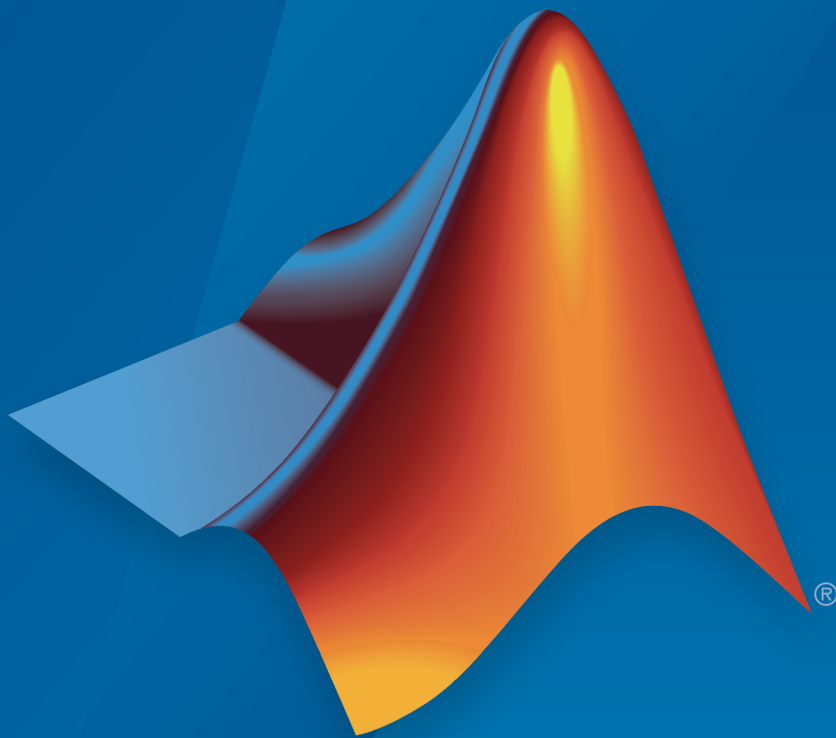


# Risk Management Toolbox™

## User's Guide



# MATLAB®

R2017a

 MathWorks®

## How to Contact MathWorks



Latest news: [www.mathworks.com](http://www.mathworks.com)  
Sales and services: [www.mathworks.com/sales\\_and\\_services](http://www.mathworks.com/sales_and_services)  
User community: [www.mathworks.com/matlabcentral](http://www.mathworks.com/matlabcentral)  
Technical support: [www.mathworks.com/support/contact\\_us](http://www.mathworks.com/support/contact_us)



Phone: 508-647-7000



The MathWorks, Inc.  
3 Apple Hill Drive  
Natick, MA 01760-2098

### *Risk Management Toolbox™ User's Guide*

© COPYRIGHT 2016 – 2017 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### **Trademarks**

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### **Patents**

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

### **Revision History**

September 2016	Online only	New for Version 1.0 (Release 2016b)
March 2017	Online only	Revised for Version 1.1 (Release 2017a)

## Getting Started

1

<b>Risk Management Toolbox Product Description</b> .....	1-2
Key Features .....	1-2
<b>Risk Modeling with Risk Management Toolbox</b> .....	1-3
Consumer Credit Risk .....	1-3
Corporate Credit Risk .....	1-3
Market Risk .....	1-5
<b>Credit Rating Migration Risk</b> .....	1-8
<b>Default Probability by Using the Merton Model for Structural Credit Risk</b> .....	1-11
<b>Concentration Indices</b> .....	1-14

## Market Risk Measurements Using VaR BackTesting Tools

2

<b>Overview of VaR Backtesting</b> .....	2-2
Binomial Test .....	2-3
Traffic Light Test .....	2-3
Kupiec's POF and TUFF Tests .....	2-4
Christoffersen's Interval Forecast Tests .....	2-5
Haas's Time Between Failures or Mixed Kupiec's Test .....	2-6
<b>VaR Backtesting Workflow</b> .....	2-8

Value-at-Risk Estimation and Backtesting .....	2-13
--	------

## Managing Consumer Credit Risk Using the Binning Explorer for Credit Scorecards

### 3

<b>Overview of Binning Explorer .....</b>	<b>3-2</b>
<b>Common Binning Explorer Tasks .....</b>	<b>3-4</b>
Import Data .....	3-4
Change Predictor Type .....	3-5
Change Binning Algorithm for One or More Predictors .....	3-6
Change Algorithm Options for Binning Algorithms .....	3-7
Split Bins for a Numeric Predictor .....	3-11
Split Bins for a Categorical Predictor .....	3-12
Manual Binning to Merge Bins for a Numeric or Categorical Predictor .....	3-14
Change Bin Boundaries for a Single Predictor .....	3-16
Change Bin Boundaries for Multiple Predictors .....	3-17
Set Options for Display .....	3-19
Export and Save the Binning .....	3-19
Troubleshoot the Binning .....	3-20
<b>Binning Explorer Case Study Example .....</b>	<b>3-26</b>
<b>Stress Testing of Consumer Credit Default Probabilities Using Panel Data .....</b>	<b>3-44</b>

## Corporate Credit Risk Simulations for Portfolios

### 4

<b>Credit Simulation Using Copulas .....</b>	<b>4-2</b>
Factor Models .....	4-3
Supported Simulations .....	4-3
<b>creditDefaultCopula Simulation Workflow .....</b>	<b>4-6</b>

<b>creditMigrationCopula Simulation Workflow . . . . .</b>	<b>4-12</b>
<b>Modeling Correlated Defaults with Copulas . . . . .</b>	<b>4-21</b>
<b>Analyze the Sensitivity of Concentration to a Given Exposure . . . . .</b>	<b>4-35</b>
<b>Compare Concentration Indices for Random Portfolios . . .</b>	<b>4-38</b>
<b>Comparison of the Merton Model Single-Point Approach to the Time-Series Approach . . . . .</b>	<b>4-42</b>

## Functions — Alphabetical List

**5**



# Getting Started

---

- “Risk Management Toolbox Product Description” on page 1-2
- “Risk Modeling with Risk Management Toolbox” on page 1-3
- “Credit Rating Migration Risk” on page 1-8
- “Default Probability by Using the Merton Model for Structural Credit Risk” on page 1-11
- “Concentration Indices” on page 1-14

## **Risk Management Toolbox Product Description**

### **Develop risk models and perform risk simulation**

Risk Management Toolbox provides functions for mathematical modeling and simulation of credit and market risk. You can model probabilities of default, create credit scorecards, perform credit portfolio analysis, and backtest models to assess potential for financial loss. The toolbox lets you assess corporate and consumer credit risk as well as market risk. It includes an app for automatic and manual binning of variables for credit scorecards. It also includes simulation tools to analyze credit portfolio risk and backtesting tools to evaluate Value-at-Risk (VaR).

### **Key Features**

- Binning explorer app for developing credit scorecards
- Copula-based simulation tools for portfolios of credit instruments
- Value-at-Risk (VaR) backtesting models for assessing market risk



## Risk Modeling with Risk Management Toolbox

### In this section...

“Consumer Credit Risk” on page 1-3

“Corporate Credit Risk” on page 1-3

“Market Risk” on page 1-5

Risk Management Toolbox provides tools for modeling three areas of risk assessment:

- Consumer credit risk
- Corporate credit risk
- Market risk

### Consumer Credit Risk

*Consumer credit risk* (also referred to as *retail credit risk*) is the risk of loss due to a customer's default (non-repayment) on a consumer credit product. These products can include a mortgage, unsecured personal loan, credit card, or overdraft. A common method for predicting credit risk is through a credit scorecard. The scorecard is a statistically based model for attributing a score to a customer that indicates the predicted probability that the customer will default. The data used to calculate the score can be from sources such as application forms, credit reference agencies, or products the customer already holds with the lender. Financial Toolbox™ provides tools for creating credit scorecards and performing credit portfolio analysis using scorecards. Risk Management Toolbox includes a Binning Explorer app for automatic or manual binning to streamline the binning phase of credit scorecard development. For more information, see “Overview of Binning Explorer” on page 3-2.

### Corporate Credit Risk

*Corporate credit risk* (also referred to as *wholesale credit risk*) is the risk that counterparties default on their financial obligations.

At an individual counterparty level, one of the main credit risk parameters is the probability of default (PD). Risk Management Toolbox allows users to estimate probabilities of default using the following methodologies:

- Structural models: `mertonmodel` and `mertonByTimeSeries`

- Reduced-form models: `cdsbootstrap` and `bondDefaultBootstrap` using Financial Toolbox
- Historical credit ratings migrations: `transprob` using Financial Toolbox
- Statistical approaches: credit scorecards using Binning Explorer and the `creditscorecard` object using Financial Toolbox, and a wide selection of predictive models in Statistics and Machine Learning Toolbox™

At a *credit portfolio* level, on the other hand, to assess credit risk, to assess this risk, the main question to ask is, Given a current credit portfolio, how much can be lost in a given time period due to defaults? In differing circumstances, the answer to this question might mean:

- How much do you expect to lose?
- How likely is it that you will lose more than a specific amount?
- What is the most you can lose under relatively normal circumstances?
- How much can you lose if things get bad?

Mathematically, these questions all depend on estimating a distribution of losses for the credit portfolio: What are the different amounts you can lose, and how likely is it that you lose each individual amount.

Corporate credit risk is fundamentally different from market risk, which is the risk that assets lose value due to market movements. The most important difference is that markets move all the time, but defaults occur infrequently. Therefore, the sample sizes to support any modeling efforts are different. The challenge is to calibrate a distribution of credit losses, because the sample sizes are small. For credit risk, even for an individual bond that has not defaulted, you cannot collect direct data on what happens in the event of default because it has not defaulted. And once the issuer actually defaults, unless you can pool default information from similar companies, that is the only data point that you have.

For corporate credit portfolio analysis, estimating credit correlations so that you can understand the benefits of diversification is also challenging. Two companies can only default in the same time window once, so you cannot collect data on how often they default together. To collect more data, you can pool data from similar companies and under similar economic conditions.

Risk Management Toolbox provides a credit default simulation framework for credit portfolios using the `creditDefaultCopula` object, where the three main elements of credit risk for a single instrument are:

- The probability of default (PD) which is the likelihood that the issuer defaults in a given time period.
- The exposure at default (EAD) which is the amount of money that is at stake. For a traditional bond, this is the bond principal.
- The loss given default (LGD) which is the fraction of the exposure that would be lost at default. When default occurs, usually some money is recovered eventually.

The assumption is that these three quantities are fixed and known for all the companies in the credit portfolio. With this assumption, the only uncertainty is whether each company defaults, which happens with probability  $PD_i$ .

At the credit portfolio level, however, the main question is, What are the default correlations between issuers? For example, for two bonds with 10MM principal each, the risk is different if you expect the companies to default together. In this scenario, you could lose 20MM minus the recovery, all at once. Alternatively, if the defaults are independent, you could lose 10MM minus recovery if one defaults, but the other company is likely still alive. Default correlations are therefore important parameters for understanding the risk at a portfolio level. These parameters are also important for understanding the diversification and concentration characteristics of the portfolio. The approach in Risk Management Toolbox is to simulate correlated variables that can be efficiently simulated and parameterized, then map the simulated values to default or nondefault states to preserve the individual default probabilities. This approach is called a *copula*. When normal variables are used, this approach is called a *Gaussian copula*. Risk Management Toolbox also provides a credit migration simulation framework for credit portfolios using the `creditMigrationCopula` object. For more information, see “Credit Rating Migration Risk” on page 1-8. Risk Management Toolbox also provides tools for portfolio concentration analysis, see “Concentration Indices” on page 1-14.

## Market Risk

*Market risk* is the risk of losses in positions arising from movements in market prices. Value-at-risk is a statistical method that quantifies the risk level associated with a portfolio. VaR measures the maximum amount of loss over a specified time horizon, at a given confidence level. For example, if the one-day 95% VaR of a portfolio is 10MM, then there is a 95% chance that the portfolio loses less than 10MM the following day. In other words, only 5% of the time (or about once in 20 days) the portfolio losses exceed 10MM.

*Backtesting*, on the other hand, measures how accurate the VaR calculations are. For many portfolios, especially trading portfolios, VaR is computed daily. At the closing of the following day, the actual profits and losses for the portfolio are known, and can be

compared to the VaR estimated the day before. You can use this daily data to assess the performance of VaR models, which is the goal of VaR backtesting. As such, backtesting is a method that looks retrospectively at data and refines the VaR models. Many VaR backtesting methodologies have been proposed. As a best practice, use more than one criterion to backtest the performance of VaR models, because all tests have strengths and weaknesses.

Risk Management Toolbox provides the following VaR backtesting individual tests:

- Traffic light test (`tl`)
- Binomial test (`bin`)
- Kupiec's tests (`pof`, `tuff`)
- Christoffersen's tests (`cc`, `cci`)
- Haas's tests (`tbf`, `tbfi`)

For information on the different tests, see “Overview of VaR Backtesting” on page 2-2.

## See Also

`bin` | `cc` | `cci` | `concentrationIndices` | `creditDefaultCopula` | `creditMigrationCopula` | `mertonByTimeSeries` | `mertonmodel` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` | `varbacktest`

## Related Examples

- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-26
- “`creditMigrationCopula` Simulation Workflow” on page 4-12
- “`creditDefaultCopula` Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-44
- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## More About

- “Credit Simulation Using Copulas” on page 4-2

- “Credit Rating Migration Risk” on page 1-8
- “Default Probability by Using the Merton Model for Structural Credit Risk” on page 1-11
- “Concentration Indices” on page 1-14
- “Traffic Light Test” on page 2-3
- “Binomial Test” on page 2-3
- “Kupiec’s POF and TUFF Tests” on page 2-4
- “Christoffersen’s Interval Forecast Tests” on page 2-5
- “Haas’s Time Between Failures or Mixed Kupiec’s Test” on page 2-6

### **External Websites**

- Credit Scorecard Modeling Using the Binning Explorer App (6 min 30 sec)
- Credit Risk Modeling with MATLAB (53 min 09 sec)

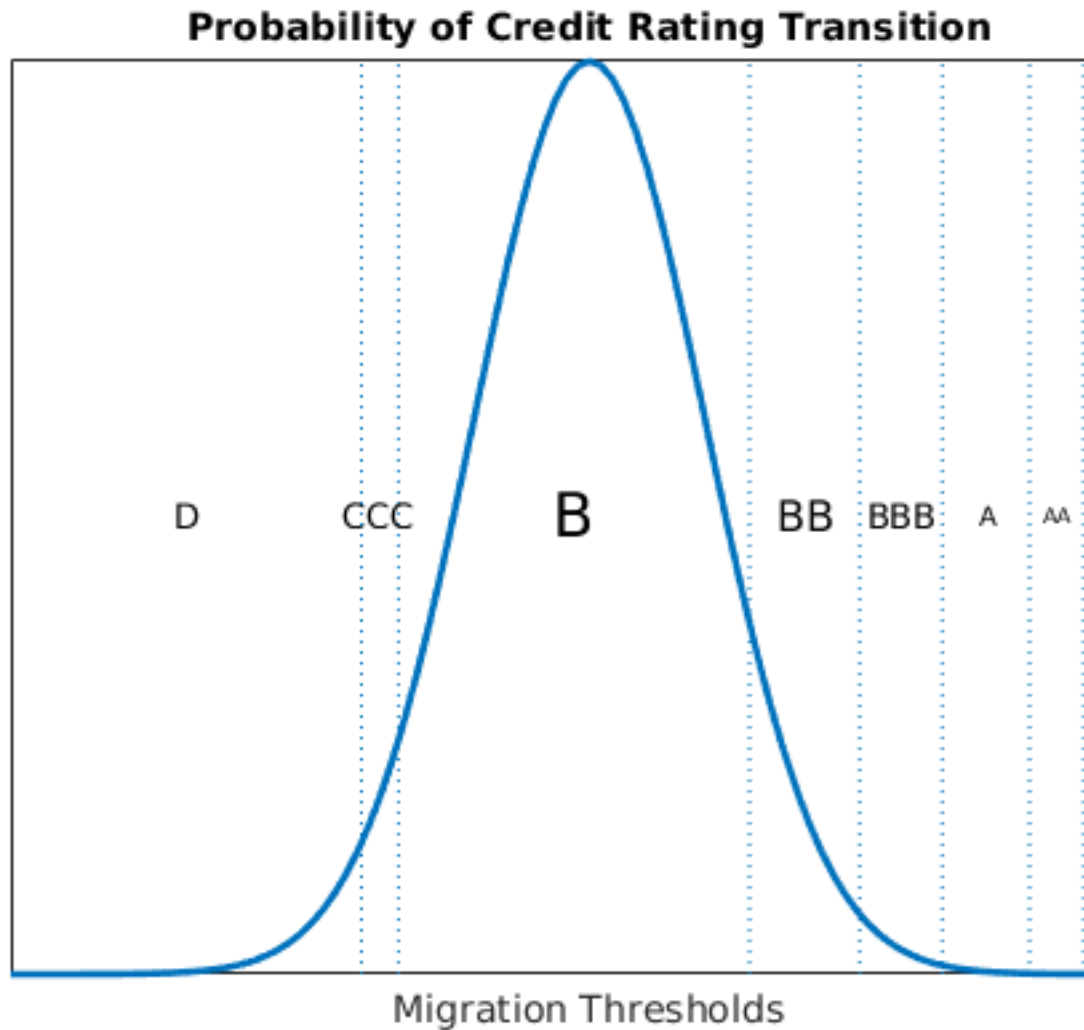
## Credit Rating Migration Risk

The migration-based multi-factor copula (`creditMigrationCopula`) is similar to the `creditDefaultCopula` object. As described in “Credit Simulation Using Copulas” on page 4-2, each counterparty’s credit quality is represented by a “latent variable” which is simulated over many scenarios. The latent variable is composed of a series of correlated factors which are weighted based on the counterparty’s sensitivity to each factor. The two objects differ in how the latent variables are used for the remainder of the analysis. Instead of thinking in terms of probability of default for each obligor, the `creditMigrationCopula` object works with each obligor’s credit rating. Credit ratings are issued by several companies (S&P, Moodys, and so on). Each rating represents a level of credit quality and ratings are changed periodically as a company’s situation improves or deteriorates.

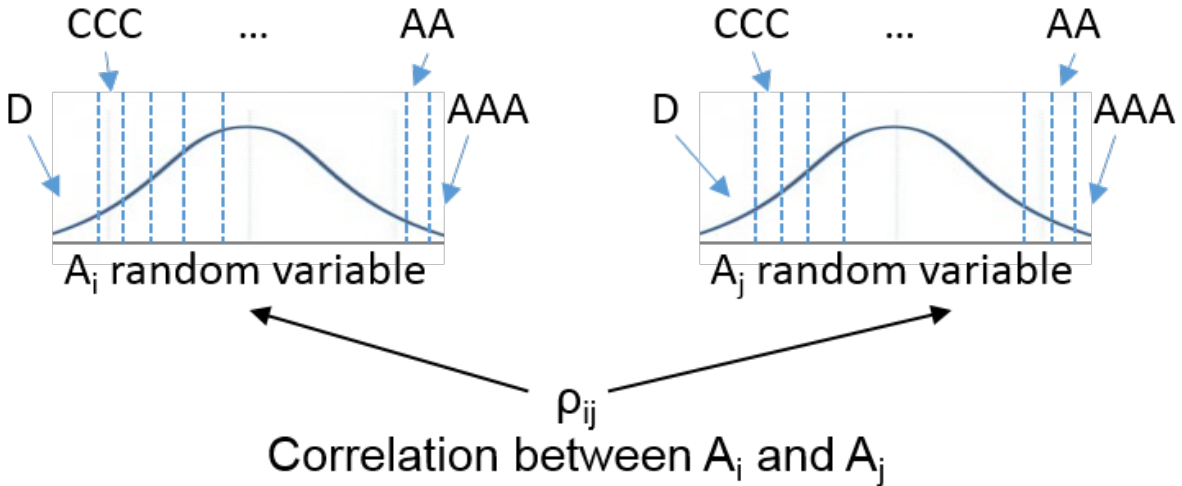
Given enough historical data, the likelihood is calculated that a company at a particular rating will migrate to a different rating over some time period. For example, this table shows the probabilities that a company with credit rating “B” will transition to each other rating.

New Rating	Probability (%)
AAA	0.001
AA	0.0062
A	0.1081
BBB	0.8697
BB	7.3366
B	86.7215
CCC	2.5169
Default	2.4399

While the `creditDefaultCopula` object is concerned with the 2.4% chance of default exclusively, a migration-based approach using an `creditMigrationCopula` object accounts for all possible rating states. Given these probabilities, the cut-points are calculated for the distribution of all possible latent variable values that correspond to each rating value.



For each scenario, the latent variable value determines the credit rating of the counterparty at the end of the time period based on these cut-points. The cut-points are set such that the probability of transitioning to each rating matches the probabilities in the provided transition table. You now have not just correlated defaults for each counterparty, but correlated rating changes across the entire range of credit ratings.



Each credit rating has a unique discount curve associated with it. As an obligor’s credit rating falls, the obligor’s bond cashflows become more deeply discounted and the total bond value drops accordingly. Conversely, if an obligor’s rating improves, the cashflows are discounted less deeply, and the bond values will rise. After repricing the portfolio with all obligors’ new ratings, the total portfolio value can be calculated as the sum of the new bond values. As with the `creditDefaultCopula` object, various risk measures are calculated and reported for the `creditMigrationCopula` object.

### See Also

`confidenceBands` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate`

### Related Examples

- “`creditMigrationCopula` Simulation Workflow” on page 4-12

### More About

- `creditMigrationCopula`



## Default Probability by Using the Merton Model for Structural Credit Risk

In 1974, Robert Merton proposed a model for assessing the structural credit risk of a company by modeling the company's equity as a call option on its assets. This method allows for the use of the Black-Scholes-Merton option pricing methods. The Merton model is structural because it provides a relationship between the default risk and the asset (capital) structure of the firm.

From an accounting standpoint, the book value of a firm's equity,  $E$ , total assets,  $A$ , and total liabilities,  $L$ , are defined by the equation:

$$A = E + L$$

These book values for  $E$ ,  $A$ , and  $L$  are all observable because they are recorded on a balance sheet. However, the book values are reported infrequently. On the other hand, only the equity's market value is observable, given by the firm's stock market price times the number of outstanding shares. The market value of the firm's assets and total liabilities are unobservable.

Merton's model relates the market value of assets, equity, and liabilities in an option pricing framework. The Merton model assumes a single liability  $L$  with maturity  $T$ , usually within one year. At time  $T$ , the firm's value to the shareholders equals the difference  $A - L$  when the assets value  $A$  is greater than the liabilities  $L$ . However, if the debt  $L$  exceeds the asset value  $A$ , then the shareholders get nothing. The value of the equity  $E_T$  at time  $T$  is related to the value of the assets and liabilities by the following formula:

$$E_T = \max(A_T - L, 0)$$

In practice, firms have multiple maturities for their liabilities, so for a selected maturity  $T$ , a liability threshold  $L$  is chosen based on the whole liability structure of the firm. The liability threshold is also referred to as the default point. For a typical time horizon of one year, the liability threshold is commonly set somewhere between the value of the short-term liabilities and the value of the total liabilities.

Assuming a lognormal distribution for the asset returns, you can use the Black-Scholes-Merton equations to relate the observable market value of equity  $E$ , and the unobservable market value of assets  $A$  at any time prior to the maturity  $T$ :

$$E = AN(d_1) - Le^{-rT}N(d_2)$$

where  $r$  is the risk-free interest rate,  $N$  is the cumulative standard normal distribution, and  $d_1$  and  $d_2$  are given by

$$d_1 = \frac{\ln\left(\frac{A}{L}\right) + (r + 0.5\sigma_A^2)T}{\sigma_A\sqrt{T}}$$

$$d_2 = d_1 - \sigma_A\sqrt{T}$$

You can solve equation using two approaches:

- The `mertonmodel` approach, a single point calibration that requires equity, liability, and equity volatility ( $\sigma_E$ ).

This approach solves for  $(A, \sigma_A)$  using a 2-by-2 system of nonlinear equations. The first equation is the preceding option pricing formula. The second equation relates the unobservable volatility of assets  $\sigma_A$  to the given equity volatility  $\sigma_E$  through the equation

$$\sigma_E = \frac{A}{E}N(d_1)\sigma_A$$

- The `mertonByTimeSeries` approach, which requires a time series for the equity and for all other model parameters.

If the equity time series has  $n$  data points, this approach calibrates a time series of  $n$  asset values  $A_1, \dots, A_n$  that solve the following system of equations:

$$E_1 = A_1N(d_1) - L_1e^{-r_1T_1}N(d_2)$$

...

$$E_n = A_nN(d_1) - L_n e^{-r_n T_n} N(d_2)$$

The volatility of assets  $\sigma_A$  is directly computed from the time series  $A_1, \dots, A_n$  as the annualized standard deviation of the log returns. This value is a single volatility

value that captures the volatility of the assets during the time period spanned by the time series.

Once the values of  $A$  and  $\sigma_A$  are known, the *distance to default* ( $DD$ ) is computed as the number of standard deviations between the expected asset value at maturity  $T$  and the liability threshold:

$$DD = \frac{\log A + (\mu_A - \sigma_A^2 / 2)T - \log(L)}{\sigma_A \sqrt{T}}$$

The *drift* parameter  $\mu_A$  is the expected return for the assets, which can be equal to the risk-free interest rate, or any other value based on expectations for that firm.

The *probability of default* ( $PD$ ) is defined as the probability that the asset value falls below the liability threshold at the end of the time horizon  $T$ :

$$PD = 1 - N(DD)$$

## See Also

`mertonByTimeSeries` | `mertonmodel`

## Related Examples

- “Comparison of the Merton Model Single-Point Approach to the Time-Series Approach” on page 4-42

## Concentration Indices

In financial risk applications, *concentration* is the opposite of *diversification*. If all or most of your risk is in one area, it is concentrated. Higher concentration is interpreted as a risk, although for someone with a high tolerance for risk and who wants higher returns, that person might prefer concentration.

You can use concentration indices to measure and monitor concentration in a credit portfolio. Ad-hoc concentration indices are typically computed by using exposures, and therefore do not usually take into account other risk parameters such as probabilities of default. Ad-hoc concentration indices are frequently included in comprehensive concentration reports, with other concentration measures and concentration limits.

When you use the `concentrationIndices` function, Risk Management Toolbox supports the following ad-hoc concentration indices or measures:

- Concentration ratio
- Deciles of the portfolio weight distribution
- Gini coefficient
- Herfindahl-Hirschman index
- Hannah-Kay index
- Hall-Tideman index
- Theil entropy index

### See Also

`concentrationIndices`

### Related Examples

- “Analyze the Sensitivity of Concentration to a Given Exposure” on page 4-35
- “Compare Concentration Indices for Random Portfolios” on page 4-38

# Market Risk Measurements Using VaR BackTesting Tools

---

- “Overview of VaR Backtesting” on page 2-2
- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Overview of VaR Backtesting

In this section...
“Binomial Test” on page 2-3
“Traffic Light Test” on page 2-3
“Kupiec’s POF and TUFF Tests” on page 2-4
“Christoffersen’s Interval Forecast Tests” on page 2-5
“Haas’s Time Between Failures or Mixed Kupiec’s Test” on page 2-6

*Market risk* is the risk of losses in positions arising from movements in market prices. Value-at-risk (VaR) is one of the main measures of financial risk. VaR is an estimate of how much value a portfolio can lose in a given time period with a given confidence level. For example, if the 1-day 95% VaR of a portfolio is 10MM, then there is a 95% chance that the portfolio loses less than 10MM the following day. In other words, only 5% of the time (or about once in 20 days) the portfolio losses exceed 10MM.

For many portfolios, especially trading portfolios, VaR is computed daily. At the closing of the following day, the actual profits and losses for the portfolio are known and can be compared to the VaR estimated the day before. You can use this daily data to assess the performance of VaR models, which is the goal of VaR backtesting. The performance of VaR models can be measured in different ways. In practice, many different metrics and statistical tests are used to identify VaR models that are performing poorly or performing better. As a best practice, use more than one criterion to backtest the performance of VaR models, because all tests have strengths and weaknesses.

Suppose that you have VaR limits and corresponding returns or profits and losses for days  $t = 1, \dots, N$ . Use  $VaR_t$  to denote the VaR estimate for day  $t$  (determined on day  $t - 1$ ). Use  $R_t$  to denote the actual return or profit and loss observed on day  $t$ . Profits and losses are expressed in monetary units and represent value changes in a portfolio. The corresponding VaR limits are also given in monetary units. Returns represent the change in portfolio value as a proportion (or percentage) of its value on the previous day. The corresponding VaR limits are also given as a proportion (or percentage). The VaR limits must be produced from existing VaR models. Then, to perform a VaR backtesting analysis, provide these limits and their corresponding returns as data inputs to the VaR backtesting tools in Risk Management Toolbox.

