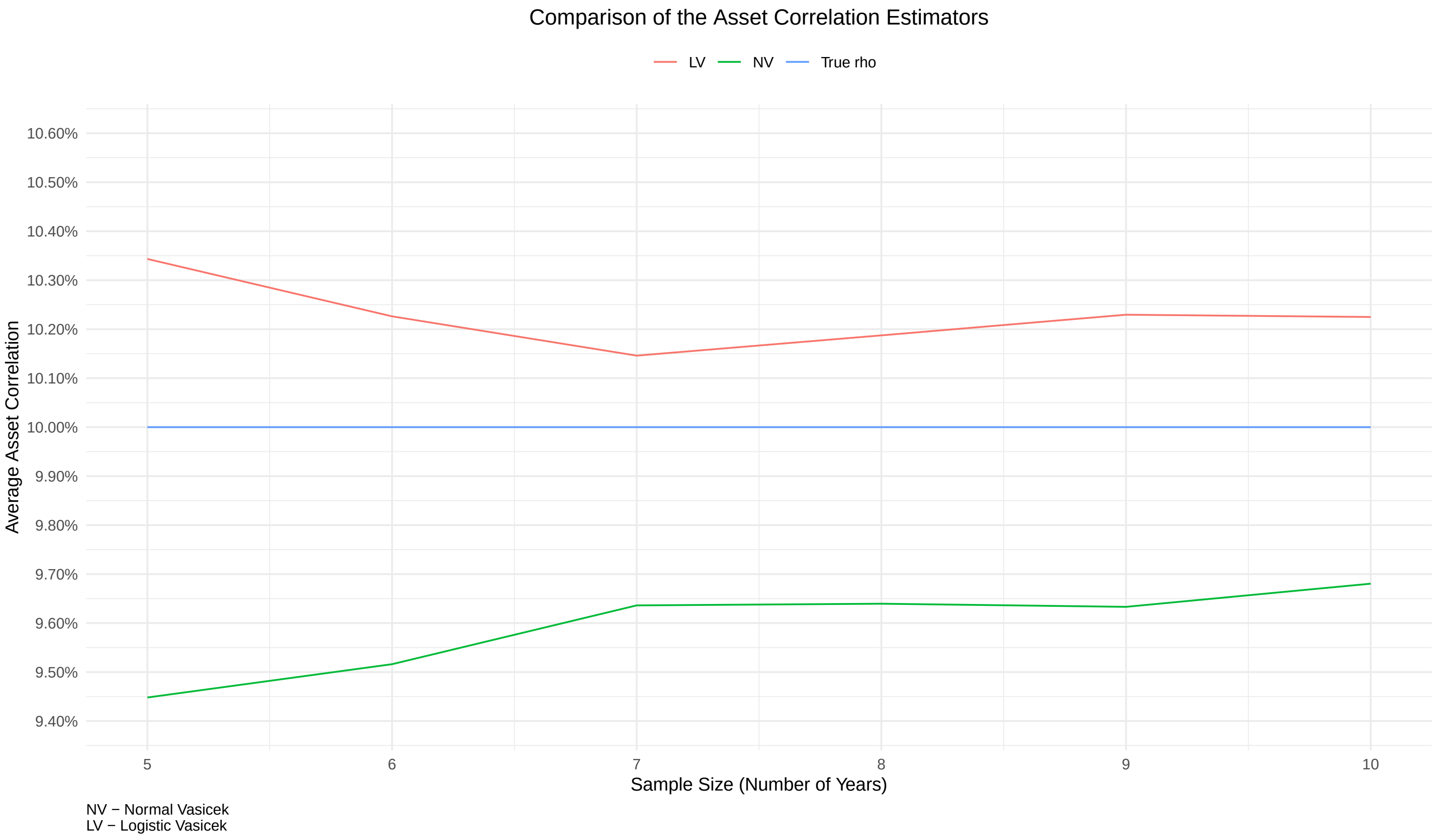
Practitioners are encouraged to test and simulate the bias of the estimators with other parameters of the both Vasicek-distributed variable.

Simulation Results

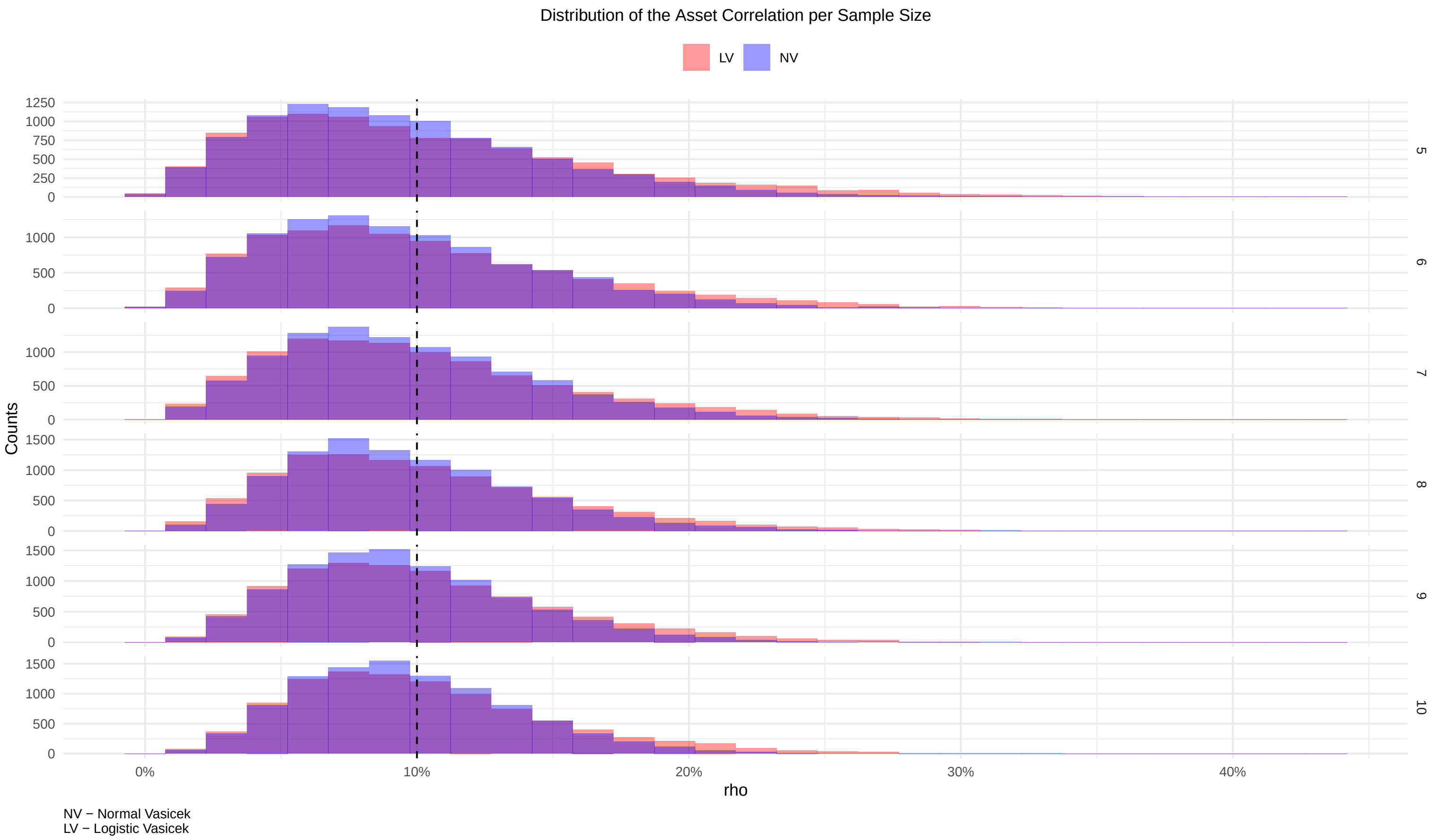


NV − Normal Vasicek
LV − Logistic Vasicek

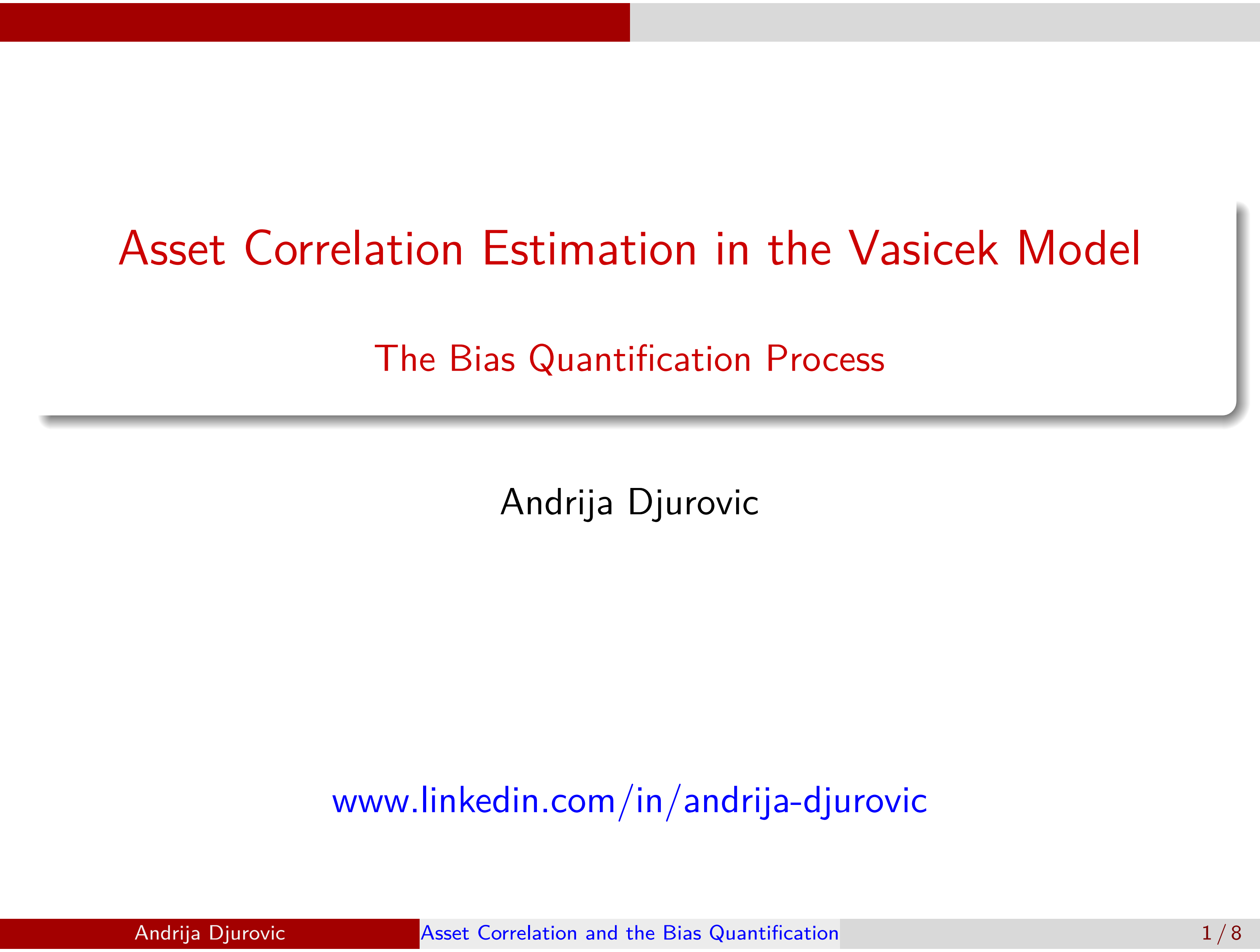
Simulation Results cont.



Simulation Results cont.



1.5 Asset Correlation Estimation in the Vasicek Model - The Bias Quantification Process



Asset Correlation Estimation in the Vasicek Model

The Bias Quantification Process

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic Asset Correlation and the Bias Quantification 1 / 8

The Functional Form and Parameters of the Vasicek-Distributed Variable

The Vasicek distribution is a two-parameter ($0 < p < 1$ and $0 < \rho < 1$) continuous distribution on the range 0 to 1. If a variable x has a Vasicek distribution, then x can be represented as:

$$x = \phi\left(\frac{\phi^{-1}(p) - \sqrt{\rho}z}{\sqrt{1-\rho}}\right)$$

where:

- p and ρ are the parameters of the distribution, commonly referred to as the average default rate and asset correlation, respectively;
- z represents the systemic factor drawn from the standard normal distribution; and
- ϕ and ϕ^{-1} denote the distribution and quantile function of the standard normal distribution, respectively.

The Parameters Estimation Methods

The parameters of the Vasicek distribution can be estimated using one of the following methods:

1. Direct Moment Matching
2. Indirect Moment Matching
3. Maximizing the Log-Likelihood of the Vasicek Probability Density Function
4. Quantile-Based Estimation

While each method produces nearly unbiased estimators for parameter p, this is not the case with parameter ρ.

The following slides detail the process of bias quantification in estimating ρ (asset correlation) using the Indirect Moment Matching (IMM) method. First, we introduce the IMM estimation method, then move on to the steps involved in bias quantification, and conclude with a simulation for a hypothetical portfolios.

Note that although only one estimation method will be presented, the same process applies to others as well.

Indirect Moment Matching

$$\hat{p} = \phi\left(\frac{\hat{\mu}_x}{\sqrt{1+\hat{\sigma}_x^2}}\right)$$

$$\hat{\rho} = \frac{\hat{\sigma}_x^2}{1+\hat{\sigma}_x^2}$$

where:

- $\hat{\mu}_x$ is defined as $\hat{\mu}_x = \frac{\sum_{i=1}^{T} \phi^{-1}(x_i)}{T}$ and ϕ^{-1} denotes the quantile function of the standard normal distribution; and
- $\hat{\sigma}_x^2$ is defined as $\hat{\sigma}_x^2 = \frac{\sum_{i=1}^{T} (\phi^{-1}(x_i) - \hat{\mu}_x)^2}{T-1}$ with ϕ^{-1} being the quantile function of the standard normal distribution.

Bias Quantification Process

The following steps outline the bias quantification process using Monte Carlo simulations:

1. Collect default rate data.
2. Select the parameters estimation method for the Vasicek model.
3. Based on the collected default rates and the selected method, estimate the parameters $p_{observed}$ and $\rho_{observed}$.
4. Given the number of observations (years) of the default rate (T) and the estimated parameter $p_{observed}$, define the data-generating process of the Vasicek model with the true value of the asset correlation parameter ρ_{true}.
5. Based on the N Monte Carlo simulations, optimize the data-generating process for ρ_{true} by minimizing the difference between the average value of the ρ_{true} distribution and the $\rho_{observed}$.

Note that the presented process assumes that the parameters of the Vasicek model estimated based on the observed data represent the average value of the true parameters' distribution.

Simulation Setup and Results

The following table presents the simulation inputs of the hypothetical portfolio:

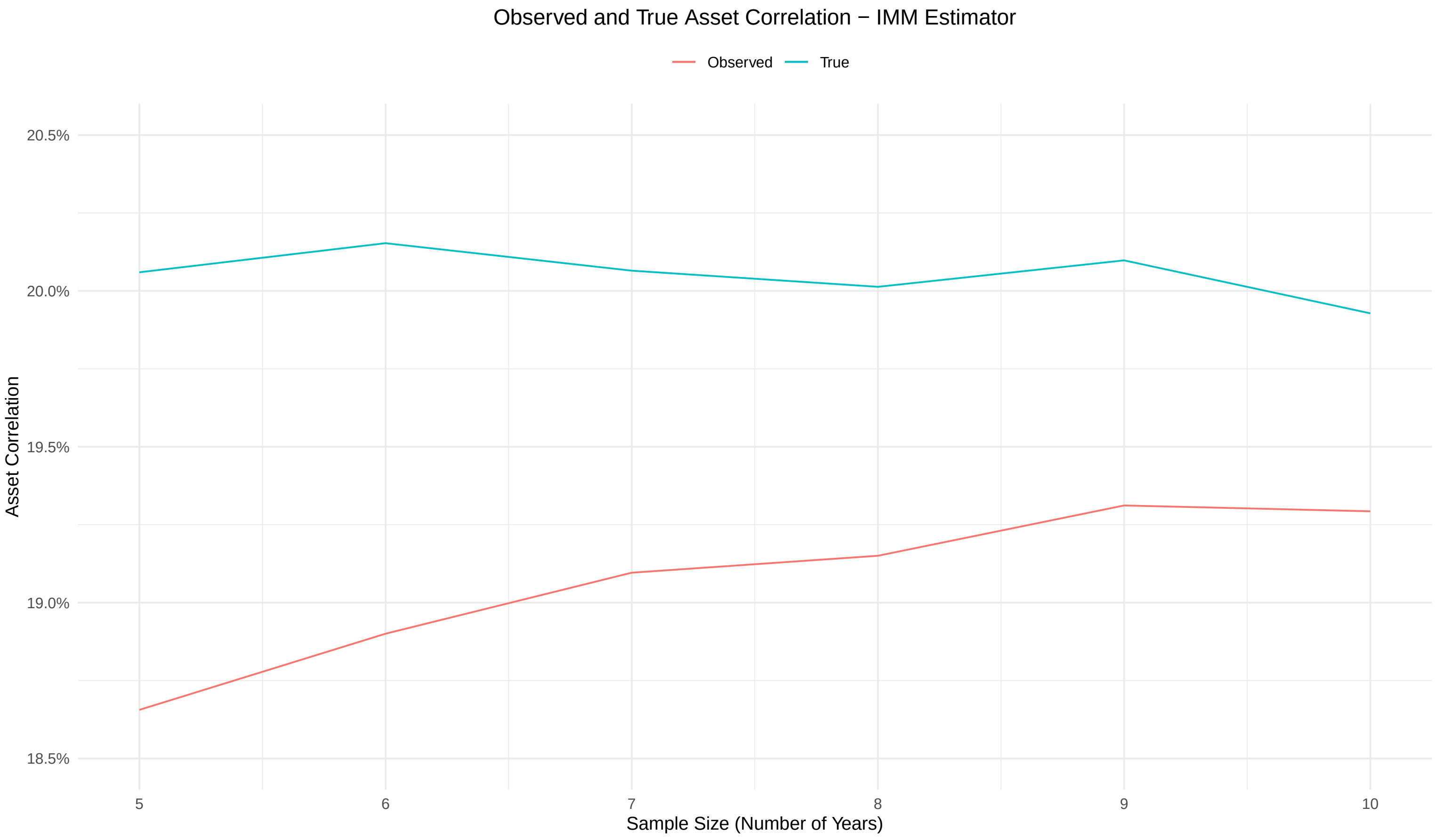
```
##    T p_observed rho_observed
##    5       0.05    0.1865590
##    6       0.05    0.1890025
##    7       0.05    0.1909595
##    8       0.05    0.1915025
##    9       0.05    0.1931158
##   10       0.05    0.1929290
```

Note that the above values are directly obtained from the Vasicek model data-generating process with the value of ρ_{true} being 0.20. Therefore, the expected bias quantification should report values close to the difference between 0.20 and the $\rho_{observed}$ values.

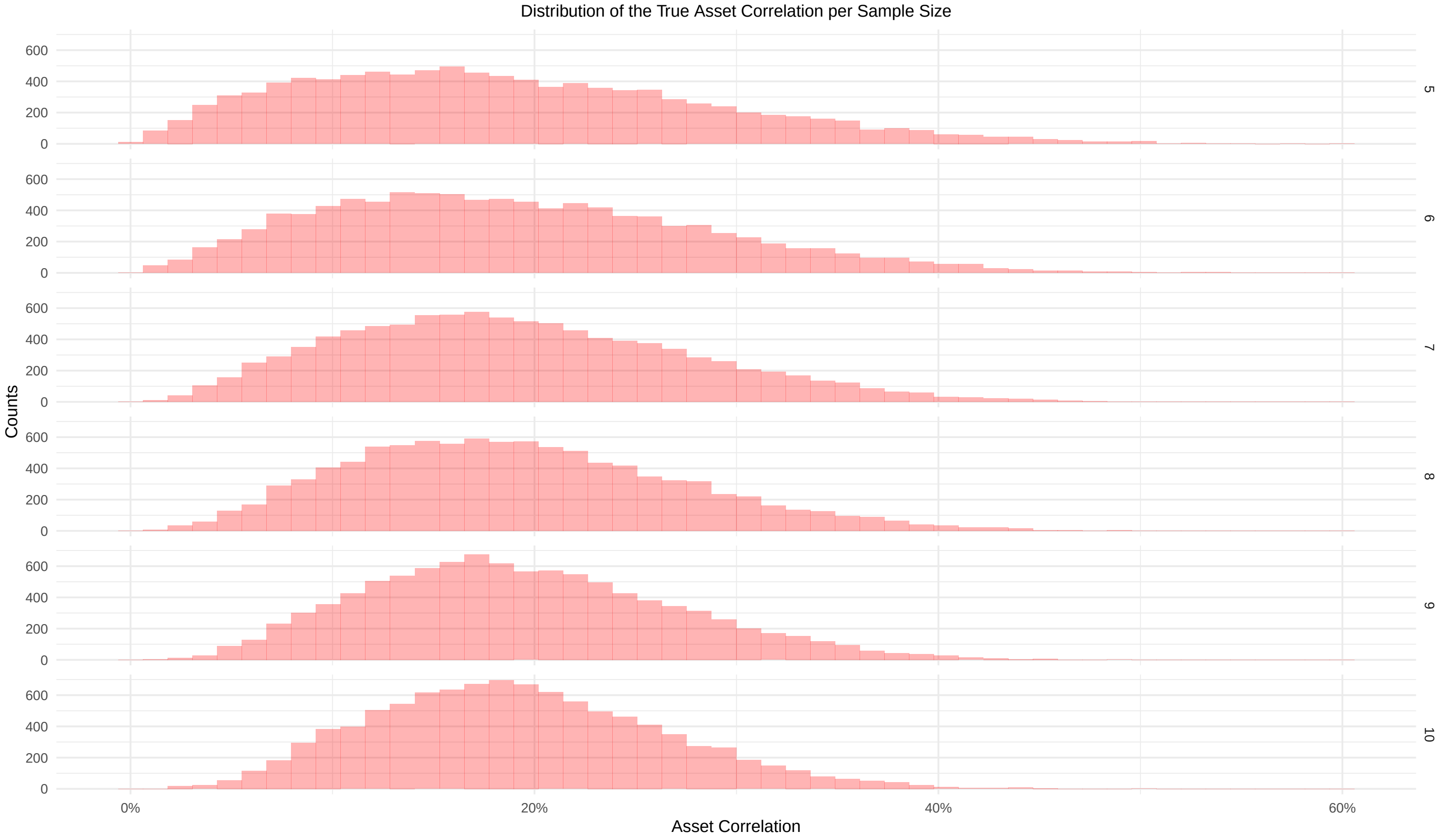
After running the $(N = 10,000)$ Monte Carlo simulations for bias quantification of the asset correlation parameter, the following additional values have been obtained:

```
##    T p_observed rho_observed         bias  rho_true
##    5       0.05    0.1865590 -0.014037978 0.2005970
##    6       0.05    0.1890025 -0.012527848 0.2015303
##    7       0.05    0.1909595 -0.009688797 0.2006483
##    8       0.05    0.1915025 -0.008628467 0.2001309
##    9       0.05    0.1931158 -0.007862726 0.2009786
##   10       0.05    0.1929290 -0.006351165 0.1992802
```

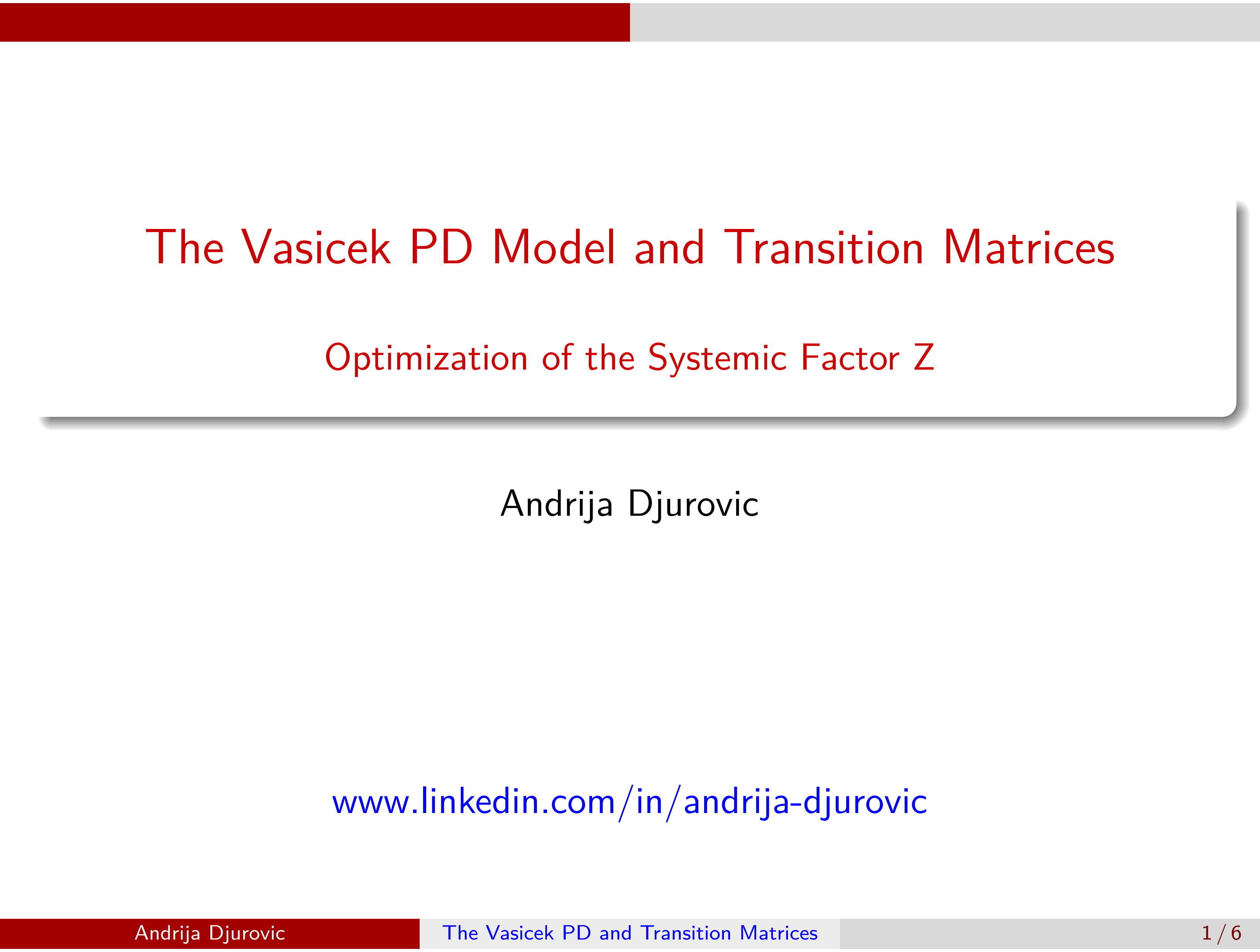
Simulation Setup and Results cont.



Simulation Setup and Results cont.



1.6 The Vasicek PD Model and Transition Matrices - Optimization of the Systemic Factor Z



The Vasicek PD Model and Transition Matrices

Optimization of the Systemic Factor Z

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

The Vasicek PD Model in IFRS9 Modeling

- The Vasicek Probability of Default (PD) model is one of the most commonly used approaches for modeling PD for IFRS9 purposes. Essentially, it relies on the assumption that observed default rates follow the Vasicek distribution.

- Vasicek distribution is a two-parameter ($0 < p < 1$ and $0 < \rho < 1$) continuous distribution on the range 0 to 1. If a variable x has a Vasicek distribution, then x (observed default rate) can be represented as:

$$x = \Phi\left(\frac{\Phi^{-1}(p) - \sqrt{\rho}Z}{\sqrt{1-\rho}}\right)$$

  where:

  - p and ρ are the parameters of the distribution, commonly referred to as the average default rate and asset correlation, respectively;
  - Z represents the systemic factor drawn from the standard normal distribution;
  - Φ and Φ^{-1} denote the distribution and quantile function of the standard normal distribution, respectively.

- Given the parameters p and ρ, which are commonly estimated at the portfolio level, the systemic factor Z is derived by manipulating the equation of the Vasicek distribution.

- But what if we shift the focus to transition matrices instead of deriving Z from the portfolio level?

- The following slides present an approach to deriving the systemic factor Z based on grade transition matrices. This approach is particularly attractive as it aligns with the common practice of applying the Vasicek model's results to grade transition rates in IFRS9 PD modeling.

The Transition Rates and Z Factor Optimization

- Rather than deriving Z at the portfolio level, practitioners can instead derive the Z factor at the transition matrix level by utilizing ρ, as well as Through-the-Cycle (TtC) and Point-in-Time (PiT) Grade-to-grade (G-to-g) transition rates.

- The usual approach is optimization by solving a least-squares problem. One way to define this optimization problem is as follows:

$$\min_{Z_t} \sum_G \sum_g \frac{n_{t,g} \left[P_t(G,g) - \Delta(x_{g+1}^G, x_g^G, Z_t) \right]^2}{\Delta(x_{g+1}^G, x_g^G, Z_t)[1 - \Delta(x_{g+1}^G, x_g^G, Z_t)]}$$

  where:

  - $P_t(G,g)$ is the PiT probability of moving from grade G to grade g between two periods;
  - $\Delta(x_{g+1}^G, x_g^G, Z_t)$ is defined as $\Phi\left(\frac{x_{g+1}^G - \sqrt{\rho}Z_t}{\sqrt{1-\rho}}\right) - \Phi\left(\frac{x_g^G - \sqrt{\rho}Z_t}{\sqrt{1-\rho}}\right)$;
  - G and g are the indices of the grades in the transition matrices at the beginning and end of the period, respectively;
  - x_g^G is the inverse standard normal transformation of the cumulative transition rate probabilities for grades G and g;
  - Φ denotes the standard normal distribution;
  - $n_{t,g}$ is the number of obligors in grade g at the time t;
  - Z_t is the systemic factor being optimized.

- The optimization procedure finds the optimal Z_t factor that best approximates the PiT transition matrix, given the TtC matrix and the parameter ρ.

- More details on this approach can be found here.

Simulation Setup

For an asset correlation value of $\rho = 0.0163$ and the TtC and PiT transition matrices provided below, this simulation aims to find the Z_t factor that best approximates the PiT matrix. The number of obligors (`Obs`) from the PiT matrix is used as an additional input in the optimization process described in the previous slide.

TtC transition matrix:

```
##   G-to-g    AAA     AA      A    BBB     BB      B    CCC      D
##      AAA 91.13%  8.00%  0.70%  0.10%  0.05%  0.01%  0.01%  0.01%
##       AA  0.70% 91.03%  7.47%  0.60%  0.10%  0.07%  0.02%  0.01%
##        A  0.10%  2.34% 91.54%  5.08%  0.61%  0.26%  0.01%  0.05%
##      BBB  0.02%  0.30%  5.65% 87.98%  4.75%  1.05%  0.10%  0.15%
##       BB  0.01%  0.11%  0.55%  7.77% 81.77%  7.95%  0.85%  1.00%
##        B  0.00%  0.05%  0.25%  0.45%  7.00% 83.50%  3.75%  5.00%
##      CCC  0.00%  0.01%  0.10%  0.30%  2.59% 12.00% 65.00% 20.00%
```

PiT transition matrix:

```
##   G-to-g Obs    AAA     AA      A    BBB     BB      B    CCC      D
##      AAA  85 92.94%  4.71%  2.35%  0.00%  0.00%  0.00%  0.00%  0.00%
##       AA 220  0.46% 92.52%  6.08%  0.47%  0.47%  0.00%  0.00%  0.00%
##        A 480  0.00%  4.45% 84.95%  9.54%  0.64%  0.00%  0.00%  0.42%
##      BBB 298  0.37%  0.37%  3.26% 85.52%  9.78%  0.37%  0.00%  0.34%
##       BB 168  0.00%  0.68%  0.00%  2.68% 82.42% 10.05%  0.00%  4.17%
##        B 161  0.00%  0.00%  0.72%  0.72%  2.89% 87.50%  5.06%  3.11%
##      CCC  16  0.00%  0.00%  0.00%  0.00%  0.00%  7.39% 73.86% 18.75%
```

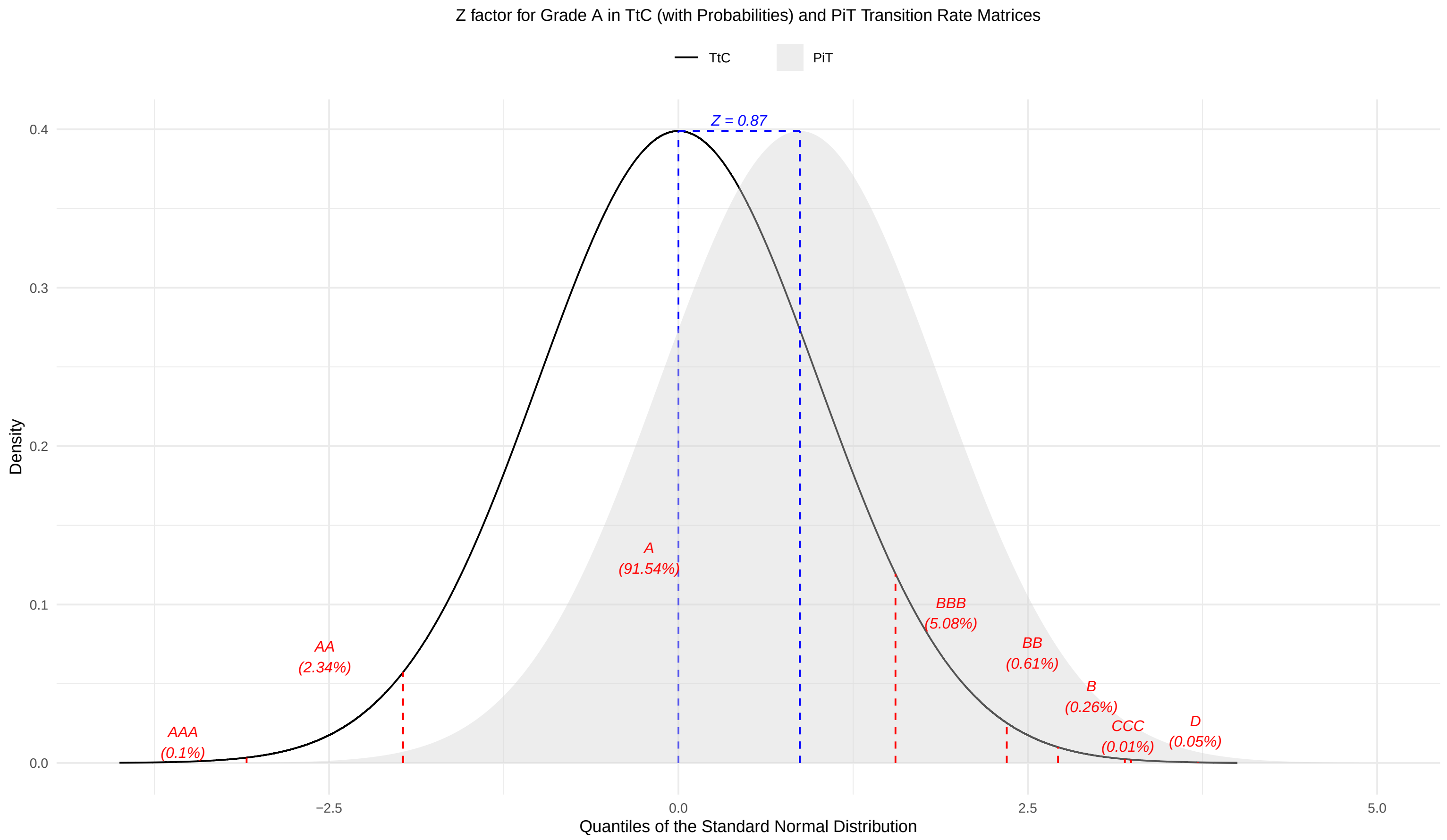
Simulation Results

- Given the simulation inputs, the optimal value of Z_t is 0.87.
- The matrix below presents the resulting transition rates that best approximate the observed PiT matrix under the described optimization process.
- The following slide visualizes the TtC to fitted PiT transformation procedure using the optimized Z_t value for the grade A.

Fitted Matrix

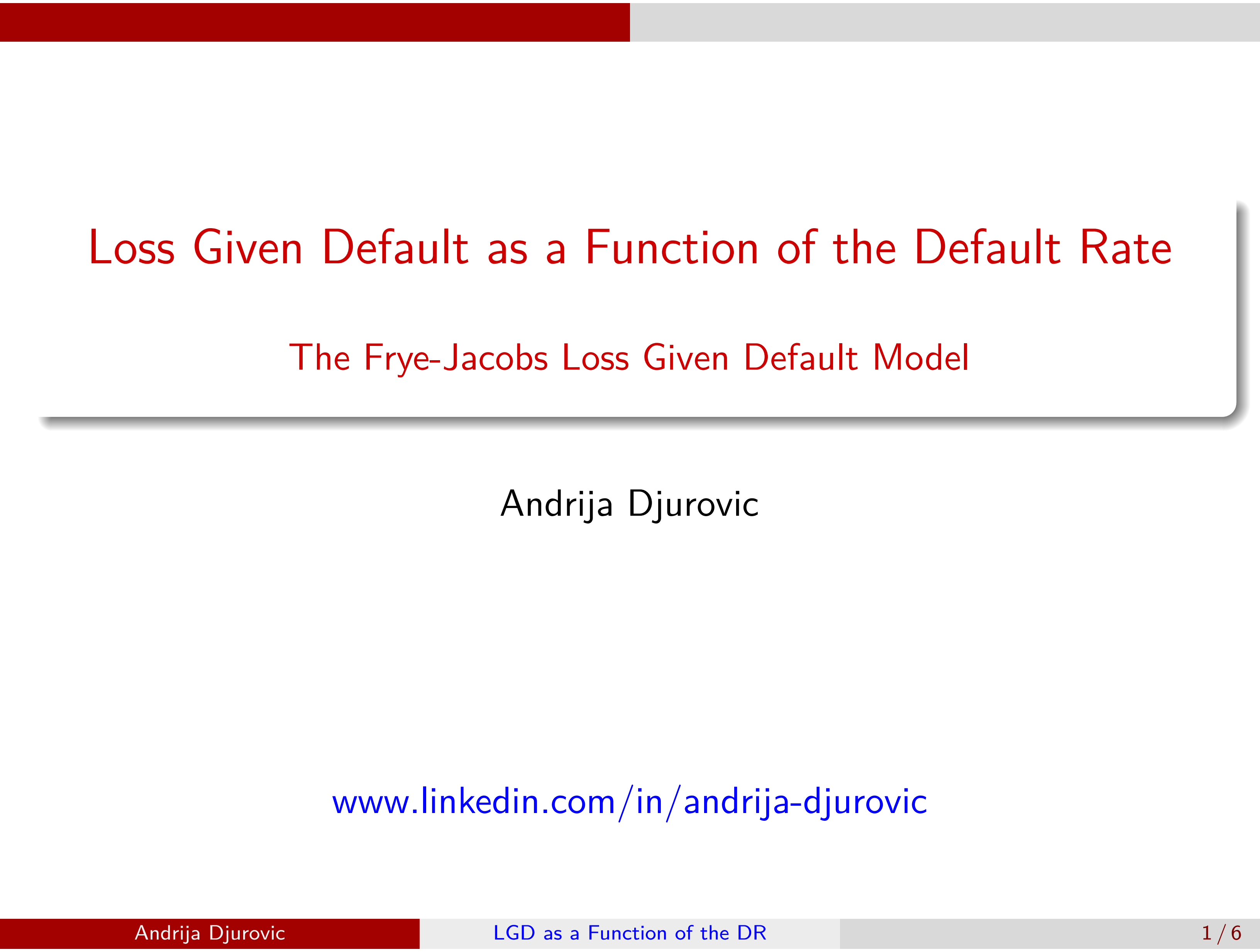
```
##   G-to-g Obs     AAA     AA      A    BBB     BB      B    CCC      D
##      AAA  85 89.40%  9.49%  0.89%  0.13%  0.07%  0.01%  0.01%  0.00%
##       AA 220  0.48% 89.61%  8.88%  0.76%  0.13%  0.09%  0.03%  0.01%
##        A 480  0.06%  1.73% 90.91%  6.10%  0.77%  0.34%  0.01%  0.07%
##      BBB 298  0.01%  0.20%  4.42% 88.04%  5.68%  1.32%  0.13%  0.20%
##       BB 168  0.01%  0.07%  0.38%  6.23% 81.66%  9.34%  1.05%  1.26%
##        B 161  0.00%  0.03%  0.17%  0.32%  5.59% 83.43%  4.36%  6.10%
##      CCC  16  0.00%  0.01%  0.06%  0.21%  1.96% 10.14% 64.57% 23.06%
```

Simulation Results cont.



2 Loss Given Default (LGD) Models

2.1 Loss Given Default as a Function of the Default Rate - The Frye-Jacobs LGD Model



Loss Given Default as a Function of the Default Rate

The Frye-Jacobs Loss Given Default Model

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | LGD as a Function of the DR | 1 / 6

The Frye-Jacobs LGD - Assumptions and the Functional Form

- A study by Frye and Jacobs predicts Loss Given Default (LGD) as a function of the default rate (DR).
- The LGD function relates to asymptotic DR or Point-in-Time Probability of Default (PiT PD) and Point-in-Time (PiT) LGD under certain assumptions.
- Main assumptions of the LGD function:
  - the Expected Loss (EL) and the DR (PiT PD) are co-monotonic;
  - EL and DR (PiT PD) follow a Vasicek two-parameter distribution;
  - correlation parameter (ρ) is shared between DR and EL.

The Frye-Jacobs LGD - Assumptions and the Functional Form cont.

The Frye-Jacobs LGD function has the following form:

$$cLGD = \frac{\Phi\left[\Phi^{-1}(cDR) - k\right]}{cDR}$$

with k being the Risk Index given by the following expression:

$$k = \frac{\Phi^{-1}(PD) - \Phi^{-1}(EL)}{\sqrt{1-\rho}}$$

where:

- cDR denotes conditional default rate (PiT PD);
- PD represents TTC PD;
- EL is the average loss rate; and
- ρ represents the asset correlation shared between DR and EL.

Simulation Setup

The simulation aims to generate competing predictions for the tail LGD (98^{th} percentile) utilizing the LGD function and linear regression as predictors. To give linear regression an advantage in this contest, the data-generating process is assumed to follow a linear model with known values of the intercept (a) and slope (b).
For the following assumed values: $TTC\ DR = 0.03$, $\rho = 0.10$, $\sigma^2 = 0.20$, $a = 0.50$, $b = 2.3$, and number of obligors $n = 1,000$, the simulation of a single year of data involves the following steps:

1. Simulate the value of the systemic factor z by drawing a random number from the standard normal distribution.
2. Given the $TTC\ DR$, ρ, and z from step 1, simulate the $PiT\ PD$ based on the Vasicek distribution.
3. Simulate the number of defaults D from the binomial distribution for a given n and simulated $PiT\ PD$.
4. Simulate conditional LGD $cLGD$ assuming the linear model of the form $a + b \cdot PiT\ PD$.
5. Given the $cLGD$, simulate LGD by drawing the value from the normal distribution with the mean $cLGD$ and standard deviation $\frac{\sigma}{\sqrt{D}}$.

The above process is repeated ten times, simulating a dataset for $T = 10$ years.

Simulation Setup cont.

The process described in the previous slide is simulated $N = 10,000$ times.

The LGD function and LGD linear regression model are estimated for each simulated dataset.

The elements of the LGD function are calculated as follows: parameter ρ is estimated by maximizing the log-likelihood of the Vasicek probability density function, $TTC\ DR$ is calculated as an average of the simulated T default rates, and the expected loss (EL) is calculated as an average of the product of the LGD and $PiT\ PD$.

The LGD of the linear model is calculated based on the selected values of coefficients a and b and the $TTC\ DR$.

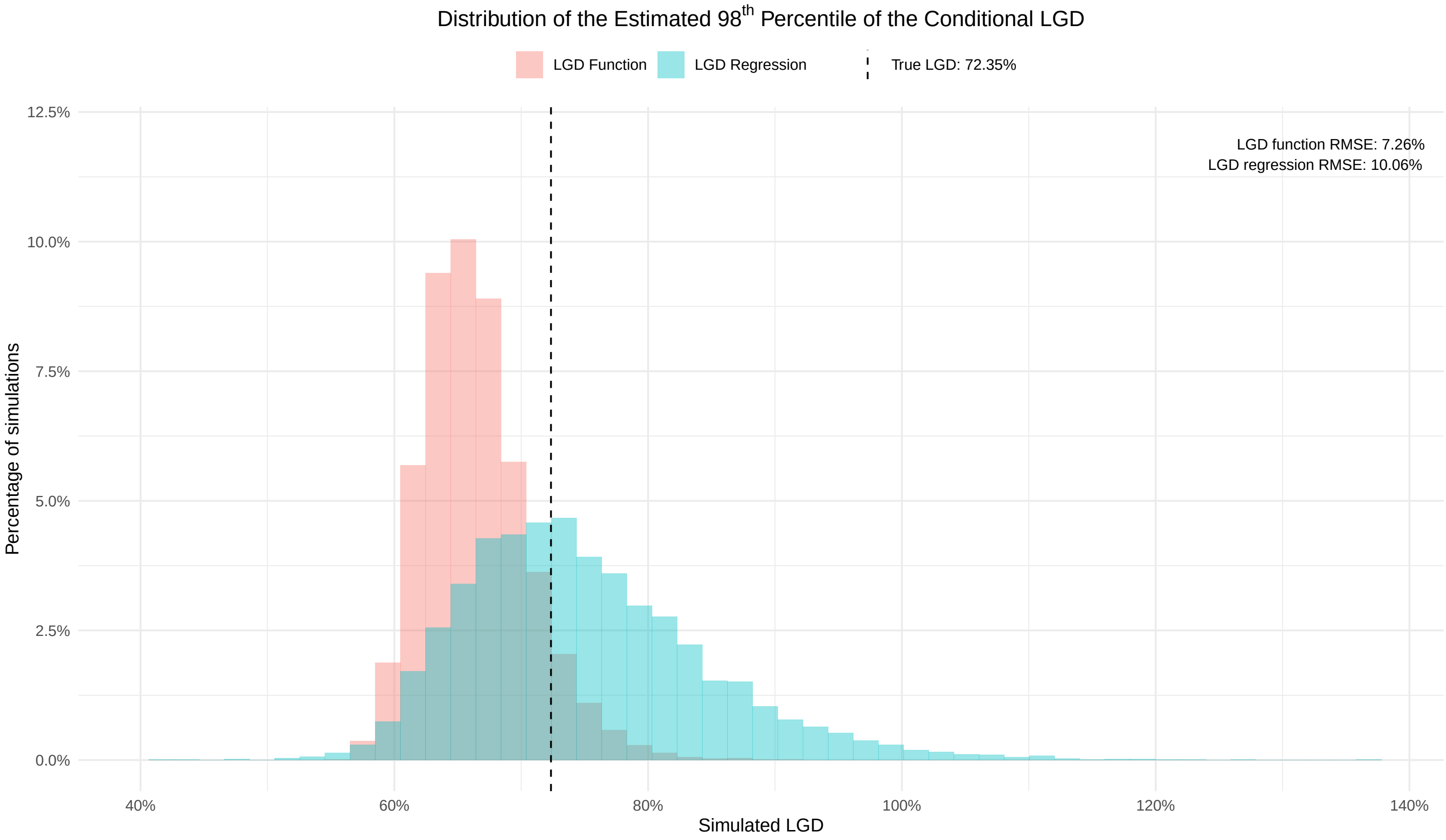
Deriving the LGD function and LGD linear regression distributions based on the $N = 10,000$ repetitions, the final step is to compare the root mean square error of these two distributions calculated against the known 98^{th} percentile of the LGD given by the following formula:

$$LGD_{98^{th}percentile} = a + b \cdot PD_{98^{th}percentile}$$

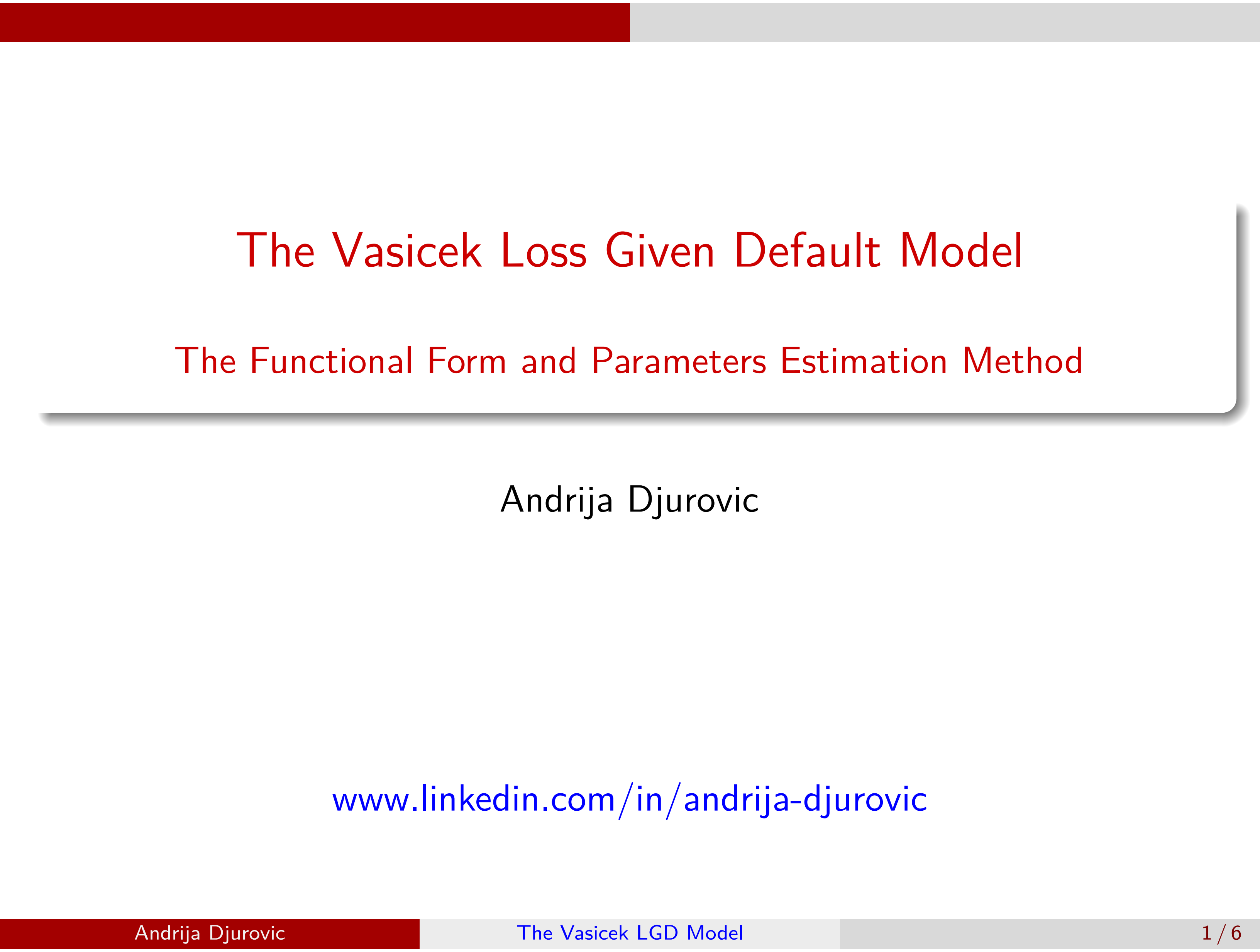
where $PD_{98^{th}percentile}$ is defined as:

$$PD_{98^{th}percentile} = \Phi\left[\frac{\Phi^{-1}(TTC\ PD) + \sqrt{\rho}\cdot\Phi^{-1}(98^{th}percentile)}{\sqrt{1-\rho}}\right]$$

Simulation Results



2.2 The Vasicek LGD Model - The Functional Form and Parameters Estimation Method



The Vasicek Loss Given Default Model

The Functional Form and Parameters Estimation Method

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

The Vasicek Probability of Default Model

The Vasicek Probability of Default (PD) model is derived from the asset return equation represented as:

$$y = \sqrt{\rho}z + \sqrt{1-\rho}\epsilon$$

where:

- ρ is asset correlation;
- z represents the systemic factor which follows a standard normal distribution
- ϵ denotes the idiosyncratic factor which follows a standard normal distribution.

Starting from the above equation, the Vasicek PD model is defined by two parameters, p and ρ, commonly referred to as the average default rate and asset correlation, respectively.

The parameter p is usually estimated as an average of the observed default rates, while the parameter ρ can be estimated by maximizing the log-likelihood of the Vasicek probability density function:

$$f_{p,\rho}(x) = \sqrt{\frac{1-\rho}{\rho}} e^{\frac{1}{2}\left(\phi^{-1}(x)^2 - \left(\frac{\sqrt{1-\rho}\phi^{-1}(x)-\phi^{-1}(p)}{\sqrt{\rho}}\right)^2\right)}$$

Details on the Vasicek (distribution) PD model and different parameter estimation methods are available at the following link.

The Vasicek Loss Given Default Model

The Vasicek Loss Given Default (LGD) model is derived from the recovery equation. The recovery equation looks similar to the asset return and is defined as:

$$r = \mu + \sigma\sqrt{q}z + \sigma\sqrt{(1-q)}\epsilon$$

where:

- μ is defined as a quantity parameter (similar to the parameter p from the PD model);
- σ represents the quality parameter;
- q is the sensitivity parameter (similar to the parameter ρ from the PD model);
- z represents the systemic factor derived from the PD model;
- ϵ denotes the idiosyncratic factor independent from z which follows a standard normal distribution.

Model parameters can be obtained by maximizing the log-likelihood of the following probability density function:

$$f_{\mu,\sigma,q}(x) = \frac{1}{\sqrt{2\pi\sigma^2(1-q)}}e^{-\frac{(x-\mu-\sigma\sqrt{q})^2}{2\sigma^2(1-q)}}$$

For more information about the Vasicek LGD model, check the following paper.

Simulation Study

The dataset below is utilized solely for simulation purposes, whereas in real-world scenarios, the development of recovery rates usually tends to exhibit less volatility.

Simulation Dataset:

```
##    T     dr     rr    lgd
##    1 0.1024 0.7433 0.2567
##    2 0.1130 0.1398 0.8602
##    3 0.0546 0.2127 0.7873
##    4 0.0295 0.8631 0.1369
##    5 0.0508 0.1141 0.8859
##    6 0.0236 0.8811 0.1189
##    7 0.0350 0.6007 0.3993
##    8 0.0285 0.9723 0.0277
##    9 0.0118 0.9432 0.0568
##   10 0.0260 0.8167 0.1833
##   11 0.0298 0.2798 0.7202
##   12 0.0251 0.3129 0.6871
##   13 0.0802 0.6168 0.3832
##   14 0.0124 0.7047 0.2953
##   15 0.0272 0.6373 0.3627
##   16 0.0493 0.3651 0.6349
##   17 0.0301 0.7007 0.2993
##   18 0.1204 0.5110 0.4890
##   19 0.0093 0.8323 0.1677
##   20 0.0291 0.6406 0.3594
```

The Vasicek PD Model Parameters:

```
##        p    rho
##   0.0444 0.0965
```

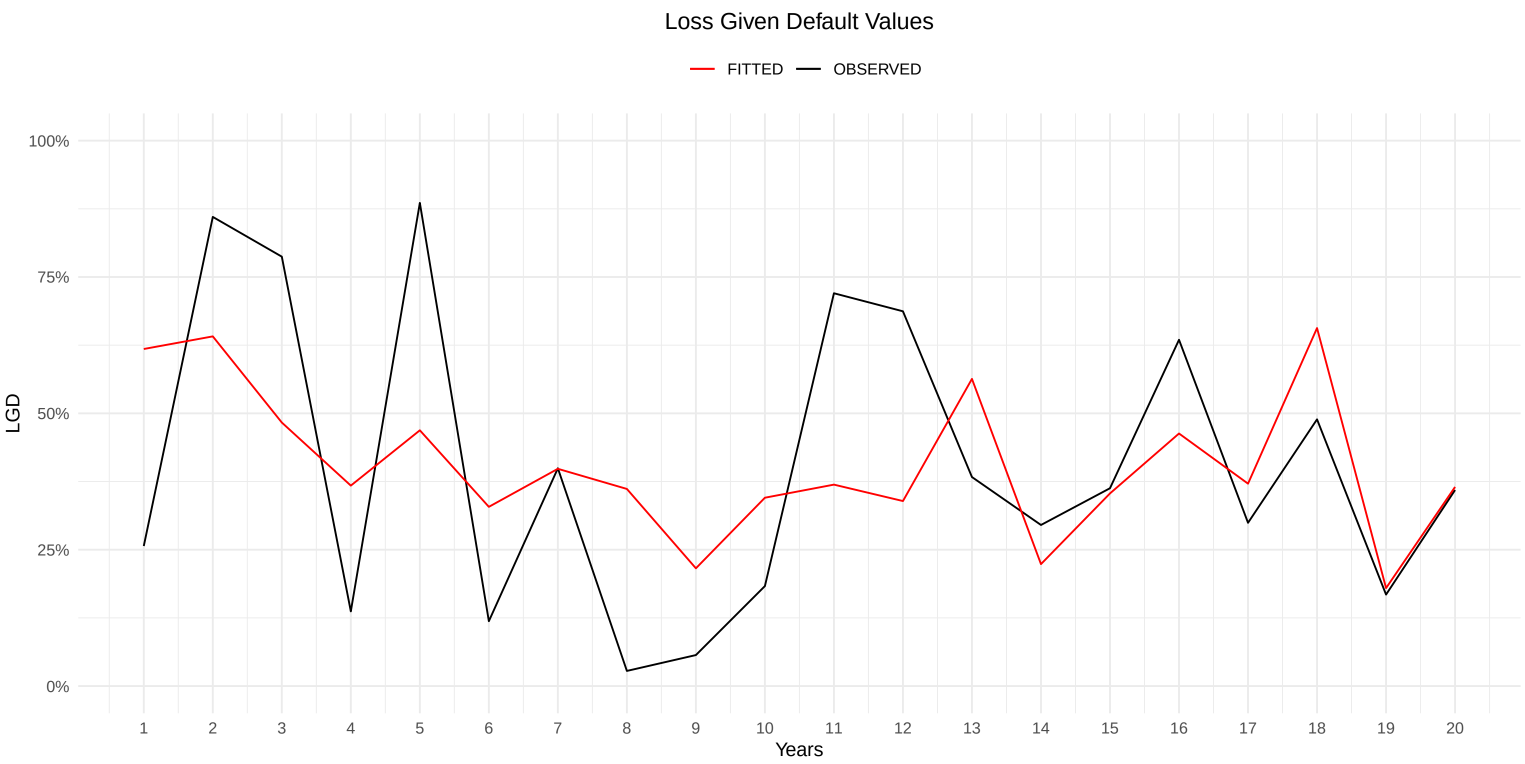
z:

```
##       1       2       3       4       5
## -1.5984 -1.7737 -0.5767  0.2998 -0.4686
##       6       7       8       9      10
##  0.5946  0.0663  0.3461  1.4488  0.4680
##      11      12      13      14      15
##  0.2862  0.5143 -1.1830  1.3904  0.4083
##      16      17      18      19      20
## -0.4241  0.2727 -1.8892  1.7238  0.3182
```

The Vasicek Recovery Model Parameters:

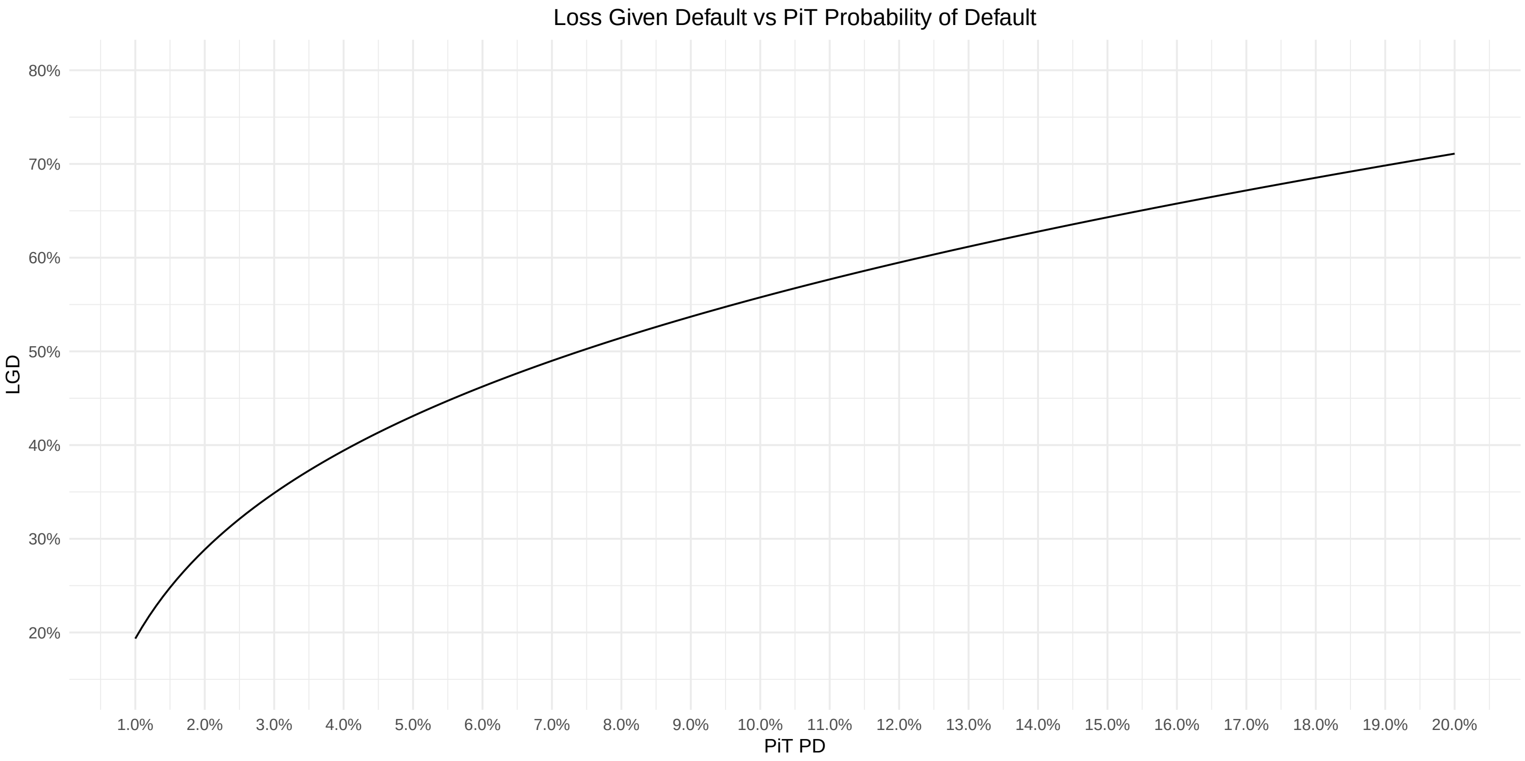
```
##     mu  sigma      q
## 0.5929 0.2644 0.2492
```

Simulation Study cont.

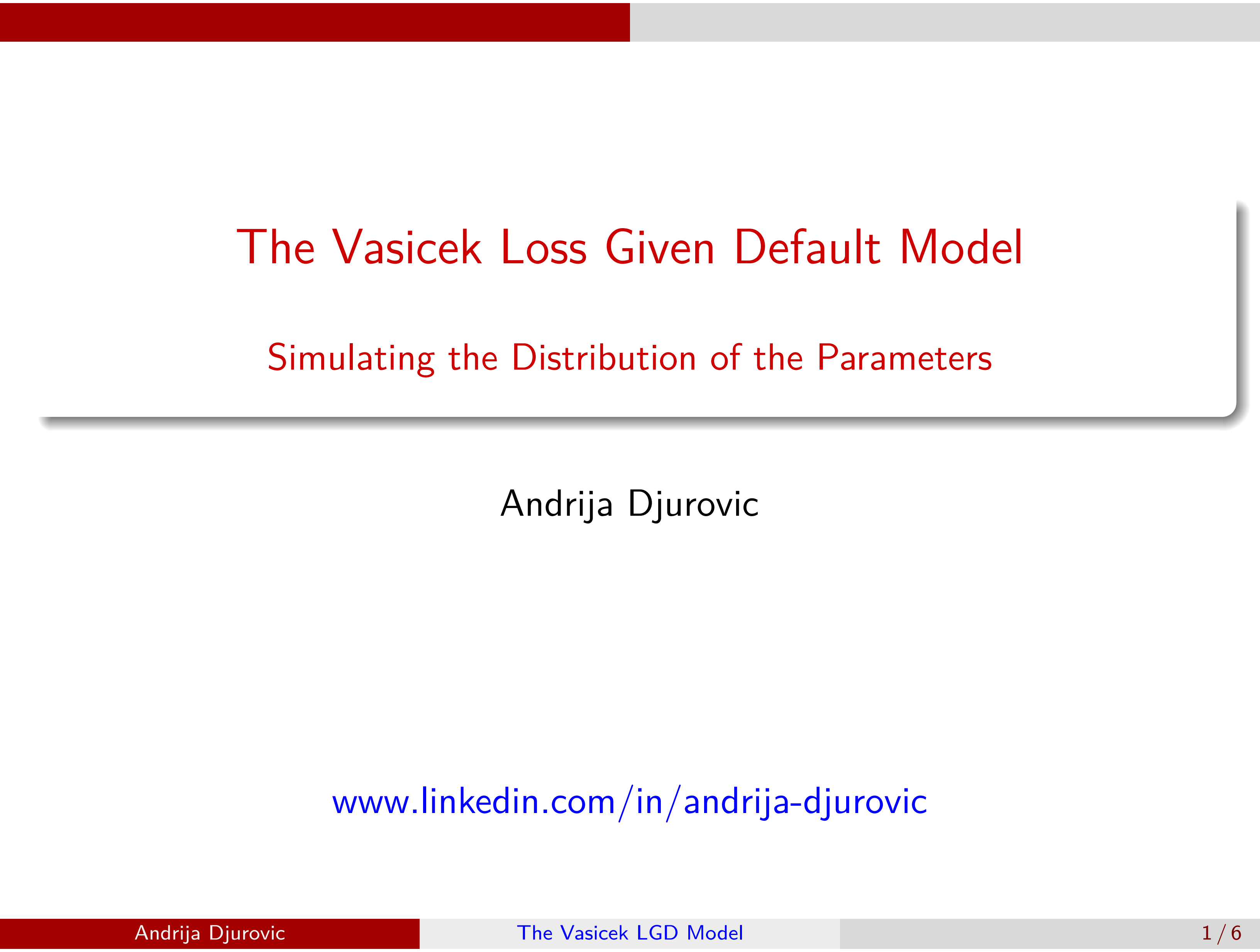


Simulation Study cont.

The following graph demonstrates the relationship between the *LGD* and *PiT PD* values under the following assumptions: $p = 0.05$, $\rho = 0.10$, $\mu = 0.60$, $\sigma = 0.30$, and $q = 0.15$.



2.3 The Vasicek LGD Model - Simulating the Distribution of the Parameters



The Vasicek Loss Given Default Model

Simulating the Distribution of the Parameters

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

The Vasicek Loss Given Default Model

The Vasicek Loss Given Default (LGD) model is derived from the recovery equation. The recovery equation, resembling the asset return, is defined as:

$$r = \mu + \sigma\sqrt{q}z + \sigma\sqrt{(1-q)}\epsilon$$

where:

- μ is a quantity parameter (similar to parameter p from the PD model);
- σ represents the quality parameter;
- q is the sensitivity parameter (similar to parameter ρ from the PD model);
- z represents the systemic factor derived from the Vasicek PD model;
- ϵ denotes the idiosyncratic factor, independent of z, which follows a standard normal distribution.

Given that r follows a normal distribution with mean $\mu + \sigma\sqrt{q}z$ and variance $\sigma^2(1-\rho)$, model parameters can be obtained by maximizing the log-likelihood of the following probability density function:

$$f_{\mu,\sigma,q}(x) = \frac{1}{\sqrt{2\pi\sigma^2(1-q)}} e^{-\frac{(x-\mu-\sigma\sqrt{q})^2}{2\sigma^2(1-q)}}$$

Simulating the Distribution of the Parameters

Assuming specific inputs of the data-generating process, this simulation aims to derive the distributions of the parameters of the Vasicek LGD model. As the Vasicek LGD model shares the systemic factor with the Vasicek PD model, the simulation process follows these steps:

1. For the selected values of p_{sim} and ρ_{sim}, simulate T observations of the default rate from the Vasicek PD distribution (for details, see Slide 2 of the document)
2. Given the simulated default rates, calculate the parameter p_{obs} as the average of the simulated observations.
3. Given the simulated default rates and p_{obs} from steps 1 and 2, estimate the asset correlation (ρ_{obs}) using the Indirect Moment Matching method (for details, see Slide 6 of the document)
4. Based on p_{obs} and ρ_{obs}, derive the values of the systemic factor z_{obs}.

Simulating the Distribution of the Parameters cont.

5. For the selected parameters μ_{sim}, σ_{sim}, q_{sim}, and observed values of the systemic factor z_{obs}, simulate the recovery rates from the Vasicek LGD model.
6. Given the simulated recoveries, use the maximum likelihood method to estimate the parameters μ_{obs}, σ_{obs}, and q_{obs}.
7. Repeat steps 2 to 6 N times.
8. Collect the values of the estimated parameters (N), calculate their expected values, and plot their distribution.

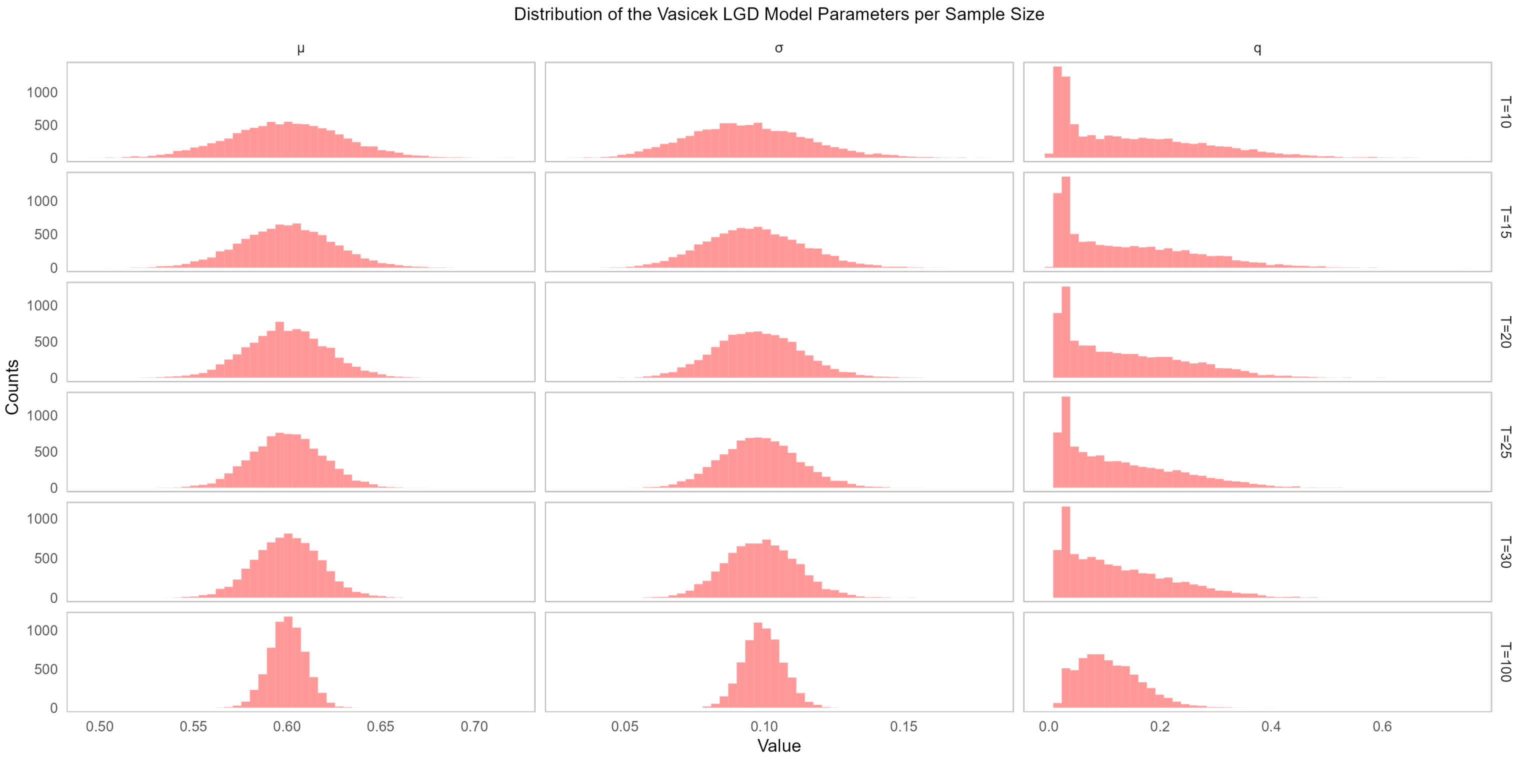
For the described simulation design, the following data-generating inputs were selected: $p_{sim} = 0.05$, $\rho_{sim} = 0.10$, $\mu_{sim} = 0.60$, $\sigma_{sim} = 0.10$, $q_{sim} = 0.10$, $T = [10, 15, 20, 25, 30, 100]$, and $N = 10,000$.

Simulation Study Results

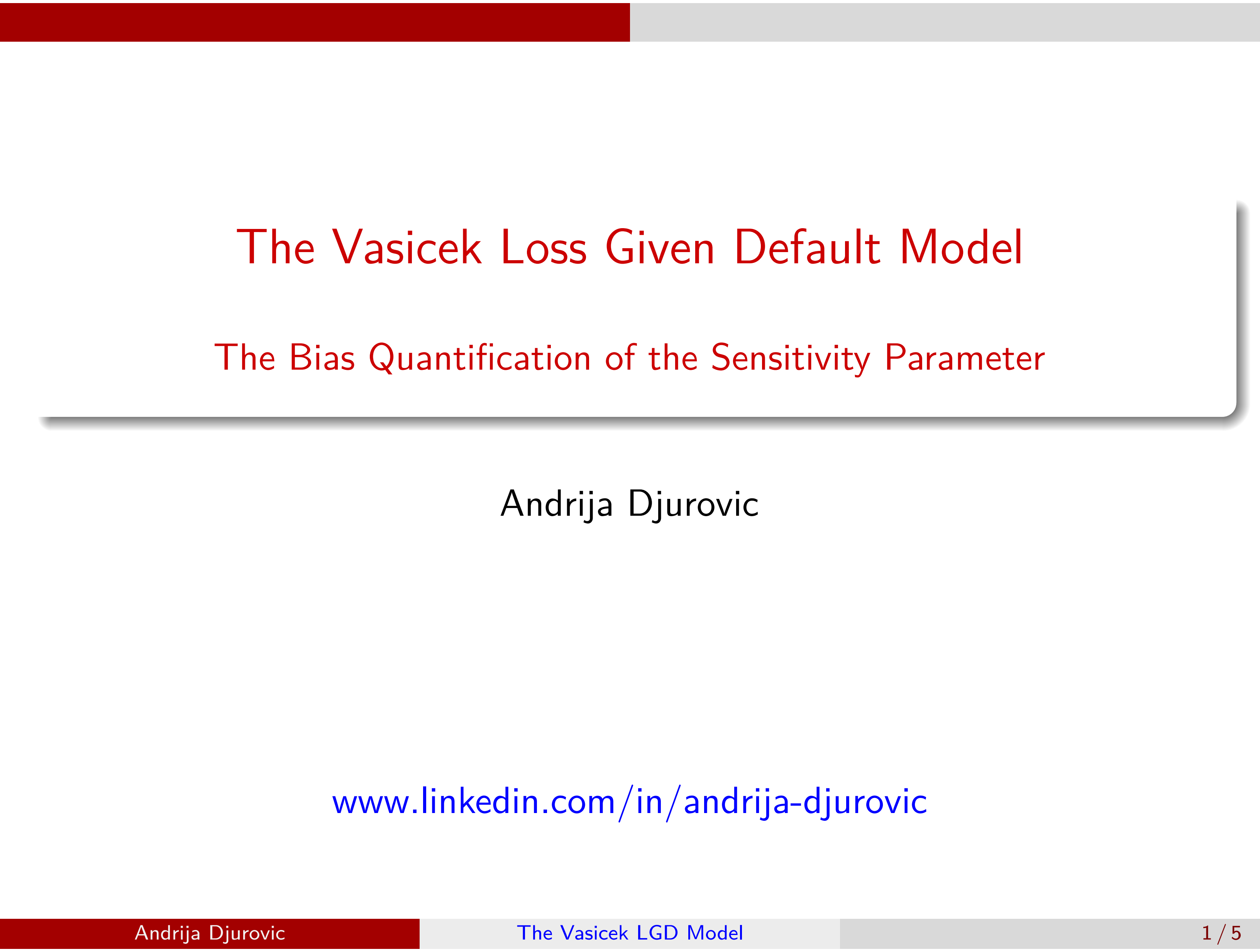
The following table presents the expected values of the simulated parameters:

```
##      T     mu  sigma      q
##     10 0.5994 0.0942 0.1451
##     15 0.5998 0.0966 0.1381
##     20 0.5997 0.0977 0.1337
##     25 0.6001 0.0985 0.1261
##     30 0.6002 0.0989 0.1237
##    100 0.6002 0.0996 0.1081
```

Simulation Study Results cont.



2.4 The Vasicek LGD Model - The Bias Quantification of the Sensitivity Parameter



The Vasicek Loss Given Default Model

The Bias Quantification of the Sensitivity Parameter

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic The Vasicek LGD Model 1 / 5

The Functional Form and Estimation Method of the Parameters

The Vasicek Loss Given Default (LGD) model is derived from the recovery equation. The recovery equation, resembling the asset return, is defined as:

$$r = \mu + \sigma\sqrt{q}z + \sigma\sqrt{(1-q)}\epsilon$$

where:

- μ is a quantity parameter (similar to parameter p from the PD model);
- σ represents the quality parameter;
- q is the sensitivity parameter (similar to parameter ρ from the PD model);
- z represents the systemic factor derived from the Vasicek PD model;
- ϵ denotes the idiosyncratic factor, independent of z, which follows a standard normal distribution.