The toolbox supports these VaR backtests:

- Binomial test

- Traffic light test
- Kupiec's tests
- Christoffersen's tests
- Haas's tests

## Binomial Test

The most straightforward test is to compare the observed number of exceptions,  $x$ , to the expected number of exceptions. From the properties of a binomial distribution, you can build a confidence interval for the expected number of exceptions. Using exact probabilities from the binomial distribution or a normal approximation, the `bin` function uses a normal approximation. By computing the probability of observing  $x$  exceptions, you can compute the probability of wrongly rejecting a good model when  $x$  exceptions occur. This is the  $p$ -value for the observed number of exceptions  $x$ . For a given test confidence level, a straightforward accept-or-reject result in this case is to fail the VaR model whenever  $x$  is outside the test confidence interval for the expected number of exceptions. "Outside the confidence interval" can mean too many exceptions, or too few exceptions. Too few exceptions might be a sign that the VaR model is too conservative.

The test statistic is

$$Z_{bin} = \frac{x - Np}{\sqrt{Np(1-p)}}$$

where  $x$  is the number of failures,  $N$  is the number of observations, and  $p = 1 - \text{VaR level}$ . The binomial test is approximately distributed as a standard normal distribution.

For more information, see Bibliography for Jorion and `bin`.

## Traffic Light Test

A variation on the binomial test proposed by the Basel Committee is the *traffic light test* or *three zones test*. For a given number of exceptions  $x$ , you can compute the probability of observing up to  $x$  exceptions. That is, any number of exceptions from 0 to  $x$ , or the cumulative probability up to  $x$ . The probability is computed using a binomial distribution. The three zones are defined as follows:

- The “red” zone starts at the number of exceptions where this probability equals or exceeds 99.99%. It is unlikely that too many exceptions will come from a correct VaR model.
- The “yellow” zone covers the number of exceptions where the probability equals or exceeds 95% but is smaller than 99.99%. Even though there high number of violations, the violation count is not exceedingly high.
- Everything below the yellow zone is "green." If you have too few failures, they fall in the green zone. Only too many failures lead to model rejections.

For more information, see Bibliography for Basel Committee on Banking Supervision and t1.

### Kupiec’s POF and TUFF Tests

Kupiec (1995) introduced a variation on the binomial test called the proportion of failures (POF) test. The POF test works with the binomial distribution approach. In addition, it uses a likelihood ratio to test whether the probability of exceptions is synchronized with the probability  $p$  implied by the VaR confidence level. If the data suggests that the probability of exceptions is different than  $p$ , the VaR model is rejected. The POF test statistic is

$$LR_{POF} = -2 \log \left( \frac{(1-p)^{N-x} p^x}{\left(1 - \frac{x}{N}\right)^{N-x} \left(\frac{x}{N}\right)^x} \right)$$

where  $x$  is the number of failures,  $N$  the number of observations and  $p = 1 - \text{VaR level}$ .

This statistic is asymptotically distributed as a chi square variable with one degree of freedom. The VaR model fails the test if this likelihood ratio exceeds a critical value. The critical value depends on the test confidence level.

Kupiec also proposed a second test called the time until first failure (TUFF). The TUFF test looks at when the first rejection occurred. If it happens too soon, the test fails the VaR model. Checking only the first exception leaves much information out, specifically, whatever happened after the first exception is ignored. The TBFi test extends the TUFF approach to include all the failures. See `tbf i`.



The TUFF test is also based on a likelihood ratio, but the underlying distribution is a geometric distribution. If  $n$  is the number of days until the first rejection, the test statistic is given by

$$LR_{TUFF} = -2 \log \left( \frac{p(1-p)^{n-1}}{\left(\frac{1}{n}\right) \left(1 - \frac{1}{n}\right)^{n-1}} \right)$$

This statistic is asymptotically distributed as a chi square variable with one degree of freedom. For more information, see Bibliography for Kupiec, `pof`, and `tuff`.

## Christoffersen's Interval Forecast Tests

Christoffersen (1998) proposed a test to measure whether the probability of observing an exception on a particular day depends on whether an exception occurred. Unlike the unconditional probability of observing an exception, Christoffersen's test measures the dependency between consecutive days only. The test statistic for independence in Christoffersen's interval forecast (IF) approach is given by

$$LR_{CCI} = -2 \log \left( \frac{(1-\pi)^{n00+n10} \pi^{n01+n11}}{\left(1-\pi_0\right)^{n00} \pi_0^{n01} \left(1-\pi_1\right)^{n10} \pi_1^{n11}} \right)$$

where

- $n00$  = Number of periods with no failures followed by a period with no failures.
- $n10$  = Number of periods with failures followed by a period with no failures.
- $n01$  = Number of periods with no failures followed by a period with failures.
- $n11$  = Number of periods with failures followed by a period with failures.

and

- $\pi_0$  — Probability of having a failure on period  $t$ , given that no failure occurred on period  $t - 1 = n01 / (n00 + n01)$
- $\pi_1$  — Probability of having a failure on period  $t$ , given that a failure occurred on period  $t - 1 = n11 / (n10 + n11)$
- $\pi$  — Probability of having a failure on period  $t = (n01 + n11) / (n00 + n01 + n10 + n11)$

This statistic is asymptotically distributed as a chi square with one degree of freedom. You can combine this statistic with the frequency POF test to get a conditional coverage (CC) mixed test:

$$LR_{CC} = LR_{POF} + LR_{CCI}$$

This test is asymptotically distributed as a chi square variable with two degrees of freedom.

For more information, see Bibliography for Christoffersen, cc, and cci.

### Haas's Time Between Failures or Mixed Kupiec's Test

Haas (2001) extended Kupiec's TUFF test to incorporate the time information between all the exceptions in the sample. Haas's test applies the TUFF test to each exception in the sample and aggregates the time between failures (TBF) test statistic.

$$LR_{TBF} = -2 \sum_{i=1}^x \log \left( \frac{p(1-p)^{n_i-1}}{\binom{1}{n_i} \left(1 - \frac{1}{n_i}\right)^{n_i-1}} \right)$$

In this statistic,  $p = 1 - \text{VaR level}$  and  $n_i$  is the number of days between failures  $i-1$  and  $i$  (or until the first exception for  $i = 1$ ). This statistic is asymptotically distributed as a chi square variable with  $x$  degrees of freedom, where  $x$  is the number of failures.

Like Christoffersen's test, you can combine this test with the frequency POF test to get a TBF mixed test, sometimes called Haas' mixed Kupiec's test:

$$LR_{TBF} = LR_{POF} + LR_{TBF}$$

This test is asymptotically distributed as a chi square variable with  $x+1$  degrees of freedom. For more information, see Bibliography for Haas, tbf, and tbf i.

### References

Basel Committee on Banking Supervision, *Supervisory framework for the use of "backtesting" in conjunction with the internal models approach to market risk capital requirements*. January 1996, <http://www.bis.org/publ/bcbs22.htm>.

Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841–862.

Cogneau, P. "Backtesting Value-at-Risk: how good is the model?" *Intelligent Risk*, PRMIA, July, 2015.

Haas, M. "New Methods in Backtesting." *Financial Engineering*, Research Center Caesar, Bonn, 2001.

Jorion, P. *Financial Risk Manager Handbook. 6th Edition*, Wiley Finance, 2011.

Kupiec, P. "Techniques for Verifying the Accuracy of Risk Management Models." *Journal of Derivatives*. Vol. 3, 1995, pp. 73–84.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management*. Princeton University Press, 2005.

Nieppola, O. "Backtesting Value-at-Risk Models." Master's Thesis, Helsinki School of Economics, 2009.

## See Also

[bin](#) | [cc](#) | [cci](#) | [pof](#) | [runtests](#) | [summary](#) | [tbf](#) | [tbfi](#) | [t1](#) | [tuff](#) | [varbacktest](#)

## Related Examples

- "VaR Backtesting Workflow" on page 2-8
- "Value-at-Risk Estimation and Backtesting" on page 2-13

## More About

- "Risk Modeling with Risk Management Toolbox" on page 1-3

# VaR Backtesting Workflow

This example shows a value-at-risk (VaR) backtesting workflow and the use of VaR backtesting tools. For a more comprehensive example of VaR backtesting, see “Value-at-Risk Estimation and Backtesting”.

### Step 1. Load the VaR backtesting data.

Use the `VaRBacktestData.mat` file to load the VaR data into the workspace. This example works with the `EquityIndex`, `Normal195`, and `Normal199` numeric arrays. These arrays are equity returns and the corresponding VaR data at 95% and 99% confidence levels is produced with a normal distribution (a variance-covariance approach). See “Value-at-Risk Estimation and Backtesting” for an example on how to generate this VaR data.

```
load('VaRBacktestData')
disp([EquityIndex(1:5) Normal195(1:5) Normal199(1:5)])

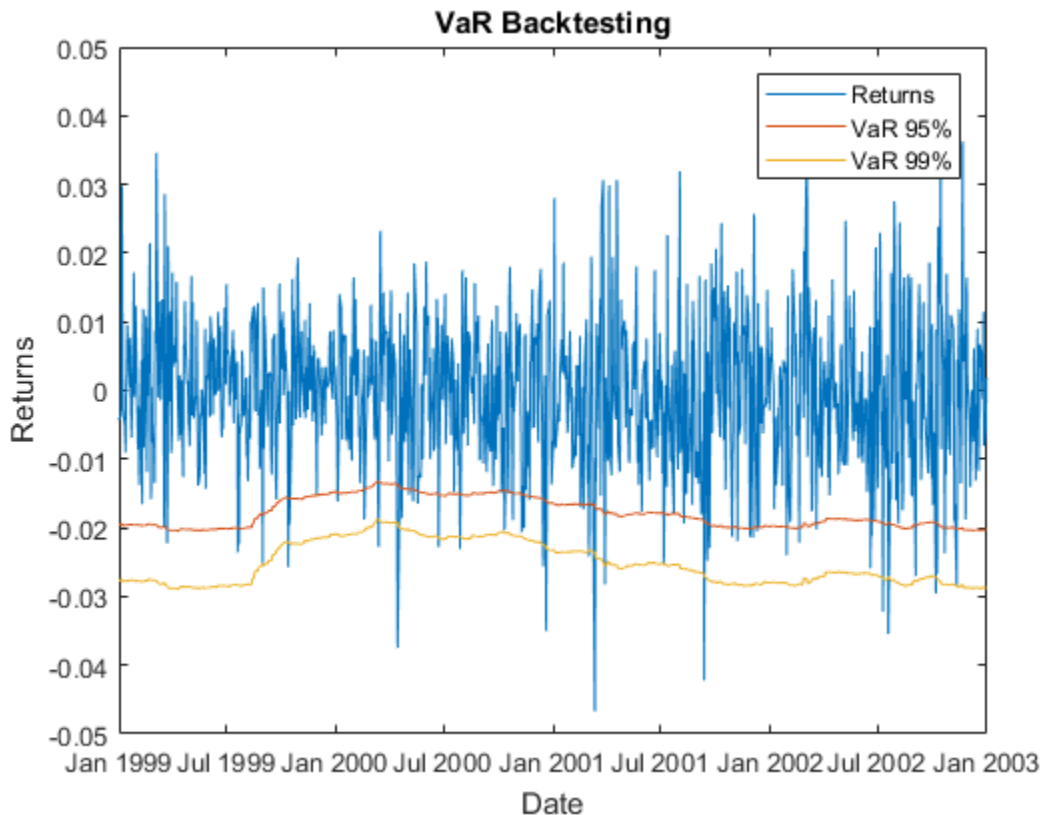
-0.0043    0.0196    0.0277
-0.0036    0.0195    0.0276
-0.0000    0.0195    0.0275
 0.0298    0.0194    0.0275
 0.0023    0.0197    0.0278
```

The first column shows three losses in the first three days, but none of these losses exceeds the corresponding VaR (columns 2 and 3). The VaR model fails whenever the loss (negative of returns) exceeds the VaR.

### Step 2. Generate a VaR backtesting plot.

Use the `plot` function to visualize the VaR backtesting data. This type of visualization is a common first step when performing a VaR backtesting analysis.

```
plot(Date,[EquityIndex -Normal195 -Normal199])
title('VaR Backtesting')
xlabel('Date')
ylabel('Returns')
legend('Returns', 'VaR 95%', 'VaR 99%')
```



### Step 3. Create a varbacktest object.

Create a varbacktest object for the equity returns and the VaRs at 95% and 99% confidence levels.

```
vbt = varbacktest(EquityIndex,[Normal195 Normal199],...
    'PortfolioID','S&P', ...
    'VaRID',{'Normal195' 'Normal199'}, ...
    'VaRLevel',[0.95 0.99]);
disp(vbt)
```

varbacktest with properties:

```
PortfolioData: [1043×1 double]
VaRData: [1043×2 double]
```

```
PortfolioID: "S&P"  
  VaRID: [ "Normal195"  "Normal199" ]  
  VaRLevel: [0.9500 0.9900]
```

### Step 4. Run a summary report.

Use the `summary` function to obtain a summary for the number of observations, the number of failures, and other simple metrics.

```
summary(vbt)
```

```
ans =
```

```
2x10 table
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Failures
"S&P"	"Normal195"	0.95	0.94535	1043	57
"S&P"	"Normal199"	0.99	0.9837	1043	17

### Step 5. Run all tests.

Use the `runtests` function to display the final test results all at once.

```
runtests(vbt)
```

```
ans =
```

```
2x11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF	
"S&P"	"Normal195"	0.95	green	accept	accept	accept	ac
"S&P"	"Normal199"	0.99	yellow	reject	accept	accept	ac

### Step 6. Run individual tests.

After running all tests, you can investigate the details of particular tests. For example, use the `t1` function to run the traffic light test.

```
t1(vbt)
```

```
ans =
```

```
2x9 table
```

PortfolioID	VaRID	VaRLevel	TL	Probability	TypeI	Increase
"S&P"	"Normal195"	0.95	green	0.77913	0.26396	
"S&P"	"Normal199"	0.99	yellow	0.97991	0.03686	0.265

### Step 7. Create VaR backtests for multiple portfolios.

You can create VaR backtests for different portfolios, or the same portfolio over different time windows. Run tests over two different subwindows of the original test window.

```
Ind1 = year(Date)<=2000;
```

```
Ind2 = year(Date)>2000;
```

```
vbt1 = varbacktest(EquityIndex(Ind1),[Normal195(Ind1,:) Normal199(Ind1:)],...
    'PortfolioID','S&P, 1999-2000',...
    'VaRID',{'Normal195' 'Normal199'},...
    'VaRLevel',[0.95 0.99]);
```

```
vbt2 = varbacktest(EquityIndex(Ind2),[Normal195(Ind2,:) Normal199(Ind2:)],...
    'PortfolioID','S&P, 2001-2002',...
    'VaRID',{'Normal195' 'Normal199'},...
    'VaRLevel',[0.95 0.99]);
```

### Step 8. Display a summary report for both portfolios.

Use the `summary` function to display a summary for both portfolios.

```
Summary = [summary(vbt1); summary(vbt2)];
disp(Summary)
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Fail
"S&P, 1999-2000"	"Normal195"	0.95	0.94626	521	28
"S&P, 1999-2000"	"Normal199"	0.99	0.98464	521	8
"S&P, 2001-2002"	"Normal195"	0.95	0.94444	522	29

```
"S&P, 2001-2002" "Normal99" 0.99 0.98276 522 9
```

**Step 9. Run all tests for both portfolios.**

Use the `runtests` function to display the final test result for both portfolios.

```
Results = [runtests(vbt1);runtests(vbt2)];
disp(Results)
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"S&P, 1999-2000"	"Normal95"	0.95	green	accept	accept	accept
"S&P, 1999-2000"	"Normal99"	0.99	green	accept	accept	accept
"S&P, 2001-2002"	"Normal95"	0.95	green	accept	accept	accept
"S&P, 2001-2002"	"Normal99"	0.99	yellow	accept	accept	accept

**See Also**

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `t1` | `tuff` | `varbacktest`

**Related Examples**

- “Overview of VaR Backtesting” on page 2-2
- “Value-at-Risk Estimation and Backtesting” on page 2-13

**More About**

- “Traffic Light Test” on page 2-3
- “Binomial Test” on page 2-3
- “Kupiec’s POF and TUFF Tests” on page 2-4
- “Christoffersen’s Interval Forecast Tests” on page 2-5
- “Haas’s Time Between Failures or Mixed Kupiec’s Test” on page 2-6



## Value-at-Risk Estimation and Backtesting

This example shows how to estimate the value-at-risk (VaR) using three methods, and how to perform a VaR backtesting analysis. The three methods are:

- 1 Normal distribution
- 2 Historical simulation
- 3 Exponential weighted moving average (EWMA)

Value-at-risk is a statistical method that quantifies the risk level associated with a portfolio. The VaR measures the maximum amount of loss over a specified time horizon and at a given confidence level.

Backtesting measures the accuracy of the VaR calculations. Using VaR methods, the loss forecast is calculated and then compared to the actual losses at the end of the next day. The degree of difference between the predicted and actual losses indicates whether the VaR model is underestimating or overestimating the risk. As such, backtesting looks retrospectively at data and helps to assess the VaR model.

The three estimation methods used in this example estimate the VaR at 95% and 99% confidence levels.

### Load the Data and Define the Test Window

Load the data. The data used in this example is from a time series of returns on the S&P index between 1993 and 2003.

```
load VaRExampleData.mat
Returns = tick2ret(sp);
DateReturns = dates(2:end);
SampleSize = length>Returns);
```

Define the estimation window as 250 trading days. The test window starts on the first day in 1996 and runs through the end of the sample.

```
TestWindowStart = find(year(DateReturns)==1996,1);
TestWindow = TestWindowStart : SampleSize;
EstimationWindowSize = 250;
```

For a VaR confidence level of 95% and 99%, set the complement of the VaR level.

```
pVaR = [0.05 0.01];
```

These values mean that there is at most a 5% and 1% probability, respectively, that the loss incurred will be greater than the maximum threshold (that is, greater than the VaR).

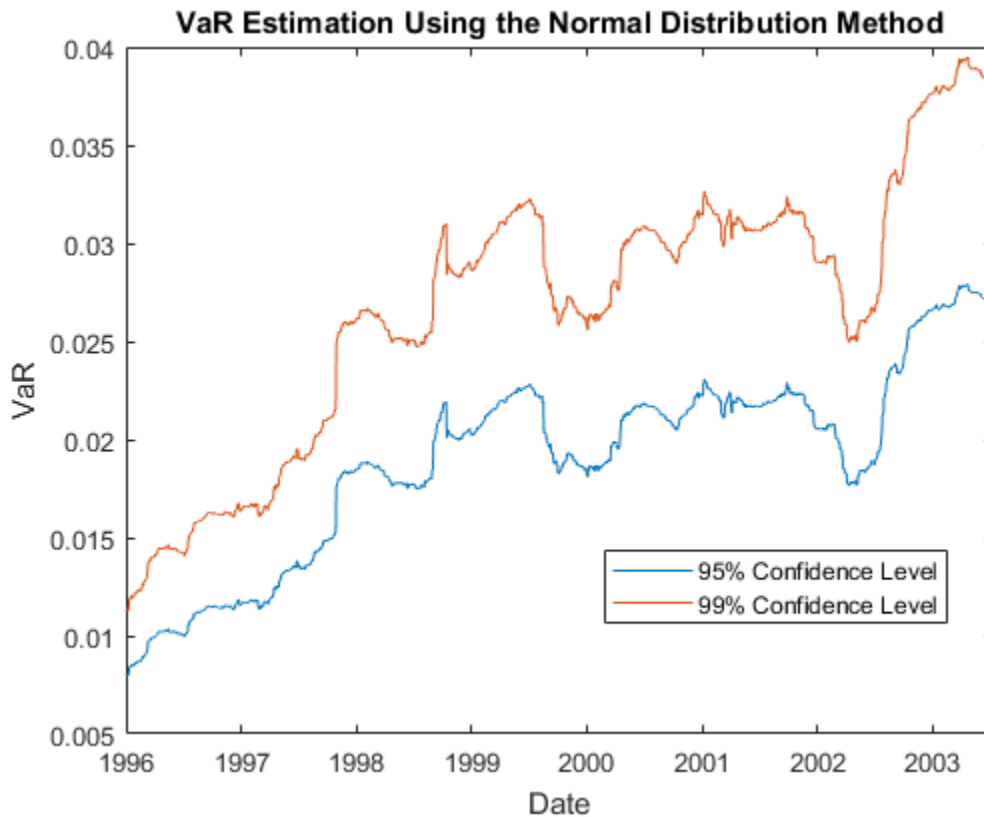
### Compute the VaR Using the Normal Distribution Method

For the normal distribution method, assume that the profit and loss of the portfolio is normally distributed. Using this assumption, compute the VaR by multiplying the  $z$ -score, at each confidence level by the standard deviation of the returns. Because VaR backtesting looks retrospectively at data, the VaR "today" is computed based on values of the returns in the last  $N = 250$  days leading to, but not including, "today."

```
Zscore = norminv(pVaR);
Normal95 = zeros(length(TestWindow),1);
Normal99 = zeros(length(TestWindow),1);

for t = TestWindow
    i = t - TestWindowStart + 1;
    EstimationWindow = t-EstimationWindowSize:t-1;
    Sigma = std>Returns(EstimationWindow));
    Normal95(i) = -Zscore(1)*Sigma;
    Normal99(i) = -Zscore(2)*Sigma;
end

figure;
plot(DateReturns(TestWindow),[Normal95 Normal99])
xlabel('Date')
ylabel('VaR')
legend({'95% Confidence Level','99% Confidence Level'},'Location','Best')
title('VaR Estimation Using the Normal Distribution Method')
```



The normal distribution method is also known as parametric VaR because its estimation involves computing a parameter for the standard deviation of the returns. The advantage of the normal distribution method is its simplicity. However, the weakness of the normal distribution method is the assumption that returns are normally distributed. Another name for the normal distribution method is the variance-covariance approach.

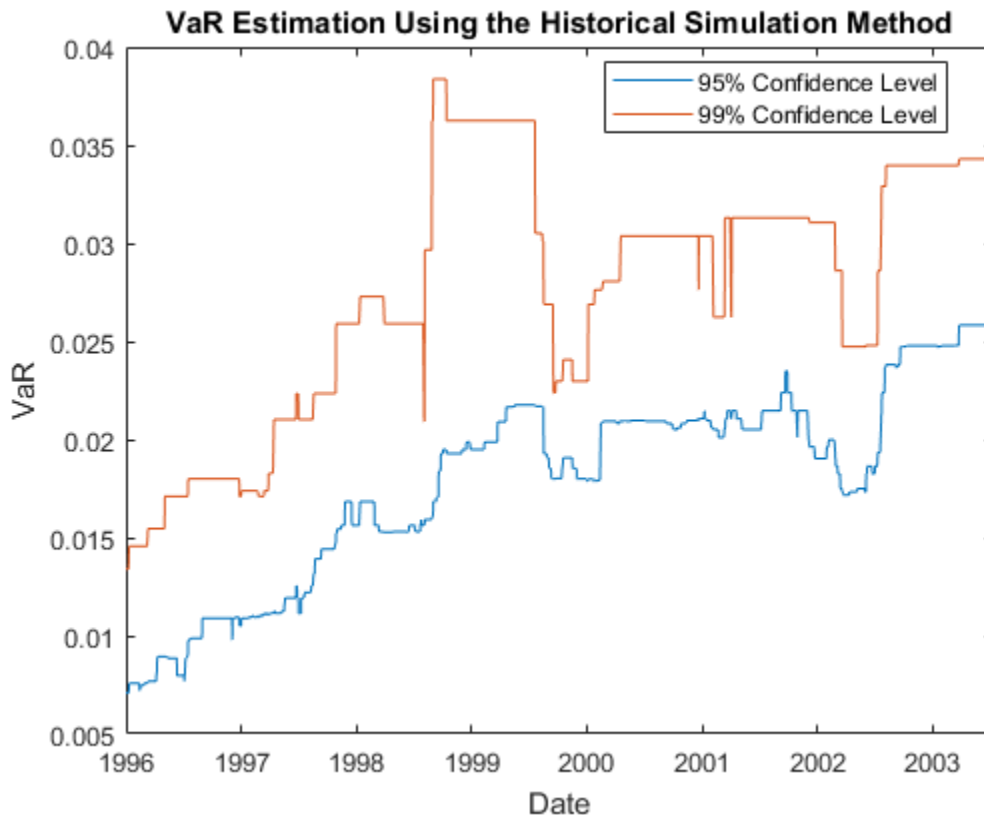
### Compute the VaR Using the Historical Simulation Method

Unlike the normal distribution method, the historical simulation (HS) is a nonparametric method. It does not assume a particular distribution of the asset returns. Historical simulation forecasts risk by assuming that past profits and losses can be used as the distribution of profits and losses for the next period of returns. The VaR "today" is computed as the  $p$  th-quantile of the last  $N$  returns prior to "today."

```
Historical95 = zeros(length(TestWindow),1);
Historical99 = zeros(length(TestWindow),1);

for t = TestWindow
    i = t - TestWindowStart + 1;
    EstimationWindow = t-EstimationWindowSize:t-1;
    X = Returns(EstimationWindow);
    Historical95(i) = -quantile(X,pVaR(1));
    Historical99(i) = -quantile(X,pVaR(2));
end

figure;
plot(DateReturns(TestWindow),[Historical95 Historical99])
ylabel('VaR')
xlabel('Date')
legend({'95% Confidence Level','99% Confidence Level'},'Location','Best')
title('VaR Estimation Using the Historical Simulation Method')
```



The preceding figure shows that the historical simulation curve has a piecewise constant profile. The reason for this is that quantiles do not change for several days until extreme events occur. Thus, the historical simulation method is slow to react to changes in volatility.

### Compute the VaR Using the Exponential Weighted Moving Average Method (EWMA)

The first two VaR methods assume that all past returns carry the same weight. The exponential weighted moving average (EWMA) method assigns nonequal weights, particularly exponentially decreasing weights. The most recent returns have higher weights because they influence "today's" return more heavily than returns further in the past. The formula for the EWMA variance over an estimation window of size  $W_E$  is:

$$\hat{\sigma}_t^2 = \frac{1}{c} \sum_{i=1}^{W_E} \lambda^{i-1} y_{t-i}^2$$

where  $c$  is a normalizing constant:

$$c = \sum_{i=1}^{W_E} \lambda^{i-1} = \frac{1 - \lambda^{W_E}}{1 - \lambda} \rightarrow \frac{1}{1 - \lambda} \text{ as } W_E \rightarrow \infty$$

For convenience, we assume an infinitely large estimation window to approximate the variance:

$$\hat{\sigma}_t^2 \approx (1 - \lambda)(y_{t-1}^2 + \sum_{i=2}^{\infty} \lambda^{i-1} y_{t-i}^2) = (1 - \lambda)y_{t-1}^2 + \lambda\hat{\sigma}_{t-1}^2$$

A value of the decay factor frequently used in practice is 0.94. This is the value used in this example. For more information, see References.

Initiate the EWMA using a warm-up phase to set up the standard deviation.

```
Lambda = 0.94;
Sigma2 = zeros(length>Returns,1);
Sigma2(1) =>Returns(1)^2;

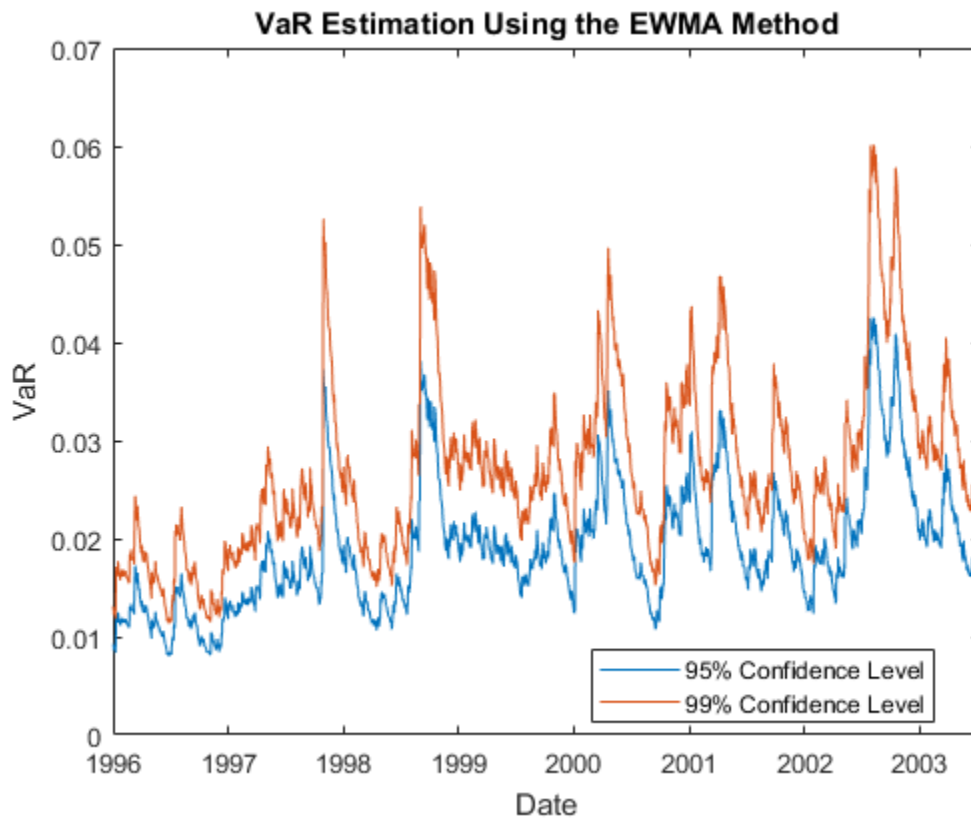
for i = 2 : (TestWindowStart-1)
    Sigma2(i) = (1-Lambda) *>Returns(i-1)^2 + Lambda * Sigma2(i-1);
end
```

Use the EWMA in the test window to estimate the VaR.

```
Zscore = norminv(pVaR);
EWMA95 = zeros(length(TestWindow),1);
EWMA99 = zeros(length(TestWindow),1);

for t = TestWindow
    k = t - TestWindowStart + 1;
    Sigma2(t) = (1-Lambda) *>Returns(t-1)^2 + Lambda * Sigma2(t-1);
    Sigma = sqrt(Sigma2(t));
    EWMA95(k) = -Zscore(1)*Sigma;
    EWMA99(k) = -Zscore(2)*Sigma;
end
```

```
figure;  
plot(DateReturns(TestWindow),[EWMA95 EWMA99])  
ylabel('VaR')  
xlabel('Date')  
legend({'95% Confidence Level','99% Confidence Level'},'Location','Best')  
title('VaR Estimation Using the EWMA Method')
```



In the preceding figure, the EWMA reacts very quickly to periods of large (or small) returns.