Given that r follows a normal distribution with mean $\mu + \sigma\sqrt{q}z$ and variance $\sigma^2(1-\rho)$, model parameters can be obtained by maximizing the log-likelihood of the following probability density function:

$$f_{\mu,\sigma,q}(x) = \frac{1}{\sqrt{2\pi\sigma^2(1-q)}}e^{-\frac{(x-\mu-\sigma\sqrt{q})^2}{2\sigma^2(1-q)}}$$

Bias Quantification Process

The following steps outline the bias quantification process using Monte Carlo simulations:

1. Collect default and recovery rate data.
2. Select the parameters estimation method for the Vasicek PD model.
3. Based on the collected default rates and the selected method, estimate the parameters $p_{observed}$ and $\rho_{observed}$.
4. Given the values from the step 3, calculate the systemic factor $z_{observed}$
5. Based on the collected recovery data and calculated values of $z_{observed}$, estimate the parameters of the Vasicek LGD model.
6. Based on the N Monte Carlo simulations, optimize the data-generating process for q_{true} by minimizing the difference between the average value of the q_{true} distribution and the $q_{observed}$ given that the true values of parameters $\mu_{observed}$ and $\sigma_{observed}$ are equal to the estimated ones.

Note that the presented process assumes that the parameter q of the Vasicek LGD model estimated based on the observed data represent the average value of the true parameters' distribution.

Simulation Setup and Results

The following table presents the simulation inputs of the hypothetical portfolio:

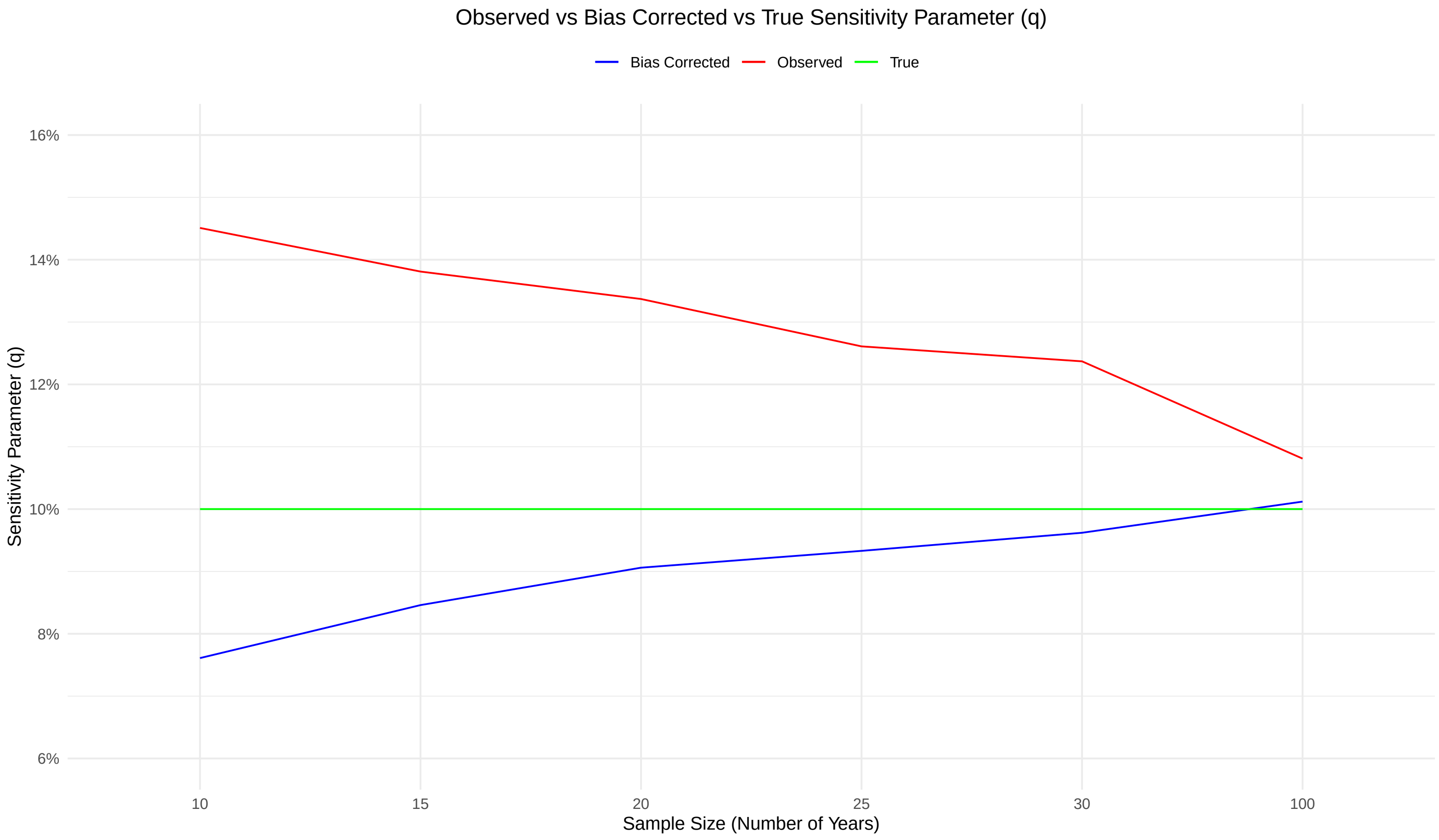
```
##     T     mu  sigma q_observed
##    10 0.5994 0.0942     0.1451
##    15 0.5998 0.0966     0.1381
##    20 0.5997 0.0977     0.1337
##    25 0.6001 0.0985     0.1261
##    30 0.6002 0.0989     0.1237
##   100 0.6002 0.0996     0.1081
```

Note that the above values are directly obtained from the Vasicek models data-generating process with the value of $p = 0.05$, $\rho = 0.10$, and $q = 0.10$, while the remaining parameters μ and σ are given in the above hypothetical portfolio. Therefore, a $q_{bias\_corrected}$ closer to 0.10 indicates a better bias-corrected estimation.

After running the ($N = 10,000$) Monte Carlo simulations for bias quantification of the asset correlation parameter, the following additional values have been obtained:

```
##     T     mu  sigma q_observed q_bias_corrected q_true
##    10 0.5994 0.0942     0.1451           0.0761    0.1
##    15 0.5998 0.0966     0.1381           0.0846    0.1
##    20 0.5997 0.0977     0.1337           0.0906    0.1
##    25 0.6001 0.0985     0.1261           0.0933    0.1
##    30 0.6002 0.0989     0.1237           0.0962    0.1
##   100 0.6002 0.0996     0.1081           0.1012    0.1
```

Simulation Setup and Results cont.



3 Ordinary Least Square (OLS) Regression

3.1 Consequences of Violating the Normality Assumption for OLS Regression

Consequences of Violating the Normality for OLS Regression

Andrija Djurovic[*]

[*] www.linkedin.com/in/andrija-djurovic

The predominant statistical approach in IFRS9 FLI modeling involves using OLS regression, with the p-value as one of the criteria for selecting the final model. Even though one of the assumptions underlying OLS regression is the normality of residuals, this assumption is often overlooked in practice. The primary reason for this neglect stems from the need for more consensus among practitioners regarding the significance of this assumption in influencing regression estimates and inference.

Even in literature, opinions vary regarding the importance of the normality assumption. This spectrum of views ranges from suggestions to completely omit testing to the more commonly held belief that violating this assumption impacts the inference drawn from OLS regression, including test statistics and p-values.

In the context of IFRS9 FLI modeling, this assumption, coupled with a relatively low ratio between the number of observations and the number of independent variables, can pose specific challenges. Consequently, this document aims to present the framework for exploring the correlation between the number of observations per independent variable, simulated deviations from normality, and type I error. It is important to emphasize that the provided example should not be viewed as a broad generalization of deviations from normality. Instead, it can serve as a basis for tailoring the process to meet specific requirements and address unique use cases.

Let's begin by defining the assumptions and implementing the simulation process in `R`.

```
#simulation parameters
sim.rep <- 100                  #simulation replicates
iv.n <- c(1, 2, 3, 5, 9, 15) #number of independent variables
n <- 45                         #number of observations
iv.l <- length(iv.n)            #num. of options for indep. vars
alpha <- 0.05                   #significance level
B <- 100                        #number of mc simulations
```

Based on the inputs provided above, we can determine the average number of observations for each set of independent variables.

```
#average number of observations per independent variables
n / iv.n
```

```
## [1] 45.0 22.5 15.0  9.0  5.0  3.0
```

By incorporating assumptions concerning the distribution of variables (precisely, a t-distribution with 2 degrees of freedom) and the anticipated zero value for the estimates of independent variables in relation to the target, we can model the impact on type I error, as demonstrated in the subsequent code snippet.

```r
#-------------------------run the simulations-------------------------#
#empty list to store the results
res <- vector("list", sim.rep)

for   (z in 1:sim.rep) {

       res.z <- vector("list", iv.l)

       for   (i in 1:iv.l) {
              set.seed(i + z)
              iv.i <- iv.n[i]

              #simulate the independent variables (t-distribution, df = 2)
              iv <- replicate(n = iv.i,
                              expr = rt(n = n, df = 2)
                              )
              iv <- data.frame(iv)
              names(iv) <- paste0("x", 1:ncol(iv))

              #store the xs into data frame
              db <- cbind.data.frame(iv)

              #regression formula
              frm <- paste0("y ~ ", paste(names(iv), collapse = " + "))

              #-------mc simulations-------#
              #empty data frame to store the results
              res.i <- data.frame(matrix(data = NA,
                                         ncol = 2 * iv.i + 2,
                                         nrow = B)
                                  )
              names(res.i) <- c("simulation",
                                names(db),
                                paste0(names(db), ".pval"),
                                "f.test"
                                )

              for   (j in 1:B) {
                     set.seed(i + j*10 + z)
                     #simulate the target (t-distribution, df = 2)
                     db$y <-  rt(n = n, df = 2)
                     #run the regression
                     lr.j  <- lm(formula = frm,
                                 data = db)
                     #regression summary
                     lr.j.s <- summary(lr.j)
                     #f-test
                     fs.j <- lr.j.s$fstatistic
                     fs.p <- pf(q = fs.j[1],
                                df1 = fs.j[2],
                                df2 = fs.j[3],
                                lower.tail = FALSE)
                     #summarize the results
                     res.j <- c(j,                            #simulation id
                                lr.j.s$coefficients[-1, 1], #estimates
                                lr.j.s$coefficients[-1, 4], #p-value of estimates
                                fs.p                        #f-test p-value
                                )
                     res.i[j, ] <- res.j
                     }

              #add the number of indep. var
              res.i <- cbind.data.frame(iv.n = iv.i,
                                        res.i)
              #store the results
              res.z[[i]] <- res.i
              }

       #type I error of the regression
       res[[z]] <- sapply(X = res.z,
                          FUN = function(x) mean(x[, "f.test"] < alpha))
       }
```

In conclusion, considering the abovementioned assumptions, we can summarize the outcomes and examine the pattern of type I error across various sets of independent variables in the model.

```
#summarise the results
res <- do.call("rbind", res)
colnames(res) <- iv.n
```

```
#calculate the average type I error
colMeans(res)
```

```
##      1      2      3      5      9     15
## 0.0540 0.0633 0.0729 0.0809 0.0883 0.0957
```

As observed in the results, a decrease in the number of observations per independent variable corresponds to an increase in type I error. Moreover, within the simulated distribution, there is a noticeable rise in type I error beyond the expected significance level. This indicates a potential issue with the inference from regression in this specific configuration.

Let's now repeat the same exercises in `Python`.

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf

# Simulation parameters
sim_rep = 100                #simulation replicates
iv_n = [1, 2, 3, 5, 9, 15]   #number of independent variables
n = 45                       #number of observations
iv_l = len(iv_n)             #num. of options for indep. vars
alpha = 0.05                 #significance level
B = 100                      #number of mc simulations

#average number of observations per independent variables
np.array([n / iv for iv in iv_n])
```

```
## array([45. , 22.5, 15. ,  9. ,  5. ,  3. ])
```

```
#-------------------------run the simulations-------------------------#
#empty list to store the results
res = [None]*sim_rep

for z in range(sim_rep):

    res_z = [None]*iv_l

    for i in range(iv_l):
        np.random.seed((i + 1) + (z + 1))
        iv_i = iv_n[i]
        #simulate independent variables (t-distribution, df = 2)
        iv = np.random.standard_t(df = 2, size = (n, iv_i))

        #store the xs into data frame
        db = pd.DataFrame(iv, columns = [f"x{i}" for i in range(0, iv_i)])

        #regression formula
        formula = f'y ~ {" + ".join(db.columns)}'

        #-------mc simulations-------#
        #empty data frame to store the results
        res_i = pd.DataFrame(np.nan, index = range(0, B),
                             columns = ["simulation"] +
                                       list(db.columns) +
                                       [f"{col}.pval" for col in db.columns] +
                                       ["f.test"])

        for j in range(0, B):
            np.random.seed((i + 1) + (j + 1) * 10 + (z + 1))
            #simulate the target (t-distribution, df = 2)
            db["y"] = np.random.standard_t(df = 2, size = n)
            #run the regression
            lr_j = smf.ols(formula = formula, data = db).fit()
            #f-test
            fs_j = lr_j.fvalue
            fs_p = lr_j.f_pvalue
            #summarize the results
            res_j = [j] + list(lr_j.params[1:]) + list(lr_j.pvalues[1:]) + [fs_p]
            res_i.loc[j] = res_j

        #add number of indep. var
        res_i.insert(0, "iv_n", iv_i)
        #store the results
        res_z[i] = res_i

    #type I error of the regression
    res_z_mean = [np.mean(x["f.test"] < alpha) for x in res_z]
    res[z] = res_z_mean
```

```
#summarise the results
res = pd.DataFrame(res, columns = iv_n)

#calculate the average type I error
res.mean()
```

```
## 1     0.0558
## 2     0.0652
## 3     0.0734
## 5     0.0858
## 9     0.0908
## 15    0.0922
## dtype: float64
```

In addition to the provided setup, readers are encouraged to explore variations such as different numbers of observations, simulations, and diverse distributions for independent variables, target, and residuals. They are also encouraged to examine individual estimates' confidence interval coverage and customize the examples for power calculation.

3.2 Consequences of Heteroscedasticity for OLS Regression

Consequences of Heteroscedasticity for OLS regression

Andrija Djurovic[*]

[*] www.linkedin.com/in/andrija-djurovic

One assumption of OLS regression is that residuals exhibit homoscedasticity, implying constant variance. When this variance is not constant, practitioners identify a heteroscedasticity problem in the OLS regression. While opinions on the significance of the normality assumption vary, a consensus on heteroscedasticity leads to unbiased but inefficient estimates. Essentially, the minimal variance of the estimates is compromised, rendering inferences from OLS unreliable.

The following simulation illustrates the potential consequences of heteroscedasticity, examining efficiency through the standard error of estimates and reporting the associated power.

Starting in `R`, the data for OLS regression is simulated with two independent variables (`x1` and `x2`) and a dependent variable (`y`) drawn from the standard normal distribution. The simulated estimates for `x1` and `x2` are set at `0.85` and `-0.65`.

```
#simulation parameters
n <- 45              #number of observations
iv <- 2              #number of independent variables

#simulate the x
set.seed(123)
x <- replicate(n = iv,
               expr = rnorm(n = n)
               )
x <- data.frame(x)
names(x) <- paste0("x", 1:iv)

#true regression parameters
intercept <- 0
beta1 <- 0.85
beta2 <- -0.65

#simulate the target
y <- intercept + beta1*x$x1 + beta2*x$x2

#store the inputs into data frame
db <- data.frame(y = y, x)
```

Now that we have generated the data above, we can proceed with the simulation and evaluate the implications of heteroscedastic errors.

```
#-------------------------run the simulations-------------------------#
#mc parameters
```

```r
alpha <- 0.05        #significance level
xw <- c(0.60, 0.35)  #x contribution to the heteroscedastic errors
B <- 1000            #number of mc simulations

#empty list to store the results (heteroscedastic residuals)
res.unw <- matrix(data = NA,
                  nrow = B,
                  ncol = 4)
res.unw <- data.frame(res.unw)
names(res.unw) <- c("b1", "b2", "b1.pval", "b2.pval")
#empty list to store the results of the weighted regression
res.wts <- res.unw

for   (i in 1:B) {
       #random seed
       set.seed(i * 2)

       #simulate the (centered) heteroscedastic errors
       error.i <- rnorm(n = n,
                        mean = 0,
                        sd = exp(xw[1]*db$x1 + xw[2]*db$x2))
       error.i <- error.i - mean(error.i)

       #utilize the weights to account for heteroscedasticity
       resid.i <- log(error.i^2)
       h.i <- exp(lm(formula = resid.i ~ db$x1 + db$x2)$fitted.values)
       weights.i <- 1 / h.i

       #simulate the target
       db$y.sim <- db$y + error.i

       #run the regressions
       lr.unw <- lm(formula = y.sim ~ x1 + x2,
                    data = db)
       lr.wts <- lm(formula = y.sim ~ x1 + x2,
                    weights = weights.i,
                    data = db)

       #store the simulation results
       res.unw[i, ] <- summary(lr.unw)$coefficients[2:3, c(1, 4)]
       res.wts[i, ] <- summary(lr.wts)$coefficients[2:3, c(1, 4)]
       }
```

Let's compare the true betas with those obtained from the regression with heteroscedastic errors (unweighted) and those derived from the regression where errors are modeled (weighted).

```
#confidence intervals
#true betas
c("beta1" = beta1, "beta2" = beta2)
```

```
## beta1 beta2
##  0.85 -0.65
```

```
#unweighted betas
b12.ci.unw <- sapply(X = res.unw[, c("b1", "b2")],
                     FUN = function(x) {
                           quantile(x = x,
                                    probs = c(alpha / 2, 0.5, 1 - alpha/2))
                            })
b12.ci.unw
```

```
##              b1         b2
## 2.5%  0.2081474 -1.2032158
## 50%   0.8413577 -0.6470709
## 97.5% 1.5091026 -0.1008995
```

```
#weighted betas
b12.ci.wts <- sapply(X = res.wts[, c("b1", "b2")],
                     FUN = function(x) {
                           quantile(x = x,
                                    probs = c(alpha / 2, 0.5, 1 - alpha/2))
                            })
b12.ci.wts
```

```
##              b1         b2
## 2.5%  0.5531600 -0.9935363
## 50%   0.8521381 -0.6535496
## 97.5% 1.1462048 -0.3274951
```

Lastly, we can examine the power in both approaches and observe the enhancement, particularly in the case of the weighted regression.

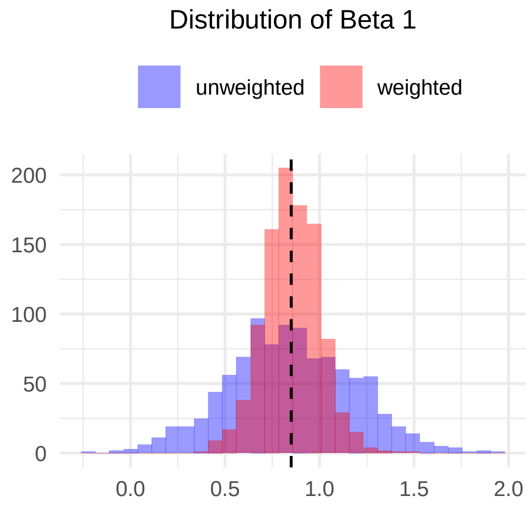
```r
#power - heteroscedastic errors
sapply(X = res.unw[, c("b1.pval", "b2.pval")],
       FUN = function(x) mean(x < alpha))
```

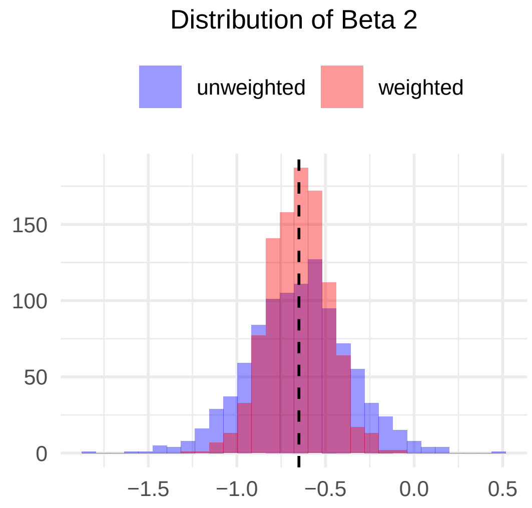
```
## b1.pval b2.pval
##   0.872   0.659
```

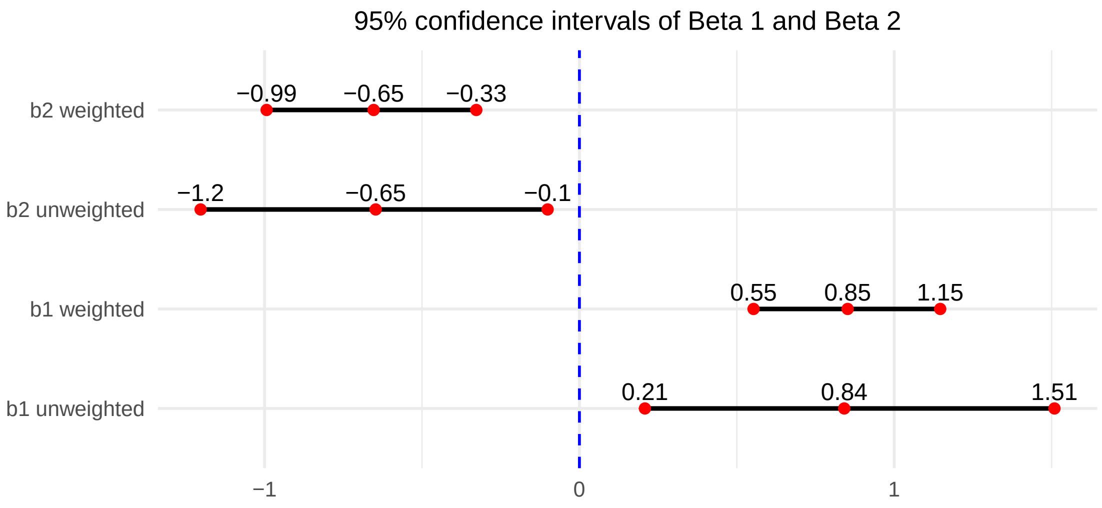
```r
#power - heteroscedastic errors modeled
sapply(X = res.wts[, c("b1.pval", "b2.pval")],
       FUN = function(x) mean(x < alpha))
```

```
## b1.pval b2.pval
##    1.00    0.97
```

To better comprehend the findings above, let's visually present the results.







Towards the conclusion of the exercise, we will export the simulated data to replicate the process in Python.

```r
#export the db to replicate the simulation in python
write.csv(x = db[, -ncol(db)],
          file = "db.csv",
          row.names = FALSE)
```

Now, let's code the same simulation in Python.

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf

#import db
db = pd.read_csv("db.csv")

#mc parameters
alpha = 0.05
B = 1000
xw = np.array([0.60, 0.35])

#create an empty dataframe to store results
columns = ["b1", "b2", "b1.pval", "b2.pval"]
res_unw = pd.DataFrame(index = range(B),
                       columns = columns,
                       dtype = float)
res_wts = res_unw.copy()

#-------------------------run the simulations-------------------------#

for i in range(B):
    #random seed
    np.random.seed((i + 1) * 2)

    #simulate the (centered) heteroscedastic errors
    error_i = np.random.normal(loc = 0,
                               scale = np.exp(xw[0] * db['x1'] +
                                              xw[1] * db['x2']))
    error_i = error_i - np.mean(error_i)
    #utilize the weights to account for heteroscedasticity
    db["resid_i"] = np.log(error_i ** 2)
    h_i = np.exp(smf.ols(formula = "resid_i ~ x1 + x2",
                         data = db).fit().fittedvalues)
    weights_i = 1 / h_i

    #simulate the target
    db['y_sim'] = db['y'] + error_i

    #run the regressions
    lr_unw = smf.ols(formula = "y_sim ~ x1 + x2",
                     data = db).fit()
    lr_wts = sm.WLS.from_formula(formula = "y_sim ~ x1 + x2",
                                 weights = weights_i,
                                 data = db).fit()

    res_unw.loc[i, ] = lr_unw.params[1:].tolist() + lr_unw.pvalues[1:].tolist()
    res_wts.loc[i, ] = lr_wts.params[1:].tolist() + lr_wts.pvalues[1:].tolist()

#confidence intervals
#true_betas
{"beta1": 0.85, "beta2": -0.65}
```

```
## {'beta1': 0.85, 'beta2': -0.65}
```

```
#unweighted betas
b12_ci_unw = res_unw[["b1", "b2"]].quantile([alpha / 2, 0.5, 1 - alpha / 2])
b12_ci_unw
```

```
##              b1        b2
## 0.025  0.188296 -1.221166
## 0.500  0.859330 -0.649491
## 0.975  1.494286 -0.061841
```

```
#weighted betas
b12_ci_wts = res_wts[["b1", "b2"]].quantile([alpha / 2, 0.5, 1 - alpha / 2])
b12_ci_wts
```

```
##              b1        b2
## 0.025  0.552355 -1.023943
## 0.500  0.849808 -0.652018
## 0.975  1.137475 -0.305847
```

```
#power
#heteroscedastic errors
(res_unw[["b1.pval", "b2.pval"]] < alpha).mean()
```

```
## b1.pval    0.862
## b2.pval    0.666
## dtype: float64
```

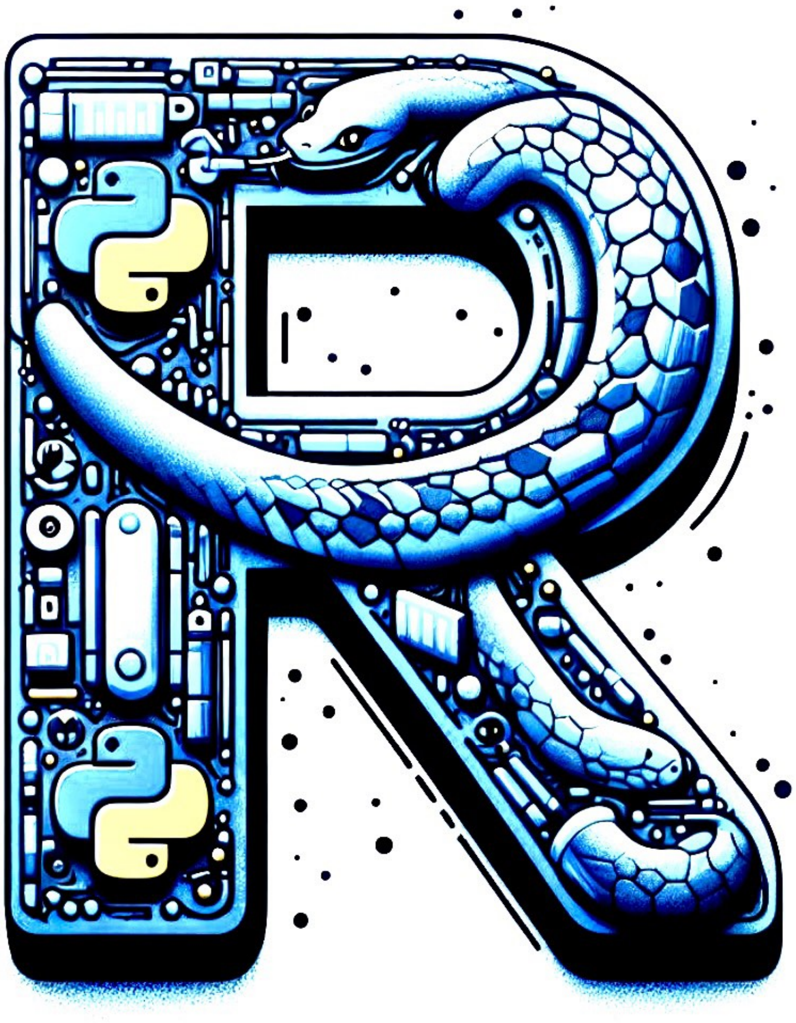
```
#heteroscedastic errors modeled
(res_wts[["b1.pval", "b2.pval"]] < alpha).mean()
```

```
## b1.pval    1.000
## b2.pval    0.953
## dtype: float64
```

While the above simulation confirms the commonly accepted consequences of heteroscedasticity, readers are encouraged to modify and test different setups and explore various methods of addressing heteroscedastic errors.

3.3 Consequences of Multicollinearity for OLS regression

Consequences of Multicollinearity for OLS regression



Andrija Djurovic[*]

[*] www.linkedin.com/in/andrija-djurovic

The absence of multicollinearity is another assumption of OLS regression. Simply put, multicollinearity occurs when two or more independent variables in a regression model are (highly) correlated with each other. The consequences of violating this assumption can be severe including imprecise estimates, a large variance in estimates, and instability in the signs of the estimates. Practitioners typically view multicollinearity as a problem only when observing instability in the estimates' signs. The reduction of precision is often overlooked, although it warrants further investigation and potential model adjustments based on expert inputs.

The following example illustrates some consequences of multicollinearity at different levels of intercorrelation between two independent variables.

Let's begin in `R` by defining simulation parameters and a helper function for the data-generating process.

```r
library(MASS)
library(dplyr)

#simulation parameters
n <- 45                               #number of observations
B <- 1000                             #simulation replicates
beta1 <- 0.50                         #beta1 estimate
beta2 <- 0.30                         #beta2 estimate
rho <- c(0, 0.25, 0.50, 0.75, 0.95)   #correlation between x1 and x2

#helper function for simulation of the betas
sim.betas <- function(n, beta1, beta2, rho) {

      #covariance matrix
      cm <- matrix(data = c(1, beta1, beta2,
                            beta1, 1, rho,
                            beta2, rho, 1),
                   nrow = 3,
                   ncol = 3)

      #simulate data
      db <- mvrnorm(n = n,
                    mu = c(0, 0, 0),
                    Sigma = cm)
      db <- data.frame(db)
      names(db) <- c("y", "x1", "x2")
      #run the regression
      ols.reg <- lm(formula = y ~ x1 + x2,
                    data = db)

      #extract the estimates
      betas <- summary(ols.reg)$coefficients[-1, 1]

return(betas)
}
```

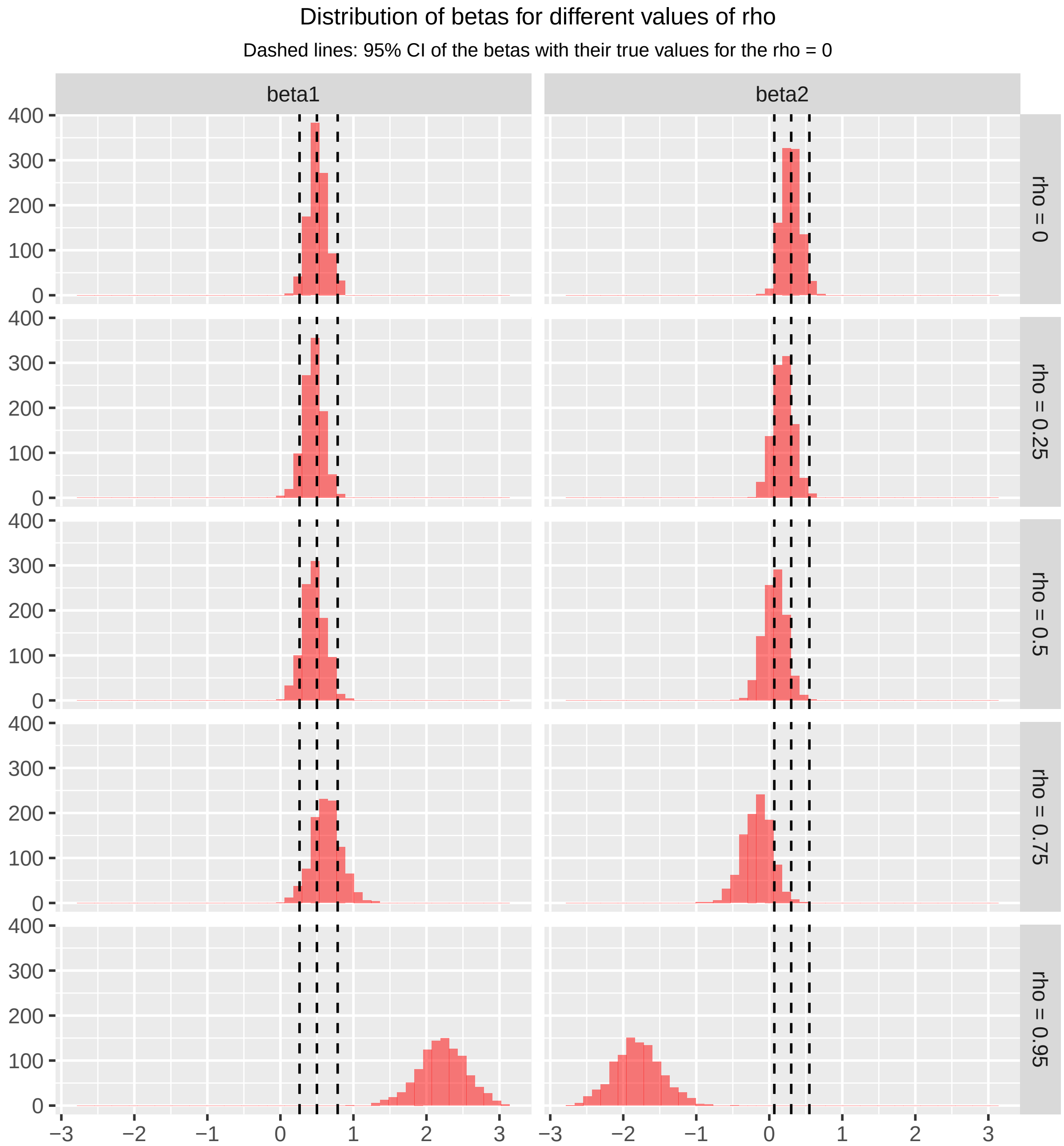
With the inputs prepared, we can now proceed to conduct the simulation.

```r
#-------------------------run the simulations-------------------------#
#empty object to store the results
res <- vector("list", B)

for   (i in 1:B) {

      #random seed
      set.seed(i)

      #simulation
      sim.i <- sapply(X = rho,
                      FUN = function(x) {
                            sim.betas(n = n,
                                      beta1 = beta1,
                                      beta2 = beta2,
                                      rho = x)
                            }
                      )

      #store the results
      res.i <- cbind.data.frame(simulation = i,
                                rho = rho,
                                data.frame(t(sim.i)))
      res[[i]] <- res.i
      }

#concatenate the results
res <- do.call("rbind", res)
```

```
#summarize the results
res %>%
group_by(rho) %>%
summarise(x1.low = quantile(x = x1, probs = 0.025),
          x1.cnt = quantile(x = x1, probs = 0.50),
          x1.upp = quantile(x = x1, probs = 0.975),
          x2.low = quantile(x = x2, probs = 0.025),
          x2.cnt = quantile(x = x2, probs = 0.50),
          x2.upp = quantile(x = x2, probs = 0.975),
          x1.std = sd(x1),
          x2.std = sd(x2)) %>%
as.data.frame()
```

```
##    rho    x1.low    x1.cnt    x1.upp      x2.low      x2.cnt     x2.upp
## 1 0.00 0.2622169 0.5024366 0.7842092  0.06852155  0.29413426  0.5491430
## 2 0.25 0.1874227 0.4511160 0.7123027 -0.07764253  0.18626673  0.4555672
## 3 0.50 0.1462334 0.4541766 0.7486144 -0.21831320  0.08039602  0.3835392
## 4 0.75 0.2452639 0.6292947 1.0465997 -0.57526176 -0.15771493  0.2144264
## 5 0.95 1.5086766 2.2185101 2.8365354 -2.44411043 -1.81307143 -1.1467552
##       x1.std    x2.std
## 1 0.1271422 0.1260787
## 2 0.1318679 0.1366010
## 3 0.1513431 0.1552601
## 4 0.2007366 0.1998318
## 5 0.3234497 0.3269169
```

To better understand the findings above, let's represent the results visually.



Concluding the exercise, we'll replicate the same simulation using `Python`.

```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf


#simulation parameters
```

```
n = 45                                    #number of observations
B = 1000                                  #simulation replicates
beta1 = 0.50                              #beta1 estimate
beta2 = 0.30                              #beta2 estimate
rho = [0, 0.25, 0.50, 0.75, 0.95]         #correlation between x1 and x2

#helper function for simulation of the betas
def sim_betas(n, beta1, beta2, rho):

    #covariance matrix
    cm = np.array([[1, beta1, beta2],
                   [beta1, 1, rho],
                   [beta2, rho, 1]])

    #simulate data
    db = np.random.multivariate_normal(mean = [0, 0, 0],
                                       cov = cm,
                                       size = n)
    db = pd.DataFrame(data = db,
                      columns = ["y", "x1", "x2"])

    #run the regression
    ols_reg = smf.ols(formula = "y ~ x1 + x2",
                      data = db).fit()

    #extract the estimates
    betas = ols_reg.params[1:]

    return betas

#-------------------------run the simulations-------------------------#
#empty object to store the results
res = [None]*B

#run the simulations
for i in range(1, B + 1):

    #set random seed
    np.random.seed(i)
    #simulate data for different rho values
    sim_results = [sim_betas(n, beta1, beta2, rho_i) for rho_i in rho]

    #store the results
    res_i = pd.DataFrame(data = sim_results)
    res_i["simulation"] = i
    res_i["rho"] = rho
    res[i - 1] = res_i

#concatenate the results
res = pd.concat(objs = res,
                ignore_index = True)

#summarize the results
res.groupby("rho").agg(
    x1_low = ("x1", lambda x: np.percentile(a = x, q = 2.5)),
    x1_cnt = ("x1", lambda x: np.percentile(a = x, q = 50)),
    x1_upp = ("x1", lambda x: np.percentile(a = x, q = 97.5)),
    x2_low = ("x2", lambda x: np.percentile(a = x, q = 2.5)),
    x2_cnt = ("x2", lambda x: np.percentile(a = x, q = 50)),
    x2_upp = ("x2", lambda x: np.percentile(a = x, q = 97.5)),
    x1_std = ("x1", lambda x: np.std(a = x, ddof = 1)),
    x2_std = ("x2", lambda x: np.std(a = x, ddof = 1))
    ).reset_index()
```

```
##     rho    x1_low    x1_cnt    x1_upp    x2_low    x2_cnt    x2_upp    x1_std  \
## 0  0.00  0.247497  0.503320  0.742984  0.049488  0.306543  0.550458  0.126296
## 1  0.25  0.189725  0.460893  0.720391 -0.077847  0.185654  0.462899  0.136916
## 2  0.50  0.163336  0.468381  0.783256 -0.227433  0.054696  0.372123  0.155673
## 3  0.75  0.238313  0.629706  1.050971 -0.581171 -0.174227  0.203122  0.200822
## 4  0.95  1.492846  2.201046  2.857388 -2.489935 -1.794232 -1.131327  0.340420
##
##      x2_std
## 0  0.127056
## 1  0.139141
## 2  0.155170
## 3  0.196688
## 4  0.334148
```

3.4 Consequences of Autocorrelation for OLS regression

Consequences of Autocorrelation for OLS regression

Andrija Djurovic[*]

[*] www.linkedin.com/in/andrija-djurovic

In an ideal regression model, it is assumed that the residuals are independent of one another, indicating the absence of any systematic pattern or correlation between observations. When there is a serial correlation among the residuals in OLS regression, we refer to it as an autocorrelation problem. An acknowledged consequence of autocorrelation is the generation of inefficient parameter estimates, thereby compromising the validity of statistical inference. In addition to this commonly accepted consequence, some practitioners argue that the presence of autocorrelated errors suggests model misspecification, omitted variable bias, or an incorrect functional form.

The following example illustrates a framework for exploring the implications of autocorrelation in OLS regression residuals. It is important to note that this provided example should not be construed as a broad generalization of autocorrelation consequences but rather as a foundation for tailoring the process to meet specific requirements and address unique use cases. Towards the conclusion of the exercise, potential adjustments to the presented design are mentioned.

To commence, let's establish the simulation parameters and the data-generating process in `R`.

```r
library(lmtest)
library(sandwich)

#simulation parameters
n <- 45              #number of observations
iv <- 2              #number of independent variables

#simulate the x
set.seed(123)
x <- replicate(n = iv,
               expr = rnorm(n = n)
               )
x <- data.frame(x)
names(x) <- paste0("x", 1:iv)

#true regression parameters
intercept <- 0
beta1 <- 0.50
beta2 <- -0.45

#simulate the target
y <- intercept + beta1*x$x1 + beta2*x$x2

#store the inputs into data frame
db <- data.frame(y = y, x)
```

Based on the provided inputs, we will conduct a simulation examining the potential bias resulting from correlated error terms. We will also compare the power based on the traditional p-value with one calculated using the correction of HAC (Heteroskedasticity-and-Autocorrelation-Consistent) standard errors. Additional simulations' assumptions include a significance level (`alpha`) set at 5% and an autoregressive coefficient (`phi`) fixed at 0.70.

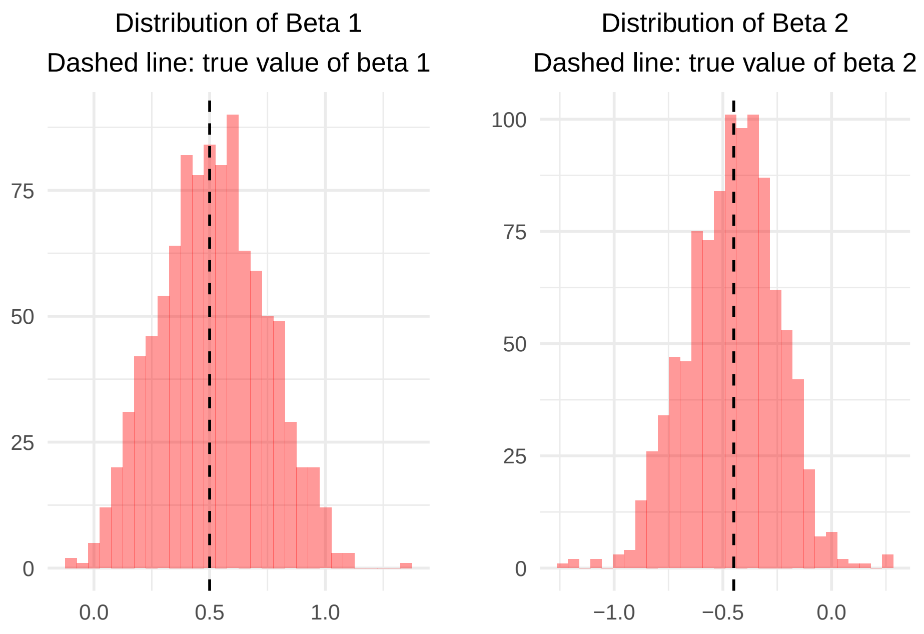
```
#-------------------------run the simulations-------------------------#
#mc parameters
alpha <- 0.05        #significance level
phi <- 0.70          #autocorrelation coefficient
B <- 1000            #number of mc simulations

res <- data.frame(b1 = rep(NA, B),
                  b2 = rep(NA, B),
                  b1.p = rep(NA, B),
                  b2.p = rep(NA, B),
                  b1.p.hac = rep(NA, B),
                  b2.p.hac = rep(NA, B))

for   (i in 1:B) {
      #random seed
      set.seed(i*3)

      #simulate the (centered) correlated errors
      error.i <- c(arima.sim(model = list(ar = phi),
                             n = n,
                             sd = 1))
      error.i <- error.i - mean(error.i)

      #simulate the target
      db$y.sim <- db$y + error.i

      #run the regression
      lr <- lm(formula = y.sim ~ x1 + x2,
               data = db)
      lr.s <- summary(lr)

      #extract the estimates and p-values
      est.p.i <-  c(lr.s$coefficients[-1, c(1, 4)])
      #store the results
      res[i, 1:4] <-  est.p.i

      #extract hac p-values
      hac.p <- coeftest(x = lr,
                        vcov. = NeweyWest(x = lr,
                                          lag = 1,
                                          prewhite = FALSE)
                        )

      #store the results
      res[i, 5:6] <- unname(hac.p[-1, 4])

      }
```

Following the simulation, let's plot the distribution of estimated betas alongside their true values. Furthermore, we perform a comparison of statistical power between two different approaches to calculate p-values.

```
#confidence intervals
ci.bs <- sapply(X = res[, c("b1", "b2")],
                FUN = function(x) {
                      quantile(x = x,
                               probs = c(0.025, 0.50, 0.975))
                      }
                )
ci.bs
```

```
##                b1          b2
## 2.5%  0.08698038 -0.85764567
## 50%   0.51242020 -0.44694846
## 97.5% 0.95819140 -0.08816721
```

```
#histogram of betas
```



```
#power calculation
p.bs <- sapply(X = res[, -c(1, 2)],
               FUN = function(x) mean(x < alpha))
p.bs
```

```
##     b1.p     b2.p b1.p.hac b2.p.hac
##    0.658    0.463    0.676    0.566
```

We will export the data generated in the above simulation to replicate the simulation in `Python`.

```
#export the db to replicate the simulation in python
write.csv(x = db[, -ncol(db)],
          file = "db.csv",
          row.names = FALSE)
```

The following code snippet demonstrates the `Python` implementation of the same exercise.

```
import pandas as pd
import numpy as np
import statsmodels.api as sm
import statsmodels.formula.api as smf

#import db
db = pd.read_csv("db.csv")

#------------------------run the simulations------------------------#
```

```
#mc parameters
n = db.shape[0]     #number of observations
alpha = 0.05        #significance level
phi = 0.70          #autocorrelation coefficient
B = 10000           #number of mc simulations

columns = ["b1", "b2", "b1.p", "b2.p", "b1.p.hac", "b2.p.hac"]
res = pd.DataFrame(index = range(1, B + 1),
                   columns = columns)

#arma simulation ar input
arparams = np.array([phi])
ar = np.r_[1, -arparams]

for i in range(1, B + 1):

    #random seed
    np.random.seed(i * 3)

    #simulate (centered) correlated errors
    error_i = sm.tsa.ArmaProcess(ar = ar).generate_sample(nsample = n)
    error_i = error_i - np.mean(error_i)

    #simulate the target
    db["y_sim"] = db["y"] + error_i

    #run the regression
    lr = smf.ols(formula = "y_sim ~ x1 + x2",
                 data = db).fit()

    #extract estimates and p-values
    est_p_i = lr.params.values[1:]
    p_values_i = lr.pvalues.values[1:]
    est_p = np.r_[est_p_i, p_values_i]

    #store the results
    res.loc[i, ["b1", "b2", "b1.p", "b2.p"]] = np.r_[est_p_i, p_values_i]

    #extract hac p-values
    hac = lr.get_robustcov_results(cov_type = "HAC", maxlags = 1)
    res.loc[i, ["b1.p.hac", "b2.p.hac"]] = hac.pvalues[1:]



#confidence intervals
ci_bs = res[["b1", "b2"]].apply(lambda x:
                                np.percentile(a = x,
                                              q = [alpha*100/2,
                                                   50,
                                                   (1 - alpha/2)*100]))
ci_bs
```

```
##          b1        b2
## 0  0.058771 -0.861513
## 1  0.499371 -0.448229
## 2  0.942962 -0.036201
```

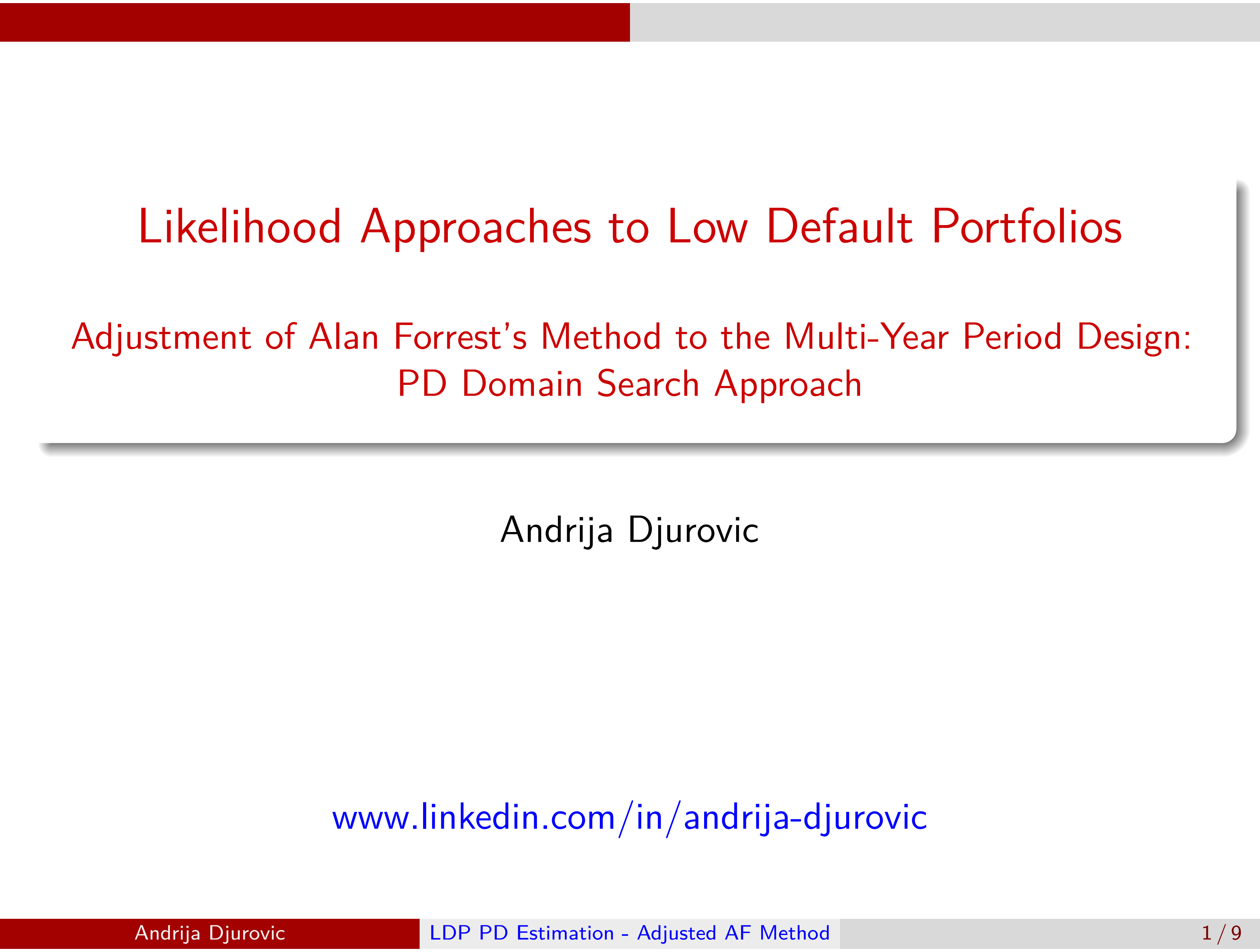
```
#power calculation
p_bs = (res[["b1.p", "b2.p", "b1.p.hac", "b2.p.hac"]] < alpha).mean()
p_bs
```

```
## b1.p        0.6463
## b2.p        0.4807
## b1.p.hac    0.6658
## b2.p.hac    0.5762
## dtype: float64
```

While the above simulations affirm the primary understanding of the implications of serially correlated errors, they should not be regarded as a universal generalization for all designs. In this context, readers are encouraged to explore additional designs, such as different values of the autocorrelation coefficient, incorporating lag of the dependent variable, experimenting with sample sizes for consistency checks, testing alternative methods for correcting autocorrelation, or investigating scenarios where HAC errors may surpass traditional ones. Additionally, one can explore combining autocorrelation errors with endogeneity problems.

4 Low Default Portfolios (LDP)

4.1 Likelihood Approaches to Low Default Portfolios - Andrija Djurovic's Adjustment of Alan Forrest's Method to the Multi-Year Period Design: PD Domain Search Approach



Likelihood Approaches to Low Default Portfolios

Adjustment of Alan Forrest's Method to the Multi-Year Period Design: PD Domain Search Approach

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | LDP PD Estimation - Adjusted AF Method | 1 / 9

Low Default Portfolio

- Qualitative descriptions of an Low Default Portfolio (LDP) leave ample room for interpretation, as different individuals may have varying opinions on whether a given portfolio qualifies as an LDP.
- The low number of defaults within a LDP undermines estimates' reliability and statistical validity for quantitative risk parameters based on historical default experience.
- LDPs are not necessarily low-data portfolios. The scarcity of defaults needs to be considered in relation to the size of the portfolio producing them.
- Regulators may be concerned that Probability of Default (PD) estimates based solely on simple historical averages or judgmental considerations may underestimate the bank's capital requirements due to default scarcity.
- Alan Forrest (AF) proposed Likelihood approaches for estimating the PD for the LDP. Compared to other methods, these approaches introduce an additional level of flexibility and utilize the data more efficiently.
- Despite the additional flexibility, comparing them with other methods is recommended.

Alan Forrest's Approach

- Compared to other approaches, AF proposed using the Likelihood and Likelihood Ratio instead of the probability of a class of data outcome. This change is crucial as it establishes a direct connection with the classical theory of statistical inference and its well-known approximations, which are valid for high default cases.
- The proposed method introduces additional flexibility and provides a framework for tailoring the approach to specific needs.
- In his paper, AF demonstrated the use of the likelihood approach for various cases, including single- and multiple-grade-level estimation with or without consideration of asset correlation and cases involving no or some defaults.
- Andrija Djurovic extended the proposed approach to one of the most prevalent designs in practice - portfolio-level multi-year period estimation, which considers asset and year-to-year correlation.
- The following slides describe the proposed adjustment to the multi-year design.

Adjustment of AF's Method to the Multi-Year Period Design

1. The change in the value of obligor i's assets over a year t is given by:

$$y_{i,t} = \sqrt{\rho} z_t + \sqrt{1-\rho}\epsilon_{i,t}$$

   where ρ is asset correlation, z and ϵ_i systemic and idiosyncratic factor, respectively.

2. Given the number of years (T) and year-to-year correlation (θ) we simulate the systemic factor for each year as follows:

$$z_{t,t\leq T} = \theta z_{t-1} + \sqrt{1-\theta^2}\epsilon_t$$

   where $z_{t,t=1}$ and ϵ_t are drawn from the standard normal distribution (`N[0, 1]`).

3. Using the $z_{t,t\leq T}$ we define the conditional PD_{c_t} for each year t, $t \leq T$ as follows:

$$PD_{c_t} = N\left[\frac{N^{-1}(PD) + z_t\sqrt{\rho}}{\sqrt{1-\rho}}\right]$$

   where PD comes from the range 0 and 1, and N and N^{-1} represent the distribution and quantile function of the standard normal distribution, respectively.

Adjustment of AF's Method to the Multi-Year Period Design cont.

4. For the multi-year period T, simulated $z_{t,t \leq T}$, and the PD that ranges from 0 to 1 we calculate the Likelihood as follows:

$$LL = \prod_{t}^{T} (PD_{c_t})^{d_t} (1 - PD_{c_t})^{n_t - d_t}$$

   where d_t and n_t represent the number of defaults and obligors at year t, respectively.
5. To obtain the Expected Likelihood for the selected PD, we repeat steps from 2 to 4 N times and calculate the average value of the N simulated Likelihoods.
6. Repeat step 5 for PD values ranging from 0 to 1 (or another selected upper threshold) to obtain the Likelihood values.
7. Given the Likelihoods from step 6, we calculate the Log-Likelihood Ratio as:

$$-2log\left(\frac{LL}{max(LL)}\right)$$

   where LL is the Expected Likelihood for different PDs.
8. Finally, calculate the upper bound of the interval for which Log-Likelihood Ratio is lower than a cutpoint defined as:

$$cp = \begin{cases} -2log(1 - cl) & : \sum_{i=t}^{T} d = 0 \\ \chi^2_{(p=cl, df=1)} & : \sum_{i=t}^{T} d > 0 \end{cases}$$

   where cl is selected confidence level and $\chi^2_{(q=cl, df=1)}$ is quantile of the χ^2 distribtuon for the probability p equal to cl and 1 degree of freedom.

Simulation Dataset

Number of obligors in each grade per year:

```
##   GRADE Y1 Y2 Y3 Y4 Y5
## 1     A  8  7  6  4  1
## 2     B 24 25 25 24 24
## 3     C 37 37 36 37 35
## 4     D 23 24 25 26 25
## 5     E  5  6  4  4  5
## 6     F  1  3  3  2  5
## 7     G  1  1  2  2  3
```

Number of defaults in each grade during the year:

```
##   GRADE Y1 Y2 Y3 Y4 Y5
## 1     A  0  0  0  0  0
## 2     B  0  0  0  0  0
## 3     C  0  0  0  0  0
## 4     D  0  0  0  0  0
## 5     E  1  0  0  0  0
## 6     F  0  1  0  0  0
## 7     G  0  0  0  1  1
```

Summary

Number of obligors per year: 99, 103, 101, 99, 98

Number of defaults per year: 1, 1, 0, 1, 1

Default rate per year: 0.0101, 0.0097, 0, 0.0101, 0.0102

Average default rate (Y1-Y5): 0.00802

R Code Extract

```
#...
#inputs
n <- c(99, 103, 101, 99, 98)    #number of obligors
d <- c(1, 1, 0, 1, 1)           #number of defaults
theta <- 0.30                   #year-to-year correlation
rho <- 0.12                     #asset correlation
N <- 1e4                        #number of simulations
cl <- 0.75                      #confidence level
pd.r <- seq(from = 0.000,
            to = 0.03,
            by = 0.0001)        #pd range
#random seed
set.seed(975)
#pd interval simulation
sim.res <- pd.interval(pd.r = pd.r,
                       n = n,
                       d = d,
                       rho = rho,
                       theta = theta,
                       cl = cl,
                       N = N)
#pd of the maximum likelihood
sim.res[["pd.mll"]]
```

```
## [1] 0.0106
```

```
#upper bound of the pd interval
max(sim.res[["pd.i"]])
```

```
## [1] 0.0212
```

Python Code Extract

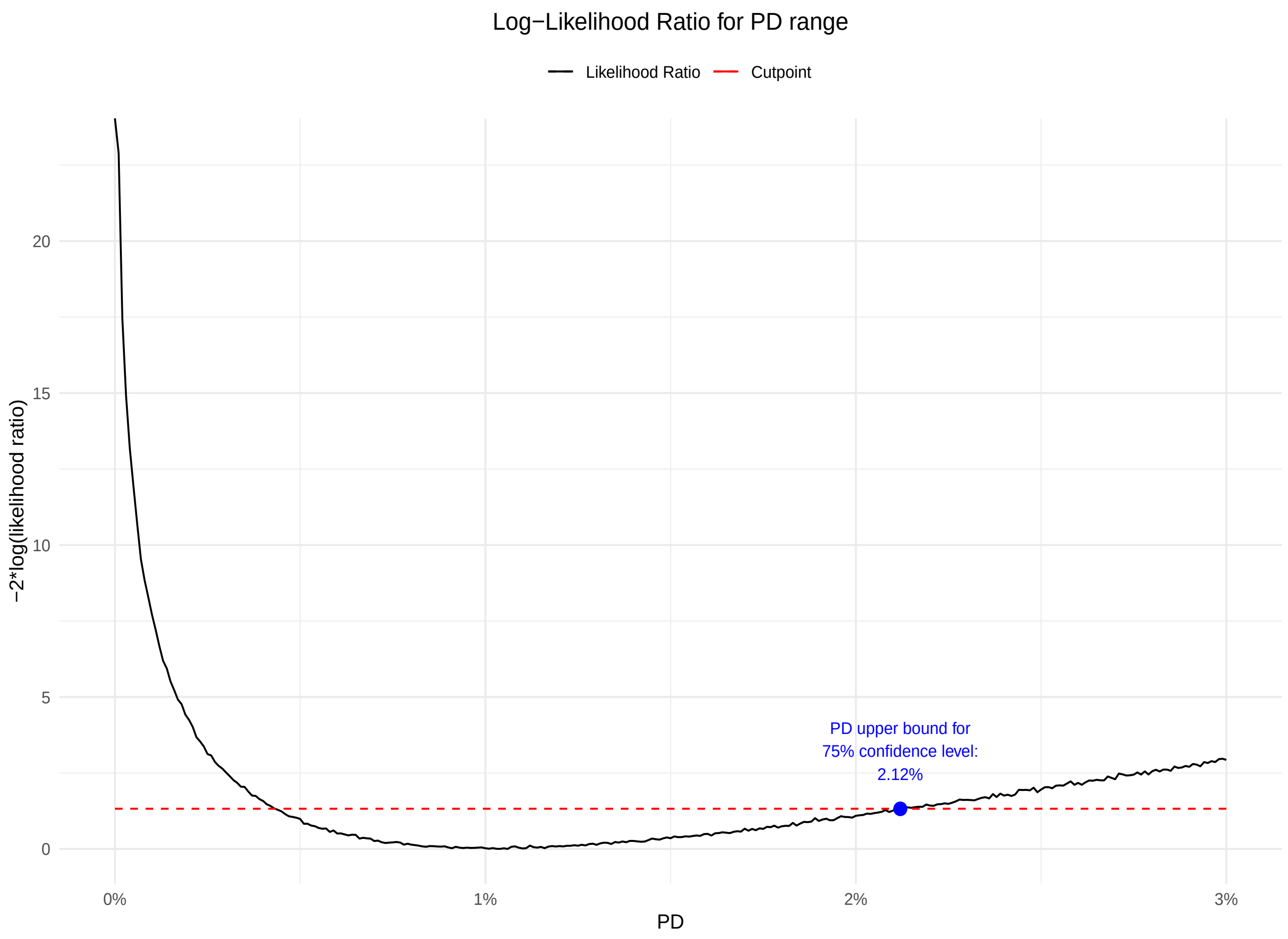
```
#...
#inputs
n = np.array([99, 103, 101, 99, 98])      #number of obligors
d = np.array([1, 1, 0, 1, 1])             #number of defaults
theta = 0.30                              #year-to-year correlation
rho = 0.12                                #asset correlation
N = int(1e4)                              #number of simulations
cl = 0.75                                 #confidence level
pd_r = np.arange(start = 0,
                 stop = 0.030001,
                 step = 0.0001)           #pd range
#random seed
np.random.seed(678)
#pd interval simulation
sim_res = pd_interval(pd_r = pd_r,
                      n = n,
                      d = d,
                      rho = rho,
                      theta = theta,
                      cl = cl,
                      N = N)
#pd of the maximum likelihood
sim_res["pd_mll"]
```

```
## 0.0108
```

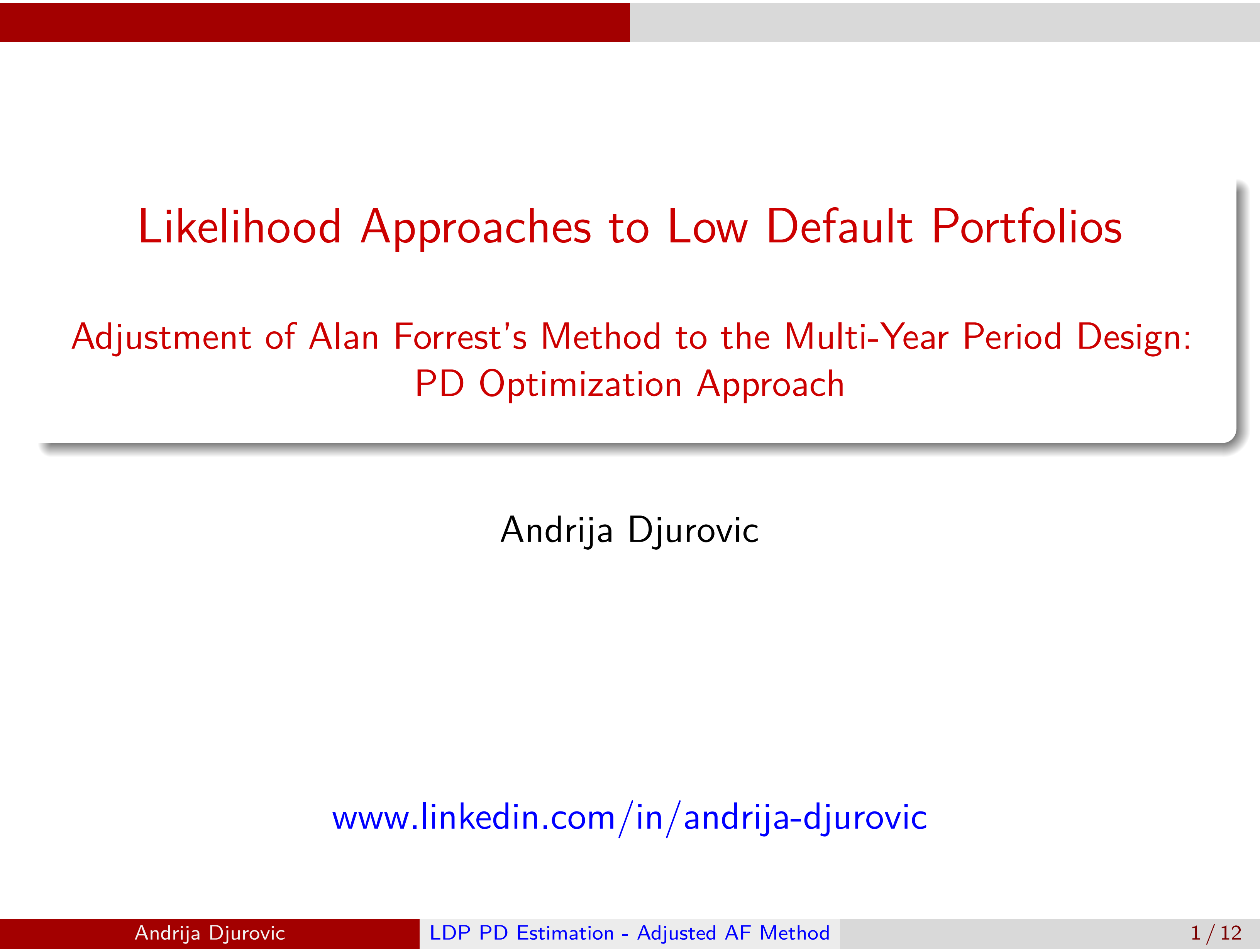
```
#upper bound of the pd interval
np.max(sim_res["pd_i"])
```

```
## 0.021400000000000002
```

Simulation Result Visualization



4.2 Likelihood Approaches to Low Default Portfolios - Andrija Djurovic's Adjustment of Alan Forrest's Method to the Multi-Year Period Design: PD Optimization Approach



Likelihood Approaches to Low Default Portfolios

Adjustment of Alan Forrest's Method to the Multi-Year Period Design:
PD Optimization Approach

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | LDP PD Estimation - Adjusted AF Method | 1 / 12

Low Default Portfolio

- Qualitative descriptions of an Low Default Portfolio (LDP) leave ample room for interpretation, as different individuals may have varying opinions on whether a given portfolio qualifies as an LDP.
- The low number of defaults within a LDP undermines estimates' reliability and statistical validity for quantitative risk parameters based on historical default experience.
- LDPs are not necessarily low-data portfolios. The scarcity of defaults needs to be considered in relation to the size of the portfolio producing them.
- Regulators may be concerned that Probability of Default (PD) estimates based solely on simple historical averages or judgmental considerations may underestimate the bank's capital requirements due to default scarcity.
- Alan Forrest (AF) proposed Likelihood approaches for estimating the PD for the LDP. Compared to other methods, these approaches introduce an additional level of flexibility and utilize the data more efficiently.
- Despite the additional flexibility, comparing them with other methods is recommended.

Alan Forrest's Approach

- Compared to other approaches, AF proposed using the Likelihood and Likelihood Ratio instead of the probability of a class of data outcome. This change is crucial as it establishes a direct connection with the classical theory of statistical inference and its well-known approximations, which are valid for high default cases.

- The proposed method introduces additional flexibility and provides a framework for tailoring the approach to specific needs.

- In his paper, AF demonstrated the use of the likelihood approach for various cases, including single- and multiple-grade-level estimation with or without consideration of asset correlation and cases involving no or some defaults.

- Andrija Djurovic extended the proposed approach to one of the most prevalent designs in practice - portfolio-level multi-year period estimation, which considers asset and year-to-year correlation.

- The following slides describe the proposed adjustment to the multi-year design. Unlike this presentation, which explains the same method using the PD domain search approach, this one defines the method as an optimization problem using the `nloptr` and `nlopt` packages in `R` and `Python`, respectively.

Adjustment of AF's Method to the Multi-Year Period Design

1. The change in the value of obligor i's assets over a year t is given by:

$$y_{i,t} = \sqrt{\rho} z_t + \sqrt{1 - \rho} \epsilon_{i,t}$$

   where ρ is asset correlation, z and ϵ_i systemic and idiosyncratic factor, respectively.

2. Given the number of years (T) and year-to-year correlation (θ) we simulate the systemic factor for each year as follows:

$$z_{t,t \leq T} = \theta z_{t-1} + \sqrt{1 - \theta^2} \epsilon_t$$

   where $z_{t,t=1}$ and ϵ_t are drawn from the standard normal distribution ($\mathtt{N[0,\ 1]}$).

3. Using the $z_{t,t \leq T}$ we define the conditional PD_{c_t} for each year t, $t \leq T$ as follows:

$$PD_{c_t} = N\left[\frac{N^{-1}(PD) + z_t \sqrt{\rho}}{\sqrt{1 - \rho}}\right]$$

   where PD comes from the range 0 and 1, and N and N^{-1} represent the distribution and quantile function of the standard normal distribution, respectively.

Adjustment of AF's Method to the Multi-Year Period Design cont.

4. For the multi-year period T, simulated $z_{t,t \leq T}$, and the PD that ranges from 0 to 1 we calculate the Likelihood as follows:

$$LL = \prod_{t}^{T} (PD_{c_t})^{d_t} (1 - PD_{c_t})^{n_t - d_t}$$

   where d_t and n_t represent the number of defaults and obligors at year t, respectively.
5. To obtain the Expected Likelihood for the selected PD, we repeat steps from 2 to 4 N times and calculate the average value of the N simulated Likelihoods.
6. Repeat step 5 for PD values ranging from 0 to 1 (or another selected upper threshold) to obtain the Likelihood values.
7. Given the Likelihoods from step 6, we calculate the Log-Likelihood Ratio as:

$$-2log\left(\frac{LL}{max(LL)}\right)$$

   where LL is the Expected Likelihood for different PDs.
8. Finally, calculate the upper bound of the interval for which Log-Likelihood Ratio is lower than a cutpoint defined as:

$$cp = \begin{cases} -2log(1 - cl) & : \sum_{i=t}^{T} d = 0 \\ \chi^2_{(p=cl, df=1)} & : \sum_{i=t}^{T} d > 0 \end{cases}$$

   where cl is selected confidence level and $\chi^2_{(q=cl, df=1)}$ is quantile of the χ^2 distribtuon for the probability p equal to cl and 1 degree of freedom.

Simulation Dataset

Number of obligors in each grade per year:

```
##   GRADE Y1 Y2 Y3 Y4 Y5
## 1     A  8  7  6  4  1
## 2     B 24 25 25 24 24
## 3     C 37 37 36 37 35
## 4     D 23 24 25 26 25
## 5     E  5  6  4  4  5
## 6     F  1  3  3  2  5
## 7     G  1  1  2  2  3
```

Number of defaults in each grade during the year:

```
##   GRADE Y1 Y2 Y3 Y4 Y5
## 1     A  0  0  0  0  0
## 2     B  0  0  0  0  0
## 3     C  0  0  0  0  0
## 4     D  0  0  0  0  0
## 5     E  1  0  0  0  0
## 6     F  0  1  0  0  0
## 7     G  0  0  0  1  1
```

Summary

Number of obligors per year: 99, 103, 101, 99, 98

Number of defaults per year: 1, 1, 0, 1, 1

Default rate per year: 0.0101, 0.0097, 0, 0.0101, 0.0102

Average default rate (Y1-Y5): 0.00802

R Code

```r
#source r script (packages and functions)
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/ldp/ldp_adj_af_adjustment_multi_year_opt.R")

#inputs
n <- c(99, 103, 101, 99, 98)    #number of obligors
d <- c(1, 1, 0, 1, 1)           #number of defaults
theta <- 0.30                   #year-to-year correlation
rho <- 0.12                     #asset correlation
N <- 1e4                        #number of simulations
cl <- 0.75                      #confidence level

#--------------------------------step 1: pd and maximum likelihood--------------------------------#
set.seed(6422)
mll.pd <- nloptr(x0 = mean(d/n),
                 eval_f = mll.f,
                 n = n,
                 d = d,
                 rho = rho,
                 theta = theta,
                 N = N,
                 lb = 0,
                 ub = 1,
                 opts = list("algorithm" = "NLOPT_GN_ORIG_DIRECT",
                             "xtol_rel = 1.0e-4",
                             "xtol_abs = 1.0e-4",
                             "ftol_rel" = 1.0e-4,
                             "ftol_abs" = 1.0e-4,
                             "maxeval" = 100))
mll.pd$solution
```

```
## [1] 0.0105929
```

R Code cont.

```
#expected value for the optimized pd
set.seed(6422)
mll <- ev.my(pd = mll.pd$solution,
             n = n,
             d = d,
             rho = rho,
             theta = theta,
             N = N)
mll
```

```
## [1] 0.0000000000226663
```

```
#------------------------------step 2: pd upper bound optimization------------------------------#
set.seed(21524)
ub <- nloptr(x0 = mean(d/n),
             eval_f = pd.ub,
             n = n,
             d = d,
             rho = rho,
             theta = theta,
             mll = mll,
             cl = cl,
             N = N,
             lb = mean(d/n),
             ub = 0.05,
             opts = list("algorithm" = "NLOPT_GN_ORIG_DIRECT",
                         "xtol_rel = 1.0e-4",
                         "xtol_abs = 1.0e-4",
                         "ftol_rel" = 1.0e-4,
                         "ftol_abs" = 1.0e-4,
                         "maxeval" = 100))
ub$solution
```

```
## [1] 0.02158985
```

Python Code

```
#source python script (packages and functions)
import requests
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/ldp/ldp_adj_af_adjustment_multi_year_opt.py"
r = requests.get(url)
exec(r.text)

#inputs
n = np.array([99, 103, 101, 99, 98])     #number of obligors
d = np.array([1, 1, 0, 1, 1])            #number of defaults
theta = 0.30                             #year-to-year correlation
rho = 0.12                               #asset correlation
N = int(1e4)                             #number of simulations
cl = 0.75                                #confidence level

#--------------------------------step 1: pd and maximum likelihood--------------------------------#
#define initial value
x0 = np.mean(d / n)
#create optimizer
mll_pd = nlopt.opt(nlopt.GN_ORIG_DIRECT, 1)
#set objective function
mll_pd.set_min_objective(lambda x,
                         grad: mll_f(x,
                                     grad = 0,
                                     n = n,
                                     d = d,
                                     rho = rho,
                                     theta = theta,
                                     N = N)
                         )
#set bounds
mll_pd.set_lower_bounds([0])
mll_pd.set_upper_bounds([1])
```

Python Code cont.

```
#set options
mll_pd.set_xtol_rel(1.0e-4)
mll_pd.set_xtol_abs(1.0e-4)
mll_pd.set_ftol_rel(1.0e-4)
mll_pd.set_ftol_abs(1.0e-4)
mll_pd.set_maxeval(100)
#perform optimization
np.random.seed(6422)
mll_solution = mll_pd.optimize([x0])
#return optimized solution
mll_solution
```

```
## array([0.0096784])
```

```
#expected value for the optimized pd
np.random.seed(6422)
mll = ev_my(pd = mll_solution,
            n = n,
            d = d,
            rho = rho,
            theta = theta,
            N = N)
mll
```

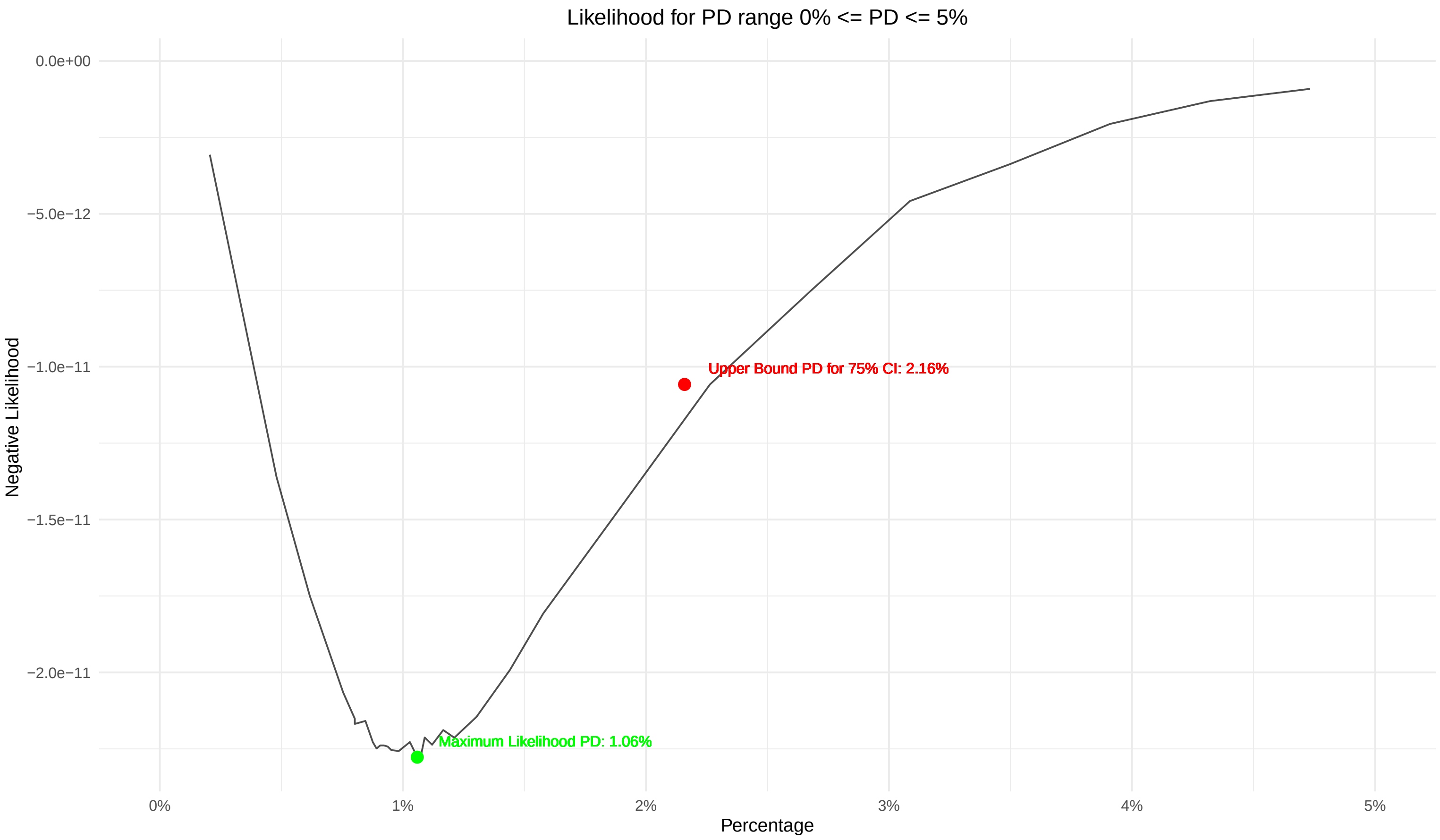
```
## 2.2611899726210518e-11
```

Python Code cont.

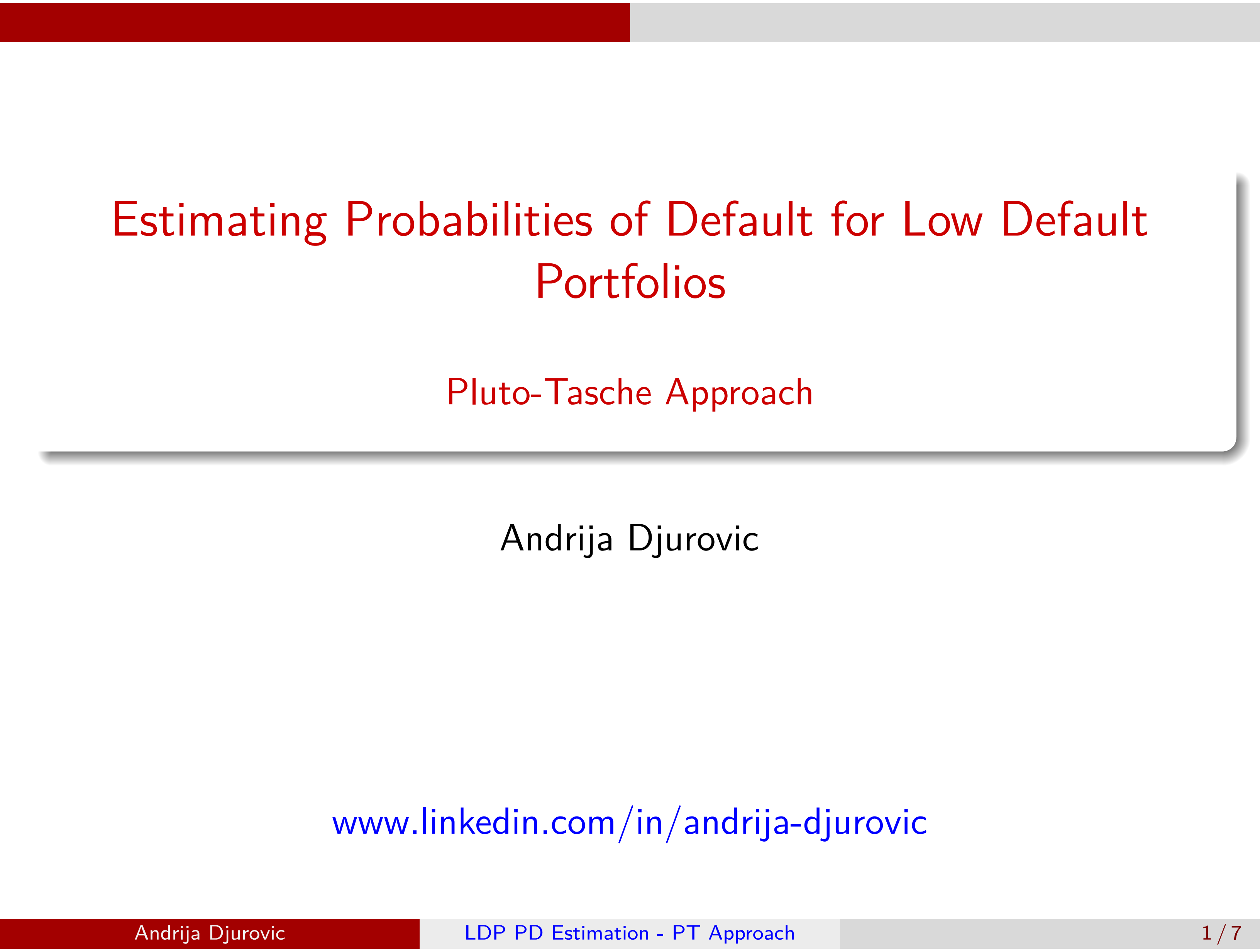
```python
#-------------------------------step 2: pd upper bound optimization-------------------------------#
#define initial value
x0 = np.mean(d / n)
#create optimizer
pd_ub_opt = nlopt.opt(nlopt.GN_ORIG_DIRECT, 1)
#set objective function
pd_ub_opt.set_min_objective(lambda x,
                         grad: pd_ub(x,
                                     grad = 0,
                                     n = n,
                                     d = d,
                                     rho = rho,
                                     theta = theta,
                                     mll = mll,
                                     cl = cl,
                                     N = N)
                         )
#set bounds
pd_ub_opt.set_lower_bounds([0])
pd_ub_opt.set_upper_bounds([0.05])
#set options
pd_ub_opt.set_xtol_rel(1.0e-4)
pd_ub_opt.set_xtol_abs(1.0e-4)
pd_ub_opt.set_ftol_rel(1.0e-4)
pd_ub_opt.set_ftol_abs(1.0e-4)
pd_ub_opt.set_maxeval(100)
#perform optimization
np.random.seed(6422)
pd_ub_opt_solution = pd_ub_opt.optimize([x0])
#return optimized solution
pd_ub_opt_solution
```

```
## array([0.02149444])
```

Simulation Result Visualization



4.3 Estimating Probabilities of Default for Low Default Portfolios - Pluto-Tasche Approach



Estimating Probabilities of Default for Low Default Portfolios

Pluto-Tasche Approach

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Low Default Portfolio

- Qualitative descriptions of an Low Default Portfolio (LDP) leave ample room for interpretation, as different individuals may have varying opinions on whether a given portfolio qualifies as an LDP.
- The low number of defaults within a LDP undermines estimates' reliability and statistical validity for quantitative risk parameters based on historical default experience.
- LDPs are not necessarily low-data portfolios. The scarcity of defaults needs to be considered in relation to the size of the portfolio producing them.
- Regulators may be concerned that Probability of Default (PD) estimates based solely on simple historical averages or judgmental considerations may underestimate the bank's capital requirements due to default scarcity.
- The Pluto-Tasche (PT) approach is one way to address the estimation of PD for the LDP.
- Alternative methods are also available and warrant examination and comparison with the PT approach. For details, practitioners can refer to the adjustment of the Alan Forrest and the BCR methods.

Pluto-Tasche Proposal for the PDs Estimation

The following steps outline the data preparation process for running the PT PD estimation for the multi-year design at the grade level, which accounts for asset correlation and year-to-year correlation:

1. Identify the LDP.
2. Define the historical sample for the LDP (multi-year period) using internal data.
3. The sample should specify each obligor's grade at the start of each year within the historical period.
4. Aggregate multi-year data by obligor rating, including the number of obligors and defaults.
5. Select the algorithm inputs: asset correlation, year-to-year correlation, confidence level, and number of simulations.
6. Run the PT estimation process for each rating grade.
7. (Optional) Rescale the estimated PDs to match the target portfolio's central tendency.

More details on the method and data preparation are available here.

The following slides briefly describe the Monte Carlo process used for PD estimation and present results from the simulation study based on the same dataset as the other two methods mentioned in the previous slide. For comparison, practitioners can refer to these presentations: Adjustment of the Alan Forrest and BCR.

Pluto-Tasche Method - Upper Bound of the Portfolio PD

1. The change in the value of obligor i's assets over a year t is given by:

$$y_{i,t} = \sqrt{\rho} z_t + \sqrt{1-\rho}\epsilon_{i,t}$$

where ρ is asset correlation, z and ϵ_i systemic and idiosyncratic factor, respectively.
The obligor defaults if the asset value is lower than the value of c for the specific year. Given the above assumptions, we are interested in the probability (PD):

$$P(y_{i,t} \leq c) = PD_{i,t} = PD$$

which describes the long-term average 1-year probability of default among the obligors that have not defaulted before and may be considered a through-the-cycle PD.

2. Given the number of years (T) and year-to-year correlation (θ) we simulate the systemic factor for each year as follows:

$$z_{t,t \leq T} = \theta z_{t-1} + \sqrt{1-\theta^2}\epsilon_t$$

where $z_{t,t=1}$ and ϵ_t are drawn from the standard normal distribution (`N[0, 1]`).

3. Using the $z_{t,t \leq T}$ we define the conditional PD (PD_{c_t}) for each year $t, t \leq T$ as follows:

$$PD_{c_t} = N\left[\frac{N^{-1}(PD) - z_t\sqrt{\rho}}{\sqrt{1-\rho}}\right]$$

where N and N^{-1} represent the distribution and quantile function of the normal standard distribution, respectively.

Pluto-Tasche Method - Upper Bound of the Portfolio PD cont.

4. For the multi-year period, we calculate the cumulative *PD* as follows:

$$PD = 1 - \prod_{t=1}^{T} (1 - PD_{c_t})$$

5. Given the total number of obligors *n* up to the selected rating, the total number of defaults *k* also up to the selected rating, and the cumulative probability of default *PD*, the likelihood of observing no more than *k* defaults among *n* obligors is calculated as:

$$1 - \gamma = \sum_{i=0}^{k} \binom{n}{i} PD^{i} (1 - PD)^{n-i}$$

   where γ denotes the selected confidence level.

6. Finally, we determine the upper bound of the rating PD through numerical optimization for *PD* using Monte Carlo simulations to minimize the difference between the average simulated and the chosen confidence level (γ).

Simulation Study - R Code

```r
#source r script
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/ldp/pt.R")

#inputs
n <- c(26, 122, 182, 123, 24, 14, 9)        #number of obligors per rating
k <- c(0, 0, 0, 0, 1, 1, 2)                 #number of defaults per rating
theta <- 0.30                               #year-to-year correlation
rho <- 0.12                                 #asset correlation
T <- 5                                      #number of years
cl <- 0.75                                  #confidence level
N <- 1e4                                    #number of simulations

#pluto-tasche pd estimates
pd <- ldp.pt(n = n,
             k = k,
             theta = theta,
             rho = rho,
             T = T,
             cl = cl,
             N = N,
             seed = 123)

#upper bound of the pd
sprintf("%.2f%%", 100 * pd)
```

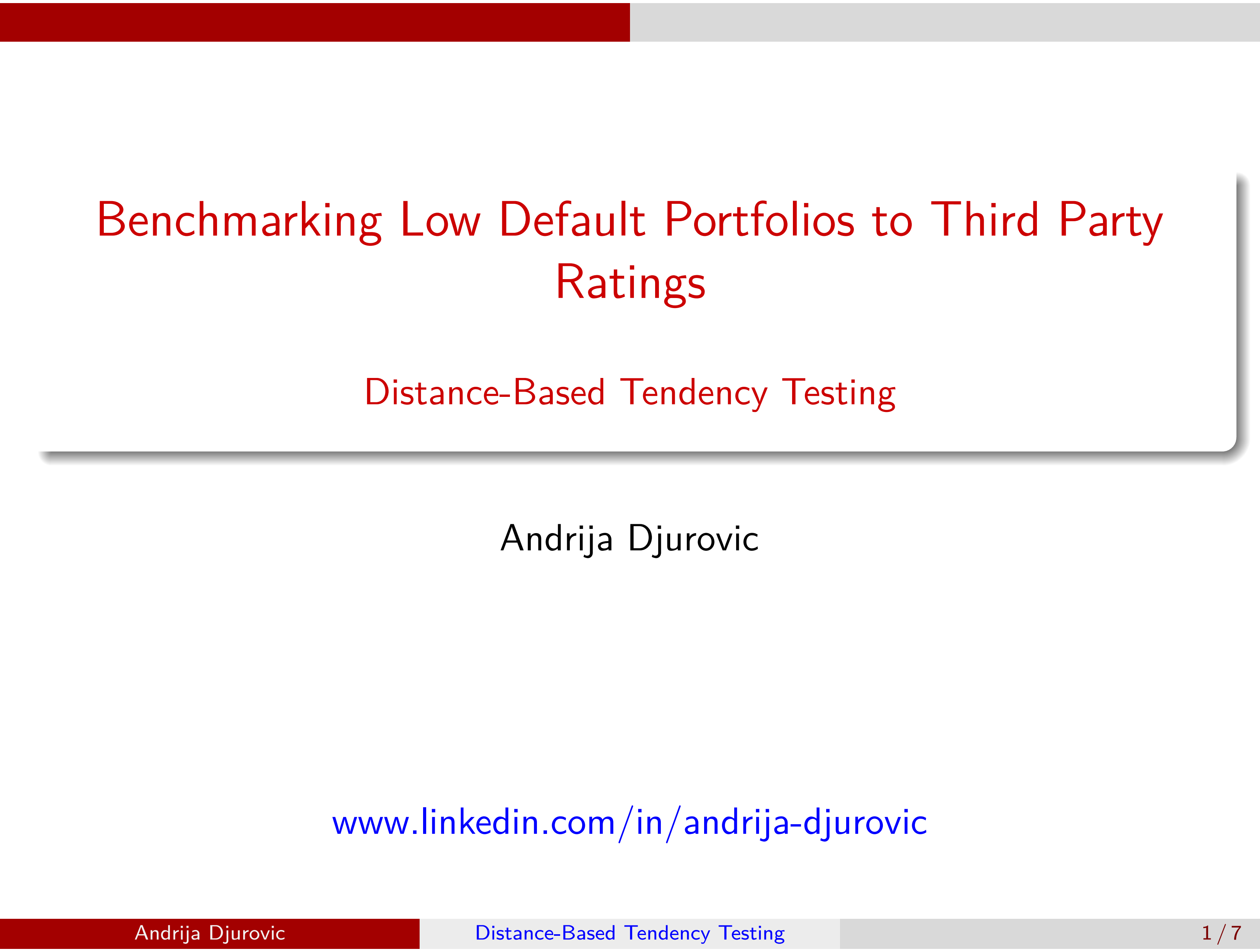
```
## [1] "0.38%"  "0.40%"  "0.53%"  "1.03%"  "3.43%"  "5.51%"  "10.44%"
```

Simulation Study - Python Code

```python
#source python script
import requests
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/ldp/pt.py"
r = requests.get(url)
exec(r.text)

#inputs
n = [26, 122, 182, 123, 24, 14, 9]    #number of obligors per rating
k = [0, 0, 0, 0, 1, 1, 2]             #number of defaults per rating
theta = 0.30                          #year-to-year correlation
rho = 0.12                            #asset correlation
T = 5                                 #number of years
cl = 0.75                             #confidence level
N = int(1e4)                          #number of simulations

#pluto-tasche pd estimates
pd = ldp_pt(n = n,
            k = k,
            theta = theta,
            rho = rho,
            T = T,
            cl = cl,
            N = N,
            seed = 123)

#upper bound of the pd
[f"{round(100 * value, 2):.2f}%" for value in pd]
```

```
## ['0.38%', '0.40%', '0.53%', '1.03%', '3.45%', '5.50%', '10.46%']
```

4.4 Benchmarking Low Default Portfolios to Third Party Ratings - Distance-Based Tendency Testing



Benchmarking Low Default Portfolios to Third Party Ratings

Distance-Based Tendency Testing

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Low Default Portfolios Benchmarks

- Validating Low Default Portfolios (LDP) presents significant challenges to practitioners, as most statistical tests applicable to high default portfolios do not have equivalents for low default portfolios.
- Therefore, practitioners often rely on a benchmarking approach to validate certain types of LDP where external ratings are available.
- Benchmarking helps assess how well the rating system distinguishes between low- and high-risk counterparties and whether the average risk assessment significantly differs from an independent observer's.
- It also evaluates whether deviations in ratings are unreasonably large.
- The drawbacks of benchmarking include that external ratings are typically available for only part of the portfolio, which may not represent the entire portfolio. Unlike default prediction, deviations from external ratings are not necessarily incorrect. Therefore, considering factors such as differences in rating and default definitions, careful analysis and detailed interpretation of benchmarking results are essential.
- A fundamental assumption of benchmarking is confidence in the correctness of external ratings. Accordingly, selecting the relevant type of external rating requires great care and an understanding of potential discrepancies.
- This presentation focuses on distance-based tendency testing.
- More on LDP benchmarking can be found in the following reference: Franz, C., & Lawrenz, C. (2007). Benchmarking low-default portfolios to third-party ratings. Journal of Credit Risk, 3(2), 87–100.

Distance-Based Tendency Testing

Tendency is one of the measures calculated based on the distances between internal and external ratings.

It evaluates the systematic bias in internal ratings compared to external ratings by categorizing deviations as optimistic, neutral, or pessimistic.

A prerequisite for testing tendency and other distance-based benchmarking procedures is the availability of a mapping process that translates internal ratings to external ratings.

The basis for the tendency calculation can be expressed as:

$$p_{-1} = \frac{N_{-1}}{N}, \quad p_0 = \frac{N_0}{N}, \quad p_{+1} = \frac{N_{+1}}{N}$$

where:

- N_{-1} is the count of debtors where the internal rating is worse than the external rating;
- N_0 is the count where internal and external ratings are equal;
- N_{+1} is the count where the internal rating is better than the external rating;
- N is the total number of debtors.

Distance-Based Tendency Testing cont.

Simultaneous confidence intervals for these proportions at a selected level α can be estimated using multinomial distribution approaches, such as Goodman's method:

$$[p_i^l, p_i^u] = \left[\frac{B_i - \sqrt{B_i^2 - 4AN_i^2/N}}{2A}, \frac{B_i + \sqrt{B_i^2 - 4AN_i^2/N}}{2A} \right]$$

where:

- A is defined as $\chi^2(1 - \frac{\alpha}{3}) + N$;
- B is defined as $\chi^2(1 - \frac{\alpha}{3}) + 2N_i$;
- with χ^2 is the quantile of the chi-square distribution for confidence level α with one degree of freedom.

Simulation Study

- The following slides provide `R` and `Python` code for testing tendency.
- The simulated dataset consists of internal and external ratings, the only inputs needed for this calculation.
- The dataset is available here.
- In addition to the proposed simultaneous confidence intervals for the multinomial proportions, practitioners should be aware that similar testing can be conducted using a test of two proportions, directly comparing p_{-1} and p_{+1} while disregarding the overlapping samples.

R Code

```r
#utils function import (mltnp.ci)
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/utils_benchmarking_ldp.R")

#data import
url <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/ldp_benchmarking.csv"
db <- read.csv(file = url,
               header = TRUE)

#tendency indicator
tendency <- ifelse(db$internal > db$external, 1,
            ifelse(db$internal < db$external, -1, 0))

#frequency table
table(tendency)
```

```
## tendency
##  -1   0   1 
## 349 173 478
```

```r
#tendency confidence intervals
mltnp.ci(tendency = tendency,
         cl = 0.95)
```

```
##      est     lower     upper
## -1 0.349 0.3138684 0.3858525
## 0  0.173 0.1462494 0.2034774
## 1  0.478 0.4404176 0.5158332
```

Python Code

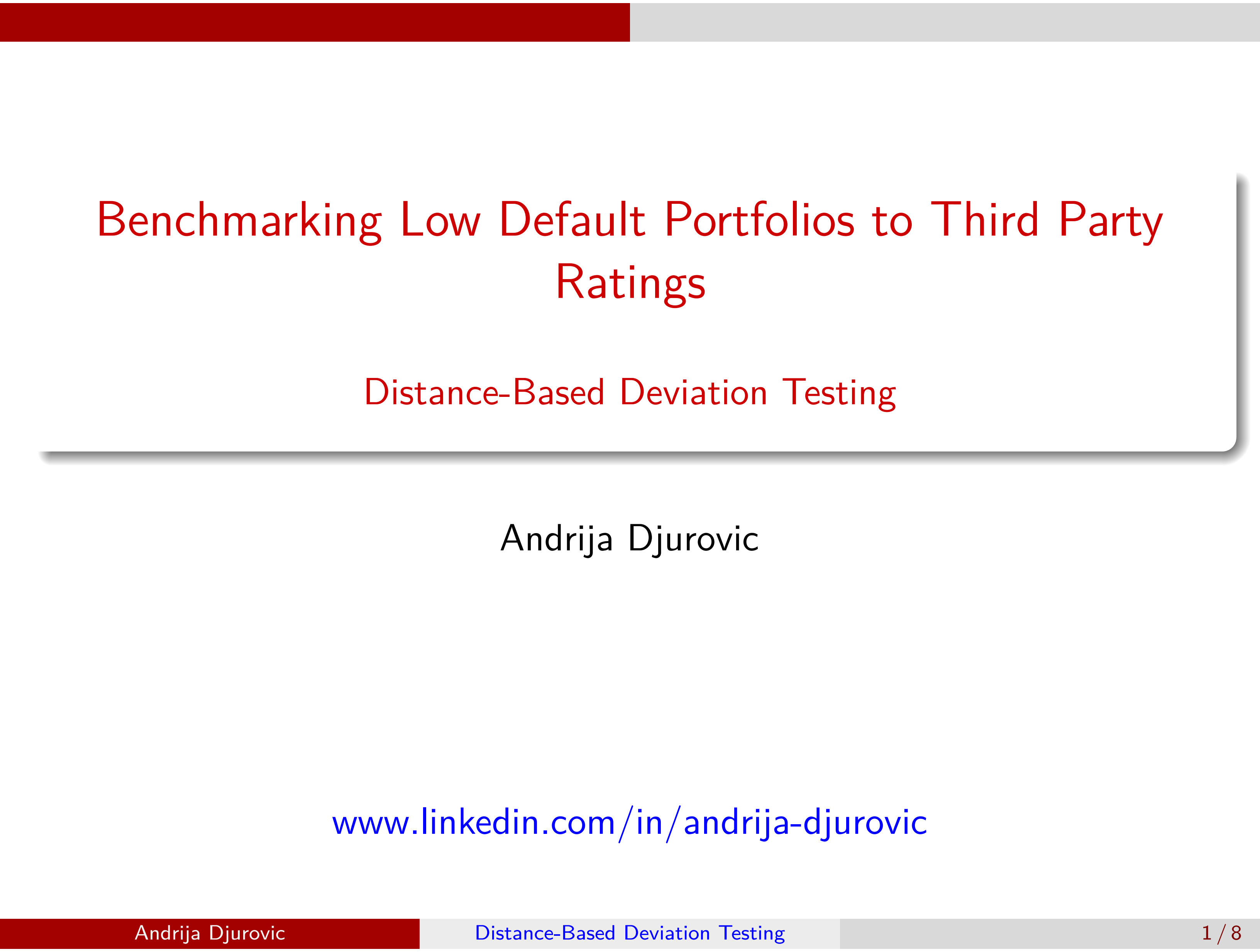
```
import pandas as pd
import numpy as np
from scipy import stats
import requests

#utils function import (mltnp.ci)
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/utils_benchmarking_ldp.py"
r = requests.get(url)
exec(r.text)

#data import
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/ldp_benchmarking.csv"
db = pd.read_csv(filepath_or_buffer = url)

#tendency indicator
tendency = np.where(db["internal"] > db["external"], 1,
           np.where(db["internal"] < db["external"], -1, 0))

#frequency table
np.unique(ar = tendency,
          return_counts = True)
```

```
## (array([-1,  0,  1]), array([349, 173, 478], dtype=int64))
```

```
#tendency confidence intervals
mltnp_ci(tendency = tendency,
         cl = 0.95)
```

```
##       est     lower     upper
## -1  0.349  0.313868  0.385853
##  0  0.173  0.146249  0.203477
##  1  0.478  0.440418  0.515833
```

4.5 Benchmarking Low Default Portfolios to Third Party Ratings - Distance-Based Deviation Testing



Benchmarking Low Default Portfolios to Third Party Ratings

Distance-Based Deviation Testing

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Distance-Based Deviation Testing | 1 / 8

Low Default Portfolios Benchmarks

- Validating Low Default Portfolios (LDP) presents significant challenges to practitioners, as most statistical tests applicable to high default portfolios do not have equivalents for low default portfolios.
- Therefore, practitioners often rely on a benchmarking approach to validate certain types of LDP where external ratings are available.
- Benchmarking helps assess how well the rating system distinguishes between low- and high-risk counterparties and whether the average risk assessment significantly differs from an independent observer's.
- It also evaluates whether deviations in ratings are unreasonably large.
- The drawbacks of benchmarking include that external ratings are typically available for only part of the portfolio, which may not represent the entire portfolio. Unlike default prediction, deviations from external ratings are not necessarily incorrect. Therefore, considering factors such as differences in rating and default definitions, careful analysis and detailed interpretation of benchmarking results are essential.
- A fundamental assumption of benchmarking is confidence in the correctness of external ratings. Accordingly, selecting the relevant type of external rating requires great care and an understanding of potential discrepancies.
- This presentation focuses on distance-based deviation testing.
- More on LDP benchmarking can be found in the following reference: Franz, C., & Lawrenz, C. (2007). Benchmarking low-default portfolios to third-party ratings. Journal of Credit Risk, 3(2), 87–100.

Distance-Based Deviation Testing

A commonly used measure alongside the tendency measure is average deviation.

Deviation evaluates the magnitude of differences between internal and external ratings, providing insight into rating volatility and calibration quality.

The most straightforward measure of calibration quality is average deviation, calculated as:

$$D = \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i)$$

where:

- y_i and x_i are the internal and external ratings for obligor i, respectively;
- N is total number of obligros.

The absolute deviation calculation can often supplement the simple average deviation.

Practitioners can use bootstrapping to obtain the confidence interval for deviation. Unlike the basic bootstrapping method, deriving the confidence interval for deviation typically involves certain assumptions. One key assumption is that external ratings are correct, allowing specific observed distributions to remain fixed. Another practical approach that can be applied for the same purpose is based on convolutions.

Simulation Study

- The following slides present `R` and `Python` code for calculating average deviation and its confidence interval.
- The dataset is available here.
- The simulated dataset consists of internal and external ratings, the only inputs needed for this calculation.
- The presented example uses convolutions to calculate the confidence interval for average deviation. In this specific case, the assumption is that the observed distribution for each unique deviation remains constant.
- Practitioners are encouraged to modify the presented framework, adjust it to their needs, and compare the results under different assumptions.

R Code

```r
#utils function import (d.ci)
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/utils_benchmarking_ldp.R")

#data import
url <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/ldp_benchmarking.csv"
db <- read.csv(file = url,
               header = TRUE)

#convert ratings to factors
db$external <- factor(x = db$external ,
                      levels = sort(unique(db$external)),
                      order = TRUE)
db$internal <- factor(x = db$internal,
                      levels = sort(unique(db$external)),
                      order = TRUE)

#calculate deviations
deviation <- as.numeric(db$internal) - as.numeric(db$external)

#frequency table
table(deviation)
```

```
## deviation
##  -4  -3  -2  -1   0   1   2   3   4   5   6
##  18  55 105 171 173 173 144 116  32  10   3
```

```r
#average deviation confidence interval
d.ci(deviation = deviation,
     cl = 0.95)
```

```
##     est lower upper
## 1 0.387 0.265 0.509
```

Python Code

```python
import pandas as pd
import numpy as np
from scipy import stats
import requests

#utils function import (d_ci)
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/utils_benchmarking_ldp.py"
r = requests.get(url)
exec(r.text)

#data import
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/ldp/ldp_benchmarking.csv"
db = pd.read_csv(filepath_or_buffer = url)

#convert ratings to factors
external_levels = np.sort(db["external"].unique())
db["external"] = pd.Categorical(db["external"],
                                categories = external_levels,
                                ordered = True)
db["internal"] = pd.Categorical(db["internal"],
                                categories = external_levels,
                                ordered = True)
```

Python Code cont.

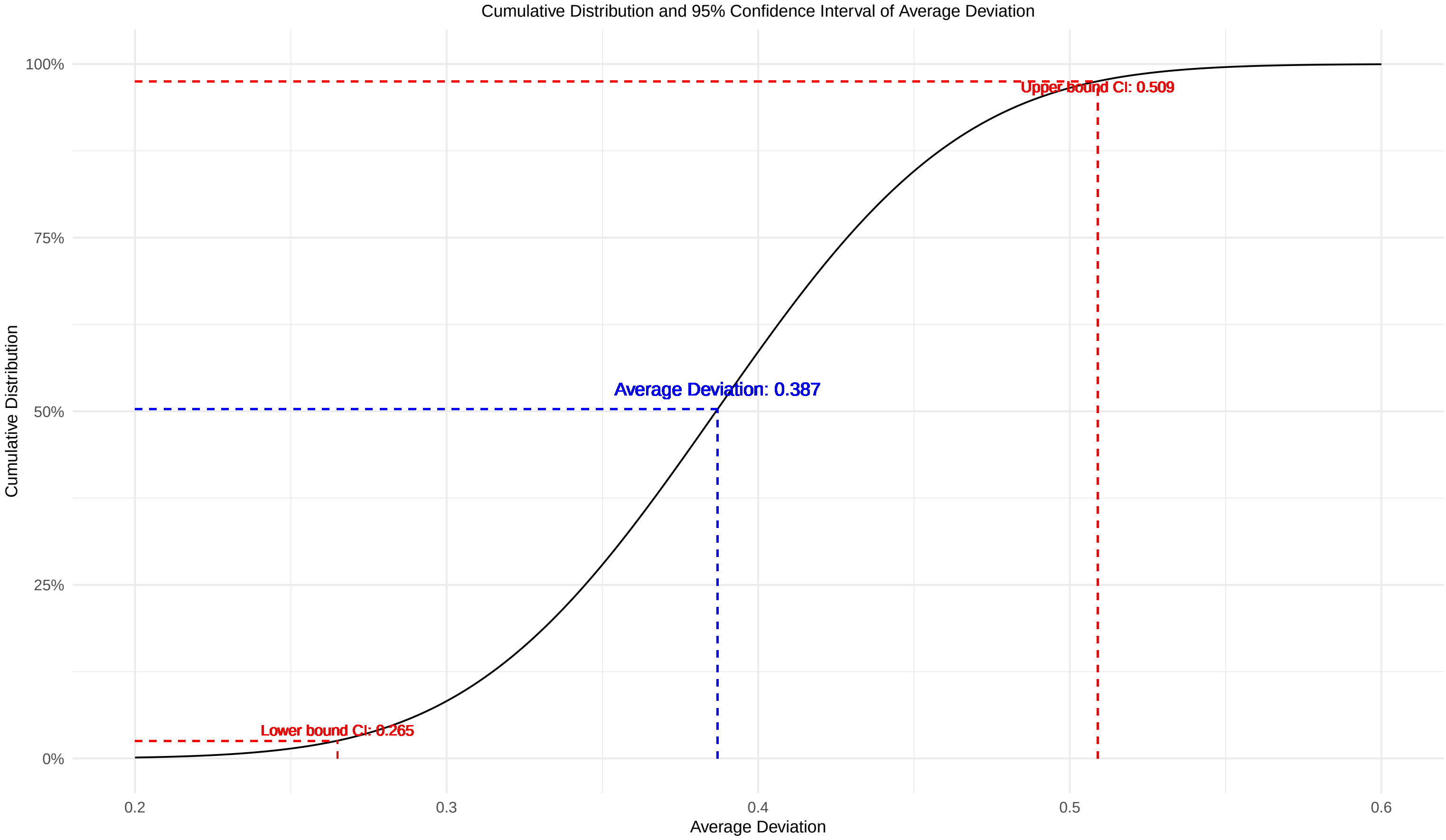
```
#calculate deviations
deviation = (db["internal"].cat.codes + 1) - (db["external"].cat.codes + 1)

#frequency table
deviation.value_counts().sort_index()
```

```
## -4     18
## -3     55
## -2    105
## -1    171
##  0    173
##  1    173
##  2    144
##  3    116
##  4     32
##  5     10
##  6      3
## Name: count, dtype: int64
```

```
#average deviation confidence interval
d_ci(deviation = np.array(deviation),
     cl = 0.95)
```

```
##      est  lower  upper
## 0  0.387  0.265  0.509
```

Simulation Results - Summary



5 Concentration Risk

5.1 Measuring Concentration Risk - A Partial Portfolio Approach

Measuring Concentration Risk - A Partial Portfolio Approach

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic Concentration Risk 1 / 8

Concentration Risk

- Concentration risk represents a significant source of risk in banking, particularly in emerging and small economies.
- Pillar 1 capital requirements don't cover concentration risk.
- Banks must autonomously estimate and set aside capital buffers to mitigate concentration risk, usually as part of Pillar 2 models.
- Inadequate recognition of concentration risk can lead to insufficient capital levels, even with apparently high capital ratios.
- A partial portfolio approach as a way to address the concentration risk and to complement the existing regulatory capital requirements (details available here).

A Partial Portfolio Approach (PPA)

Assuming a portfolio of `n` exposures with a non-granular sub-portfolio of `m` exposures (`m <= n`), the PPA can be outlined in the following steps after determining the number of simulations (`B`):

1. Simulate a value for the systemic risk factor `z` from the standard normal distribution (`N[0, 1]`).
2. For each exposure in the non-granular sub-portfolio, simulate the idiosyncratic factor ϵ from the standard normal distribution (`N[0, 1]`).
3. Given the simulated systemic, idiosyncratic factor and calculated asset correlation, for each exposure in the non-granular sub-portfolio, simulate the asset return value as:

   $$y_i = \sqrt{\rho} z + \sqrt{1 - \rho \epsilon_i}$$

   where ρ is asset correlation, z and ϵ_i systemic and idiosyncratic factor, respectively.
4. For each exposure in the non-granular sub-portfolio, simulate the default indicator as follows:

   $$I_i = y_i < N^{-1}(\overline{PD_i})$$

   where y_i is the asset return (step 3), $N^{-1}(\overline{PD_i})$ is the quantile of the standard normal variable, and $\overline{PD_i}$ is unconditional Probability of Default for exposure i.

A Partial Portfolio Approach (PPA) cont.

5. Calculate the loss for the non-granular sub-portfolio as

$$Loss^{non-granular} = \sum_{i=1}^{m} I_i LGD_i EAD_i$$

   where I_i is the indicator from the step 4, and LGD_i and EAD_i Loss Given Default and Exposure at Default, respectively.

6. Calculate the loss for the granular sub-portfolio as:

$$Loss^{granular} = \sum_{i=1}^{n-m} N\left[\frac{N^{-1}(\overline{PD_i}) - z\sqrt{\rho}}{\sqrt{1-\rho}}\right] LGD_i EAD_i$$

   where N and N^{-1} present standard normal cumulative distribution and quantile function.

7. Sum up losses from the non-granular and granular portfolio.
8. After repeating steps from 1 to 7 selected B times, calculate the 99.9 percentile of the loss distribution and subtract the expected loss to get the required Credit VaR.

Simulation Setup

Simulation dataset available here.

R:

```r
#lgd
lgd <- 0.30
#asset correlation
rho <- 0.05
#ead threshold
seq.ft <- c(seq(from = 0,
                to = 1.5,
                length.out = 100),
            10) / 100
#number of simulations
B <- 100000
#confidence level
cl <- 0.999
```

Python:

```python
#lgd
lgd = 0.30
#asset correlation
rho = 0.05
#ead threshold
seq_ft = np.concatenate([np.linspace(start = 0,
                                     stop = 1.5,
                                     num = 100),
                         [10]]) / 100
#number of simulations
B = 100000
#confidence level
cl = 0.999
```

R Code Extract

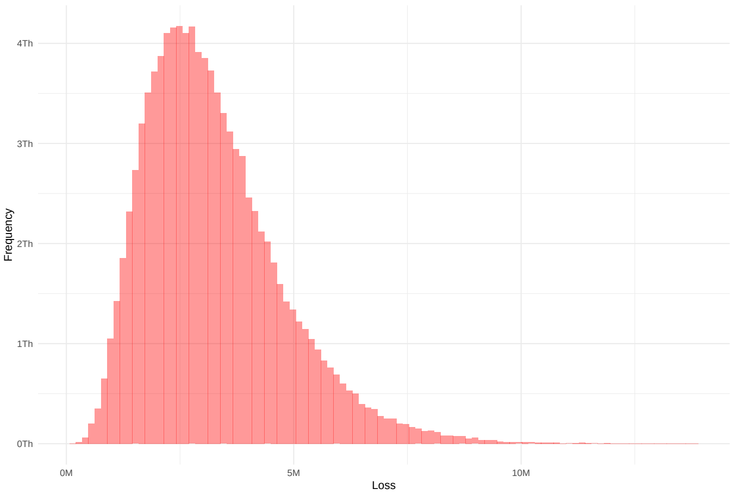
```r
...

for   (i in 1:B) {
      #set random seed
      set.seed(i)
      #systemic factor
      z <- rnorm(n = 1)
      #idiosyncratic factor
      epsilon <- rnorm(n = nrow(db.ng))
      #asset return
      ar <- sqrt(rho)*z + sqrt(1 - rho)*epsilon
      #default indicator (non-granular portfolio)
      def.ind <- ifelse(ar < qnorm(p = db.ng$pd), 1, 0)
      #loss of the non-granular portfolio
      loss.ng <- sum(def.ind * db.ng$ead * lgd)
      #loss of the granular portfolio
      pd.cond <- pnorm(q = (qnorm(p = db.g$pd) - sqrt(rho)*z) / sqrt(1 - rho))
      loss.g <- sum(db.g$ead * pd.cond * lgd)
      #portfolio loss
      res[i] <- loss.ng + loss.g
      }

...
```

Python Code Extract

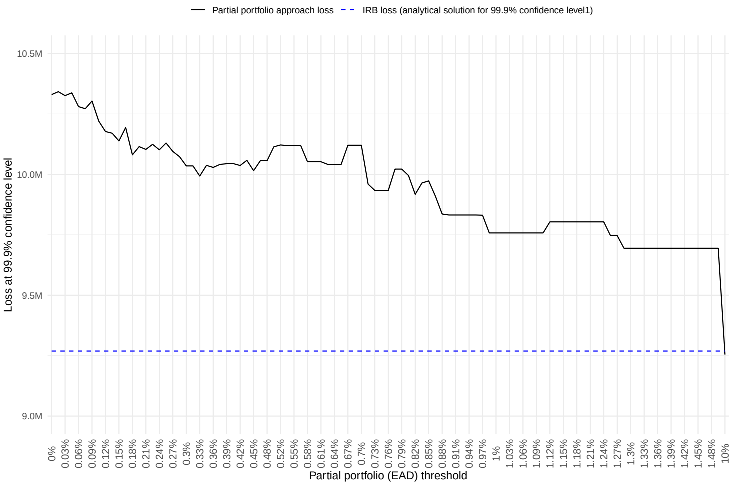
```
import numpy as np
from scipy.stats import norm
...

for i in range(B):
    #set random seed
    np.random.seed(i + 1)
    #systemic factor
    z = np.random.normal(size = 1)
    #idiosyncratic factor
    epsilon = np.random.normal(size = db_ng.shape[0])
    #asset return
    ar = np.sqrt(rho) * z + np.sqrt(1 - rho) * epsilon
    #default indicator (non-granular portfolio)
    def_ind = np.where(ar <  norm.ppf(db_ng["pd"]), 1, 0)
    #loss of the non-granular portfolio
    loss_ng = np.sum(def_ind * db_ng["ead"] * lgd)
    #loss of the granular portfolio
    pd_cond = norm.cdf((norm.ppf(db_g["pd"]) - np.sqrt(rho) * z) /
                       np.sqrt(1 - rho))
    loss_g = np.sum(db_g["ead"] * pd_cond * lgd)
    #portfolio loss
    res[i] = loss_ng + loss_g

...
```

Simulation Result Extract

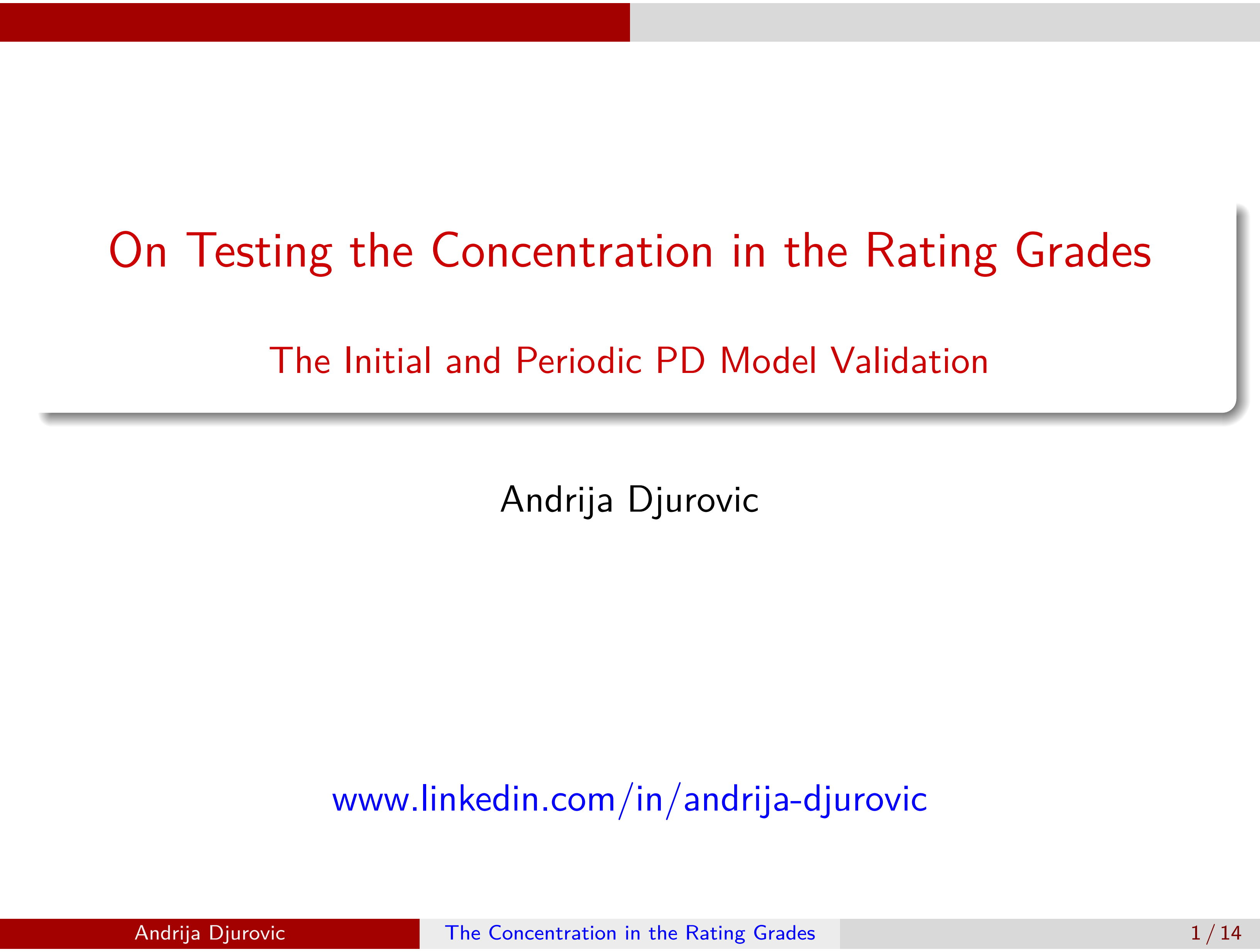
Loss Distribution - Full Non-Granular Portfolio:



Loss at 99.9% CL - Varying EAD Threshold:



5.2 On Testing the Concentration in the Rating Grades - The Initial and Periodic PD Model Validation



On Testing the Concentration in the Rating Grades

The Initial and Periodic PD Model Validation

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Rating Grade Concentration in PD Scale

- Practitioners usually adhere to certain principles when building the Probability of Default (PD) rating scale. These principles result in specific characteristics of the rating scale.
- One such characteristic, which is assessed during the initial model development and is also regularly monitored over time, is the concentration within the rating grades.
- Ideally, the concentration within each rating grade should remain below a certain threshold and should not increase significantly over time compared to when the model was developed.
- Besides the issue of regulatory non-compliance, some other consequences of the excessive concentration within the rating scale include:
  - loss of risk differentiation and mispricing of risk;
  - inaccurate capital requirements;
  - underestimation or overestimation of PD.
- The following slides present the approach currently mandated by the ECB for IRB validation reporting exercises, which most banks have adopted as the standard for assessing concentration in rating grades. They also explore the challenges associated with this approach and examine potential alternatives to address them.

The Standard Practice for Assessing the Concentration

In the Instructions for reporting the validation results of internal models, the ECB requires banks to report the Herfindahl Index (HI) value for the current application portfolio, along with the p-value of statistical tests used to assess changes in the Herfindahl Index from the model's development phase to its current application.

The formula for the Herfindahl Index is provided as follows:

$$HI = 1 + \frac{\log\left(\frac{CV^2+1}{K}\right)}{\log(K)}$$

with CV being a coefficient of variation:

$$CV = \sqrt{K \sum_{i=1}^{K} \left(R_i - \frac{1}{K}\right)^2}$$

where:

- K is the number of rating grades;
- R_i is the relative frequency of the i-th rating grade.

The Standard Practice for Assessing the Concentration cont.

The change in the HI is assessed indirectly by testing the change in the CV using a normal approximation and the assumption of a deterministic CV from the model's development phase.

The test's null hypothesis is that the HI at the time of the model application is lower than or equal to the HI at the time of development. Based on these assumptions, the p-value is calculated as follows:

$$1 - \Phi\left(\frac{\sqrt{K-1}(CV_{curr} - CV_{init})}{\sqrt{CV_{curr}^2\,(0.5 + CV_{curr}^2)}}\right)$$

where:

- Φ is the cumulative distribution function of the standard normal distribution;
- CV_{curr} is the coefficient of variation at the time of the model application;
- CV_{init} is the coefficient of variation at the time of model development (initial validation);
- K is the number of rating grades or bins of the risk factors.

The Challenges Associated with the Standard Practice

While the value of the HI is typically compared directly to specific thresholds - such as 0.20, 0.25, and 0.30 - the change in HI is assessed indirectly through variations in the CV. This raises two key questions: First, does a change in the relative dispersion measure reflect changes in the HI value? Second, how does the assumption of the deterministic CV_{init} influence the test results?

Consider the following data: relative frequencies at the time of the model's development are [0.13, 0.14, 0.23, 0.20, 0.11, 0.10, 0.09], and at the time of the model's application, they are [1, 0, 0, 0, 0, 0, 0]. This simulated data assumes that the HI shifts to a fully concentrated rating scale at the time of the model's application.

Let's examine the HI values during these two periods. The HI value at the time of the model's development is 0.057 while the HI value at the time of the model's application is 1.

Given the assumption of strong concentration, practitioners would expect the p-value for the test of change in the HI to be well below the commonly used significance levels of 0.01 or 0.05. However, the calculated p-value based on the proposed approach is 0.2043.

As observed, the calculated p-value is significantly higher than the commonly used significance levels, leading us to fail to reject the null hypothesis.

The Challenges Associated with the Standard Practice cont.

In addition to the previous example, testing the change in the CV relies on the assumption that the variable is normally distributed. This raises question about the validity of this assumption, particularly when analyzing the relative frequencies of grade in the rating scale at different times.

Although the effects of different underlying distributional assumptions and the relatively small sample size on the assumed test statistics are not explored here, they are important considerations. Investigating these factors is crucial for evaluating the overall validity of the asymptotic normality assumption of the CV, which is given as follows:

$$AN\left(\mu = CV_{sample},\ \sigma = \sqrt{\frac{CV_{sample}^2\left(0.5 + CV_{sample}^2\right)}{K-1}}\right)$$

The Proposed Alternatives

- Instead of indirectly testing changes in the HI based on the CV change, the proposed alternatives employ Monte Carlo simulations to directly assess and test changes in HI values between the two analyzed periods.
- To align with the assumption of deterministic metric value from the time of model development, the first method considers only the variability arising from the data available at the time of model application.
- The second alternative accounts for the variability in both observed relative frequencies while maintaining the portfolio size from the time of model application.
- Details on both alternatives are provided in the following slides.

The Proposed Alternatives - Method 1

Testing process:

1. Calculate the HI metric according to the formula provided on slide 3, using the data from model development. Denote this value as `HI_init`.
2. Based on the application portfolio, for the observed sample size `n` and relative frequencies `f_1` to `f_g` (where `g` is the number of rating grades), simulate `g` observations from a multinomial distribution.
3. Using the simulated observations, calculate the HI value according to the formula provided on slide 3.
4. Repeat steps 2 and 3 `N` times, storing the resulting HI values.
5. Calculate the percentage of simulated HI values greater than `HI_init`, and denote this percentage as `HI_g`.
6. Determine the p-value as `1 - HI_g`.
7. If the p-value is lower than the significance level, conclude that the HI at the time of model application has significantly increased compared to that at the time of model development.

The Proposed Alternatives - Method 2

Testing process:

1. Based on the application portfolio, simulate `g` observations from a multinomial distribution using the observed sample size `n` and relative frequencies `fa_1` to `fa_g` (where `g` is the number of rating grades).
2. Using the same sample size `n` from the application portfolio, simulate `g` observations from a multinomial distribution using the relative frequencies `fd_1` to `fd_g` from the development portfolio.
3. Calculate the HI values using the simulated observations from steps 1 and 2, and denote these results as `HI_curr` and `HI_init`.
4. Repeat steps 1 through 3 `N` times, storing the resulting HI values.
5. Calculate the difference between `HI_curr` and `HI_init` for each simulation.
6. Calculate the percentage of simulated differences greater than 0, and denote this percentage as `D_g`.
7. Determine the p-value as `1 - D_g`.
8. If the p-value is lower than the significance level, conclude that the HI at the time of model application has significantly increased compared to that at the time of model development.

Simulation Study - Method 1

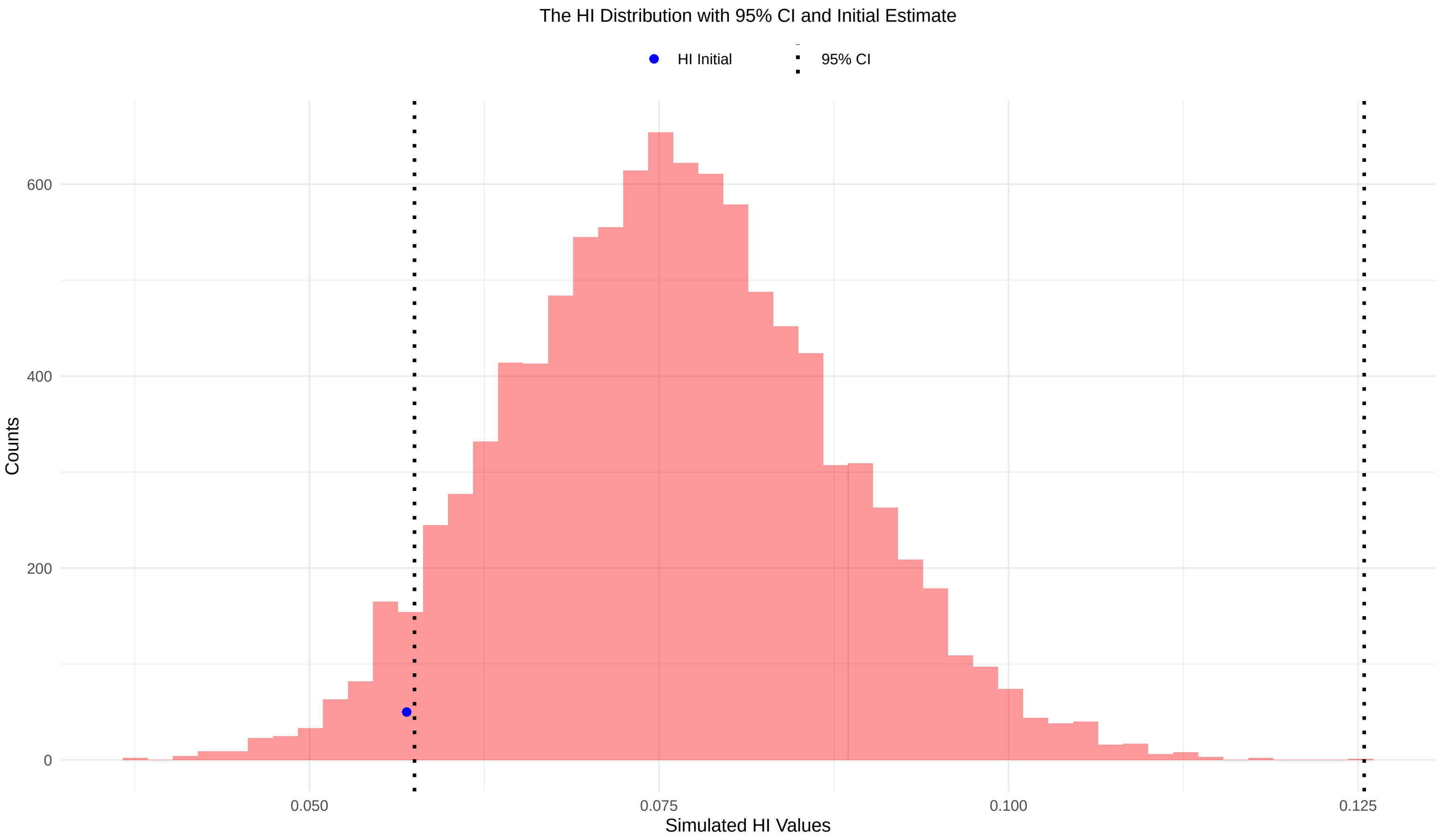
Simulation assumptions:

1. Relative frequencies at the time of model development: 0.13, 0.14, 0.23, 0.2, 0.11, 0.1, 0.09;
2. Relative frequencies at the time of model application: 0.13, 0.14, 0.24, 0.21, 0.11, 0.09, 0.08;
3. Sample size of the application portfolio: `n = 1,000`;
4. Number of simulations: `N = 10,000`.

Simulation results:

1. `HI_init`: 0.057;
2. `HI_curr`: 0.0734;
3. Test `p-value`: 0.047.

Simulation Study - Method 1 cont.



Simulation Study - Method 2

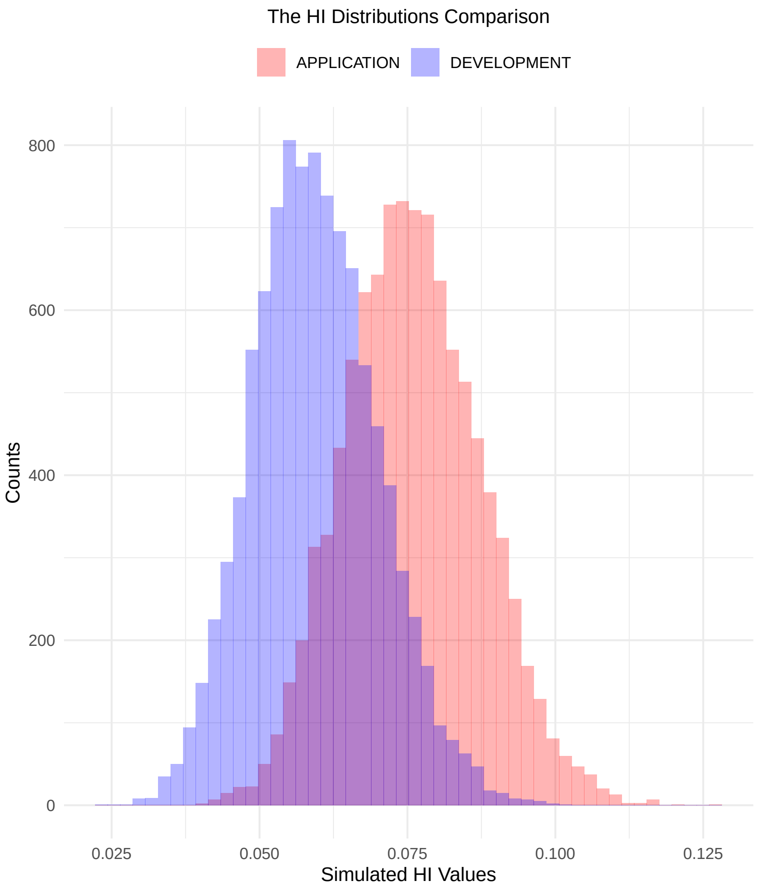
Simulation assumptions:

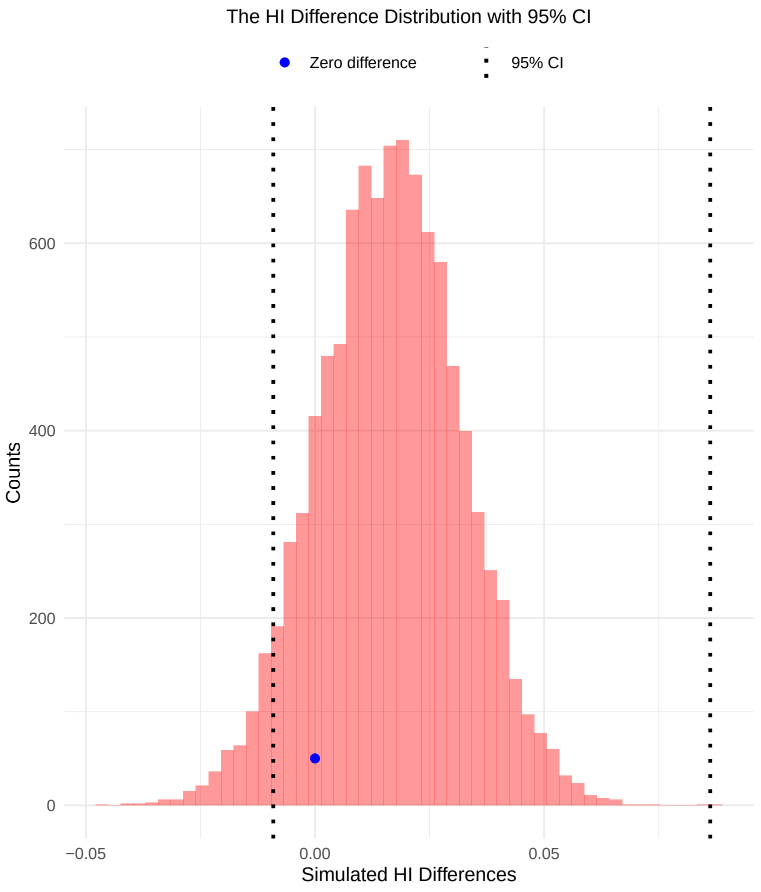
1. Relative frequencies at the time of model development: 0.13, 0.14, 0.23, 0.2, 0.11, 0.1, 0.09;
2. Relative frequencies at the time of model application: 0.13, 0.14, 0.24, 0.21, 0.11, 0.09, 0.08;
3. Sample size of the application portfolio: `n = 1,000`;
4. Number of simulations: `N = 10,000`.

Simulation results:

1. `HI_init`: 0.057;
2. `HI_curr`: 0.0734;
3. Test `p-value`: 0.1464.

Simulation Study - Method 2 cont.





Concluding Remarks

- Besides comparing the HI value to a specific threshold, practitioners often examine whether the HI significantly increased at the time of model application compared to the model development phase.
- The most commonly applied method for reporting on these procedures is the one required by the ECB in the Instructions for reporting the validation results of internal models.
- The standard approach in practice has certain challenges, primarily related to detecting significant changes compared to fully concentrated rating scales and the underlying assumptions of the statistical test.
- Alternative methods can be tested and applied under the same or similar assumptions, directly testing changes in HI across two analyzed periods.
- Both proposed methods can identify statistically significant differences even for small changes in HI when applied to larger portfolios. Therefore, it is recommended to enhance the test results with additional criteria, like practical significance measures that are less affected by sample size, to achieve a more reliable conclusion.
- Given the potential weaknesses of alternative methods, it is advisable to properly balance the direct comparison of the HI value against a specific threshold with the observed change in HI over two periods when making a final decision on rating grade concentration.
- Given the above points, practitioners can also extend the use of alternative methods to LGD and EAD models.

6 Financial Mathematics of Loans

6.1 Loan Repayment Plan

Loan repayment plan using `R` and `Python`

Andrija Djurovic[*]

[*] www.linkedin.com/in/andrija-djurovic

The following example illustrates creating a loan repayment plan using `R` and `Python`.

Loan inputs:

```r
#loan amount
amount <- 5000
#maturity (in months)
maturity <- 18
#yearly interest rate
ir.y <- 0.0649
#monthly interest rate
ir.m <- ir.y / 12
```

User-defined function that generates the loan repayment plan:

```r
create.repayment.plan <- function(p, r, m) {
      #p - loan amount
      #r - monthly interest rate
      #m - loan maturity in months

      #annuity
      annuity <- p * r / (1 - (1 + r)^(-m))
      #prepare the repayment plan data frame
      rp <- data.frame(month = 1:m,
                       remaining.principal = NA,
                       monthly.principal = NA,
                       monthly.interest = NA,
                       annuity = rep(annuity, m))
      #assign the loan amount as a starting principal
      rp$remaining.principal[1] <- p
      #construct the repaymen plan
      for   (i in 1:m) {
            if    (i == m) {
                  rp$monthly.principal[i] <- rp$remaining.principal[i]
                  rp$monthly.interest[i] <- rp$annuity[i] -
                                            rp$monthly.principal[i]
                  } else {
                  rp$monthly.interest[i] <- rp$remaining.principal[i] * r
                  rp$monthly.principal[i] <- rp$annuity[i] -
                                             rp$monthly.interest[i]
                  rp$remaining.principal[i + 1] <- rp$remaining.principal[i] -
```

```
                                                   rp$monthly.principal[i]
                  }
            }
return(rp)
}
```

Loan repayment plan:

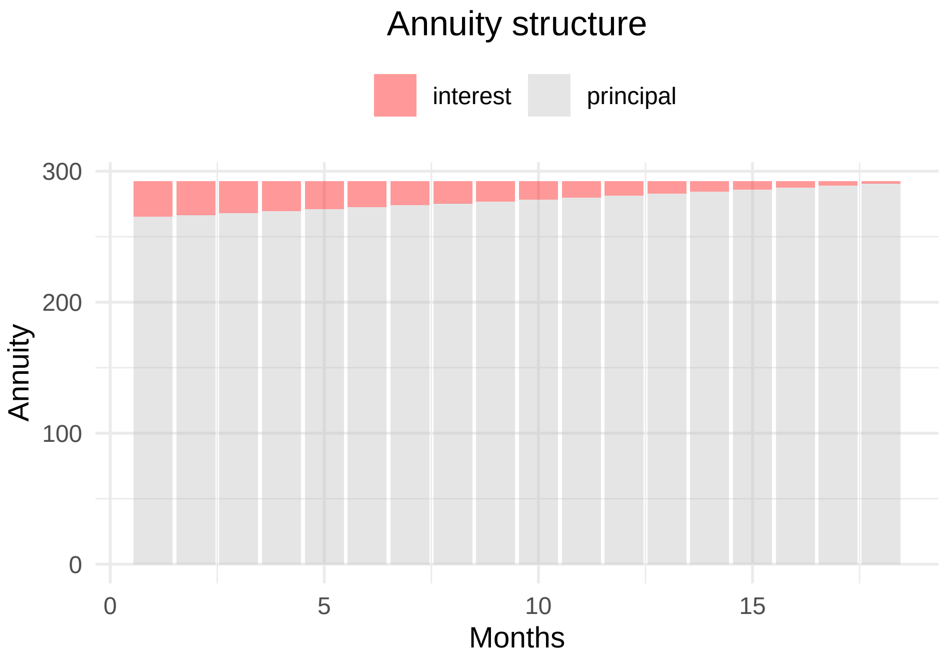
```
rp <- create.repayment.plan(p = amount,
                            r = ir.m,
                            m = maturity)
rp
```

```
##    month remaining.principal monthly.principal monthly.interest  annuity
## 1      1           5000.0000          265.2262        27.041667 292.2678
## 2      2           4734.7738          266.6606        25.607235 292.2678
## 3      3           4468.1132          268.1028        24.165046 292.2678
## 4      4           4200.0104          269.5528        22.715056 292.2678
## 5      5           3930.4576          271.0106        21.257225 292.2678
## 6      6           3659.4470          272.4763        19.791509 292.2678
## 7      7           3386.9707          273.9500        18.317866 292.2678
## 8      8           3113.0207          275.4316        16.836254 292.2678
## 9      9           2837.5891          276.9212        15.346628 292.2678
## 10    10           2560.6679          278.4189        13.848946 292.2678
## 11    11           2282.2490          279.9247        12.343163 292.2678
## 12    12           2002.3243          281.4386        10.829237 292.2678
## 13    13           1720.8857          282.9607         9.307124 292.2678
## 14    14           1437.9250          284.4911         7.776778 292.2678
## 15    15           1153.4339          286.0297         6.238155 292.2678
## 16    16            867.4042          287.5766         4.691211 292.2678
## 17    17            579.8276          289.1319         3.135901 292.2678
## 18    18            290.6957          290.6957         1.572179 292.2678
```

```
#check net present value of the cash flow
sum(rp$annuity / ((1 + ir.m)^(1:maturity)))
```

```
## [1] 5000
```

Annuity structure:



An alternative approach is available for those seeking a more advanced solution that offers precise values for the remaining principal, principal, and interest portion in an annuity at a certain period. The subsequent code demonstrates the calculation of the principal portion in the annuity for the 5$^{\text{th}}$ month. The remaining steps in the repayment plan calculation are straightforward and are left for the reader to complete.

```
#annuity
a <-  amount * ir.m / (1 - (1 + ir.m)^(-maturity))
#define the nth period
n <- 5
#principal portion in the annuity at the 5th period
(a - amount * ir.m) * ((1 + ir.m)^(n - 1))
```

```
## [1] 271.0106
```

Loan repayment plan in `Python`:

```
import pandas as pd
import numpy as np

#input parameters
#loan amount
amount = 5000
#maturity (in months)
```

```
maturity = 18
#yearly interest rate
ir_y = 0.0649
#monthly interest rate
ir_m = ir_y / 12

#loan repayment function
def create_repayment_plan(p, r, m):
    #p - loan amount
    #r - monthly interest rate
    #m - loan maturity in months

    #annuity
    annuity = p * r / (1 - (1 + r)**(-m))
    #prepare the repayment plan data frame
    rp = pd.DataFrame({"month" : [*range(1, m + 1)],
                       "remaining_principal" : [None]*m,
                       "monthly_principal" : [None]*m,
                       "monthly_interest" : [None]*m,
                       "annuity" : [annuity]*m
                      })
    #assign loan amount as a starting principal
    rp.loc[0, "remaining_principal"] = p
    #construct the repaymen plan
    for i in rp.index:
        if (i == rp.index[-1]):
            rp.loc[i, "monthly_principal"] = rp.remaining_principal[i]
            rp.loc[i, "monthly_interest"] = rp.annuity[i] - \
                                            rp.monthly_principal[i]
        else:
            rp.loc[i, "monthly_interest"] = rp.remaining_principal[i] * r
            rp.loc[i, "monthly_principal"] = rp.annuity[i] - \
                                             rp.monthly_interest[i]
            rp.loc[i + 1, "remaining_principal"] = rp.remaining_principal[i] - \
                                                   rp.monthly_principal[i]
    return(rp)


#create repayment plan
```

```
rp = create_repayment_plan(p = amount,
                           r = ir_m,
                           m = maturity)
rp
```

```
##     month remaining_principal monthly_principal monthly_interest     annuity
## 0       1                5000        265.226177        27.041667  292.267843
## 1       2         4734.773823        266.660608        25.607235  292.267843
## 2       3         4468.113215        268.102798        24.165046  292.267843
## 3       4         4200.010417        269.552787        22.715056  292.267843
## 4       5          3930.45763        271.010618        21.257225  292.267843
## 5       6         3659.447012        272.476334        19.791509  292.267843
## 6       7         3386.970678        273.949977        18.317866  292.267843
## 7       8         3113.020701         275.43159        16.836254  292.267843
## 8       9         2837.589112        276.921216        15.346628  292.267843
## 9      10         2560.667896        278.418898        13.848946  292.267843
## 10     11         2282.248998         279.92468        12.343163  292.267843
## 11     12         2002.324318        281.438606        10.829237  292.267843
## 12     13         1720.885712         282.96072         9.307124  292.267843
## 13     14         1437.924992        284.491066         7.776778  292.267843
## 14     15         1153.433927        286.029688         6.238155  292.267843
## 15     16          867.404239        287.576632         4.691211  292.267843
## 16     17          579.827607        289.131942         3.135901  292.267843
## 17     18          290.695664        290.695664         1.572179  292.267843
```

```
#check net present value of the cash flow
np.sum(rp["annuity"] / (1 + ir_m) ** np.arange(1, maturity + 1))
```

```
## 5000.000000000088
```

```
#annuity
a = amount * ir_m / (1 - (1 + ir_m)**(-maturity))
#define the nth period
n = 5
#principal portion in the annuity at the nth period
(a - amount * ir_m) * ((1 + ir_m)**(n - 1))
```

```
## 271.0106182999124
```

6.2 Effective Interest Rate

Effective interest rate using R and Python

Andrija Djurovic[*]

[*] www.linkedin.com/in/andrija-djurovic

The following example illustrates the calculation of the effective interest rate (EIR) using `R` and `Python`. We will begin by specifying the loan characteristics.

```
#loan amount
amount <- 10e3
#yearly nominal interest rate
ir.y <- 0.0599
#monthly nominal interest rate
ir.m <- ir.y / 12
#maturity (in months)
maturity <- 60
```

Let's further assume that the borrower has an initial cost, which is immediately deducted from the loan amount and proceed with EIR calculation.

```
#initial loan costs
cost <- 150
#annuity
annuity <- amount * ir.m /(1 - (1 + ir.m) ^ (-maturity))
annuity
```

```
## [1] 193.2815
```

```
#check npv of the cash flow
sum(rep(annuity, maturity) / cumprod(1 + rep(ir.m, maturity)))
```

```
## [1] 10000
```

```
#optimization function
eir.opt <- function(amount, annuity, maturity, ir, cost) {
      #amount - loan amount
      #annuity - annuity
      #ir - effective interest rate (monthly)
      #cost - initial cost

      #cash flow
      cf <- rep(annuity, maturity)
      #discounted cash flow
      df <- cumprod(1 + rep(ir, maturity))
      #optimization value
      opt.array <- sum(cf / df) - (amount - cost)
return(opt.array)
}
```

```
#calculate the effective interest rate
eir <- uniroot(f = eir.opt,
               amount = amount,
               annuity = annuity,
               maturity = maturity,
               cost = cost,
               interval = c(0, 1))$root
#monthly effective interest rate
eir
```

```
## [1] 0.005515214
```

```
#yearly effective interest rate
eir * 12
```

```
## [1] 0.06618257
```

```
#check the cash flow under the effective interest rate
sum(rep(annuity, maturity) / cumprod(1 + rep(eir, maturity)))
```

```
## [1] 9850.453
```

Let’s now replicate the same process in `Python`.

```
import numpy as np
from scipy.optimize import root_scalar

#loan amount
amount = 10e3
#yearly nominal interest rate
ir_y = 0.0599
#monthly nominal interest rate
ir_m = ir_y / 12
#maturity (in months)
maturity = 60
#initial loan costs
cost = 150
#monthly annuity
annuity = amount * ir_m / (1 - (1 + ir_m) ** (-maturity))
annuity
```

```
## 193.28151998435652
```

```
#check npv of the cash flow
np.sum(np.repeat(annuity, maturity) /
np.cumprod(1 + np.repeat(ir_m, maturity)))
```

```
## 9999.999999999876
```

```
#optimization function
def eir_opt(ir, amount, annuity, maturity, cost):
    #amount - loan amount
    #annuity - annuity
    #ir - effective interest rate (monthly)
    #cost - initial cost

    #cash flow
    cf = np.repeat(annuity, maturity)
    #discounted cash flow
    df = np.cumprod(1 + np.repeat(ir, maturity))
    #optimization value
    opt_array = np.sum(cf / df) - (amount - cost)

    return opt_array

#calculate the effective interest rate
eir = root_scalar(f = eir_opt,
                  args = (amount, annuity, maturity, cost),
                  bracket = [0, 1],
                  x0 = ir_m).root
#monthly effective interest rate
eir
```

```
## 0.005516816331682868
```

```
#yearly effective interest rate
eir * 12
```

```
## 0.0662017959801944
```

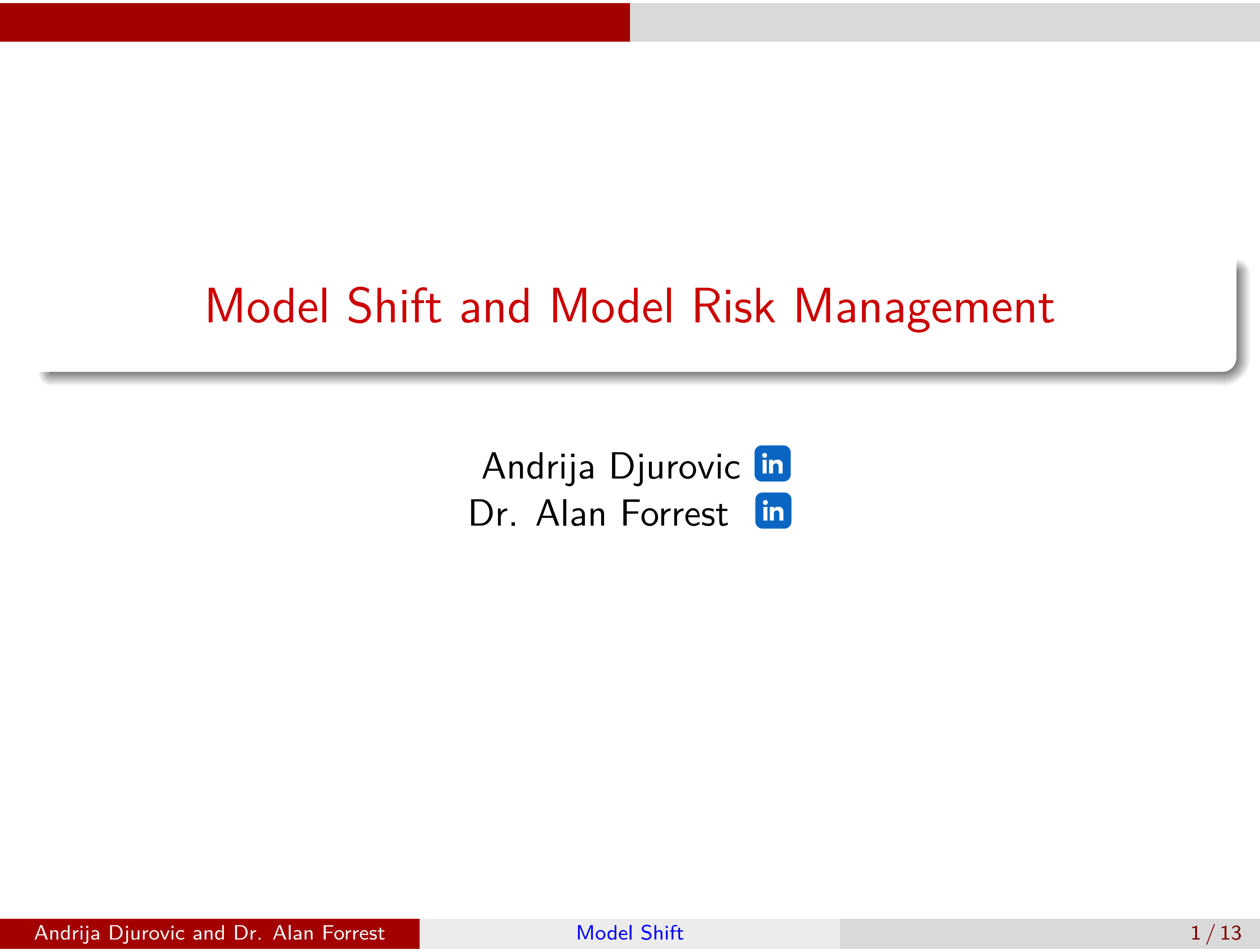
```
#check the cash flow under the effective interest rate
np.sum(np.repeat(annuity, maturity) /
np.cumprod(1 + np.repeat(eir, maturity)))
```

```
## 9849.999999925732
```

Both `R` and `Python` offer numerous methods for computing the effective interest rate. Generally, this computation is analogous to calculating the internal rate of return, a functionality provided by various packages in each programming language.

7 Model Risk Management

7.1 Model Shift and Model Risk Management



Model Shift and Model Risk Management

Andrija Djurovic
Dr. Alan Forrest

Andrija Djurovic and Dr. Alan Forrest | Model Shift | 1 / 13

Model Shift and Model Risk Management

Model shift refers to the quantitative change in a model's parameters and outputs resulting from shifts in the input data. It is particularly useful in credit risk modeling for understanding how models react to changes in their underlying assumptions, data, or environment. The concept of model shift enables practitioners to:

- efficiently address "What if...?" scenarios, quantifying how data shifts impact model outputs without needing complete model redevelopment;
- provide a systematic way to assess and respond to model sensitivities and weaknesses, enhancing model validation, monitoring, and risk management.

The following slides describe the framework for quantifying the model shift in one of the most commonly used methods in credit risk: logistic regression with categorical risk factors.

In this framework both data and models can be presented as observed and expected counts in a high-dimensional contingency table. These in turn are converted to points in a Data Space of high dimensional vectors. Here a data point is defined by the proportion of the observed population in each cell, viewed as a vector of real numbers indexed by the cells. Likewise, the model point is defined by the proportion of the expected population in each cell etc. We can keep track separately of the total population size for purposes of inference, but note that the data point and model point do not vary as population size changes. The maximum likelihood construction of logistic regression model from data depends solely on the proportions.