### VaR Backtesting

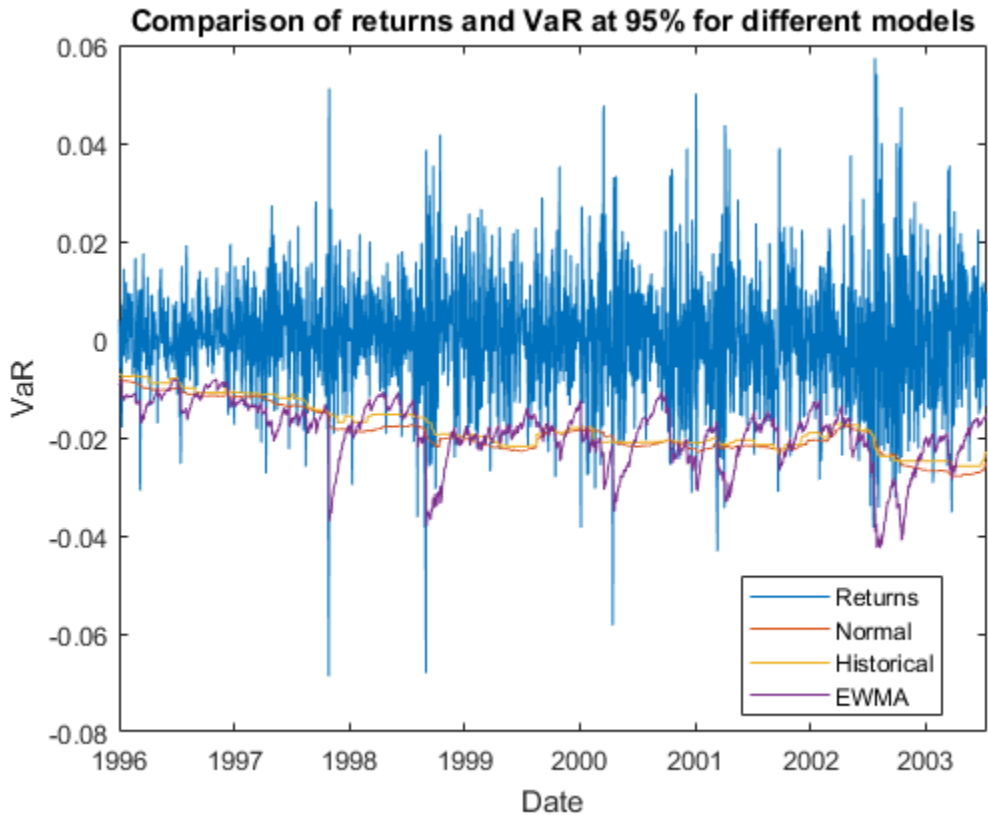
In the first part of this example, VaR was estimated over the test window with three different methods and at two different VaR confidence levels. The goal of VaR

backtesting is to evaluate the performance of VaR models. A VaR estimate at 95% confidence is violated only about 5% of the time, and VaR failures do not cluster. Clustering of VaR failures indicates the lack of independence across time because the VaR models are slow to react to changing market conditions.

A common first step in VaR backtesting analysis is to plot the returns and the VaR estimates together. Plot all three methods at the 95% confidence level and compare them to the returns.

```
ReturnsTest = Returns(TestWindow);
DatesTest   = DateReturns(TestWindow);
figure;
plot(DatesTest,[ReturnsTest -Normal95 -Historical95 -EWMA95])
ylabel('VaR')
xlabel('Date')
legend({'Returns','Normal','Historical','EWMA'},'Location','Best')
title('Comparison of returns and VaR at 95% for different models')
```



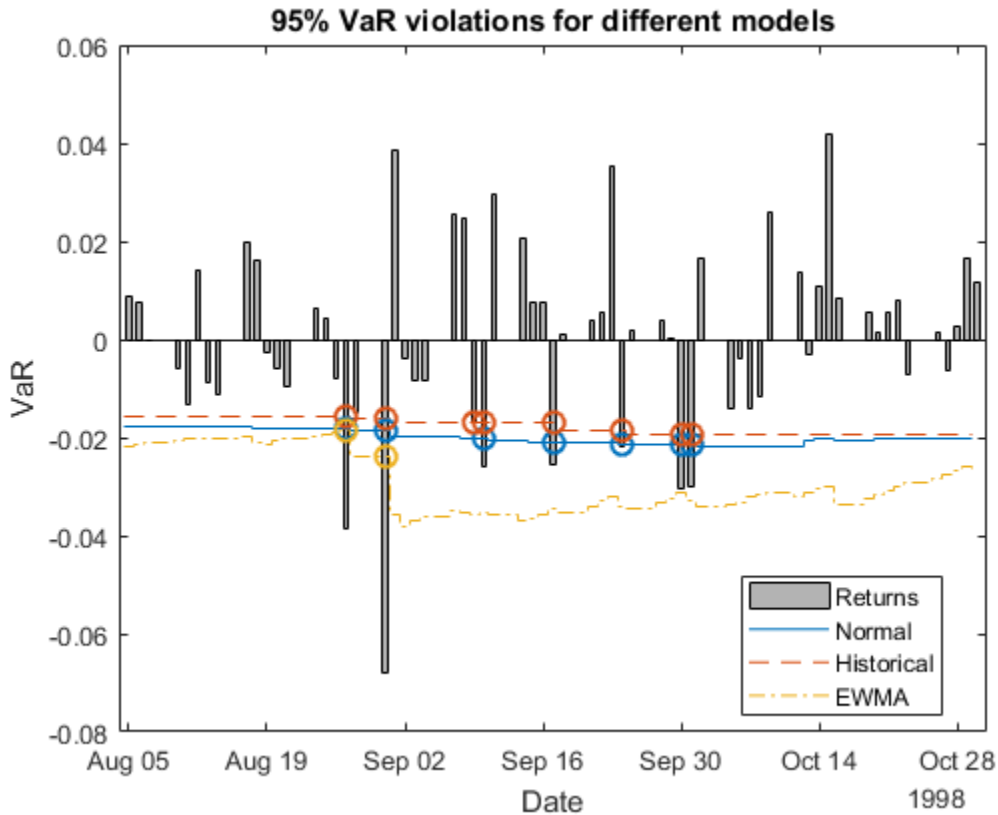


To highlight how the different approaches react differently to changing market conditions, you can zoom in on the time series where there is a large and sudden change in the value of returns. For example, around August 1998:

```
ZoomInd = (DatesTest >= datestr('5-Aug-1998','local')) & (DatesTest <= datestr('31-Oct-1998','local'));
VaRData = [-Normal195(ZoomInd) -Historical195(ZoomInd) -EWMA95(ZoomInd)];
VaRFormat = {'-','-','-'};
D = DatesTest(ZoomInd);
R = ReturnsTest(ZoomInd);
N = Normal195(ZoomInd);
H = Historical195(ZoomInd);
E = EWMA95(ZoomInd);
IndN95 = (R < -N);
IndHS95 = (R < -H);
```

```
IndEWMA95 = (R < -E);
figure;
bar(D,R,0.5, 'FaceColor', [0.7 0.7 0.7]);
hold on
for i = 1 : size(VaRData,2)
    stairs(D-0.5,VaRData(:,i),VaRFormat{i});
end
ylabel('VaR')
xlabel('Date')
legend({'Returns','Normal','Historical','EWMA'},'Location','Best','AutoUpdate','Off')
title('95% VaR violations for different models')
ax = gca;
ax.ColorOrderIndex = 1;

plot(D(IndN95), -N(IndN95), 'o', D(IndHS95), -H(IndHS95), 'o', ...
     D(IndEWMA95), -E(IndEWMA95), 'o', 'MarkerSize', 8, 'LineWidth', 1.5)
xlim([D(1)-1, D(end)+1])
hold off;
```



A VaR failure or violation happens when the returns have a negative VaR. A closer look around August 27th to August 31st shows a significant dip in the returns. On the dates starting from August 27th onward, the EWMA follows the trend of the returns closely and more accurately. Consequently, EWMA has fewer VaR violations (two (2) violations, yellow diamonds) compared to the Normal Distribution approach (seven (7) violations, blue stars) or the Historical Simulation method (eight (8) violations, red squares).

Besides visual tools, you can use statistical tests for VaR backtesting. In Risk Management Toolbox™, a `varbacktest` object supports multiple statistical tests for VaR backtesting analysis. In this example, start by comparing the different test results for the normal distribution approach at the 95% and 99% VaR levels.

```
vbt = varbacktest>ReturnsTest,[Normal95 Normal99], 'PortfolioID', 'S&P', 'VaRID', ...
```

```
{'Normal195', 'Normal199'}, 'VaRLevel', [0.95 0.99]);
summary(vbt)
```

```
ans =
```

```
2x10 table
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Failures
"S&P"	"Normal195"	0.95	0.94863	1966	101
"S&P"	"Normal199"	0.99	0.98372	1966	32

The summary report shows that the observed level is close enough to the defined VaR level. The 95% and 99% VaR levels have at most  $(1 - \text{VaR\_level}) \times N$  expected failures, where  $N$  is the number of observations. The failure ratio shows that the Normal195 VaR level is within range, whereas the Normal199 VaR Level is imprecise and under-forecasts the risk. To run all tests supported in `varbacktest`, use `runtests`.

```
runtests(vbt)
```

```
ans =
```

```
2x11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"S&P"	"Normal195"	0.95	green	accept	accept	accept
"S&P"	"Normal199"	0.99	yellow	reject	reject	accept

The 95% VaR passes the frequency tests, such as traffic light, binomial and proportion of failures tests (TL, Bin, and POF columns). The 99% VaR does not pass these same tests, as indicated by the yellow and reject results. Both confidence levels got rejected in the conditional coverage independence, and time between failures independence (CCI and TBFI columns). This result suggests that the VaR violations are not independent, and there are probably periods with multiple failures in a short span. Also, one failure may make it more likely that other failures will follow in subsequent days. For more information on the tests methodologies and the interpretation of results, see `varbacktest` and the individual tests.

Using a `varbacktest` object, run the same tests on the portfolio for the three approaches at both VaR confidence levels.

```
vbt = varbacktest>ReturnsTest,[Normal195 Historical195 EWMA95 Normal199 Historical199 ...
    EWMA99], 'PortfolioID', 'S&P', 'VaRID', {'Normal195', 'Historical195', 'EWMA95', ...
    'Normal199', 'Historical199', 'EWMA99'}, 'VaRLevel', [0.95 0.95 0.95 0.99 0.99 0.99]);
runtests(vbt)
```

ans =

6×11 table

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"S&P"	"Normal195"	0.95	green	accept	accept	accept
"S&P"	"Historical195"	0.95	yellow	accept	accept	accept
"S&P"	"EWMA95"	0.95	green	accept	accept	accept
"S&P"	"Normal199"	0.99	yellow	reject	reject	accept
"S&P"	"Historical199"	0.99	yellow	reject	reject	accept
"S&P"	"EWMA99"	0.99	red	reject	reject	accept

The results are similar to the previous results, and at the 95% level, the frequency results are generally acceptable. However, the frequency results at the 99% level are generally rejections. Regarding independence, most tests pass the conditional coverage independence test (`cci`), which tests for independence on consecutive days. Notice that all tests fail the time between failures independence test (`tbfi`), which takes into account the times between all failures. This result suggests that all methods have issues with the independence assumption.

To better understand how these results change given market conditions, look at the years 2000 and 2002 for the 95% VaR confidence level.

```
Ind2000 = (year(DatesTest) == 2000);
vbt2000 = varbacktest>ReturnsTest(Ind2000),[Normal195(Ind2000) Historical195(Ind2000) EWMA95(Ind2000) ...
    'PortfolioID', 'S&P, 2000', 'VaRID', {'Normal', 'Historical', 'EWMA'});
runtests(vbt2000)
```

ans =

3×11 table

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"S&P, 2000"	"Normal"	0.95	green	accept	accept	accept
"S&P, 2000"	"Historical"	0.95	green	accept	accept	accept
"S&P, 2000"	"EWMA"	0.95	green	accept	accept	accept

```
Ind2002 = (year(DatesTest) == 2002);
vbt2002 = varbacktest>ReturnsTest(Ind2002),[Normal195(Ind2002) Historical195(Ind2002) EWMA195(Ind2002)]
'PortfolioID', 'S&P, 2002', 'VaRID', {'Normal', 'Historical', 'EWMA'});
runtests(vbt2002)
```

ans =

3×11 table

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"S&P, 2002"	"Normal"	0.95	yellow	reject	reject	accept
"S&P, 2002"	"Historical"	0.95	yellow	reject	accept	accept
"S&P, 2002"	"EWMA"	0.95	green	accept	accept	accept

For the year 2000, all three methods pass all the tests. However, for the year 2002, the test results are mostly rejections for all methods. The EWMA method seems to perform better in 2002, yet all methods fail the independence tests.

To get more insight into the independence tests, look into the conditional coverage independence (cci) and the time between failures independence (tbfi) test details for the year 2002. To access the test details for all tests, run the individual test functions.

```
cci(vbt2002)
```

ans =

3×13 table

PortfolioID	VaRID	VaRLevel	CCI	LRatioCCI	PValueCCI	Obs
-------------	-------	----------	-----	-----------	-----------	-----

"S&P, 2002"	"Normal"	0.95	reject	12.591	0.0003877	261
"S&P, 2002"	"Historical"	0.95	reject	6.3051	0.012039	261
"S&P, 2002"	"EWMA"	0.95	reject	4.6253	0.031504	261

In the CCI test, the probability  $p_{01}$  of having a failure at time  $t$ , knowing that there was no failure at time  $t-1$  is given by

$$p_{01} = \frac{N_{01}}{N_{01} + N_{00}}$$

The probability  $p_{11}$  of having a failure at time  $t$ , knowing that there was failure at time  $t-1$  is given by

$$p_{11} = \frac{N_{11}}{N_{11} + N_{10}}$$

From the  $N_{00}$ ,  $N_{10}$ ,  $N_{01}$ ,  $N_{11}$  columns in the test results, the value of  $p_{01}$  is at around 5% for the three methods, yet the values of  $p_{11}$  are above 20%. Because there is evidence that a failure is followed by another failure much more frequently than 5% of the time, this CCI test fails.

In the time between failures independence test, look at the minimum, maximum, and quartiles of the distribution of times between failures, in the columns TBFMin, TBFQ1, TBFQ2, TBFQ3, TBFMax.

```
tbfi(vbt2002)
```

```
ans =
```

```
3x14 table
```

PortfolioID	VaRID	VaRLevel	TBFI	LRatioTBFI	PValueTBFI	OB
"S&P, 2002"	"Normal"	0.95	reject	53.936	0.00010087	20
"S&P, 2002"	"Historical"	0.95	reject	45.274	0.0010127	20
"S&P, 2002"	"EWMA"	0.95	reject	25.756	0.027796	20

For a VaR level of 95%, you expect an average time between failures of 20 days, or one failure every 20 days. However, the median of the time between failures for the year 2002

ranges between 5 and 7.5 for the three methods. This result suggests that half of the time, two consecutive failures occur within 5 to 7 days, much more frequently than the 20 expected days. Consequently, more test failures occur. For the normal method, the first quartile is 1, meaning that 25% of the failures occur on consecutive days.

### References

Nieppola, O. *Backtesting Value-at-Risk Models*. Helsinki School of Economics. 2009.

Danielsson, J. *Financial Risk Forecasting: The Theory and Practice of Forecasting Market Risk, with Implementation in R and MATLAB®*. Wiley Finance, 2012.

### See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tbf1` | `t1` | `tuff` | `varbacktest`

### Related Examples

- “Overview of VaR Backtesting” on page 2-2
- “VaR Backtesting Workflow” on page 2-8

### More About

- “Traffic Light Test” on page 2-3
- “Binomial Test” on page 2-3
- “Kupiec’s POF and TUFF Tests” on page 2-4
- “Christoffersen’s Interval Forecast Tests” on page 2-5
- “Haas’s Time Between Failures or Mixed Kupiec’s Test” on page 2-6



# Managing Consumer Credit Risk Using the Binning Explorer for Credit Scorecards

---

- “Overview of Binning Explorer” on page 3-2
- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-26
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-44

## Overview of Binning Explorer

The **Binning Explorer** app enables you to interactively bin credit scorecard data. Use the **Binning Explorer** to:

- Select an automatic binning algorithm.
- Shift bin boundaries.
- Split bins.
- Merge bins.
- Save and export a `creditscorecard` object.

**Binning Explorer** complements the overall workflow for developing a credit scorecard model.

<b>Using Binning Explorer:</b>	
1.	<p>Open the <b>Binning Explorer</b> app.</p> <ul style="list-style-type: none"> <li>• MATLAB® Toolstrip: On the <b>Apps</b> tab, under <b>Computational Finance</b>, click the app icon.</li> <li>• MATLAB Command prompt: Enter <code>binningExplorer</code>.</li> </ul>
2.	<p>Import the data into the app.</p> <p>You can import data into <b>Binning Explorer</b> by either starting directly from a data set or by loading an existing <code>creditscorecard</code> object from the MATLAB workspace.</p>
3.	<p>Use <b>Binning Explorer</b> to work interactively with the binning assignments for a scorecard.</p>
4.	<p>Export the scorecard to a new <code>creditscorecard</code> object.</p> <p>Continue the workflow from the MATLAB command line using <code>creditscorecard</code> object functions from Financial Toolbox. For more information, see <code>creditscorecard</code>.</p>
<b>Using <code>creditscorecard</code> Object Functions in Financial Toolbox:</b>	
5.	<p>Fit a logistic regression model.</p>
6.	<p>Review and format the credit scorecard points.</p>
7.	<p>Score the data.</p>

<b>Using Binning Explorer:</b>	
8.	Calculate the probabilities of default for the data.
9.	Validate the quality of the credit scorecard model.

For more detailed information on this workflow, see “Binning Explorer Case Study Example” on page 3-26.

## See Also

### Apps

Binning Explorer

### Functions

creditscorecard

## Related Examples

- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-26
- “Case Study for a Credit Scorecard Analysis” (Financial Toolbox)

## More About

- “Credit Scorecard Modeling Workflow” (Financial Toolbox)

## External Websites

- Credit Scorecard Modeling Using the Binning Explorer App (6 min 30 sec)

## Common Binning Explorer Tasks

The **Binning Explorer** app supports the following tasks:

### In this section...

“Import Data” on page 3-4

“Change Predictor Type” on page 3-5

“Change Binning Algorithm for One or More Predictors” on page 3-6

“Change Algorithm Options for Binning Algorithms” on page 3-7

“Split Bins for a Numeric Predictor” on page 3-11

“Split Bins for a Categorical Predictor” on page 3-12

“Manual Binning to Merge Bins for a Numeric or Categorical Predictor” on page 3-14

“Change Bin Boundaries for a Single Predictor” on page 3-16

“Change Bin Boundaries for Multiple Predictors” on page 3-17

“Set Options for Display” on page 3-19

“Export and Save the Binning” on page 3-19

“Troubleshoot the Binning” on page 3-20

## Import Data

**Binning Explorer** enables you to import data by either starting directly from the data stored in a MATLAB table or by loading an existing `creditscorecard` object.

### Clean Start from Data

To start directly from data for a credit scorecard:

- 1 Place the credit scorecard data in your MATLAB workspace. The data must be in a MATLAB table, where each column of data can be any one of the following data types:
  - Numeric
  - Logical
  - Cell array of character vectors
  - Character array

- Categorical

In addition, the table must contain a binary response variable.

- 2 Open **Binning Explorer** from the MATLAB toolstrip: On the **Apps** tab, under **Computational Finance**, click the app icon.
- 3 Select the data from the **Step 1** pane of the Import Data window.
- 4 From the **Step 2** pane, set the **Variable Type** for each of the predictors, as needed.
- 5 From the **Step 3** pane, select an initial binning algorithm and click **Import Data**. The bins are plotted and displayed for each predictor. By clicking an individual predictor plot, the details for that predictor plot display in the **Bin Information** and **Predictor Information** panes.

### Start from an Existing creditcorecard Object

To start using an existing creditcorecard object:

- 1 Place the creditcorecard object in your MATLAB workspace. Create the creditcorecard object either by using the creditcorecard function or by clicking **Export** in the **Binning Explorer** to export and save a creditcorecard object to the MATLAB workspace.
- 2 Open Binning Explorer from the MATLAB toolstrip: On the **Apps** tab, under **Computational Finance**, click the app icon.
- 3 From **Step 1** pane of the Import Data window, select the creditcorecard object.
- 4 From the **Step 3** pane, select a binning algorithm. When using an existing creditcorecard object, it is recommended to select the **No Binning** option. To display the predictor plots, click **Import Data**.

The bins are plotted and displayed for each predictor. By clicking an individual predictor plot, the details for that predictor plot display in the **Bin Information** and **Predictor Information** panes.

### Change Predictor Type

After you import data or a creditcorecard object into **Binning Explorer**, you can change the predictor type.

- 1 Click any predictor plot. The name of the selected predictor displays on the **Binning Explorer** toolbar under **Selected Predictor**.

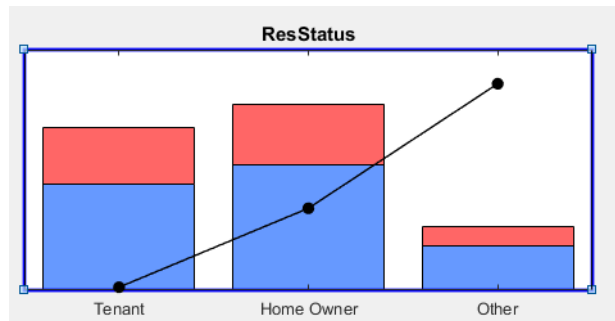
On the **Binning Explorer** toolbar, the predictor type for the selected predictor displays under **Predictor Type**.

- 2 To change the predictor type, under **Predictor Type**, select: **Numeric**, **Categorical**, or **Ordinal**. The predictor plot is updated for a change in the predictor type and the details in the **Bin Information** and **Predictor Information** panes are also updated.

## Change Binning Algorithm for One or More Predictors

After you import data or a `creditscorecard` object into **Binning Explorer**, you can change the binning algorithm for an individual predictor or for multiple predictors.

- 1 Click any predictor plot. The selected predictor plot displays with a blue outline.

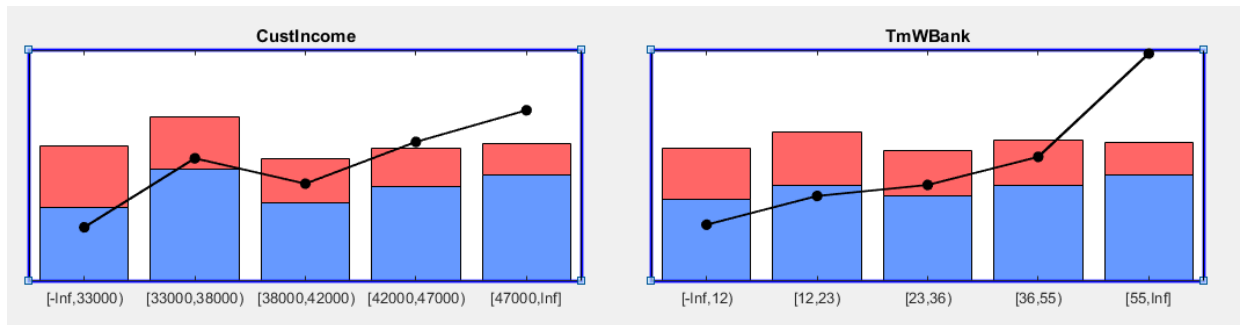


---

**Tip:** When you select a predictor plot with the blue outline, a status message appears at the bottom of the **Binning Explorer** that displays the last binning information for that predictor. Use this information to determine which binning algorithm is most recently applied to an individual predictor plot.

---

- 2 On the **Binning Explorer** toolbar, under **Apply Monotone**, select **Monotone**, **Equal Frequency**, or **Equal Width**. The predictor plot is updated with a change of algorithm. The details in the **Bin Information** and **Predictor Information** panes are also updated.
- 3 To change the binning algorithm for multiple predictors, multiselect more than one predictor plot by using **Ctrl** + click to highlight each predictor plot with a blue outline.

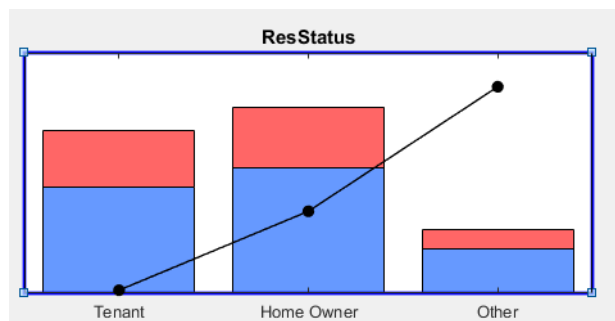


- 4 Under **Apply Monotone**, select **Monotone**, **Equal Frequency**, or **Equal Width**. All the selected predictor plots are updated for a change of algorithm.

## Change Algorithm Options for Binning Algorithms

After you import data or a `creditscorecard` object into **Binning Explorer**, you can change the binning algorithm for an individual predictor or for multiple predictors.

- 1 Click any predictor plot. The predictor plot displays with a blue outline.

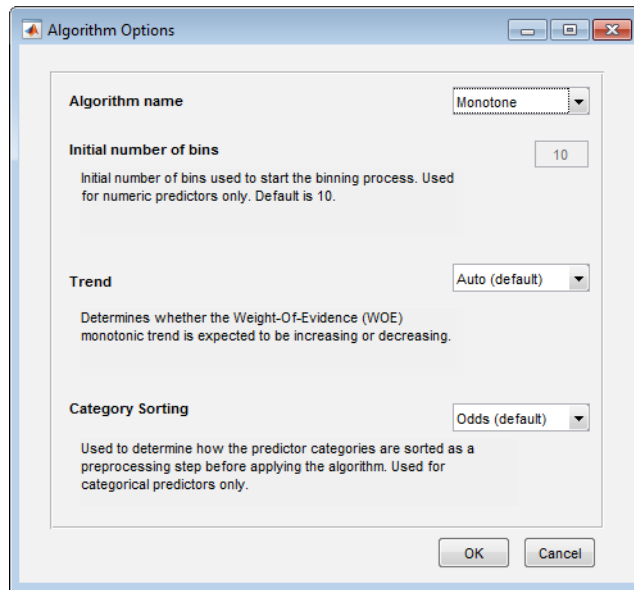



---

**Tip:** When you select a predictor plot with the blue outline, a status message appears at the bottom of the **Binning Explorer** that displays the last binning information for that predictor. Use this information to determine which binning algorithm is most recently applied to an individual predictor plot.

---

- 2 On the **Binning Explorer** toolbar, click **Algorithm Options** to open the Algorithm Options dialog box.



- 3 From the Algorithm Options dialog box, select an **Algorithm name**:
  - Monotone
    - For **Initial number of bins**, enter an initial number of bins (default is 10). The initial number of bins must be an integer > 2. Used for numeric predictors only.
    - For **Trend**, select one of the following:
      - **Auto (default)** — Automatically determines if the WOE trend is increasing or decreasing.
      - **Increasing** — Looks for an increasing WOE trend.
      - **Decreasing** — Looks for a decreasing WOE trend.

The value of **Trend** does not necessarily reflect that of the resulting WOE curve. The **Trend** option tells the algorithm to look for an increasing or decreasing trend, but the outcome might not show the desired trend. For example, the algorithm cannot find a decreasing trend when the data actually



has an increasing WOE trend. For more information on the **Trend** option, see “Monotone” (Financial Toolbox).

- For **Category Sorting**, select one of the following:
  - **Odds (default)** — The categories are sorted by order of increasing values of odds, defined as the ratio of “Good” to “Bad” observations, for the given category.
  - **Goods** — The categories are sorted by order of increasing values of “Good.”
  - **Bads** — The categories are sorted by order of increasing values of “Bad.”
  - **Totals** — The categories are sorted by order of increasing values of the total number of observations (“Good” plus “Bad”).
  - **None** — No sorting is applied. The existing order of the categories is unchanged before applying the algorithm.

---

**Note:** **Category Sorting** can only be used with categorical predictors.

---

- Equal Frequency
  - For **Number of bins**, enter the number of bins. The default is 5, and the number of bins must be a positive number.
  - For **Category Sorting**, select one of the following:
    - **Odds (default)** — The categories are sorted by order of increasing values of odds, defined as the ratio of “Good” to “Bad” observations, for the given category.
    - **Goods** — The categories are sorted by order of increasing values of “Good.”
    - **Bads** — The categories are sorted by order of increasing values of “Bad.”
    - **Totals** — The categories are sorted by order of increasing values of the total number of observations (“Good” plus “Bad”).
    - **None** — No sorting is applied. The existing order of the categories is unchanged before applying the algorithm.

---

**Note:** You can use **Category Sorting** with categorical predictors only.

---

- Equal Width

- For **Number of bins**, enter the number of bins. The default is 5 and the number of bins must be a positive number.
- For **Category Sorting**, select one of the following:
  - **Odds (default)** — The categories are sorted by order of increasing values of odds, defined as the ratio of “Good” to “Bad” observations, for the given category.
  - **Goods** — The categories are sorted by order of increasing values of “Good.”
  - **Bads** — The categories are sorted by order of increasing values of “Bad.”
  - **Totals** — The categories are sorted by order of increasing values of the total number of observations (“Good” plus “Bad”).
  - **None** — No sorting is applied. The existing order of the categories is unchanged before applying the algorithm.

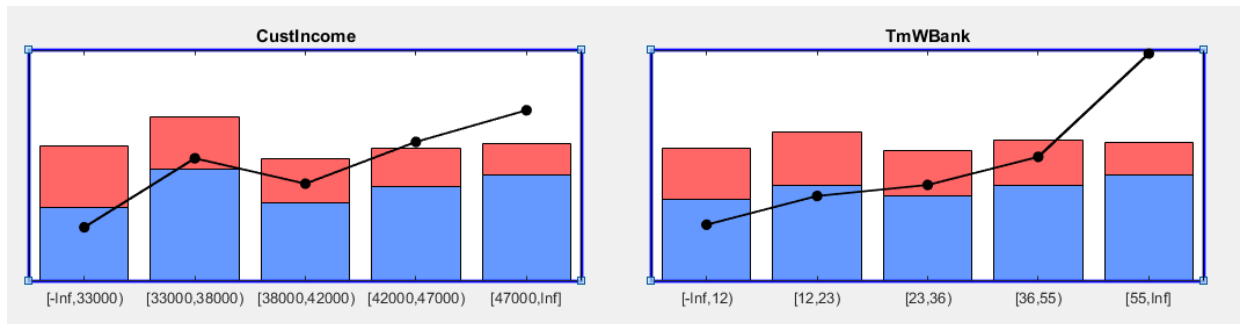
---

**Note:** You can use **Category Sorting** with categorical predictors only.

---

Click **OK**. The predictor plot is updated with the change of algorithm. The details in the **Bin Information** and **Predictor Information** panes are also updated.

- 4 To change the binning algorithm for multiple predictors, multiselect more than one predictor plot by using **Ctrl+** click to highlight each predictor plot with a blue outline.

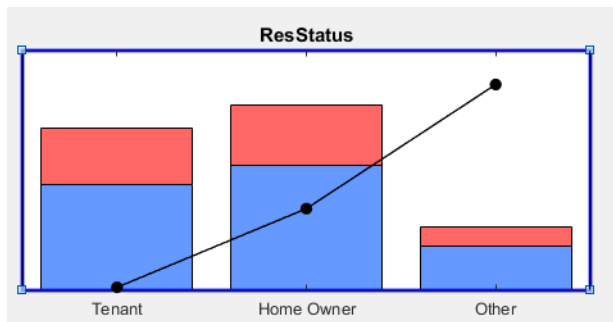


- 5 On the **Binning Explorer** toolbar, click **Algorithm Options** to open the Algorithm Options dialog box. Make your selection from the Algorithm Options dialog box and click **OK**. The selected predictor plots are updated for the change of algorithm.

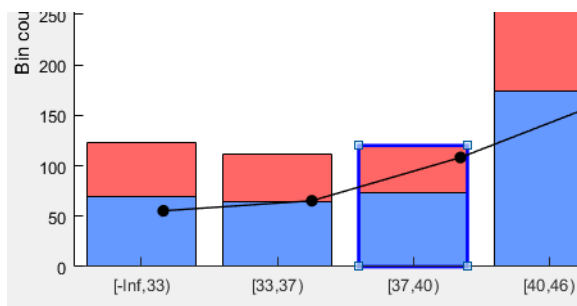
## Split Bins for a Numeric Predictor

After you import data or a `creditscorecard` object into **Binning Explorer**, you can split bins for a numeric predictor.

- 1 Click any numeric predictor plot. The predictor plot displays with a blue outline.



- 2 On the **Binning Explorer** toolbar, click **Manual Binning** to open the selected numeric predictor in a new tabbed window.
- 3 Click a bin to enable the **Split** button for that bin.

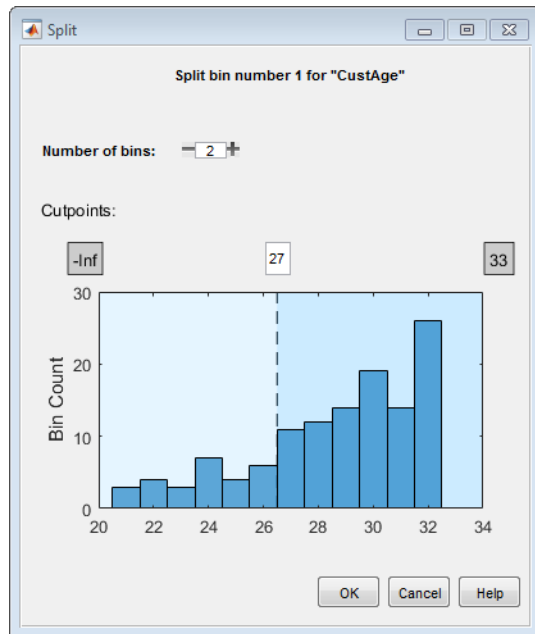



---

**Note:** The **Split** button is enabled when the data range of the selected bin has more than one value.

---

- 4 On the **Binning Explorer** toolbar, the **Edges** text boxes display values for the edges of the selected bin. Click **Split** to open the Split dialog box.



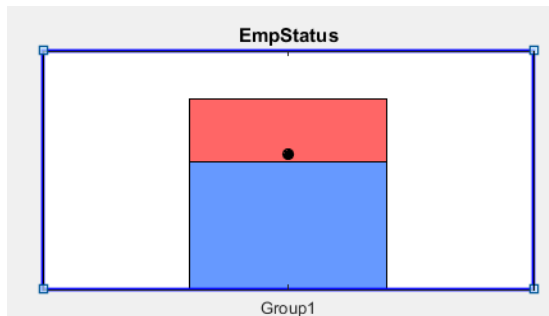
- 5 Use the **Number of bins** control to split the selected bin into multiple bins. Click **OK** to complete the split operation.

The plot for the selected numeric predictor is updated with the new bin information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

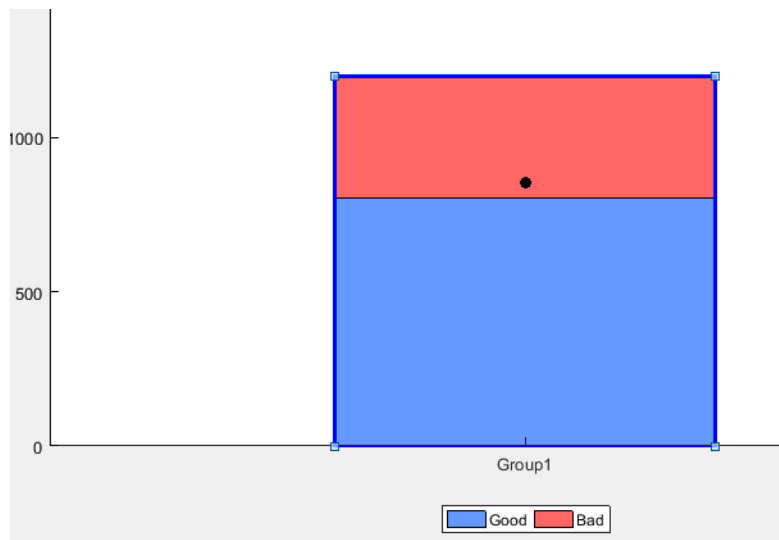
## Split Bins for a Categorical Predictor

After you import data or a `creditscorecard` object into **Binning Explorer**, you can split bins for a categorical predictor.

- 1 Click any categorical predictor plot. The predictor plot displays with a blue outline.



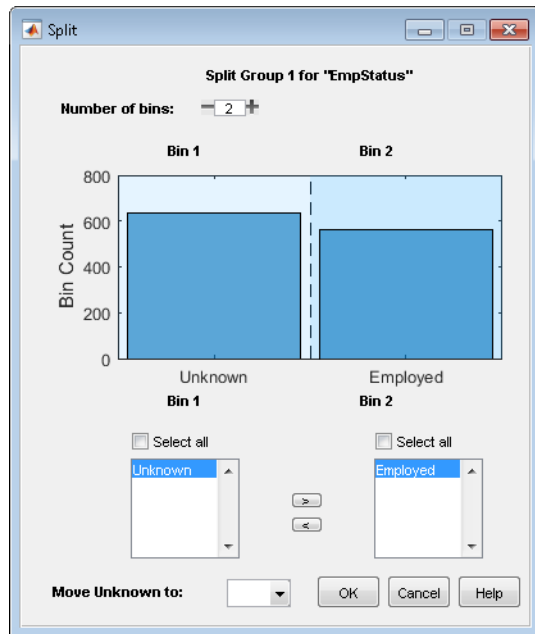
- 2 On the **Binning Explorer** toolbar, click **Manual Binning** to open the selected categorical predictor in a new tabbed window.
- 3 Click a bin to enable the **Split** button for that bin.



---

**Note:** The **Split** button is enabled when the selected bin has more than one category in it.

---



Use the **Number of bins** control to split the selected bin into multiple bins.

Use the arrow controls on the Split dialog box to control the contents for each of the bins that you are splitting the selected bin into.

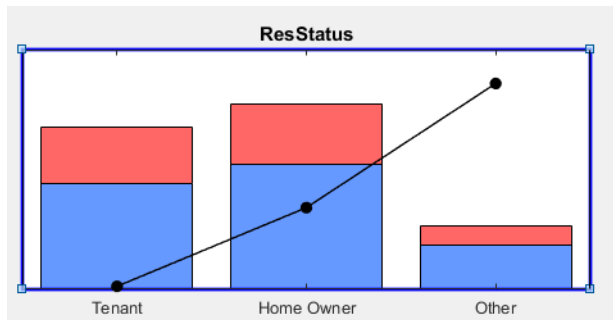
- 4 Click **OK** to complete the split operation.

The plot for the selected categorical predictor is updated with the new bin information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

## Manual Binning to Merge Bins for a Numeric or Categorical Predictor

After you import data or a `creditscorecard` object into **Binning Explorer**, you can split or merge bins for a predictor.

- 1 Click any predictor plot. The predictor plot displays with a blue outline.



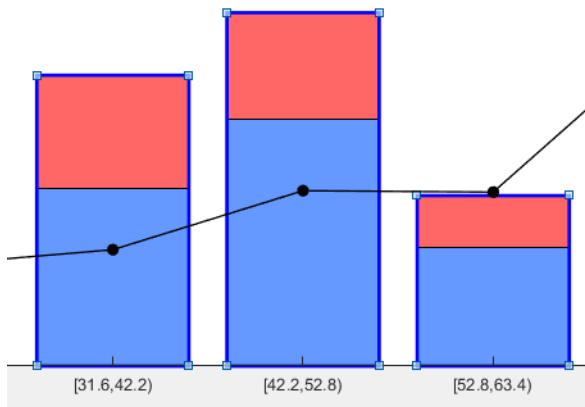
- 2 On the **Binning Explorer** toolbar, click **Manual Binning** to open the selected predictor in a new tabbed window.

---

**Note:** The **Merge** button is active only when more than one bin is selected. Only adjacent bins can be merged for numeric or ordinal predictors. Nonadjacent bins can be merged for categorical predictors.

---

- 3 To merge bins, select two or more bins for merging by using **Ctrl** + click to multiselect bins to display with blue outlines.



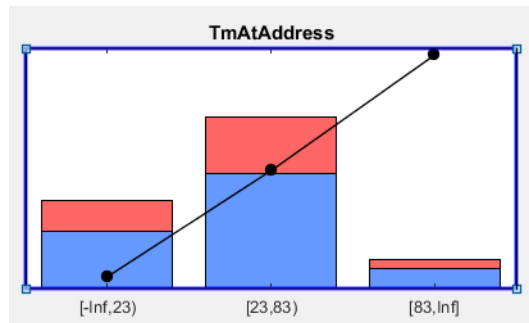
When performing a merge with a numeric predictor, the **Edges** text boxes on the **Binning Explorer** toolbar display the values for the edges of the selected bins to merge.

- 4 Click **Merge** to complete the merge operation. The plot for the selected predictor is updated with the new bin information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

## Change Bin Boundaries for a Single Predictor

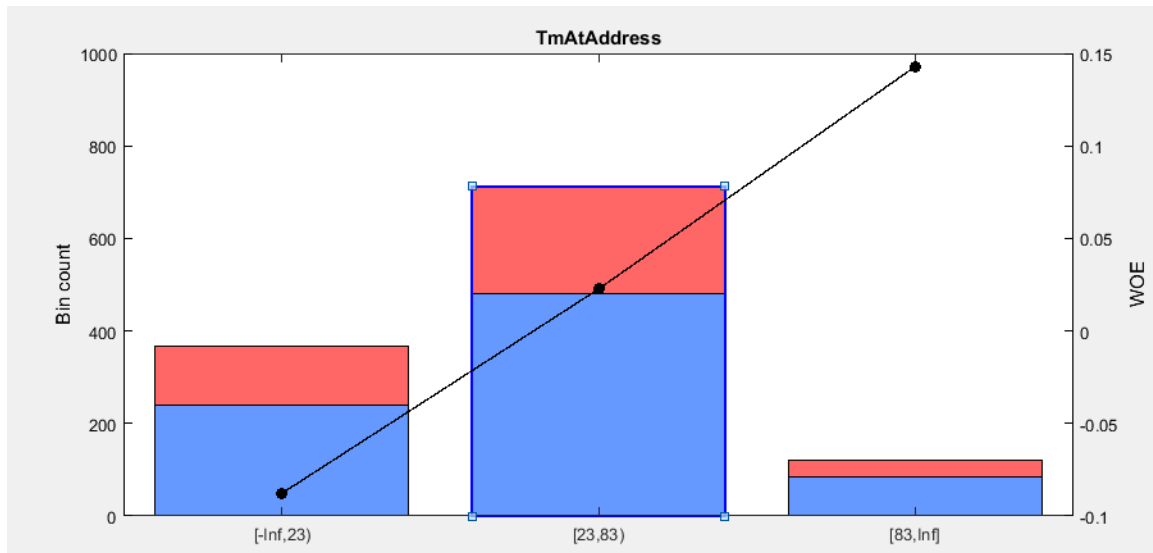
After you import data or a `creditscorecard` object into **Binning Explorer**, you can change the bin boundaries for a single predictor.

- 1 Click any predictor plot. The predictor plot displays with a blue outline.



- 2 On the **Binning Explorer** toolbar, click **Manual Binning**. Click to select a specific bin where you want to change the bin dimensions. The selected bin displays with a blue outline.





- 3 On the **Binning Explorer** toolbar, the **Edges** text boxes display values for the edges of the selected bin.



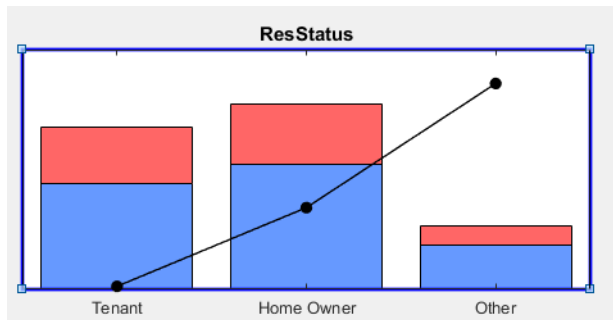
Edit the values in the **Edges** text boxes to change the selected bin's dimensions.

- 4 Press **Enter** to complete the operation. The plot for the selected predictor is updated with the updated bin's dimension information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

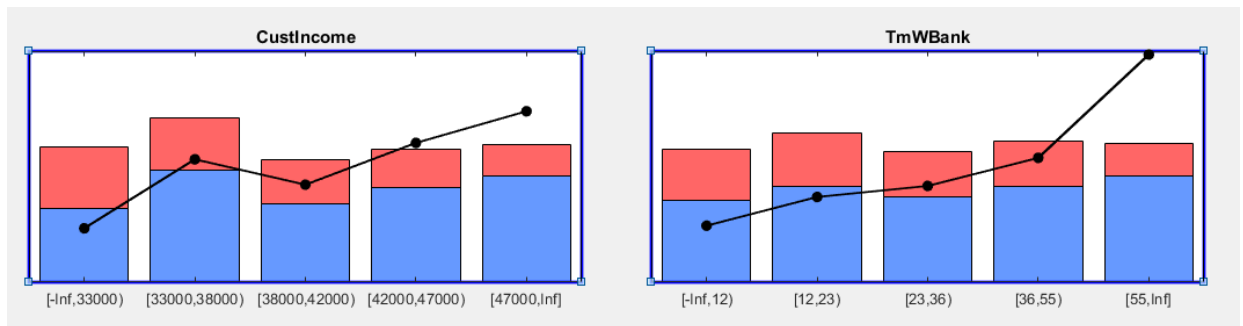
## Change Bin Boundaries for Multiple Predictors

After you import data or a `creditscorecard` object into **Binning Explorer**, you can change the algorithm applied to one or more predictors and you can also redefine the number of bins.

- 1 Click any predictor plot. The predictor plot displays with a blue outline.



Alternatively, select two or more predictors by using **Ctrl** + click to multiselect predictors to display with blue outlines.



- 2 On the **Binning Explorer** toolbar, click **Algorithm Options** to open the Algorithm Options dialog box.
- 3 From the Algorithm Options dialog box, under **Algorithm name**, select a binning algorithm.
  - For the **EqualWidth** and **EqualFrequency** options, enter a number in the **Number of bins** box. Optionally, for **EqualWidth** and **EqualFrequency** options, under **Category Sorting**, specify the type of sorting.
  - For the **Monotone** option, the default of 10 is used for the **Initial number of bins**. Optionally, you can set values for **Trend** and **Category Sorting**.
- 4 Click **OK** to complete the operation. The plots for the selected predictors are updated with the new bin information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

## Set Options for Display

**Binning Explorer** has options for displaying predictor plots and plot options and the associated tables displayed in **Bin Information**.

### Plot Options

- 1 From the **Binning Explorer** toolbar item for **Plot Options**, select any of the following predictor plot options:
  - **No labels** (default)
  - **Bin count**
  - **% Bin level**
  - **% Data level**
  - **% Total count**
  - **WOE curve**
- 2 The selected label is applied to all predictor plots.

### Table Options

You can set the table display options for predictor information displayed in **Bin Information**.

- 1 From the **Binning Explorer** toolbar item for **Table Columns**, select any of the following options:
  - **Odds**
  - **WOE**
  - **InfoValue**
  - **Entropy**
  - **Members** (option is enabled for categorical predictors)
- 2 When selected, these options are applied to all predictors for the information displayed in **Bin Information**.

## Export and Save the Binning

**Binning Explorer** enables you to export and save your credit scorecard binning definitions to a `creditscorecard` object.

- 1 Click **Export** and provide a `creditscorecard` object name. The `creditscorecard` object is saved to the MATLAB workspace.

---

**Note:** If you export a previously existing `creditscorecard` object that was fit (using `fitmodel`), all fitting settings in the `creditscorecard` object are lost. You must rerun `fitmodel` on the updated `creditscorecard` object.

---

- 2 To reopen a previously saved `creditscorecard` object, click **Import Data** and select the `creditscorecard` object from the **Step 1** pane of the Import Data window.

## Troubleshoot the Binning

- “Numeric Predictor Converted to Categorical Predictor Does Not Display Split Data Properly” on page 3-20
- “Predictor Plot Appears Distorted” on page 3-22

This topic shows some of the results when using **Binning Explorer** with credit scorecards that need troubleshooting. For details on the overall process of creating and developing credit scorecards, see “Overview of Binning Explorer” on page 3-2 and “Binning Explorer Case Study Example” on page 3-26.

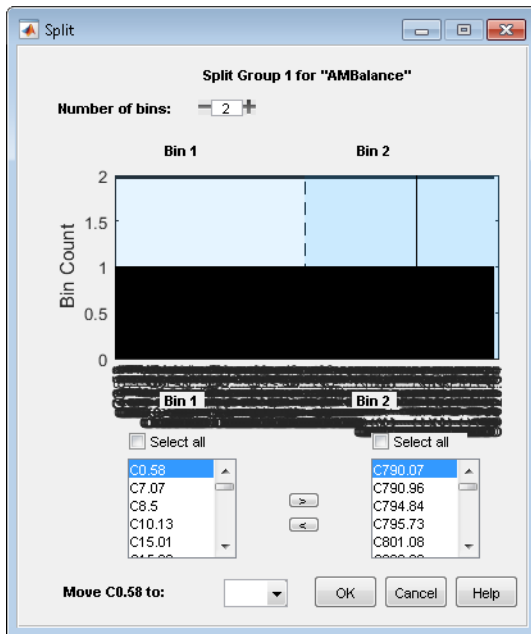
### Numeric Predictor Converted to Categorical Predictor Does Not Display Split Data Properly

When you convert a numeric predictor with hundreds of values (for example, continuous data) to categorical data, the resulting data has hundreds of categories. The following example illustrates this scenario.

```
load CreditCardData
```

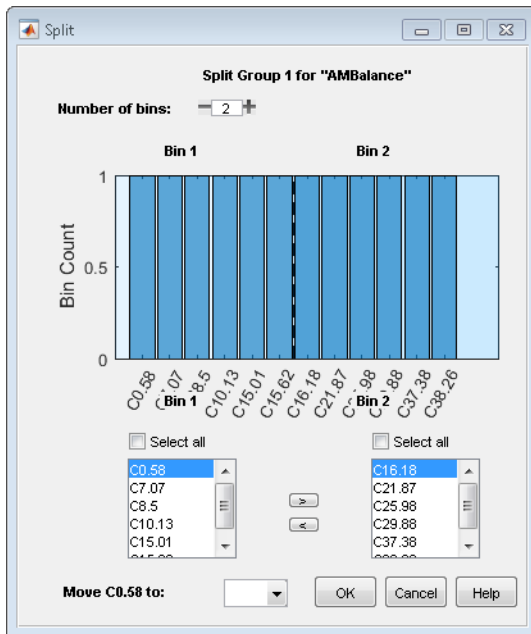
Open the **Binning Explorer** and select the numeric predictor **AMBalance**. From the Binning Explorer toolbar, change the predictor type to **Categorical**.

Select **Manual Binning** on the **Binning Explorer** toolbar and click **Split**. The Split dialog box displays as follows:



The predictor has too many categories to display properly.

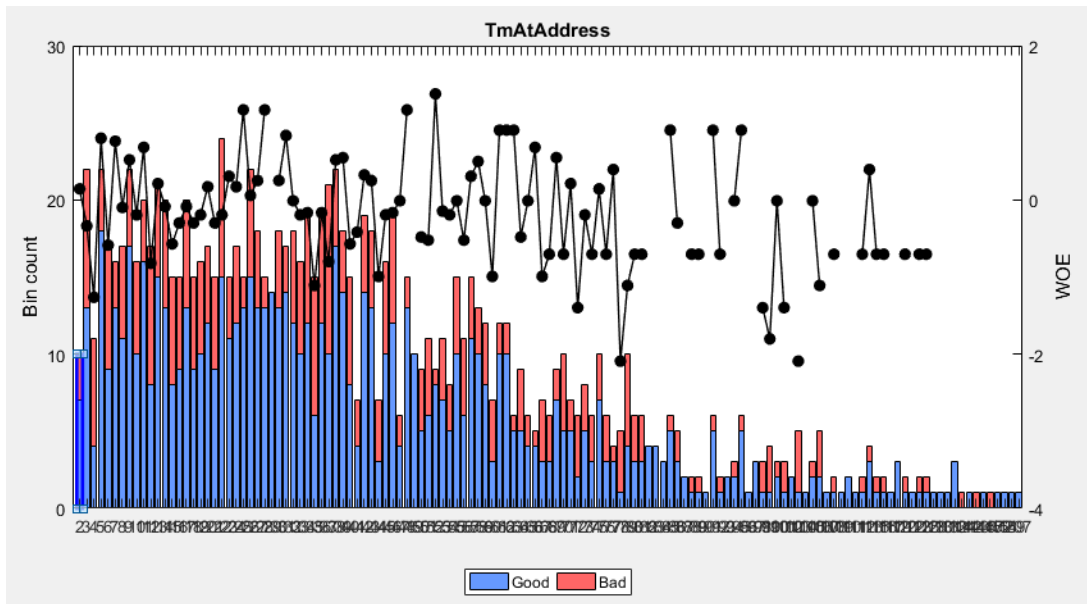
**Solution:** If you have a categorical predictor with a large number of categories, use the **Algorithm Options** to change the binning algorithm for that predictor to **Equal Frequency**, with the **Number of bins** set to 100 (or another smaller value). The Split dialog box then displays properly.



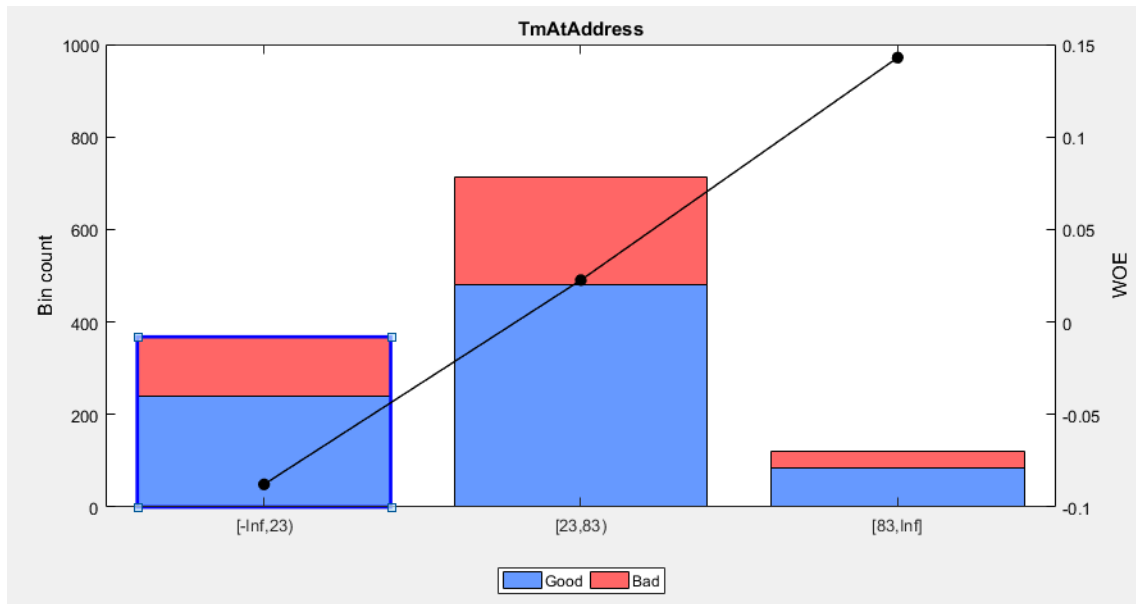
#### Predictor Plot Appears Distorted

When using the **Binning Explorer**, if you import data that has not been previously binned and you select **No Binning** from the Import Data window, the resulting plots might be distorted. For example, if you load the following data set into the MATLAB workspace and use **Binning Explorer** to import the data using **No Binning**, the following plot displays for the **TmAtAddress** predictor.

```
load CreditCardData
```



**Solution:** When you import data that has not been previously binned, select **Monotone** from the Import Data window instead. The following plot displays for the **TmAtAddress** predictor.