Applications of Model Shift

Model shift in response to data shift is more than an academic exercise. It is clearly closely related to data drift and concept drift, well known concerns in model management, and it underlies modeling techniques such as imputation, bootstrapping and rebalancing. We are interested in it as a method for important analyses in modern model risk management.

The following list outlines some use cases of the model shift:

- *Dynamic Model Reweighting*: Enables real-time updates of model parameters as new data streams in, ensuring agility in model adjustments without waiting for periodic reviews.
- *Prioritizing Validation Investigations*: Quickly computes model shifts for various data shift scenarios, enabling efficient triage and focus on the most impactful concerns.
- *Quantifying Business Impacts*: Links data shifts to business-relevant metrics like Probability of Default (PD) in Risk-Weighted Assets (RWA), ensuring sensitivity analyses are connected to actionable outcomes.
- *Sensitive Data Shift Identification*: Enables the identification of data shifts that have the most significant impact on models, enriching the validation narrative with actionable insights.
- *Bespoke Model Monitoring*: Defines monitoring metrics for sensitive data shifts, creating early warning systems, particularly for population shifts that do not immediately affect model outputs.
- *Automated Validation and Monitoring*: Streamlines validation and monitoring processes, integrating them with dynamic model updates for continuous, real-time risk management.

Practitioners can refer to this document for further details.

Methods for Quantifying Model Shift

- The methods for quantifying model shift are:

  1. matrix multiplication approach;
  2. weighted binomial logistic regression;
  3. weighted quasi-binomial regression (weighted fractional logistic regression).

- Dr. Alan Forrest proposes a first-order approximation using a matrix multiplication approach to quantify changes in model parameters directly.
- Andrija Djurovic introduces two exact alternative methods (weighted binomial and fractional logistic regression) based on re-estimating model parameters, both of which can address the same task.
- The direct but approximate approach is useful where many thousands or millions of shifts are to be compared or summarised. The exact approaches are fully accurate and ideal for testing smaller numbers of sensitivities or scenarios.
- All three approaches are related to the widely used binomial logistic regression method with categorized risk factors commonly employed in developing PD models.
- Similarly, practitioners can extend the proposed framework to Ordinary Least Squares (OLS) regression, a commonly used method for modeling Loss Given Default (LGD) and Exposure at Default (EaD).

Matrix Multiplication Approach

The first-order model shift (Δp) can be explicitly represented as a matrix multiplication of the data shift. The following formulas illustrate the process of approximating parameter changes given the data shifts (Δx^+ and Δx^-):

$$\Delta p = C^{-1} D^T \left[(I + Z)^{-1} \Delta x^+ - (I + Z^{-1})^{-1} \Delta x^- \right]$$

where:

- D is the design matrix;
- Y^+ and Y^- are the diagonal matrices of modeled frequencies restricted to binary output 1 and 0, respectively;
- $Z = Y^+(Y^-)^{-1}$ is diagonal matrix of modeled odds ratios;
- I is identity matrix dimensions `nrow(Z) x nrow(Z)`;
- $Y = (I + Z)^{-1}(I + Z^{-1})^{-1}(Y^+ + Y^-)$;
- $C = D^T Y D$;
- Δx^+ and Δx^- are the shifts in the proportions of input factors for binary outputs 1 and 0, respectively.

Practitioners can refer to this document for further details.

Weighted Binomial Logistic Regression

Another way to quantify changes in the parameters of the logistic regression based on the data shift is to re-estimate the weighted binomial logistic regression.

The following formula presents the log-likelihood function used to estimate the parameters (β) of the weighted logistic regression::

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} w_i \left[y_i \log\left(\frac{1}{1+\exp(-\mathbf{x}_i\beta)}\right) + (1-y_i)\log\left(1 - \frac{1}{1+\exp(-\mathbf{x}_i\beta)}\right)\right]$$

where:

- y_i is the binary response variable for the `i`-th observation (either 0 or 1);
- $\mathbf{x}_i$ is the vector of predictors for the `i`-th observation;
- w_i is the associated weight of the `i`-th observation.

Weighted Quasi-Binomial Regression

The third method for quantifying changes in logistic regression parameters, based on the data shift, is by re-estimating the weighted quasi-binomial regression. Unlike binomial logistic regression, which requires a dichotomous target (0/1), weighted quasi-binomial regression processes fractions between 0 and 1. The weighted fractional logistic regression parameters can be estimated similarly to binomial logistic regression by maximizing the log-likelihood function with an additional term to account for dispersion. Since the additional term affects only the standard error of estimates, the estimated coefficients between weighted binomial and weighted quasi-binomial regression are identical.

The following formula presents the log-likelihood function used to estimate the model parameters ($\boldsymbol{\beta}$), along with the adjustment of the variance-covariance matrix ($\hat{\Sigma}$) based on the dispersion parameter:

$$\mathcal{L}(\beta) = \sum_{i=1}^{n} w_i \left[y_i \log\left(\frac{1}{1+\exp(-\mathbf{x}_i\boldsymbol{\beta})}\right) + (1-y_i)\log\left(1 - \frac{1}{1+\exp(-\mathbf{x}_i\boldsymbol{\beta})}\right)\right]$$

$$\hat{\Sigma} = \hat{\Phi}\hat{V}$$

where:

- y_i is the binary response variable for the `i`-th observation (either 0 or 1);
- $\mathbf{x}_i$ is the vector of predictors for the `i`-th observation;
- w_i is the associated weight of the `i`-th observation;
- $\hat{\Phi}$ is the estimate of the dispersion parameter;
- $\hat{V}$ is the estimated variance-covariance matrix assuming a binomial distribution (the “naive” variance-covariance matrix).

Simulation Study

The following steps outline the simulation framework used to quantify changes in model parameters based on a simulated scenario:

1. Assume a simplified PD model consisting of the target variable `Creditability` and two categorical risk factors: `Account_Balance` and `Maturity`. The simulation dataset is available here.

2. The risk factor `Account_Balance` includes four categories with the following distribution of observations:

```
##  01  02  03  04
## 274 269  63 394
```

3. The risk factor `Maturity` includes five categories with the following distribution of observations:

```
## 01 (-Inf,8)   02 [8,16)  03 [16,36)  04 [36,45) 05 [45,Inf)
##          87         344         399         100          70
```

Simulation Study cont.

4. The final PD model is estimated using binomial logistic regression and dummy encoding in the form: `Creditability ~ Account_Balance + Maturity` with the following estimated coefficients:

   ```
   ##                      Estimate Std. Error z value Pr(>|z|)
   ## (Intercept)           -1.3234     0.3720 -3.5579   0.0004
   ## Account_Balance02     -0.5064     0.1809 -2.7997   0.0051
   ## Account_Balance03     -1.0873     0.3332 -3.2629   0.0011
   ## Account_Balance04     -2.0194     0.2029 -9.9507   0.0000
   ## Maturity02 [8,16)      0.9783     0.3873  2.5262   0.0115
   ## Maturity03 [16,36)     1.4282     0.3809  3.7495   0.0002
   ## Maturity04 [36,45)     1.8817     0.4248  4.4297   0.0000
   ## Maturity05 [45,Inf)    2.4041     0.4491  5.3532   0.0000
   ```

5. Assume the following scenario: the portfolio structure changes as the bank plans to increase loan approvals for riskier groups, specifically clients in the `Account_Balance` category `01`, by 40%. Simultaneously, loan approvals for clients in category `04` will decrease by the same number. Given this scenario, the new allocation of `Account_Balance` modalities is:

   ```
   ##    01    02    03    04
   ## 383.6 269.0  63.0 284.4
   ```

   An additional assumption is that the observed default rates remain unchanged.

6. Based on this scenario and the resulting portfolio structure changes, the objective is to quantify the change in the estimated parameters of the final PD model using the three methods presented in the previous slides.

Simulation Results - Matrix Multiplication Approach

Data Points x:

```
##    Account_Balance    Maturity     1     0
## 1               01 01 (-Inf,8) 0.004 0.018
## 2               01   02 [8,16) 0.033 0.053
## 3               01  03 [16,36) 0.063 0.055
## 4               01  04 [36,45) 0.019 0.010
## 5               01 05 [45,Inf) 0.016 0.003
## 6               02 01 (-Inf,8) 0.004 0.013
## 7               02   02 [8,16) 0.029 0.061
## 8               02  03 [16,36) 0.038 0.064
## 9               02  04 [36,45) 0.014 0.014
## 10              02 05 [45,Inf) 0.020 0.012
## 11              03 01 (-Inf,8) 0.000 0.008
## 12              03   02 [8,16) 0.007 0.021
## 13              03  03 [16,36) 0.006 0.015
## 14              03  04 [36,45) 0.001 0.005
## 15              04 01 (-Inf,8) 0.001 0.039
## 16              04   02 [8,16) 0.011 0.129
## 17              04  03 [16,36) 0.022 0.136
## 18              04  04 [36,45) 0.008 0.029
## 19              04 05 [45,Inf) 0.004 0.015
```

Model Points y:

```
##    Account_Balance    Maturity      1      0
## 1               01 01 (-Inf,8) 0.0046 0.0174
## 2               01   02 [8,16) 0.0357 0.0503
## 3               01  03 [16,36) 0.0621 0.0559
## 4               01  04 [36,45) 0.0184 0.0106
## 5               01 05 [45,Inf) 0.0142 0.0048
## 6               02 01 (-Inf,8) 0.0024 0.0146
## 7               02   02 [8,16) 0.0269 0.0631
## 8               02  03 [16,36) 0.0409 0.0611
## 9               02  04 [36,45) 0.0144 0.0136
## 10              02 05 [45,Inf) 0.0205 0.0115
## 11              03 01 (-Inf,8) 0.0007 0.0073
## 12              03   02 [8,16) 0.0054 0.0226
## 13              03  03 [16,36) 0.0057 0.0153
## 14              03  04 [36,45) 0.0022 0.0038
## 15              04 01 (-Inf,8) 0.0014 0.0386
## 16              04   02 [8,16) 0.0120 0.1280
## 17              04  03 [16,36) 0.0203 0.1377
## 18              04  04 [36,45) 0.0070 0.0300
## 19              04 05 [45,Inf) 0.0053 0.0137
```

Data Shifts Δx^{+} and Δx^{-}:

```
##    Account_Balance    Maturity   n dx_plus dx_minus
## 1               01 01 (-Inf,8)  22 -0.0016  -0.0072
## 2               01   02 [8,16)  86 -0.0132  -0.0212
## 3               01  03 [16,36) 118 -0.0252  -0.0220
## 4               01  04 [36,45)  29 -0.0076  -0.0040
## 5               01 05 [45,Inf)  19 -0.0064  -0.0012
## 6               02 01 (-Inf,8)  17  0.0000   0.0000
## 7               02   02 [8,16)  90  0.0000   0.0000
## 8               02  03 [16,36) 102  0.0000   0.0000
## 9               02  04 [36,45)  28  0.0000   0.0000
## 10              02 05 [45,Inf)  32  0.0000   0.0000
## 11              03 01 (-Inf,8)   8  0.0000   0.0000
## 12              03   02 [8,16)  28  0.0000   0.0000
## 13              03  03 [16,36)  21  0.0000   0.0000
## 14              03  04 [36,45)   6  0.0000   0.0000
## 15              04 01 (-Inf,8)  40  0.0003   0.0108
## 16              04   02 [8,16) 140  0.0031   0.0359
## 17              04  03 [16,36) 158  0.0061   0.0378
## 18              04  04 [36,45)  37  0.0022   0.0081
## 19              04 05 [45,Inf)  19  0.0011   0.0042
```

C Matrix (MxM):

```
##                     (Intercept) Account_Balance02 Account_Balance03 Account_Balance04 Maturity02 [8,16) Maturity03 [16,36) Maturity04 [36,45) Maturity05 [45,Inf)
## (Intercept)              0.1741            0.0598            0.0105            0.0395            0.0551             0.0758             0.0208              0.0148
## Account_Balance02        0.0598            0.0598            0.0000            0.0000            0.0189             0.0245             0.0070              0.0074
## Account_Balance03        0.0105            0.0000            0.0105            0.0000            0.0044             0.0042             0.0014              0.0000
## Account_Balance04        0.0395            0.0000            0.0000            0.0395            0.0110             0.0177             0.0057              0.0038
## Maturity02 [8,16)        0.0551            0.0189            0.0044            0.0110            0.0551             0.0000             0.0000              0.0000
## Maturity03 [16,36)       0.0758            0.0245            0.0042            0.0177            0.0000             0.0758             0.0000              0.0000
## Maturity04 [36,45)       0.0208            0.0070            0.0014            0.0057            0.0000             0.0000             0.0208              0.0000
## Maturity05 [45,Inf)      0.0148            0.0074            0.0000            0.0038            0.0000             0.0000             0.0000              0.0148
```

The Estimated Coefficient Changes:

```
##         (Intercept)  Account_Balance02  Account_Balance03  Account_Balance04  Maturity02 [8,16)  Maturity03 [16,36)  Maturity04 [36,45) Maturity05 [45,Inf)
##              0.0150             0.0056            -0.0055             0.0041            -0.0027             -0.0160             -0.0158             -0.0920
```

Simulation Results - Weighted Binomial Logistic Regression

Sample of the Aggregated Dataset with Initial Counts (`n`):

```
##  Account_Balance    Maturity Creditability  n
##               01 01 (-Inf,8)             0 18
##               01 01 (-Inf,8)             1  4
##               01   02 [8,16)             0 53
##               01   02 [8,16)             1 33
##               01  03 [16,36)             0 55
##               01  03 [16,36)             1 63
##               01  04 [36,45)             0 10
##               01  04 [36,45)             1 19
##               01 05 [45,Inf)             0  3
##               01 05 [45,Inf)             1 16
```

Sample of the Aggregated Dataset with Simulated Counts (`n_s`):

```
##  Account_Balance    Maturity Creditability  n_s
##               01 01 (-Inf,8)             0 25.2
##               01 01 (-Inf,8)             1  5.6
##               01   02 [8,16)             0 74.2
##               01   02 [8,16)             1 46.2
##               01  03 [16,36)             0 77.0
##               01  03 [16,36)             1 88.2
##               01  04 [36,45)             0 14.0
##               01  04 [36,45)             1 26.6
##               01 05 [45,Inf)             0  4.2
##               01 05 [45,Inf)             1 22.4
```

The Estimated Coefficient Changes:

```
##        (Intercept)   Account_Balance02   Account_Balance03   Account_Balance04   Maturity02 [8,16)  Maturity03 [16,36)  Maturity04 [36,45) Maturity05 [45,Inf)
##             0.0158              0.0056             -0.0052              0.0042             -0.0048             -0.0165             -0.0150             -0.0923
```

Simulation Results - Weighted Quasi-Binomial Regression

Sample of the Aggregated Dataset with Initial Counts (n):

```
##  Account_Balance    Maturity   n      frac
##               01 01 (-Inf,8)  22 0.1818182
##               01   02 [8,16)  86 0.3837209
##               01  03 [16,36) 118 0.5338983
##               01  04 [36,45)  29 0.6551724
##               01 05 [45,Inf)  19 0.8421053
##               02 01 (-Inf,8)  17 0.2352941
##               02   02 [8,16)  90 0.3222222
##               02  03 [16,36) 102 0.3725490
##               02  04 [36,45)  28 0.5000000
##               02 05 [45,Inf)  32 0.6250000
```

Sample of the Aggregated Dataset with Simulated Counts (n_s):

```
##  Account_Balance    Maturity   n      frac   n_s
##               01 01 (-Inf,8)  22 0.1818182  30.8
##               01   02 [8,16)  86 0.3837209 120.4
##               01  03 [16,36) 118 0.5338983 165.2
##               01  04 [36,45)  29 0.6551724  40.6
##               01 05 [45,Inf)  19 0.8421053  26.6
##               02 01 (-Inf,8)  17 0.2352941  17.0
##               02   02 [8,16)  90 0.3222222  90.0
##               02  03 [16,36) 102 0.3725490 102.0
##               02  04 [36,45)  28 0.5000000  28.0
##               02 05 [45,Inf)  32 0.6250000  32.0
```

The Estimated Coefficient Changes:

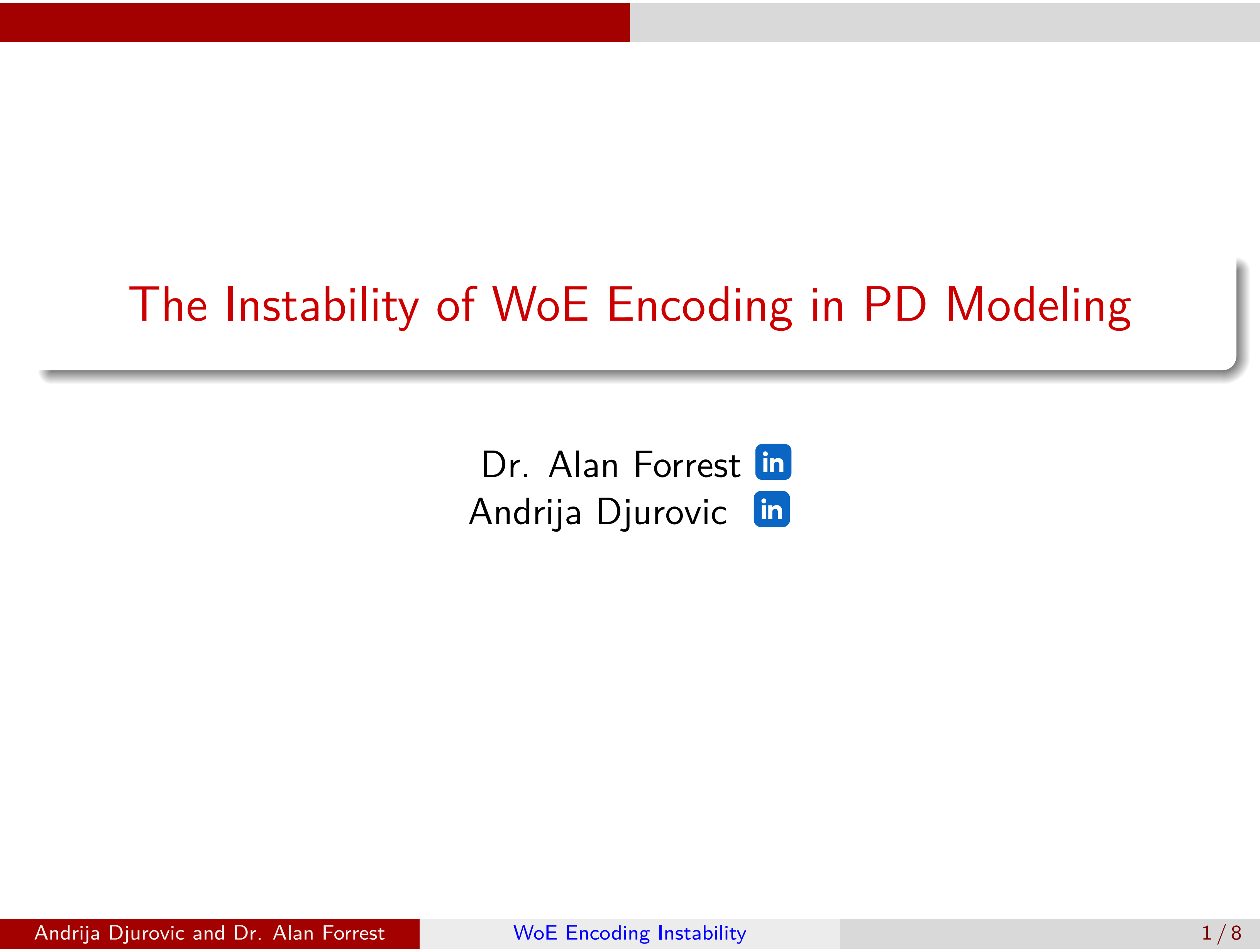
```
##         (Intercept)    Account_Balance02    Account_Balance03    Account_Balance04    Maturity02 [8,16)  Maturity03 [16,36)  Maturity04 [36,45) Maturity05 [45,Inf)
##              0.0158               0.0056              -0.0052               0.0042              -0.0048             -0.0165             -0.0150             -0.0923
```

Simulation Results - Summary

The table below provides a summary and comparison of the model shift simulation results:

```
##          coefficient matrix multiplication weighted logistic weighted quasi-binomial
##          (Intercept)                0.0150            0.0158                  0.0158
##    Account_Balance02                0.0056            0.0056                  0.0056
##    Account_Balance03               -0.0055           -0.0052                 -0.0052
##    Account_Balance04                0.0041            0.0042                  0.0042
##    Maturity02 [8,16)               -0.0027           -0.0048                 -0.0048
##   Maturity03 [16,36)               -0.0160           -0.0165                 -0.0165
##   Maturity04 [36,45)               -0.0158           -0.0150                 -0.0150
##  Maturity05 [45,Inf)               -0.0920           -0.0923                 -0.0923
```

7.2 The Instability of WoE Encoding in PD Modeling



The Instability of WoE Encoding in PD Modeling

Dr. Alan Forrest
Andrija Djurovic

Andrija Djurovic and Dr. Alan Forrest | WoE Encoding Instability | 1 / 8

Standard Approaches to PD Modeling

- Probability of Default (PD) rating modeling is widely considered a cornerstone of credit risk.
- The most commonly used method for modeling PD is binomial logistic regression with categorized risk factors.
- Within this framework, two standard approaches for encoding categorical risk factors are dummy encoding and Weights of Evidence (WoE) encoding. These are the simplest kinds of pre-processing and factor engineering available to modelers.
- Practitioners often debate using one encoding method over another, while some advocate combining both within the final model. This is usually seen as a trade-off between degrees of freedom, explainability and accuracy.
- This presentation investigates factor encoding in a different way: as a possible source of model instability. It does this by examining the replicability of a model build using perfect outcome data. In other words we follow the following steps:
    1. we fix a factor encoding method;
    2. we take a training dataset on which we build a PD rating model using that adopted factor encoding method;
    3. that PD model predicts an expected outcome on the training dataset, and we use that expected outcome data as a second training dataset for a second model, built again using the adopted factor encoding method;
    4. we ask: does the second model equal the first one?
- This presentation observes that dummy variable encoding gives perfect replication (in theory and in practice), and WoE encoding does not (by example). It further explores what this means for the model risk management and potential instability of the WoE encoding method.

Dummy Encoding - Does the Model Replicate?

The following steps outline replicating the model estimates on the observed and predicted target. As can be seen, the dummy encoding replicates the exact estimates.

The simulation dataset can be found here.

1. Estimate the model of the form `Creditability ~ Maturity + Amount + Age`:

```
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -1.55       0.38   -4.08     0.00
## Maturity02 [8,12)          0.67       0.44    1.51     0.13
## Maturity03 [12,16)         0.97       0.38    2.53     0.01
## Maturity04 [16,36)         1.30       0.37    3.48     0.00
## Maturity05 [36,45)         1.57       0.43    3.68     0.00
## Maturity06 [45,Inf)        2.06       0.46    4.50     0.00
## Amount02 [3914,6758)       0.19       0.21    0.90     0.37
## Amount03 [6758,Inf)        0.58       0.25    2.29     0.02
## Age02 [26,35)             -0.54       0.19   -2.79     0.01
## Age03 [35,Inf)            -0.90       0.19   -4.68     0.00
```

2. Using the model from step 1, generate within-sample predictions `pred`.

3. Estimate the model of the form `pred ~ Maturity + Amount + Age` and compare the estimates with those from step 1:

```
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -1.55       0.38   -4.08     0.00
## Maturity02 [8,12)          0.67       0.44    1.51     0.13
## Maturity03 [12,16)         0.97       0.38    2.53     0.01
## Maturity04 [16,36)         1.30       0.37    3.48     0.00
## Maturity05 [36,45)         1.57       0.43    3.68     0.00
## Maturity06 [45,Inf)        2.06       0.46    4.50     0.00
## Amount02 [3914,6758)       0.19       0.21    0.90     0.37
## Amount03 [6758,Inf)        0.58       0.25    2.29     0.02
## Age02 [26,35)             -0.54       0.19   -2.79     0.01
## Age03 [35,Inf)            -0.90       0.19   -4.68     0.00
```

WoE Encoding - Does the Model Replicate?

Using the same dataset as in the previous slide, the following steps outline replicating the model estimates on the observed and predicted target using WoE encoding. Unlike dummy encoding, this process involves recalculating the risk factor WoE on the predicted sample. As can be seen, WoE encoding does not replicate the estimates of the perfect model.

1. Replace each modality of the risk factors with the observed WoE values.
2. Estimate the model of the form `Creditability ~ Maturity + Amount + Age`:

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.85       0.07  -11.69     0.00
## Maturity       -0.83       0.16   -5.12     0.00
## Amount         -0.49       0.22   -2.20     0.03
## Age            -1.05       0.23   -4.67     0.00
```

3. Using the model from step 2, generate within-sample predictions `pred`.
4. Recalculate the WoE values for the risk factor `Maturity` and compare them with the values from step 1:

WoE Values on the Observed Target:

```
##          rf          bin   woe
## 8  Maturity 01 (-Inf,8)  1.31
## 2  Maturity   02 [8,12)  0.58
## 3  Maturity  03 [12,16)  0.27
## 1  Maturity  04 [16,36) -0.11
## 19 Maturity  05 [36,45) -0.52
## 14 Maturity 06 [45,Inf) -1.13
```

WoE Values on the Predicted Target:

```
##         rf          bin   woe
## 1 Maturity 01 (-Inf,8)  1.23
## 2 Maturity   02 [8,12)  0.55
## 3 Maturity  03 [12,16)  0.29
## 4 Maturity  04 [16,36) -0.09
## 5 Maturity  05 [36,45) -0.56
## 6 Maturity 06 [45,Inf) -1.17
```

Since the WoE values calculated on the observed and predicted targets differ, this implies that the perfect model does not replicate.

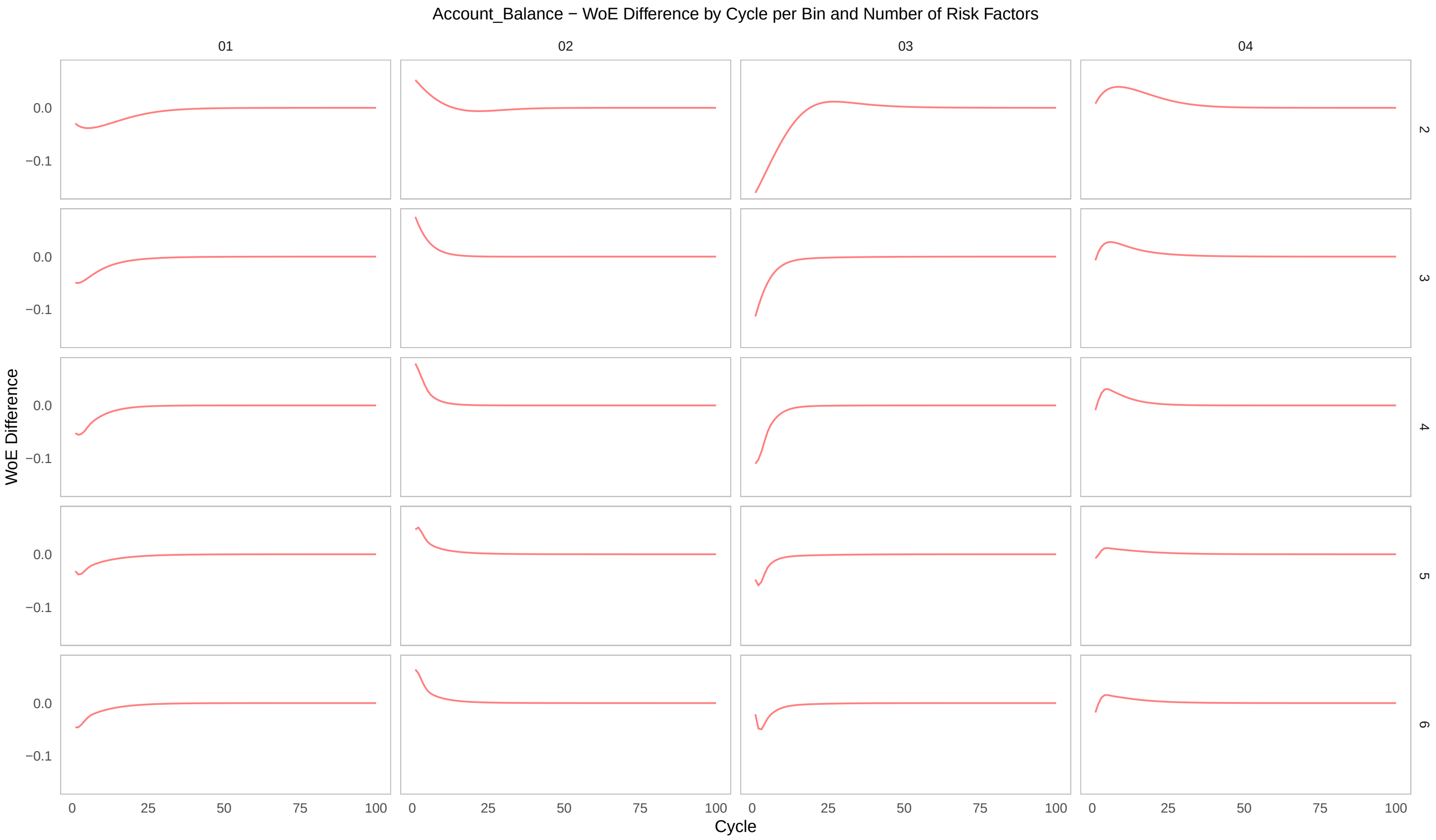
Simulation Study

The following slides present simulations visualizing the WoE difference per model recycle iteration and the number of risk factors in the model. In other words, it calculates the difference in WoE values for the same modalities of a risk factor between the initial WoE value and the one recalculated based on the predicted target (perfect model), given the number of risk factors in the final model.
The number of risk factors in the final model starts at two and increases to 13, following the order of columns in the simulation dataset.
The target variable used is `Creditability`.

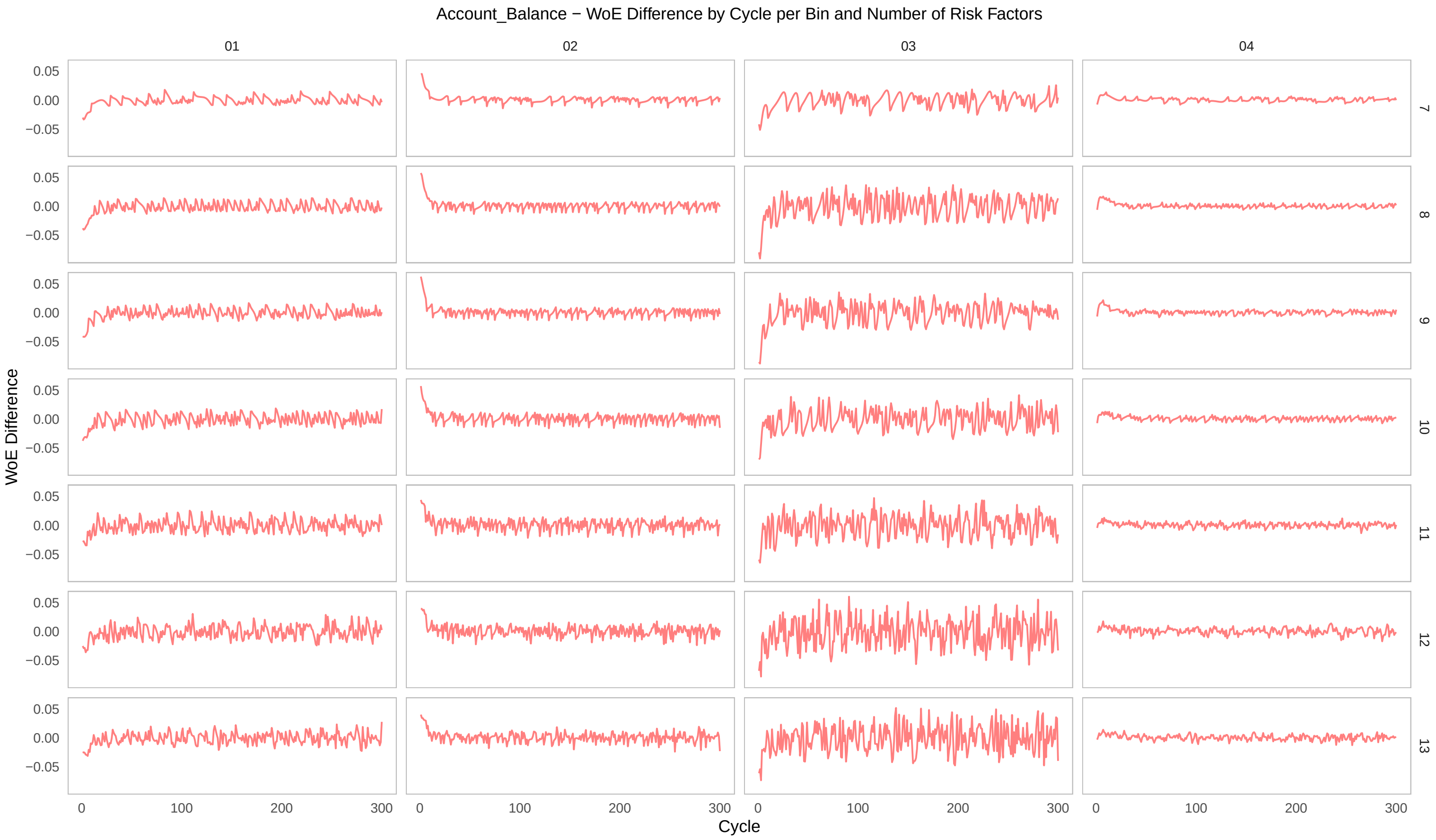
The simulation dataset is available here.

For the sake of simplicity, only the WoE differences for the `Account_Balance` risk factor are presented. Practitioners are also encouraged to test and examine this phenomenon for other risk factors.

Simulation Results



Simulation Results cont.



Discussion Points

- We have found that WoE factor encoding shows a kind of instability or non-replicability: even on the best possible model outcome - perfect prediction - we find that we would rebuild the model differently. This is unsettling, but it is not clear how we should respond, and we open this up for further discussion and development:
    - Precisely what model risks are present in this WoE encoding instability? How far should model validators and model risk managers be concerned about this?
    - How should this concern be expressed, the risk managed? What practices should we change or adopt?
    - What performance outcomes would justify keeping the model unchanged? In some cases it seems that a perfect performance is not good enough: do we seek new kinds of model monitoring?
- We think this instability can be found in all methods that pre-process factors to reduce degrees of freedom, and more generally where upstream model outputs feed models downstream. How large can such instability become? What new insights does it give, or new practices to adopt?
- It is known that complex systems, such as neural networks for image recognition, degrade when they are trained on data that has been labelled by themselves or by other such systems. We think the unstable model replication phenomenon in this note is a similar phenomenon on a simpler scale. If this is so, these problems of machine learning degradation emerge very early, at any complexity above the simplest regression method.

7.3 Discriminatory Power Shortfalls in IRB Credit Risk Models - RWA Impact Analysis

Discriminatory Power Shortfalls in IRB Credit Risk Models

Risk-Weighted Assets Impact Analysis

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Discriminatory Power Shortfalls | 1 / 8

Discriminatory Power in IRB Modeling

- One critical step in IRB model validation is examining the model's discriminatory power, also known as its ranking order power.

- Discriminatory power is evaluated throughout the entire model lifecycle, meaning practitioners assess it during model development, initial validation, and periodic validations.

- Often, discriminatory power is compared against a minimum threshold that the model must meet, and its changes are monitored over time to ensure no significant decline compared to the model development phase.

- The most commonly used metrics for measuring discriminatory power in IRB models are the Area Under the ROC (Receiver Operating Characteristic) Curve and Somers' D.

Consequences of Discriminatory Power Shortfalls

- Recognizing that discriminatory power is a critical aspect of model development and validation, the question often arises: What are the consequences of discriminatory power shortfalls?
- Although they heavily overlap, practitioners typically consider two main dimensions of discriminatory power shortfalls: business and regulatory.
- The business dimension often relates to adverse selection and its effect on portfolio returns, while the regulatory dimension focuses on the impact on Risk-Weighted Assets (RWA).
- Practitioners can refer to this document for further details on adverse selection and its effect on portfolio returns.
- The following slides present a simulation design for measuring the impact of discriminatory power shortfalls in the Probability of Default (PD) on RWA for a model with a discrete rating scale. Practitioners are encouraged to customize the simulation setup to reflect specific assumptions and to explore the combined effects of discriminatory power shortfalls and the potential impact of the model's lack of predictive ability.

Simulation Setup

Note:

As a general rule, improving discriminatory power is expected to decrease the RWA value. However, the opposite trend may occur for certain exposure distributions across rating grades.

Simulation Prerequisites:

- To assess discriminatory power shortfalls, practitioners must first define specific benchmarks.
- Two critical inputs are the current level of discriminatory power relative to the chosen benchmark and the definition of a "perfect" or optimal ranking order for the model.
- Validation procedures typically prescribe a minimum value for the metric used to measure discriminatory power (e.g., an AUC of 75%). Additionally, the impact of specific percentage improvements can be set as an input and used for if-else analysis.
- Practitioners can establish the best or perfect ranking order by independently sorting the target variable and model output in ascending order for the given model. This sorted order can then serve as a benchmark for evaluating the impact of improvements in discriminatory power.

Simulation Setup cont.

Simulation Steps:

Assuming practitioners measure the change in RWA based on a certain percentage of observations improving their ranking order, the following steps outline the simulation design:

1. Define benchmark inputs, including the "perfect" ranking order and the percentage of observations with improved rankings.
2. Cross-tabulate the observed rankings against the "perfect" ranking order.
3. (Optional) Define weights for each ranking mismatch based on the number of observations and the PD differences.
4. Determine the inflows and outflows for each rating based on the improved ranking order of specific observations.
5. Simulate a new allocation of observations per rating based on the calculated inflows and outflows.
6. Recalculate a simulated PD for the new allocation of observations.
7. Compute the RWA for both the original rating scale (RWA_i) and the simulated one (RWA_s).
8. Determine the RWA change using the formula: $\Delta\text{RWA} = \frac{\text{RWA}s - \text{RWA}i}{\text{RWA}_i}$.

The final step is to compare the RWA change against a predefined threshold to evaluate the significance of the discriminatory power shortfall.

Simulation Design:

The objective is to calculate the change in RWA for the rating scale below, with an AUC of 0.7942, assuming a 20% improvement in observations with mismatched ratings.

```
##                 rating   n     pd
## 1     01 (-Inf,-2.206) 223 0.0628
## 2  02 [-2.206,-1.4497) 191 0.1257
## 3 03 [-1.4497,-1.0122) 106 0.1887
## 4 04 [-1.0122,-0.1703) 230 0.3565
## 5  05 [-0.1703,0.5508) 163 0.5828
## 6      06 [0.5508,Inf)  87 0.7471
```

The simulation dataset is available here.

Simulation Results

1. The selected percentage of observations with improved ratings is 20%, while the “perfect” ranking order is summarized in the table below:

```
##          rating_best default_indicator   n
##      01 (-Inf,-2.206)            0.0000 223
##   02 [-2.206,-1.4497)            0.0000 191
##  03 [-1.4497,-1.0122)            0.0000 106
##  04 [-1.0122,-0.1703)            0.2174 230
##   05 [-0.1703,0.5508)            1.0000 163
##       06 [0.5508,Inf)            1.0000  87
```

2. Observed versus “perfect” ranking order ratings:

```
##        rating_initial          rating_best default_indicator   n
##      01 (-Inf,-2.206)     01 (-Inf,-2.206)                 0 209
##      01 (-Inf,-2.206) 04 [-1.0122,-0.1703)                 1  14
##   02 [-2.206,-1.4497)     01 (-Inf,-2.206)                 0  14
##   02 [-2.206,-1.4497)  02 [-2.206,-1.4497)                 0 153
##   02 [-2.206,-1.4497) 04 [-1.0122,-0.1703)                 1  24
##  03 [-1.4497,-1.0122)  02 [-2.206,-1.4497)                 0  38
##  03 [-1.4497,-1.0122) 03 [-1.4497,-1.0122)                 0  48
##  03 [-1.4497,-1.0122) 04 [-1.0122,-0.1703)                 1  12
##  03 [-1.4497,-1.0122)  05 [-0.1703,0.5508)                 1   8
##  04 [-1.0122,-0.1703) 03 [-1.4497,-1.0122)                 0  58
##  04 [-1.0122,-0.1703) 04 [-1.0122,-0.1703)                 0  90
##  04 [-1.0122,-0.1703)  05 [-0.1703,0.5508)                 1  82
##   05 [-0.1703,0.5508) 04 [-1.0122,-0.1703)                 0  68
##   05 [-0.1703,0.5508)  05 [-0.1703,0.5508)                 1  73
##   05 [-0.1703,0.5508)      06 [0.5508,Inf)                 1  22
##       06 [0.5508,Inf) 04 [-1.0122,-0.1703)                 0  22
##       06 [0.5508,Inf)      06 [0.5508,Inf)                 1  65
```

Simulation Results cont.

3. Weights for each ranking mismatch based on the number of observations and the PD differences.

```
##                rating            rating.a  n dr  weight
##      01 (-Inf,-2.206) 04 [-1.0122,-0.1703) 14  1 0.02102
##   02 [-2.206,-1.4497)      01 (-Inf,-2.206) 14  0 0.09819
##   02 [-2.206,-1.4497) 04 [-1.0122,-0.1703) 24  1 0.04584
##  03 [-1.4497,-1.0122)  02 [-2.206,-1.4497) 38  0 0.26589
##  03 [-1.4497,-1.0122) 04 [-1.0122,-0.1703) 12  1 0.03153
##  03 [-1.4497,-1.0122)  05 [-0.1703,0.5508)  8  1 0.00895
##  04 [-1.0122,-0.1703) 03 [-1.4497,-1.0122) 58  0 0.15239
##  04 [-1.0122,-0.1703)  05 [-0.1703,0.5508) 82  1 0.15979
##   05 [-0.1703,0.5508) 04 [-1.0122,-0.1703) 68  0 0.13251
##   05 [-0.1703,0.5508)       06 [0.5508,Inf) 22  1 0.05905
##       06 [0.5508,Inf) 04 [-1.0122,-0.1703) 22  0 0.02484
```

4. Net migrations (inflows and outflows) for each rating based on the improved ranking order of specific observations (`nt`):

```
##                rating   n    net
##      01 (-Inf,-2.206) 223   5.59
##   02 [-2.206,-1.4497) 191   8.82
##  03 [-1.4497,-1.0122) 106 -11.15
##  04 [-1.0122,-0.1703) 230  -4.09
##   05 [-0.1703,0.5508) 163  -1.65
##       06 [0.5508,Inf)  87   2.48
```

5. New allocation of observations per rating based on the net migrations:

```
##                rating    n.s
##      01 (-Inf,-2.206) 228.59
##   02 [-2.206,-1.4497) 199.82
##  03 [-1.4497,-1.0122)  94.85
##  04 [-1.0122,-0.1703) 225.91
##   05 [-0.1703,0.5508) 161.35
##       06 [0.5508,Inf)  89.48
```

Simulation Results cont.

6. Simulated PD for the new allocation of observations.

```
##                rating    n.s   pd.s
##       01 (-Inf,-2.206) 228.59 0.0546
##    02 [-2.206,-1.4497) 199.82 0.1035
##   03 [-1.4497,-1.0122)  94.85 0.1800
##   04 [-1.0122,-0.1703) 225.91 0.3433
##    05 [-0.1703,0.5508) 161.35 0.6380
##        06 [0.5508,Inf)  89.48 0.7742
```

7. The following formula gives the RWA for Revolving Retail Exposures:

$$R = 0.04$$

$$K = LGD \cdot N\left[\frac{G(PD)}{\sqrt{1-R}} + \sqrt{\frac{R}{1-R}} \cdot G(0.999)\right] - PD \cdot \mathrm{LGD}$$

$$RWA = K \cdot 12.5 \cdot EAD$$

For an LGD of 75% and an EAD equal to the number of observations, the initial and simulated RWAs are:

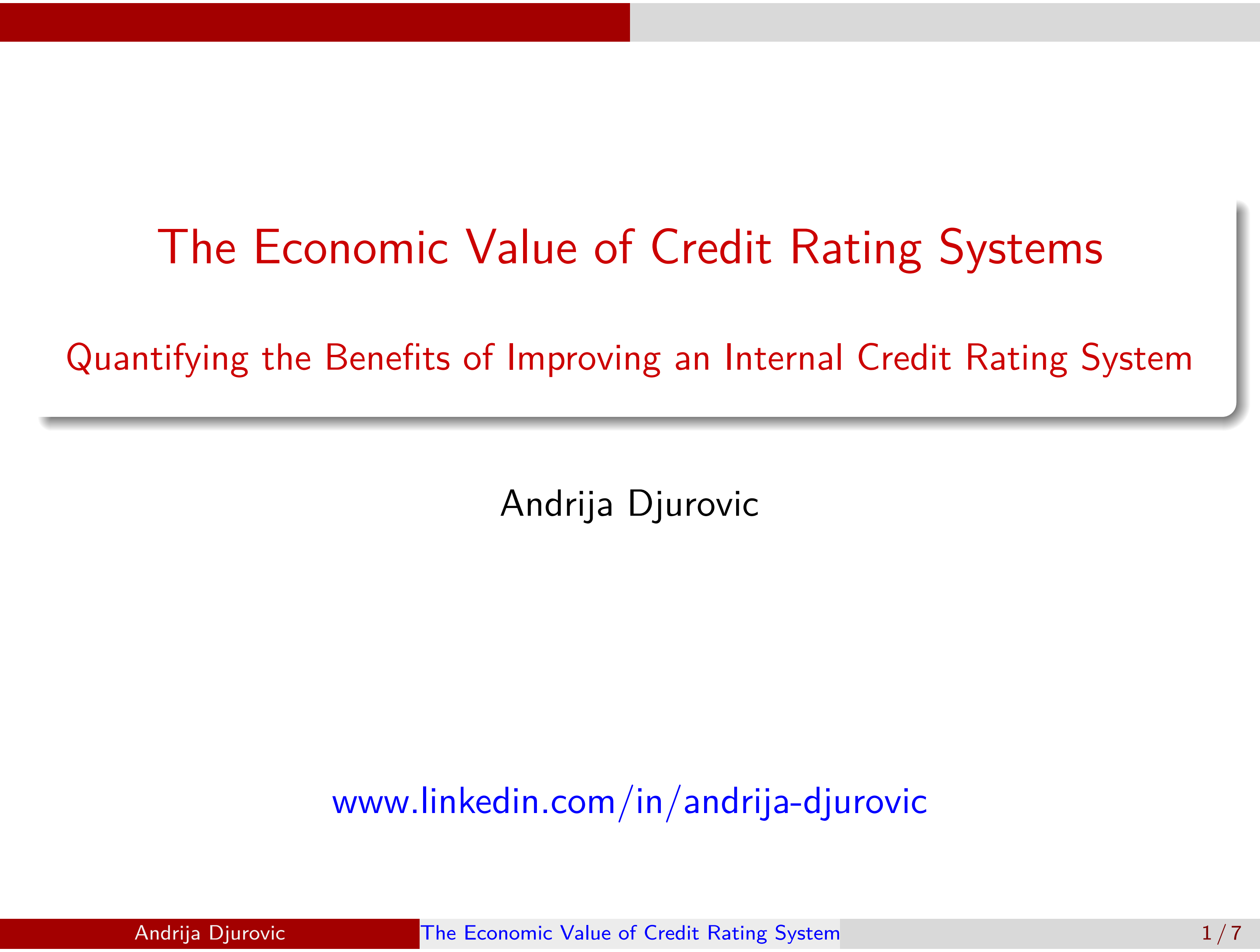
Initial model (RWA_i): 1730.24.
Simulated model (RWA_s): 1625.58.

8. RWA change -6.05%.

Note:

Similar results can be achieved using a simulation approach with random sampling of observations, though this method is more computationally intensive.

7.4 The Economic Value of Credit Rating Systems - Quantifying the Benefits of Improving an Internal Credit Rating System



The Economic Value of Credit Rating Systems

Quantifying the Benefits of Improving an Internal Credit Rating System

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Economic Value of Credit Rating Systems

- Poor statistical power in a bank's internal rating system can negatively impact economic performance due to adverse selection.
- Adverse selection occurs when customers with better credit quality than those assessed by the bank leave, leaving behind a portfolio with lower-than-expected credit quality.
- Enhancing the statistical power of a rating system can positively influence economic performance.
- The magnitude of this positive impact depends on the competitiveness of the market environment.
- Investing in rating system improvements entails organizational, IT, and data management expenses. Therefore, comparing the benefits of enhancement against these costs can be valuable for banks.
- The following slides outline a framework for quantifying the benefits of such an investment. For more details on the proposed framework, refer to this document.

Modeling Framework - Key Concepts and Assumptions

The following points outline the main concepts and assumptions of the modeling framework for evaluating the benefits of an improved credit rating system:

- *Statistical Power*: Improved accuracy in estimating individual Probabilities of Default (PD) reduces adverse selection and impacts pricing.
- *Adverse Selection*: Poor rating systems may misprice risk, leading to higher-quality customers leaving and only riskier clients remaining.
- *Market Competitivity (Customer Elasticity)*: Modeled by the probability that customers switch banks if offered unfavorable credit spreads.
- *Customer Response to Spread Errors*: If overestimated PD results in a high spread, the customer might leave, modeled by elasticity parameter alpha.

The Loan Pricing Mechanism

Assuming the bank adopts risk-adjusted pricing, the loan pricing mechanism can be outlined as follows:

1. The bank must charge an interest rate $\texttt{r}$ to cover all “general” costs unrelated to credit risk and expected losses.
2. Credit risk associated with expected losses is accounted for by adding a credit spread $\texttt{s}$.
3. The credit spread depends on the Probability of Default (PD) and the Loss Given Default (LGD) of the individual exposures. The bank receives $1 + \mathrm{r} + \mathrm{s}$ if no default occurs. If a default occurs, the bank receives $(1 + \mathrm{r} + \mathrm{s}) \cdot (1 - \mathrm{LGD})$.
4. The expected payoff of the loan must equal the risk-free payoff, leading to the equation:

$$1 + \mathrm{r} = (1 - PD) \cdot (1 + \mathrm{r} + \mathrm{s}) + PD \cdot (1 - LGD) \cdot (1 + \mathrm{r} + \mathrm{s})$$

   .

5. Solving for the credit spread $\texttt{s}$ gives:

$$s = (1 + \mathrm{r}) \cdot \frac{PD \cdot LGD}{1 - PD \cdot LGD}$$

   .

The Adverse Selection Concept

Building on the pricing mechanism outlined in the previous slide, the concept of adverse selection can be explained as follows:

1. Customers offered a spread that is too high are likely to leave the bank with a probability that depends on the magnitude `m` of the deviation from the spread corresponding to their true PD. The magnitude of this deviation is defined as:

$$m = s_{estimated} - s_{true} = (1+\mathrm{r}) \cdot \frac{PD_{estimated} \cdot LGD}{1 - PD_{estimated} \cdot LGD} - (1+\mathrm{r}) \cdot \frac{PD_{true} \cdot LGD}{1 - PD_{true} \cdot LGD}$$

2. Assuming $\mathrm{PD}_{\mathrm{estimated}}$ is available with a given measurement error σ, it is defined as:

$$PD_{estimated} = \frac{1}{1 + e^{-}(score_{true} + \sigma)}$$

with the $score_{true}$ given by $ln(\frac{1-PD_{true}}{PD_{true}})$.

3. Based on customer elasticity (α) and the magnitude of the spread deviation (m), the probability of leaving can be defined as:

$$P(Leave) = 1 - e^{-\alpha \cdot \mathrm{m}}$$

This probability models the impact of adverse selection.

4. For customers who remain with the bank after being offered the spread, the individual loan return r_i is defined as:

$$1 + \mathrm{r}_i = \begin{cases} 1 + \mathrm{r} + \mathrm{s} & : \ default = 0 \\ (1 + \mathrm{r} + \mathrm{s}) \cdot (1 - \mathrm{LGD}) & : \ default = 1 \end{cases}$$

5. Finally, given the individual customer returns, the portfolio return is calculated as:

$$r_{\mathrm{portfolio}} = \frac{1}{N} \sum_{i=1}^{N} r_i$$

where N is the number of customers who stayed with the bank.

Simulation Setup

The following points outline the simulation setup:

1. Define `n = 10,000` as the total number of customers.
2. For each customer, simulate the PD using a random beta distribution with parameters `shape1 = 0.7` and `shape2 = 37.6`.
3. For each customer, simulate a default indicator equal to 1 if a random number from the uniform distribution (`min = 0`, `max = 1`) is less than the PD; otherwise, set it to 0.
4. Define measurement errors σ as [2, 0.5, 0.1, 0.01] for rating systems with low, medium, high, and perfect accuracy, respectively.
5. Assume a constant LGD value of 45% for all customers.
6. Assume medium customer elasticity α equals 500.
7. Define the interest rate `r` for all "general" costs as 0.03.
8. Calculate the change (increase) in portfolio returns from transitioning from a rating system with low accuracy to systems with medium, high, and perfect accuracy.

Note:

The above simulation setup is merely designed for simulation purposes. Practitioners are encouraged to test real-world figures and adjust the simulation design to incorporate the effects of other parameters as needed.

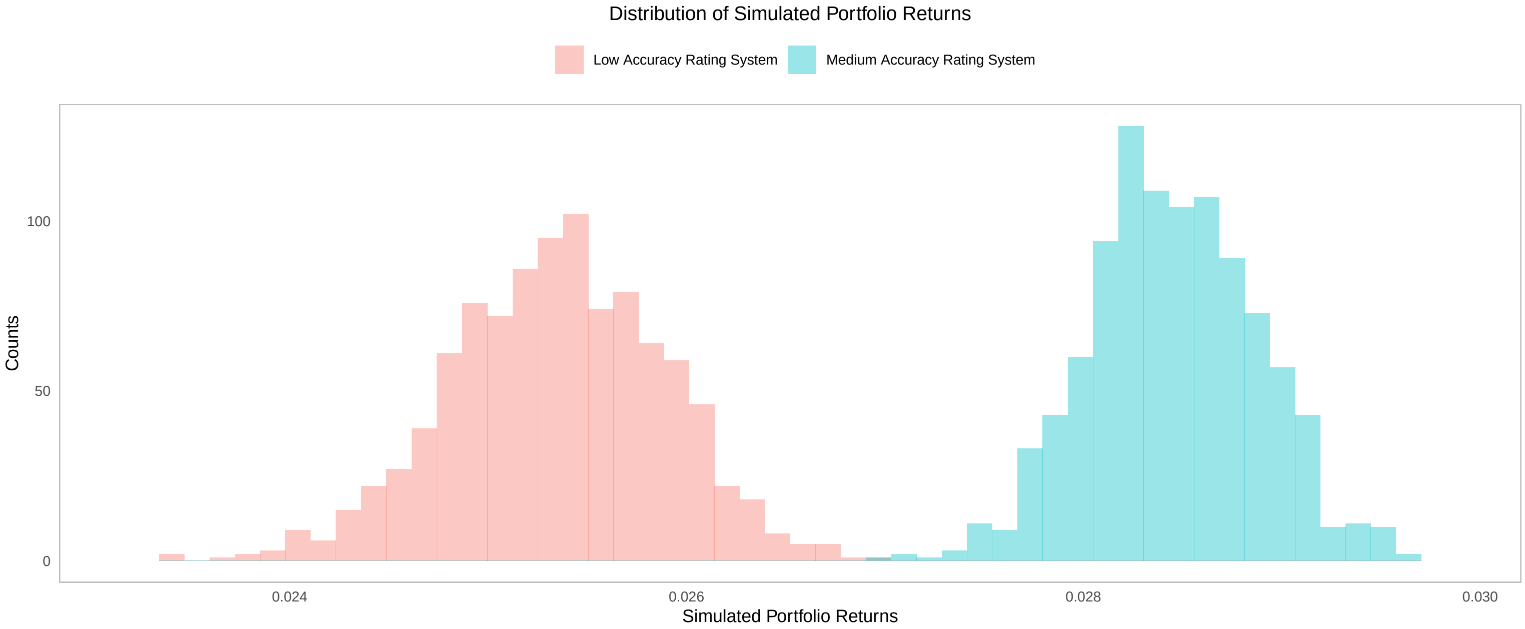
Simulation Results

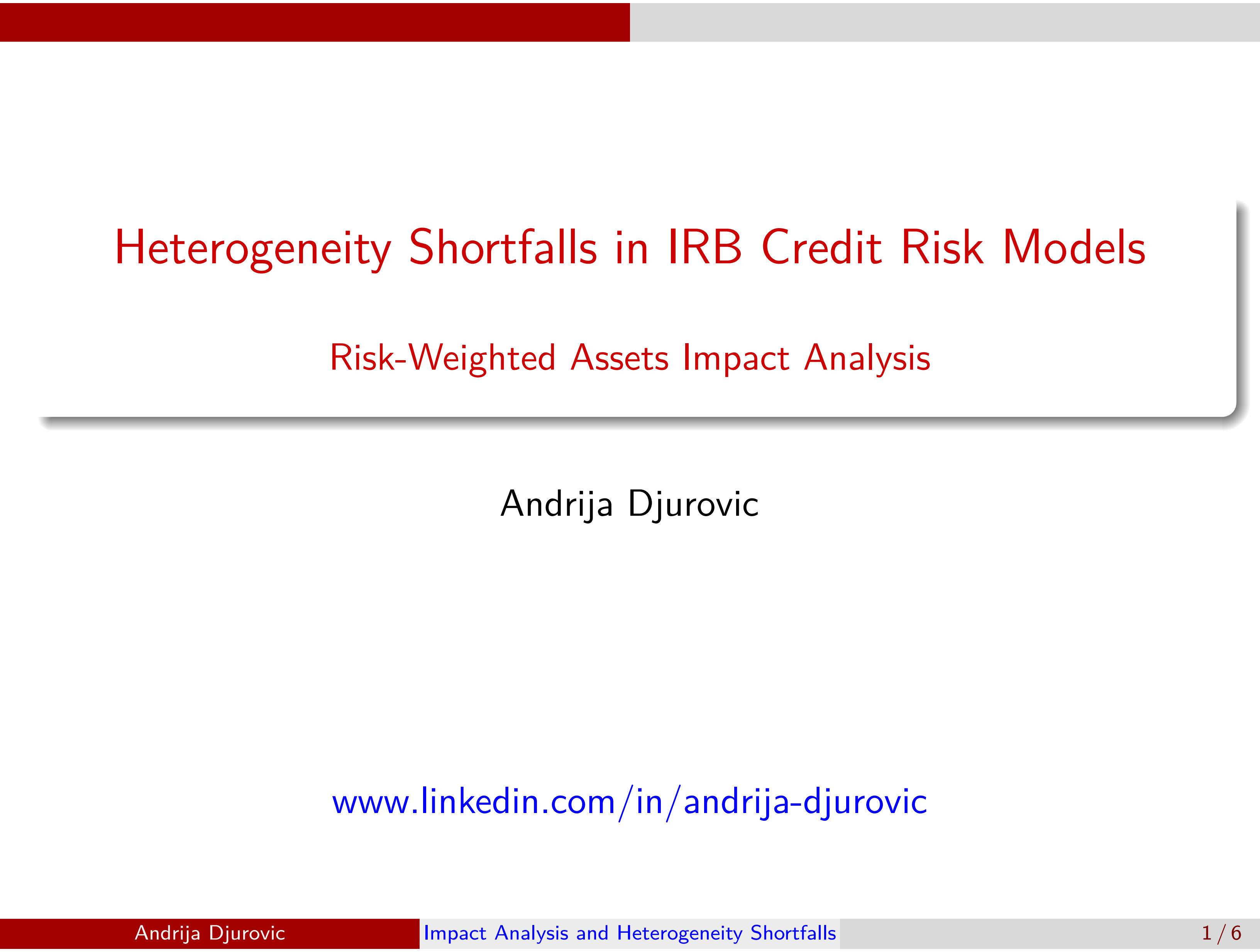
Expected Portfolio Returns

```
##   Accuracy Expected Return
##        Low          0.0253
##     Medium          0.0284
##       High          0.0298
##    Perfect          0.0300
```

Portfolio Returns Increase (in bps.)

```
##       Improvement Return Increase
##   Low  ->  Medium           31.05
##   Low  ->    High           44.85
##   Low  -> Perfect           46.70
```





Heterogeneity Shortfalls in IRB Credit Risk Models

Risk-Weighted Assets Impact Analysis

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Model Heterogeneity

- A typical step in building credit risk models is discretizing the model output into ratings, pools, or buckets.
- Practitioners generally follow established principles for this discretization.
- These principles result in specific characteristics, some of which are mandatory, while others vary by model and are desirable but not essential.
- Monotonicity and heterogeneity are typically regarded as mandatory characteristics.
- In this context, heterogeneity refers to adequate differentiation in risk profiles across ratings, pools, or buckets. It is commonly tested in Probability of Default (PD), Loss Given Default (LGD), and Exposure at Default (EAD) models, often using tests like the two-proportion test and t-test.
- Heterogeneity is usually monitored over time, and practitioners are often challenged to assess the impact of potential heterogeneity shortfalls for thorough model validation.
- The following slides present a simplified simulation design for measuring the impact of heterogeneity shortfalls in the PD rating scale on Risk-Weighted Assets (RWA). Practitioners are encouraged to adjust the simulation setup to reflect specific assumptions and to combine the effect of the heterogeneity shortfall with the potential impact of the model's lack of predictive ability.

Simulation Setup

The following steps outline the simulation design for assessing the impact of a heterogeneity shortfall, measured by the change in RWA:

1. Select the model output (rating scale, pools, or buckets) for a specific exposure type.
2. Test the heterogeneity of adjacent ratings, pools, or buckets.
3. Identify adjacent pairs where heterogeneity testing fails.
4. Locate the pair with the closest risk profiles, indicating a lack of heterogeneity.
5. Merge the adjacent pair identified in step 4 into a single group.
6. After merging the pairs identified in step 5, calculate the weighted average calibrated PD for the merged ratings, pools, or buckets and aggregate any additional elements needed to reassess heterogeneity.
7. Reassess heterogeneity.
8. Repeat steps 3 to 6 as needed.
9. Calculate the RWA for the original calibrated PD (RWA_i) and the weighted PD from step 6 (RWA_s).
10. Calculate the RWA change as: $\Delta\text{RWA} = \frac{\text{RWA}_s - \text{RWA}_i}{\text{RWA}_i}$.

The final step is ideally to compare the RWA change against a specified threshold to assess the significance of the heterogeneity shortfall.

The following slides present simulation results, assuming heterogeneity is tested in the PD rating model using a two-proportion test for revolving retail exposures.

Simulation Results

1. PD rating scale for the revolving retail exposure (`no` - number of observations, `nb` - number of defaults, `pd` - calibrated PD, `odr` - observed default rate):

```
##   rating  no  nb     pd    odr
## 1    R01 170   3 0.0241 0.0176
## 2    R02 118  10 0.0937 0.0847
## 3    R03 274  47 0.1786 0.1715
## 4    R04 100  45 0.3194 0.4500
## 5    R05  91  43 0.4822 0.4725
## 6    R06 196 122 0.6277 0.6224
## 7    R07  51  44 0.8704 0.8627
```

2. Heterogeneity testing (`p-value` - p-value from the two-proportion test for adjacent ratings, `significance level` - selected test significance level, `test results` - test outcome):

```
##   rating p-value significance level           test results
## 1    R01      NA               0.05                   <NA>
## 2    R02  0.0035               0.05  H1: DR(R02) > DR(R01)
## 3    R03  0.0127               0.05  H1: DR(R03) > DR(R02)
## 4    R04  0.0000               0.05  H1: DR(R04) > DR(R03)
## 5    R05  0.3775               0.05 H0: DR(R05) <= DR(R04)
## 6    R06  0.0084               0.05  H1: DR(R06) > DR(R05)
## 7    R07  0.0006               0.05  H1: DR(R07) > DR(R06)
```

Simulation Results cont.

3. Heterogeneity testing failed for the pair R04 - R05.
4. The only pair that failed is the one with the closest risk profiles.
5. Merge ratings R04 and R05 into a single rating R05.
6. Recalculate elements needed for reassessing heterogeneity.
7. Retest heterogeneity on the rating scale with the merged ratings R04 and R05:

```
##   rating      p-value significance level          test results
## 1    R01           NA               0.05                  <NA>
## 2    R02 3.494655e-03               0.05 H1: DR(R02) > DR(R01)
## 3    R03 1.267879e-02               0.05 H1: DR(R03) > DR(R02)
## 4    R05 6.939438e-12               0.05 H1: DR(R05) > DR(R03)
## 6    R06 7.047683e-04               0.05 H1: DR(R06) > DR(R05)
## 7    R07 5.645600e-04               0.05 H1: DR(R07) > DR(R06)
```

Simulation Results cont.

8. The heterogeneity test passes for all adjacent pairs.

9. Adjust the initial rating scale to address the failed heterogeneity for the R04 - R05 pair and add the weighted average calibrated PD (pd.w):

```
##   rating  no  nb     pd    odr rating.m   pd.w
## 1    R01 170   3 0.0241 0.0176      R01 0.0241
## 2    R02 118  10 0.0937 0.0847      R02 0.0937
## 3    R03 274  47 0.1786 0.1715      R03 0.1786
## 4    R04 100  45 0.3194 0.4500      R05 0.3970
## 5    R05  91  43 0.4822 0.4725      R05 0.3970
## 6    R06 196 122 0.6277 0.6224      R06 0.6277
## 7    R07  51  44 0.8704 0.8627      R07 0.8704
```

For the revolving portfolio type, assuming a fixed LGD value of 75% and an equal EAD of 100 for each rating, the RWA is calculated using the following formulas:

$$R = 0.04$$

$$K = LGD \cdot N\left[\frac{G(PD)}{\sqrt{1-R}} + \sqrt{\frac{R}{1-R}} \cdot G(0.999)\right] - PD \cdot \mathrm{LGD}$$

$$RWA = K \cdot 12.5 \cdot EAD$$

with results of RWA_i of 1105.3 and RWA_s of 1115.47

10. The RWA change is recorded as an increase of 0.92% of the RWA_i.

7.6 Heterogeneity Shortfalls in IRB Credit Risk Models - Portfolio Returns Impact Analysis

Heterogeneity Shortfalls in IRB Credit Risk Models

Portfolio Returns Impact Analysis

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Portfolio Returns and Heterogeneity | 1 / 8

Model Heterogeneity

- A typical step in building credit risk models is discretizing the model output into ratings, pools, or buckets.
- Practitioners generally follow established principles for this discretization.
- These principles result in specific characteristics, some of which are mandatory, while others vary by model and are desirable but not essential.
- Monotonicity and heterogeneity are typically regarded as mandatory characteristics.
- In this context, heterogeneity refers to adequate differentiation in risk profiles across ratings, pools, or buckets. It is commonly tested in Probability of Default (PD), Loss Given Default (LGD), and Exposure at Default (EAD) models, often using tests like the two-proportion test and t-test.
- Heterogeneity is usually monitored over time, and practitioners are often challenged to assess the impact of potential heterogeneity shortfalls for thorough model validation.
- Besides the effect on the capital requirements, heterogeneity shortfall can impact the bank's pricing mehanism thus affecting the overall portfolio returns.
- The following slides first introduce the loan pricing mechanism and concept of the adverse selection. and then present simplified simulation for measuring the impact of heterogeneity shortfalls in the PD rating scale on the portfolio returns.
- Details on the economic value of the rating system can be found here.

The Loan Pricing Mechanism

Assuming the bank adopts risk-adjusted pricing, the loan pricing mechanism can be outlined as follows:

1. The bank must charge an interest rate `r` to cover all “general” costs unrelated to credit risk and expected losses.
2. Credit risk associated with expected losses is accounted for by adding a credit spread `s`.
3. The credit spread depends on the Probability of Default (PD) and the Loss Given Default (LGD) of the individual exposures. The bank receives $1 + \mathrm{r} + \mathrm{s}$ if no default occurs. If a default occurs, the bank receives $(1 + \mathrm{r} + \mathrm{s}) \cdot (1 - \mathrm{LGD})$.
4. The expected payoff of the loan must equal the risk-free payoff, leading to the equation:

$$1 + \mathrm{r} = (1 - PD) \cdot (1 + \mathrm{r} + \mathrm{s}) + PD \cdot (1 - LGD) \cdot (1 + \mathrm{r} + \mathrm{s})$$

.

5. Solving for the credit spread `s` gives:

$$s = (1 + \mathrm{r}) \cdot \frac{PD \cdot LGD}{1 - PD \cdot LGD}$$

.

The Adverse Selection Concept

Building on the pricing mechanism outlined in the previous slide, the concept of adverse selection can be explained as follows:

1. Customers offered a spread that is too high are likely to leave the bank with a probability that depends on the magnitude `m` of the deviation from the spread corresponding to their true PD. The magnitude of this deviation is defined as:

$$m = s_{estimated} - s_{true} = (1+\mathrm{r}) \cdot \frac{PD_{estimated} \cdot LGD}{1 - PD_{estimated} \cdot LGD} - (1+\mathrm{r}) \cdot \frac{PD_{true} \cdot LGD}{1 - PD_{true} \cdot LGD}$$

2. Assuming $\mathrm{PD}_{\mathrm{estimated}}$ is available with a given measurement error σ, it is defined as:

$$PD_{estimated} = \frac{1}{1 + e^{-}(score_{true} + \sigma)}$$

with the $score_{true}$ given by $ln(\frac{1-PD_{true}}{PD_{true}})$.

3. Based on customer elasticity (α) and the magnitude of the spread deviation (m), the probability of leaving can be defined as:

$$P(Leave) = 1 - e^{-\alpha \cdot \mathrm{m}}$$

This probability models the impact of adverse selection.

4. For customers who remain with the bank after being offered the spread, the individual loan return r_i is defined as:

$$1 + \mathrm{r}_i = \begin{cases} 1 + \mathrm{r} + \mathrm{s} & : \; default = 0 \\ (1 + \mathrm{r} + \mathrm{s}) \cdot (1 - \mathrm{LGD}) & : \; default = 1 \end{cases}$$

5. Finally, given the individual customer returns, the portfolio return is calculated as:

$$r_{\mathrm{portfolio}} = \frac{1}{N} \sum_{i=1}^{N} r_i$$

where N is the number of customers who stayed with the bank.

Simulation Setup

The following steps outline the simulation design for assessing the impact of a heterogeneity shortfall, measured by the change in portfolio returns:

1. Test the heterogeneity of adjacent ratings, pools, or buckets.
2. Identify adjacent pairs where heterogeneity testing fails.
3. Locate the pair with the closest risk profiles, indicating a lack of heterogeneity.
4. Merge the adjacent pair identified in step 4 into a single group.
5. After merging the pairs identified in step 5, calculate the weighted average calibrated PD for the merged ratings, pools, or buckets and aggregate any additional elements needed to reassess heterogeneity.
6. Reassess heterogeneity.
7. Repeat steps 3 to 6 as needed.
8. Calculate standard error of the score transformed PD values for the adjacent pairs for which heterogeneity shortfall is observed.
9. Simulate the effects of the adverse selection process adding estimation error only to the adjacent pair for which heterogeneity shortfall is observed and for the selected of general interest rate - r, customer elasticity α, LGD, calculate the average change of the portfolio returns.
10. Compare the average of the simulated portfolio returns to the expected portfolio return under the assumption of the true PDs given by the initial rating scale.

Besides comparing solely the average simulated portfolio return and the expected one, practitioners can inspect the distribution of this difference.

Simulation Results

1. PD rating scale (`rating` - rating, `no` - number of observations, `nb` - number of defaults, `pd` - calibrated PD, `odr` - observed default rate):

```
##   rating  no  nb     pd    odr
## 1    R01 170   3 0.0241 0.0176
## 2    R02 118  10 0.0937 0.0847
## 3    R03 274  47 0.1786 0.1715
## 4    R04 100  45 0.3194 0.4500
## 5    R05  91  43 0.4822 0.4725
## 6    R06 196 122 0.6277 0.6224
## 7    R07  51  44 0.8704 0.8627
```

2. Heterogeneity testing (`p-value` - p-value from the two-proportion test for adjacent ratings, `significance level` - selected test significance level, `test results` - test outcome):

```
##   rating rating.m   pd.w        p.val alpha                   res
## 1    R01      R01 0.0241           NA  0.05                  <NA>
## 2    R02      R02 0.0937 3.494655e-03  0.05  H1: DR(R02) > DR(R01)
## 3    R03      R03 0.1786 1.267879e-02  0.05  H1: DR(R03) > DR(R02)
## 4    R04      R05 0.3970 1.561927e-08  0.05  H1: DR(R04) > DR(R03)
## 5    R05      R05 0.3970 3.775379e-01  0.05 H0: DR(R05) <= DR(R04)
## 6    R06      R06 0.6277 8.407305e-03  0.05  H1: DR(R06) > DR(R05)
## 7    R07      R07 0.8704 5.645600e-04  0.05  H1: DR(R07) > DR(R06)
```

Simulation Results cont.

3. Heterogeneity testing failed for the pair R04 - R05.

4. The only pair that failed is the one with the closest risk profiles.

5. Merge ratings R04 and R05 into a single rating R05.

6. Recalculate elements needed for reassessing heterogeneity.

7. Retest heterogeneity on the rating scale with the merged ratings R04 and R05:

   ```
   ##   rating      p-value significance level          test results
   ## 1    R01           NA             0.05                   <NA>
   ## 2    R02 3.494655e-03             0.05 H1: DR(R02) > DR(R01)
   ## 3    R03 1.267879e-02             0.05 H1: DR(R03) > DR(R02)
   ## 4    R05 6.939438e-12             0.05 H1: DR(R05) > DR(R03)
   ## 6    R06 7.047683e-04             0.05 H1: DR(R06) > DR(R05)
   ## 7    R07 5.645600e-04             0.05 H1: DR(R07) > DR(R06)
   ```

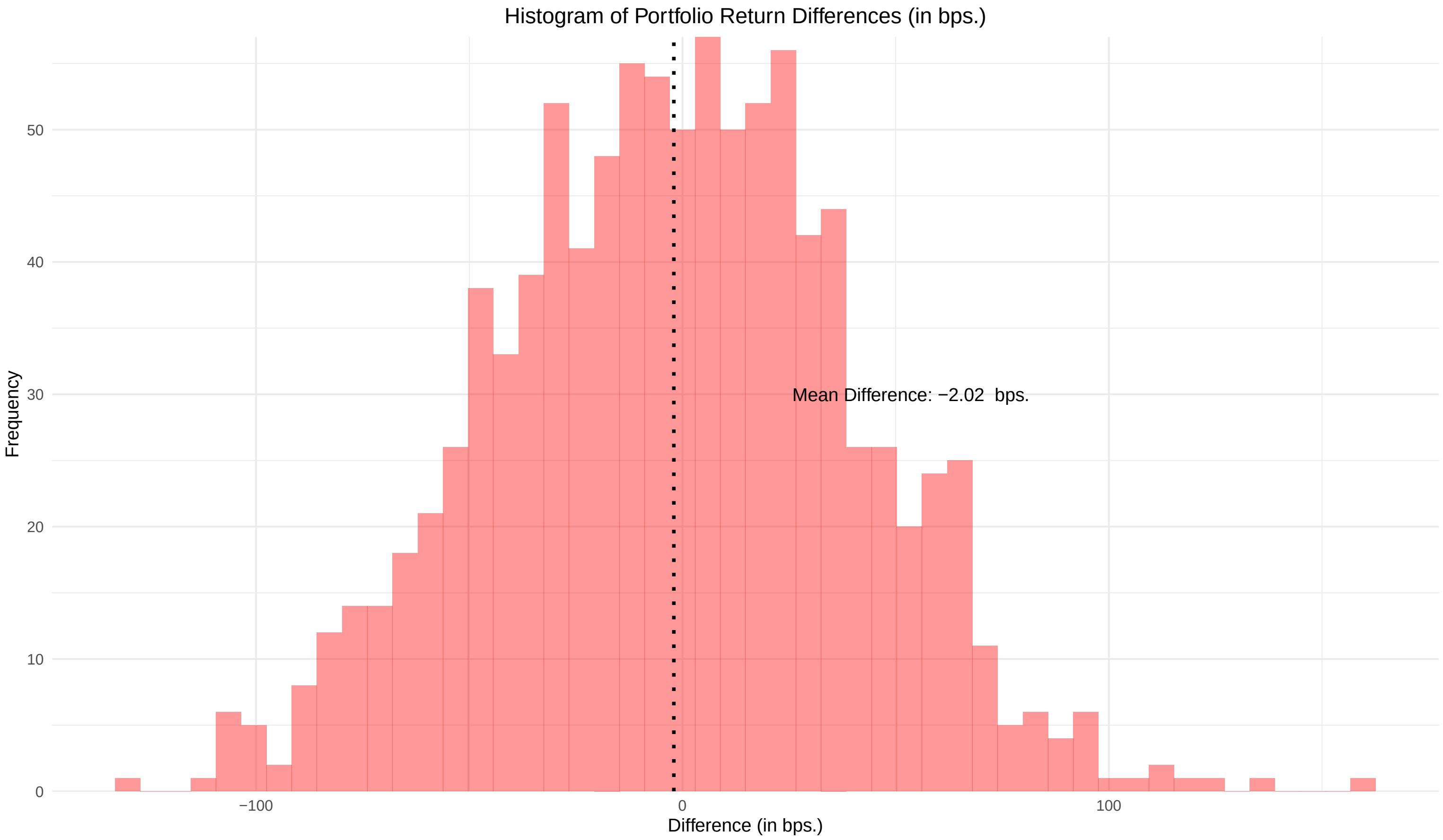
   The heterogeneity test passes for all adjacent pairs.

8. Calculate standard error of the score transformed PD values for the adjacent pairs for which heterogeneity shortfall is observed - $\sigma = 0.34$

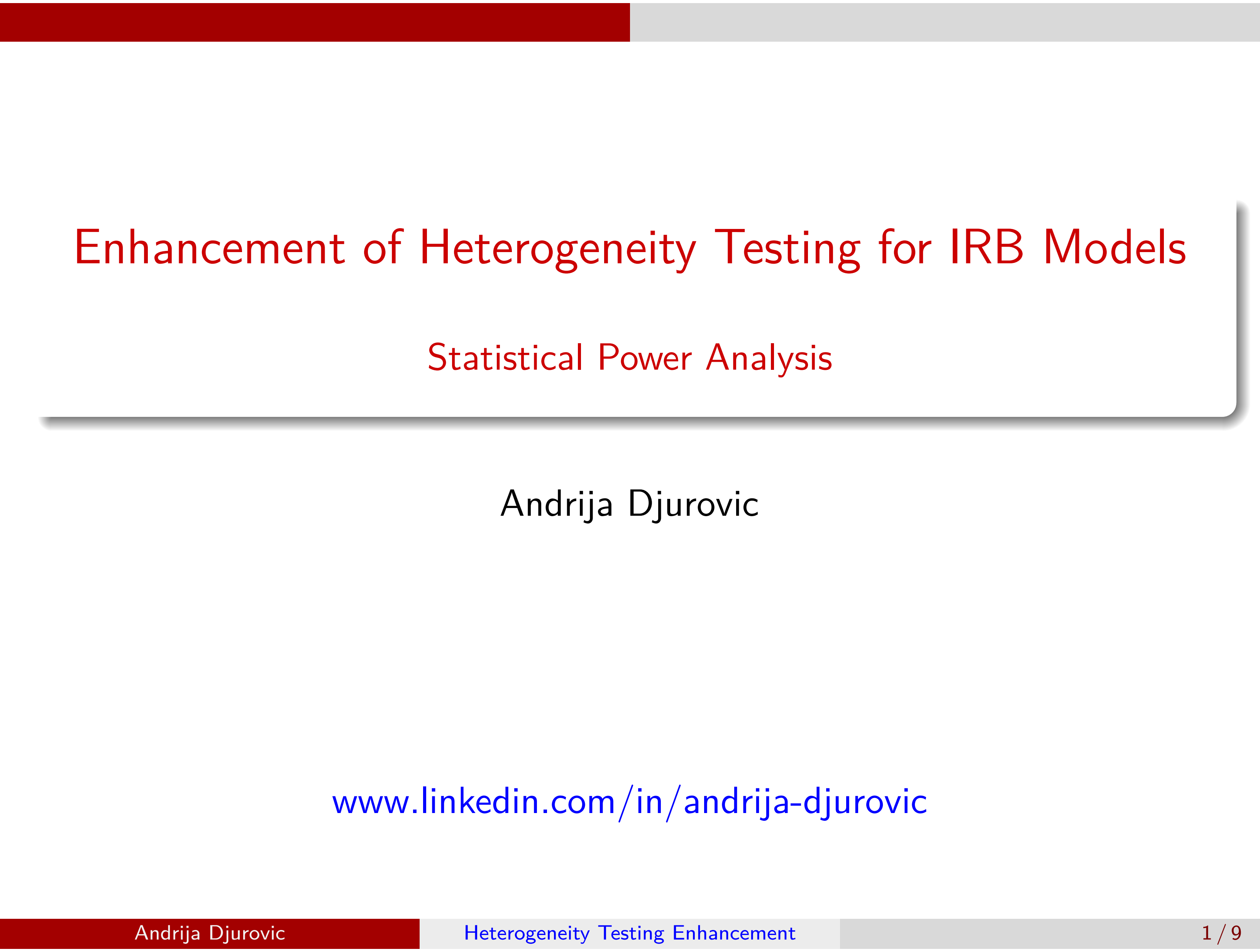
9. Run 1,000 simulations of the effects of the adverse selection process by adding estimation error only to the adjacent pair for which heterogeneity shortfall is observed and under assumptions of general interest rate $r = 0.03$, customer elasticity $\alpha = 500$, $LGD = 0.75$. Calculate the average change of the portfolio returns.

10. Compare the average simulated portfolio return and the expected one:
    Simulated average portfolio return: 0.0234349;
    Expected portfolio return: 0.0236373;
    Difference (in bps.): -2.02.

Simulation Results cont.



7.7 Enhancement of Heterogeneity Testing for IRB Models - Statistical Power Analysis



Enhancement of Heterogeneity Testing for IRB Models

Statistical Power Analysis

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Model Heterogeneity

- A typical step in building credit risk models is discretizing the model output into ratings, pools, or buckets.
- Practitioners generally follow established principles for this discretization.
- These principles result in specific characteristics, some of which are mandatory, while others vary by model and are desirable but not essential.
- Monotonicity and heterogeneity are typically regarded as mandatory characteristics.
- In this context, heterogeneity refers to adequate differentiation in risk profiles across ratings, pools, or buckets. It is commonly tested in Probability of Default (PD), Loss Given Default (LGD), and Exposure at Default (EAD) models, often using tests like the two-proportion test and t-test.
- Heterogeneity is assessed during the initial model validation and monitored over time.

Practical Challenges in Heterogeneity Testing

- A common approach to heterogeneity testing relies on observed (realized) average values between adjacent grades, pools, or buckets.

- Additionally, heterogeneity is often tested at the overall modeling dataset level during initial model validation and at a specific reference date or across a few consolidated reference dates during periodic model validation.

- Given the different possible levels of testing, practitioners often face challenges in assessing overall heterogeneity. A common approach is to define a percentage of pairs for which the specified test fails.

- Analyzing the common approach used in practice, a few questions arise:

  - Is having a unified threshold for each portfolio type or rating scale/pools sufficient?
  - Is it sufficient to assess heterogeneity using only observed averages?
  - If the model is well-calibrated, what are the conclusions of heterogeneity testing?
  - If the model is well-calibrated, what is the probability of observing differences between adjacent grades, pools, or buckets?
  - What statistical methods can help practitioners support heterogeneity analysis?

- The following slides present a simplified simulation using statistical power analysis to support heterogeneity testing by providing the probabilities of identifying the difference between the PDs of adjacent rating grades in the case of a well-calibrated model. Building on the results from the power analysis, the final slide presents the probabilities of failure for different numbers of rating pairs in the PD model. Practitioners are encouraged to adjust the simulation setup to reflect specific assumptions and to combine the effect of heterogeneity shortfall with the potential impact of the model's lack of predictive ability.

- A similar approach can be used to calculate the probabilities of monotonicity disruption for different numbers of rating, pool, or bucket pairs.

- Enhancing the standard approach to heterogeneity testing can help practitioners better understand this aspect of model validation and support conclusions with additional valuable analysis.

Simulation Setup

Dataset

The dataset used for the following simulation is shown below:

```
##   rating   no nb    odr     pd  p.val
##      RG1 1500  6 0.0040 0.0057     NA
##      RG2 1920 14 0.0073 0.0105 0.1051
##      RG3 2925 36 0.0123 0.0169 0.0455
##      RG4 4515 95 0.0210 0.0310 0.0026
##      RG5 2535 90 0.0355 0.0530 0.0001
##      RG6 1365 83 0.0608 0.0793 0.0001
##      RG7   91  9 0.0989 0.1451 0.0741
##      RG8  148 26 0.1757 0.2590 0.0515
```

where:

- `rating` denotes the rating grade;
- `no` represents the number of obligors per rating grade;
- `odr` is the observed default rate per rating grade;
- `pd` is the calibrated PD estimate;
- `p.val` is the p-value of the heterogeneity test (test of two proportions) for adjacent rating pairs.

Statistical Power Analysis

- The following slides demonstrate the calculation of statistical power for adjacent rating grades under the assumption that the model is well-calibrated (i.e., the true PDs are equal to the calibrated PDs).
- In this context, statistical power provides insights into the probability of correctly identifying differences between PDs in adjacent pairs if the true PDs match the calibrated ones.
- Results are presented using an analytical solution and a Monte Carlo simulation.
- Finally, the statistical power results from the Monte Carlo simulation are used to calculate and graphically present the probability of heterogeneity failure for different numbers of rating pairs.

R Code

```r
#utils functions import
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/mrm/utils.R")

#data import
url <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/mrm/rs.csv"
rs <- read.csv(file = url,
               header = TRUE)

#statistical power - analytical solution
rs$power.a <- NA
for (i in 2:nrow(rs)) {
      rs$power.a[i] <- ht.power.a(n1 = rs$no[i-1],
                                  n2 = rs$no[i],
                                  p1 = rs$pd[i-1],
                                  p2 = rs$pd[i],
                                  alpha = 0.05)
      }
rs$power.a <- round(x = rs$power.a,
                    digits = 4)
#print rs data frame
rs
```

```
##   rating   no nb    odr     pd  p.val power.a
## 1    RG1 1500  6 0.0040 0.0057     NA      NA
## 2    RG2 1920 14 0.0073 0.0105 0.1051  0.4512
## 3    RG3 2925 36 0.0123 0.0169 0.0455  0.5775
## 4    RG4 4515 95 0.0210 0.0310 0.0026  0.9882
## 5    RG5 2535 90 0.0355 0.0530 0.0001  0.9970
## 6    RG6 1365 83 0.0608 0.0793 0.0001  0.9350
## 7    RG7   91  9 0.0989 0.1451 0.0741  0.6700
## 8    RG8  148 26 0.1757 0.2590 0.0515  0.6777
```

R Code cont.