## See Also

### Apps

Binning Explorer

### Functions

creditscorecard

## Related Examples

- “Binning Explorer Case Study Example” on page 3-26
- “Case Study for a Credit Scorecard Analysis” (Financial Toolbox)

## More About

- “Overview of Binning Explorer” on page 3-2
- “Credit Scorecard Modeling Workflow” (Financial Toolbox)



## **External Websites**

- Credit Scorecard Modeling Using the Binning Explorer App (6 min 30 sec)

## Binning Explorer Case Study Example

This example shows how to create a credit scorecard using the **Binning Explorer** app. Use the **Binning Explorer** to bin the data, plot the binned data information, and export a `creditscorecard` object. Then use the `creditscorecard` object with functions from Financial Toolbox to fit a logistic regression model, determine a score for the data, determine the probabilities of default, and validate the credit scorecard model using three different metrics.

### Step 1. Load credit scorecard data into the MATLAB workspace.

Use the `CreditCardData.mat` file to load the data into the MATLAB workspace (using a dataset from Refaat 2011).

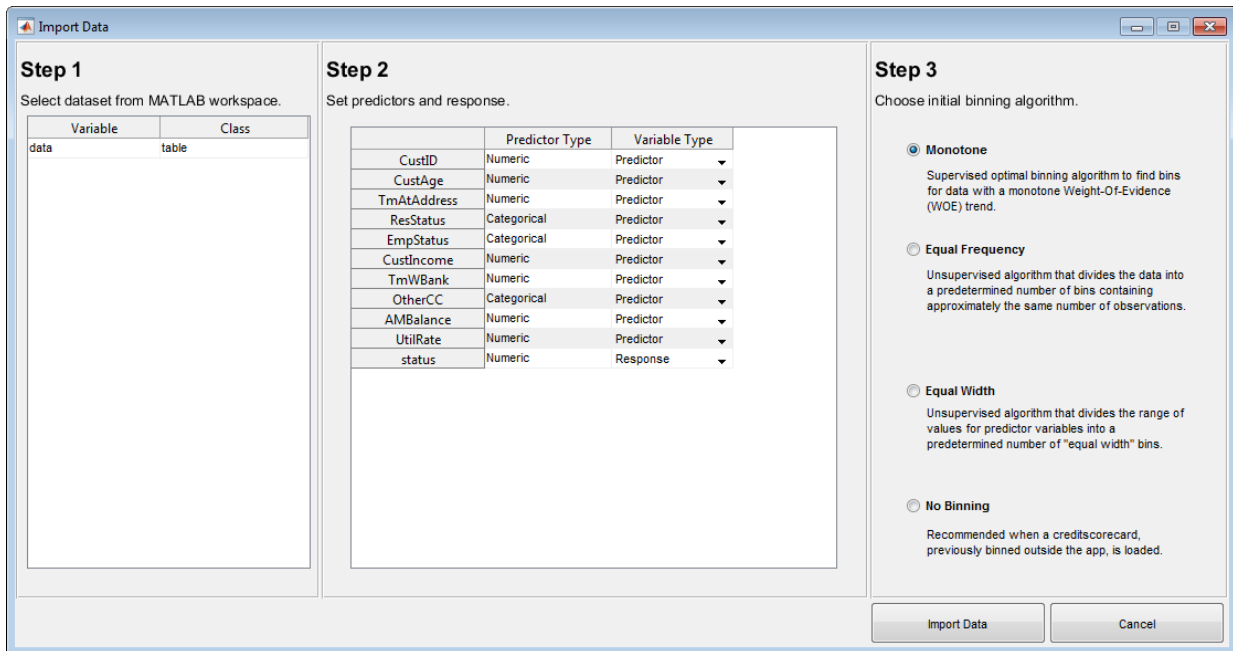
```
load CreditCardData
disp(data(1:10,:))
```

CustID	CustAge	TmAtAddress	ResStatus	EmpStatus	CustIncome	TmWBank	OtherCC	AMBalance	UtilRate
1	53	62	Tenant	Unknown	50000	55	Yes	1055.9	0.22
2	61	22	Home Owner	Employed	52000	25	Yes	1161.6	0.24
3	47	30	Tenant	Employed	37000	61	No	877.23	0.29
4	50	75	Home Owner	Employed	53000	20	Yes	157.37	0.08
5	68	56	Home Owner	Employed	53000	14	Yes	561.84	0.11
6	65	13	Home Owner	Employed	48000	59	Yes	968.18	0.15
7	34	32	Home Owner	Unknown	32000	26	Yes	717.82	0.02
8	50	57	Other	Employed	51000	33	No	3041.2	0.13
9	50	10	Tenant	Unknown	52000	25	Yes	115.56	0.02
10	49	30	Home Owner	Unknown	53000	23	Yes	718.5	0.17

### Step 2. Import the data into Binning Explorer.

Open **Binning Explorer** from the MATLAB toolstrip: On the **Apps** tab, under **Computational Finance**, click the app icon. Alternatively, you can enter `binningExplorer` on the command line.

From the **Binning Explorer** toolbar, select **Import Data** to open the Import Data window.



Under **Step 1**, select data.

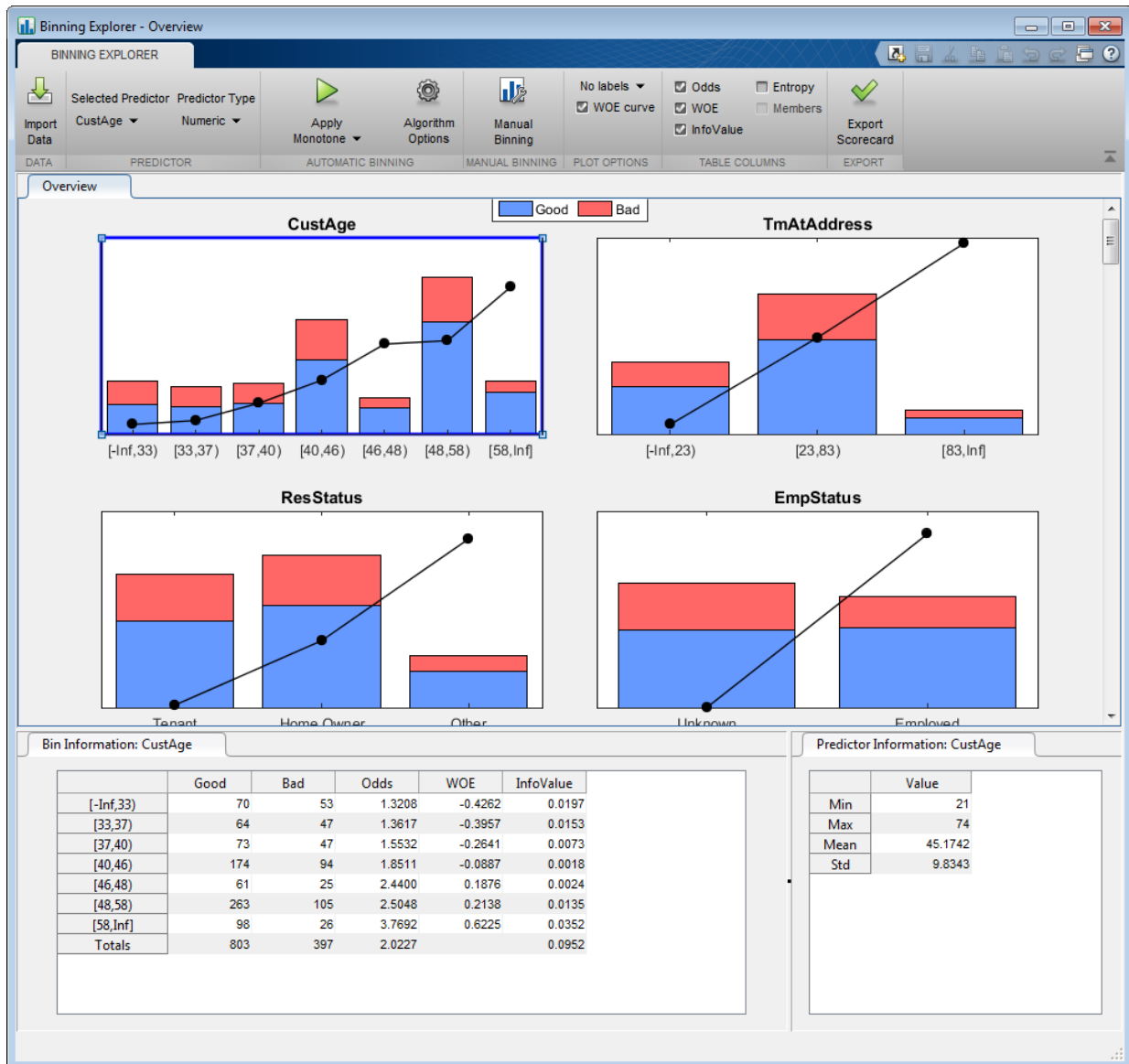
Under **Step 2**, optionally set the **Variable Type** for each of the predictors. By default, the last column in the data ('status' in this example) is set to 'Response'. The response value with the highest count (0 in this example) is set to 'Good'. All other variables are considered predictors. However, in this example, because 'CustID' is not a predictor, set the **Variable Type** column for 'CustID' to **Do not include**.

Under **Step 3**, leave **Monotone** as the default initial binning algorithm.

Click **Import Data** to complete the import operation. Automatic binning using the selected algorithm is applied to all predictors as they are imported into **Binning Explorer**.

The bins are plotted and displayed for each predictor. By clicking to select an individual predictor plot, the details for that predictor plot display in the **Bin Information** and **Predictor Information** panes at the bottom of the app.

### 3 Managing Consumer Credit Risk Using the Binning Explorer for Credit Scorecards



**Binning Explorer** performs automatic binning for every predictor variable, using the default 'Monotone' algorithm with default algorithm options. A monotonic,

ideally linear trend in the Weight of Evidence (WOE) is often desirable for credit scorecards because this translates into linear points for a given predictor. WOE trends are visualized on the plots for each predictor in **Binning Explorer**.

Perform some initial data exploration. Inquire about predictor statistics for the 'ResStatus' categorical variable.

Click the **ResStatus** plot. The **Bin Information** pane contains the “Good” and “Bad” frequencies and other bin statistics such as weight of evidence (WOE).

Bin Information: ResStatus					
	Good	Bad	Odds	WOE	InfoValue
Tenant	307	167	1.8383	-0.0956	0.0037
Home Owner	365	177	2.0621	0.0193	1.6820e-04
Other	131	53	2.4717	0.2005	0.0059
Totals	803	397	2.0227		0.0098

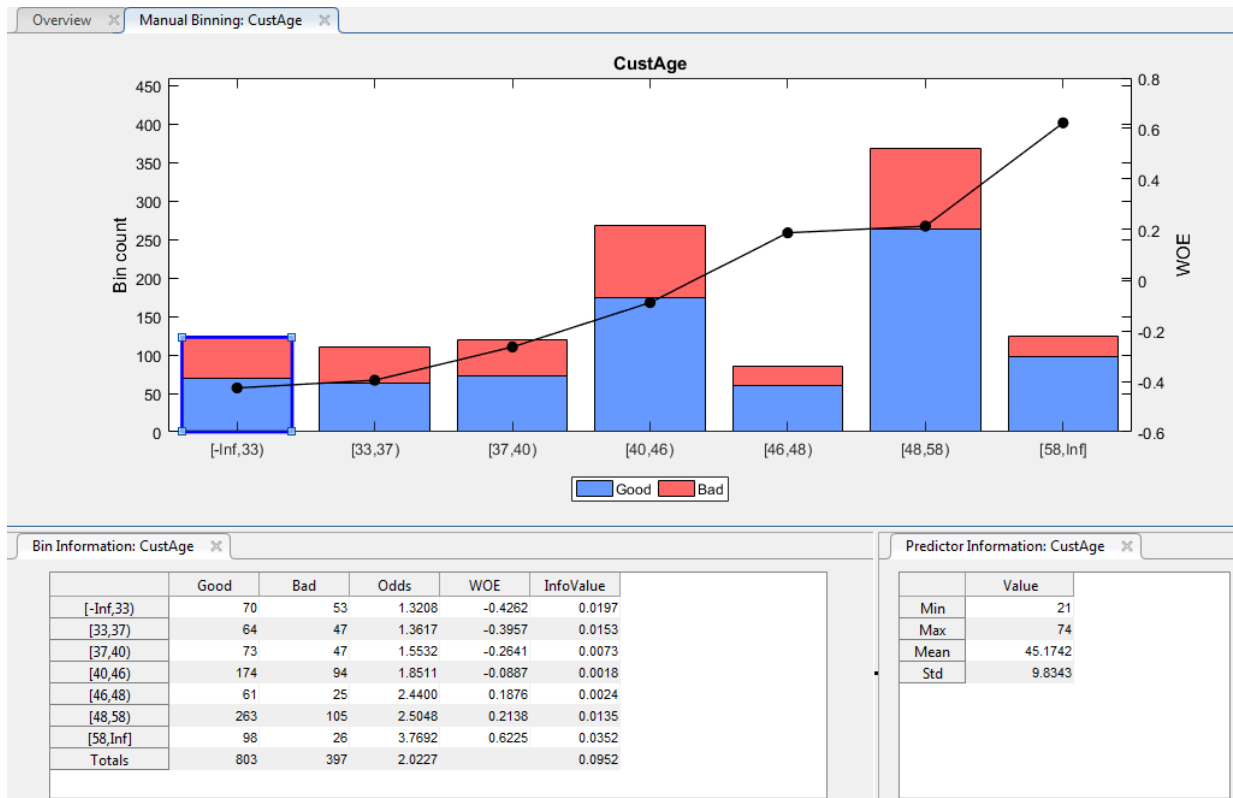
For numeric data, the same statistics are displayed. Click the **CustIncome** plot. The **Bin Information** is updated with the information about **CustIncome**.

Bin Information: CustIncome					
	Good	Bad	Odds	WOE	InfoValue
[-Inf,29000)	53	58	0.9138	-0.7946	0.0636
[29000,33000)	74	49	1.5102	-0.2922	0.0091
[33000,35000)	68	36	1.8889	-0.0684	4.1042e-04
[35000,40000)	193	98	1.9694	-0.0267	1.7359e-04
[40000,42000)	68	34	2	-0.0113	1.0819e-05
[42000,47000)	164	66	2.4848	0.2058	0.0078
[47000,Inf]	183	56	3.2679	0.4797	0.0417
Totals	803	397	2.0227		0.1228

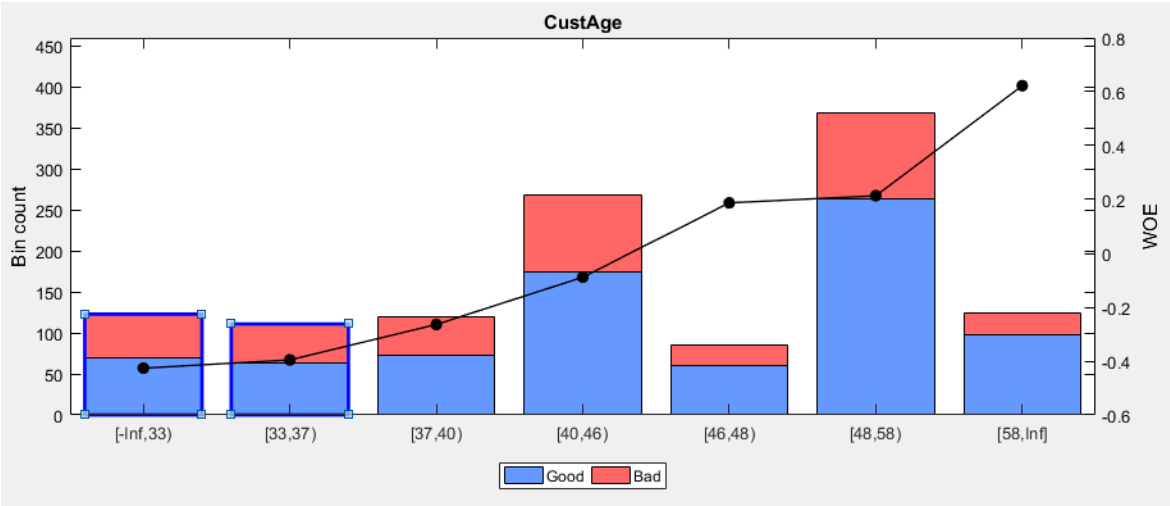
### Step 3. Fine-tune the bins using manual binning in Binning Explorer.

Click the **CustAge** predictor plot. Notice that bins 1 and 2 have similar WOE, as do bins 5 and 6.

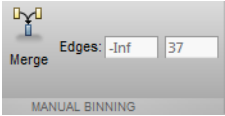
### 3 Managing Consumer Credit Risk Using the Binning Explorer for Credit Scorecards



To merge bins 1 and 2, from the **Binning Explorer** toolbar, click **Manual Binning** to open the selected predictor in a new tabbed window. Alternatively, double-click the predictor plot to open the **Manual Binning** tab. Select bin 1 and 2 for merging by using **Ctrl + click** to multiselect these bins to display with blue outlines.



On the **Binning Explorer** toolbar, the **Edges** text boxes display values for the edges of the selected bins to merge.



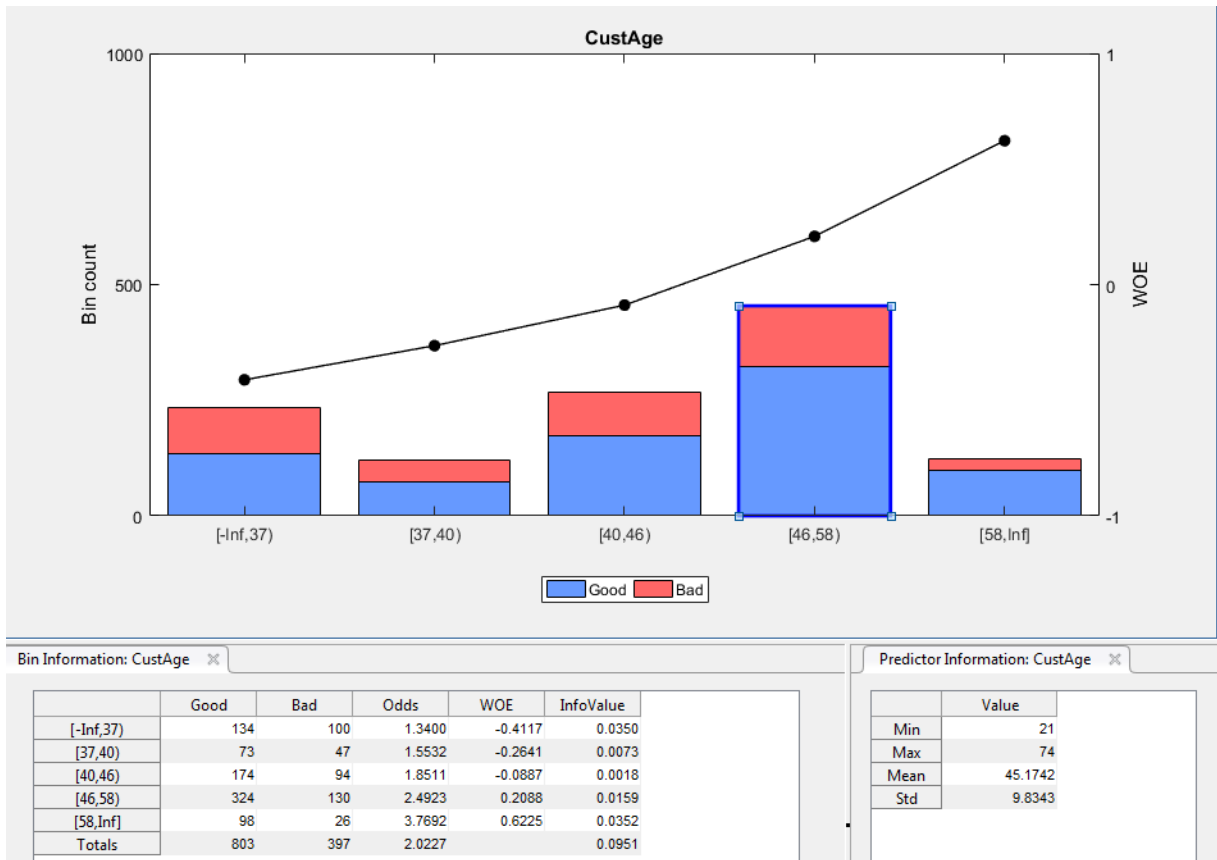
Click **Merge** to finish merging bins 1 and 2. The **CustAge** predictor plot is updated for the new bin information and the details in the **Bin Information** and **Predictor Information** panes are also updated.

### 3 Managing Consumer Credit Risk Using the Binning Explorer for Credit Scorecards



Next, merge bins 4 and 5, because they also have similar WOE.





The **CustAge** predictor plot is updated with the new bin information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

Repeat this merge operation for the following bins that have similar WOE's:

- For **CustIncome**, merge bins 3, 4 and 5.
- For **TmWBank**, merge bins 2 and 3.
- For **AMBalance**, merge bins 2 and 3.

Now the bins for all predictors have close-to-linear WOE trends.

#### Step 4. Export the creditcard object from Binning Explorer.

After you complete your binning assignments, using **Binning Explorer**, click **Export** and provide a `creditcard` object name. The `creditcard` object (`sc`) is saved to the MATLAB workspace.

#### Step 5. Fit a logistic regression model.

Use the `fitmodel` function to fit a logistic regression model to the WOE data. `fitmodel` internally bins the training data, transforms it into WOE values, maps the response variable so that 'Good' is 1, and fits a linear logistic regression model. By default, `fitmodel` uses a stepwise procedure to determine which predictors belong in the model.

```
sc = fitmodel(sc);
```

1. Adding CustIncome, Deviance = 1490.8954, Chi2Stat = 32.545914, PValue = 1.1640961e-08
2. Adding TmWBank, Deviance = 1467.3249, Chi2Stat = 23.570535, PValue = 1.2041739e-06
3. Adding AMBalance, Deviance = 1455.858, Chi2Stat = 11.466846, PValue = 0.00070848829
4. Adding EmpStatus, Deviance = 1447.6148, Chi2Stat = 8.2432677, PValue = 0.0040903428
5. Adding CustAge, Deviance = 1442.06, Chi2Stat = 5.5547849, PValue = 0.018430237
6. Adding ResStatus, Deviance = 1437.9435, Chi2Stat = 4.1164321, PValue = 0.042468555
7. Adding OtherCC, Deviance = 1433.7372, Chi2Stat = 4.2063597, PValue = 0.040272676

Generalized Linear regression model:

```
logit(status) ~ 1 + CustAge + ResStatus + EmpStatus + CustIncome + TmWBank + OtherCC + AMBalance
Distribution = Binomial
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	0.7024	0.064	10.975	5.0407e-28
CustAge	0.61562	0.24783	2.4841	0.012988
ResStatus	1.3776	0.65266	2.1107	0.034799
EmpStatus	0.88592	0.29296	3.024	0.0024946
CustIncome	0.69836	0.21715	3.216	0.0013001
TmWBank	1.106	0.23266	4.7538	1.9958e-06
OtherCC	1.0933	0.52911	2.0662	0.038806
AMBalance	1.0437	0.32292	3.2322	0.0012285

1200 observations, 1192 error degrees of freedom

Dispersion: 1

Chi^2-statistic vs. constant model: 89.7, p-value = 1.42e-16

#### Step 6. Review and format scorecard points.

After fitting the logistic model, the points are unscaled by default and come directly from the combination of WOE values and model coefficients. Use the `displaypoints` function to summarize the scorecard points.

```
p1 = displaypoints(sc);
```

disp(p1)

Predictors	Bin	Points
'CustAge'	'[-Inf,37)'	-0.15314
'CustAge'	'[37,40)'	-0.062247
'CustAge'	'[40,46)'	0.045763
'CustAge'	'[46,58)'	0.22888
'CustAge'	'[58,Inf]'	0.48354
'ResStatus'	'Tenant'	-0.031302
'ResStatus'	'Home Owner'	0.12697
'ResStatus'	'Other'	0.37652
'EmpStatus'	'Unknown'	-0.076369
'EmpStatus'	'Employed'	0.31456
'CustIncome'	'[-Inf,29000)'	-0.45455
'CustIncome'	'[29000,33000)'	-0.1037
'CustIncome'	'[33000,42000)'	0.077768
'CustIncome'	'[42000,47000)'	0.24406
'CustIncome'	'[47000,Inf]'	0.43536
'TmWBank'	'[-Inf,12)'	-0.18221
'TmWBank'	'[12,45)'	-0.038279
'TmWBank'	'[45,71)'	0.39569
'TmWBank'	'[71,Inf]'	0.95074
'OtherCC'	'No'	-0.193
'OtherCC'	'Yes'	0.15868
'AMBalance'	'[-Inf,558.88)'	0.3552
'AMBalance'	'[558.88,1597.44)'	-0.026797
'AMBalance'	'[1597.44,Inf]'	-0.21168

Use `modifybins` to give the bins more descriptive labels.

```
sc = modifybins(sc,'CustAge','BinLabels',...
{'Up to 36' '37 to 39' '40 to 45' '46 to 57' '58 and up'});

sc = modifybins(sc,'CustIncome','BinLabels',...
{'Up to 28999' '29000 to 32999' '33000 to 41999' '42000 to 46999' '47000 and up'});

sc = modifybins(sc,'TmWBank','BinLabels',...
{'Up to 11' '12 to 44' '45 to 70' '71 and up'});

sc = modifybins(sc,'AMBalance','BinLabels',...
{'Up to 558.87' '558.88 to 1597.43' '1597.44 and up'});

p1 = displaypoints(sc);
disp(p1)
```

Predictors	Bin	Points
'CustAge'	'Up to 36'	-0.15314
'CustAge'	'37 to 39'	-0.062247
'CustAge'	'40 to 45'	0.045763
'CustAge'	'46 to 57'	0.22888
'CustAge'	'58 and up'	0.48354
'ResStatus'	'Tenant'	-0.031302
'ResStatus'	'Home Owner'	0.12697

'ResStatus'	'Other'	0.37652
'EmpStatus'	'Unknown'	-0.076369
'EmpStatus'	'Employed'	0.31456
'CustIncome'	'Up to 28999'	-0.45455
'CustIncome'	'29000 to 32999'	-0.1037
'CustIncome'	'33000 to 41999'	0.077768
'CustIncome'	'42000 to 46999'	0.24406
'CustIncome'	'47000 and up'	0.43536
'TmWBank'	'Up to 11'	-0.18221
'TmWBank'	'12 to 44'	-0.038279
'TmWBank'	'45 to 70'	0.39569
'TmWBank'	'71 and up'	0.95074
'OtherCC'	'No'	-0.193
'OtherCC'	'Yes'	0.15868
'AMBalance'	'Up to 558.87'	0.3552
'AMBalance'	'558.88 to 1597.43'	-0.026797
'AMBalance'	'1597.44 and up'	-0.21168

Points are usually scaled and are also often rounded. To round and scale the points, use the `formatpoints` function. For example, you can set a target level of points corresponding to a target odds level and also set the required points-to-double-the-odds (PDO).

```
TargetPoints = 500;
TargetOdds = 2;
PDO = 50; % Points to double the odds

sc = formatpoints(sc, 'PointsOddsAndPDO', [TargetPoints TargetOdds PDO]);
p2 = displaypoints(sc);
disp(p2)
```

Predictors	Bin	Points
'CustAge'	'Up to 36'	53.239
'CustAge'	'37 to 39'	59.796
'CustAge'	'40 to 45'	67.587
'CustAge'	'46 to 57'	80.796
'CustAge'	'58 and up'	99.166
'ResStatus'	'Tenant'	62.028
'ResStatus'	'Home Owner'	73.445
'ResStatus'	'Other'	91.446
'EmpStatus'	'Unknown'	58.777
'EmpStatus'	'Employed'	86.976
'CustIncome'	'Up to 28999'	31.497
'CustIncome'	'29000 to 32999'	56.805
'CustIncome'	'33000 to 41999'	69.896

```
'CustIncome' '42000 to 46999' 81.891
'CustIncome' '47000 and up' 95.69
'TmWBank' 'Up to 11' 51.142
'TmWBank' '12 to 44' 61.524
'TmWBank' '45 to 70' 92.829
'TmWBank' '71 and up' 132.87
'OtherCC' 'No' 50.364
'OtherCC' 'Yes' 75.732
'AMBalance' 'Up to 558.87' 89.908
'AMBalance' '558.88 to 1597.43' 62.353
'AMBalance' '1597.44 and up' 49.016
```

**Step 7. Score the data.**

Use the `score` function to compute the scores for the training data. You can also pass an optional `data` input. to `score`, for example, validation data. The points per predictor for each customer are provided as an optional output.