```r
#statistical power - mc simulation
rs$power.s <- NA
for   (i in 2:nrow(rs)) {
      rs$power.s[i] <- ht.power.s(n1 = rs$no[i-1],
                                  n2 = rs$no[i],
                                  p1 = rs$pd[i-1],
                                  p2 = rs$pd[i],
                                  alpha = 0.05,
                                  sim = 1e4,
                                  seed = 2211 + i)
      }
rs$power.s <- round(x = rs$power.s,
                    digits = 4)
#print rs data frame
rs
```

```
##   rating   no nb    odr     pd  p.val power.a power.s
## 1    RG1 1500  6 0.0040 0.0057     NA      NA      NA
## 2    RG2 1920 14 0.0073 0.0105 0.1051  0.4512  0.4550
## 3    RG3 2925 36 0.0123 0.0169 0.0455  0.5775  0.5837
## 4    RG4 4515 95 0.0210 0.0310 0.0026  0.9882  0.9887
## 5    RG5 2535 90 0.0355 0.0530 0.0001  0.9970  0.9969
## 6    RG6 1365 83 0.0608 0.0793 0.0001  0.9350  0.9362
## 7    RG7   91  9 0.0989 0.1451 0.0741  0.6700  0.6618
## 8    RG8  148 26 0.1757 0.2590 0.0515  0.6777  0.6878
```

Python Code

```python
import pandas as pd
import numpy as np
from scipy.stats import norm, binom
from statsmodels.stats.proportion import proportions_ztest
import requests
#utils functions import
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/mrm/utils.py"
r = requests.get(url)
exec(r.text)
#data import
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/mrm/rs.csv"
rs = pd.read_csv(filepath_or_buffer = url)
#statistical power - analytical solution
rs["power_a"] = np.nan
for i in range(1, len(rs)):
    rs.loc[i, "power_a"] = ht_power_a(n1 = rs.loc[i-1, "no"],
                                      n2 = rs.loc[i, "no"],
                                      p1 = rs.loc[i-1, "pd"],
                                      p2 = rs.loc[i, "pd"],
                                      alpha = 0.05)

rs["power_a"] = rs["power_a"].round(4)
#print rs data frame
rs
```

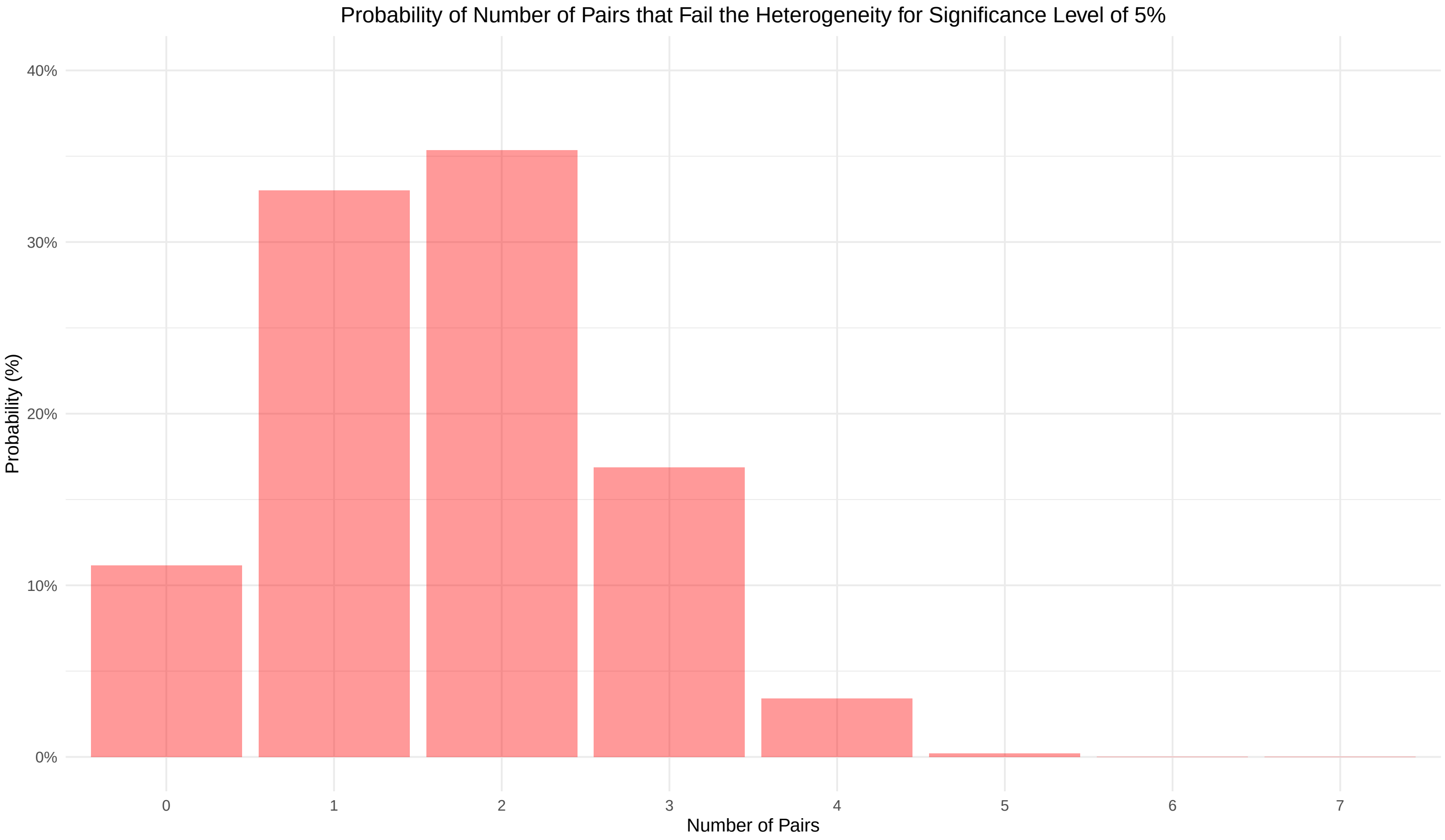
```
##   rating    no  nb     odr      pd   p.val  power_a
## 0    RG1  1500   6  0.0040  0.0057     NaN      NaN
## 1    RG2  1920  14  0.0073  0.0105  0.1051   0.4512
## 2    RG3  2925  36  0.0123  0.0169  0.0455   0.5775
## 3    RG4  4515  95  0.0210  0.0310  0.0026   0.9882
## 4    RG5  2535  90  0.0355  0.0530  0.0001   0.9970
## 5    RG6  1365  83  0.0608  0.0793  0.0001   0.9350
## 6    RG7    91   9  0.0989  0.1451  0.0741   0.6700
## 7    RG8   148  26  0.1757  0.2590  0.0515   0.6777
```

Python Code cont.

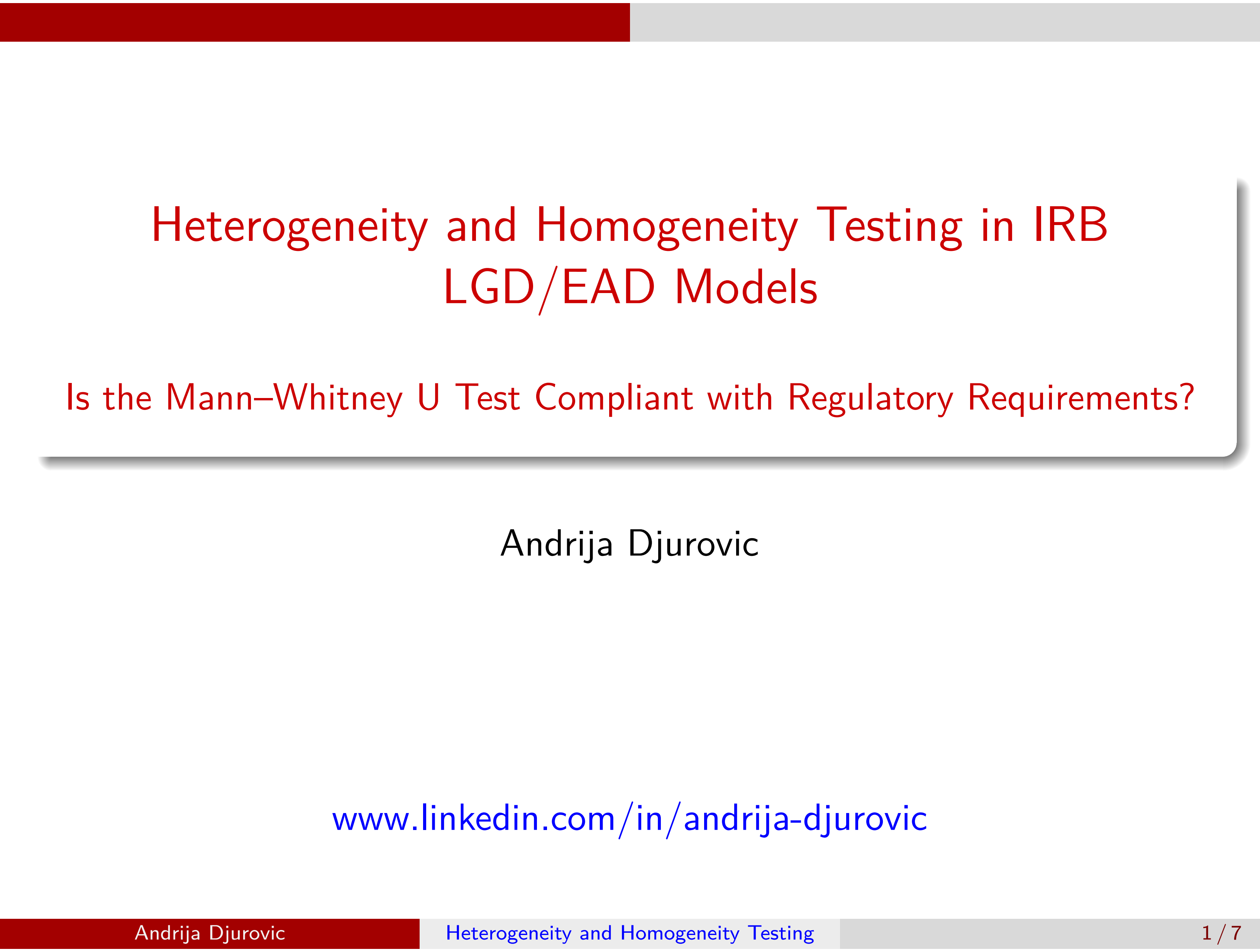
```python
#statistical power - mc simulation
rs["power_s"] = np.nan
for i in range(1, len(rs)):
    rs.loc[i, "power_s"] = ht_power_s(n1 = rs.loc[i-1, "no"],
                                      n2 = rs.loc[i, "no"],
                                      p1 = rs.loc[i-1, "pd"],
                                      p2 = rs.loc[i, "pd"],
                                      alpha = 0.05,
                                      sim = 10000,
                                      seed = 2211 + i)

rs["power_s"] = rs["power_s"].round(4)
#print rs data frame
rs
```

```
##   rating    no  nb     odr      pd   p.val  power_a  power_s
## 0    RG1  1500   6  0.0040  0.0057     NaN      NaN      NaN
## 1    RG2  1920  14  0.0073  0.0105  0.1051   0.4512   0.4585
## 2    RG3  2925  36  0.0123  0.0169  0.0455   0.5775   0.5933
## 3    RG4  4515  95  0.0210  0.0310  0.0026   0.9882   0.9875
## 4    RG5  2535  90  0.0355  0.0530  0.0001   0.9970   0.9973
## 5    RG6  1365  83  0.0608  0.0793  0.0001   0.9350   0.9408
## 6    RG7    91   9  0.0989  0.1451  0.0741   0.6700   0.6645
## 7    RG8   148  26  0.1757  0.2590  0.0515   0.6777   0.6940
```

Simulation Results



7.8 Heterogeneity and Homogeneity Testing in IRB LGD/EAD Models - Is the Mann–Whitney U Test Compliant with Regulatory Requirements?



Heterogeneity and Homogeneity Testing in IRB LGD/EAD Models

Is the Mann–Whitney U Test Compliant with Regulatory Requirements?

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Heterogeneity and Homogeneity Testing | 1 / 7

Regulatory Requirements

- Heterogeneity and homogeneity are key aspects in analyzing the distribution and allocation of obligors and facilities in IRB models.
- In this context, heterogeneity refers to adequate differentiation in risk profiles across ratings, pools, or buckets.
- Homogeneity, however, means that all exposures within a grade or pool exhibit similar risk characteristics.
- Paragraph 175, Credit risk, of the ECB Guide to Internal Models outlines regulatory expectations regarding meaningful risk differentiation and the quantification of LGD models.
- This paragraph explicitly addresses the requirements for heterogeneity and homogeneity in LGD models, in addition to expectations about the adequate distribution of facilities across pools.
- While it does not prescribe specific statistical methods, points (b) and (c) of Paragraph 175, Credit risk set precise requirements for demonstrating sufficient levels of heterogeneity and homogeneity between and within pools based on average realized values.
- Although not explicitly stated, these exact requirements are, in practice, also applied to EAD models.
- The explicit requirement to provide empirical evidence based on the average realized values raises the question of which statistical tests are appropriate and compliant with regulatory expectations.

Heterogeneity and Homogeneity Testing in Practise

- Heterogeneity and homogeneity testing are typically conducted based on realized values within and between pools.
- The most commonly used statistical tests are the Welch t-test and the Mann–Whitney U test.
- Practitioners often assess whether the values follow a normal distribution to determine which test is more appropriate for the distribution of LGD or CCF realized values. If normality is confirmed, the Welch t-test is used; otherwise, the Mann–Whitney U test is applied.
- Given the explicit regulatory requirement to base conclusions on average realized values when assessing heterogeneity and homogeneity, it is questionable whether the commonly used approach is appropriate. Specifically, the Mann–Whitney U test does not test for differences in means, which may lead to incorrect conclusions and non-compliance with regulatory expectations.
- The following slides will present the test statistics for both methods, demonstrate how they can yield conflicting results, and show how applying the Mann–Whitney U test under the wrong null hypothesis may lead to an inflated Type I error for typical distributions of realized LGD or CCF values.

Welch T-Test

The test statistic for the Welch t-test, which is used to compare the means of two samples with potentially unequal variances, is defined as:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}$$

where:

- $\bar{x}_1$ and $\bar{x}_2$ are the sample means;
- s_1^2 and s_2^2 are the sample variances;
- n_1 and n_2 are the sample sizes.

This test statistic follows a t-distribution with degrees of freedom given by the Welch–Satterthwaite equation:

$$\text{df} = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2 - 1}}$$

To assess whether the means differ significantly, a p-value is computed from the t-distribution using the calculated test statistic and degrees of freedom.

Mann–Whitney U Test

The Mann-Whitney U test statistic is defined as:

$$U = \sum_{i=1}^{n_x} R(x_i) - \frac{n_x(n_x+1)}{2}$$

where $R(x_i)$ is the rank of the i-th observation from group x in the combined sample.

The mean and standard deviation of U under the null hypothesis are:

$$\mu_U = \frac{n_x n_y}{2}, \quad \sigma_U = \sqrt{\frac{n_x n_y (n_x + n_y + 1)}{12} - \sum_t \frac{t^3 - t}{(n_x + n_y)(n_x + n_y - 1)}}$$

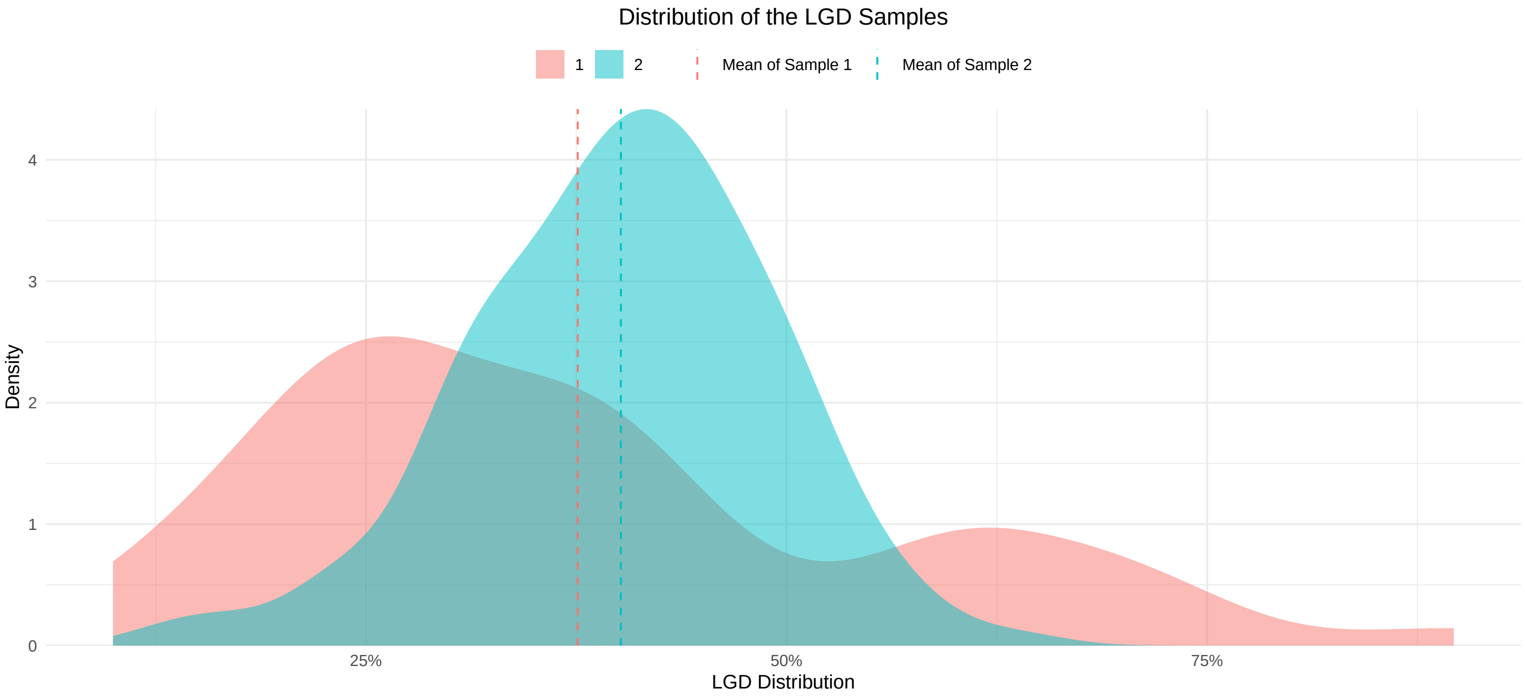
where n_x and n_y are the sample sizes and t represents the number of tied ranks in the data. The standardized test statistic is:

$$Z = \frac{U - \mu_U}{\sigma_U}$$

is approximately normally distributed under the null hypothesis, and the p-value is calculated accordingly. Unlike the Welch t-test, which tests for a difference in means, the Mann–Whitney U test assesses stochastic superiority.

Practical Challenges of Conflicting Results

It is relatively easy in practice to observe or simulate situations where the Welch t-test and Mann–Whitney U test lead to different conclusions. This is especially true for the distribution of realized LGD and EAD models, where the variances can differ across pools. One such example is available here. The observed p-values are 34.54% for the two-sided Welch t-test and 1.37% for the Mann–Whitney U test. The graph below shows the distribution of simulated values per sample.



Simulation Study

If practitioners perform the Mann–Whitney U test under the incorrect assumption that it tests for equality of means, it is easy to show that the Type I error (the percentage of times the null hypothesis is incorrectly rejected) is significantly inflated compared to the Welch t-test. The steps below outline one such simulation design and reveal the simulation results. Simulation assumes the realized values come from a beta distribution with specific parameters, featuring equal means (0.40) but different variances (0.20 and 0.1069).

Simulation Design

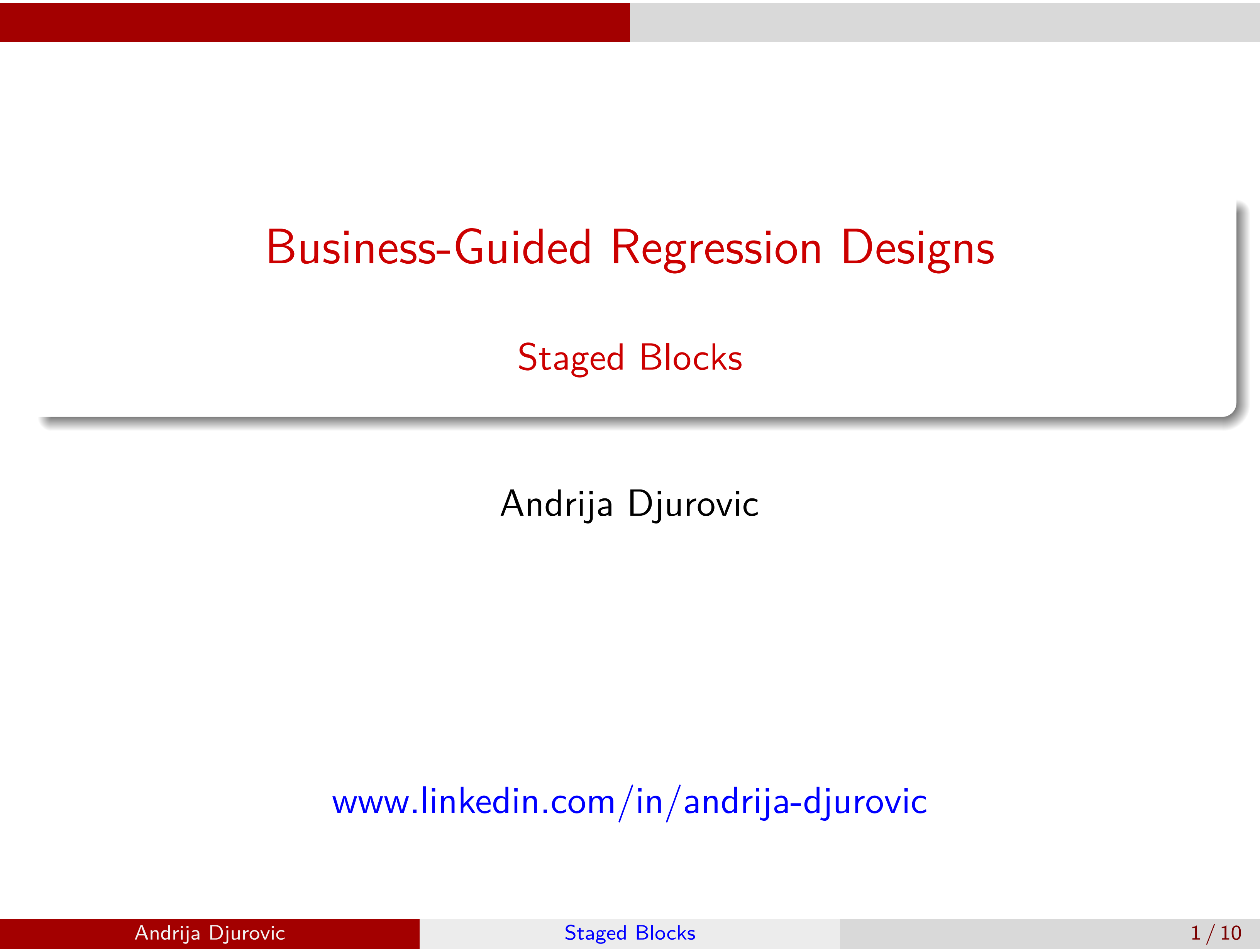
1. Simulate values for the first sample by randomly drawing 50 values from a beta distribution with shape parameters $\alpha = 2$ and $\beta = 3$.
2. Simulate values for the second sample by randomly drawing 150 values from a beta distribution with shape parameters $\alpha = 8$ and $\beta = 12$.
3. Calculate p-values for the two-sided Welch t-test and the Mann–Whitney U test.
4. Repeat steps 1 to 3 10,000 times and store the resulting p-values.
5. Calculate the Type I error as the percentage of simulations that reject the null hypothesis at a significance level of 10%.

Simulation Results

Running the above simulation results in Type I errors of 9.69% for the Welch t-test and 18.46% for the Mann–Whitney U test, respectively.

8 Business-Guided Regression Designs

8.1 Staged Blocks



Business-Guided Regression Designs

Staged Blocks

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Business-Guided Model Designs

- Developing credit risk models is not just a regulatory exercise; first and foremost, they should serve business needs.
- Recognizing this, businesses and modelers should collaborate closely when designing models.
- Often, model quality is not solely determined by strong statistical performance but by how well it supports business needs while maintaining sufficient statistical performance.
- Designing a model is not just about collaboration between businesses and modelers in selecting risk factors for the final model. It is usually a highly iterative process that incorporates multiple dimensions.
- A recent trend in credit risk model development shows that practitioners often opt for a blockwise or modular approach rather than developing a model using a fully consolidated dataset.
- Providing inputs for various modeling areas and ensuring robust statistical modeling are key objectives for businesses and modelers.
- Good practice shows that inputs from business experts plays a key role in model development. Modelers consider and incorporate these inputs into the overall model design.
- This presentation focuses on the staged blocks approach, which is briefly described in the following slides. The rest of the presentation provides a simplified simulation study demonstrating the implementation of this design in `R` and `Python`.

Staged Blocks Approach

- Practitioners usually prefer a blockwise (modular) approach over running a single optimization algorithm on a consolidated modeling dataset.
- This modular strategy employs multiple submodels or blocks based on business inputs, data coverage, and industry insights. These submodels are then combined to form a unified model score.
- One blockwise approach is the staged approach, where predictions from the previous block serve as offsets for models in subsequent blocks.
- The rationale for staging can be related to:
  - data sources, particularly their cost, quality, reliability, and availability;
  - risk factor prioritization;
  - extensive knowledge of end-users about predictive risk factors.
- All the above points essentially relate to prioritizing risk factors from different perspectives.
- Implementing staged blocks and other blockwise designs presents various challenges in practice. One of the biggest is properly handling the different samples used within the block model development.
- Given these challenges, modelers are strongly encouraged to consider the statistical consequences of using different samples and to collaborate closely with the business to better integrate process-related considerations.

Simulation Setup

- Assume a modeling dataset with 12 risk factors and binary target.
- Further, assume that these risk factors are classified into two groups based on the opinion of business experts.
- An additional business input modelers should incorporate into model development is that risk factors from the first group should be prioritized, with only the remaining variance modeled using risk factors from the second group.
- Finally, assume that the modelers chose Weights of Evidence (WoE) encoding for categorical risk factors.
- The dataset used for this simulation can be accessed at the following link.

Dataset Overview

```
## 'data.frame':    1000 obs. of  13 variables:
##  $ Creditability        : num  0 0 0 0 0 ...
##  $ Account_Balance      : chr  "1" "1" ...
##  $ Duration             : chr  "03 [16,36)" "02 [8,16)" ...
##  $ Payment_Status       : chr  "4" "4" ...
##  $ Purpose              : chr  "2" "0" ...
##  $ Credit_Amount        : chr  "01 (-Inf,3914)" "01 (-Inf,3914)" ...
##  $ Savings              : chr  "1" "1" ...
##  $ Employment           : chr  "2" "3" ...
##  $ Installment          : chr  "4" "2" ...
##  $ Gender_Marital_Status: chr  "2" "3" ...
##  $ Guarantors           : chr  "1" "1" ...
##  $ Available_Asset      : chr  "2" "1" ...
##  $ Age                  : chr  "01 (-Inf,26)" "03 [35,Inf)" ...
```

Blocks Design Overview

```
##                        rf block
## 1                Duration     1
## 2             Installment     1
## 3   Gender_Marital_Status     1
## 4              Guarantors     1
## 5                     Age     1
## 6         Account_Balance     2
## 7          Payment_Status     2
## 8                 Purpose     2
## 9           Credit_Amount     2
## 10                Savings     2
## 11             Employment     2
## 12        Available_Asset     2
```

Simulation Results - R Code

```
library(openxlsx)
library(dplyr)

#data import
fp <- "https://andrija-djurovic.github.io/adsfcr/bgrd/db_00_bgrd.xlsx"
db <- read.xlsx(xlsxFile = fp,
                sheet = 1)

#utils function import (woe.calc)
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/bgrd/utils.R")

#woe encoding
db.woe <- db
rf <- names(db)[-1]
for   (i in 1:length(rf)) {
      rf.i <- rf[i]
      woe.res.i <- woe.calc(db = db,
                            x = rf.i,
                            y = "Creditability")
      db.woe[, rf.i] <- woe.res.i[[2]]
      }
#sample of encoded dataset
head(db.woe[, c(1:3, 10:12)])
```

```
##   Creditability Account_Balance   Duration Gender_Marital_Status   Guarantors Available_Asset
## 1             0      -0.8180987 -0.1086883            -0.2353408 0.0005250722     -0.02857337
## 2             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
## 3             0      -0.4013918  0.3466246            -0.2353408 0.0005250722      0.46103496
## 4             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
## 5             0      -0.8180987  0.3466246             0.1655476 0.0005250722     -0.02857337
## 6             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
```

Simulation Results - R Code cont.

```r
#block 1 model
frm.b1 <- "Creditability ~ Duration + Installment + Gender_Marital_Status +
                           Guarantors + Age"
b1.r <- glm(formula = frm.b1,
            family = "binomial",
            data = db.woe)
round(x = summary(b1.r)$coefficients,
      digits = 4)
```

```
##                         Estimate Std. Error  z value Pr(>|z|)
## (Intercept)              -0.8560     0.0733 -11.6733   0.0000
## Duration                 -1.0292     0.1474  -6.9821   0.0000
## Installment              -1.1127     0.4566  -2.4369   0.0148
## Gender_Marital_Status    -1.1202     0.3503  -3.1977   0.0014
## Guarantors               -0.9451     0.4108  -2.3007   0.0214
## Age                      -0.9362     0.2301  -4.0684   0.0000
```

```r
#block 1 model predictions
b1.p <- unname(predict(object = b1.r))
head(b1.p)
```

```
## [1]  0.1890835 -1.8657478 -0.6275319 -1.7645730 -1.5177348 -1.9723970
```

Simulation Results - R Code cont.

```r
#block 2 model
frm.b2 <- "Creditability ~ Account_Balance + Payment_Status + Purpose + Credit_Amount +
                          Savings +  Employment + Available_Asset"
b2.r <- glm(formula = frm.b2,
            family = "binomial",
            data = db.woe,
            offset = b1.p)
round(x = summary(b2.r)$coefficients,
      digits = 4)
```

```
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -0.0102     0.0840 -0.1210   0.9037
## Account_Balance   -0.7931     0.1054 -7.5263   0.0000
## Payment_Status    -0.7184     0.1564 -4.5942   0.0000
## Purpose           -1.0230     0.2065 -4.9542   0.0000
## Credit_Amount     -0.5171     0.2323 -2.2258   0.0260
## Savings           -0.7689     0.1998 -3.8473   0.0001
## Employment        -0.5847     0.2784 -2.1001   0.0357
## Available_Asset   -0.5302     0.2580 -2.0550   0.0399
```

Simulation Results - Python Code

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
import requests

#data import
fp = "https://andrija-djurovic.github.io/adsfcr/bgrd/db_00_bgrd.xlsx"
db = pd.read_excel(io = fp,
                   sheet_name = 0,
                   dtype = "category")
db["Creditability"] = pd.to_numeric(db["Creditability"])
blocks = pd.read_excel(io = fp,
                       sheet_name = 1)

#utils function import (woe_calc)
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/bgrd/utils.py"
r = requests.get(url)
exec(r.text)

#woe encoding
db_woe = db.copy()
rf = db.columns[1:].tolist()
for i in range(len(rf)):
    rf_i = rf[i]
    woe_res_i = woe_calc(db = db,
                         x = rf_i,
                         y = "Creditability")
    db_woe[rf_i] = woe_res_i["x_trans"]
```

Simulation Results - Python Code cont.

```
#block 1 model
frm_b1 = "Creditability ~ Duration + Installment + Gender_Marital_Status + \
                           Guarantors + Age"
b1_r = smf.glm(formula = frm_b1,
               family = sm.families.Binomial(),
               data = db_woe).fit()
round(b1_r.summary2().tables[1], 4)
```

```
##                         Coef.  Std.Err.        z   P>|z|  [0.025  0.975]
## Intercept             -0.8560    0.0733 -11.6733  0.0000 -0.9997 -0.7123
## Duration              -1.0292    0.1474  -6.9821  0.0000 -1.3181 -0.7403
## Installment           -1.1127    0.4566  -2.4369  0.0148 -2.0076 -0.2178
## Gender_Marital_Status -1.1202    0.3503  -3.1977  0.0014 -1.8067 -0.4336
## Guarantors            -0.9451    0.4108  -2.3007  0.0214 -1.7503 -0.1400
## Age                   -0.9362    0.2301  -4.0684  0.0000 -1.3872 -0.4852
```

```
#block 1 model predictions
b1_p = b1_r.predict(which = "linear")
b1_p[0:5]
```

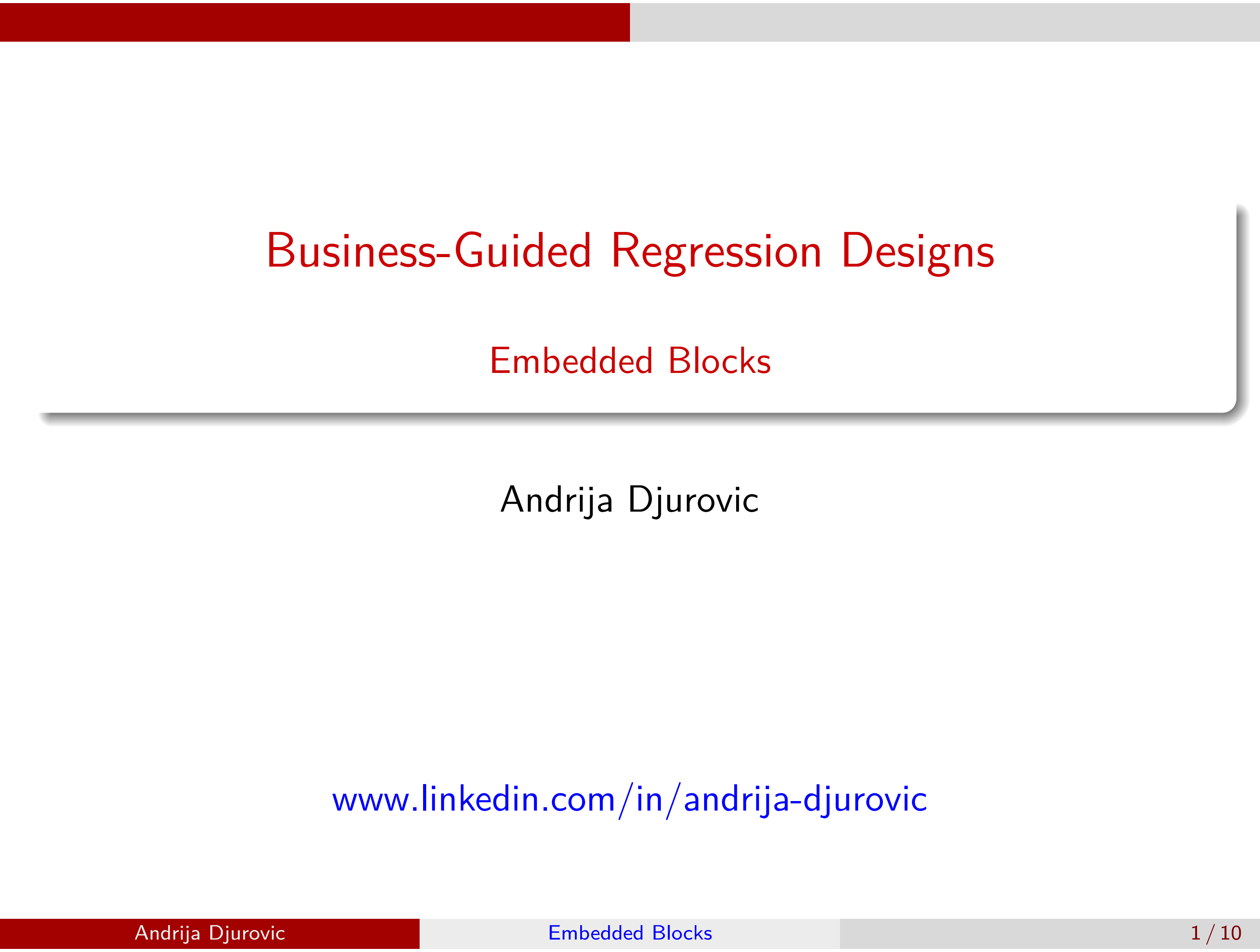
```
## array([ 0.18908348, -1.86574779, -0.62753191, -1.764573  , -1.51773477])
```

Simulation Results - Python Code cont.

```
#block 2 model
frm_b2 = "Creditability ~ Account_Balance + Payment_Status + Purpose + Credit_Amount + \
                          Savings + Employment + Available_Asset"
b2_r = smf.glm(formula = frm_b2,
               data = db_woe,
               family = sm.families.Binomial(),
               offset = b1_p).fit()
round(b2_r.summary2().tables[1], 4)
```

```
##                   Coef.  Std.Err.       z   P>|z|  [0.025  0.975]
## Intercept       -0.0102    0.0840 -0.1210  0.9037 -0.1747  0.1544
## Account_Balance -0.7931    0.1054 -7.5263  0.0000 -0.9997 -0.5866
## Payment_Status  -0.7184    0.1564 -4.5942  0.0000 -1.0248 -0.4119
## Purpose         -1.0230    0.2065 -4.9542  0.0000 -1.4277 -0.6183
## Credit_Amount   -0.5171    0.2323 -2.2258  0.0260 -0.9724 -0.0618
## Savings         -0.7689    0.1998 -3.8473  0.0001 -1.1605 -0.3772
## Employment      -0.5847    0.2784 -2.1001  0.0357 -1.1303 -0.0390
## Available_Asset -0.5302    0.2580 -2.0550  0.0399 -1.0358 -0.0245
```

8.2 Embedded Blocks



Business-Guided Regression Designs

Embedded Blocks

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Business-Guided Model Designs

- Developing credit risk models is not just a regulatory exercise; first and foremost, they should serve business needs.
- Recognizing this, businesses and modelers should collaborate closely when designing models.
- Often, model quality is not solely determined by strong statistical performance but by how well it supports business needs while maintaining sufficient statistical performance.
- Designing a model is not just about collaboration between businesses and modelers in selecting risk factors for the final model. It is usually a highly iterative process that incorporates multiple dimensions.
- A recent trend in credit risk model development shows that practitioners often opt for a blockwise or modular approach rather than developing a model using a fully consolidated dataset.
- Providing inputs for various modeling areas and ensuring robust statistical modeling are key objectives for businesses and modelers.
- Good practice shows that inputs from business experts plays a key role in model development. Modelers consider and incorporate these inputs into the overall model design.
- This presentation focuses on the embedded blocks approach, which is briefly described in the following slides. The rest of the presentation provides a simplified simulation study demonstrating the implementation of this design in `R` and `Python`.

Embedded Blocks Approach

- Practitioners usually prefer a blockwise (modular) approach over running a single optimization algorithm on a consolidated modeling dataset.
- This modular strategy employs multiple submodels or blocks based on business inputs, data coverage, and industry insights. These submodels are then combined to form a unified model score.
- One blockwise approach is the staged approach, where predictions from the previous block serve as offsets for models in subsequent blocks.
- The rationale for embedding is similar to other blockwise approaches and can be attributed to:
    - data sources, particularly their cost, quality, reliability, and availability;
    - risk factor prioritization;
    - extensive knowledge of end-users about predictive risk factors.
- All the above points essentially relate to prioritizing risk factors from different perspectives.
- Implementing embedded blocks and other blockwise designs presents various challenges in practice. One of the biggest is properly handling the different samples used within the block model development.
- Given these challenges, modelers are strongly encouraged to consider the statistical consequences of using different samples and to collaborate closely with the business to better integrate process-related considerations.

Simulation Setup

- Assume a modeling dataset with 12 risk factors and binary target.
- Further, assume that these risk factors are classified into two groups based on the opinion of business experts.
- An additional business input that modelers should incorporate into model development is prioritizing risk factors from the first group, making them a mandatory driver for the model run on the next block of risk factors.
- Finally, assume that the modelers chose Weights of Evidence (WoE) encoding for categorical risk factors.
- The dataset used for this simulation can be accessed at the following link.

Dataset Overview

```
## 'data.frame':    1000 obs. of  13 variables:
##  $ Creditability        : num  0 0 0 0 0 ...
##  $ Account_Balance      : chr  "1" "1" ...
##  $ Duration             : chr  "03 [16,36)" "02 [8,16)" ...
##  $ Payment_Status       : chr  "4" "4" ...
##  $ Purpose              : chr  "2" "0" ...
##  $ Credit_Amount        : chr  "01 (-Inf,3914)" "01 (-Inf,3914)" ...
##  $ Savings              : chr  "1" "1" ...
##  $ Employment           : chr  "2" "3" ...
##  $ Installment          : chr  "4" "2" ...
##  $ Gender_Marital_Status: chr  "2" "3" ...
##  $ Guarantors           : chr  "1" "1" ...
##  $ Available_Asset      : chr  "2" "1" ...
##  $ Age                  : chr  "01 (-Inf,26)" "03 [35,Inf)" ...
```

Blocks Design Overview

```
##                          rf block
## 1               Duration      1
## 2            Installment      1
## 3  Gender_Marital_Status      1
## 4             Guarantors      1
## 5                    Age      1
## 6        Account_Balance      2
## 7         Payment_Status      2
## 8                Purpose      2
## 9          Credit_Amount      2
## 10               Savings      2
## 11            Employment      2
## 12       Available_Asset      2
```

Simulation Results - R Code

```r
library(openxlsx)
library(dplyr)

#data import
fp <- "https://andrija-djurovic.github.io/adsfcr/bgrd/db_00_bgrd.xlsx"
db <- read.xlsx(xlsxFile = fp,
                sheet = 1)

#utils function import (woe.calc)
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/bgrd/utils.R")

#woe encoding
db.woe <- db
rf <- names(db)[-1]
for   (i in 1:length(rf)) {
      rf.i <- rf[i]
      woe.res.i <- woe.calc(db = db,
                            x = rf.i,
                            y = "Creditability")
      db.woe[, rf.i] <- woe.res.i[[2]]
      }
#sample of encoded dataset
head(db.woe[, c(1:3, 10:12)])
```

```
##   Creditability Account_Balance   Duration Gender_Marital_Status   Guarantors Available_Asset
## 1             0      -0.8180987 -0.1086883            -0.2353408 0.0005250722     -0.02857337
## 2             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
## 3             0      -0.4013918  0.3466246            -0.2353408 0.0005250722      0.46103496
## 4             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
## 5             0      -0.8180987  0.3466246             0.1655476 0.0005250722     -0.02857337
## 6             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
```

Simulation Results - R Code cont.

```r
#block 1 model
frm.b1 <- "Creditability ~ Duration + Installment + Gender_Marital_Status +
                           Guarantors + Age"
b1.r <- glm(formula = frm.b1,
            family = "binomial",
            data = db.woe)
round(x = summary(b1.r)$coefficients,
      digits = 4)
```

```
##                        Estimate Std. Error  z value Pr(>|z|)
## (Intercept)             -0.8560     0.0733 -11.6733   0.0000
## Duration                -1.0292     0.1474  -6.9821   0.0000
## Installment             -1.1127     0.4566  -2.4369   0.0148
## Gender_Marital_Status   -1.1202     0.3503  -3.1977   0.0014
## Guarantors              -0.9451     0.4108  -2.3007   0.0214
## Age                     -0.9362     0.2301  -4.0684   0.0000
```

```r
#block 1 model predictions
db.woe$block_1 <- unname(predict(object = b1.r))
head(db.woe$block_1)
```

```
## [1]  0.1890835 -1.8657478 -0.6275319 -1.7645730 -1.5177348 -1.9723970
```

Simulation Results - R Code cont.

```
#block 2 model
frm.b2 <- "Creditability ~ block_1 + Account_Balance + Payment_Status + Purpose +
                          Credit_Amount + Savings +  Employment + Available_Asset"
b2.r <- glm(formula = frm.b2,
            family = "binomial",
            data = db.woe)
round(x = summary(b2.r)$coefficients,
      digits = 4)
```

```
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -0.1409     0.1278 -1.1020   0.2705
## block_1            0.8308     0.1240  6.7002   0.0000
## Account_Balance   -0.7892     0.1043 -7.5657   0.0000
## Payment_Status    -0.7193     0.1544 -4.6598   0.0000
## Purpose           -0.9934     0.2047 -4.8532   0.0000
## Credit_Amount     -0.5773     0.2341 -2.4662   0.0137
## Savings           -0.7580     0.1979 -3.8303   0.0001
## Employment        -0.6073     0.2754 -2.2046   0.0275
## Available_Asset   -0.5371     0.2544 -2.1114   0.0347
```

Simulation Results - Python Code

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
import requests

#data import
fp = "https://andrija-djurovic.github.io/adsfcr/bgrd/db_00_bgrd.xlsx"
db = pd.read_excel(io = fp,
                   sheet_name = 0,
                   dtype = "category")
db["Creditability"] = pd.to_numeric(db["Creditability"])
blocks = pd.read_excel(io = fp,
                       sheet_name = 1)

#utils function import (woe_calc)
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/bgrd/utils.py"
r = requests.get(url)
exec(r.text)

#woe encoding
db_woe = db.copy()
rf = db.columns[1:].tolist()
for i in range(len(rf)):
    rf_i = rf[i]
    woe_res_i = woe_calc(db = db,
                         x = rf_i,
                         y = "Creditability")
    db_woe[rf_i] = woe_res_i["x_trans"]
```

Simulation Results - Python Code cont.

```
#block 1 model
frm_b1 = "Creditability ~ Duration + Installment + Gender_Marital_Status + \
                           Guarantors + Age"
b1_r = smf.glm(formula = frm_b1,
               family = sm.families.Binomial(),
               data = db_woe).fit()
round(b1_r.summary2().tables[1], 4)
```

```
##                          Coef.  Std.Err.        z   P>|z|   [0.025  0.975]
## Intercept              -0.8560    0.0733 -11.6733  0.0000 -0.9997 -0.7123
## Duration               -1.0292    0.1474  -6.9821  0.0000 -1.3181 -0.7403
## Installment            -1.1127    0.4566  -2.4369  0.0148 -2.0076 -0.2178
## Gender_Marital_Status -1.1202    0.3503  -3.1977  0.0014 -1.8067 -0.4336
## Guarantors             -0.9451    0.4108  -2.3007  0.0214 -1.7503 -0.1400
## Age                    -0.9362    0.2301  -4.0684  0.0000 -1.3872 -0.4852
```

```
#block 1 model predictions
db_woe["block_1"] = b1_r.predict(which = "linear")
db_woe["block_1"].iloc[0:5]
```

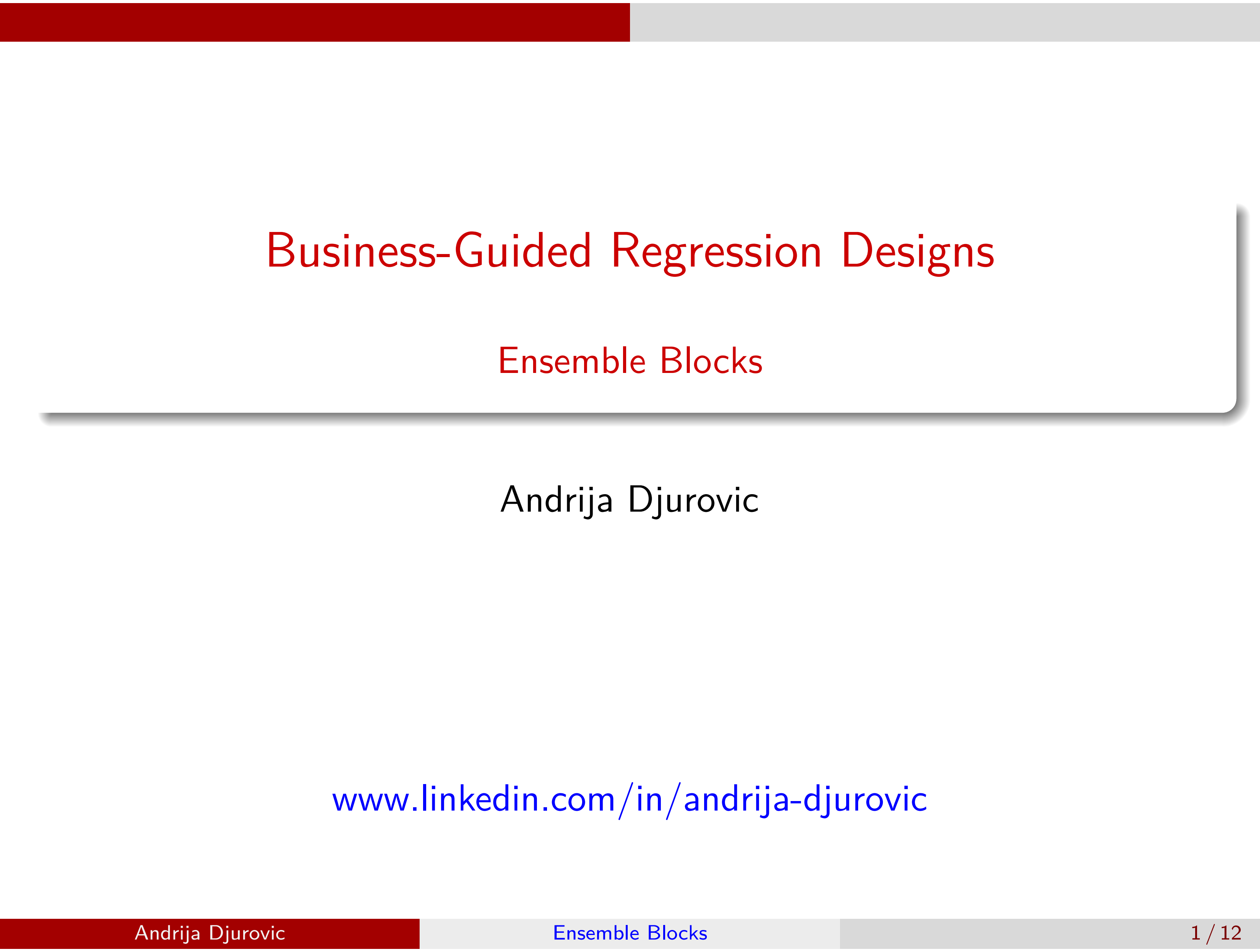
```
## 0    0.189083
## 1   -1.865748
## 2   -0.627532
## 3   -1.764573
## 4   -1.517735
## Name: block_1, dtype: float64
```

Simulation Results - Python Code cont.

```
#block 2 model
frm_b2 = "Creditability ~ block_1 + Account_Balance + Payment_Status + Purpose + \
                          Credit_Amount + Savings + Employment + Available_Asset"
b2_r = smf.glm(formula = frm_b2,
               data = db_woe,
               family = sm.families.Binomial()).fit()
round(b2_r.summary2().tables[1], 4)
```

```
##                   Coef.  Std.Err.       z   P>|z|  [0.025  0.975]
## Intercept       -0.1409    0.1278 -1.1020  0.2705 -0.3914  0.1097
## block_1          0.8308    0.1240  6.7002  0.0000  0.5877  1.0738
## Account_Balance -0.7892    0.1043 -7.5657  0.0000 -0.9936 -0.5847
## Payment_Status  -0.7193    0.1544 -4.6598  0.0000 -1.0219 -0.4168
## Purpose         -0.9934    0.2047 -4.8532  0.0000 -1.3946 -0.5922
## Credit_Amount   -0.5773    0.2341 -2.4662  0.0137 -1.0362 -0.1185
## Savings         -0.7580    0.1979 -3.8303  0.0001 -1.1459 -0.3702
## Employment      -0.6073    0.2754 -2.2046  0.0275 -1.1471 -0.0674
## Available_Asset -0.5371    0.2544 -2.1114  0.0347 -1.0356 -0.0385
```

8.3 Ensemble Blocks



Business-Guided Regression Designs

Ensemble Blocks

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Business-Guided Model Designs

- Developing credit risk models is not just a regulatory exercise; first and foremost, they should serve business needs.
- Recognizing this, businesses and modelers should collaborate closely when designing models.
- Often, model quality is not solely determined by strong statistical performance but by how well it supports business needs while maintaining sufficient statistical performance.
- Designing a model is not just about collaboration between businesses and modelers in selecting risk factors for the final model. It is usually a highly iterative process that incorporates multiple dimensions.
- A recent trend in credit risk model development shows that practitioners often opt for a blockwise or modular approach rather than developing a model using a fully consolidated dataset.
- Providing inputs for various modeling areas and ensuring robust statistical modeling are key objectives for businesses and modelers.
- Good practice shows that inputs from business experts plays a key role in model development. Modelers consider and incorporate these inputs into the overall model design.
- This presentation focuses on the ensemble blocks approach, which is briefly described in the following slides. The rest of the presentation provides a simplified simulation study demonstrating the implementation of this design in `R` and `Python`.

Ensemble Blocks Approach

- Practitioners usually prefer a blockwise (modular) approach over running a single optimization algorithm on a consolidated modeling dataset.

- This modular strategy employs multiple submodels or blocks based on business inputs, data coverage, and industry insights. These submodels are then combined to form a unified model score.

- One blockwise approach is the staged approach, where predictions from the previous block serve as offsets for models in subsequent blocks.

- The rationale for ensembling is similar to other blockwise approaches and can be attributed to:
  - data sources, particularly their cost, quality, reliability, and availability;
  - risk factor prioritization;
  - extensive knowledge of end-users about predictive risk factors.

- All the above points essentially relate to prioritizing risk factors from different perspectives and enabling decision-making based on results for a specific block.

- Implementing ensemble blocks and other blockwise designs presents various challenges in practice. One of the biggest is properly handling the different samples used within the block model development.

- Given these challenges, modelers are strongly encouraged to consider the statistical consequences of using different samples and to collaborate closely with the business to better integrate process-related considerations.

Simulation Setup

- Assume a modeling dataset with 12 risk factors and binary target.
- Further, assume that these risk factors are classified into two groups based on the opinion of business experts.
- An additional business input modelers should incorporate into model development is the preference for using separate block models for decision-making while obtaining a final integrated score.
- Finally, assume that the modelers chose Weights of Evidence (WoE) encoding for categorical risk factors.
- The dataset used for this simulation can be accessed at the following link.

Dataset Overview

```
## 'data.frame':    1000 obs. of  13 variables:
##  $ Creditability        : num  0 0 0 0 0 ...
##  $ Account_Balance      : chr  "1" "1" ...
##  $ Duration             : chr  "03 [16,36)" "02 [8,16)" ...
##  $ Payment_Status       : chr  "4" "4" ...
##  $ Purpose              : chr  "2" "0" ...
##  $ Credit_Amount        : chr  "01 (-Inf,3914)" "01 (-Inf,3914)" ...
##  $ Savings              : chr  "1" "1" ...
##  $ Employment           : chr  "2" "3" ...
##  $ Installment          : chr  "4" "2" ...
##  $ Gender_Marital_Status: chr  "2" "3" ...
##  $ Guarantors           : chr  "1" "1" ...
##  $ Available_Asset      : chr  "2" "1" ...
##  $ Age                  : chr  "01 (-Inf,26)" "03 [35,Inf)" ...
```

Blocks Design Overview

```
##                        rf block
## 1                Duration     1
## 2             Installment     1
## 3   Gender_Marital_Status     1
## 4              Guarantors     1
## 5                     Age     1
## 6         Account_Balance     2
## 7          Payment_Status     2
## 8                 Purpose     2
## 9           Credit_Amount     2
## 10                Savings     2
## 11             Employment     2
## 12        Available_Asset     2
```

Simulation Results - R Code

```r
library(openxlsx)
library(dplyr)

#data import
fp <- "https://andrija-djurovic.github.io/adsfcr/bgrd/db_00_bgrd.xlsx"
db <- read.xlsx(xlsxFile = fp,
                sheet = 1)

#utils function import (woe.calc)
source("https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/bgrd/utils.R")

#woe encoding
db.woe <- db
rf <- names(db)[-1]
for   (i in 1:length(rf)) {
      rf.i <- rf[i]
      woe.res.i <- woe.calc(db = db,
                            x = rf.i,
                            y = "Creditability")
      db.woe[, rf.i] <- woe.res.i[[2]]
      }
#sample of encoded dataset
head(db.woe[, c(1:3, 10:12)])
```

```
##   Creditability Account_Balance   Duration Gender_Marital_Status   Guarantors Available_Asset
## 1             0      -0.8180987 -0.1086883            -0.2353408 0.0005250722     -0.02857337
## 2             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
## 3             0      -0.4013918  0.3466246            -0.2353408 0.0005250722      0.46103496
## 4             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
## 5             0      -0.8180987  0.3466246             0.1655476 0.0005250722     -0.02857337
## 6             0      -0.8180987  0.3466246             0.1655476 0.0005250722      0.46103496
```

Simulation Results - R Code cont.

```
#block 1 model
frm.b1 <- "Creditability ~ Duration + Installment + Gender_Marital_Status +
                           Guarantors + Age"
b1.r <- glm(formula = frm.b1,
            family = "binomial",
            data = db.woe)
round(x = summary(b1.r)$coefficients,
      digits = 4)
```

```
##                        Estimate Std. Error  z value Pr(>|z|)
## (Intercept)             -0.8560     0.0733 -11.6733   0.0000
## Duration                -1.0292     0.1474  -6.9821   0.0000
## Installment             -1.1127     0.4566  -2.4369   0.0148
## Gender_Marital_Status  -1.1202     0.3503  -3.1977   0.0014
## Guarantors              -0.9451     0.4108  -2.3007   0.0214
## Age                     -0.9362     0.2301  -4.0684   0.0000
```

```
#block 1 model predictions
db.woe$block_1 <- unname(predict(object = b1.r))
head(db.woe$block_1)
```

```
## [1]  0.1890835 -1.8657478 -0.6275319 -1.7645730 -1.5177348 -1.9723970
```

Simulation Results - R Code cont.

```
#block 2 model
frm.b2 <- "Creditability ~ Account_Balance + Payment_Status + Purpose +
                           Credit_Amount + Savings +  Employment + Available_Asset"
b2.r <- glm(formula = frm.b2,
            family = "binomial",
            data = db.woe)
round(x = summary(b2.r)$coefficients,
      digits = 4)
```

```
##                   Estimate Std. Error  z value Pr(>|z|)
## (Intercept)        -0.8466     0.0808 -10.4802   0.0000
## Account_Balance   -0.8190     0.1020  -8.0319   0.0000
## Payment_Status    -0.7712     0.1487  -5.1845   0.0000
## Purpose           -0.8956     0.1954  -4.5826   0.0000
## Credit_Amount     -0.9264     0.2240  -4.1355   0.0000
## Savings           -0.7442     0.1926  -3.8647   0.0001
## Employment        -0.7465     0.2659  -2.8078   0.0050
## Available_Asset   -0.6515     0.2438  -2.6727   0.0075
```

```
#b2 model predictions
db.woe$block_2 <- unname(predict(object = b2.r))
head(db.woe$block_2)
```

```
## [1] -0.2893572 -0.6997119 -0.9388005 -0.6997119 -0.3807266 -0.3722147
```

Simulation Results - R Code cont.

```
#block 1 and block 2 integration
frm.b1b2 <- "Creditability ~ block_1 + block_2"
b1b2.r <- glm(formula = frm.b1b2,
              family = "binomial",
              data = db.woe)
round(x = summary(b1b2.r)$coefficients,
      digits = 4)
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.6171     0.1338  4.6115        0
## block_1       0.7812     0.1198  6.5225        0
## block_2       0.9365     0.0777 12.0590        0
```

Simulation Results - Python Code

```python
import numpy as np
import pandas as pd
import statsmodels.api as sm
import statsmodels.formula.api as smf
import requests

#data import
fp = "https://andrija-djurovic.github.io/adsfcr/bgrd/db_00_bgrd.xlsx"
db = pd.read_excel(io = fp,
                   sheet_name = 0,
                   dtype = "category")
db["Creditability"] = pd.to_numeric(db["Creditability"])
blocks = pd.read_excel(io = fp,
                       sheet_name = 1)

#utils function import (woe_calc)
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/refs/heads/main/bgrd/utils.py"
r = requests.get(url)
exec(r.text)

#woe encoding
db_woe = db.copy()
rf = db.columns[1:].tolist()
for i in range(len(rf)):
    rf_i = rf[i]
    woe_res_i = woe_calc(db = db,
                         x = rf_i,
                         y = "Creditability")
    db_woe[rf_i] = woe_res_i["x_trans"]
```

Simulation Results - Python Code cont.

```
#block 1 model
frm_b1 = "Creditability ~ Duration + Installment + Gender_Marital_Status + \
                           Guarantors + Age"
b1_r = smf.glm(formula = frm_b1,
               family = sm.families.Binomial(),
               data = db_woe).fit()
round(b1_r.summary2().tables[1], 4)
```

```
##                          Coef.  Std.Err.        z   P>|z|   [0.025  0.975]
## Intercept              -0.8560    0.0733 -11.6733  0.0000 -0.9997 -0.7123
## Duration               -1.0292    0.1474  -6.9821  0.0000 -1.3181 -0.7403
## Installment            -1.1127    0.4566  -2.4369  0.0148 -2.0076 -0.2178
## Gender_Marital_Status -1.1202    0.3503  -3.1977  0.0014 -1.8067 -0.4336
## Guarantors             -0.9451    0.4108  -2.3007  0.0214 -1.7503 -0.1400
## Age                    -0.9362    0.2301  -4.0684  0.0000 -1.3872 -0.4852
```

```
#block 1 model predictions
db_woe["block_1"] = b1_r.predict(which = "linear")
db_woe["block_1"].iloc[0:5]
```

```
## 0    0.189083
## 1   -1.865748
## 2   -0.627532
## 3   -1.764573
## 4   -1.517735
## Name: block_1, dtype: float64
```

Simulation Results - Python Code cont.

```
#block 2 model
frm_b2 = "Creditability ~ Account_Balance + Payment_Status + Purpose + \
                          Credit_Amount + Savings + Employment + Available_Asset"
b2_r = smf.glm(formula = frm_b2,
               data = db_woe,
               family = sm.families.Binomial()).fit()
round(b2_r.summary2().tables[1], 4)
```

```
##                   Coef.  Std.Err.        z   P>|z|  [0.025  0.975]
## Intercept       -0.8466    0.0808 -10.4802  0.0000 -1.0049 -0.6882
## Account_Balance -0.8190    0.1020  -8.0319  0.0000 -1.0189 -0.6192
## Payment_Status  -0.7712    0.1487  -5.1845  0.0000 -1.0627 -0.4796
## Purpose         -0.8956    0.1954  -4.5826  0.0000 -1.2787 -0.5126
## Credit_Amount   -0.9264    0.2240  -4.1355  0.0000 -1.3654 -0.4873
## Savings         -0.7442    0.1926  -3.8647  0.0001 -1.1216 -0.3668
## Employment      -0.7465    0.2659  -2.8078  0.0050 -1.2676 -0.2254
## Available_Asset -0.6515    0.2438  -2.6727  0.0075 -1.1293 -0.1737
```

```
#b2model predictions
db_woe["block_2"] = b2_r.predict(which = "linear")
db_woe["block_2"].iloc[0:5]
```

```
## 0   -0.289357
## 1   -0.699712
## 2   -0.938801
## 3   -0.699712
## 4   -0.380727
## Name: block_2, dtype: float64
```

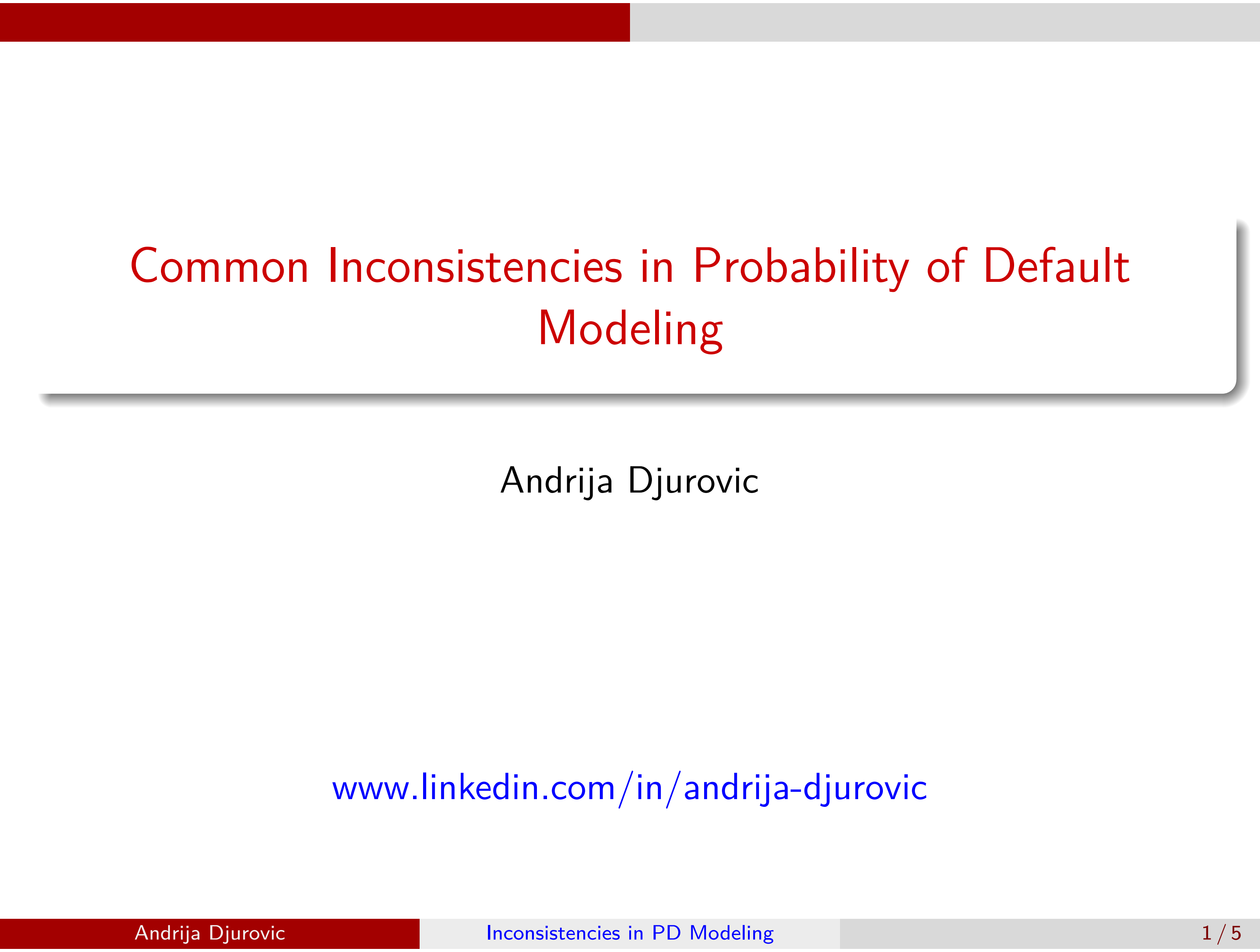
Simulation Results - Python Code cont.