```
[Scores,Points] = score(sc);
disp(Scores(1:10))
disp(Points(1:10,:))
```

```
528.2044
554.8861
505.2406
564.0717
554.8861
586.1904
441.8755
515.8125
524.4553
508.3169
```

CustAge	ResStatus	EmpStatus	CustIncome	TmWBank	OtherCC	AMBalance
80.796	62.028	58.777	95.69	92.829	75.732	62.353
99.166	73.445	86.976	95.69	61.524	75.732	62.353
80.796	62.028	86.976	69.896	92.829	50.364	62.353
80.796	73.445	86.976	95.69	61.524	75.732	89.908
99.166	73.445	86.976	95.69	61.524	75.732	62.353
99.166	73.445	86.976	95.69	92.829	75.732	62.353
53.239	73.445	58.777	56.805	61.524	75.732	62.353
80.796	91.446	86.976	95.69	61.524	50.364	49.016
80.796	62.028	58.777	95.69	61.524	75.732	89.908
80.796	73.445	58.777	95.69	61.524	75.732	62.353

**Step 8. Calculate the probability of default.**

To calculate the probability of default, use the `probdefault` function.

```
pd = probdefault(sc);
```

Define the probability of being “Good” and plot the predicted odds versus the formatted scores. Visually analyze that the target points and target odds match and that the points-to-double-the-odds (PDO) relationship holds.

```
ProbGood = 1-pd;
PredictedOdds = ProbGood./pd;

figure
scatter(Scores,PredictedOdds)
title('Predicted Odds vs. Score')
xlabel('Score')
ylabel('Predicted Odds')

hold on

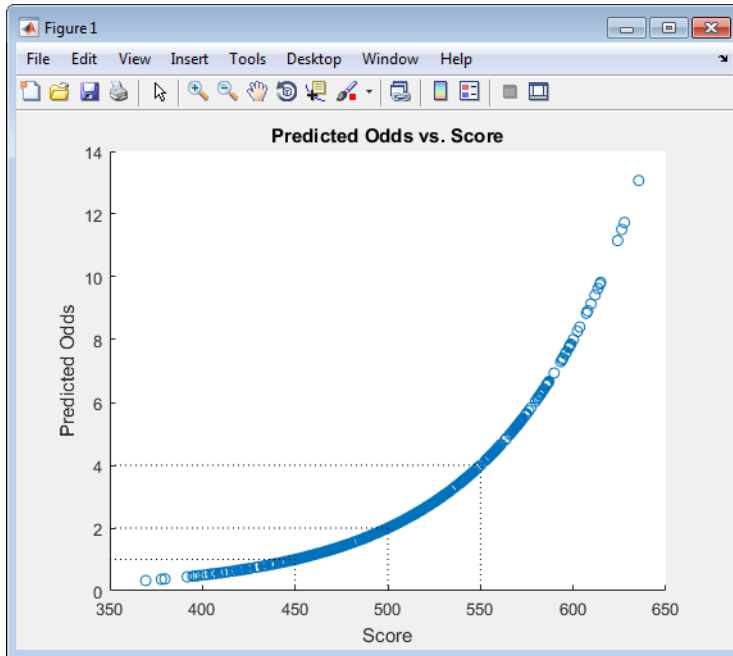
xLimits = xlim;
yLimits = ylim;

% Target points and odds
plot([TargetPoints TargetPoints],[yLimits(1) TargetOdds],'k:')
plot([xLimits(1) TargetPoints],[TargetOdds TargetOdds],'k:')

% Target points plus PDO
plot([TargetPoints+PDO TargetPoints+PDO],[yLimits(1) 2*TargetOdds],'k:')
plot([xLimits(1) TargetPoints+PDO],[2*TargetOdds 2*TargetOdds],'k:')

% Target points minus PDO
plot([TargetPoints-PDO TargetPoints-PDO],[yLimits(1) TargetOdds/2],'k:')
plot([xLimits(1) TargetPoints-PDO],[TargetOdds/2 TargetOdds/2],'k:')

hold off
```



**Step 9. Validate the credit scorecard model using the CAP, ROC, and Kolmogorov-Smirnov statistic**

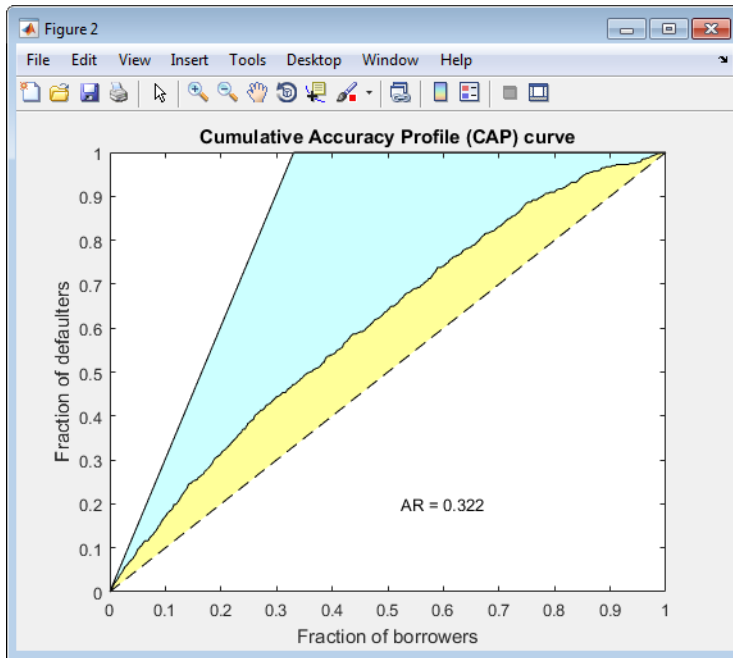
The `creditscorecard` object supports three validation methods, the Cumulative Accuracy Profile (CAP), the Receiver Operating Characteristic (ROC), and the Kolmogorov-Smirnov (KS) statistic. For more information on CAP, ROC, and KS, see `validatemodel`.

```
[Stats,T] = validatemodel(sc, 'Plot', {'CAP', 'ROC', 'KS'});
disp(Stats)
disp(T(1:15,:))
```

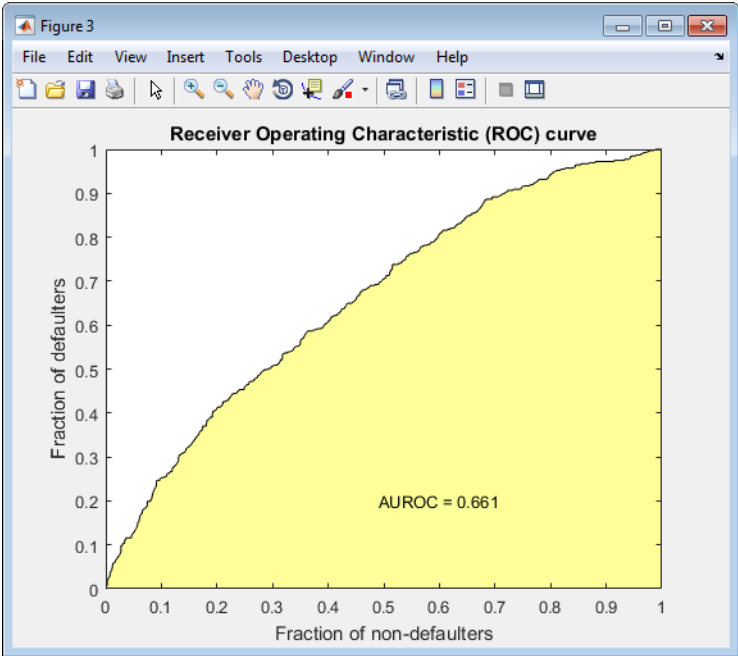
Measure		Value						
'Accuracy Ratio'		0.32225						
'Area under ROC curve'		0.66113						
'KS statistic'		0.22324						
'KS score'		499.18						
Scores	ProbDefault	TrueBads	FalseBads	TrueGoods	FalseGoods	Sensitivity	FalseAlarm	PctObs
369.4	0.7535	0	1	802	397	0	0.0012453	0.00083333

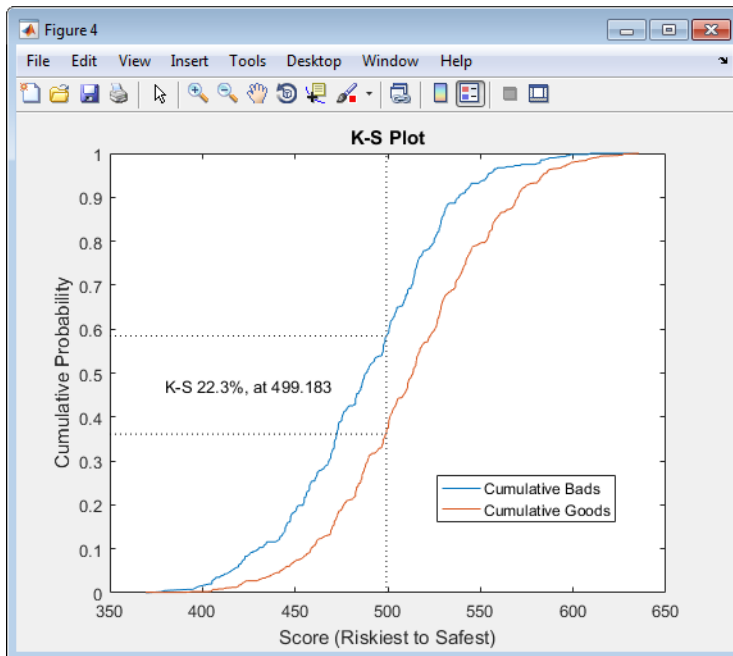
### 3 Managing Consumer Credit Risk Using the Binning Explorer for Credit Scorecards

377.86	0.73107	1	1	802	396	0.0025189	0.0012453	0.0016667
379.78	0.7258	2	1	802	395	0.0050378	0.0012453	0.0025
391.81	0.69139	3	1	802	394	0.0075567	0.0012453	0.0033333
394.77	0.68259	3	2	801	394	0.0075567	0.0024907	0.0041667
395.78	0.67954	4	2	801	393	0.010076	0.0024907	0.005
396.95	0.67598	5	2	801	392	0.012594	0.0024907	0.0058333
398.37	0.67167	6	2	801	391	0.015113	0.0024907	0.0066667
401.26	0.66276	7	2	801	390	0.017632	0.0024907	0.0075
403.23	0.65664	8	2	801	389	0.020151	0.0024907	0.0083333
405.09	0.65081	8	3	800	389	0.020151	0.003736	0.0091667
405.15	0.65062	11	5	798	386	0.027708	0.0062267	0.013333
405.37	0.64991	11	6	797	386	0.027708	0.007472	0.014167
406.18	0.64735	12	6	797	385	0.030227	0.007472	0.015
407.14	0.64433	13	6	797	384	0.032746	0.007472	0.015833









#### See Also

`autobinning` | `bindata` | `bininfo` | `creditscorecard` | `displaypoints` | `fitmodel` | `formatpoints` | `modifybins` | `modifypredictor` | `plotbins` | `predictorinfo` | `probdefault` | `score` | `setmodel` | `validatemodel`

#### Related Examples

- “Troubleshooting Credit Scorecard Results” (Financial Toolbox)
- “Credit Rating by Bagging Decision Trees” (Statistics and Machine Learning Toolbox)
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-44

#### More About

- “About Credit Scorecards” (Financial Toolbox)
- “Credit Scorecard Modeling Workflow” (Financial Toolbox)

- Monotone Adjacent Pooling Algorithm (MAPA) (Financial Toolbox)
- `creditscorecard`

### **External Websites**

- Credit Scorecard Modeling Using the Binning Explorer App (6 min 30 sec)

## Stress Testing of Consumer Credit Default Probabilities Using Panel Data

This example shows how to work with consumer (retail) credit panel data to visualize observed default rates at different levels. It also shows how to fit a model to predict probabilities of default and perform a stress-testing analysis.

The panel data set of consumer loans enables you to identify default rate patterns for loans of different ages, or years on books. You can use information about a score group to distinguish default rates for different score levels. In addition, you can use macroeconomic information to assess how the state of the economy affects consumer loan default rates.

A standard logistic regression model, a type of generalized linear model, is fitted to the retail credit panel data with and without macroeconomic predictors. The example describes how to fit a more advanced model to account for panel data effects, a generalized linear mixed effects model. However, the panel effects are negligible for the data set in this example and the standard logistic model is preferred for efficiency.

The standard logistic regression model predicts probabilities of default for all score levels, years on books, and macroeconomic variable scenarios. When the standard logistic regression model is used for a stress-testing analysis, the model predicts probabilities of default for a given baseline, as well as default probabilities for adverse and severely adverse macroeconomic scenarios.

### Panel Data Description

The main data set (`data`) contains the following variables:

- `ID`: Loan identifier.
- `ScoreGroup`: Credit score at the beginning of the loan, discretized into three groups: High Risk, Medium Risk, and Low Risk.
- `YOB`: Years on books.
- `Default`: Default indicator. This is the response variable.
- `Year`: Calendar year.

There is also a small data set (`dataMacro`) with macroeconomic data for the corresponding calendar years:

- `Year`: Calendar year.

- **GDP:** Gross domestic product growth (year over year).
- **Market:** Market return (year over year).

The variables **YOB**, **Year**, **GDP**, and **Market** are observed at the end of the corresponding calendar year. The score group is a discretization of the original credit score when the loan started. A value of 1 for **Default** means that the loan defaulted in the corresponding calendar year.

There is also a third data set (**dataMacroStress**) with baseline, adverse, and severely adverse scenarios for the macroeconomic variables. This table is used for the stress-testing analysis.

This example uses simulated data, but the same approach has been successfully applied to real data sets.

### Load the Panel Data

Load the data and view the first 10 and last 10 rows of the table. The panel data is stacked, in the sense that observations for the same ID are stored in contiguous rows, creating a tall, thin table. The panel is unbalanced, because not all IDs have the same number of observations.

```
load RetailCreditPanelData.mat

fprintf('\nFirst ten rows:\n')
disp(data(1:10,:))

fprintf('\nLast ten rows:\n')
disp(data(end-9:end,:))

nRows = height(data);
UniqueIDs = unique(data.ID);
nIDs = length(UniqueIDs);

fprintf('Total number of IDs: %d\n',nIDs)
fprintf('Total number of rows: %d\n',nRows)
```

```
First ten rows:
   ID   ScoreGroup   YOB   Default   Year
   --   -
   1     Low Risk     1     0     1997
```

1	Low Risk	2	0	1998
1	Low Risk	3	0	1999
1	Low Risk	4	0	2000
1	Low Risk	5	0	2001
1	Low Risk	6	0	2002
1	Low Risk	7	0	2003
1	Low Risk	8	0	2004
2	Medium Risk	1	0	1997
2	Medium Risk	2	0	1998

Last ten rows:

ID	ScoreGroup	YOB	Default	Year
96819	High Risk	6	0	2003
96819	High Risk	7	0	2004
96820	Medium Risk	1	0	1997
96820	Medium Risk	2	0	1998
96820	Medium Risk	3	0	1999
96820	Medium Risk	4	0	2000
96820	Medium Risk	5	0	2001
96820	Medium Risk	6	0	2002
96820	Medium Risk	7	0	2003
96820	Medium Risk	8	0	2004

Total number of IDs: 96820

Total number of rows: 646724

#### Default Rates by Score Groups and Years on Books

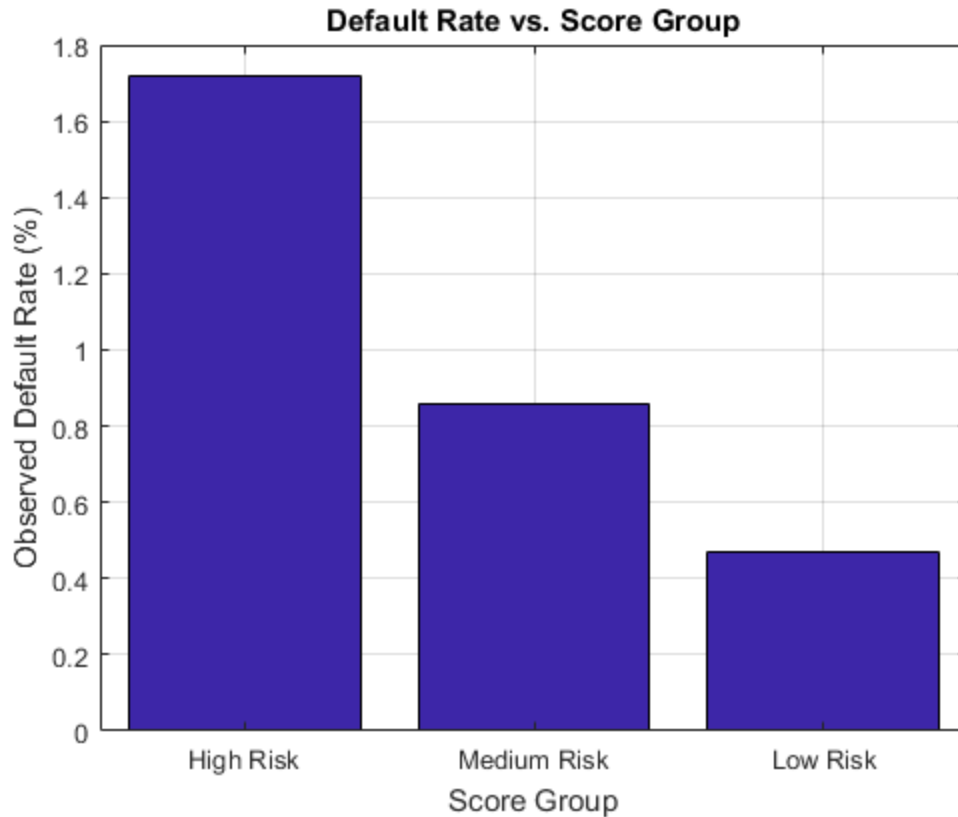
Use the credit score group as a grouping variable to compute the observed default rate for each score group. For this, use the `varfun` function to compute the mean of the `Default` variable, grouping by the `ScoreGroup` variable. Plot the results on a bar chart. As expected, the default rate goes down as the credit quality improves.

```
DefRateByScore = varfun(@mean,data,'InputVariables','Default',...  
    'GroupingVariables','ScoreGroup');  
NumScoreGroups = height(DefRateByScore);  
  
disp(DefRateByScore)  
  
figure;  
bar(double(DefRateByScore.ScoreGroup),DefRateByScore.mean_Default*100)  
set(gca,'XTickLabel',categories(data.ScoreGroup))
```

```

title('Default Rate vs. Score Group')
xlabel('Score Group')
ylabel('Observed Default Rate (%)')
grid on
    
```

ScoreGroup	GroupCount	mean_Default
High Risk	2.0999e+05	0.017167
Medium Risk	2.1743e+05	0.0086006
Low Risk	2.193e+05	0.0046784



Next, compute default rates grouping by years on books (represented by the YOB variable). The resulting rates are conditional one-year default rates. For example, the default rate for the third year on books is the proportion of loans defaulting in the third year, relative to the number of loans that are in the portfolio past the second year. In other words, the default rate for the third year is the number of rows with YOB = 3 and Default = 1, divided by the number of rows with YOB = 3.

Plot the results. There is a clear downward trend, with default rates going down as the number of years on books increases. Years three and four have similar default rates. However, it is unclear from this plot whether this is a characteristic of the loan product or an effect of the macroeconomic environment.

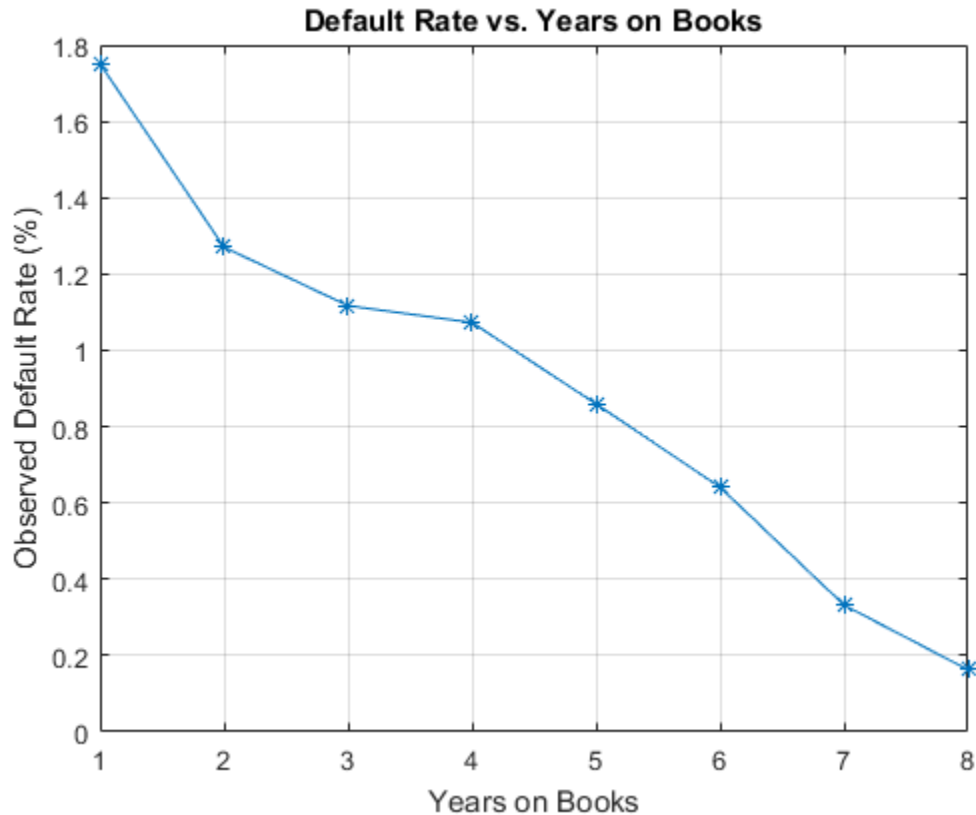
```
DefRateByYOB = varfun(@mean,data,'InputVariables','Default',...
    'GroupingVariables','YOB');
NumYOB = height(DefRateByYOB);

disp(DefRateByYOB)

figure;
plot(double(DefRateByYOB.YOB),DefRateByYOB.mean_Default*100,'-*')
title('Default Rate vs. Years on Books')
xlabel('Years on Books')
ylabel('Observed Default Rate (%)')
grid on
```

YOB	GroupCount	mean_Default
1	96820	0.017507
2	94535	0.012704
3	92497	0.011168
4	91068	0.010728
5	89588	0.0085949
6	88570	0.006413
7	61689	0.0033231
8	31957	0.0016272





Now, group both by score group and number of years on books. Plot the results. The plot shows that all score groups behave similarly as time progresses, with a general downward trend. Years three and four are an exception to the downward trend: the rates flatten for the High Risk group, and go up in year three for the Low Risk group.

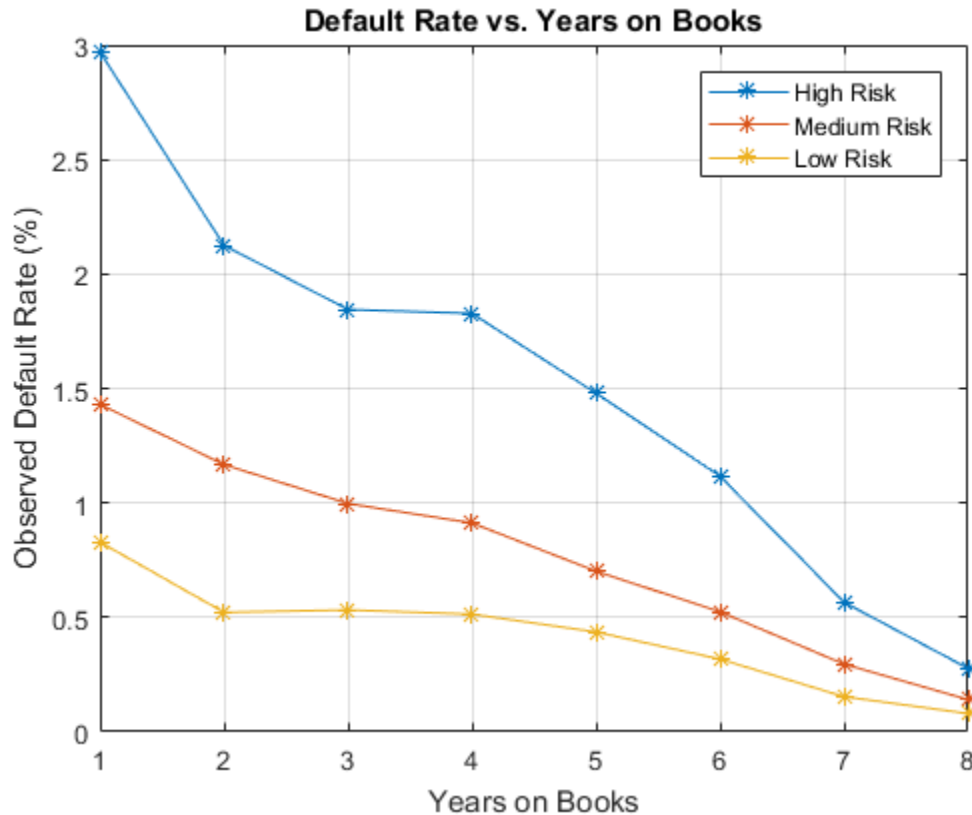
```
DefRateByScoreYOB = varfun(@mean,data,'InputVariables','Default',...
    'GroupingVariables',{'ScoreGroup','YOB'});
```

```
% Display output table to show the way it is structured
% Display only the first 10 rows, for brevity
disp(DefRateByScoreYOB(1:10,:))
disp('    ...')
```

```
DefRateByScoreYOB2 = reshape(DefRateByScoreYOB.mean_Default,...
```

```
    NumYOB, NumScoreGroups);  
figure;  
plot(DefRateByScoreYOB2*100, '-*')  
title('Default Rate vs. Years on Books')  
xlabel('Years on Books')  
ylabel('Observed Default Rate (%)')  
legend(categories(data.ScoreGroup))  
grid on
```

ScoreGroup	YOB	GroupCount	mean_Default
High Risk	1	32601	0.029692
High Risk	2	31338	0.021252
High Risk	3	30138	0.018448
High Risk	4	29438	0.018276
High Risk	5	28661	0.014794
High Risk	6	28117	0.011168
High Risk	7	19606	0.0056615
High Risk	8	10094	0.0027739
Medium Risk	1	32373	0.014302
Medium Risk	2	31775	0.011676
...			



### Years on Books Versus Calendar Years

The data contains three cohorts, or vintages: loans started in 1997, 1998, and 1999. No loan in the panel data started after 1999.

This section shows how to visualize the default rate for each cohort separately. The default rates for all cohorts are plotted, both against the number of years on books and against the calendar year. Patterns in the years on books suggest the loan product characteristics. Patterns in the calendar years suggest the influence of the macroeconomic environment.

From years two through four on books, the curves show different patterns for the three cohorts. When plotted against the calendar year, however, the three cohorts show similar behavior from 2000 through 2002. The curves flatten during that period.

```
% Get IDs of 1997, 1998, and 1999 cohorts
IDs1997 = data.ID(data.YOB==1&data.Year==1997);
IDs1998 = data.ID(data.YOB==1&data.Year==1998);
IDs1999 = data.ID(data.YOB==1&data.Year==1999);
% IDs2000AndUp is unused, it is only computed to show that this is empty,
% no loans started after 1999
IDs2000AndUp = data.ID(data.YOB==1&data.Year>1999);

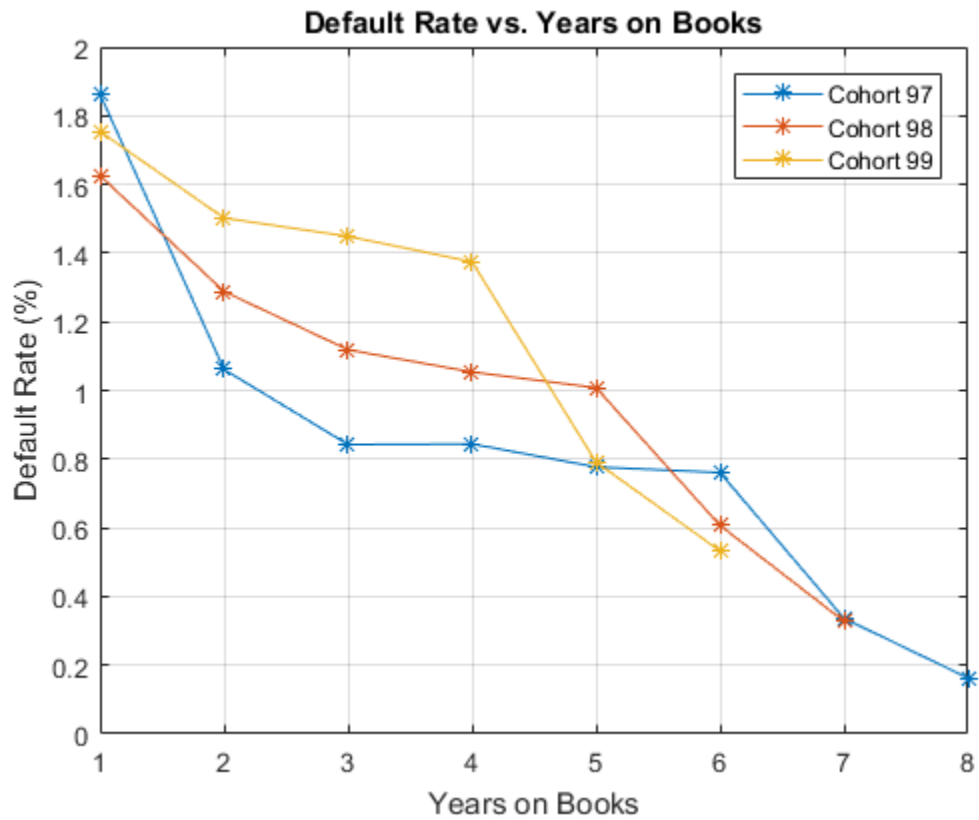
% Get default rates for each cohort separately
ObsDefRate1997 = varfun(@mean,data(ismember(data.ID,IDs1997),:),...
    'InputVariables','Default','GroupingVariables','YOB');

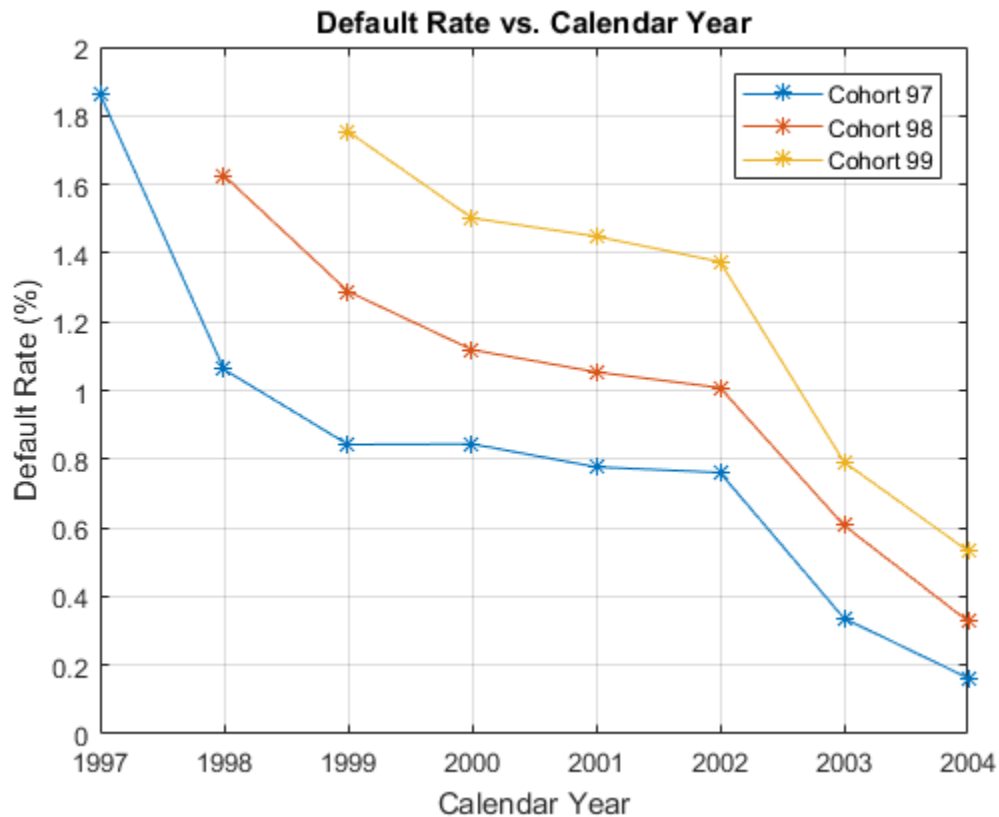
ObsDefRate1998 = varfun(@mean,data(ismember(data.ID,IDs1998),:),...
    'InputVariables','Default','GroupingVariables','YOB');

ObsDefRate1999 = varfun(@mean,data(ismember(data.ID,IDs1999),:),...
    'InputVariables','Default','GroupingVariables','YOB');

% Plot against the years on books
figure;
plot(ObsDefRate1997.YOB,ObsDefRate1997.mean_Default*100,'-*')
hold on
plot(ObsDefRate1998.YOB,ObsDefRate1998.mean_Default*100,'-*')
plot(ObsDefRate1999.YOB,ObsDefRate1999.mean_Default*100,'-*')
hold off
title('Default Rate vs. Years on Books')
xlabel('Years on Books')
ylabel('Default Rate (%)')
legend('Cohort 97','Cohort 98','Cohort 99')
grid on

% Plot against the calendar year
Year = unique(data.Year);
figure;
plot(Year,ObsDefRate1997.mean_Default*100,'-*')
hold on
plot(Year(2:end),ObsDefRate1998.mean_Default*100,'-*')
plot(Year(3:end),ObsDefRate1999.mean_Default*100,'-*')
hold off
title('Default Rate vs. Calendar Year')
xlabel('Calendar Year')
ylabel('Default Rate (%)')
legend('Cohort 97','Cohort 98','Cohort 99')
grid on
```





#### Model of Default Rates Using Score Group and Years on Books

After you visualize the data, you can build predictive models for the default rates.

Split the panel data into training and testing sets, defining these sets based on ID numbers.

```
NumTraining = floor(0.6*nIDs);  
  
rng('default');  
TrainIDInd = randsample(nIDs,NumTraining);  
TrainDataInd = ismember(data.ID,UniqueIDs(TrainIDInd));  
TestDataInd = ~TrainDataInd;
```

The first model uses only score group and number of years on books as predictors of the default rate  $p$ . The odds of defaulting are defined as  $p/(1-p)$ . The logistic model relates the logarithm of the odds, or *log odds*, to the predictors as follows:

$$\log\left(\frac{p}{1-p}\right) = a_H + a_M 1_M + a_L 1_L + b_{YOB} YOB + \epsilon$$

$1_M$  is an indicator with a value 1 for Medium Risk loans and 0 otherwise, and similarly for  $1_L$  for Low Risk loans. This is a standard way of handling a categorical predictor such as ScoreGroup. There is effectively a different constant for each risk level:  $a_H$  for High Risk,  $a_H + a_M$  for Medium Risk, and  $a_H + a_L$  for Low Risk.

To calibrate the model, call the `fitglm` function from Statistics and Machine Learning Toolbox™. The formula above is expressed as

```
Default ~ 1 + ScoreGroup + YOB
```

The `1 + ScoreGroup` terms account for the baseline constant and the adjustments for risk level. Set the optional argument `Distribution` to `binomial` to indicate that a logistic model is desired (that is, a model with log odds on the left side).

```
ModelNoMacro = fitglm(data(TrainDataInd,:),...
    'Default ~ 1 + ScoreGroup + YOB',...
    'Distribution','binomial');
disp(ModelNoMacro)
```

```
Generalized linear regression model:
logit(Default) ~ 1 + ScoreGroup + YOB
Distribution = Binomial
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	-3.2453	0.033768	-96.106	0
ScoreGroup_Medium Risk	-0.7058	0.037103	-19.023	1.1014e-80
ScoreGroup_Low Risk	-1.2893	0.045635	-28.253	1.3076e-175
YOB	-0.22693	0.008437	-26.897	2.3578e-159

388018 observations, 388014 error degrees of freedom

Dispersion: 1

Chi^2-statistic vs. constant model: 1.83e+03, p-value = 0

For any row in the data, the value of  $p$  is not observed, only a 0 or 1 default indicator is observed. The calibration finds model coefficients, and the predicted values of  $p$  for individual rows can be recovered with the `predict` function.

The `Intercept` coefficient is the constant for the `High Risk` level (the  $aH$  term), and the `ScoreGroup_Medium Risk` and `ScoreGroup_Low Risk` coefficients are the adjustments for `Medium Risk` and `Low Risk` levels (the  $aM$  and  $aL$  terms).

The default probability  $p$  and the log odds (the left side of the model) move in the same direction when the predictors change. Therefore, because the adjustments for `Medium Risk` and `Low Risk` are negative, the default rates are lower for better risk levels, as expected. The coefficient for number of years on books is also negative, consistent with the overall downward trend for number of years on books observed in the data.

To account for panel data effects, a more advanced model using mixed effects can be fitted using the `fitglm` function from Statistics and Machine Learning Toolbox™. Although this model is not fitted in this example, the code is very similar:

```
ModelNoMacro = fitglm(data(TrainDataInd,:), ...  
'Default ~ 1 + ScoreGroup + YOB + (1|ID)', ...  
'Distribution', 'binomial');
```

The `(1|ID)` term in the formula adds a *random effect* to the model. This effect is a predictor whose values are not given in the data, but calibrated together with the model coefficients. A random value is calibrated for each ID. This additional calibration requirement substantially increases the computational time to fit the model in this case, because of the very large number of IDs. For the panel data set in this example, the random term has a negligible effect. The variance of the random effects is very small and the model coefficients barely change when the random effect is introduced. The simpler logistic regression model is preferred, because it is faster to calibrate and to predict, and the default rates predicted with both models are essentially the same.

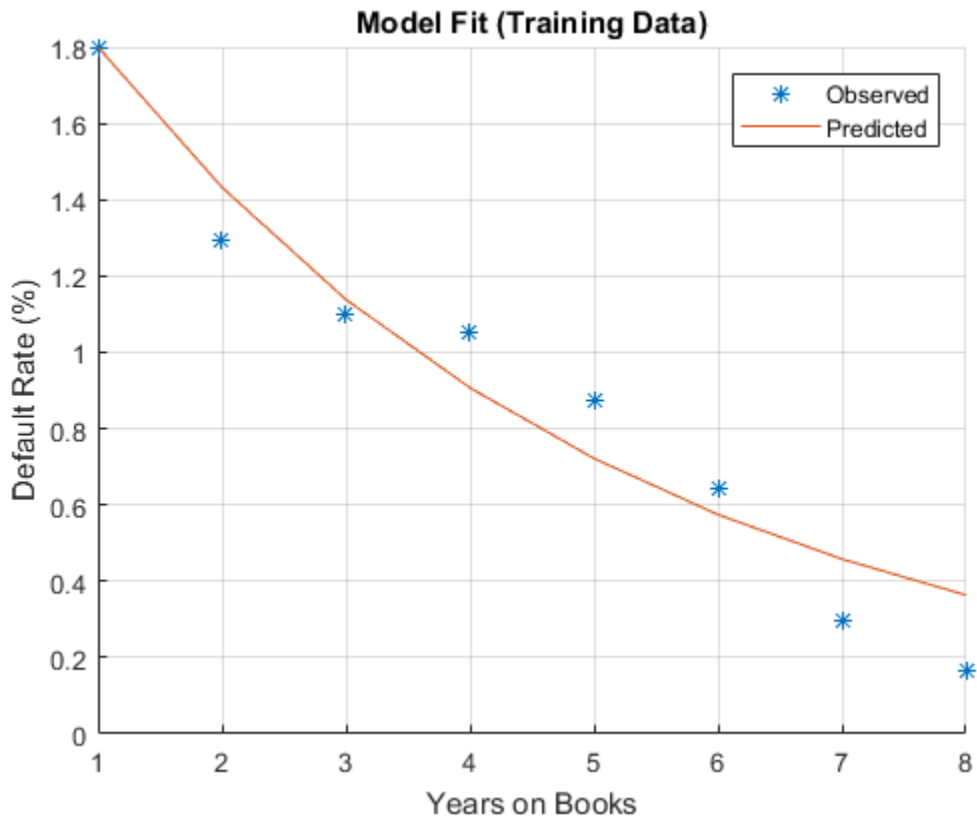
Predict the probability of default for training and testing data.

```
data.PDNoMacro = zeros(height(data),1);  
  
% Predict in-sample  
data.PDNoMacro(TrainDataInd) = predict(ModelNoMacro,data(TrainDataInd,:));  
% Predict out-of-sample  
data.PDNoMacro(TestDataInd) = predict(ModelNoMacro,data(TestDataInd,:));
```

Visualize the in-sample fit.

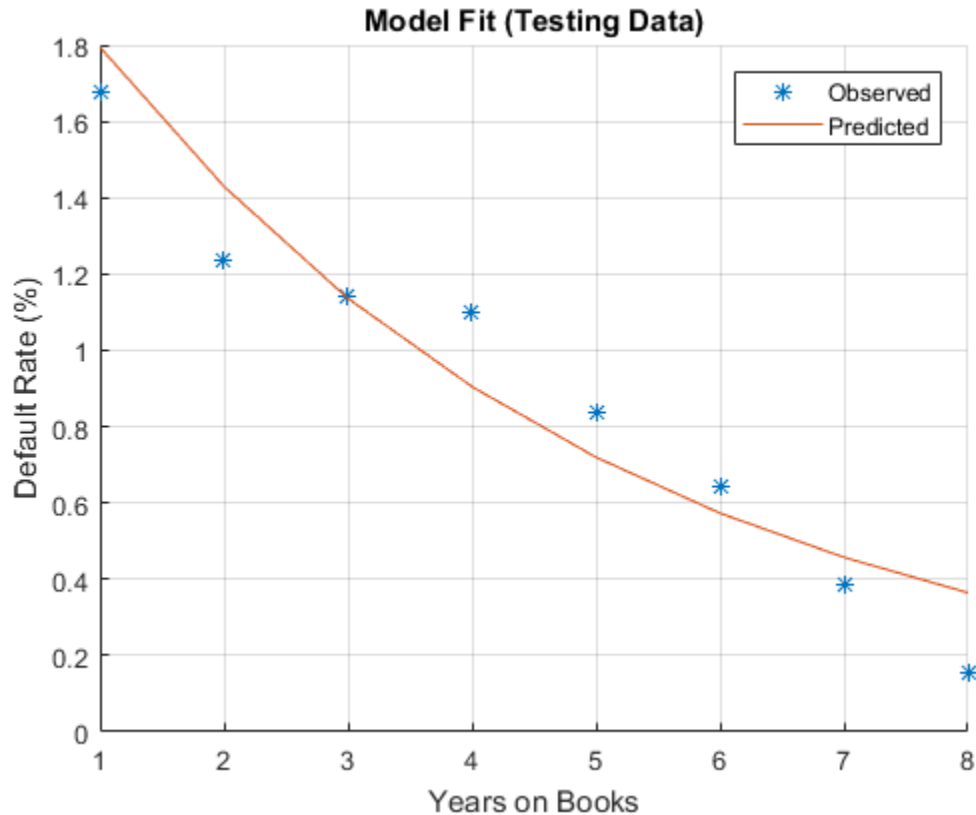


```
PredPDTrainYOB = varfun(@mean,data(TrainDataInd,:),...  
    'InputVariables',{'Default','PDNoMacro'},'GroupingVariables','YOB');  
  
figure;  
scatter(PredPDTrainYOB.YOB,PredPDTrainYOB.mean_Default*100,'*');  
hold on  
plot(PredPDTrainYOB.YOB,PredPDTrainYOB.mean_PDNoMacro*100);  
hold off  
xlabel('Years on Books')  
ylabel('Default Rate (%)')  
legend('Observed','Predicted')  
title('Model Fit (Training Data)')  
grid on
```



Visualize the out-of-sample fit.

```
PredPDTestYOB = varfun(@mean,data(TestDataInd,:),...  
    'InputVariables',{ 'Default', 'PDNoMacro'}, 'GroupingVariables', 'YOB');  
  
figure;  
scatter(PredPDTestYOB.YOB,PredPDTestYOB.mean_Default*100, '*');  
hold on  
plot(PredPDTestYOB.YOB,PredPDTestYOB.mean_PDNoMacro*100);  
hold off  
xlabel('Years on Books')  
ylabel('Default Rate (%)')  
legend('Observed','Predicted')  
title('Model Fit (Testing Data)')  
grid on
```



Visualize the in-sample fit for all score groups. The out-of-sample fit can be computed and visualized in a similar way.

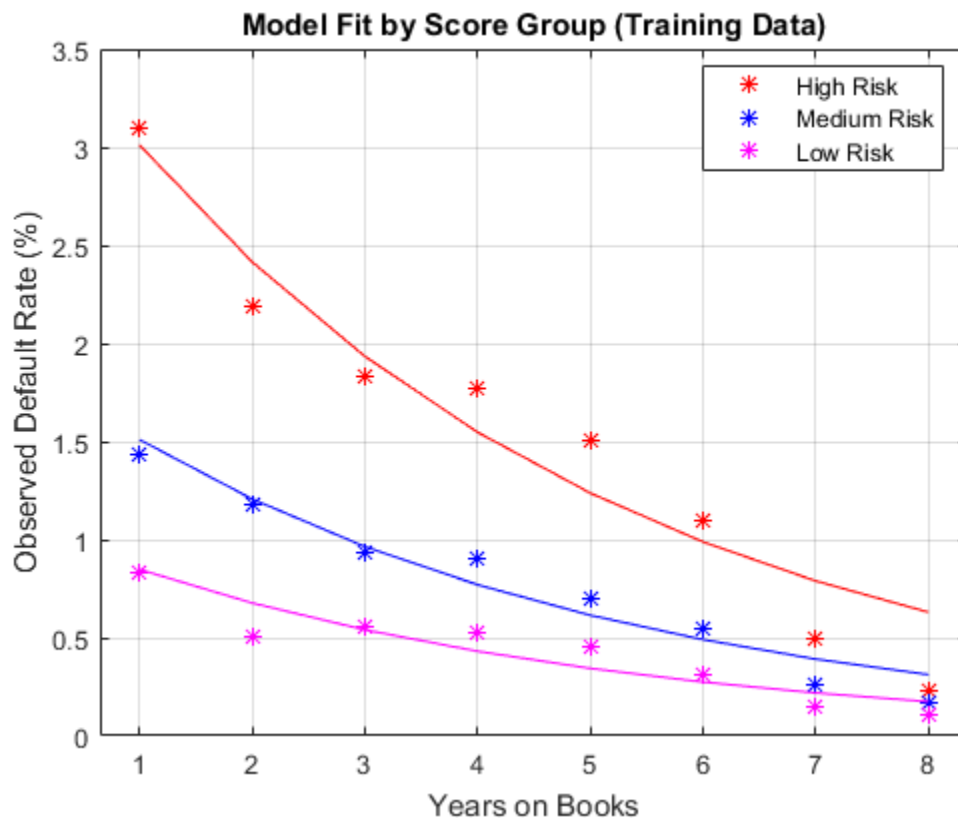
```
PredPDTrainScoreYOB = varfun(@mean,data(TrainDataInd,:),...
    'InputVariables',{'Default','PDNoMacro'},...
    'GroupingVariables',{'ScoreGroup','YOB'});

figure;
hs = gscatter(PredPDTrainScoreYOB.YOB,...
    PredPDTrainScoreYOB.mean_Default*100,...
    PredPDTrainScoreYOB.ScoreGroup,'rbmgk','*');
mean_PDNoMacroMat = reshape(PredPDTrainScoreYOB.mean_PDNoMacro,...
    NumYOB,NumScoreGroups);
hold on
```

```

hp = plot(mean_PDNoMacroMat*100);
for ii=1:NumScoreGroups
    hp(ii).Color = hs(ii).Color;
end
hold off
xlabel('Years on Books')
ylabel('Observed Default Rate (%)')
legend(categories(data.ScoreGroup))
title('Model Fit by Score Group (Training Data)')
grid on

```



#### Model of Default Rates Including Macroeconomic Variables

The trend predicted with the previous model, as a function of years on books, has a very regular decreasing pattern. The data, however, shows some deviations from that trend.

To try to account for those deviations, add the gross domestic product annual growth (represented by the GDP variable) and stock market annual returns (represented by the Market variable) to the model.

$$\log\left(\frac{P}{1-P}\right) = a_H + a_M 1_M + a_L 1_L + b_{YOB} YOB + b_{GDP} GDP + b_{Market} Market + \epsilon$$

Expand the data set to add one column for GDP and one for Market, using the data from the dataMacro table.

```
data.GDP = dataMacro.GDP(data.Year-1996);
data.Market = dataMacro.Market(data.Year-1996);
disp(data(1:10,:))
```

ID	ScoreGroup	YOB	Default	Year	PDNoMacro	GDP	Market
1	Low Risk	1	0	1997	0.0084797	2.72	7.61
1	Low Risk	2	0	1998	0.0067697	3.57	26.24
1	Low Risk	3	0	1999	0.0054027	2.86	18.1
1	Low Risk	4	0	2000	0.0043105	2.43	3.19
1	Low Risk	5	0	2001	0.0034384	1.26	-10.51
1	Low Risk	6	0	2002	0.0027422	-0.59	-22.95
1	Low Risk	7	0	2003	0.0021867	0.63	2.78
1	Low Risk	8	0	2004	0.0017435	1.85	9.48
2	Medium Risk	1	0	1997	0.015097	2.72	7.61
2	Medium Risk	2	0	1998	0.012069	3.57	26.24

Fit the model with the macroeconomic variables by expanding the model formula to include the GDP and the Market variables.

```
ModelMacro = fitglm(data(TrainDataInd,:),...
    'Default ~ 1 + ScoreGroup + YOB + GDP + Market',...
    'Distribution','binomial');
disp(ModelMacro)
```

```
Generalized linear regression model:
logit(Default) ~ 1 + ScoreGroup + YOB + GDP + Market
Distribution = Binomial
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	-2.667	0.10146	-26.287	2.6919e-152
ScoreGroup_Medium Risk	-0.70751	0.037108	-19.066	4.8223e-81
ScoreGroup_Low Risk	-1.2895	0.045639	-28.253	1.2892e-175
YOB	-0.32082	0.013636	-23.528	2.0867e-122
GDP	-0.12295	0.039725	-3.095	0.0019681
Market	-0.0071812	0.0028298	-2.5377	0.011159

```
388018 observations, 388012 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 1.97e+03, p-value = 0
```

Both macroeconomic variables show a negative coefficient, consistent with the intuition that higher economic growth reduces default rates.

Predict the probability of default for the training and testing data.

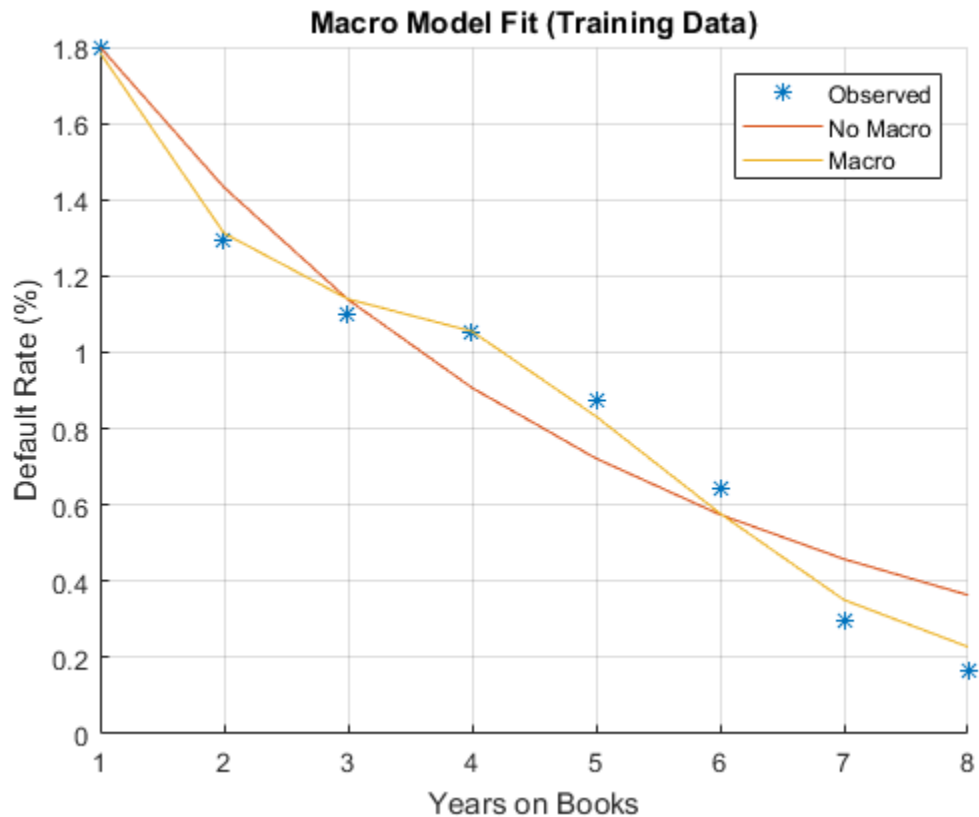
```
data.PDMacro = zeros(height(data),1);

% Predict in-sample
data.PDMacro(TrainDataInd) = predict(ModelMacro,data(TrainDataInd,:));
% Predict out-of-sample
data.PDMacro(TestDataInd) = predict(ModelMacro,data(TestDataInd,:));
```

Visualize the in-sample fit. As desired, the model including macroeconomic variables, or macro model, deviates from the smooth trend predicted by the previous model. The rates predicted with the macro model match more closely with the observed default rates.

```
PredPDTrainYOBBMacro = varfun(@mean,data(TrainDataInd,:),...
    'InputVariables',{'Default','PDMacro'},'GroupingVariables','YOB');

figure;
scatter(PredPDTrainYOBBMacro.YOB,PredPDTrainYOBBMacro.mean_Default*100,'*');
hold on
plot(PredPDTrainYOBBMacro.YOB,PredPDTrainYOBBMacro.mean_PDNoMacro*100); % No Macro
plot(PredPDTrainYOBBMacro.YOB,PredPDTrainYOBBMacro.mean_PDMacro*100); % Macro
hold off
xlabel('Years on Books')
ylabel('Default Rate (%)')
legend('Observed','No Macro','Macro')
title('Macro Model Fit (Training Data)')
grid on
```

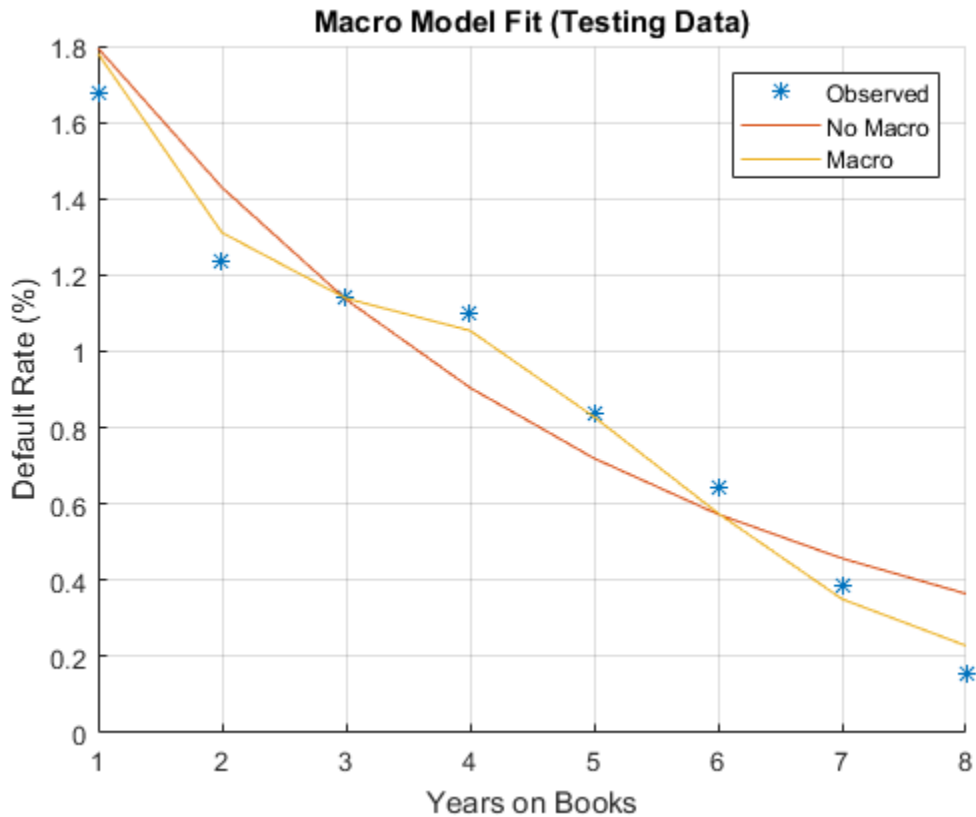


Visualize the out-of-sample fit.