```
#block 1 and block 2 integration
frm_b1b2 = "Creditability ~ block_1 + block_2"
b1b2_r = smf.glm(formula = frm_b1b2,
                 data = db_woe,
                 family = sm.families.Binomial()).fit()
round(b1b2_r.summary2().tables[1], 4)
```

```
##              Coef.  Std.Err.        z  P>|z|  [0.025  0.975]
## Intercept  0.6171    0.1338   4.6115    0.0  0.3548  0.8794
## block_1    0.7812    0.1198   6.5225    0.0  0.5464  1.0159
## block_2    0.9365    0.0777  12.0590    0.0  0.7843  1.0888
```

9 Model Development and Validation

9.1 Common Inconsistencies in PD Modeling



Common Inconsistencies in Probability of Default Modeling

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

PD Modeling

- Most statistical approaches used in Probability of Default (PD) model development, validation, and application rely on specific assumptions.
- In addition to the inherent statistical assumptions of particular procedures, practitioners sometimes introduce additional assumptions they believe are necessary for the overall modeling process.
- When implementing different statistical procedures, it is crucial to understand the appropriateness of these additional assumptions and their potential consequences.
- Practitioners are often inconsistent in applying and relying on various assumptions during model development, validation (both initial and periodic), and application.
- Considering potential inconsistencies, several questions arise:
  - Is it sufficient to argue that certain inconsistencies are widely accepted concepts or standard practices?
  - What are the consequences of these inconsistencies on the final procedure results?
  - How do these inconsistencies influence the discussions between model developers and validators (internal, external, or regulators)?
- The following slides highlight some typical inconsistencies observed in PD modeling.

Example 1: Assumptions on Uncorrelated and Correlated Defaults

Assumptions of Uncorrelated Defaults

When testing the predictive power of the PD model, a commonly used method is the z-score test, defined by the following formula:

$$z = \frac{ODR - PD}{\sqrt{\frac{PD(1-PD)}{n}}}$$

One of the main assumptions of this test is that defaults are uncorrelated. For further details, refer to this document.

Assumptions of Correlated defaults

When practitioners use the IRB PD model to calculate RWA, for instance, in the context of retail residential mortgage exposures, the following formulas are employed:

$$Correlation = R = 0.15$$

$$\text{Capital requirement} = K = LGD \cdot N\left[\frac{G(PD)}{\sqrt{1-R}} + \sqrt{\frac{R}{1-R}} \cdot G(0.999)\right] - PD \cdot \text{LGD}$$

$$RWA = K \cdot 12.5 \cdot EAD$$

In the calculation above for the upper bound of the PD, it is assumed that the PD follows the Vasicek distribution, where one of the parameters is interpreted as the asset correlation.

Example 2: Assumptions of the Deterministic Metric from the Development Sample

Standard Error Calculated from Development Sample

The following formula gives the most common form of the z-score test for assessing predictive power:

$$z = \frac{ODR - PD}{\sqrt{\frac{PD(1-PD)}{n}}}$$

This formula assumes that the standard error of the test statistic is calculated using the calibrated PD obtained from the model development phase.

Deterministic Metric from the Development Sample

When assessing changes in the concentration of a rating scale, a commonly used approach is to examine the change in the coefficient of variation (CV), defined by the following formula:

$$1 - \Phi\left(\frac{\sqrt{K-1}(CV_{curr} - CV_{init})}{\sqrt{CV_{curr}^2\,(0.5 + CV_{curr}^2)}}\right)$$

This formula assumes a normal approximation and that the CV is deterministic based on the model's development phase. For further details, refer to this document.

The same assumption is often applied when testing the discriminatory power of credit risk models.

Example 3: Assumptions of Independent Observations

Autocorrelation in PD FLI IFRS9 Modeling

Ordinary Least Squares (OLS) regression is one of the most commonly used methods for forward-looking Probability of Default (PD) modeling in IFRS9. The standard form of the OLS regression is given by:

$$\hat{y}_t = \beta_0 + \beta_1 x_{1t} + \beta_2 x_{2t} + \cdots + \beta_p x_{pt} + \epsilon_t$$

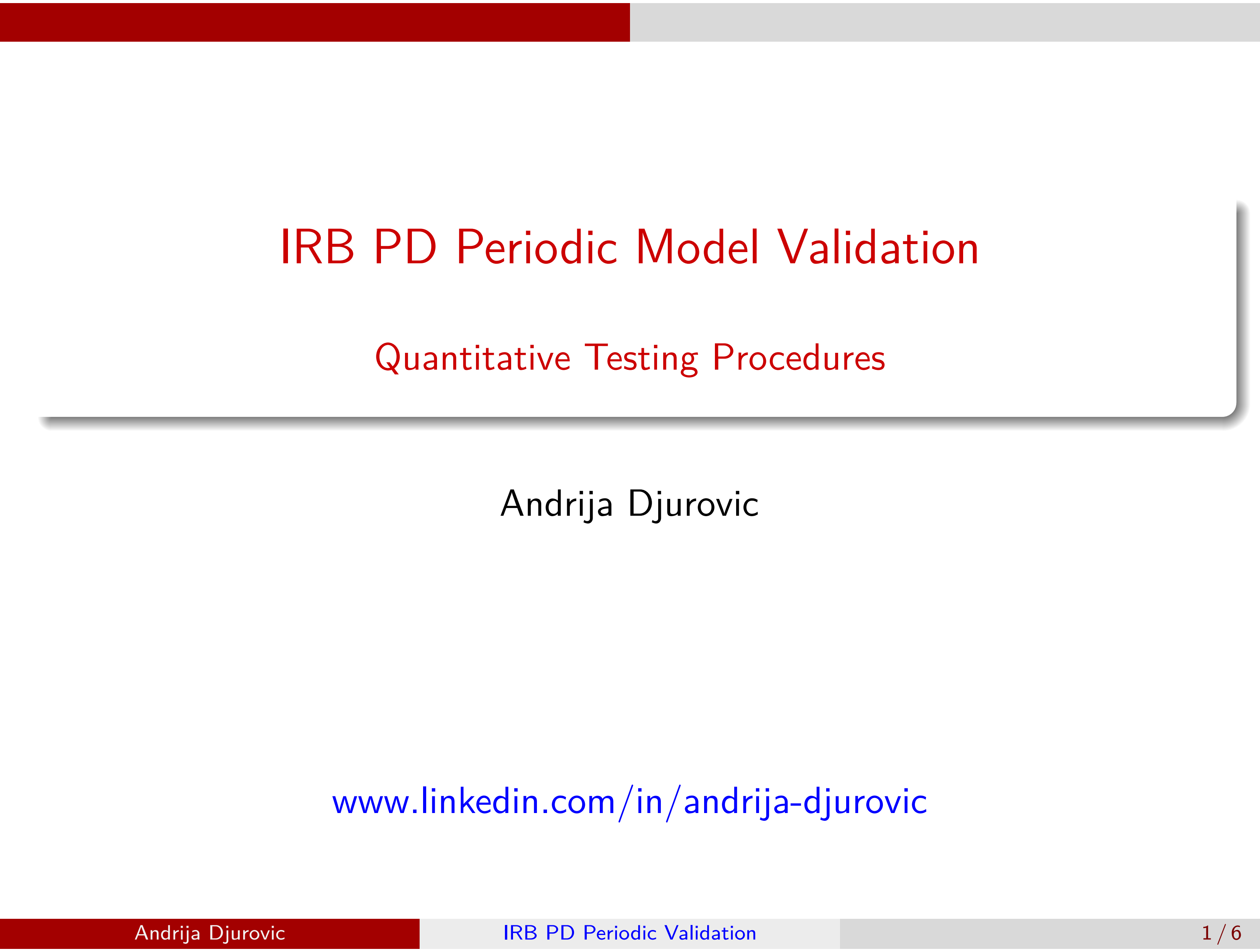
When using this method, practitioners address the problem of autocorrelation in the residuals (ϵ_t), acknowledging its potential consequences.

Independence in Quantifying the Uncertainty of the Central Tendency

Various methods can be used to manage the variability associated with calculating the Central Tendency (CT) of default rates. The most frequently used method relies on the standard error of the mean, defined as:

$$\frac{\sigma}{\sqrt{n}}$$

When applying this method, practitioners often assume by default that the observed default rates are independent. For further details, refer to this document.



IRB PD Periodic Model Validation

Quantitative Testing Procedures

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Periodic Model Validation

- Periodic model validation is a critical part of the Probability of Default (PD) model lifecycle.
- It typically refers to a robust system consisting of clearly defined tests and processes that assess whether the model under investigation is still suitable for its intended purpose.
- A common practice is to split the validation process into quantitative and qualitative procedures.
- The following slides provide a brief overview of specific areas related to quantitative validation procedures. Although different approaches exist in practice, one approach is to group procedures into three main areas:
  - model structure;
  - model calibration (sometimes referred to as backtesting or review of estimates);
  - Margin of Conservatism (MoC) challengers.

PD Model Structure

The tests and metrics used in this area of investigation aim to assess whether the model sufficiently differentiates risk between observations in the validation sample. In general, various aspects of the model are validated, such as:

- changes in discriminatory power over time;
- stability of risk factors and model output;
- representativeness;
- heterogeneity of the rating scale;
- homogeneity of the rating scale;
- concentration in the rating grades and risk factor modalities.

Some of the most commonly applied statistical metrics and tests for this purpose include:

- Area Under the Receiver Operating Characteristics Curve (AUC);
- test of proportions;
- Population Stability Index (PSI);
- Herfindahl-Hirschman Index (HHI).

Model Calibration

The model calibration is closely related to risk quantification process. It assesses the predictive power of the model in use. The predictive power analysis aims to ensure that the PD parameter accurately predicts the occurrence of defaults, meaning that PD estimates provide reliable forecasts of default rates.

In practice, the four statistical tests are commonly used, and they are:

- exact binomial test;
- Jeffreys' test;
- z-score test;
- Hosmer-Lemeshow test.

Only the Hosmer-Lemeshow test is designed for the overall rating scale, while others are usually applied on the level of the rating grade and portfolio level.

Margin of Conservatism Challengers

Due to the variety of possible sources of model uncertainty, challengers of the margin of conservatism typically focus on uncertainty arising from general estimation error. These challenges are usually based on the variability of long-run default rates (at the portfolio or rating grade level) and primarily rely on the confidence interval of binomial proportions but may also incorporate other methods.

Some commonly applied statistical approaches include:

- empirical quantiles;
- Clopper-Pearson confidence interval;
- Jeffreys' confidence interval;
- normal-based approximated confidence interval;
- Pluto-Tasche methods;
- Benjamin-Cathcart-Ryan approach;
- Alan Forrest's approach.

Concluding Remarks

- The primary purpose of periodic model validation is to determine whether the model under investigation remains suitable for its intended purpose by examining various aspects of the model.
- Periodic model validation involves clearly defined qualitative and quantitative testing procedures.
- Properly defined testing hypotheses are a strong foundation for effective validation. Practitioners should ensure that these hypotheses are correctly formulated and support the validation process.
- Each statistical test used in the quantitative analysis relies on specific assumptions that practitioners should be aware of when implementing and interpreting the results.
- In addition to considering the results based on the p-value of statistical tests, practitioners should also consider measures of practical significance. More details on p-values and statistical tests are available here.
- Even favorable p-values should not be accepted blindly without further investigation. More details on favorable p-values are available here.

9.3 IRB Model Validation - Example of an Automated Validation Report

IRB Model Validation

Example of an Automated Validation Report

Andrija Djurovic

Contents

1 Introduction

The validation of credit risk models is typically divided into qualitative and quantitative procedures, with the latter often subject to automation. This report demonstrates the use of `R` and `Python` to automate quantitative procedures. While it does not provide a detailed overview of these procedures and tests, it focuses on the technical aspects of integrating `R` and `Python` for validation. The approach presented reflects the author's preferences regarding technical methods, but practitioners are encouraged to explore and adapt it to best suit their specific needs. The following sections are generated entirely in `R`, with specific procedures executed and imported from `Python` using the `reticulate` package. The PDF format is also generated with the `rmarkdown` package, while tables are formatted using the `flextable` package. Graph examples are created with the `ggplot2` package.

Practitioners can find details on specific tests and approaches used in IRB model validation on the following GitHub page.

The rest of the report is organized as follows: The Report Configuration section briefly outlines the `R` session used to generate the report and the `Python` setup for running scripts from `R`. Subsequent sections provide technical details on the most common methods used in validation reports, typically covering data import and processing, descriptive statistics, data visualization, and quantitative testing. Finally, the report raises awareness of alternative approaches and possible improvements for real-world case scenarios.

2 Report Configuration

To run this report, specific packages must be installed for `R` and `Python`. In `R`, these include `rmarkdown`, `knitr`, `flextable`, `ggplot2`, and `reticulate`. The following output provides details of the R session used to generate this report:

```
sessionInfo()
```

```
## R version 4.3.2 (2023-10-31 ucrt)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 11 x64 (build 22631)
##
## Matrix products: default
##
##
## locale:
## [1] LC_COLLATE=English_United States.utf8
## [2] LC_CTYPE=English_United States.utf8
## [3] LC_MONETARY=English_United States.utf8
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.utf8
##
## time zone: Europe/Budapest
## tzcode source: internal
##
```

```
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] flextable_0.9.5   ggplot2_3.4.4     reticulate_1.34.0 knitr_1.45
## [5] rmarkdown_2.26
##
## loaded via a namespace (and not attached):
##  [1] utf8_1.2.4              generics_0.1.3          fontLiberation_0.1.0
##  [4] xml2_1.3.6              lattice_0.21-9          httpcode_0.3.0
##  [7] digest_0.6.33           magrittr_2.0.3          evaluate_0.23
## [10] grid_4.3.2              fastmap_1.1.1           jsonlite_1.8.8
## [13] Matrix_1.6-1.1          zip_2.3.0               crul_1.4.0
## [16] tinytex_0.49            promises_1.2.1          fansi_1.0.6
## [19] scales_1.3.0            fontBitstreamVera_0.1.1 textshaping_0.3.7
## [22] cli_3.6.1               shiny_1.8.0             rlang_1.1.2
## [25] fontquiver_0.2.1        crayon_1.5.2            ellipsis_0.3.2
## [28] munsell_0.5.0           yaml_2.3.7              withr_2.5.2
## [31] gfonts_0.2.0            gdtools_0.3.7           officer_0.6.5
## [34] tools_4.3.2             uuid_1.2-0              dplyr_1.1.4
## [37] colorspace_2.1-0        httpuv_1.6.13           curl_5.2.0
## [40] vctrs_0.6.5             R6_2.5.1                mime_0.12
## [43] png_0.1-8               lifecycle_1.0.4         ragg_1.2.7
## [46] pkgconfig_2.0.3         pillar_1.9.0            later_1.3.2
## [49] gtable_0.3.4            data.table_1.14.10      glue_1.6.2
## [52] Rcpp_1.0.11             systemfonts_1.0.5       highr_0.10
## [55] xfun_0.41               tibble_3.2.1            tidyselect_1.2.0
## [58] farver_2.1.1            xtable_1.8-4            htmltools_0.5.7
## [61] labeling_0.4.3          compiler_4.3.2          askpass_1.2.0
## [64] openssl_2.1.1
```

The `reticulate` package configures the environment to run `Python` from `R`. First, a virtual environment is created and initiated using the following code:

```
#create virtual environment
reticulate::virtualenv_create(envname = my_envname,
                              python = path_to_the_python_exe_file)
#initiate virtual environment
reticulate::use_virtualenv(virtualenv = my_envname)
```

After successfully setting up the environment, the `pandas` and `scipy` packages are installed using the following code:

```
reticulate::virtualenv_install(envname = my_envname,
                               packages = c("pandas", "scipy"))
```

To check the versions of installed `Python` packages, practitioners can run the following command:

```
import importlib.metadata
```

```
#list of packages to check
packages = ["pandas", "scipy", "numpy"]

#print installed versions
for pkg in packages:
    try:
        version = importlib.metadata.version(pkg)
        print(f"{pkg}: {version}")
    except importlib.metadata.PackageNotFoundError:
        print(f"{pkg} is not installed.")
```

```
## pandas: 2.2.3
## scipy: 1.11.4
## numpy: 1.26.2
```

3 Data Import

The first step in any data analysis is usually data import and processing. In credit risk model validation, practitioners typically access various datasets stored in centralized systems or imported from different sources. The most common approach combines data from specific risk marts (often stored on different SQL servers) with other IT systems/applications that collect relevant data.

For simplicity, this section focuses on creating data directly within the session and demonstrating how data can be exchanged between `Python` and `R`.
Let's start by generating a dummy data frame in `R`.

```
#rating scale data frame
rating <- paste0("R0", 1:7)
no <- c(170, 118, 274, 100, 91, 196, 51)
nb <- c(3, 10, 47, 31, 43, 122, 44)
rs <- data.frame(rating, no, nb, odr = nb/no)
```

The created data frame can be printed directly in the report or formatted using a package such as `flextable`. The following code prints the data frame as is and then formats it using the `flextable` function from the same package.
Print `rs` data frame:

```
rs
```

```
##   rating  no  nb        odr
## 1    R01 170   3 0.01764706
## 2    R02 118  10 0.08474576
## 3    R03 274  47 0.17153285
## 4    R04 100  31 0.31000000
## 5    R05  91  43 0.47252747
## 6    R06 196 122 0.62244898
## 7    R07  51  44 0.86274510
```

Print `rs` as `flextable`:

```
flextable(data = rs)
```

rating	no	nb	odr
R01	170	3	0.01764706
R02	118	10	0.08474576
R03	274	47	0.17153285
R04	100	31	0.31000000
R05	91	43	0.47252747
R06	196	122	0.62244898
R07	51	44	0.86274510

Once the `rs` data frame is created in `R`, it can be accessed and processed from `Python`. Let's confirm that `rs` is accessable and print it from `Python` session.

```
import pandas as pd

#access rs by running r.rs
rs_py = r.rs
#print rs_py
rs_py
```

```
##   rating     no     nb       odr
## 0    R01  170.0    3.0  0.017647
## 1    R02  118.0   10.0  0.084746
## 2    R03  274.0   47.0  0.171533
## 3    R04  100.0   31.0  0.310000
## 4    R05   91.0   43.0  0.472527
## 5    R06  196.0  122.0  0.622449
## 6    R07   51.0   44.0  0.862745
```

Let's add a column to `rs_py` in `Python`:

```
rs_py["segment"] = "Retail"
```

Now print the data frame as a `flextable` in `R`:

```
flextable(data = py$rs_py)
```

rating	no	nb	odr	segment
R01	170	3	0.01764706	Retail
R02	118	10	0.08474576	Retail
R03	274	47	0.17153285	Retail
R04	100	31	0.31000000	Retail

rating	no	nb	odr	segment
R05	91	43	0.47252747	Retail
R06	196	122	0.62244898	Retail
R07	51	44	0.86274510	Retail

Besides the demonstrated ability to pass objects between `R` and `Python`, `reticulate` also provides an option to run `Python` scripts and functions within `R`. This is especially useful when part of the data import and processing is prepared in `Python`. The following code demonstrates the execution of a `Python` command within `R`:

```
#run python command
py_run_string("x = 10")
#print object x in r
py$x
```

```
## [1] 10
```

To demonstrate how practitioners can execute or source a `Python` script from `R`, let's first write a `Python` file containing a function and a list.
`Python` code:

```
#define file name
file_name = "python_script.py"
#define file content
content = """def hello_world():
   print(\"Hello, World!\")
import numpy as np
np_list = np.array([1, 2, 3, 4, 5])
"""
#write file
with open(file_name, "w") as file:
   _ = file.write(content)
```

Once the `Python` script is ready, we can source or execute it within `R` as follows.
`R` script to source a `Python` script:

```
#source python script
source_python("python_script.py")
#print np_list object
py$np_list
```

```
## [1] 1 2 3 4 5
```

```
#run python function
py_run_string("hello_world()")
```

```
## Hello, World!
```

`R` script to execute a `Python` script:

```
#execute python script
py_run_file("python_script.py")
#print np_list object
py$np_list
```

```
## [1] 1 2 3 4 5
```

```
#run python function
py_run_string("hello_world()")
```

```
## Hello, World!
```

4 Descriptive Statistics and Summary Tables

After successfully importing the data, practitioners typically summarize specific metrics and data subject to analysis and validation. This step often involves printing and presenting tables in different formats, emphasizing specific figures or points that require attention.
The following examples demonstrate how to print and format tables primarily using the `flextable` package. For this purpose, we will use the previously defined data frame **`rs`** from the last section.
`R` script:

```
#print rs
rs
```

```
##   rating  no  nb        odr
## 1    R01 170   3 0.01764706
## 2    R02 118  10 0.08474576
## 3    R03 274  47 0.17153285
## 4    R04 100  31 0.31000000
## 5    R05  91  43 0.47252747
## 6    R06 196 122 0.62244898
## 7    R07  51  44 0.86274510
```

```
#print rs as flextable
flextable(data = rs)
```

rating	no	nb	odr
R01	170	3	0.01764706
R02	118	10	0.08474576
R03	274	47	0.17153285
R04	100	31	0.31000000
R05	91	43	0.47252747
R06	196	122	0.62244898
R07	51	44	0.86274510

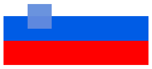
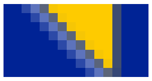
```
#format odr column to 2 decimals
rs.f <- flextable(rs) |>
        colformat_double(j = 4,
                         digits = 2)
rs.f
```

rating	no	nb	odr
R01	170	3	0.02
R02	118	10	0.08
R03	274	47	0.17
R04	100	31	0.31
R05	91	43	0.47
R06	196	122	0.62
R07	51	44	0.86

Sometimes data aggregation consolidates figures from the different countries. The `flextable` package provides easy to use ways of incorporating images to the table. The following code showcase how country flags can be added to the table.

```
#random seed
set.seed(1234)
#simulate exposure values
Average_Exposure <- runif(n = 6,
                          min = 100,
                          max = 1000)
Average_Exposure <- round(x = Average_Exposure)
#create country level data
cd <- data.frame(Country = "",
                 Average_Exposure = Average_Exposure)
#path to the country flags
flags <- c("me.png",
           "mk.png",
           "si.png",
           "rs.png",
           "ba.png",
           "hr.png")
#flextable
cd.f <- flextable(cd) |>
        compose(j = 1,
                value = as_paragraph(as_image(src = flags,
                                              width = 0.75,
                                              height = 0.75))) |>
        autofit()
```

```
#print flextable
cd.f
```

Country	Average_Exposure
	202
	660
	648
	661
	875
	676

5 Data Visualization

Data visualization is an essential step in any data analysis, including the validation of IRB models. The way practitioners visualize data largely depends on the type of data and the context of validation. Both `R` and `Python` have well-developed graphical engines.

The following examples demonstrate two commonly used methods for presenting data. The first method involves inline plots, where graphs are embedded directly into tables to provide additional insights for practitioners. The second method uses standard visualizations, such as histograms and line charts, to display data separately in a more traditional format.

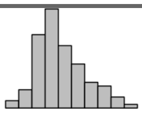
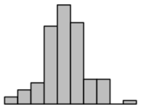
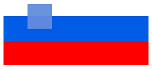
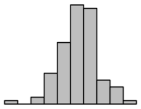
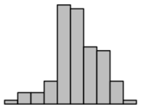
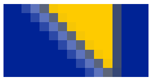
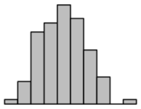
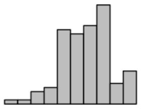
Let's start with inline plots embedded in `flextable`.
`R` script for inline plots:

```
#dummy data frame at the country level
cd <- data.frame(Country = rep(x = "", times = 6),
                 Distribution = "")
#simulated data per country
sim.data <- lapply(X = 1:nrow(cd),
                   FUN = function(x) rnorm(n = 100))
#flags to be added to the table
flags <- c("me.png",
           "mk.png",
           "si.png",
           "rs.png",
           "ba.png",
           "hr.png")
```

```r
#function to generate histograms as images
generate.hist <- function(data, filename) {
      p <- ggplot(data.frame(x = data), aes(x)) +
           geom_histogram(bins = 10, fill = "gray", color = "black") +
           theme_void()
ggsave(filename, plot = p, width = 2, height = 1.5, dpi = 100)
}

#generate histogram images
hist.files <- paste0("hist_", 1:length(sim.data), ".png")
mapply(generate.hist, sim.data, hist.files)
```

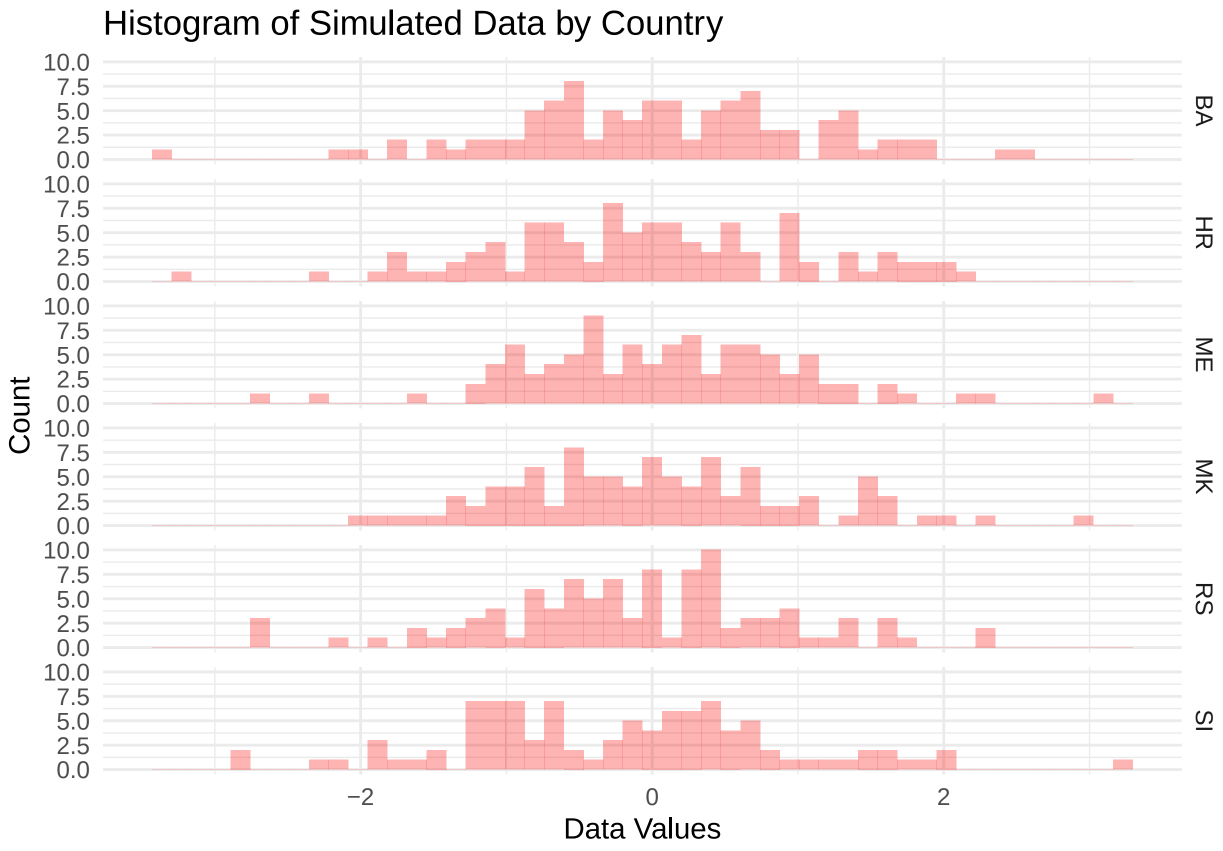
[1] “hist_1.png” “hist_2.png” “hist_3.png” “hist_4.png” “hist_5.png” [6] “hist_6.png”

```r
#prepare flextable
ft <- flextable(cd) |>
      compose(j = 1, value = as_paragraph(as_image(src = flags,
                                                   width = 0.75,
                                                   height = 0.75))) |>
      compose(j = 2, value = as_paragraph(as_image(src = hist.files,
                                                   width = 0.75,
                                                   height = 0.75))) |>
      autofit()
#print flextable
ft
```

Country	Distribution
	
	
	
	
	
	

R script for ggplot2 graph:

```
library(ggplot2)

#iso country code
iso <- c("ME", "MK", "SI", "RS", "BA", "HR")
#convert the named list into a dataframe
sim.data <- unname(c(sim.data, recursive = TRUE))
db <- data.frame(Country = rep(x = iso,
                               time = 100),
                 Data = sim.data)

#distribution of Data variable by country
gg <- ggplot(db, aes(x = Data)) +
      geom_histogram(alpha = 0.30, bins = 50, fill = "red") +
      facet_grid(Country ~ .) +
      theme_minimal() +
      labs(title = "Histogram of Simulated Data by Country",
           x = "Data Values",
           y = "Count")

gg
```



6 Quantitative Testing

One of the most critical aspects of IRB model validation is quantitative testing. For this task, practitioners typically perform a predefined set of statistical tests to examine and validate the model from different perspectives. Key aspects assessed in quantitative testing include model structure, model calibration, and the Margin of Conservatism. More details on specific tests and approaches for Probability of Default (PD) modeling can be found here.

Since this exercise aims to showcase aspects of report automation, the following code snippets will demonstrate how testing procedures can be executed, formatted, and presented within tables.

Let's start with the exact binomial test, commonly used for testing PD models.

R script:

```r
#set seed
set.seed(2211)
#add calibrated pds to the rating scale rs
rs$pd <- rs$odr - runif(n = nrow(rs), min = -0.10, max = 0.10)
#print rs
rs
```

```
##   rating  no  nb        odr           pd
## 1    R01 170   3 0.01764706 0.0004717041
## 2    R02 118  10 0.08474576 0.1356595537
## 3    R03 274  47 0.17153285 0.2334632658
## 4    R04 100  31 0.31000000 0.2276874342
## 5    R05  91  43 0.47252747 0.3840618209
## 6    R06 196 122 0.62244898 0.6803554137
## 7    R07  51  44 0.86274510 0.8417771827
```

```r
#run exact binomial test
rs$binom.test <- pbinom(q = rs$nb - 1,
                        size = rs$no,
                        prob = rs$pd,
                        lower.tail = FALSE)
#print the p-values
rs$binom.test
```

```
## [1] 0.00007959578 0.96682999912 0.99495191372 0.03590130777 0.05299616355
## [6] 0.96379413142 0.43041998679
```

Once the test results are available, practitioners typically compare the observed p-value to a predefined threshold. Let's assume the threshold is set at 5% for this exercise. The following code will compare the observed p-value against this threshold and format the results accordingly.

```r
#define threshold
threshold <- 0.05
#define color
col <- ifelse(rs$binom.test < threshold, "red", "green")
#prepare the table with results
rs.ft <- flextable(rs) |>
```

```
          color(j = 6,
                color = col) |>
          colformat_double(j = 6,
                           digits = 2) |>
          autofit()
#print the results
rs.ft
```

rating	no	nb	odr	pd	binom.test
R01	170	3	0.01764706	0.0004717041	0.00
R02	118	10	0.08474576	0.1356595537	0.97
R03	274	47	0.17153285	0.2334632658	0.99
R04	100	31	0.31000000	0.2276874342	0.04
R05	91	43	0.47252747	0.3840618209	0.05
R06	196	122	0.62244898	0.6803554137	0.96
R07	51	44	0.86274510	0.8417771827	0.43

Similarly to the examples from the previous sections, quantitative procedures can be executed in Python and printed and formatted in R. The following code demonstrates the execution of the same statistical test in Python.

```
from scipy.stats import binom

#access r object
rs_py = r.rs

#run exact binomial test
rs_py["binom_test"] = 1 - binom.cdf(rs_py["nb"] - 1,
                                    rs_py["no"],
                                    rs_py["pd"])
#print rs_py
rs_py
```

```
##   rating     no     nb       odr        pd  binom.test  binom_test
## 0    R01  170.0    3.0  0.017647  0.000472    0.000080    0.000080
## 1    R02  118.0   10.0  0.084746  0.135660    0.966830    0.966830
## 2    R03  274.0   47.0  0.171533  0.233463    0.994952    0.994952
## 3    R04  100.0   31.0  0.310000  0.227687    0.035901    0.035901
## 4    R05   91.0   43.0  0.472527  0.384062    0.052996    0.052996
## 5    R06  196.0  122.0  0.622449  0.680355    0.963794    0.963794
## 6    R07   51.0   44.0  0.862745  0.841777    0.430420    0.430420
```

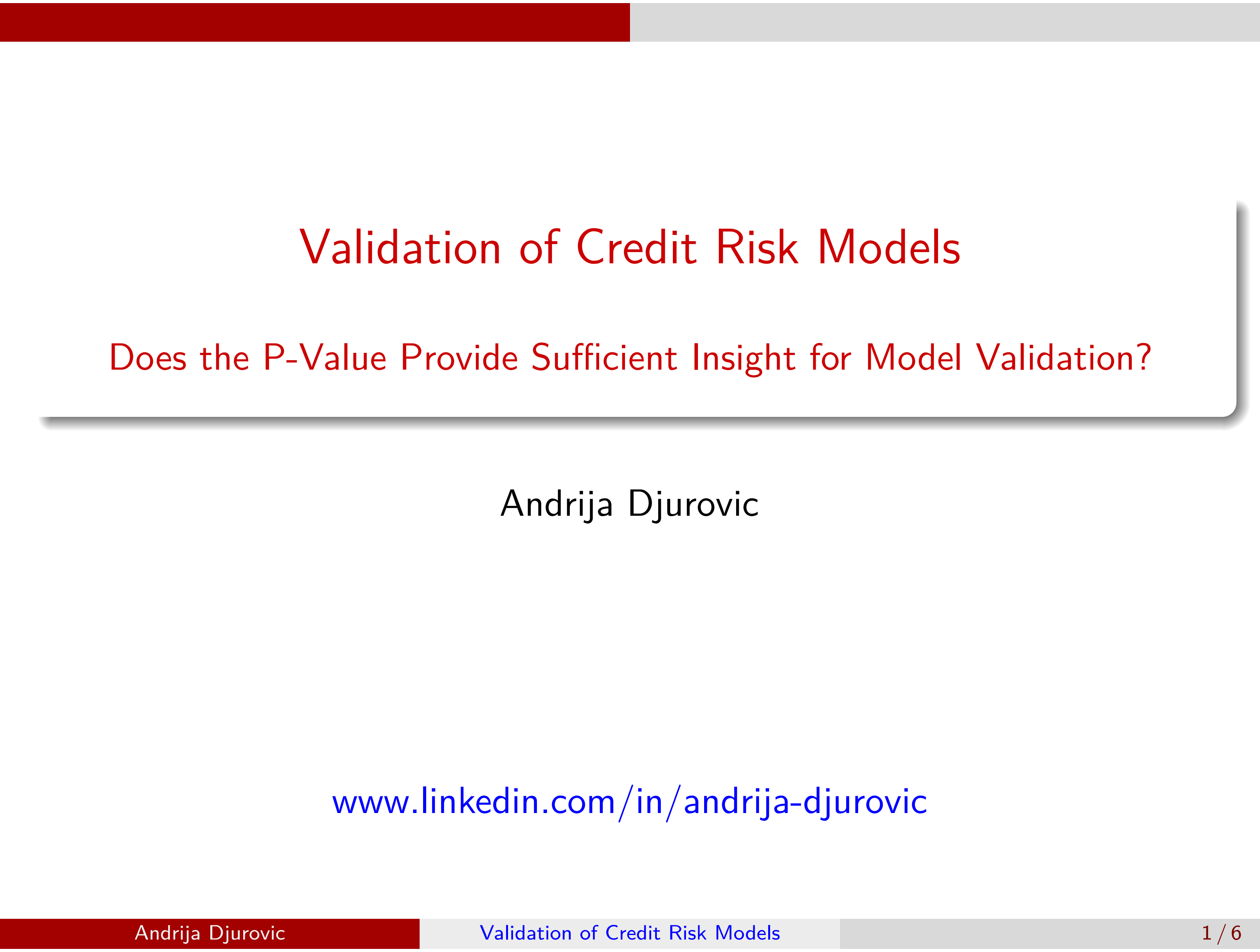
7 Alternative Methods and Report Customization

Although the presented examples demonstrate only a small part of the technical details of report automation, they provide a solid basis for extension to real-world cases. Practitioners can opt for formats other than the presented PDF format, such as MS Word, MS PowerPoint, or HTML. When evaluating different formats, practitioners should consider their pros and cons, as not all formatting options are available.

Other packages besides the `R` packages used for tables and graphs are available. Among the most popular are `tinytable` and `gt`. For practitioners who prefer `Python` over `R`, the same tasks can be accomplished in `Python`.

Finally, successful report automation depends not solely on the choice of statistical programs or packages but on the overall process design. Practitioners should prioritize process design as a key factor in ensuring a flexible and maintainable reporting system.

9.4 Validation of Credit Risk Models - Does the P-Value Provide Sufficient Insight for Model Validation?



Validation of Credit Risk Models

Does the P-Value Provide Sufficient Insight for Model Validation?

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Validation of Credit Risk Models | 1 / 6

Initial and Periodic Model Validation

- When validating credit risk models, practitioners typically formulate statistical hypotheses to evaluate various aspects of the model.
- The p-value resulting from statistical hypothesis testing is often the sole criterion used in reaching a final conclusion.
- Relying solely on the p-value raises several questions, such as:
    - Should practitioners adopt a unified approach based on the p-value for all portfolio types?
    - Should practitioners adopt a unified approach based on the p-value for all test types?
    - Is the p-value a sufficient criterion for making validation decisions?
- Practitioners rarely supplement the p-value from statistical tests with additional criteria, such as practical significance measures, during validation.
- The following slides present some of the most commonly used statistical procedures for testing predictive power as a function of different sample sizes.
  The main objective of the simulation is to demonstrate that, in practice, a p-value can show statistically significant results even when the change in the tested metric is not meaningful from a business perspective.
  Practitioners are encouraged to extend these examples to different simulation designs.

Z-score Test and Simulation Design

Z-score Test:

One of the most commonly used procedures for testing the predictive power of Probability of Default (PD) models is the z-score test, which is given in the following form:

$$z = \frac{ODR - PD}{\sqrt{\frac{PD(1-PD)}{n}}}$$

where:

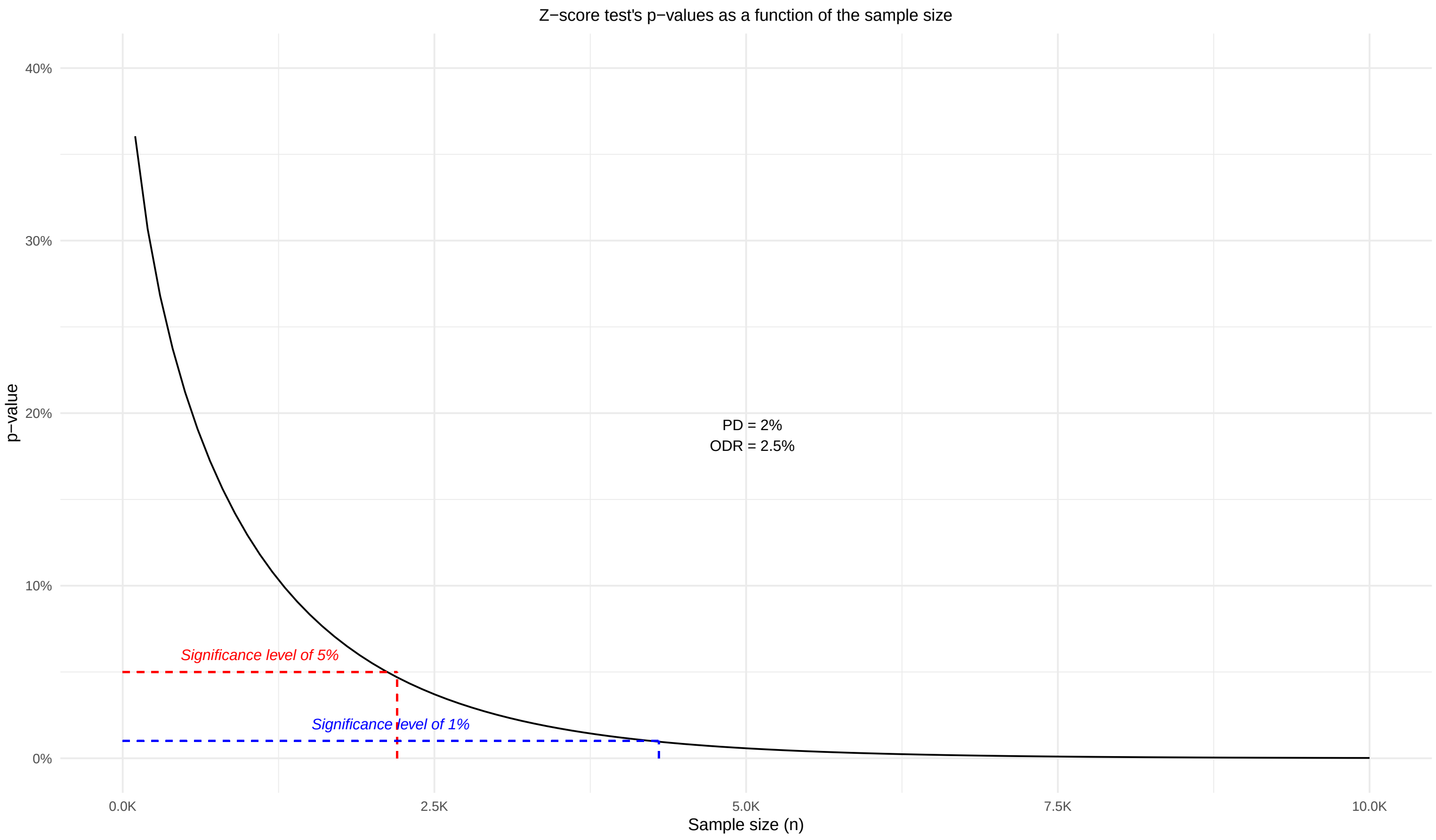
- ODR is the observed default rate;
- PD is the calibrated PD;
- n is the sample size.

Under the assumption that the `z` test statistic follows the standard normal distribution, a `p-value` is calculated accordingly.

Simulation Design:

The following slide presents the `p-value` as a function of the sample size for a calibrated PD of 2% and an ODR of 2.5%.

Z-score Test Simulation Results



Paired T-test and Simulation Design

Paired T-test:

A typical procedure for testing the predictive power of the Loss Given Default (LGD) and Exposure at Default (EAD) models is the paired t-test. The test statistic is calculated as follows:

$$t = \frac{\bar{x}}{\frac{s}{\sqrt{n}}}$$

where:

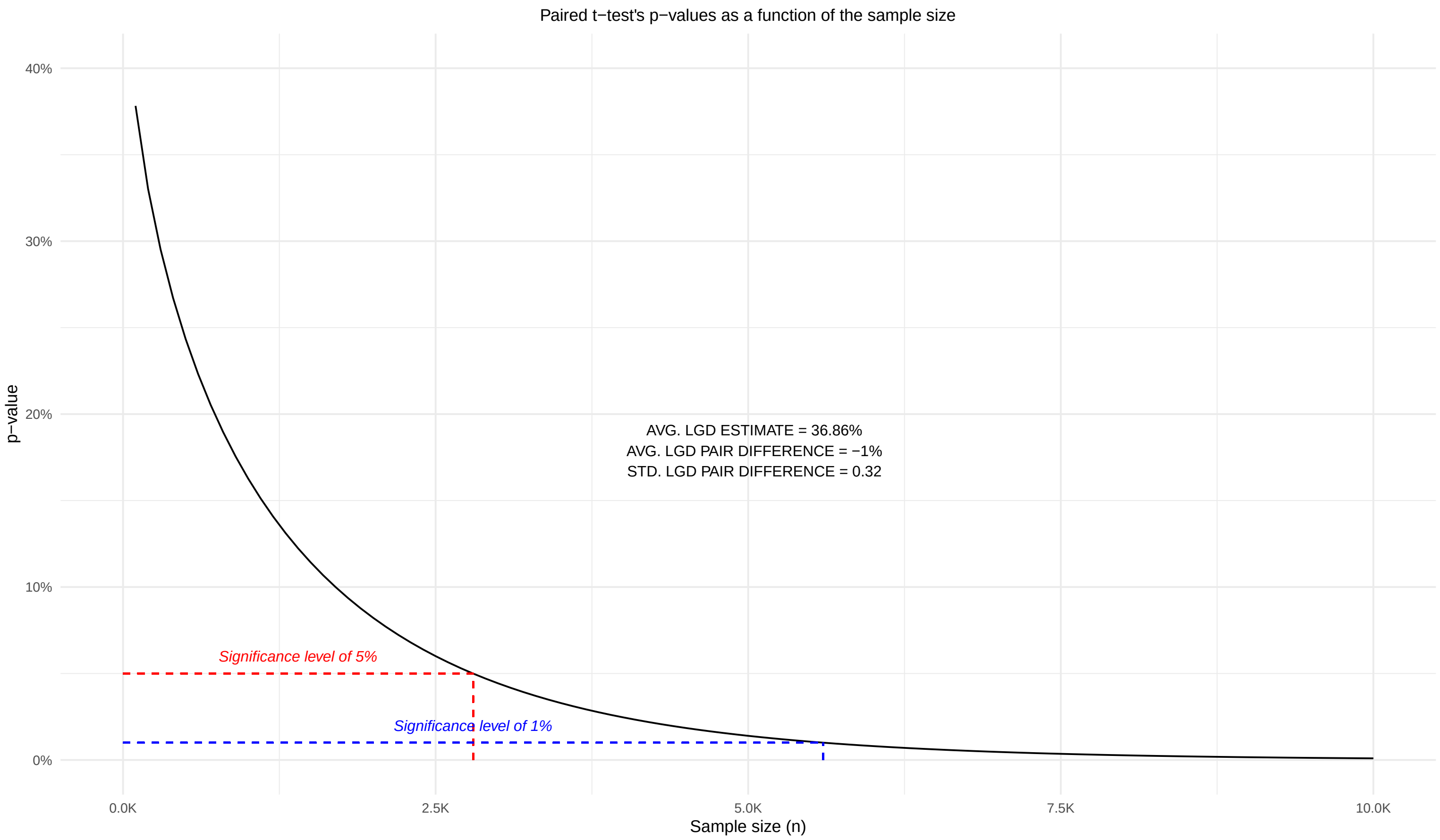
- $\bar{x}$ is the mean of the difference between model estimations and observed LGD/CCF values;
- s is the sample standard deviation of the difference between model estimations and observed LGD/CCF values;
- n denotes the sample size.

Assuming that the `t` test statistic follows a t-distribution with `n-1` degrees of freedom, a `p-value` is calculated accordingly.

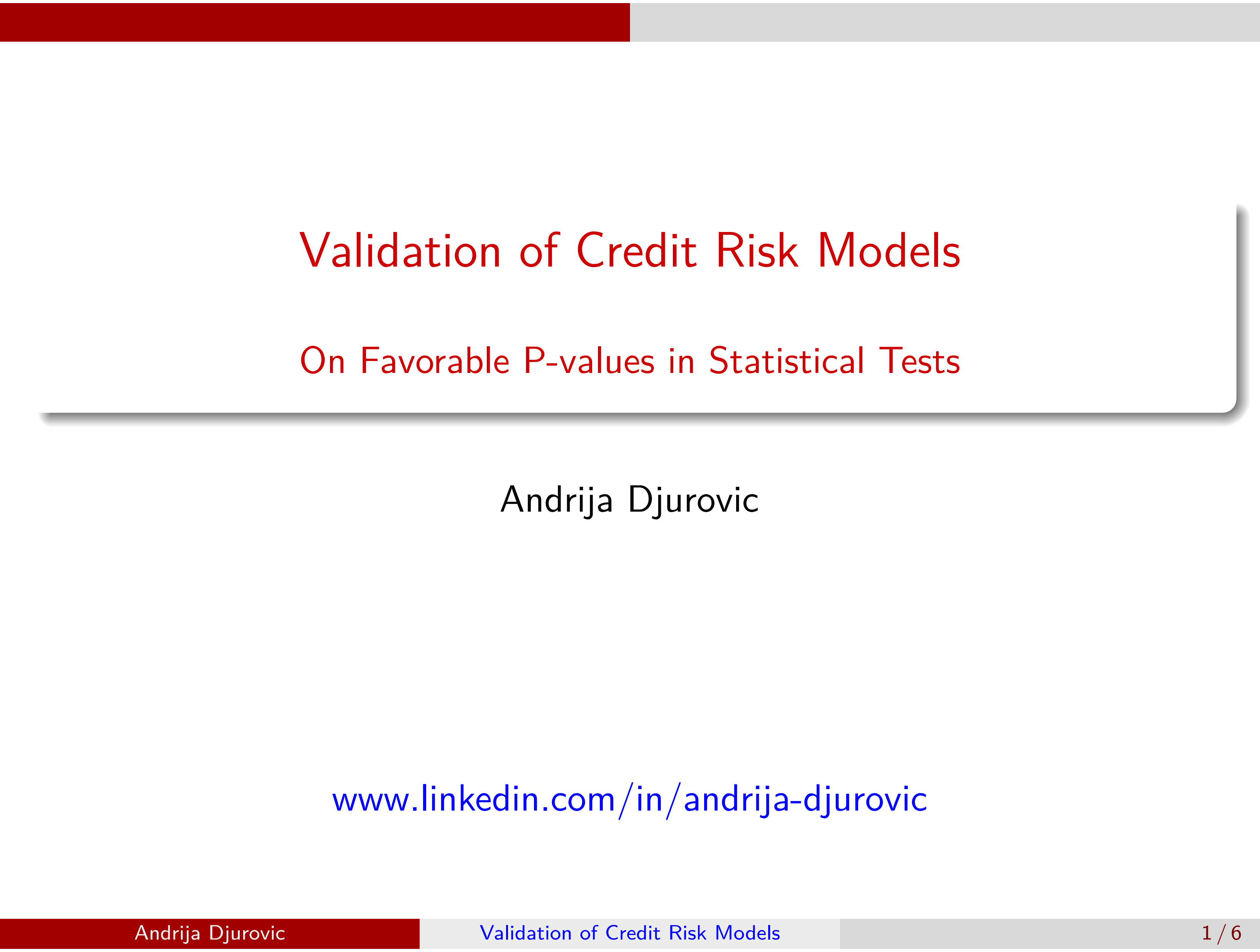
Simulation Design:

The following slide presents the p-value as a function of the sample size for an average LGD estimate of 36.86%, an average LGD pair difference of -1%, and a standard deviation of the LGD pair difference of 0.32.

Paired T-test Simulation Results



9.5 Validation of Credit Risk Models - On Favorable P-values in Statistical Tests



Validation of Credit Risk Models

On Favorable P-values in Statistical Tests

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Validation of Credit Risk Models | 1 / 6

Initial and Periodic Model Validation

- When validating credit risk models, practitioners typically formulate statistical hypotheses to evaluate various aspects of the model.
- The p-value resulting from statistical hypothesis testing is often the sole criterion for reaching a final conclusion.
- When the obtained p-value is unfavorable, practitioners may attempt to explain the test results from a business or practical perspective, seeking reasons why the unfavorable outcome might not be problematic.
- But what about favorable p-values in statistical tests? Should the conclusion be accepted without further investigation?
- The following slides present simulations of p-values from two predictive power tests commonly used to validate the Probability of Default (PD) models.
  The primary goal of these simulations is to show that, in practice, a favorable p-value can be obtained even when the observed default rate (ODR) consistently exceeds the calibrated PDs.
  Practitioners are encouraged to explore these examples further by testing different simulation designs and reconsidering the automatic acceptance of favorable test results based solely on p-values.

Z-score Test and Simulation Design

Z-score Test:

One of the most commonly used procedures for testing the predictive power of Probability of Default (PD) models is the z-score test, which is given in the following form:

$$z = \frac{ODR - PD}{\sqrt{\frac{PD(1-PD)}{n}}}$$

where:

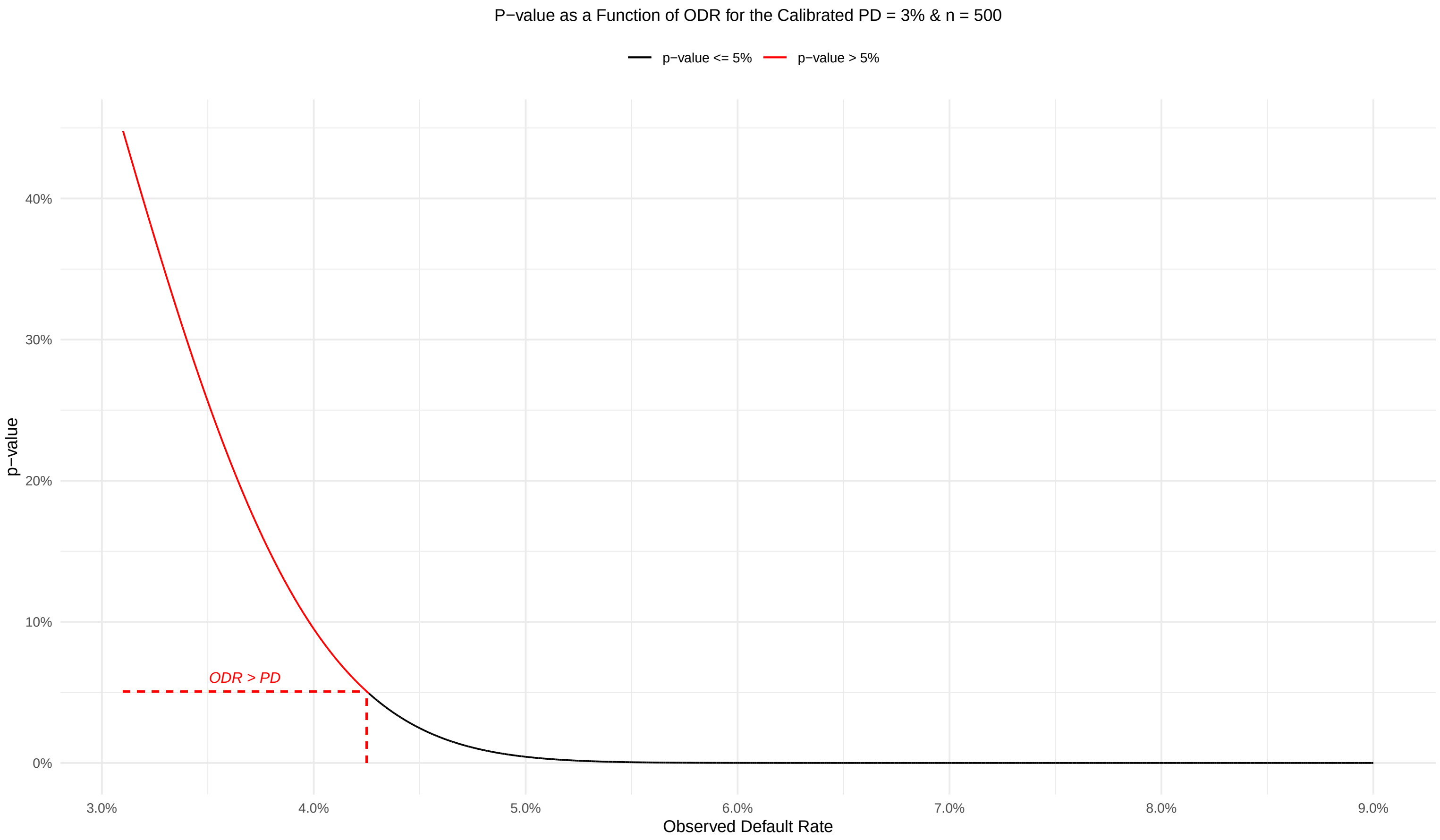
- ODR is the observed default rate;
- PD is the calibrated PD;
- n is the sample size.

Under the assumption that the z test statistic follows the standard normal distribution, a `p-value` is calculated accordingly.

Simulation Design:

The following slide presents the `p-value` as a function of the simulated ODR, with a calibrated PD of 3% and a sample size of 500.

Z-score Test Simulation Results



Multi-Period Normal Test and Simulation Design

Multi-Period Normal Test:

Under the null hypothesis that none of the true probabilities of default in the years `T` exceed their corresponding forecasted `PDs`, the test statistic for the multi-period normal test is calculated as follows:

$$Z_{nt} = \frac{\sum_{i=1}^{T}(\mathrm{ODR}_i - \mathrm{PD}_i)}{\sqrt{T}se}$$

where:

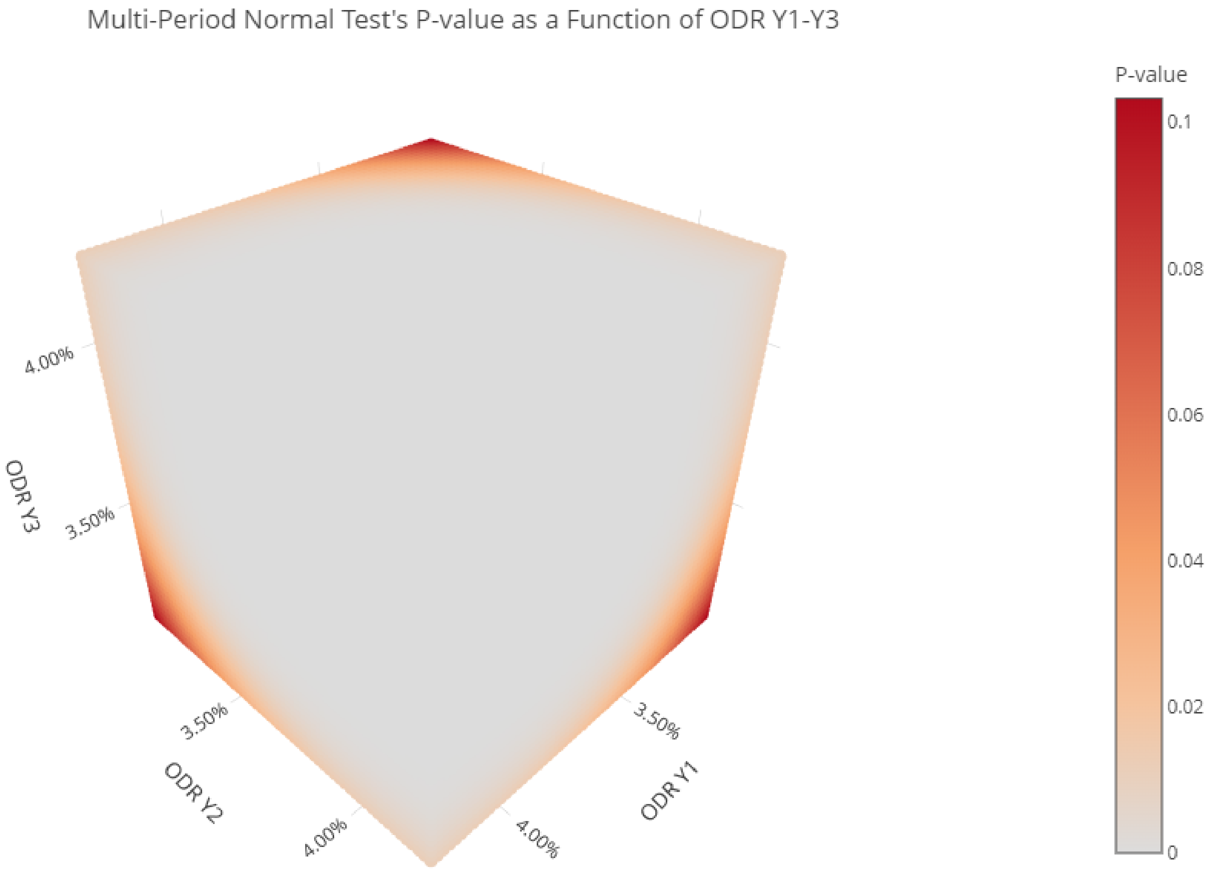
- PD is the calibrated PD;
- T is the number of years;
- se is the standard error defined as $\sqrt{\frac{1}{T-1}\left(\sum_{i=1}^{T}(ODR_i - PD_i)^2 - \frac{\left(\sum_{i=1}^{T}(ODR_i - PD_i)\right)^2}{T}\right)}$.

Assuming that the Z_{nt} test statistic follows the standard normal distribution, a `p-value` is calculated accordingly.

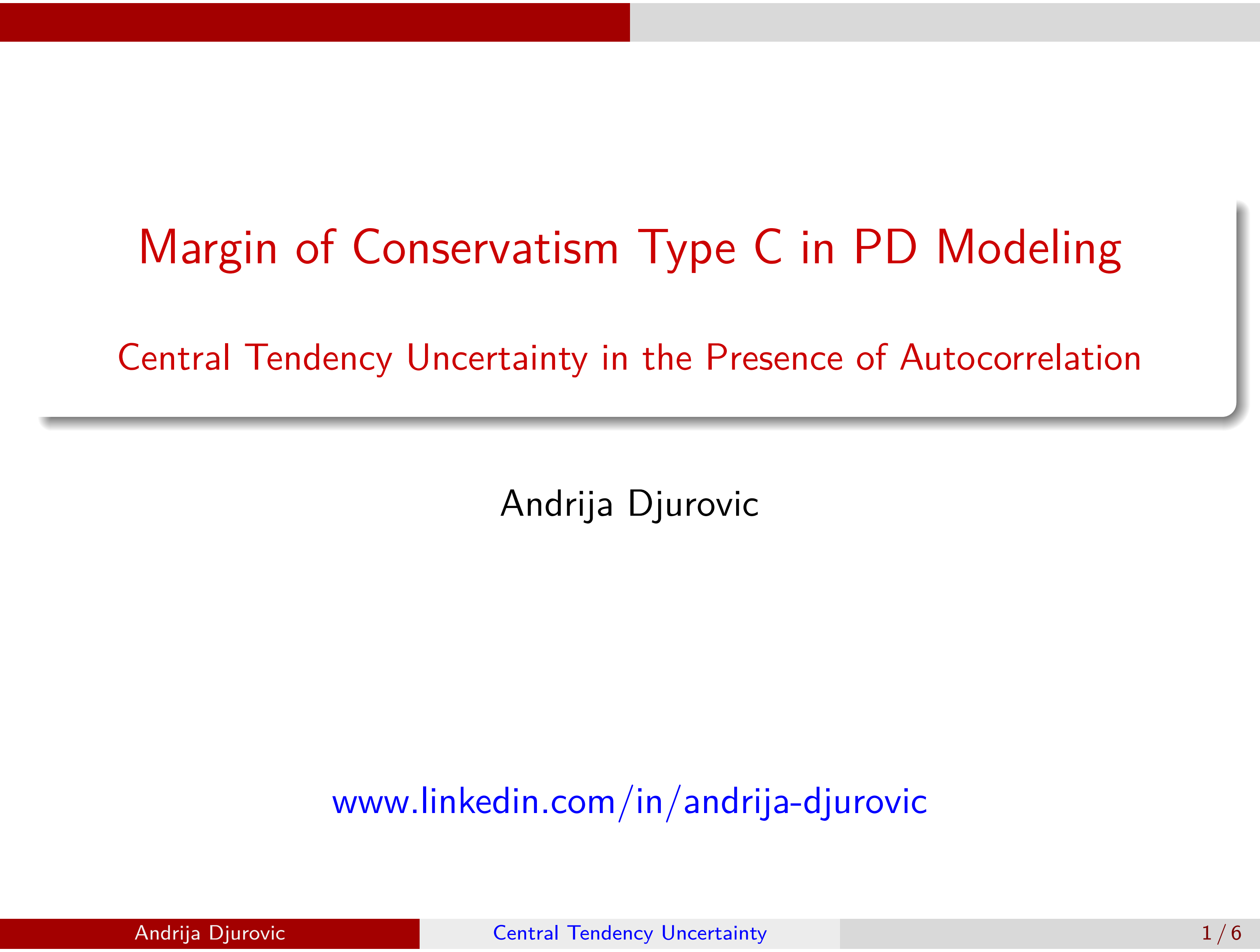
Simulation Design:

For three years (`T = 3`) ODR ranging from 3.1% to 4.25%, the following slide presents the `p-value` of the normal test for a calibrated PD of 3%. An interactive plot can be found here.

Multi-Period Normal Test Simulation Results



9.6 Margin of Conservatism Type C in PD Modeling - Central Tendency Uncertainty in the Presence of Autocorrelation



Margin of Conservatism Type C in PD Modeling

Central Tendency Uncertainty in the Presence of Autocorrelation

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic Central Tendency Uncertainty 1 / 6

Margin of Conservatism in PD modeling

- After completing the Probability of Default (PD) model development, practitioners usually address overall model uncertainties by defining the so-called Margin of Conservatism (MoC).
- The European Banking Authority (EBA) defines three types (categories) for distinct sources of uncertainty in the estimated parameters:
  1. Type A: covers the uncertainty related to the data and methodology deficiencies;
  2. Type B: covers the uncertainty related to changes in risk appetite and internal processes;
  3. Type C: covers the statistical uncertainty from the estimated parameter.
- One of the most commonly used MoC Type C is the uncertainty related to the estimation of the default rate central tendency.

Central Tendency Uncertainty in Practice

The central tendency (CT), defined as the mean of default rates over a specific period, inherently involves variability and uncertainty in its calculation.

Various methods can be employed to manage the variability associated with calculating the CT of default rates. The straightforward and most frequently used method in practice is the one that relies on the standard error of the mean, defined as:

$$\frac{\sigma}{\sqrt{n}}$$

with σ being the standard deviation of the observed default rates and n the number of observations in the observed period.

The direct application of the above method assumes that default rates between observed periods are independent. However, how realistic is this assumption in practice, considering the possibility of calculating the CT from non-overlapping and overlapping time frames?

How does autocorrelation affect the estimation of the standard error of the CT?

The following slides provide a simulation framework and compare results between independent and autocorrelated observations of default rates.

Simulation Setup

The following steps outline a Monte Carlo simulation to compare the standard error of the CT under assumptions of independence and correlated observations over the observation period. To align with the IRB framework, the observed default rates are simulated from the Vasicek distribution for parameters $PD = 0.05$ and $\rho = 0.10$, while the autoregressive assumption is implemented in developing the systemic factor z.

1. Select the sample size n and the autoregressive structure.
2. For the selected n, simulate independent observations from the Vasicek distribution with parameters $PD = 0.05$ and $\rho = 0.10$.
3. For the selected n and autoregressive structure (ϕ), first simulate the values of the systemic factor z:

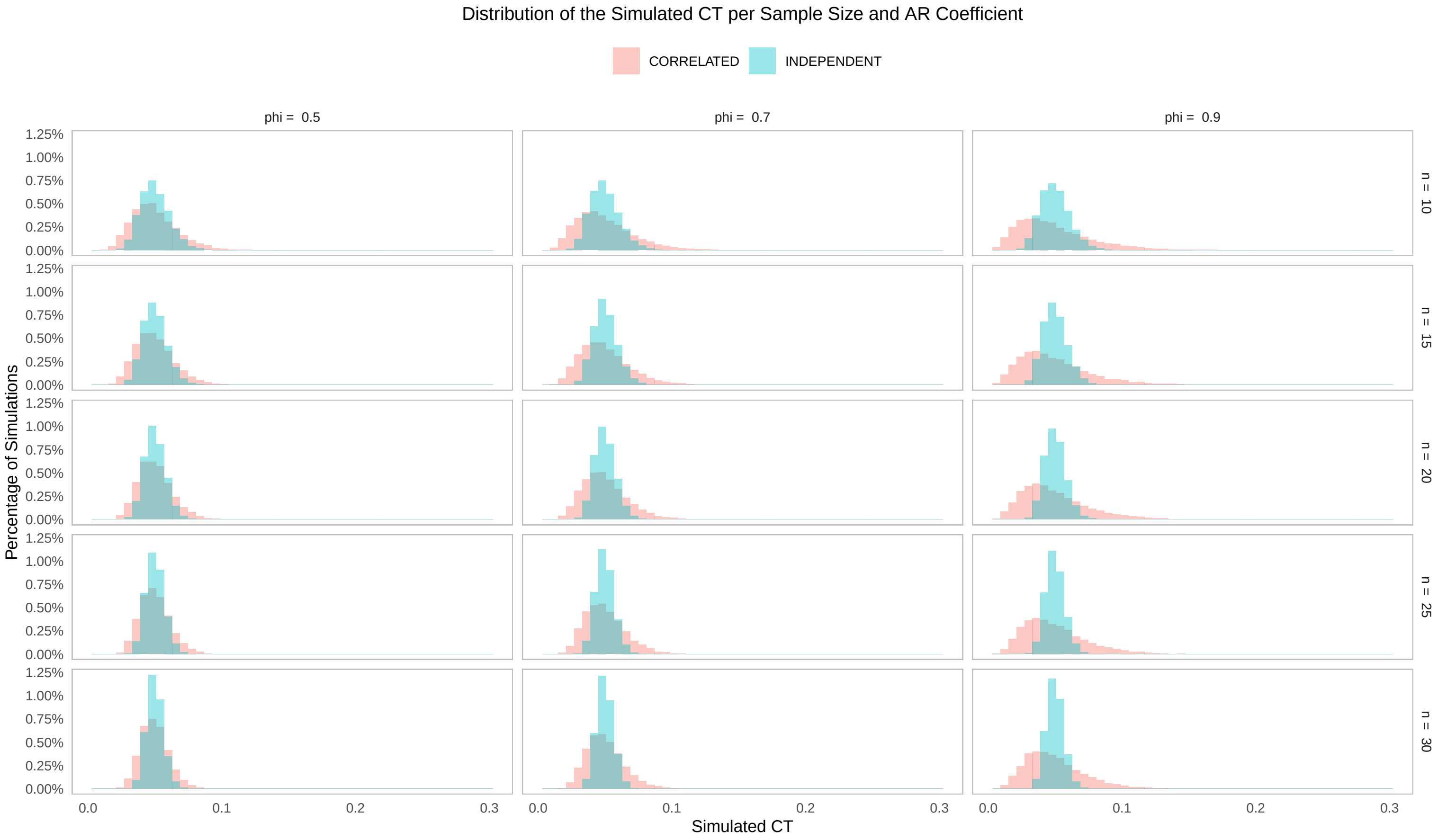
   $$z_t = \phi z_{t-1} + \sqrt{1-\phi^2}\epsilon_t$$

   with ϕ is the autoregressive coefficient of order 1 and ϵ is the error term from the standard normal distribution.Then, simulate the observed default rates from the Vasicek distribution given parameters $PD = 0.05$ and $\rho = 0.10$, and the autocorrelated values of the systemic factor z.
4. Calculate the mean of the simulated observed default rates from steps 2 and 3.
5. Repeat the steps 2 to 4, $N = 10,000$ times.
6. Compare the distributions and standard deviations of the simulated means under the assumptions of independence and correlated observations.

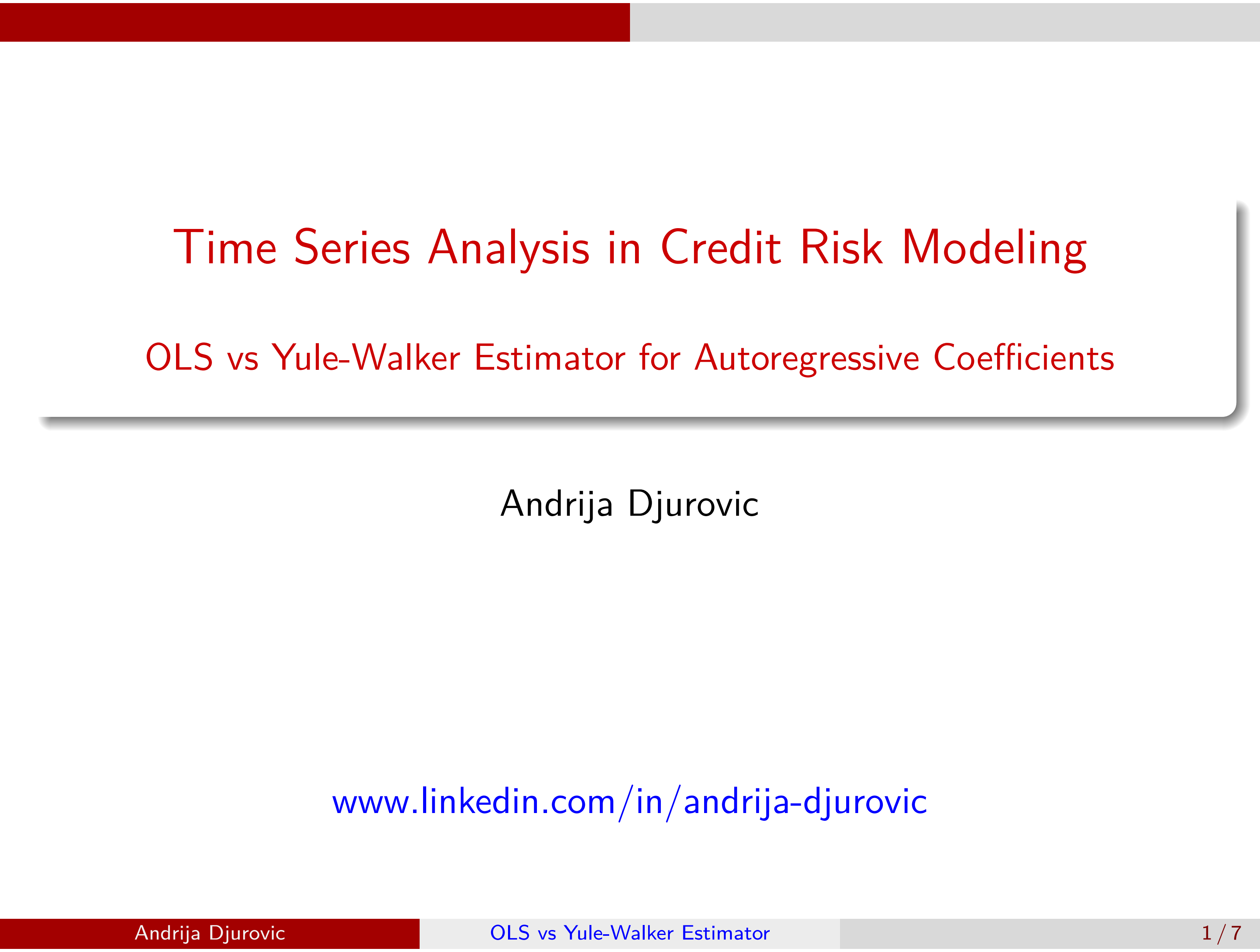
Simulation Results

```
##     pd rho phi  n sd correlated sd independent
##   0.05 0.1 0.5 10         0.0172        0.0111
##   0.05 0.1 0.5 15         0.0146        0.0090
##   0.05 0.1 0.5 20         0.0127        0.0078
##   0.05 0.1 0.5 25         0.0114        0.0070
##   0.05 0.1 0.5 30         0.0105        0.0063
##   0.05 0.1 0.7 10         0.0219        0.0111
##   0.05 0.1 0.7 15         0.0187        0.0089
##   0.05 0.1 0.7 20         0.0166        0.0078
##   0.05 0.1 0.7 25         0.0152        0.0069
##   0.05 0.1 0.7 30         0.0140        0.0063
##   0.05 0.1 0.9 10         0.0291        0.0110
##   0.05 0.1 0.9 15         0.0270        0.0090
##   0.05 0.1 0.9 20         0.0257        0.0079
##   0.05 0.1 0.9 25         0.0240        0.0069
##   0.05 0.1 0.9 30         0.0223        0.0064
```

Simulation Results cont.



9.7 Time Series Analysis in Credit Risk Modeling - OLS vs Yule-Walker Estimator for Autoregressive Coefficients



Time Series Analysis in Credit Risk Modeling

OLS vs Yule-Walker Estimator for Autoregressive Coefficients

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | OLS vs Yule-Walker Estimator | 1 / 7

Time Series Modeling in Credit Risk

- Certain areas of credit risk modeling involve working with and modeling time series.
- In Probability of Default (PD) modeling, time series play an important role in at least three regulatory areas:
  1. IFRS9 forward-looking modeling;
  2. quantifying central tendency uncertainty;
  3. producing simulation-based PD estimates for low-default portfolios.
- A crucial distinction between time series and independently collected data is that the order of observations significantly affects the modeling exercises.
- One of the most challenging aspects of time series modeling is appropriately addressing the autocorrelation structure.
- In simple terms, autocorrelation refers to the relationship between a variable's current and past values.
- Two commonly used methods for estimating autocorrelation coefficients in credit risk modeling are Ordinary Least Squares (OLS) and Yule-Walker (YW).
- The following slides present the basic formulas for OLS and YW estimators and compare their performance through Monte Carlo simulations.

OLS Estimator

Parameter estimation:

$$\hat{\beta} = (X^{'}X)^{-1}X^{'}Y$$

Standard error of parameters:

$$se(\hat{\beta}) = \sqrt{\hat{\sigma}^2(X'X)^{-1}}$$

with $\hat{\sigma}^2$ being:

$$\hat{\sigma}^2 = (n-k-1)^{-1}\hat{\varepsilon}^{'}\hat{\varepsilon}$$

where:

- X is the design matrix;
- Y denotes the vector of observed values (dependent variable);
- $\hat{\varepsilon}$ represents the regression residuals.

Yule-Walker Estimator and Ljung-Box Test

The Yule-Walker equations relate the autocorrelation function of a time series to the parameters of the autoregressive (AR) model. By solving the following system of equations, the model parameters are obtained:

$$\begin{bmatrix} \rho(0) & \rho(1) & \cdots & \rho(p-1) \\ \rho(1) & \rho(0) & \cdots & \rho(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ \rho(p-1) & \rho(p-2) & \cdots & \rho(0) \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_p \end{bmatrix} = \begin{bmatrix} \rho(1) \\ \rho(2) \\ \vdots \\ \rho(p) \end{bmatrix}$$

where:

- $\rho(k)$ is the autocorrelation function at lag k;
- ϕ_i represents the parameters of the AR model;
- p is the order of the autoregressive model.

The Ljung-Box test statistic is given by the following expression:

$$Q = n(n+2) \sum_{k=1}^{h} \frac{\hat{\rho}_k^2}{n-k}$$

where:

- n is the sample size;
- h is the number of lags tested;
- $\hat{\rho}_k$ is the sample autocorrelation at lag k.

The test statistic Q follows a chi-square distribution with h degrees of freedom under the null hypothesis that the data are independently distributed.

Simulation Setup

The following steps detail the simulation design used to compare the two autoregressive coefficient estimators' bias and examine the statistical power of the OLS and Ljung-Box tests. For simplicity, only an autoregressive process of order one is considered.

1. Select the order of the autoregressive process ($\phi = 0.5$).
2. Select sample size n (10, 15, 20, 25, 30, 100).
3. Simulate an autoregressive process of order 1 for the chosen values of ϕ and n:

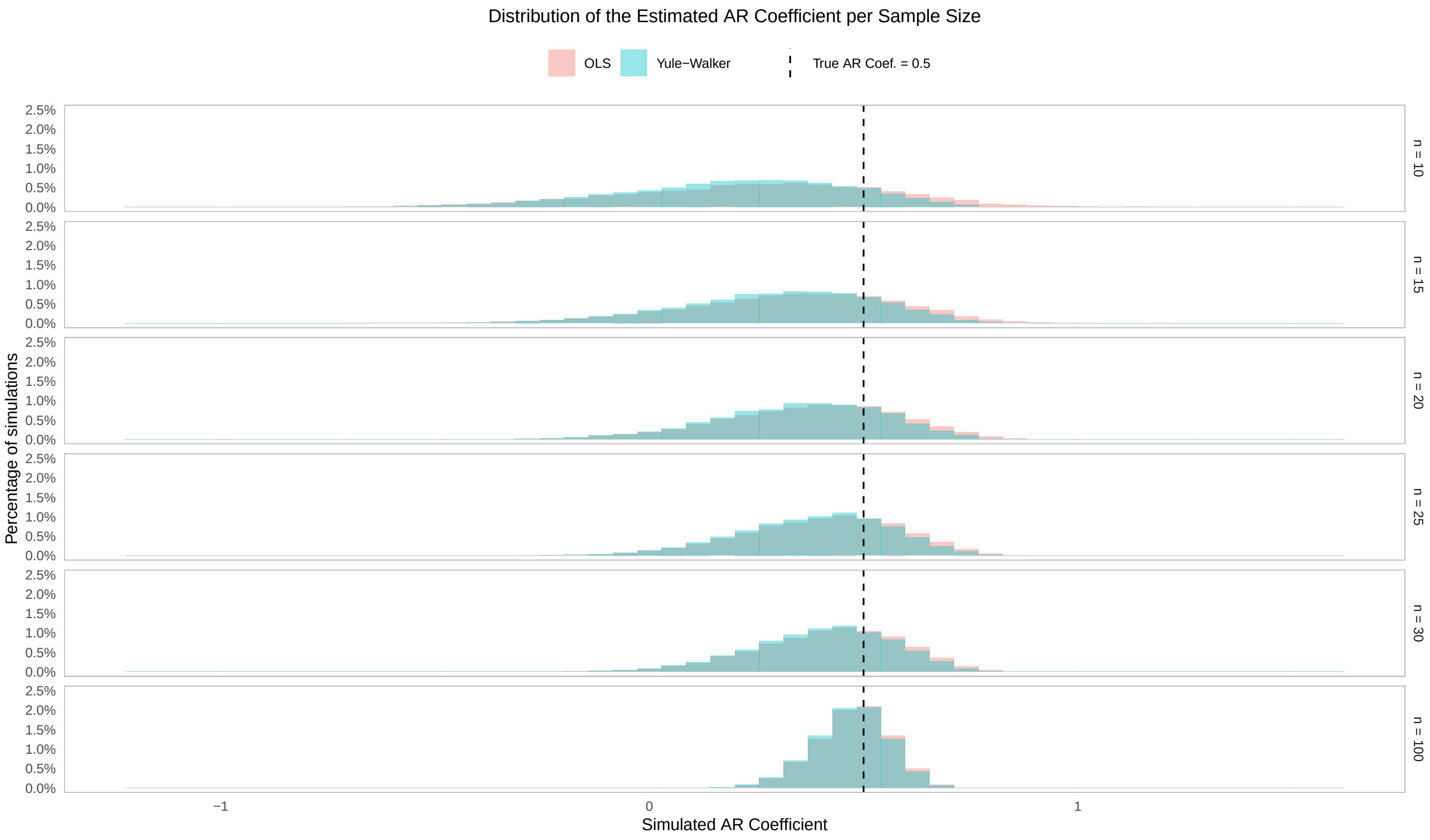
   $$x_{t,t \leq n} = \phi x_{t-1} + \sqrt{1 - \phi^2} \varepsilon_t$$

   where ε_t is drawn from the standard normal distribution.
4. Using the simulated values of x from step 3, estimate the autoregressive coefficient using the OLS and Yule-Walker methods. Calculate the standard error and p-value of the OLS estimate and obtain the p-value from the Ljung-Box test.
5. Repeat steps 3 and 4 for $N = 10,000$ simulations, storing the results of the estimations.
6. Calculate the bias of the estimators as the difference between the average value of the simulated estimates and the true autoregressive coefficient ($\phi = 0.5$).
7. Calculate the statistical power of the OLS and Ljung-Box tests at a significance level of 0.05.

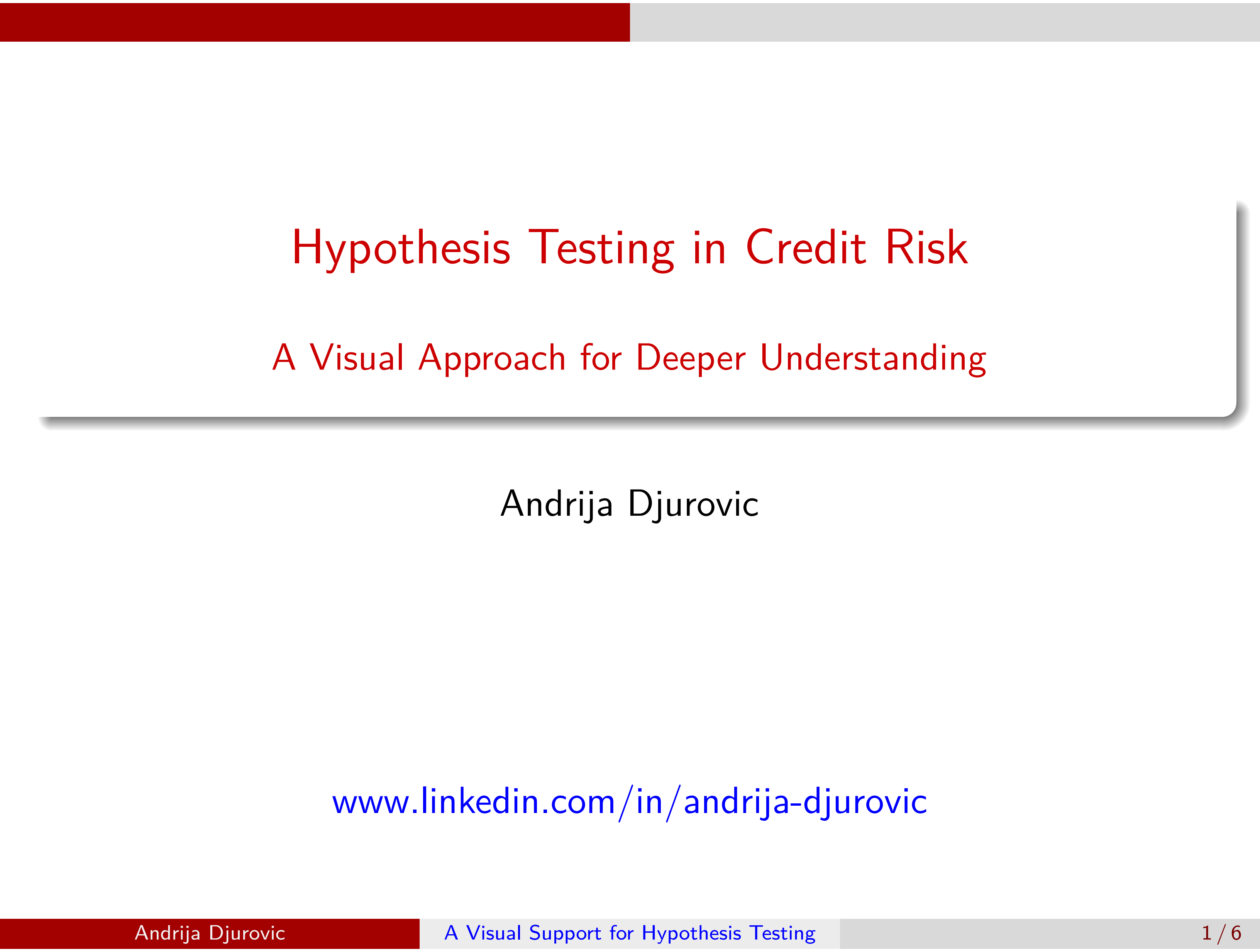
Simulation Results

```
##        type   T mean.estimator    bias stat.power
##         OLS  10         0.2440 -0.2560     0.0712
## Yule-Walker  10         0.2043 -0.2957     0.1018
##         OLS  15         0.3263 -0.1737     0.2129
## Yule-Walker  15         0.2990 -0.2010     0.2613
##         OLS  20         0.3719 -0.1281     0.3775
## Yule-Walker  20         0.3494 -0.1506     0.4167
##         OLS  25         0.3987 -0.1013     0.5219
## Yule-Walker  25         0.3803 -0.1197     0.5550
##         OLS  30         0.4170 -0.0830     0.6415
## Yule-Walker  30         0.4014 -0.0986     0.6662
##         OLS 100         0.4745 -0.0255     0.9968
## Yule-Walker 100         0.4696 -0.0304     0.9969
```

Simulation Results cont.



9.8 Hypothesis Testing in Credit Risk - A Visual Approach for Deeper Understanding



Hypothesis Testing in Credit Risk

A Visual Approach for Deeper Understanding

Andrija Djurovic

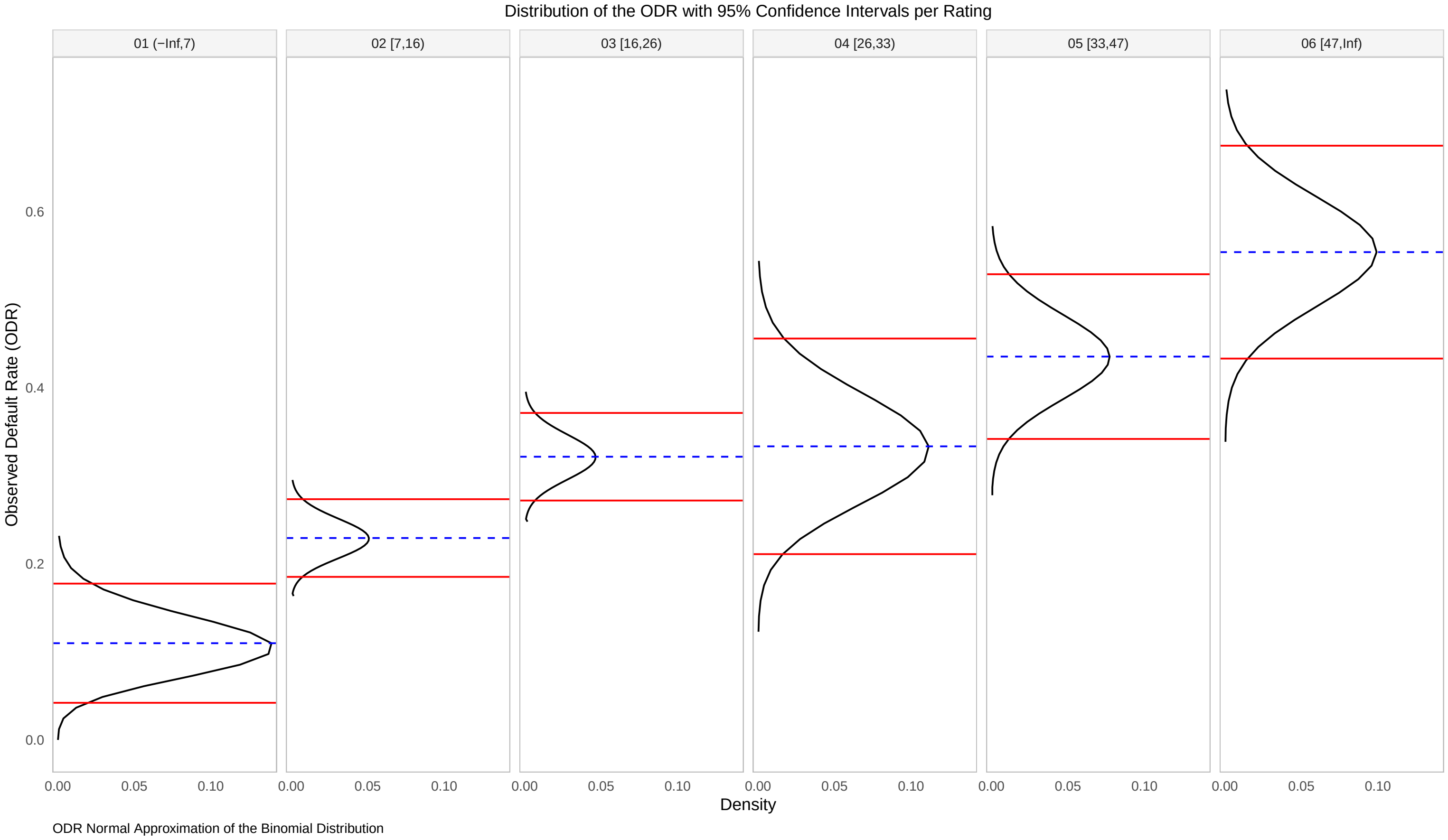
www.linkedin.com/in/andrija-djurovic

Andrija Djurovic A Visual Support for Hypothesis Testing 1 / 6

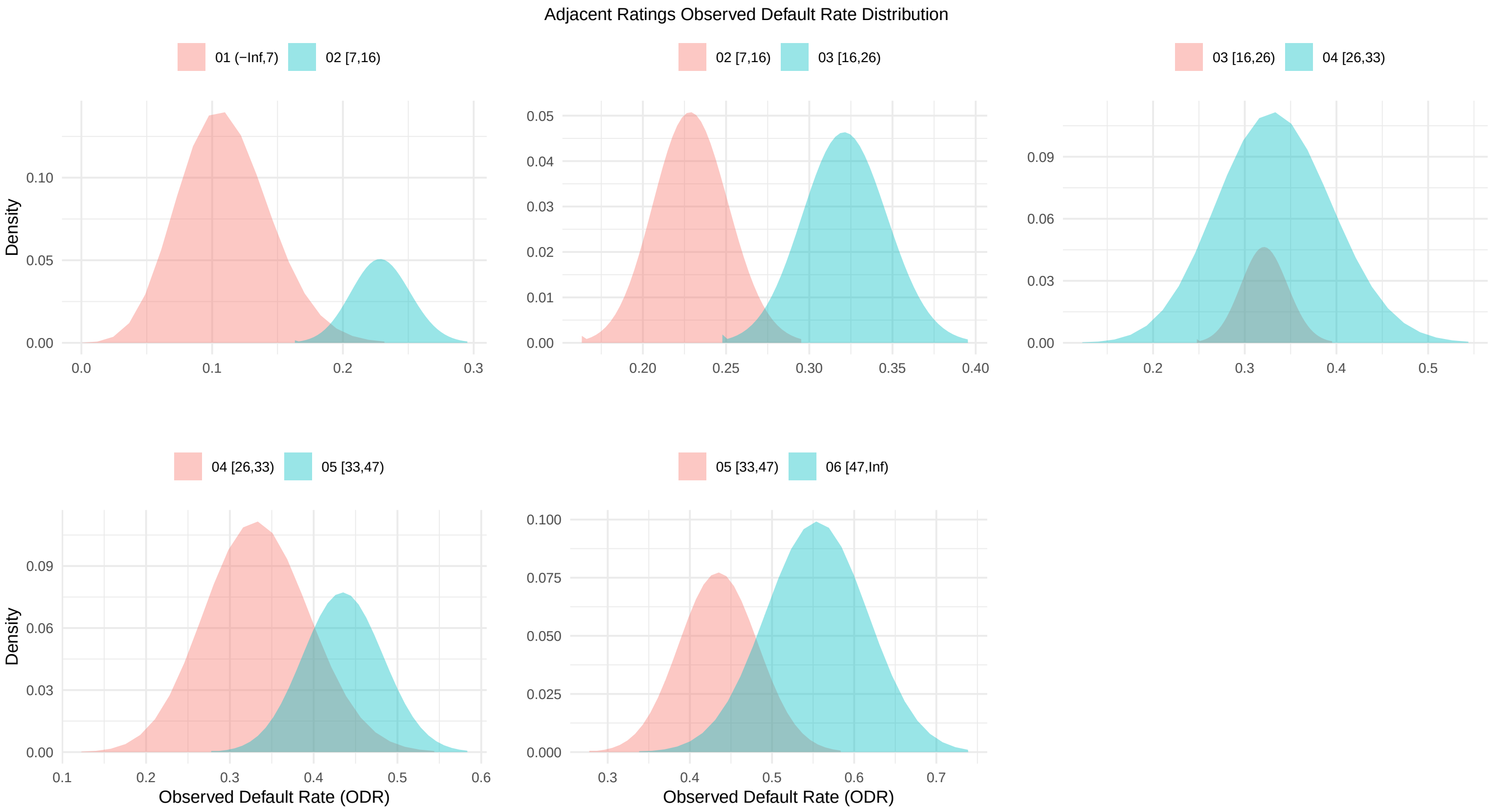
Hypothesis Testing in Credit Risk

- Statistical hypothesis testing is extensively used in credit risk, especially for validating various types of credit risk models, including Probability of Default (PD), Loss Given Default (LGD), and Exposure at Default (EAD).
- Practitioners typically formulate statistical hypotheses to assess various aspects of the model, such as the quality of the ranking order, the model's predictive power, or the model's structure quality.
- Two commonly applied tests that evaluate the quality of a model's structure are heterogeneity and homogeneity. These procedures are used across all risk models. The tests are conducted at the rating grade level for PD models, while for LGD and EAD models, they are applied at the pool or bucket levels.
- Heterogeneity testing focuses on assessing the differences in risk profiles between adjacent ratings, pools, or buckets of facilities or obligors. On the other hand, homogeneity testing examines the consistency of risk profiles within a single rating, pool, or bucket.
- Practitioners may sometimes need assistance understanding what is being tested in these procedures. In such cases, a graphical representation of the metric and its distribution can help clarify the purpose of the analysis and support the final decision-making.
- The following slides present visualization examples that can assist in heterogeneity and homogeneity testing for credit risk models. Practitioners are encouraged to see these examples as flexible guidelines rather than fixed templates and to adapt them to their specific needs. Readers can find the underlying data for the graphs here: PD, LGD.

Example 1: Heterogeneity Testing in PD Modeling

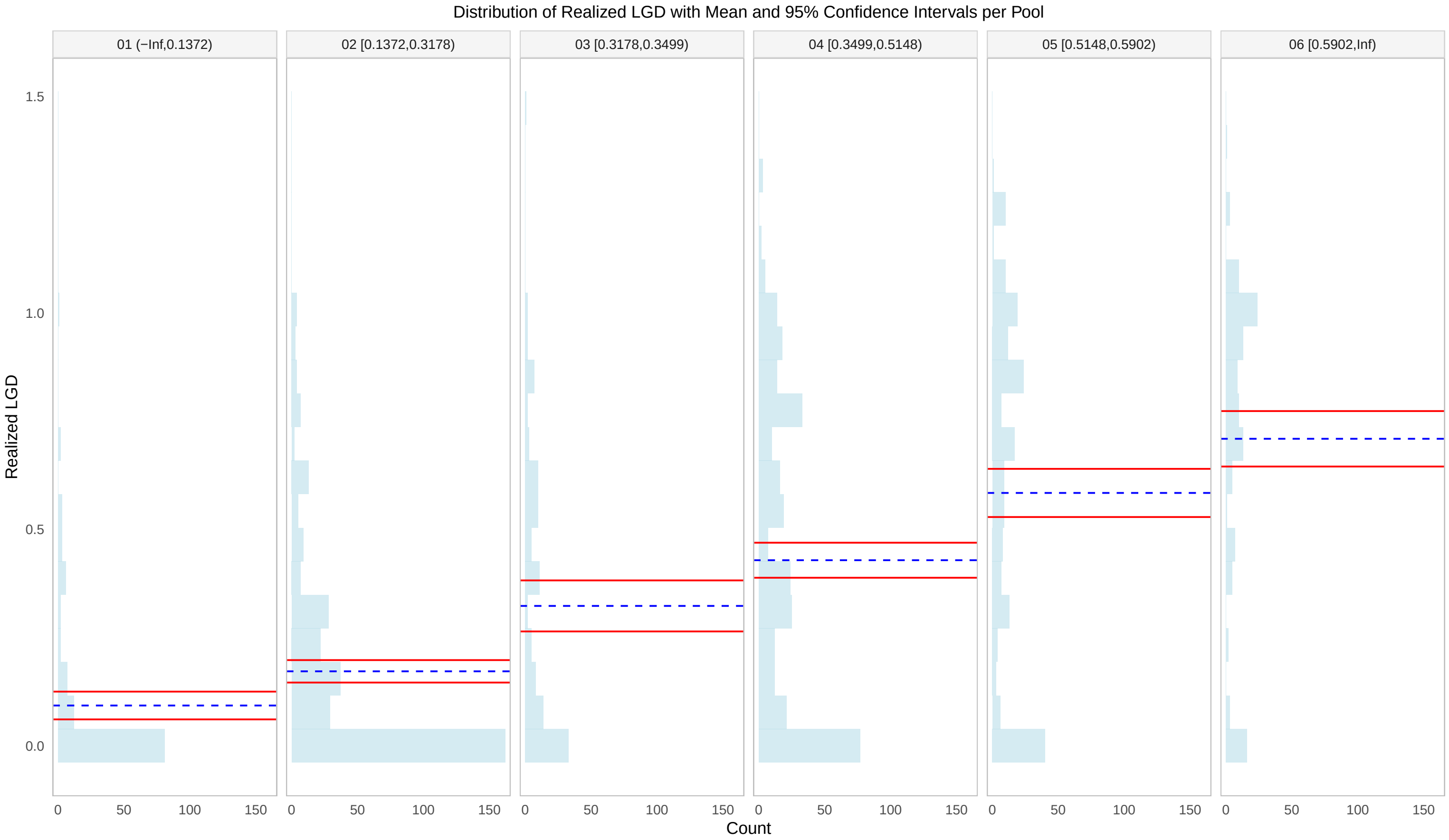


Example 2: Heterogeneity Testing in PD Modeling

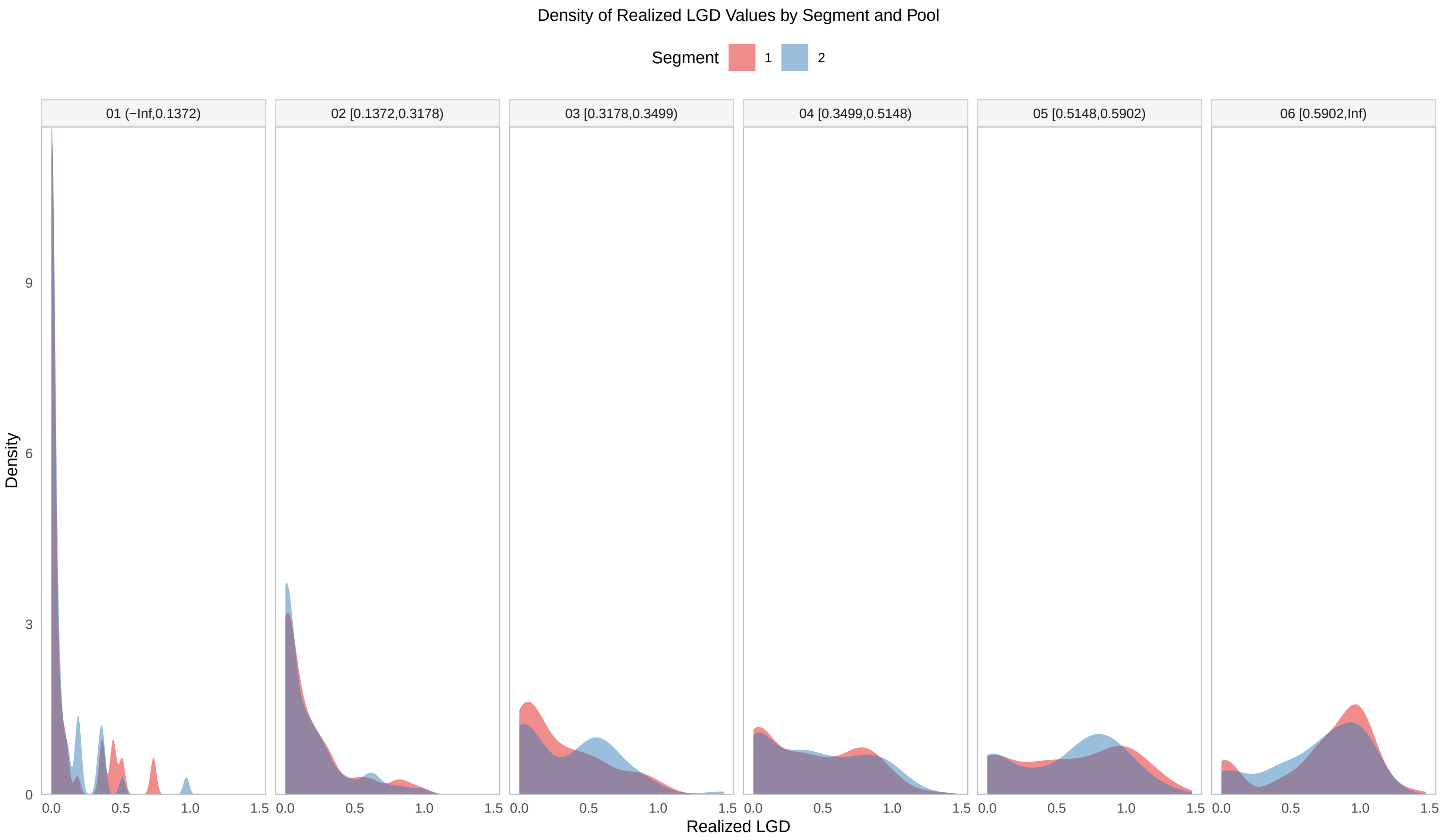


ODR Normal Approximation of the Binomial Distribution

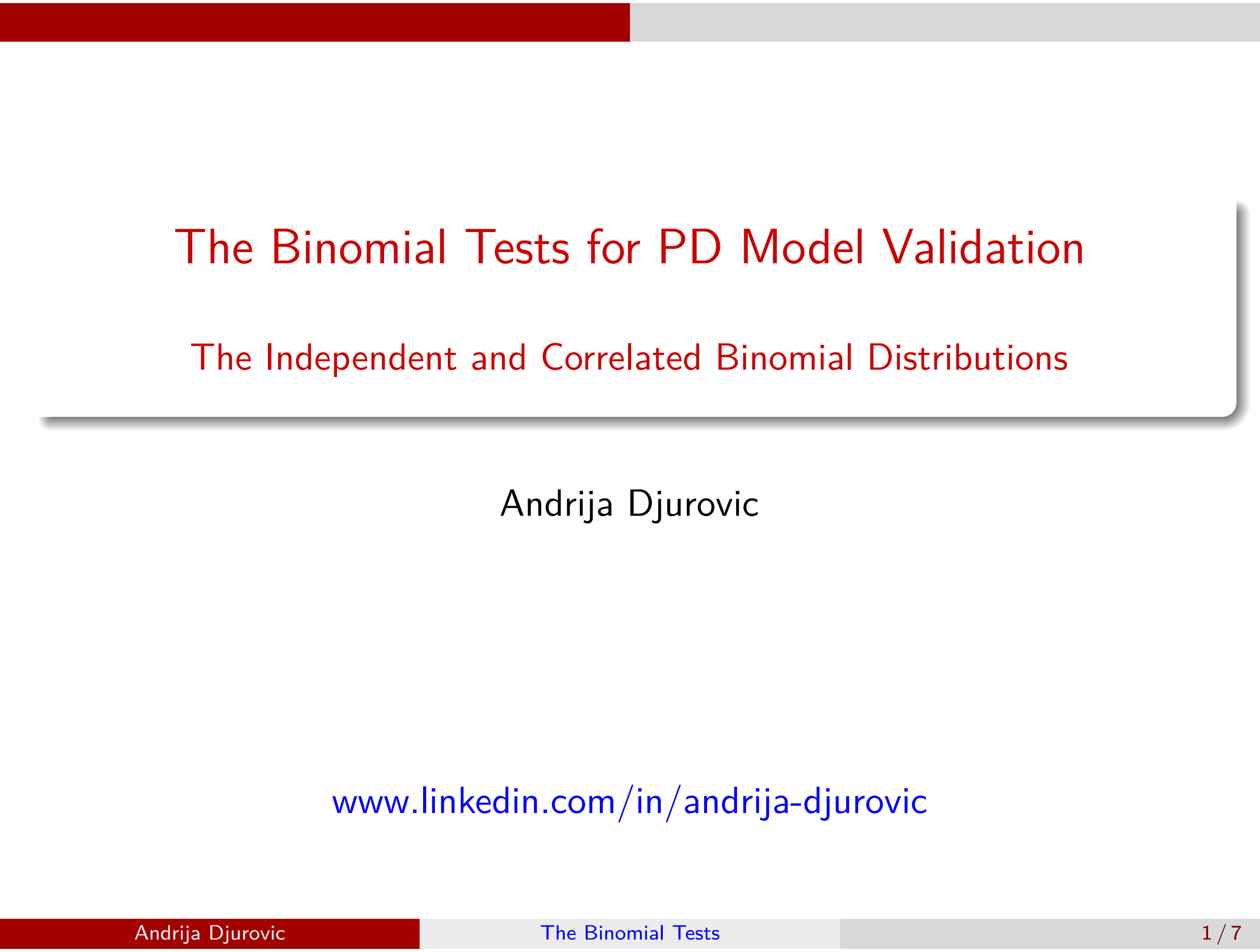
Example 3: Heterogeneity Testing in LGD Modeling



Example 4: Homogeneity Testing in LGD Modeling



9.9 The Binomial Tests for PD Model Validation - The Independent and Correlated Binomial Distributions



The Binomial Tests for PD Model Validation

The Independent and Correlated Binomial Distributions

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | The Binomial Tests | 1 / 7

The Binomial Test for PD Model Validation

- The binomial test is a frequently employed method for validating Probability of Default (PD) estimates reported for each rating category of internal rating systems.
- The binomial test works under the null hypothesis that the PD of a rating is not underestimated.
- The most common implementation of the binomial test relies on the assumption that default events in the rating category under consideration are independent.
- How can practitioners adjust the standard binomial test to account for correlated defaults?
- How do the results of independent and correlated binomial tests compare?

The Binomial Test - Independent Defaults

The most commonly used version of the binomial test relies on the assumption that default events in the rating category under consideration are independent.

Given a confidence level q, the null hypothesis is rejected if the number of defaulters k in this rating category is greater than or equal to a critical value $k*$, which is defined as:

$$k^* = min\left\{ k \mid \sum_{i=k}^{n} \binom{n}{i} PD^i (1 - PD)^{n-i} \leq 1 - q \right\}$$

where:

- n is the number of debtors;
- PD is the probability of default in the rating category.

The Binomial Test - Correlated Defaults

Applying the binomial test under the assumption of correlated defaults complicates the mathematical framework.

Two methods are presented to analyze the impact of default correlation on the critical value of the number of defaulters.

The first method involves exact numerical calculation through numerical integration:

$$P(D_n \leq k) = \int_{-\infty}^{\infty} \sum_{i=0}^{k} \binom{n}{i} \left[\Phi \left(\frac{\Phi^{-1}(PD) - \sqrt{\rho}\, x}{\sqrt{1-\rho}} \right) \right]^{i} \left[1 - \Phi \left(\frac{\Phi^{-1}(PD) - \sqrt{\rho}\, x}{\sqrt{1-\rho}} \right) \right]^{n-i} \phi(x)\, dx$$

The Binomial Test - Correlated Defaults cont.

The second method is an analytical approximation defined as follows:

$$P(D_n \leq k) = \Phi\left(\frac{\sqrt{1-\rho}\,\Phi^{-1}\binom{k}{n} - \Phi^{-1}(PD)}{\sqrt{\rho}}\right)$$

where:

- D_n is the observed number of defaults;
- n is the total number of debtors in the rating category;
- PD is the probability of default in the rating category;
- ρ represents the asset correlation;
- $\phi(x)$ denotes the standard normal density;
- Φ denotes the distribution of the standard normal distribution.

Simulation Results

Correlated Defaults

Example 1:

```
   pd    n  rho    q exact k appox. k
 0.01 1000 0.00 0.99      19       11
 0.01 1000 0.05 0.99      35       32
 0.01 1000 0.10 0.99      49       47
 0.01 1000 0.15 0.99      63       62
 0.01 1000 0.20 0.99      77       76
```

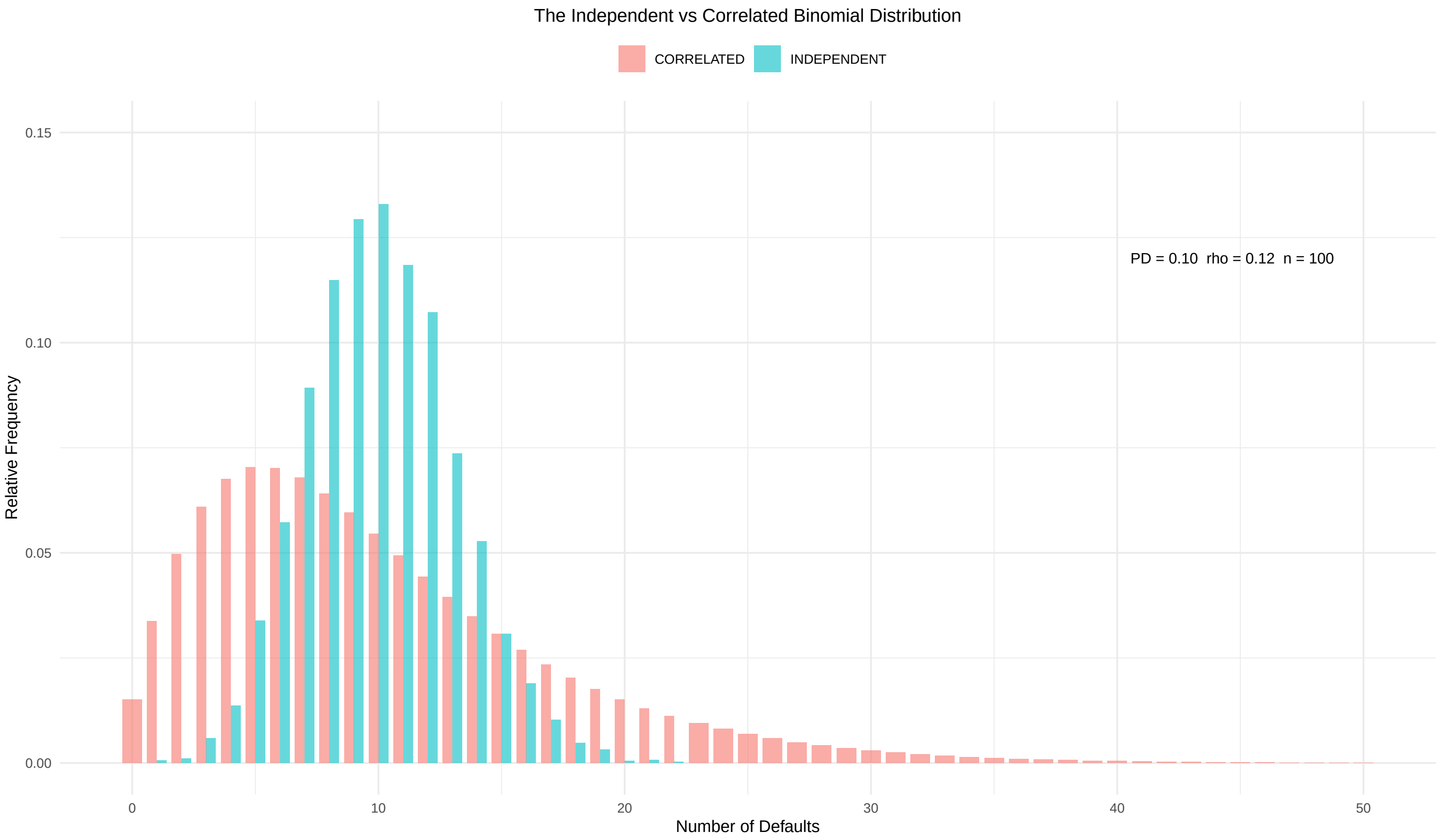
Example 2:

```
   pd    n  rho    q exact k appox. k
 0.05 1000 0.00 0.99      68       51
 0.05 1000 0.05 0.99     128      125
 0.05 1000 0.10 0.99     172      169
 0.05 1000 0.15 0.99     212      210
 0.05 1000 0.20 0.99     252      250
```

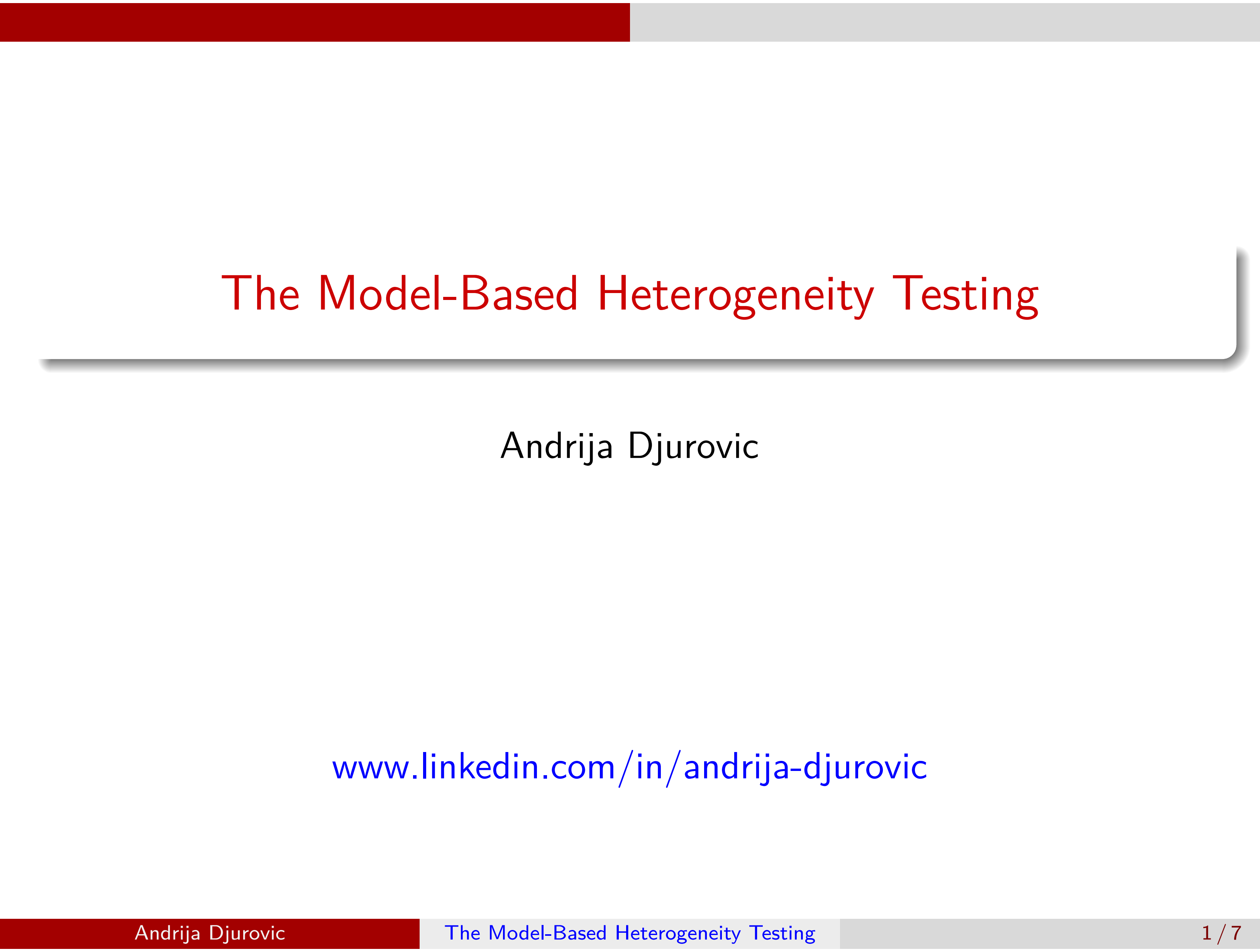
Independent Defaults

```
   pd    n    q  k
 0.01 1000 0.99 19
 0.05 1000 0.99 68
```

Simulation Results cont.



9.10 The Model-Based Heterogeneity Testing



The Model-Based Heterogeneity Testing

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic The Model-Based Heterogeneity Testing 1 / 7

Probability of Default Rating Scale

- Practitioners usually adhere to certain principles when building the Probability of Default (PD) rating scale.
- These principles lead to specific rating scale characteristics, with some being compulsory, while others may vary from one model to another and are often considered desirable but not mandatory.
- Among the compulsory characteristics of the rating scale, practitioners usually consider monotonicity and heterogeneity.

Heterogeneity Testing

- A typical way to test for heterogeneity of the rating scale is the test of two proportions on adjacent grades.
- Model-based testing offers an alternative method for conducting heterogeneity testing.
- This type of testing is based on logistic regression and the so-called nested dummy encoding of the rating grades.
- The main advantage of this type of testing is that it produces estimates that align with the most commonly used method for estimating PD models: logistic regression.
- The following slides explain the concept of nested dummy encodings, their interpretation within logistic regression, and the comparison of results between the standard heterogeneity testing method and model-based methods.

Nested Dummy Encoding in Credit Risk Modeling

- Connecting adjacent categories makes this encoding method particularly attractive in credit risk modeling.
- It is especially appealing when practitioners choose to bin numeric risk factors and seek statistically significant risk profiles between adjacent categories.
- Extending the above application, this method demonstrates its adaptability, providing opportunities to assess diverse binning methods in standard multivariate analyses.
- Another use of this method is in model-based hypothesis testing of the heterogeneity of the rating scale.

Constructing Nested Dummies

Assign a value of 0 to all categories below the chosen one and 1 to the selected category and those positioned above it.

Example

```
categories = c("A", "A", "B", "B", "C", "C", "C", "D", "D", "D")
```

```
nested_dummies
```

```
##    category_A_vs_BCD category_AB_vs_CD category_ABC_vs_D
## 1                  0                 0                 0
## 2                  0                 0                 0
## 3                  1                 0                 0
## 4                  1                 0                 0
## 5                  1                 1                 0
## 6                  1                 1                 0
## 7                  1                 1                 0
## 8                  1                 1                 1
## 9                  1                 1                 1
## 10                 1                 1                 1
```

Interpreting Nested Dummies

Binomial logistic regression with nested dummies

```
##
## Call:  glm(formula = y ~ category_A_vs_BCD + category_AB_vs_CD + category_ABC_vs_D,
##     family = "binomial", data = db)
##
## Coefficients:
##       (Intercept)  category_A_vs_BCD  category_AB_vs_CD  category_ABC_vs_D
##           -1.8554             0.2557            -0.2901             0.3589
##
## Degrees of Freedom: 999 Total (i.e. Null);  996 Residual
## Null Deviance:       855.8
## Residual Deviance: 852.6     AIC: 860.6
```

Log-odds of traget average per categorical variable

```
##         A         B         C         D
## -1.855351 -1.599634 -1.889740 -1.530876
```

Coefficient replicates

```
"(Intercept)" = lo(average["A"])
"category_A_vs_BCD" = lo(average["B"]) - lo(average["A"])
"category_AB_vs_CD" = lo(average["C"]) - lo(average["B"])
"category_ABC_vs_D" = lo(average["D"]) - lo(average["C"])
```

```
##       (Intercept) category_A_vs_BCD category_AB_vs_CD category_ABC_vs_D
##        -1.8553506         0.2557167        -0.2901060         0.3588643
```

Comparison of Two Approaches

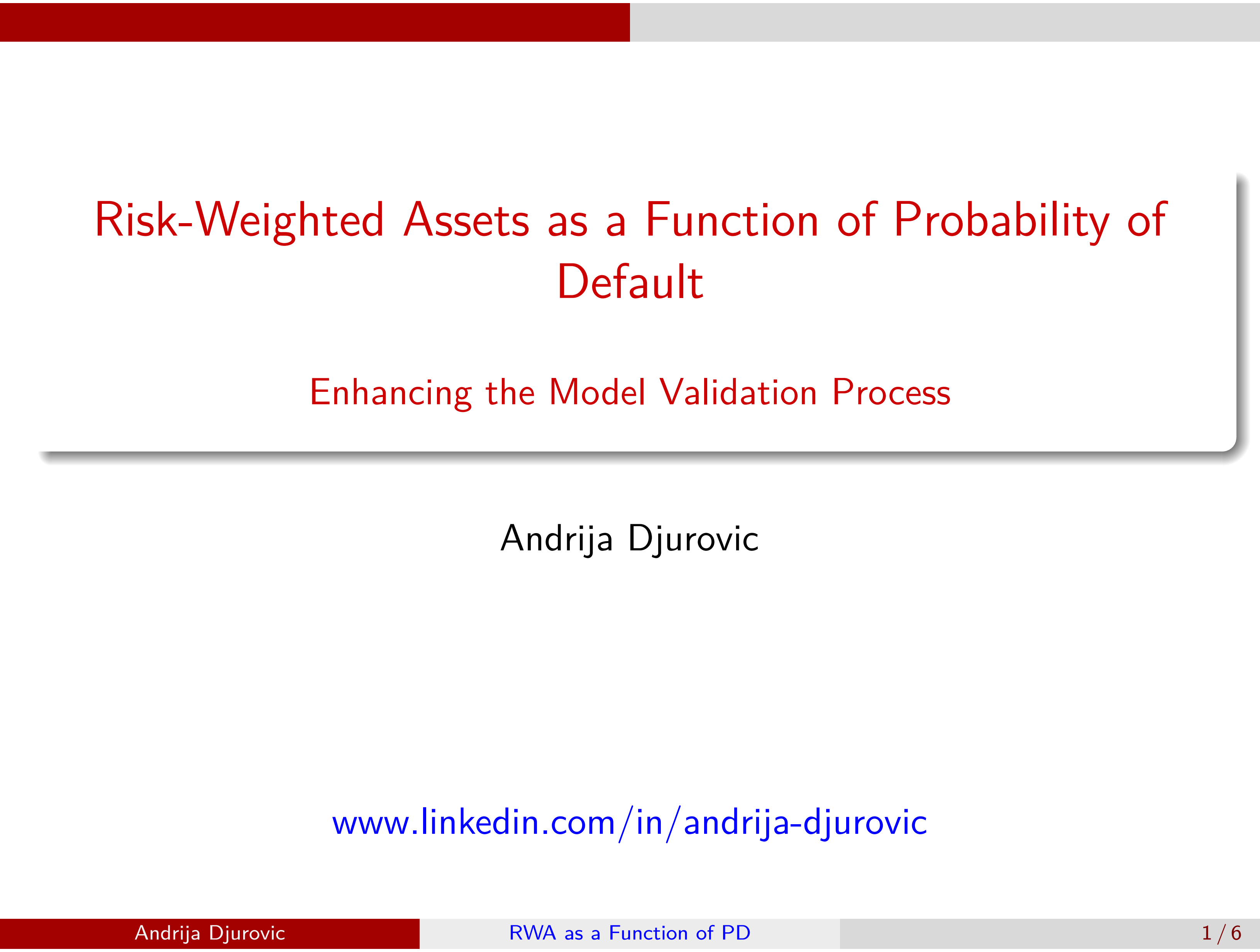
Test of Two Proportions ($H_1 : DR_{rating_{i+1}} > DR_{rating_i}$)

```
##         rating  no  nb     dr X.squared.stat  p.val
## 1 01 (-Inf,7)  82   9 0.1098             NA     NA
## 2   02 [7,16) 349  80 0.2292         5.7839 0.0081
## 3  03 [16,26) 339 109 0.3215         7.3540 0.0033
## 4  04 [26,33)  57  19 0.3333         0.0311 0.4301
## 5  05 [33,47) 108  47 0.4352         1.6127 0.1021
## 6 06 [47,Inf)  65  36 0.5538         2.2892 0.0651
```

Model-Based Testing ($H_1 : DR_{rating_{i+1}} \neq DR_{rating_i}$)

```
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -2.0932     0.3533 -5.9252   0.0000
## `02 [7,16)`       0.8806     0.3755  2.3448   0.0190
## `03 [16,26)`      0.4660     0.1725  2.7019   0.0069
## `04 [26,33)`      0.0536     0.3041  0.1762   0.8601
## `05 [33,47)`      0.4324     0.3415  1.2663   0.2054
## `06 [47,Inf)`     0.4769     0.3161  1.5088   0.1314
```

9.11 RWA as a Function of PD - Enhancing the Model Validation Process



Risk-Weighted Assets as a Function of Probability of Default

Enhancing the Model Validation Process

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | RWA as a Function of PD | 1 / 6

Risk-Weighted Assets in IRB Modeling

- Risk-Weighted Assets (RWA) is a measure used to determine the minimum capital required for banks to absorb potential losses.
- Under the Internal Ratings-Based (IRB) approach, RWA is calculated based on a bank's own estimates of risk parameters, including Probability of Default (PD), Loss Given Default (LGD), and Exposure at Default (EAD).
- In the IRB formulas, PD directly impacts the calculated RWA.
- When PD is underestimated, RWA may also be lower than necessary, potentially leading to insufficient capital to cover unexpected losses.
- If PD increases (or is recalculated to a more accurate, higher value), the corresponding RWA will increase, affecting the capital requirements.
- Beyond the standard use of RWA formulas for determining minimum capital requirements, RWA can support model validation by highlighting the impact of potential under- or overestimation of risk parameters.
- The following slides present a simplified simulation design illustrating the change in RWA for a specific increase in PD. Though simplified, this simulation provides a solid foundation for further adjustments. Practitioners are encouraged to adapt the design to meet specific needs.

Simulation Design

Exposure Type and RWA Formula

Residential mortgage exposures:

$$R = 0.15$$

$$K = LGD \cdot N\left[\frac{G(PD)}{\sqrt{1-R}} + \sqrt{\frac{R}{1-R}} \cdot G(0.999)\right] - PD \cdot \mathrm{LGD}$$

$$RWA = K \cdot 12.5 \cdot EAD$$

where:

- R denotes the asset correlation;
- K is capital requirement;
- G is the quantile function of the standard normal distribution;
- PD, LGD, EAD denote risk parameters.

Dataset

```
##  Rating     PD  LGD EAD
##   RG_01 0.0003 0.45 100
##   RG_02 0.0005 0.45 100
##   RG_03 0.0010 0.45 100
##   RG_04 0.0025 0.45 100
##   RG_05 0.0040 0.45 100
##   RG_06 0.0050 0.45 100
##   RG_07 0.0075 0.45 100
##   RG_08 0.0100 0.45 100
##   RG_09 0.0130 0.45 100
##   RG_10 0.0150 0.45 100
##   RG_11 0.0200 0.45 100
##   RG_12 0.0250 0.45 100
##   RG_13 0.0300 0.45 100
##   RG_14 0.0400 0.45 100
##   RG_15 0.0500 0.45 100
##   RG_16 0.0600 0.45 100
##   RG_17 0.1000 0.45 100
##   RG_18 0.1500 0.45 100
##   RG_19 0.2000 0.45 100
```

Simulation Design cont.

1. Using the data inputs and formulas provided on the previous slide, calculate the initial RWA value ($\text{RWA}_{initial}$).
2. Define the PD multipliers as a range of values between 1.01 and 4.
3. Apply each multiplier in the defined range to increase the initial PDs.
4. For each simulated PD, calculate the new RWA.
5. For each new RWA, calculate the RWA change as follows:

$$\Delta\text{RWA} = \frac{\text{RWA}_{simulated} - \text{RWA}_{initial}}{\text{RWA}_{initial}}$$

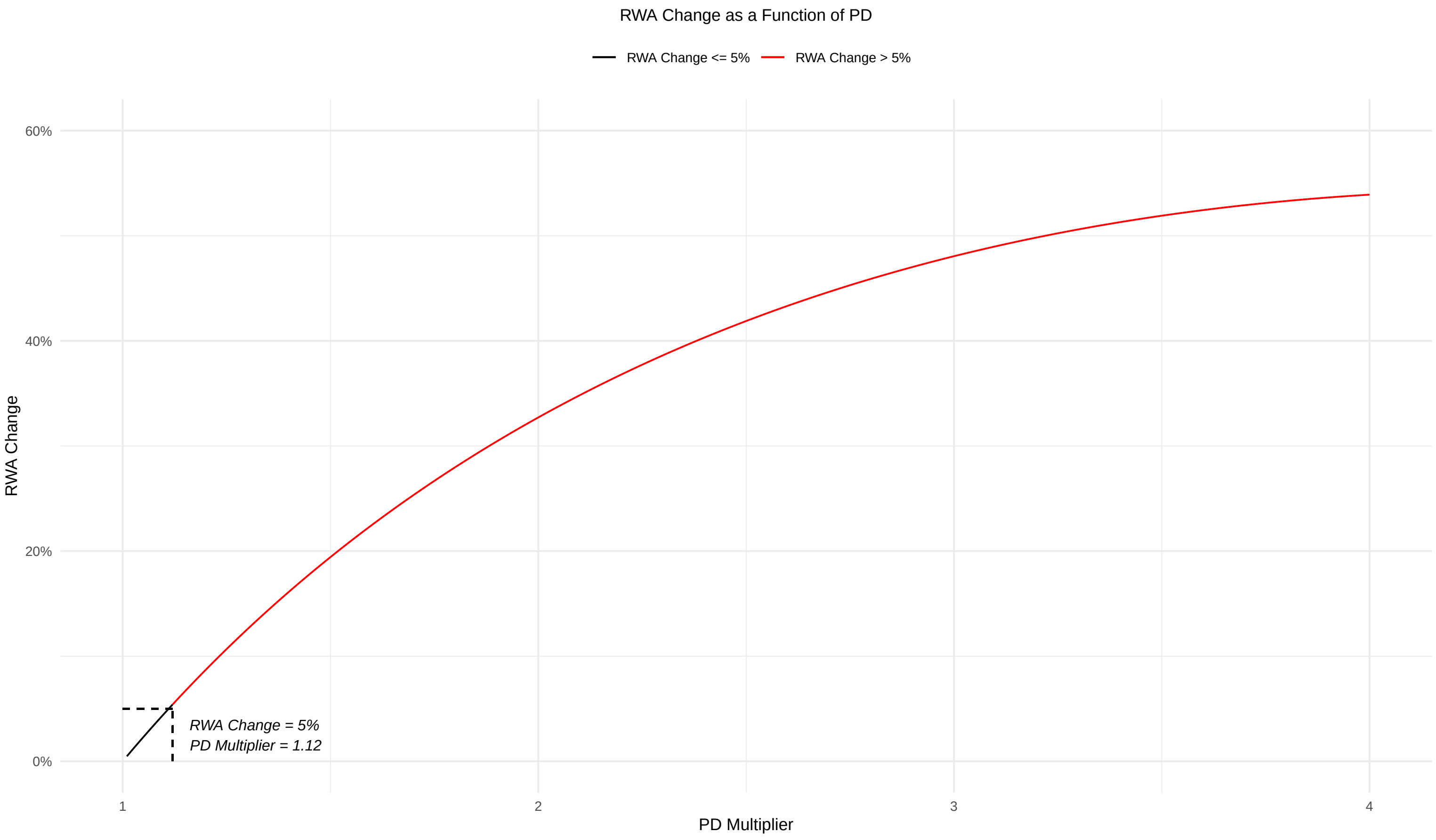
6. Plot the RWA change as a function of PD.

This design allows practitioners to assess the effect of different PD levels on RWA changes. By defining acceptable RWA change thresholds, practitioners can enhance the model validation process, ultimately supporting the final decision on validation outcomes.

Simulation Results

```
##   Rating     PD  LGD EAD RWA_initial
##    RG_01 0.0003 0.45 100      4.1492
##    RG_02 0.0005 0.45 100      6.2302
##    RG_03 0.0010 0.45 100     10.6896
##    RG_04 0.0025 0.45 100     21.2975
##    RG_05 0.0040 0.45 100     29.9447
##    RG_06 0.0050 0.45 100     35.0792
##    RG_07 0.0075 0.45 100     46.4635
##    RG_08 0.0100 0.45 100     56.3989
##    RG_09 0.0130 0.45 100     66.9950
##    RG_10 0.0150 0.45 100     73.4441
##    RG_11 0.0200 0.45 100     87.9350
##    RG_12 0.0250 0.45 100    100.6391
##    RG_13 0.0300 0.45 100    111.9876
##    RG_14 0.0400 0.45 100    131.6309
##    RG_15 0.0500 0.45 100    148.2221
##    RG_16 0.0600 0.45 100    162.5188
##    RG_17 0.1000 0.45 100    204.4105
##    RG_18 0.1500 0.45 100    235.7225
##    RG_19 0.2000 0.45 100    253.1188
```

Simulation Results cont.



9.12 Statistical Approach to Third Party Ratings Treatment - Modeling Third Party Ratings Adjustment

Statistical Approach to Third Party Ratings Treatment

Modeling Third Party Ratings Adjustment

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic Modeling Third Party Ratings 1 / 8

Third Party Support in Credit Risk

- The ownership structure of a corporate entity significantly influences its creditworthiness.
- Third party support can strengthen a company's credit profile if the supporting party is financially stable and willing to assist during times of stress.
- Conversely, if the third party has a weaker credit profile and relies on the subsidiary's cash flows to meet its obligations, its risk profile may deteriorate.
- Third party support can take various forms and be shaped by multiple factors.
- When assessing its impact on rating assignments, practitioners typically evaluate, among other factors, the timeliness and adequacy of this support in preventing default.
- Regulatory frameworks recognize the role of third party support and guide the incorporation of third party ratings into credit risk assessments.

Regulatory Framework for Third Party Ratings

- Regulatory guidance outlines three main, non-mutually exclusive approaches to incorporating third party ratings into Probability of Default (PD) models:

  1. Rating Transfer (Explicit Guarantees);
  2. Rating-Based Overrides;
  3. Third Party Rating as a Risk Driver.

- The use of third party ratings is not limited to statistical models; they can also influence non-statistical decision-making.

- If third party ratings are not directly incorporated into the PD model, they may still justify overrides, provided institutions ensure no double counting.

- Among other regulatory documents, the Guidelines on PD Estimation, LGD Estimation, and the Treatment of Defaulted Exposures and the ECB Guide to Internal Models provide more details on the treatment of third party ratings.

Statistical Approach to Third Party Ratings Treatment

- Unlike other standardized statistical approaches for model development, practitioners must consider additional inputs and modeling constraints when treating third party rating modeling.
- Given the variety of supporting structures between the third party and subsidiary and data availability, developing a modeling framework that suits all situations is challenging. Therefore, practitioners often opt for tailor-made modeling approaches when tackling this exercise.
- The following slide outlines one of the approaches proposed by Andrija Djurovic, which can serve as a basis for modeling third party support.
- Like other approaches, this relies on specific assumptions, such as the availability and comparability of scores between the third party and subsidiary. It also assumes that the new score always lies between the third party and subsidiary scores.
- The last slide highlights additional points, potential improvements, and adjustments to this framework, complementing the presented framework and simulation study.

Statistical Approach to Third Party Ratings Treatment cont.

The following points outline the steps for a statistical approach to third party rating modeling under the assumptions mentioned on the previous slide:

1. With both subsidiary (ss_{score}) and the third party (gr_{score}) scores available, estimate a logistic regression model of the form:
$$y \sim \alpha + \beta \cdot \text{Score}$$
where y is the default indicator, and α and β are the regression estimates.
2. Using the results from step 1, compute the log odds separately for the subsidiary ($ss.lo$) and the third party ($gr.lo$).
3. Following the idea of threshold models, create two variables that reflect cases where the subsidiary score is lower than the third party score - n - and cases where the subsidiary score is greater than or equal to the third party score - p. The variables n and p are defined as the absolute difference between the log odds obtained in step 1.
4. To estimate potential improvements in the subsidiary score given the third party score, fit the following constrained logistic regression:
$$y \sim \beta_1 \cdot n + \beta_2 \cdot p$$
while setting the offset equal to the subsidiary log-odds ($ss.lo$) and imposing the constraints $-1 < \beta_1 < 0$ and $0 < \beta_2 < 1$. By applying these constraints along with the offsets, the predicted probability always falls between those of the subsidiary and the third party.

Practitioners should remember that, depending on the relationship between the initial scores and the default indicator (y), the constraints in step 4 may need to be reversed.

The following slides present the results of a simulation study conducted on the dataset available here.

Simulation Results

The following points present the main results and output of the simulation study, while the graph on the next slide illustrates the log odds distribution for the subsidiary, third party, and combined log odds from the final constrained model.

1. Initially estimated models:

```
##             Estimate Std. Error  z value Pr(>|z|)
## (Intercept)   2.0341     0.3496   5.8179        0
## ss           -0.0059     0.0005 -12.9350        0
```

```
##             Estimate Std. Error  z value Pr(>|z|)
## (Intercept)   2.6401     0.3780   6.9842        0
## gr           -0.0066     0.0005 -13.3741        0
```

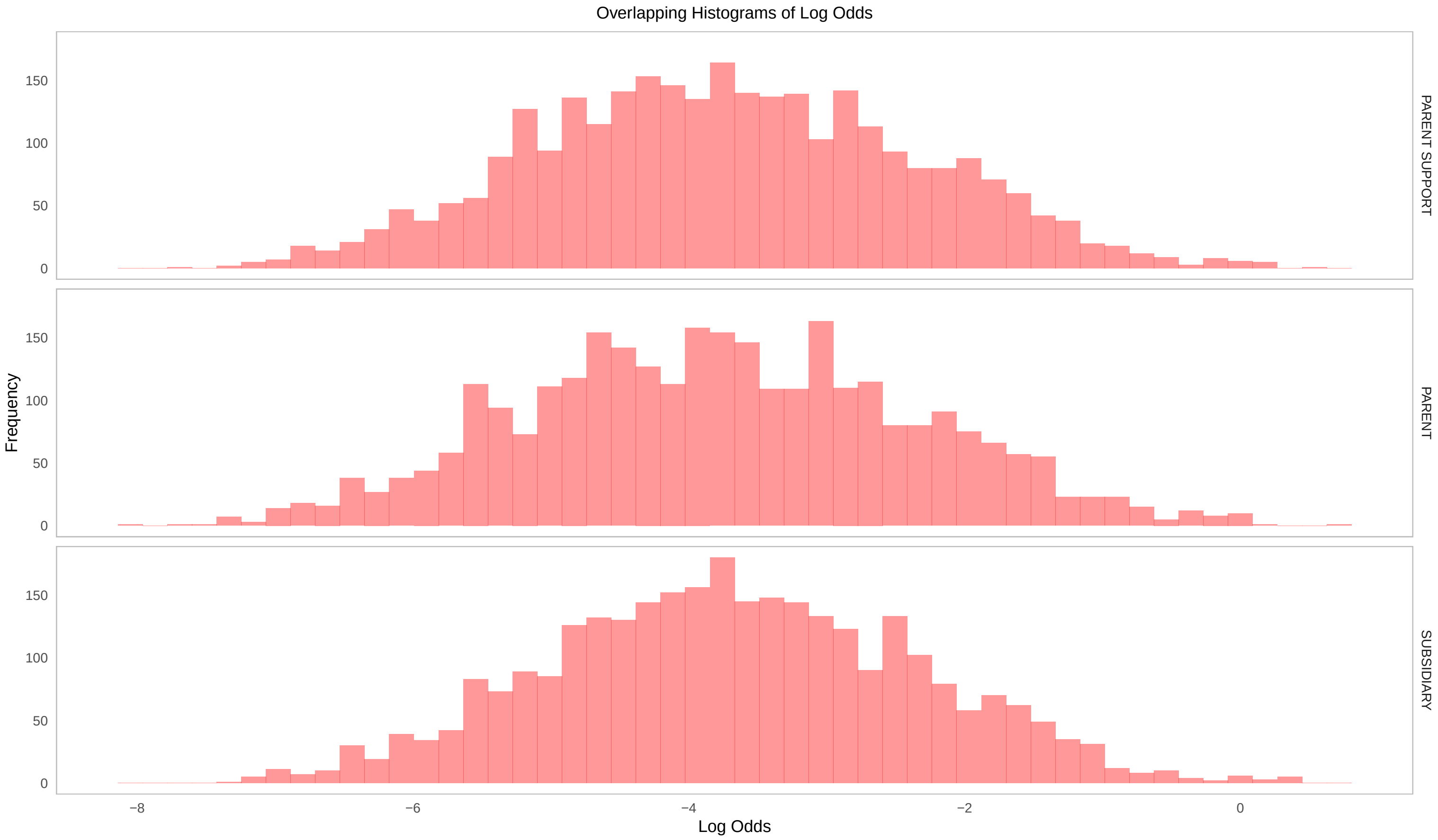
2. Sample of derived variables *n* and *p*, reflecting different relationships between subsidiary and the third party scores:

```
##        n      p
## 1 0.6543 0.0000
## 2 0.5928 0.0000
## 3 0.0000 0.6431
## 4 0.0000 0.4340
## 5 0.6648 0.0000
## 6 0.7022 0.0000
```

3. Results of the constrained logistic regression with an offset:

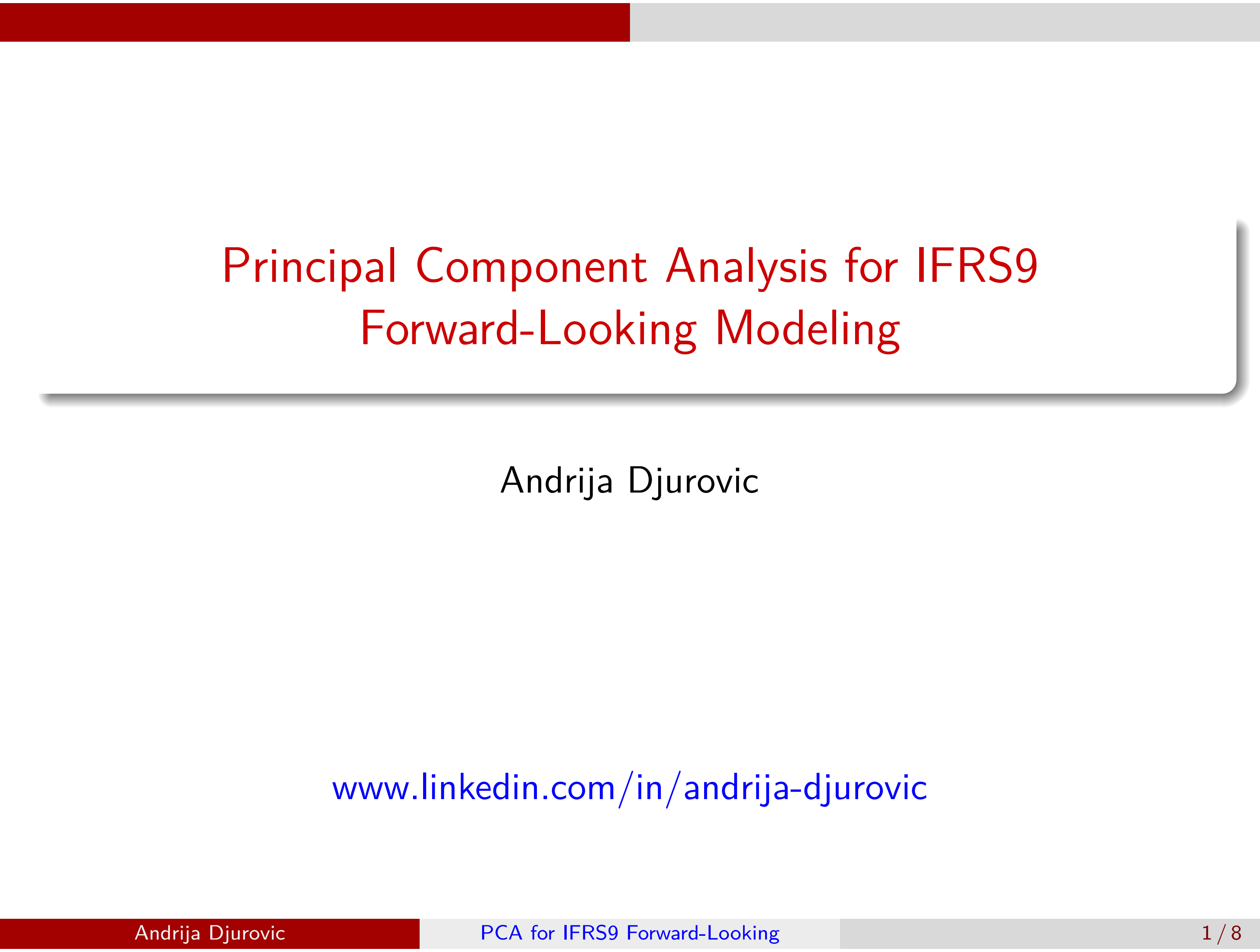
```
##       n      p
## -0.6962  0.7641
```

Simulation Results cont.



Discussion Points

- How do the source and comparability of subsidiary and third party scores affect the statistical method?
- How does the discriminatory power of individual scores (subsidiary and third party) impact the final estimates and conclusions?
- How can practitioners incorporate expert-based inputs, such as different categories of third party support, into the estimation process?
- How can practitioners adjust the proposed method if third party support is expressed as a combination of different variables (continuous and categorical)?



Principal Component Analysis for IFRS9 Forward-Looking Modeling

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic PCA for IFRS9 Forward-Looking 1 / 8

Principal Component Analysis

- Principal Component Analysis (PCA) enables practitioners to efficiently capture essential patterns and variability in the data by transforming the original variables into a smaller set of uncorrelated principal components.
- After reducing the data dimensionality, the selected principal components typically serve as inputs for the regression model.
- In the context of IFRS9 forward-looking modeling, PCA is a notable approach practitioners employ.
- PCA addresses a significant challenge in forward-looking modeling exercises: the relatively low ratio between the number of observations and the number of independent variables.
- Despite its popularity, is PCA always the optimal solution?
- Practitioners should avoid unthinkingly applying PCA for IFRS9 forward-looking modeling.
- Special consideration is essential for evaluating the observed directional relationship between the variables comprising the selected PCA and the target variable and comparing it to the anticipated directional relationship.

Case Study - Setup

1. The IFRS9 PD modeling dataset comprises four variables collected with year-on-year change: Observed Default Rate (`odr`), Gross Domestic Product (`gdp`), Unemployment (`unemployment`), and Wages (`wage`).
2. The observed default rate is the target variable, while the other variables are considered independent.
3. The dataset is structured quarterly, covering the most recent 50 quarters of the observed indicators.
4. The expected directional relationship between the independent variables and the target is assumed to be negative for `gdp` and `wage` while positive for `unemployment`. A negative directional relationship implies that as the independent variable increases, the target decreases, and vice versa. The positive directional relationship implies the same directional change between the independent and dependent variable.
5. Suppose we set a threshold of 20 observations per independent variable when selecting the maximum number of variables for the regression model. This limitation would allow us to utilize only two independent variables.
6. To incorporate all three independent variables in the model, we will transform them using PCA and then use only the first principal component as an input for the OLS regression.
7. The case study's ultimate goal is to evaluate the alignment between the anticipated and observed directional relationships among the raw independent variables after running the PCA and the target variable.

Case Study - R Code

```r
#data import
url <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/model_dev_and_vld/pca.csv"
db <- read.csv(file = url,
               header = TRUE)
#expected directional relationship
dr.e <- c("gdp" = "-", "unemployment" = "+", "wage" = "-")
#pca
pca.res <- prcomp(x = db[, -1],
                  scale = TRUE)
#pca rotation
pca.res$rotation
```

```
##                     PC1         PC2        PC3
## gdp          -0.6057804  0.49897450 -0.6197213
## unemployment -0.4113107 -0.86314827 -0.2929139
## wage         -0.6810680  0.07745652  0.7281119
```

```r
#extract pc1
db$pca.1 <- pca.res$x[, "PC1"]
```

Case Study - R Code cont.

```
#ols - principal component
ols.p <- lm(formula = odr ~ pca.1,
            data = db)
ols.p
```

```
##
## Call:
## lm(formula = odr ~ pca.1, data = db)
##
## Coefficients:
## (Intercept)        pca.1
##    -0.02026      0.19604
```

```
#observed directional relationship
dr.o <- sign(coef(ols.p)[2]*pca.res$rotation[, "PC1"])
dr.o <- ifelse(dr.o == 1, "+", "-")

#compare the expected and observed directional relationship
dr.c <- data.frame(EXPECTED = dr.e,
                   OBSERVED = dr.o)
dr.c
```

```
##                EXPECTED OBSERVED
## gdp                   -        -
## unemployment          +        -
## wage                  -        -
```

Case Study - Python Code

```python
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
import statsmodels.formula.api as smf

#data import
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/model_dev_and_vld/pca.csv"
db = pd.read_csv(filepath_or_buffer = url)

#expected directional relationship
dr_e = {"gdp": "-", "unemployment": "+", "wage": "-"}
#standardize pca inputs
mv = ["gdp", "unemployment", "wage"]
db[mv] = (db[mv] - np.mean(db[mv], axis = 0)) / np.std(db[mv], axis = 0, ddof = 1)
#pca
pca = PCA(svd_solver = "full")
pca_res = pca.fit_transform(X = db.iloc[:, 1:])
#pca rotations
pca.components_
```

```
## array([[-0.60578043, -0.41131073, -0.68106795],
##        [-0.4989745 ,  0.86314827, -0.07745652],
##        [-0.61972132, -0.2929139 ,  0.7281119 ]])
```

```python
#extract pc1
db["pca_1"] = pca_res[:, 0]
```

Case Study - Python Code cont.

```
#ols - principal component
ols_p = smf.ols(formula = "odr ~ pca_1",
                data = db).fit()
ols_p.params
```

```
## Intercept   -0.020262
## pca_1        0.196038
## dtype: float64
```

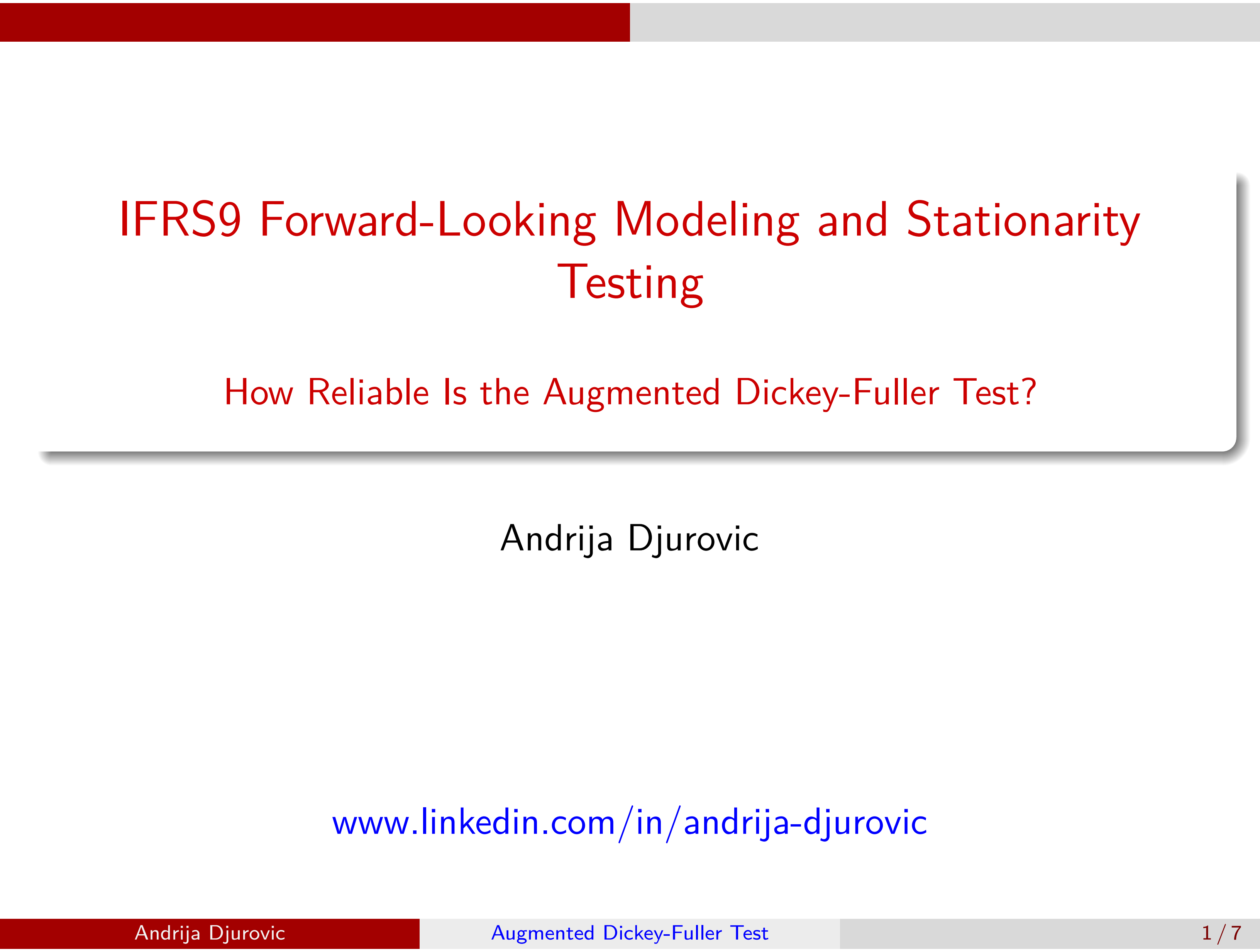
```
#observed directional relationship
dr_o = np.sign(ols_p.params["pca_1"]* pca.components_[0])
dr_o = np.where(dr_o == 1, "+", "-")

#compare the expected and observed directional relationship
dr_c = pd.DataFrame({"EXPECTED": dr_e.values(),
                     "OBSERVED": dr_o},
                     index = dr_e.keys())
dr_c
```

```
##                EXPECTED OBSERVED
## gdp                   -        -
## unemployment          +        -
## wage                  -        -
```

Discussion Points

- What adjustments are necessary when employing more than one principal component?
- How does the integration of time lags for independent variables add complexity to the PCA procedure?
- What are the alternatives to the PCA to address the same challenge in the IFRS9 forward-looking modeling?
- How can PCA be applied in other credit risk domains, such as IRB model development or scorecard modeling?

9.14 IFRS9 Forward-Looking Modeling and Stationarity Testing - How Reliable Is the Augmented Dickey-Fuller Test?



IFRS9 Forward-Looking Modeling and Stationarity Testing

How Reliable Is the Augmented Dickey-Fuller Test?

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Augmented Dickey-Fuller Test | 1 / 7

IFRS9 Forward-Looking Modeling

- With the introduction of IFRS9, one key requirement practitioners had to address was the inclusion of reasonable and supportable information that is available without undue cost or effort when measuring expected credit losses.
- Several regulatory paragraphs also emphasized incorporating such information when recognizing lifetime expected credit losses.
- After several years of implementation, the most common approach observed in practice quantifies the macroeconomic environment's effect on a bank's risk parameters, usually aggregated at the segment level. For example, practitioners calculate the default rate at the segment level and regress it against macroeconomic indicators.
- Many models rely on Ordinary Least Squares (OLS) regression but differ in their design and level of aggregation. As a result, one aspect practitioners inevitably investigate is stationarity.
- Stationarity testing is typically performed either at the level of the variables used in Forward-Looking (FLI) modeling or on the model residuals under the assumption of cointegration.
- One of the most commonly used tests for stationarity is the Augmented Dickey-Fuller (ADF) test.
- Although practitioners often rely solely on statistical tests, the power of these tests is heavily influenced by the characteristics of the FLI context - most notably, the relatively low data-to-variable ratio, which presents significant challenges not only for stationarity testing but also for other critical modeling tasks.

Augmented Dickey-Fuller Test

The Augmented Dickey-Fuller test is one of the most commonly used stationarity tests in FLI modeling.

The general form of the ADF regression is:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^{p} \delta_i \Delta y_{t-i} + \varepsilon_t$$

where:

- Δy_t is the first difference of the series of the variable being analyzed;
- α is a constant (drift term);
- t is a deterministic time trend;
- β is the coefficient on a time trend;
- y_{t-1} is the lagged level of the variable;
- γ is the coefficient on y_{t-1} and the core element of the test;
- p is the number of lagged differences included in the regression;
- δ_i are the coefficients on the lagged differences;
- ε_t is the white noise error term.

Augmented Dickey-Fuller Test cont.

- The null hypothesis of the ADF test ($H_0 : \gamma = 0$) implies non-stationarity, while the alternative ($H_1 : \gamma < 0$) indicates stationarity.
- Significance is assessed using critical values from the Dickey-Fuller distribution and comparing the interpolated p-value to the chosen significance level.
- Depending on the properties of the variable being tested, the model may include a constant and/or a trend.
- The number of lags is typically selected based on the time series length or using information criteria such as AIC or BIC.
- Given the limited number of observations in FLI modeling, the choice of ADF specification can significantly influence the test outcome.
- The following slides present a framework for evaluating the power of the ADF test under specific model configurations. Practitioners are encouraged to use it as a general reference and adapt it to their specific context, as conclusions may vary with different simulation inputs.

Simulation Study

This simulation aims to investigate the ADF test's statistical power under a specific model configuration. The following steps describe the simulation design:

1. Select the order of the autoregressive process ($\phi = 0.75$).
2. Select the sample size n (20, 25, 30, 35, 40, 45, 50, 55, 60, 100, 250).
3. Simulate an autoregressive process of order one for the chosen values of ϕ and n:

   $$x_{t,t \leq n} = \phi x_{t-1} + \sqrt{1 - \phi^2} \varepsilon_t$$

   where ε_t is drawn from the standard normal distribution.
4. Apply the ADF test to the simulated values of x from step 3.
5. Repeat steps 3 and 4 for $N = 10, !000$ simulations, storing the estimation results.
6. Calculate the power of the ADF test as the proportion of simulations that reject the null hypothesis of non-stationarity at the 5% significance level.

Note:

The simulation is implemented in `R` using the functions `adf.test` and `ur.df` from the `tseries` (version 0.10-55) and `urca` (version 1.3-4) packages. Both functions use a fixed lag order of 1.

The `adf.test` function includes a drift and a trend by default, while the `ur.df` function excludes both in this simulation design.

Simulation Results

The following tables present the simulation results for different sample sizes and ADF test specifications.

ADF Specification: With Constant and Trend

```
##    Sample Size   Power
## 1           20   9.02%
## 2           25   9.78%
## 3           30  11.05%
## 4           35  12.58%
## 5           40  15.81%
## 6           45  19.37%
## 7           50  23.14%
## 8           55  26.43%
## 9           60  33.06%
## 10         100  74.77%
## 11         250 100.00%
```

ADF Specification: Without Constant and Trend

```
##    Sample Size   Power
## 1           20  32.10%
## 2           25  40.63%
## 3           30  50.82%
## 4           35  60.46%
## 5           40  71.13%
## 6           45  77.97%
## 7           50  83.70%
## 8           55  88.84%
## 9           60  92.81%
## 10         100  99.92%
## 11         250 100.00%
```

Conclusions

- Model design is a critical success factor in IFRS9 FLI modeling, especially given limited data availability.
- Overreliance on statistical methods without integrating expert input can compromise model quality.
- Cointegration and stationarity should not be treated as purely statistical criteria in IFRS9 FLI modeling.
- The modeling objective must guide the balance between statistical rigor and business relevance.
- Statistical diagnostics (e.g., p-values, autocorrelation, spurious regression) should be interpreted within the context of the modeling purpose.

9.15 IFRS9 Forward-Looking Modeling - OLS Regression and Predictor Importance

IFRS9 Forward-Looking Modeling

OLS Regression and Predictor Importance

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Predictor Importance | 1 / 11

IFRS9 Forward-Looking Modeling

- With the introduction of IFRS9, one key requirement practitioners had to address was the inclusion of reasonable and supportable information available without undue cost or effort when measuring expected credit losses.
- Several regulatory paragraphs also emphasized incorporating such information when recognizing lifetime expected credit losses.
- After several years of implementation, the most common approach observed in practice quantifies the macroeconomic environment's effect on a bank's risk parameters, usually aggregated at the segment level. For example, practitioners calculate the default rate at the segment level and regress it against macroeconomic indicators.
- Many models rely on Ordinary Least Squares (OLS) regression but differ in their design and level of aggregation.
- Given the regression design, when analyzing forward-looking (FLI) models, practitioners often assess the importance of each predictor in the final model.
- In doing so, the most commonly used approach in practice is to assess predictor importance using standardized regression coefficients. This raises the question of whether this is the only meaningful way to interpret a predictor's importance, and what other approaches may offer additional insights.

OLS Regression and Predictor Importance

According to Achen (1982), in his book Interpreting and Using Regression, the concept of variable importance is ambiguous without clarifying: important for what purpose?
He classifies importance into three groups, each answering a different question:

1. Theoretical Importance
2. Level Importance
3. Dispersion Importance

The following slides briefly explain each type of importance and illustrate their calculations using R and Python.
The data used for these examples can be imported using the code below.

R Code:

```r
url <- "http://andrija-djurovic.github.io/adsfcr/model_dev_and_vld/db_pi.csv"
db <- read.csv(url)
```

Python Code:

```python
import pandas as pd

url = "http://andrija-djurovic.github.io/adsfcr/model_dev_and_vld/db_pi.csv"
db = pd.read_csv(url)
```

Theoretical Importance

Conceptual Overview:

- Theoretical importance measures the potential effect of a variable, holding all else constant.
- It is captured by the unstandardized coefficient (β) in the regression and indicates how much the dependent variable changes per unit increase in the predictor.
- The interpretation of this type of importance is not sample-dependent.
- Meaningful comparisons between predictors can only be made if they are on the same scale.
- For IFRS9 FLI modeling, it helps answer questions such as: “If GDP increases by one percentage point, how much would we expect the default rate to change?”
- In this context, practitioners often refer to this importance as sensitivity.

R Code:

```r
#ols regression
ols <- lm(formula = odr ~ gdp + unemployment,
          data = db)
#theoretical importance
coef(ols)[-1]
```

```
##          gdp unemployment
##   -0.2586861    0.1682093
```

Theoretical Importance cont.

Python Code:

```
import numpy as np
import statsmodels.formula.api as smf

#ols regression
ols = smf.ols(formula = "odr ~ gdp + unemployment",
              data = db).fit()
#theoretical importance
ols.params.drop("Intercept")
```

```
## gdp            -0.258686
## unemployment    0.168209
## dtype: float64
```

Level Importance

Conceptual Overview:

- Level importance measures a predictor's actual contribution to the mean level of the dependent variable in a given sample.
- It is computed as the product of the regression coefficient (β) and the mean of the predictor.
- It is sample-dependent and reflects historical influence, not abstract potential.
- Although it is rarely used in practice in IFRS9 FLI modeling, it helps answer important questions such as: "How much of the observed average default rate in the past is explained by average unemployment?"
- Despite reflecting historical influence, it plays a valuable role in this context by clarifying the contribution of predictors to the forecasted value, thereby supporting the assessment of forecast validity and the impact of predictors.

R Code:

```r
#ols regression
ols <- lm(formula = odr ~ gdp + unemployment,
          data = db)
#predictor coefficients
coef.p <- coef(ols)[-1]
#gdp mean
gdp.m <- mean(db$gdp)
#unemployment mean
unempl.m <- mean(db$unemployment)
```

Level Importance cont.

R Code cont:

```
#level importance
coef.p * c(gdp.m, unempl.m)
```

```
##           gdp  unemployment
## -0.0037307972  0.0002566554
```

Python Code:

```
import numpy as np
import statsmodels.formula.api as smf

#ols regression
ols = smf.ols(formula = "odr ~ gdp + unemployment",
              data = db).fit()
#predictor coefficients
coef_p = ols.params.drop("Intercept")
#gdp mean
gdp_m = db["gdp"].mean()
#unemployment mean
unempl_m = db["unemployment"].mean()
#level importance
coef_p * np.array([gdp_m, unempl_m])
```

```
## gdp            -0.003731
## unemployment    0.000257
## dtype: float64
```

Dispersion Importance

Conceptual Overview:

- Dispersion importance measures how much variation in the dependent variable is explained by a predictor.
- It is captured by the standardized coefficient.
- It shows only how much of the spread is explained rather than indicating the effect of a one-unit change or the average contribution.
- In the context of IFRS9, it is the most commonly used measure and helps answer questions such as: "How much of the spread of the dependent variable is explained by each predictor in a given sample?"
- The following code demonstrates two different ways to calculate standardized regression coefficients. Method 1 applies OLS regression to a standardized dataset, while Method 2 transforms unstandardized regression coefficients into standardized ones.

R Code for Method 1:

```r
#ols regression on standardized variables
ols.s <- lm(formula = scale(odr) ~ scale(gdp) + scale(unemployment),
            data = db)
#dispersion importance
coef(ols.s)[-1]
```

```
##          scale(gdp) scale(unemployment)
##          -0.8344894           0.2612825
```

Dispersion Importance cont.

R Code for Method 2:

```r
#ols regression
ols <- lm(formula = odr ~ gdp + unemployment,
          data = db)
#unstandardized coefficients
coef.p <- coef(ols)[-1]
#standard deviation of odr
sd.odr <- sd(db$odr)
#standard deviation of predictors
sd.p <- sapply(X = db[, c("gdp", "unemployment")],
               FUN = sd)
#dispersion importance
coef.p * sd.p / sd.odr
```

```
##          gdp unemployment
##   -0.8344894    0.2612825
```

Dispersion Importance cont.

Python Code for Method 1:

```python
import numpy as np
import statsmodels.formula.api as smf

#variable standardization
db["odr_z"] = (db["odr"] - db["odr"].mean()) / db["odr"].std()
db["gdp_z"] = (db["gdp"] - db["gdp"].mean()) / db["gdp"].std()
db["unemployment_z"] = (db["unemployment"] - db["unemployment"].mean()) / \
                        db["unemployment"].std()
#ols regression on standardized variables
ols_s = smf.ols(formula = "odr_z ~ gdp_z + unemployment_z",
                data = db).fit()
#dispersion importance
ols_s.params.drop("Intercept")
```

```
## gdp_z            -0.834489
## unemployment_z    0.261282
## dtype: float64
```

Dispersion Importance cont.

Python Code for Method 2:

```
#ols regression
ols = smf.ols(formula = "odr ~ gdp + unemployment",
              data = db).fit()
#ustandardized coefficients
coef_p = ols.params.drop("Intercept")
#standard deviation of odr
sd_odr = db["odr"].std()
#standard deviation of predictors
sd_p = db[["gdp", "unemployment"]].std()
#dispersion importance
coef_p * sd_p / sd_odr
```

```
## gdp              -0.834489
## unemployment      0.261282
## dtype: float64
```

9.16 IFRS9 Forward-Looking Modeling - Do We Use OLS Regression Efficiently?

IFRS9 Forward-Looking Modeling

Do We Use OLS Regression Efficiently?

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic IFRS9 FLI and OLS Regression 1 / 6

IFRS9 Regulatory Requirement and Practical Implementation

- With the introduction of IFRS9, one of the requirements practitioners had to address was the inclusion of reasonable and supportable information that is available without undue cost or effort when measuring expected credit losses.
- Also, related to measuring expected credit losses, several regulatory paragraphs emphasized the importance of incorporating this reasonable and supportable information in recognizing lifetime expected credit losses.
- After several years of implementing IFRS9, the most common approach observed in practice quantifies the macroeconomic environment's effect on a bank's risk parameters, usually aggregated at the segment level. For instance, practitioners calculate the default rate at the segment level and regress it against macroeconomic indicators.
- In addition to the choice of modeling level, various model designs appear in practice, with most of them using the Ordinary Least Squares (OLS) method as the estimation technique.

IFRS9 Regulatory Requirement and Practical Implementation cont.

- Among the most frequently used approaches, practitioners select:
  - OLS regression in the form of the target variable against macroeconomic indicators (with or without time lags);
  - OLS regression, including macroeconomic indicators and an autoregressive term of the target;
  - Two-step error correction models;
  - Principal Component Analysis (PCA) combined with OLS regression;
  - OLS regression with various transformations of the target variable, such as logit or probit.
- As seen above, this list of model designs is not exhaustive. In practice, various combinations of transformations and methods may be used - most of which rely on OLS as the estimation method. However, each design and its combinations present specific challenges when applied to IFRS9 forward-looking (FLI) modeling.
- In IFRS9 FLI modeling, model design plays a critical role - arguably even more so than in other credit risk exercises.
- Given the importance of model design, this presentation aims to challenge the efficient use of the OLS estimation method in the context of IFRS9 forward-looking modeling.

IFRS9 Forward-Looking Modeling Principles

When developing FLI models, practitioners usually adhere to certain principles. The following list outlines some of the most commonly applied ones:

- The model should convey a clear and coherent narrative.
- The model should capture all relevant historical events significantly impacting the target variable.
- The model should incorporate key events that are not reflected in macroeconomic forecasts.
- The model should balance the impact of the economy with other factors, such as internal changes or regulatory shifts.
- The statistical techniques used to develop the model should be straightforward and interpretable yet robust enough to capture key historical patterns and forward-looking assumptions.
- The relationship between model variables should remain reasonably stable over time.
- The model's forecasting performance must be tested on an out-of-time sample.
- Expert input should have a controlled influence on model outcomes to minimize potential bias.

The above principles often lead to specific challenges that practitioners encounter during modeling. While there isn't always a one-to-one correspondence between principles and challenges, they are closely related and can often be meaningfully linked. The following slide summarizes some of these key challenges and proposes possible approaches within the OLS regression framework.

IFRS9 Forward-Looking Modeling Challenges and OLS Regression

One of the most critical aspects of developing FLI models is model design. Too often, practitioners emphasize the statistical method rather than how the model is structured. Achieving the right balance between statistical rigor and a design integrating expert input (business insights) is crucial for developing a successful FLI model. The following list highlights some common challenges practitioners face during this process and potential enhancements to the standard OLS regression approach to help address them.

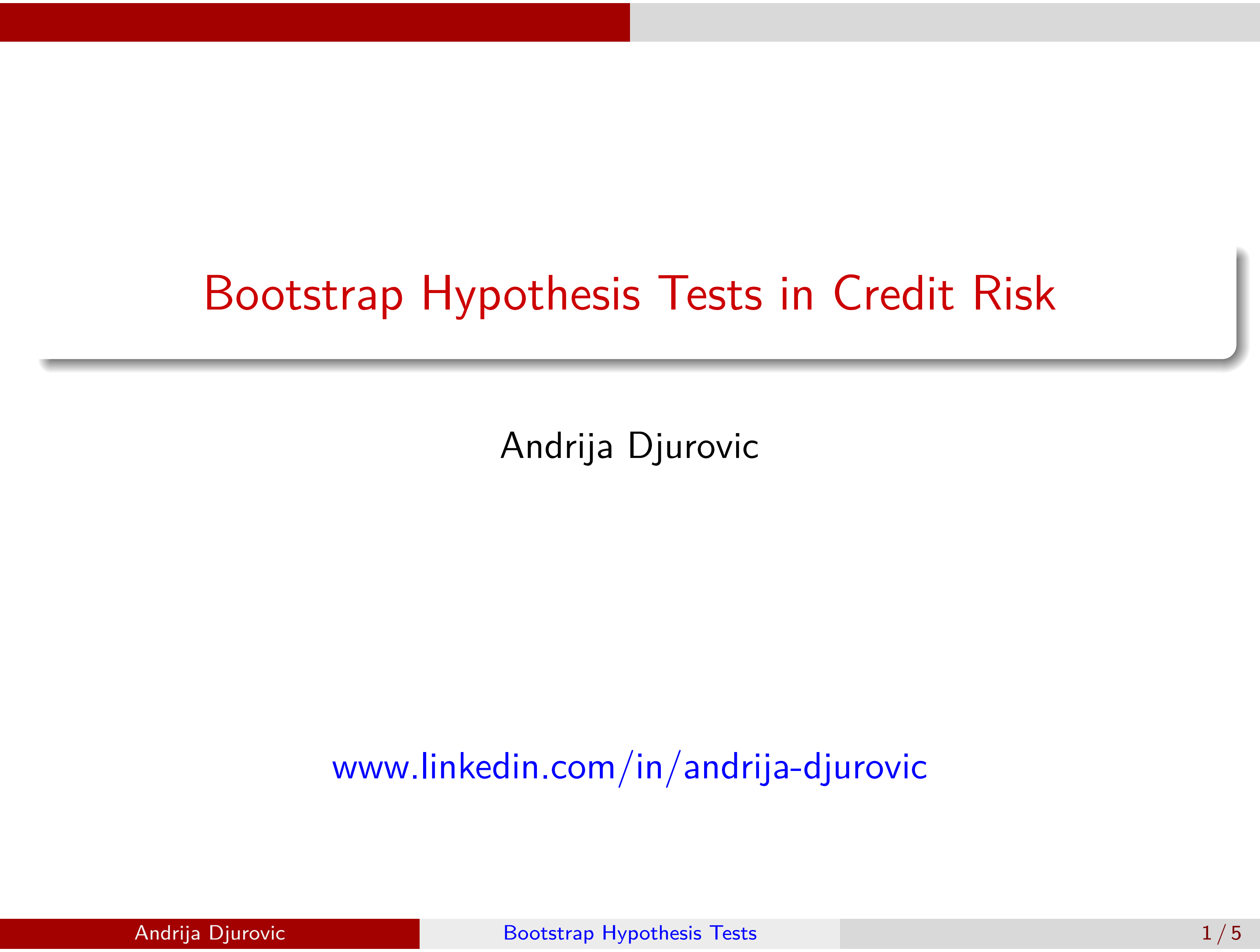
1. - Challenge: Low ratio between the number of observations and the available predictors.
   - Approach: OLS model averaging approach, blockwise model design.
2. - Challenge: Incorporating expert input regarding the expected macroeconomic indicators.
   - Approach: OLS model averaging approach, blockwise model design, PCA + OLS regression, supervised macroeconomic index, constrained OLS regression.

IFRS9 Forward-Looking Modeling Challenges and OLS Regression cont.

3.
   - Challenge: Asymmetric or opposing impacts of macroeconomic indicators under different economic conditions or when specific indicator thresholds are reached.
   - Approach: Constrained OLS regression, threshold OLS regression, supervised macroeconomic index.
4.
   - Challenge: Balancing the impact of macroeconomic indicators and other predictors.
   - Approach: Constrained OLS regression.
5.
   - Challenge: Ensuring a stable model and reliable forecasts.
   - Approach: Constrained OLS regression.
6.
   - Challenge: Meeting statistical criteria for model selection.
   - Approach: Simulation-based analysis of violations or limitations in OLS regression assumptions.

Practitioners should bear in mind that the above approaches are not mutually exclusive and are intended only as suggestions for addressing specific challenges. Furthermore, the list is not exhaustive; practitioners may encounter other challenges in real-world applications and adopt alternative solutions.

9.17 Bootstrap Hypothesis Tests in Credit Risk



Bootstrap Hypothesis Tests in Credit Risk

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Bootstrapping in Credit Risk

- Generally underutilized in the credit risk area.
- Bootstrapping finds application across various domains within credit risk modeling.
- Particularly useful for testing metrics where there is little consensus on the standard error of the estimate or when statistical testing procedures are absent.
- Various methods exist for calculating bootstrap confidence intervals and p-values, with the percentile method likely being the most commonly used. Computing the p-value is not always straightforward because bootstrapping does not produce data conforming to the null hypothesis.
- Advantages:
  - does not rely on any distributional assumption
  - flexible application across a wide range of metrics
  - useful for identifying bias
  - easy to implement.
- Disadvantages:
  - computational intensity
  - assumption of independence (although this can be addressed to a certain extent)
  - accuracy concerns for smaller sample sizes.
- The examples from the following slides demonstrate the percentile method with centered values as: `estimate - bootstrapped distribution + null hypothesis value` and with the p-value calculated as a percentage of more extreme values than the estimate.

Example 1: Population Stability Index (1-sided test)

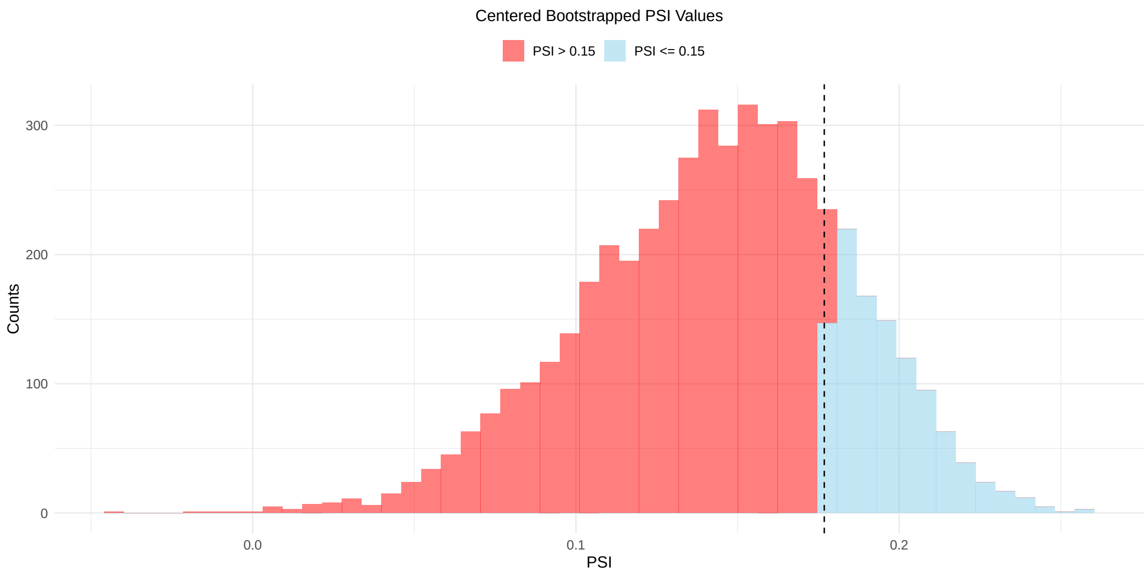
Dataset:

```
##   Bin Base cnt. Base pct. Target cnt. Target pct.  PSI
## 1   1       119      0.22         155        0.35 0.18
## 2   2       130      0.24         139        0.31 0.18
## 3   3        39      0.07          24        0.05 0.18
## 4   4       263      0.48         131        0.29 0.18
```

Testing Hypothesis:

$H_0 : PSI \leq 0.15$

Visualization:



p-value:

```
## [1] "21.26%"
```

Example 2: Herfindahl-Hirschman Index (1-sided test)

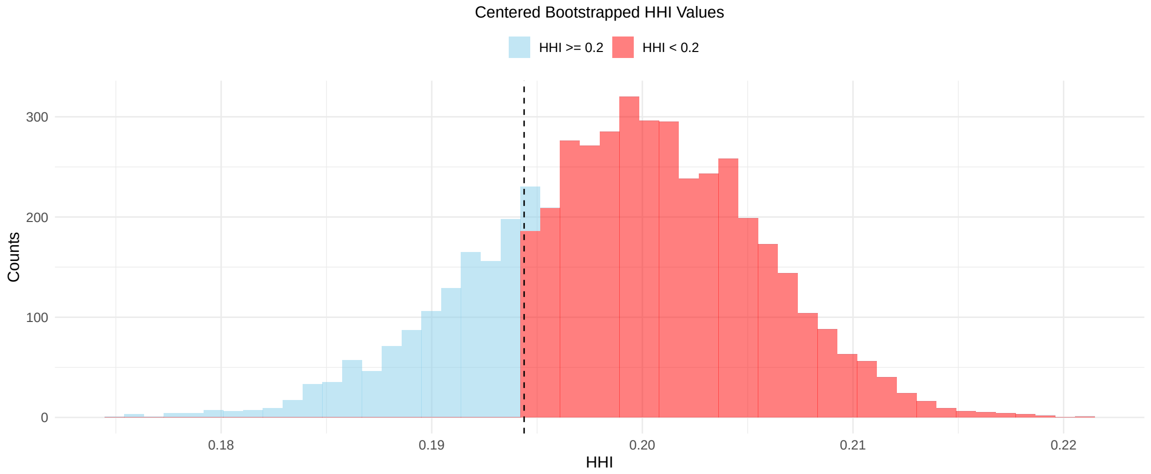
Dataset:

```
##         Rating Grade # obs.   DR   HHI
## 1    01 (-Inf,0.0199)    202 0.01 0.194
## 2 02 [0.0199,0.0263)     54 0.02 0.194
## 3 03 [0.0263,0.0369)     96 0.03 0.194
## 4 04 [0.0369,0.0903)    204 0.06 0.194
## 5    05 [0.0903,0.15)    103 0.11 0.194
## 6     06 [0.15,0.197)     41 0.12 0.194
## 7      07 [0.197,Inf)     50 0.32 0.194
```

Testing Hypothesis:

$H_0 : HHI \geq 0.20$

Visualization:



p-value:

```
## [1] "23.72%"
```

Example 3: Area Under Curve (2-sided test)

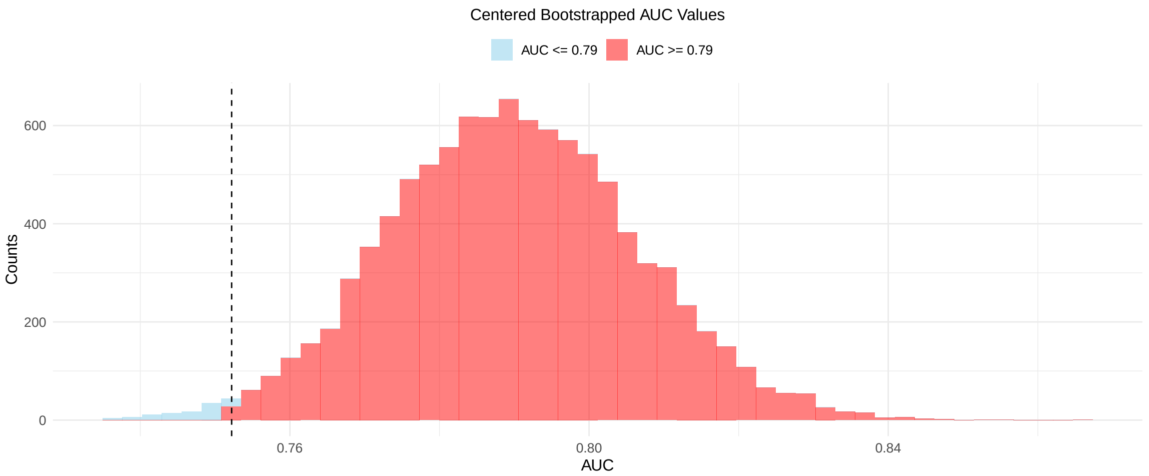
Dataset:

Development sample AUC 79%. Application portfolio AUC 75.2%.

Testing Hypothesis:

$H_0 : AUC = 0.79$

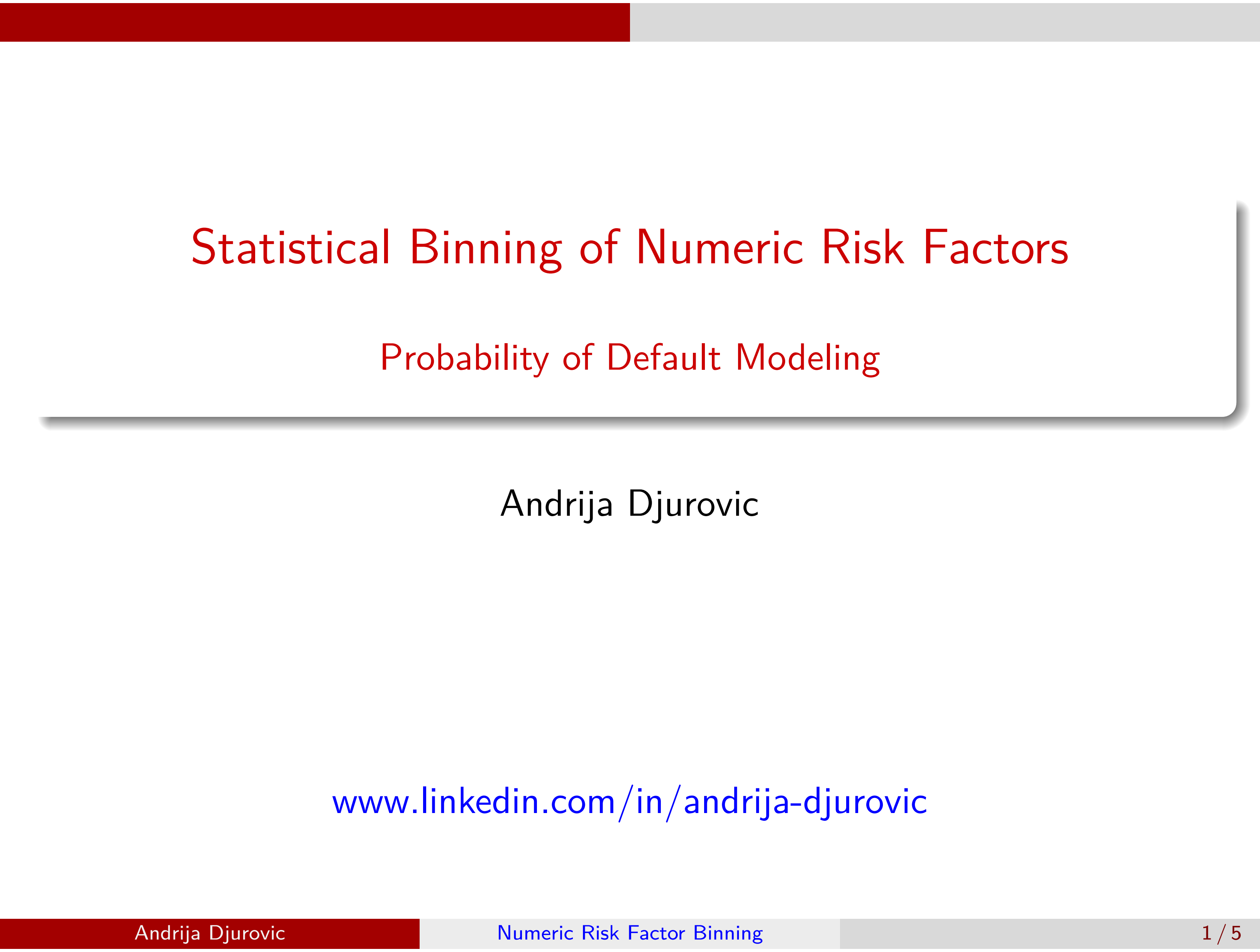
Visualization:



p-value:

2*min(c(left-side p-value, right-side p-value))

```
## [1] "2.06%"
```



Statistical Binning of Numeric Risk Factors

Probability of Default Modeling

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic | Numeric Risk Factor Binning | 1 / 5

Statistical Binning in Credit Risk

Binning is not a mandatory step in model development but does offer several advantages over the use of continuous risk factors.

The benefits mainly include:

- the reduction of outliers
- the ability to justify the empirical relationship between the risk factor and the target variable
- the involvement of business judgment in the binning process.

Principles of Good Binning Process

The most commonly used principles of the good binning process:

- each bin should contain at least 5% of the observations
- each bin should contain at least one bad case
- adjacent bins should have different riskiness levels
- risk level of the bins should have either a monotonic or U-shape trend
- number of bins should not be greater than ten.

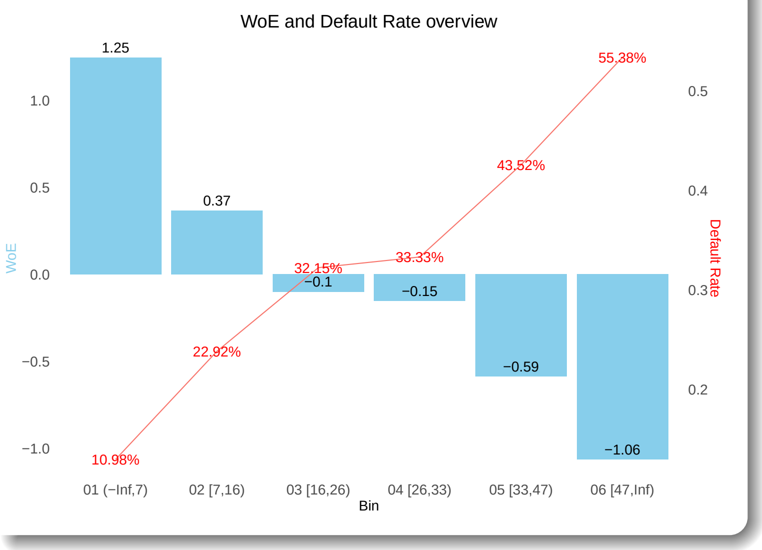
Do we have to follow all the principles?

How does binning link with the other modeling steps (e.g., encoding or multivariate analysis)?

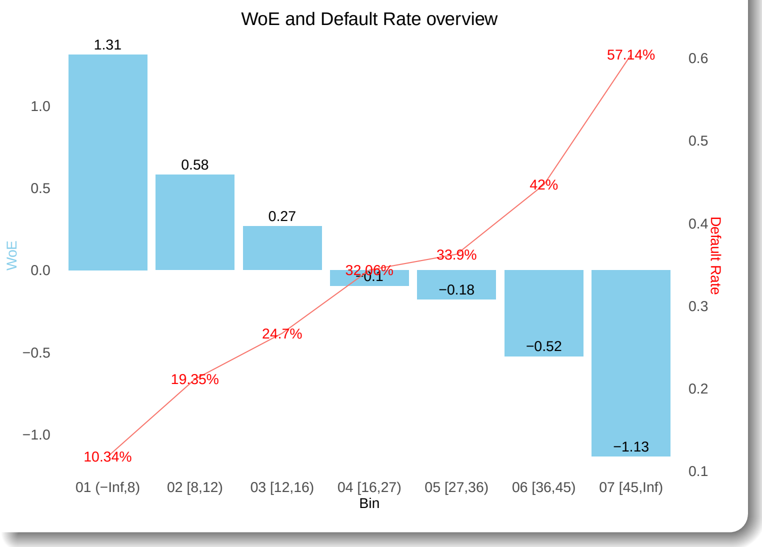
Monotonic Binning

Different binning algorithms exist and some of them are readily available in `R` and `Python` (e.g., monobin and monobinpy).

Monotone Adjacent Pooling Algorithm (MAPA)



Isotonic Regression



U-shape Binning

U-shape trend is sometimes a desired property of the relationship between the risk factor and the target variable. This relationship is present if we find a point in our risk factor (usually called an inflection point), after which the relationship changes direction.

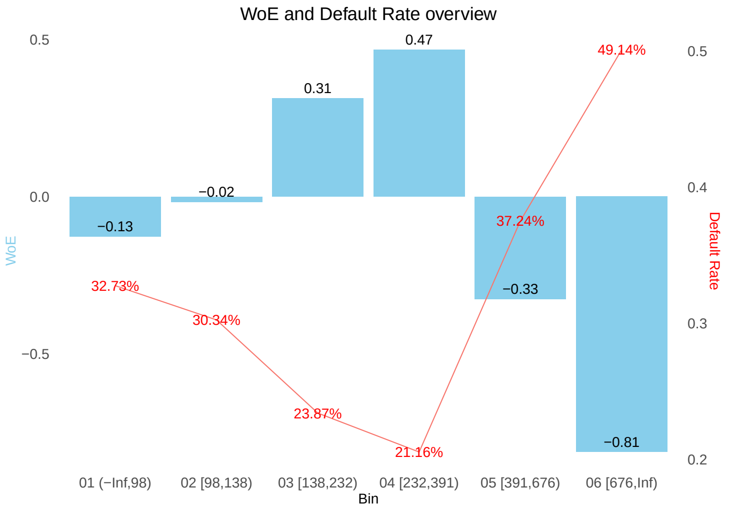
Binning process

- Determine inflection point (expertly, statistically or both)
- Perform binning for values before and after inflection point.

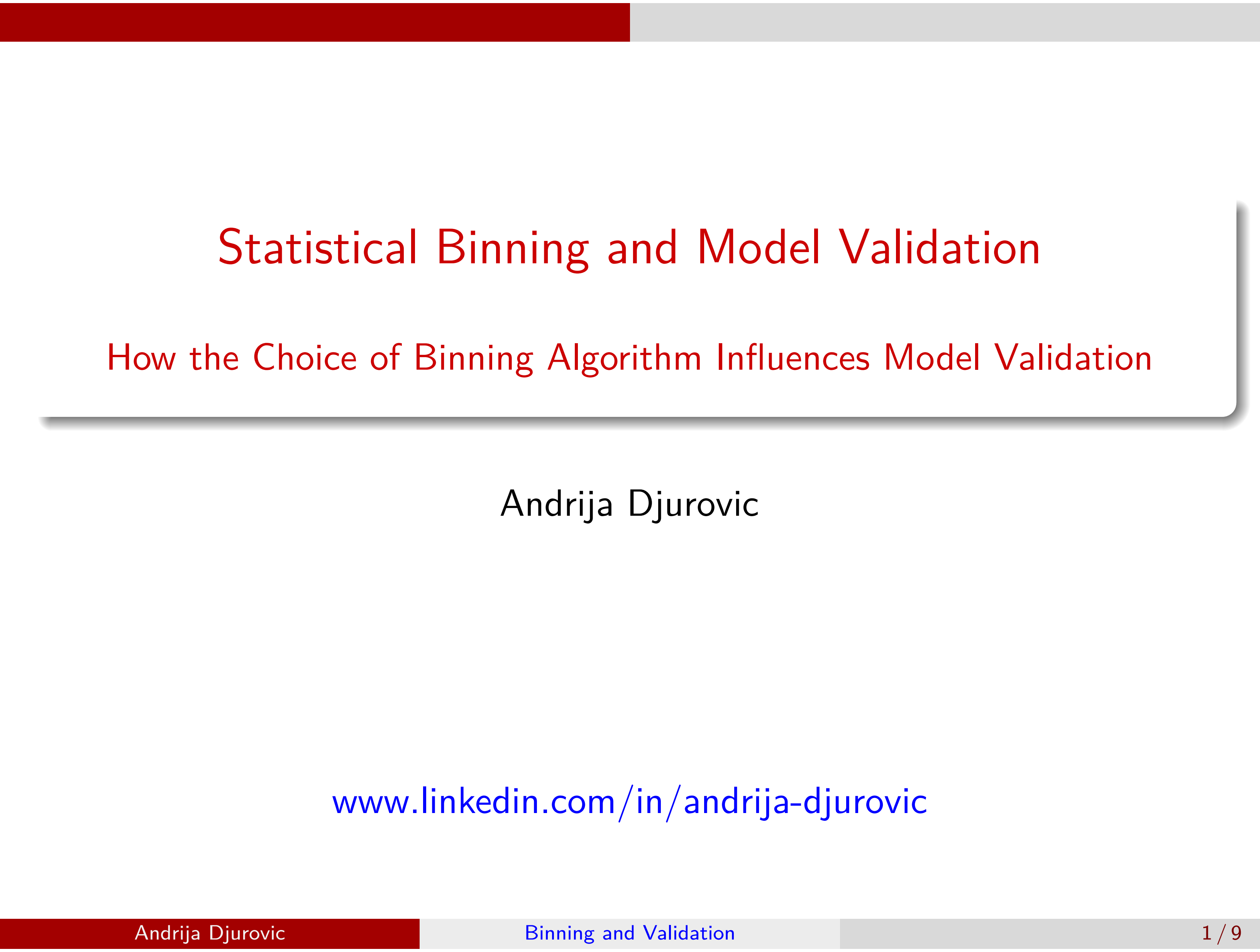
B-spline basis functions can be convinient for testing and determining the inflection point.

Testing and U-shape binning available in R package PDtoolkit via functions `ush.test` and `ush.bin`.

U-shape Binning based on Isotonic Regression



9.19 Statistical Binning and Model Validation - How the Choice of Binning Algorithm Influences Model Validation



Statistical Binning and Model Validation

How the Choice of Binning Algorithm Influences Model Validation

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic Binning and Validation 1 / 9

Statistical Binning in Credit Risk

- Binning is not a mandatory step in model development but offers several advantages over continuous risk factors.
- Acknowledging its benefits, practitioners often employ binning while developing credit risk models.
- The most commonly used principles of a good binning process are:
    - each bin should contain at least 5% of the observations;
    - each bin should contain at least one bad case;
    - adjacent bins should have different riskiness levels;
    - the risk levels of the bins should follow either a monotonic or U-shape trend;
    - the number of bins should not be greater than ten.
- Refer to this document for more details on the statistical binning of numeric risk factors.
- In practice, questions often arise about whether all the principles must be followed.
- It is also crucial to understand how binning connects to other modeling steps and validation processes.

Practical Challenges with the Binning Process

- Binning can be applied to both numeric and categorical risk factors, with numeric factors often being the focus of automated statistical binning procedures.
- The binning process for numeric risk factors typically follows a monotonic or U-shaped trend.
- In practice, there are various ways to perform binning for risk factors, leading to different levels of granularity in the final number of bins.
- Although there are no formal guidelines on which binning algorithms to use, their selection should consider a combination of statistical and business objectives.
- One of the biggest challenges in the model validation process is addressing binning procedures that produce a “large” number of bins.
- Such risk drivers are often deemed unstable by validators, but is that inherently a problem?
- The following slides present a simplified simulation design demonstrating how different binning algorithms can impact the final rating scale.

Simulation Design and Expected Results

Simulation Design

The following steps outline a simplified simulation demonstrating how different binning algorithms affect the number of rating grades derived from various models:

- Assume a development sample of 1,000 observations, with the target variable being a default indicator (`default`) and three numeric risk factors (`maturity`, `age`, `amount`).
- We categorized the numeric risk factors using two different monotonic binning algorithms. For demonstration purposes, the following slides utilize two functions implemented identically in `R` and `Python`: `iso.bin` and `sts.bin` from the `monobin` package in `R`, and `iso_bin` and `sts_bin` from the `monobinpy` package in `Python`.
- Due to their differing underlying algorithms, these two functions are expected to produce different numbers of bins for some or all risk factors undergoing the binning process.
- After binning and creating new discretized risk factors (`maturity.iso`, `age.iso`, `amount.iso`, `maturity.sts`, `age.sts`, `amount.sts`), we ran two logistic regressions using the discretized risk factors from each binning procedure.
- Finally, we generate within-sample predictions from both regressions and apply `sts.bin`/`sts_bin` to create the final rating scale.

Expected results

As the results on the following slide demonstrate, the two models produce different numbers of final rating grades.

Given these differences in the final output, can we definitively determine whether one binning algorithm is superior?

R Code

```r
library(monobin)

#data import
url <- "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/model_dev_and_vld/bin_vld.csv"
db <- read.csv(file = url, header = TRUE)

#numeric risk factors
rf <- names(db)[-1]

#------------------------------iso.bin algorithm-----------------------------#
iso.b <- sapply(X = rf, FUN = function(x) {
                              iso.bin(x = db[, x], y = db$default)[[2]]})
#prepare data frame
iso.b <- data.frame(iso.b)
names(iso.b) <- c("maturity.iso", "age.iso", "amount.iso")
#check unique number of bins per risk factor
sapply(X = iso.b, FUN = function(x) length(unique(x)))
```

```
## maturity.iso      age.iso   amount.iso
##            7            4            3
```

```r
#add discretized risk factors
db <- cbind.data.frame(db, iso.b)

#------------------------------sts.bin algorithm-----------------------------#
sts.b <- sapply(X = rf, FUN = function(x) {
                              sts.bin(x = db[, x], y = db$default)[[2]]})
#prepare data frame
sts.b <- data.frame(sts.b)
names(sts.b) <- c("maturity.sts", "age.sts", "amount.sts")
#check unique number of bins per risk factor
sapply(X = sts.b, FUN = function(x) length(unique(x)))
```

```
## maturity.sts      age.sts   amount.sts
##            5            3            3
```

```r
#add discretized risk factors
db <- cbind.data.frame(db, sts.b)
```

R Code cont.

```
#------------------------------model comparison----------------------------#
#run glm based on iso.bin risk factors
lr.1 <- glm(formula = default ~ `maturity.iso` + `age.iso` + `amount.iso`,
            family = binomial,
            data = db)
#run glm based on sts.bin risk factors
lr.2 <- glm(formula = default ~ `maturity.sts` + `age.sts` + `amount.sts`,
            family = binomial,
            data = db)

#create rating scale for lr.1 based on sts.bin algorithm
sts.bin(x = predict(object = lr.1), y = db$default)[[1]][, 1:4]
```

```
##                     bin  no y.sum      y.avg
## 1    01 (-Inf,-1.9637)  73     4 0.05479452
## 2 02 [-1.9637,-1.3642) 181    30 0.16574586
## 3 03 [-1.3642,-0.9815) 208    51 0.24519231
## 4 04 [-0.9815,-0.3044) 335   113 0.33731343
## 5  05 [-0.3044,0.0315) 128    56 0.43750000
## 6      06 [0.0315,Inf)  75    46 0.61333333
```

```
#create rating scale for lr.2 based on sts.bin algorithm
sts.bin(x = predict(object = lr.2), y = db$default)[[1]][, 1:4]
```

```
##                     bin  no y.sum      y.avg
## 1    01 (-Inf,-1.9011)  73     4 0.05479452
## 2 02 [-1.9011,-1.5505) 152    23 0.15131579
## 3 03 [-1.5505,-1.0054) 233    58 0.24892704
## 4 04 [-1.0054,-0.2518) 362   123 0.33977901
## 5     05 [-0.2518,Inf) 180    92 0.51111111
```

Python Code

```
import pandas as pd
import numpy as np
from monobinpy import iso_bin, sts_bin
import statsmodels.api as sm
import statsmodels.formula.api as smf

#data import
url = "https://raw.githubusercontent.com/andrija-djurovic/adsfcr/main/model_dev_and_vld/bin_vld.csv"
db = pd.read_csv(filepath_or_buffer = url)

#numeric risk factors
rf = db.columns[1:]

#-----------------------------iso_bin algorithm-----------------------------#
iso_b = {x: iso_bin(x = db[x], y = db["default"])[1] for x in rf}
#prepare data frame
iso_b = pd.DataFrame(iso_b)
iso_b.columns = ["maturity.iso", "age.iso", "amount.iso"]

#check unique number of bins per risk factor
{col: iso_b[col].nunique() for col in iso_b.columns}
```

```
## {'maturity.iso': 7, 'age.iso': 4, 'amount.iso': 3}
```

```
#add discretized risk factors to the original data
db = pd.concat([db, iso_b], axis = 1)

#-----------------------------sts_bin algorithm-----------------------------#
sts_b = {x: sts_bin(x = db[x], y = db["default"])[1] for x in rf}
#prepare data frame
sts_b = pd.DataFrame(sts_b)
sts_b.columns = ["maturity.sts", "age.sts", "amount.sts"]

#check unique number of bins per risk factor
{col: sts_b[col].nunique() for col in sts_b.columns}
```

```
## {'maturity.sts': 5, 'age.sts': 3, 'amount.sts': 3}
```

```
#add discretized risk factors to the original data
db = pd.concat([db, sts_b], axis = 1)
```

Python Code cont.

```
# -----------------------------model comparison----------------------------#
#run glm based on iso_bin risk factors
formula_iso = "default ~ Q('maturity.iso') + Q('age.iso') + Q('amount.iso')"
lr_1 = smf.glm(formula = formula_iso,
               data = db,
               family = sm.families.Binomial()).fit()

#run glm based on sts_bin risk factors
formula_sts = "default ~ Q('maturity.sts') + Q('age.sts') + Q('amount.sts')"
lr_2 = smf.glm(formula = formula_sts,
               data = db,
               family = sm.families.Binomial()).fit()

#create rating scale for lr_1 based on sts_bin algorithm
pred_lr_1 = pd.Series(lr_1.predict(which = "linear"))
sts_bin(x = pred_lr_1, y = db["default"])[0].iloc[:, :4]
```

```
##                       bin   no  y_sum     y_avg
## 0     01 (-inf, -1.9637)   73      4  0.054795
## 1  02 [-1.9637, -1.3642)  181     30  0.165746
## 2  03 [-1.3642, -0.9815)  208     51  0.245192
## 3  04 [-0.9815, -0.3044)  335    113  0.337313
## 4   05 [-0.3044, 0.0315)  128     56  0.437500
## 5       06 [0.0315, inf)   75     46  0.613333
```

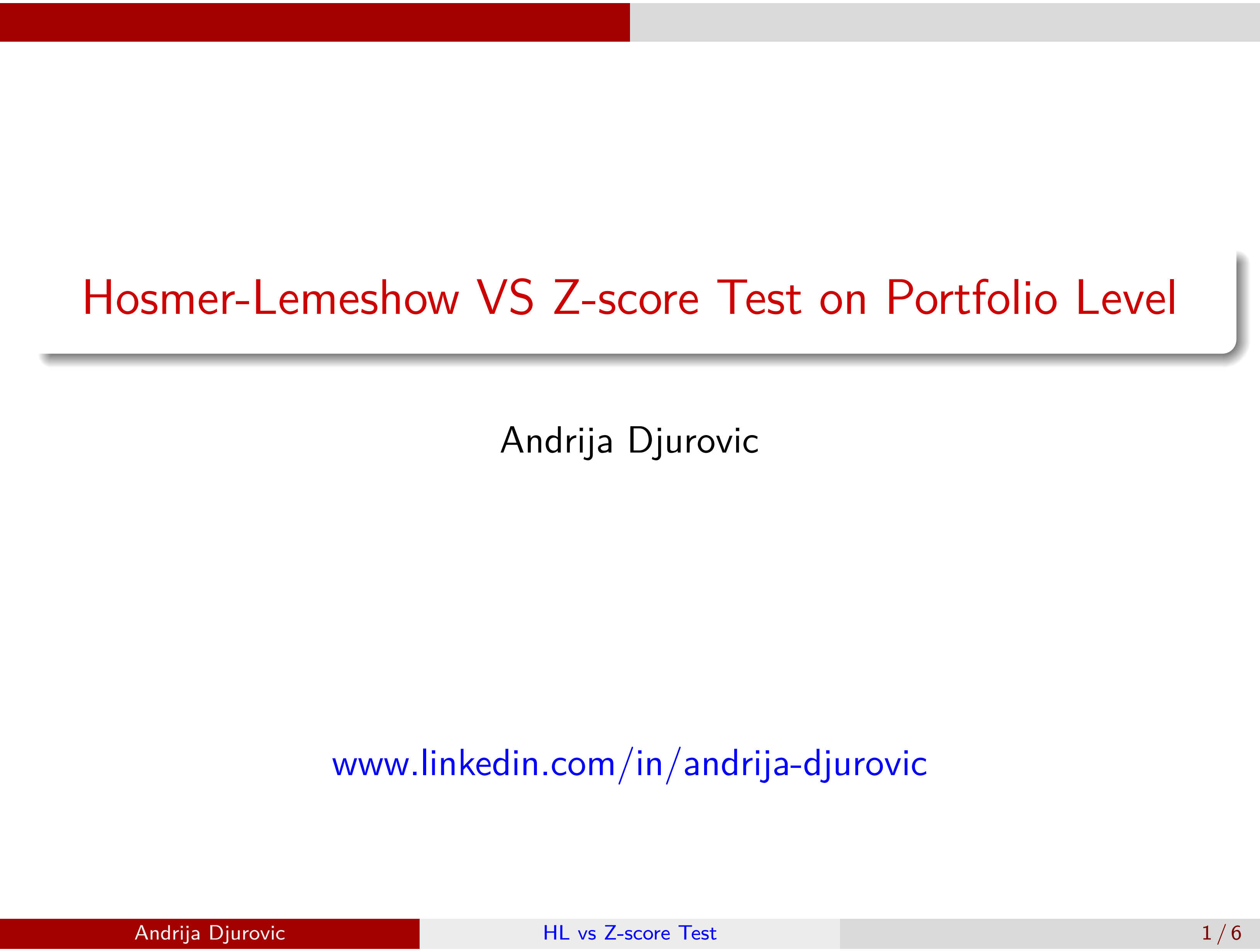
```
#create rating scale for lr_2 based on sts_bin algorithm
pred_lr_2 = pd.Series(lr_2.predict(which = "linear"))
sts_bin(x = pred_lr_2, y = db["default"])[0].iloc[:, :4]
```

```
##                       bin   no  y_sum     y_avg
## 0     01 (-inf, -1.9011)   73      4  0.054795
## 1  02 [-1.9011, -1.5505)  152     23  0.151316
## 2  03 [-1.5505, -1.0054)  233     58  0.248927
## 3  04 [-1.0054, -0.2518)  362    123  0.339779
## 4      05 [-0.2518, inf)  180     92  0.511111
```

Discussion Points

- How should model validators (internal, external, or regulators) analyze models containing discretized risk factors with an excessive number of bins?
- Should validators automatically penalize the potential instability of discretized risk factors with an excessive number of bins?
- What steps should modelers take during the development process to anticipate potential issues during validation (both initial and periodic)?
- What are the differences between using numeric risk factors and those discretized with an excessive number of bins?
- How should validators address this issue when dealing with machine learning models used in credit risk modeling?

9.20 Hosmer-Lemeshow VS Z-score Test on Portfolio Level



Hosmer-Lemeshow VS Z-score Test on Portfolio Level

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic HL vs Z-score Test 1 / 6

Probability of Default Predictive Power Tests

- Divided into two groups:
  1. tests applicable on the rating scale level
  2. tests applicable on the rating grade or portfolio level.
- Sometimes, practitioners compare the p-value of a test at the rating scale level with the p-value of a test at the portfolio level. However, are they comparable?
- What is the testing hypothesis for each group and test?

Hosmer-Lemeshow Test - Rating Scale Level

$$HL = \sum_{g=1}^{G} \frac{(N_g PD_g - d_g)^2}{N_g PD_g (1 - PD_g)}$$

where:

- G is the number of rating grades
- N_g is the number of observations in the rating grade g
- PD_g is the calibrated PD for the rating grade g
- d_g is the number of observed defaults in the rating grade g.

Under the assumption that the HL test statistic follows the chi-square distribution with G degrees of freedom, a $p-value$ is calculated accordingly.

Testing hypothesis - the calibrated PD is true.

Z-score Test - Rating Grade or Portfolio Level

$$Z_{score} = \frac{ODR - PD}{\sqrt{\frac{PD(1-PD)}{n}}}$$

where:

- ODR is the observed default rate
- PD is the calibrated PD
- n is the number of observations.

Under the assumption that the Z_{score} test statistic follows the standard normal distribution, a $p - value$ is calculated accordingly.

The most commonly used testing hypothesis - the calibrated PD is not underestimated.

Hosmer-Lemeshow VS Z-score on Portfolio Level

Are they comparable?

Example:

```
##   Rating Grade # observations # defaults    ODR     PD
## 1          RG1             47          3 0.0638 0.0307
## 2          RG2             95         20 0.2105 0.1161
## 3          RG3             68         17 0.2500 0.2907
## 4          RG4             53         24 0.4528 0.5514
## 5          RG5             37         28 0.7568 0.7648
```

p-values:

```
##   Hosmer-Lemeshow test Z-score test on portfolio level
## 1                2.70%                          38.89%
```

Hosmer-Lemeshow VS Z-score on Portfolio Level cont.

Can we adjust inputs to account only for the underestimation?

Example:

```
##   Rating Grade # observations # defaults    ODR     PD PD corrected
## 1          RG1             47          3 0.0638 0.0307       0.0307
## 2          RG2             95         20 0.2105 0.1161       0.1161
## 3          RG3             68         17 0.2500 0.2907       0.2500
## 4          RG4             53         24 0.4528 0.5514       0.4528
## 5          RG5             37         28 0.7568 0.7648       0.7568
```

p-values:

```
##   Hosmer-Lemeshow test Z-score test on portfolio level
## 1                7.53%                           8.58%
```

9.21 Power Play: Probability of Default Predictive Ability Testing

Power Play: Probability of Default Predictive Ability Testing

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Andrija Djurovic PD Predictive Ability Testing 1 / 7

Probability of Default Predictive Ability Tests

Three of the most commonly used tests:

- Exact binomial
- Z-score test
- Jeffreys test.

Testing the hypothesis that the calibrated probability of default (PD) is not underestimated.

Exact binomial

$$Pr(X \geq d) = \binom{n}{d} PD^{d}(1 - PD)^{n-d}$$

where:

- n is the number of observations
- d is the number of defaults
- PD is the calibrated PD.

Z-score test:

$$Z_{score} = \frac{ODR - PD}{\sqrt{\frac{PD(1-PD)}{n}}}$$

where:

- ODR is the observed default rate
- PD is the calibrated PD
- n is the number of observations.

Under the assumption that the Z_{score} test statistic follows the standard normal distribution, a $p - value$ is calculated accordingly.

Jeffreys test

$$Pr(X \geq d) = \beta(q = PD, shape1 = d + 0.5, shape2 = n - d + 0.5)$$

where:

- β is the beta distribution
- PD is the calibrated PD
- d is the number of defaults
- n is the number of observations.

What if. . . ?

What if the tests lead to the opposite conclusion?

Example:

```
#number of defaults
nb = 15
#number of observations
no = 99
#calibrated pd
pdc = 0.09656014
#significance level
alpha = 0.05
```

Testing results:

```
##   Exact binomial Z-score test Jeffreys test
## 1          5.30%        3.21%         3.87%
```

Monte Carlo Simulations for Statistical Power

It helps identify which test has higher statistical power.

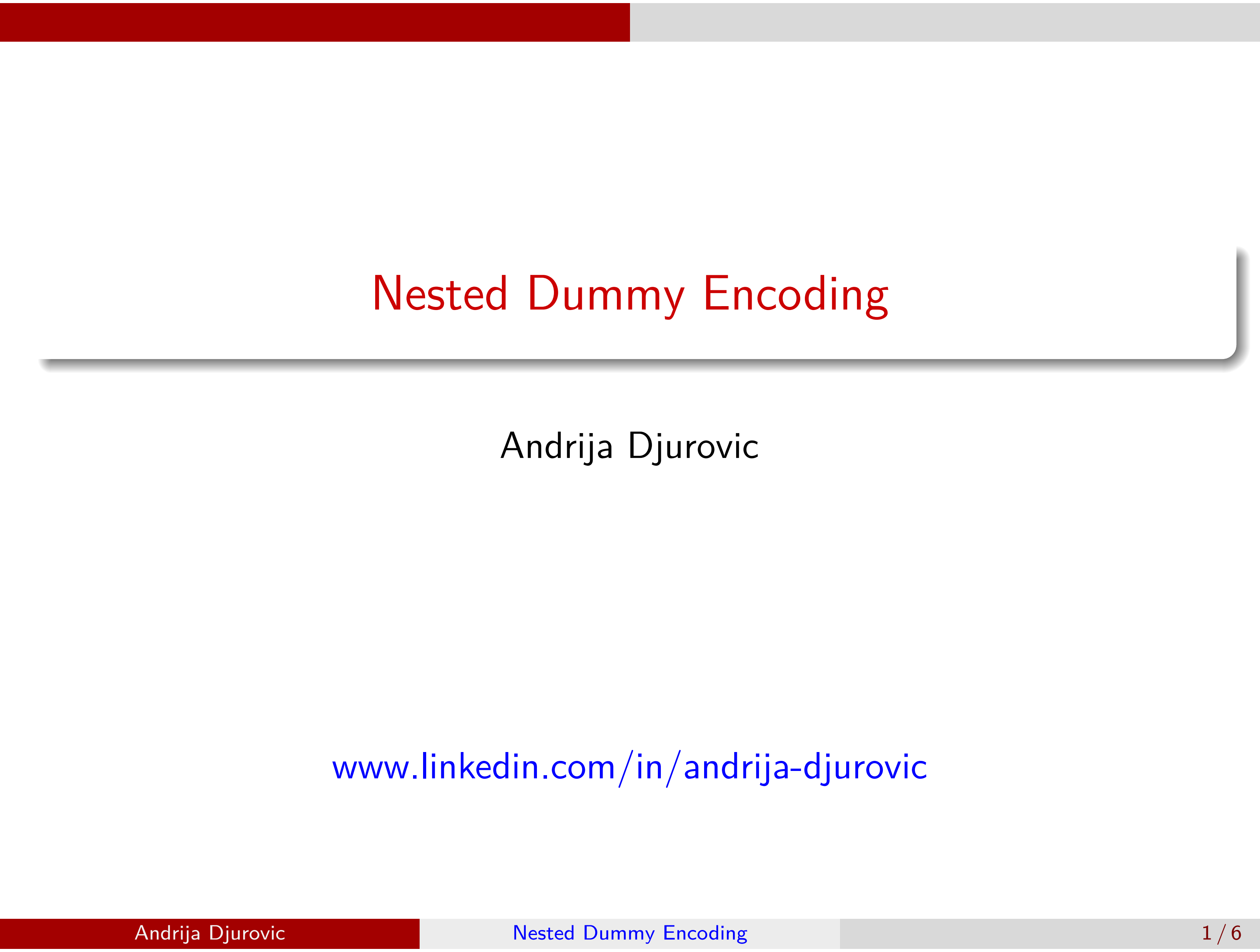
Steps:

1. The observed default rate is the true calibrated PD (`ODR = nb / no`)
2. Simulate random numbers from the binomial distribution with parameters `ODR` and `no`
3. Calculate the simulated `ODR` and `nb`
4. Collect the results of the tests
5. Repeat steps 2 to 4 `N` times
6. Calculate the average number of simulations for which tests reject the null hypothesis for an `alpha` significance level (a higher average indicates greater power)

Simulation results:

```
##   Exact binomial Z-score test Jeffreys test
## 1         43.10%       54.17%        54.17%
```

9.22 Nested Dummy Encoding



Nested Dummy Encoding

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Nested Dummy VS One-Hot Encoding

- Nested dummy encoding, an alternative to standard one-hot encoding, but less commonly used in credit risk modeling.
- Nested dummy encoding applies to categorical variables that exhibit ordinal relationships among their categories.
- Nested dummy encoding creates links between adjacent categories, unlike one-hot encoding, even though in a bivariate setup both yield identical model outputs.

Nested Dummy Encoding in Credit Risk Modeling

- Establishing connections between adjacent categories makes this encoding method particularly attractive in credit risk modeling, especially when practitioners choose to bin numeric risk factors and seek statistically significant risk profiles between adjacent categories.
- This method provides opportunities to assess diverse binning methods in standard bivariate and multivariate analyses.

Constructing Nested Dummies

Assign a value of 0 to all categories below the chosen one and 1 to the selected category and those positioned above it.

Example

```
categories = c("A", "A", "B", "B", "C", "C", "C", "D", "D", "D")
```

```
nested_dummies
```

```
##    category_A_vs_BCD category_AB_vs_CD category_ABC_vs_D
## 1                  0                 0                 0
## 2                  0                 0                 0
## 3                  1                 0                 0
## 4                  1                 0                 0
## 5                  1                 1                 0
## 6                  1                 1                 0
## 7                  1                 1                 0
## 8                  1                 1                 1
## 9                  1                 1                 1
## 10                 1                 1                 1
```

Interpreting Nested Dummies

OLS regression with nested dummies

```
##
## Call:
## lm(formula = y1 ~ category_A_vs_BCD + category_AB_vs_CD + category_ABC_vs_D,
##     data = db)
##
## Coefficients:
##       (Intercept)  category_A_vs_BCD  category_AB_vs_CD  category_ABC_vs_D
##           0.01862            0.06086            0.05233           -0.25819
```

Target average per categorical variable

```
##           A           B           C           D
##  0.01861955  0.07948448  0.13181361 -0.12638054
```

Coefficient replicates

```
"(Intercept)" = average["A"]
"category_A_vs_BCD" = average["B"] - average["A"]
"category_AB_vs_CD" = average["C"] - average["B"]
"category_ABC_vs_D" = average["D"] - average["C"]
```

```
##       (Intercept) category_A_vs_BCD category_AB_vs_CD category_ABC_vs_D
##        0.01861955        0.06086492        0.05232913       -0.25819415
```

Interpreting Nested Dummies cont.

Binomial logistic regression with nested dummies

```
##
## Call:  glm(formula = y2 ~ category_A_vs_BCD + category_AB_vs_CD + category_ABC_vs_D,
##     family = "binomial", data = db)
##
## Coefficients:
##       (Intercept)  category_A_vs_BCD  category_AB_vs_CD  category_ABC_vs_D
##           -1.5667            -0.3259             0.1926            -0.4247
##
## Degrees of Freedom: 999 Total (i.e. Null);  996 Residual
## Null Deviance:       813.5
## Residual Deviance: 808.4     AIC: 816.4
```

Log-odds of traget average per categorical variable

```
##         A         B         C         D
## -1.566676 -1.892564 -1.699952 -2.124698
```

Coefficient replicates

```
"(Intercept)" = lo(average["A"])
"category_A_vs_BCD" = lo(average["B"]) - lo(average["A"])
"category_AB_vs_CD" = lo(average["C"]) - lo(average["B"])
"category_ABC_vs_D" = lo(average["D"]) - lo(average["C"])
```

```
##       (Intercept) category_A_vs_BCD category_AB_vs_CD category_ABC_vs_D
##        -1.5666761        -0.3258881         0.1926122        -0.4247462
```

9.23 Marginal Information Value

Marginal Information Value

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Probability of Default Modeling and Stepwise Procedures

- The stepwise logistic regression procedures most commonly used for modeling the PD still rely on the p-values of the coefficients.
- An alternative, less frequently employed in practice, is based on Marginal Information Values (MIV).
- Can we benefit from combining them?

Marginal Information Value (MIV)

- The concept behind the MIV score for a new variable is that, after constructing a model using a specific set of available variables, the MIV evaluates the new variable by measuring the additional information it is likely to contribute compared to the predictions generated by the current model.
- Include a sufficient number of variables to account for the variation in outcomes across the sample, but avoid adding unnecessary ones.

Example of MIV Calculation

MIV table for the variable Maturity

	Observed			Expected		
Maturity	# good obs.	# bad obs.	WoE obs.	# good exp.	# bad exp.	WoE exp.
[4,7]	78	9	1.31	62.29	24.71	0.08
(7,15]	264	80	0.35	240.20	103.80	-0.01
(15,42]	328	171	-0.20	348.60	150.40	-0.01
(42,Inf]	30	40	-1.13	48.91	21.09	-0.01

MIV score

```
delta = "WoE obs." - "WoE exp."
miv_g = sum("# good obs." * delta) / sum("# good obs.")
miv_b = sum("# bad obs." * delta) / sum("# bad obs.")
miv = miv_g - miv_b
miv
```

```
## [1] 0.2611226
```

MIV Threshold and Supporting Statistical Test

- The most frequently used MIV threshold is 0.02.
- Sometimes the absolute MIV threshold is supported by the Marginal Chi-square test.

9.24 Scorecard Scaling

Scorecard Scaling

Andrija Djurovic

www.linkedin.com/in/andrija-djurovic

Scorecard Scaling

The most commonly used method for developing a scorecard is logistic regression. In practice, this is closely linked to the presentation of its output through a process known as scaling. Scorecard scaling refers to transforming logistic regression outputs into a user-friendly scoring format. The objective is to map probabilities or odds into a numerical score range that is easy to interpret and use for decision-making. A standard industry approach uses a logarithmic scale, where the odds double every fixed number of points - known as "points to double the odds" (PDO).
The following equation gives the general relationship between score and odds:

$$\text{Score} = \text{Offset} + \text{Factor} \cdot \ln(\text{odds})$$

where the offset and factor are determined based on reference points chosen by the user.
For instance, if a score of 600 corresponds to odds of 50:1, and the odds double every 20 points, then the factor and offset can be calculated as:

$$\text{Factor} = \frac{\text{PDO}}{\ln(2)} = \frac{20}{\ln(2)} \approx 28.85$$
$$\text{Offset} = 600 - (28.85 \cdot \ln(50)) \approx 487.12$$

Each score in the scorecard is then derived using this formula, ensuring consistency and interpretability across different scorecards.

The example above demonstrates how to transform the model output into scores. However, practitioners often include an additional step in which each modality of a risk factor (i.e., each characteristic attribute) is assigned a specific score. The total sum of the individual modality scores should match the overall model score produced by the method described earlier. The following slides illustrate score allocation to risk factor modalities for the two most commonly used encoding methods: Weight of Evidence (WoE) and dummy encoding.
In addition, for more details on scorecard scaling using WoE encoding, practitioners can refer to Siddiqi, N. (2012). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. John Wiley & Sons, Inc.

Scorecard Scaling - WoE Encoding

Since the scorecard is typically developed using WoE encoding, the final score for an applicant is determined by summing the points assigned to each modality of the risk factors in the scorecard.

By applying the Factor and Offset, points are allocated proportionally to the each modality. The final credit score is obtained by summing the points assigned to all relevant modalities. The point allocation follows the trends established in the WoE analysis. Expressed in terms of formulas, the steps below demonstrate the process of point allocation to the modalities of each risk factor:

$$\text{Score} = \text{Offset} + \ln(\text{odds}) \cdot \text{Factor} =$$

$$-\left(\sum_{j=1}^{k}\sum_{i=1}^{n}(\text{WoE}_j \cdot \beta_i) + \alpha\right) \cdot \text{Factor} + \text{Offset} =$$

$$-\left(\sum_{j=1}^{k}\sum_{i=1}^{n}(\text{WoE}_j \cdot \beta_i + \frac{\alpha}{n})\right) \cdot \text{Factor} + \text{Offset} =$$

$$\sum_{j=1}^{k}\sum_{i=1}^{n}\left(-\text{WoE}_j \cdot \beta_i + \frac{\alpha}{n}\right) \cdot \text{Factor} + \frac{\text{Offset}}{n}$$

Scorecard Scaling - WoE Encoding cont.

where:

- WoE_j is the Weight of Evidence for each modality;
- β_i is the regression coefficient for each risk factor;
- α denotes the intercept term from logistic regression;
- n is the number of risk factors in the model;
- k denotes the number of modalities in each risk factor;
- Factor is a scaling parameter used to adjust the scores of each modality within a given risk factor;
- Offset is the constant used to shift the score into a desired range.

Scorecard Scaling - Dummy Encoding

Unlike WoE encoding, dummy encoding produces multiple logistic regression coefficients directly associated with the modalities of the risk factors. Although the logic of point allocation remains similar, the assignment process differs slightly. Since the impact of each risk factor's reference modality is absorbed into the model's intercept, the points allocated to these reference modalities will be the same across all risk factors. Expressed in terms of formulas, the steps below demonstrate the process of point allocation to the modalities of each risk factor using dummy encoding:

$$\text{Score} = \text{Offset} + \ln(\text{odds}) \cdot \text{Factor} =$$

$$-\left(\sum_{j=1}^{k}\sum_{i=1}^{n}(D_j \cdot \beta_j) + \alpha\right) \cdot \text{Factor} + \text{Offset} =$$

$$-\left(\sum_{j=1}^{k}\sum_{i=1}^{n}(D_j \cdot \beta_j + \frac{\alpha}{n})\right) \cdot \text{Factor} + \text{Offset} =$$

$$\sum_{j=1}^{k}\sum_{i=1}^{n}\left(-D_j \cdot \beta_j + \frac{\alpha}{n}\right) \cdot \text{Factor} + \frac{\text{Offset}}{n}$$

Scorecard Scaling - Dummy Encoding cont.

where:

- D_j is the dummy variable indicator for modality j (1 if present, 0 otherwise);
- β_j is the regression coefficient for modality j;
- α is the intercept term from the logistic regression;
- n is the number of risk factors in the model;
- k is the number of modalities in each risk factor;
- Factor is a scaling parameter used to adjust the scores of each modality within a given risk factor;
- Offset is the constant used to shift the score into a desired range.

Simulation Study - WoE Encoding

Assume that the model's binary target `Creditability` is estimated using only two categorical risk factors, `Account_Balance` and `Maturity`, with four and five unique modalities, respectively. A simulation dataset is available here. Under the assumption that the model uses WoE encoding, this and the following slides present the necessary steps and elements for transforming risk factor modalities into scores.

1. After WoE encoding of `Account_Balance` and `Maturity`, the estimated model in the form `Creditability ~ Account_Balance_WoE + Maturity_WoE` produces the following result:

```
##                      Estimate Std. Error  z value Pr(>|z|)
## (Intercept)           -0.8470     0.0774 -10.9364        0
## Account_Balance_WoE   -0.9913     0.0976 -10.1606        0
## Maturity_WoE          -0.9774     0.1489  -6.5664        0
```

2. Calculate the scaling inputs, Factor and Offset for a score of 600 at odds of 50:1, with 20 points to double the odds. Given these inputs, the value of the Factor is 28.85, and the Offset is 487.12.

3. Inputs from steps 1 and 2 are now sufficient to perform the score assignment process for each modality of the risk factors using the formula from slide 3.

```
##                rf         bin  no  ng  nb     woe intercept    beta score
## 1 Account_Balance          01 274 139 135 -0.8181    -0.847 -0.9913   232
## 2 Account_Balance          02 269 164 105 -0.4014    -0.847 -0.9913   244
## 3 Account_Balance          03  63  49  14  0.4055    -0.847 -0.9913   267
## 4 Account_Balance          04 394 348  46  1.1763    -0.847 -0.9913   289
## 5        Maturity 01 (-Inf,8)  87  78   9  1.3122    -0.847 -0.9774   293
## 6        Maturity   02 [8,16) 344 264  80  0.3466    -0.847 -0.9774   266
## 7        Maturity  03 [16,36) 399 270 129 -0.1087    -0.847 -0.9774   253
## 8        Maturity  04 [36,45) 100  58  42 -0.5245    -0.847 -0.9774   241
## 9        Maturity 05 [45,Inf)  70  30  40 -1.1350    -0.847 -0.9774   224
```

Simulation Study - WoE Encoding cont.

The final step is to verify that the scaling has been performed correctly. This can be done by comparing the score produced directly from the model output (model's log odds) with the sum of the individual modality scores for each observation in the development sample. The table below presents the unique model outputs alongside the model scores and the corresponding sums of individual modality scores. As shown, the final scores match precisely.

```
##    Account_Balance     Maturity Model_Score Account_Balance_Score Maturity_Score Individual_Sum_Score
## 1               01   03 [16,36)         485                   232            253                  485
## 2               01    02 [8,16)         498                   232            266                  498
## 3               02    02 [8,16)         510                   244            266                  510
## 4               01  01 (-Inf,8)         525                   232            293                  525
## 5               04   03 [16,36)         542                   289            253                  542
## 6               02   03 [16,36)         497                   244            253                  497
## 7               02  05 [45,Inf)         468                   244            224                  468
## 8               02   04 [36,45)         485                   244            241                  485
## 9               04    02 [8,16)         555                   289            266                  555
## 10              03   04 [36,45)         508                   267            241                  508
## 11              03   03 [16,36)         520                   267            253                  520
## 12              04   04 [36,45)         530                   289            241                  530
## 13              04  05 [45,Inf)         513                   289            224                  513
## 14              04  01 (-Inf,8)         582                   289            293                  582
## 15              03    02 [8,16)         533                   267            266                  533
## 16              02  01 (-Inf,8)         537                   244            293                  537
## 17              03  01 (-Inf,8)         560                   267            293                  560
## 18              01  05 [45,Inf)         456                   232            224                  456
## 19              01   04 [36,45)         473                   232            241                  473
```

Simulation Study - Dummy Encoding

This and the next slide present a simulation study on the same dataset with two key differences. First, dummy encoding is used instead of WoE encoding. Second, the scaling inputs are set with the Factor equal to 10,000 and the Offset equal to 0. The following steps demonstrate the scaling process under these assumptions.

1. The estimated model in the form `Creditability ~ Account_Balance_WoE + Maturity_WoE` produces the following result:

```
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)          -1.3234     0.3720 -3.5579   0.0004
## Account_Balance02    -0.5064     0.1809 -2.7997   0.0051
## Account_Balance03    -1.0873     0.3332 -3.2629   0.0011
## Account_Balance04    -2.0194     0.2029 -9.9507   0.0000
## Maturity02 [8,16)     0.9783     0.3873  2.5262   0.0115
## Maturity03 [16,36)    1.4282     0.3809  3.7495   0.0002
## Maturity04 [36,45)    1.8817     0.4248  4.4297   0.0000
## Maturity05 [45,Inf)   2.4041     0.4491  5.3532   0.0000
```

2. The Factor and Offset are predefined with values of 10,000 and 0, respectively.

3. Inputs from steps 1 and 2 are now sufficient to perform the score assignment process for each modality of the risk factors using the formula from slide 5.

```
##                rf         bin intercept    beta  score
## 1 Account_Balance          01   -1.3234  0.0000   6617
## 2 Account_Balance          02   -1.3234 -0.5064  11681
## 3 Account_Balance          03   -1.3234 -1.0873  17490
## 4 Account_Balance          04   -1.3234 -2.0194  26811
## 5        Maturity 01 (-Inf,8)   -1.3234  0.0000   6617
## 6        Maturity   02 [8,16)   -1.3234  0.9783  -3166
## 7        Maturity  03 [16,36)   -1.3234  1.4282  -7665
## 8        Maturity  04 [36,45)   -1.3234  1.8817 -12200
## 9        Maturity 05 [45,Inf)   -1.3234  2.4041 -17424
```

Simulation Study - Dummy Encoding cont.

The final step is to verify that the scaling has been performed correctly. This can be done by comparing the score produced directly from the model output (model's log odds) with the sum of the individual modality scores for each observation in the development sample. The table below presents the unique model outputs alongside the model scores and the corresponding sums of individual modality scores. As shown, the final scores match precisely.

```
##    Account_Balance    Maturity Model_Score Account_Balance_Score Maturity_Score Individual_Sum_Score
## 1               01  03 [16,36)       -1048                  6617          -7665                -1048
## 2               01   02 [8,16)        3451                  6617          -3166                 3451
## 3               02   02 [8,16)        8515                 11681          -3166                 8515
## 4               01 01 (-Inf,8)       13234                  6617           6617                13234
## 5               04  03 [16,36)       19146                 26811          -7665                19146
## 6               02  03 [16,36)        4016                 11681          -7665                 4016
## 7               02 05 [45,Inf)       -5743                 11681         -17424                -5743
## 8               02  04 [36,45)        -519                 11681         -12200                 -519
## 9               04   02 [8,16)       23645                 26811          -3166                23645
## 10              03  04 [36,45)        5290                 17490         -12200                 5290
## 11              03  03 [16,36)        9825                 17490          -7665                 9825
## 12              04  04 [36,45)       14611                 26811         -12200                14611
## 13              04 05 [45,Inf)        9387                 26811         -17424                 9387
## 14              04 01 (-Inf,8)       33428                 26811           6617                33428
## 15              03   02 [8,16)       14324                 17490          -3166                14324
## 16              02 01 (-Inf,8)       18298                 11681           6617                18298
## 17              03 01 (-Inf,8)       24107                 17490           6617                24107
## 18              01 05 [45,Inf)      -10807                  6617         -17424               -10807
## 19              01  04 [36,45)       -5583                  6617         -12200                -5583
```