```
PredPDTestYOBMacro = varfun(@mean,data(TestDataInd,:),...
    'InputVariables',{'Default','PDMacro'},'GroupingVariables','YOB');

figure;
scatter(PredPDTestYOBMacro.YOB,PredPDTestYOBMacro.mean_Default*100,'*');
hold on
plot(PredPDTestYOB.YOB,PredPDTestYOB.mean_PDNoMacro*100); % No Macro
plot(PredPDTestYOBMacro.YOB,PredPDTestYOBMacro.mean_PDMacro*100); % Macro
hold off
xlabel('Years on Books')
ylabel('Default Rate (%)')
legend('Observed','No Macro','Macro')
```

```
title('Macro Model Fit (Testing Data)')
grid on
```



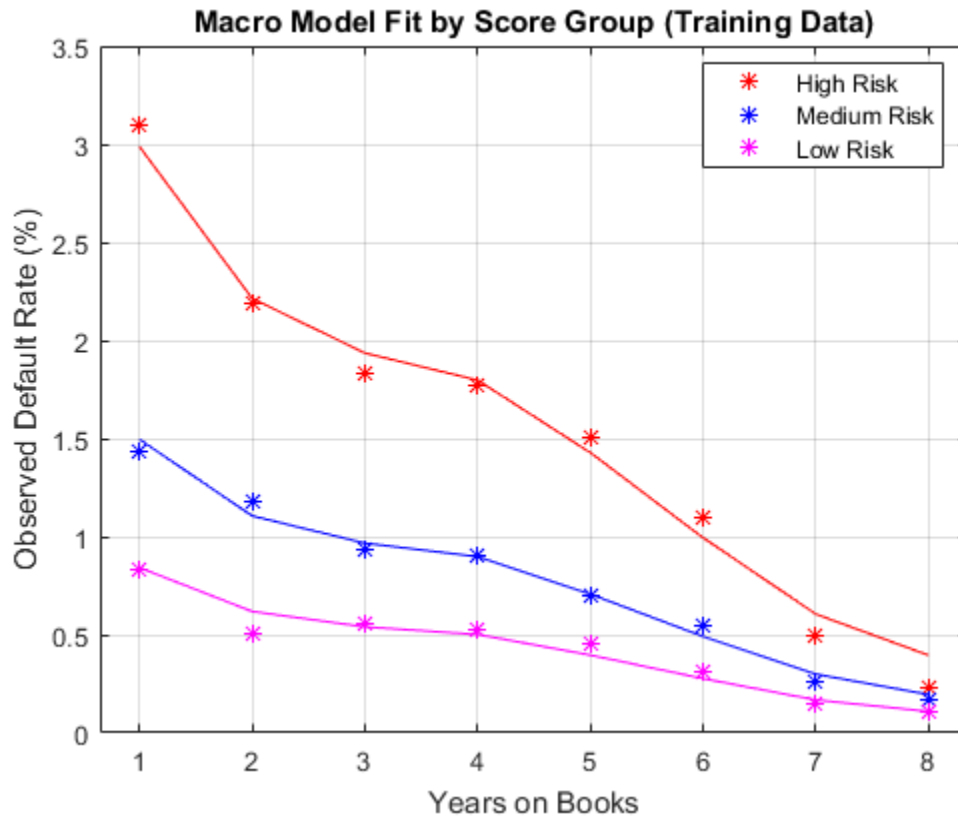
Visualize the in-sample fit for all score groups.

```
PredPDTrainScoreYOBSMacro = varfun(@mean,data(TrainDataInd,:),...
    'InputVariables',{'Default','PDMacro'},...
    'GroupingVariables',{'ScoreGroup','YOB'});

figure;
hs = gscatter(PredPDTrainScoreYOBSMacro.YOB,...
    PredPDTrainScoreYOBSMacro.mean_Default*100,...
    PredPDTrainScoreYOBSMacro.ScoreGroup,'rbmgk','*');
mean_PDMacroMat = reshape(PredPDTrainScoreYOBSMacro.mean_PDMacro,...
    NumYOB,NumScoreGroups);
```



```
hold on
hp = plot(mean_PDMacroMat*100);
for ii=1:NumScoreGroups
    hp(ii).Color = hs(ii).Color;
end
hold off
xlabel('Years on Books')
ylabel('Observed Default Rate (%)')
legend(categories(data.ScoreGroup))
title('Macro Model Fit by Score Group (Training Data)')
grid on
```



### Stress Testing of Probability of Default

Use the fitted macro model to stress-test the predicted probabilities of default.

Assume the following are stress scenarios for the macroeconomic variables provided, for example, by a regulator.

```
disp(dataMacroStress)
```

	GDP	Market
Baseline	2.27	15.02
Adverse	1.31	4.56
Severe	-0.22	-5.64

Set up a basic data table for predicting the probabilities of default. This is a dummy data table, with one row for each combination of score group and number of years on books.

```
dataBaseline = table;  
[ScoreGroup,YOB]=meshgrid(1:NumScoreGroups,1:NumYOB);  
dataBaseline.ScoreGroup = categorical(ScoreGroup(:),1:NumScoreGroups,...  
    categories(data.ScoreGroup),'Ordinal',true);  
dataBaseline.YOB = YOB(:);  
dataBaseline.ID = ones(height(dataBaseline),1);  
dataBaseline.GDP = zeros(height(dataBaseline),1);  
dataBaseline.Market = zeros(height(dataBaseline),1);
```

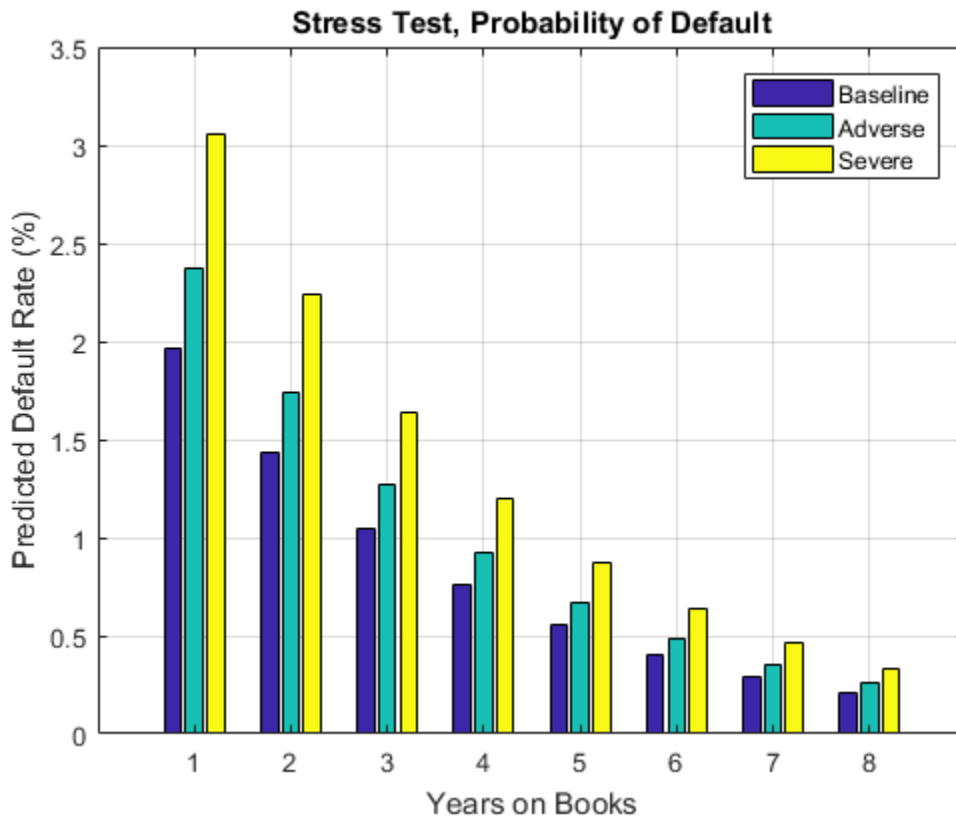
To make the predictions, set the same macroeconomic conditions (baseline, adverse, or severely adverse) for all combinations of score groups and number of years on books.

```
% Predict baseline the probabilities of default  
dataBaseline.GDP(:) = dataMacroStress.GDP('Baseline');  
dataBaseline.Market(:) = dataMacroStress.Market('Baseline');  
dataBaseline.PD = predict(ModelMacro,dataBaseline);  
  
% Predict the probabilities of default in the adverse scenario  
dataAdverse = dataBaseline;  
dataAdverse.GDP(:) = dataMacroStress.GDP('Adverse');  
dataAdverse.Market(:) = dataMacroStress.Market('Adverse');  
dataAdverse.PD = predict(ModelMacro,dataAdverse);  
  
% Predict the probabilities of default in the severely adverse scenario  
dataSevere = dataBaseline;  
dataSevere.GDP(:) = dataMacroStress.GDP('Severe');  
dataSevere.Market(:) = dataMacroStress.Market('Severe');  
dataSevere.PD = predict(ModelMacro,dataSevere);
```

Visualize the average predicted probability of default across score groups under the three alternative regulatory scenarios. Here, all score groups are implicitly weighted equally. However, predictions can also be made at a loan level for any given portfolio to make the predicted default rates consistent with the actual distribution of loans in the portfolio. The same visualization can be produced for each score group separately.

```
PredPDYOB = zeros(NumYOB,3);
PredPDYOB(:,1) = mean(reshape(dataBaseline.PD,NumYOB,NumScoreGroups),2);
PredPDYOB(:,2) = mean(reshape(dataAdverse.PD,NumYOB,NumScoreGroups),2);
PredPDYOB(:,3) = mean(reshape(dataSevere.PD,NumYOB,NumScoreGroups),2);

figure;
bar(PredPDYOB*100);
xlabel('Years on Books')
ylabel('Predicted Default Rate (%)')
legend('Baseline','Adverse','Severe')
title('Stress Test, Probability of Default')
grid on
```



#### References

- 1 Generalized Linear Models documentation: <http://www.mathworks.com/help/stats/generalized-linear-regression.html>.
- 2 Generalized Linear Mixed Effects Models documentation: <http://www.mathworks.com/help/stats/generalized-linear-mixed-effects-models.html>.
- 3 Federal Reserve, Comprehensive Capital Analysis and Review (CCAR): <http://www.federalreserve.gov/bankinforeg/ccar.htm>.
- 4 Bank of England, Stress Testing: <http://www.bankofengland.co.uk/financialstability/pages/fpc/stresstest.aspx>.

- 5 European Banking Authority, EU-Wide Stress Testing: <http://www.eba.europa.eu/risk-analysis-and-data/eu-wide-stress-testing>.

## See Also

`fitglm` | `fitglme`

## Related Examples

- “Credit Rating by Bagging Decision Trees” (Statistics and Machine Learning Toolbox)

## More About

- “About Credit Scorecards” (Financial Toolbox)
- “Credit Scorecard Modeling Workflow” (Financial Toolbox)
- `creditscorecard`



# Corporate Credit Risk Simulations for Portfolios

---

- “Credit Simulation Using Copulas” on page 4-2
- “creditDefaultCopula Simulation Workflow” on page 4-6
- “creditMigrationCopula Simulation Workflow” on page 4-12
- “Modeling Correlated Defaults with Copulas” on page 4-21
- “Analyze the Sensitivity of Concentration to a Given Exposure” on page 4-35
- “Compare Concentration Indices for Random Portfolios” on page 4-38
- “Comparison of the Merton Model Single-Point Approach to the Time-Series Approach” on page 4-42

## Credit Simulation Using Copulas

### In this section...

“Factor Models” on page 4-3

“Supported Simulations” on page 4-3

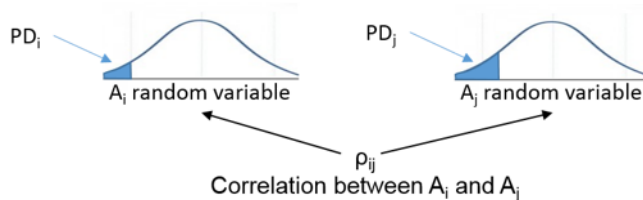
Predicting the credit losses for a counterparty depends on three main elements:

- Probability of default (PD)
- Exposure at default (EAD), the value of the instrument at some future time
- Loss given default (LGD), which is defined as  $1 - \text{Recovery}$

If these quantities are known at future time  $t$ , then the expected loss is  $\text{PD} \times \text{EAD} \times \text{LGD}$ . In this case, you can model the expected loss for a single counterparty by using a binomial distribution. The difficulty arises when you model a portfolio of these counterparties and you want to simulate them with some default correlation.

To simulate correlated defaults, the copula model associates each counterparty with a random variable, called a “latent” variable. These latent variables are correlated using some proxy for their credit worthiness, for example, their stock price. These latent variables are then mapped to default or nondefault outcomes such that the default occurs with probability PD.

This figure summarizes the copula simulation approach.



The random variable  $A_i$  associated to the  $i$ th counterparty falls in the default shaded region with probability  $PD_i$ . If the simulated value falls in that region, it is interpreted as a default. The  $j$ th counterparty follows a similar pattern. If the  $A_i$  and  $A_j$  random variables are highly correlated, they tend to both have high values (no default), or both have low values (fall in the default region). Therefore, there is a default correlation.



## Factor Models

For  $M$  issuers,  $M(M - 1)/2$  correlation parameters are required. For  $M = 1000$ , this is about half a million correlations. One practical variation of the approach is the one-factor model, which makes all the latent variables dependent on a single factor. This factor  $Z$  represents the underlying systemic credit quality in the economy. This model also includes a random idiosyncratic error.

$$A_i = w_i Z + \sqrt{1 - w_i^2} \varepsilon_i$$

This significantly reduces the input-data requirements, because now you need only the  $M$  sensitivities, that is, the weights  $w_1, \dots, w_M$ . If  $Z$  and  $\varepsilon_i$  are standard normal variables, then  $A_i$  is also a standard normal.

An extension of the one-factor model is a multifactor model.

$$A_i = w_{i1} Z_1 + \dots + w_{iK} Z_K + w_{i\varepsilon} \varepsilon_i$$

This model has several factors, each one associated with some underlying credit driver. For example, you can have factors for different regions or countries, or for different industries. Each latent variable is now a combination of several random variables plus the idiosyncratic error (epsilon) again.

When the latent variables  $A_i$  are normally distributed, there is a Gaussian copula. A common alternative is to let the latent variables follow a  $t$  distribution, which leads to a  $t$  copula.  $t$  copulas result in heavier tails than Gaussian copulas. Implied credit correlations are also larger with  $t$  copulas. Switching between these two copula approaches can provide important information on model risk.

## Supported Simulations

Risk Management Toolbox supports simulations for counterparty credit defaults and counterparty credit rating migrations.

### Credit Default Simulation

The `creditDefaultCopula` object is used to simulate and analyze multifactor credit default simulations. These simulations assume that you calculated the main inputs to this model on your own. The main inputs to this model are:

- **PD** — Probability of default
- **EAD** — Exposure at default
- **LGD** — Loss given default ( $1 - Recovery$ )
- **Weights** — Factor and idiosyncratic weights
- **FactorCorrelation** — An optional factor correlation matrix for multifactor models

The `creditDefaultCopula` object enables you to simulate defaults using the multifactor copula and return the results as a distribution of losses on a portfolio and counterparty level. You can also use the `creditDefaultCopula` object to calculate several risk measures at the portfolio level and the risk contributions from individual obligors. The outputs of the `creditDefaultCopula` model and the associated functions are:

- The full simulated distribution of portfolio losses across scenarios and the losses on each counterparty across scenarios. For more information, see `creditDefaultCopula` object properties and `simulate`.
- Risk measures (VaR, CVaR, EL, Std) with confidence intervals. See `portfolioRisk`.
- Risk contributions per counterparty (for EL and CVaR). See `riskContribution`.
- Risk measures and associated confidence bands. See `confidenceBands`.
- Counterparty scenario details for individual losses for each counterparty. See `getScenarios`.

### Credit Rating Migration Simulation

The `creditMigrationCopula` object enables you to simulate changes in credit rating for each counterparty.

The `creditMigrationCopula` object is used to simulate counterparty credit migrations. These simulations assume that you calculated the main inputs to this model on your own. The main inputs to this model are:

- **migrationValues** — Values of the counterparty positions for each credit rating.
- **ratings** — Current credit rating for each counterparty.
- **transitionMatrix** — Matrix of credit rating transition probabilities.
- **LGD** — Loss given default ( $1 - Recovery$ )
- **Weights** — Factor and idiosyncratic model weights

You can also use the `creditMigrationCopula` object to calculate several risk measures at the portfolio level and the risk contributions from individual obligors. The outputs of the `creditMigrationCopula` model and the associated functions are:

- The full simulated distribution of portfolio values. For more information, see `creditMigrationCopula` object properties and `simulate`.
- Risk measures (VaR, CVaR, EL, Std) with confidence intervals. See `portfolioRisk`.
- Risk contributions per counterparty (for EL and CVaR). See `riskContribution`.
- Risk measures and associated confidence bands. See `confidenceBands`.
- Counterparty scenario details for each counterparty. See `getScenarios`.

## See Also

`creditDefaultCopula` | `creditMigrationCopula`

## Related Examples

- “`creditDefaultCopula` Simulation Workflow” on page 4-6
- “`creditMigrationCopula` Simulation Workflow” on page 4-12
- “Modeling Correlated Defaults with Copulas” on page 4-21

## More About

- `creditDefaultCopula`
- “Corporate Credit Risk” on page 1-3
- `creditMigrationCopula`
- “Credit Rating Migration Risk” on page 1-8

## creditDefaultCopula Simulation Workflow

This example shows a common workflow for using a `creditDefaultCopula` object for a portfolio of credit instruments.

For an example of an advanced workflow using the `creditDefaultCopula` object, see “Modeling Correlated Defaults with Copulas”.

### Step 1. Create a `creditDefaultCopula` object with a two-factor model.

Load saved portfolio data. Create a `creditdefaultCopula` object with a two-factor model using the `creditdefaultCopula` constructor with the values EAD, PD, LGD, and `Weights2F`.

```
load CreditPortfolioData.mat;
cdc = creditDefaultCopula(EAD, PD, LGD,Weights2F, 'FactorCorrelation',FactorCorr2F);
disp(cdc)
disp(cdc.Portfolio(1:10:100,:))
```

creditDefaultCopula with properties:

```
Portfolio: [100x5 table]
FactorCorrelation: [2x2 double]
VaRLevel: 0.9500
PortfolioLosses: []
```

ID	EAD	PD	LGD	Weights		
1	21.627	0.0050092	0.35	0.35	0	0.65
11	29.338	0.0050092	0.55	0.35	0	0.65
21	3.8275	0.0020125	0.25	0.1125	0.3375	0.55
31	26.286	0.0020125	0.55	0.1125	0.0375	0.85
41	42.868	0.0050092	0.55	0.25	0	0.75
51	7.1259	0.00099791	0.25	0	0.25	0.75
61	10.678	0.0020125	0.35	0	0.15	0.85
71	2.395	0.00099791	0.55	0	0.15	0.85
81	26.445	0.060185	0.55	0	0.45	0.55
91	7.1637	0.11015	0.25	0.35	0	0.65

**Step 2. Set the VaRLevel to 99%.**

Set the `VaRLevel` property for the `creditDefaultCopula` object to 99% (the default is 95%).

```
cdc.VaRLevel = 0.99;
```

**Step 3. Run a simulation.**

Use the `simulate` function to run a simulation on the `creditDefaultCopula` object for 100,000 scenarios.

```
cdc = simulate(cdc,1e5)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```
    Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
    VaRLevel: 0.9900
PortfolioLosses: [1×100000 double]
```

**Step 4. Generate a report for the portfolio risk.**

Use the `portfolioRisk` function to obtain a report for risk measures and confidence intervals for EL, Std, VaR, and CVaR.

```
[portRisk,RiskConfidenceInterval] = portfolioRisk(cdc)
```

```
portRisk =
```

```
1×4 table
```

EL	Std	VaR	CVaR
24.774	23.693	101.57	120.22

```
RiskConfidenceInterval =
```

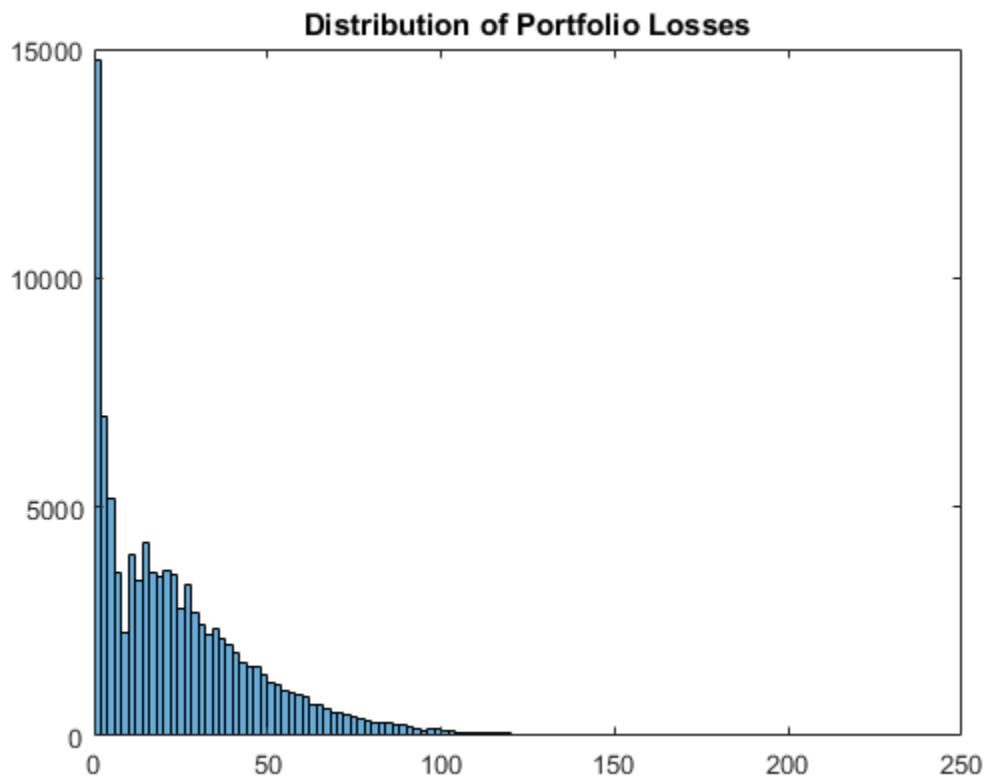
1×4 table

EL		Std		VaR		CVaR	
24.627	24.92	23.589	23.797	100.65	102.82	119.1	121.35

### Step 5. Visualize the distribution.

Use the `histogram` function to display the distribution for EL, VaR, and CVaR.

```
histogram(cdc.PortfolioLosses);  
title('Distribution of Portfolio Losses');
```



**Step 6. Generate a risk contributions report.**

Use the `riskContribution` function to display the risk contribution. The risk contributions, EL and CVaR, are *additive*. If you sum each of these two metrics over all the counterparties, you get the values reported for the entire portfolio in the `portfolioRisk` table.

```
rc = riskContribution(cdc);
disp(rc(1:10,:))
```

ID	EL	CVaR
1	0.038604	0.12868
2	0.067068	0.24527
3	1.2527	2.3103
4	0.0023253	0.0026274
5	0.11766	0.26223
6	0.12437	0.47915
7	0.82913	1.6516
8	0.00085629	0.00089197
9	0.91406	4.009
10	0.24352	2.2781

**Step 7. Simulate the risk exposure with a t copula.**

Use the `simulate` function with optional input arguments for `Copula` and `t`. Save the results to a new `creditDefaultCopula` object (`cct`).

```
cdct = simulate(cdc,1e5,'Copula','t','DegreesOfFreedom',10)
```

```
cdct =
```

```
creditDefaultCopula with properties:
```

```
Portfolio: [100x5 table]
FactorCorrelation: [2x2 double]
VaRLevel: 0.9900
PortfolioLosses: [1x100000 double]
```

**Step 8. Compare confidence bands for different copulas.**

Use the confidenceBands function to compare confidence bands for the two different copulas.

```
confidenceBands(cdc, 'RiskMeasure', 'Std', 'ConfidenceIntervalLevel', 0.90, 'NumPoints', 10)
confidenceBands(cdct, 'RiskMeasure', 'Std', 'ConfidenceIntervalLevel', 0.90, 'NumPoints', 10)
```

ans =

10×4 table

NumScenarios	Lower	Std	Upper
10000	23.467	23.739	24.019
20000	23.781	23.977	24.176
30000	23.589	23.747	23.908
40000	23.484	23.62	23.758
50000	23.486	23.608	23.732
60000	23.521	23.632	23.745
70000	23.558	23.662	23.767
80000	23.575	23.672	23.769
90000	23.602	23.693	23.785
1e+05	23.606	23.693	23.78

ans =

10×4 table

NumScenarios	Lower	Std	Upper
10000	32.578	32.957	33.345
20000	32.428	32.694	32.966
30000	32.505	32.723	32.945
40000	32.131	32.318	32.507
50000	32.018	32.185	32.353
60000	31.875	32.026	32.179
70000	31.984	32.124	32.266
80000	32.006	32.137	32.27
90000	31.939	32.063	32.188
1e+05	31.903	32.02	32.138



## See Also

confidenceBands | creditDefaultCopula | getscenarios | portfolioRisk | riskContribution | simulate

## Related Examples

- “Credit Simulation Using Copulas” on page 4-2
- “creditMigrationCopula Simulation Workflow” on page 4-12
- “Modeling Correlated Defaults with Copulas” on page 4-21

## More About

- “Risk Modeling with Risk Management Toolbox” on page 1-3
- creditDefaultCopula

# creditMigrationCopula Simulation Workflow

This example shows a common workflow for using a `creditMigrationCopula` object for a portfolio of counterparty credit ratings.

### Step 1. Create a `creditMigrationCopula` object with a 4-factor model

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a `creditMigrationCopula` object with a 4-factor model using the `creditMigrationCopula` constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...  
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =
```

```
creditMigrationCopula with properties:
```

```
Portfolio: [250x5 table]  
FactorCorrelation: [4x4 double]  
RatingLabels: [8x1 string]  
TransitionMatrix: [8x8 double]  
VaRLevel: 0.9500  
PortfolioValues: []
```

### Step 2. Set the `VaRLevel` to 99%.

Set the `VaRLevel` property for the `creditMigrationCopula` object to 99% (the default is 95%).

```
cmc.VaRLevel = 0.99;
```

### Step 3. Run a simulation.

Use the `simulate` function to simulate 100,000 scenarios.

```

cmc = simulate(cmc,1e5)

cmc =
    creditMigrationCopula with properties:
        Portfolio: [250x5 table]
        FactorCorrelation: [4x4 double]
        RatingLabels: [8x1 string]
        TransitionMatrix: [8x8 double]
        VaRLevel: 0.9900
        PortfolioValues: [1x100000 double]

```

**Step 4. Generate a report for the portfolio risk.**

Use the `portfolioRisk` function to obtain a report for risk measures and confidence intervals for EL, Std, VaR, and CVaR.

```
[portRisk,RiskConfidenceInterval] = portfolioRisk(cmc)
```

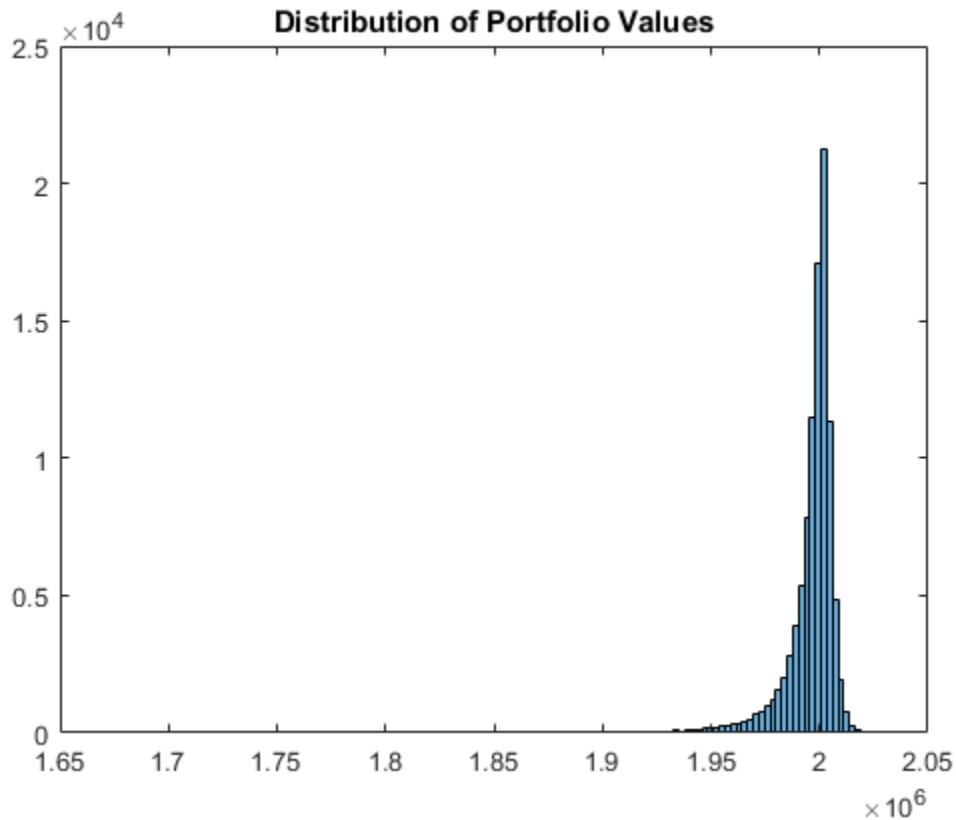
```
portRisk =
    1x4 table
        EL      Std      VaR      CVaR
    _____  _____  _____  _____
    4573.9      13039      56515      84463
```

```
RiskConfidenceInterval =
    1x4 table
        EL      Std      VaR      CVaR
    _____  _____  _____  _____
    4493.1      4654.7      12982      13096      55043      58038      82485      86441
```

**Step 5. Visualize the distribution.**

View a histogram of the portfolio values.

```
figure  
h = histogram(cmc.PortfolioValues,125);  
title('Distribution of Portfolio Values');
```



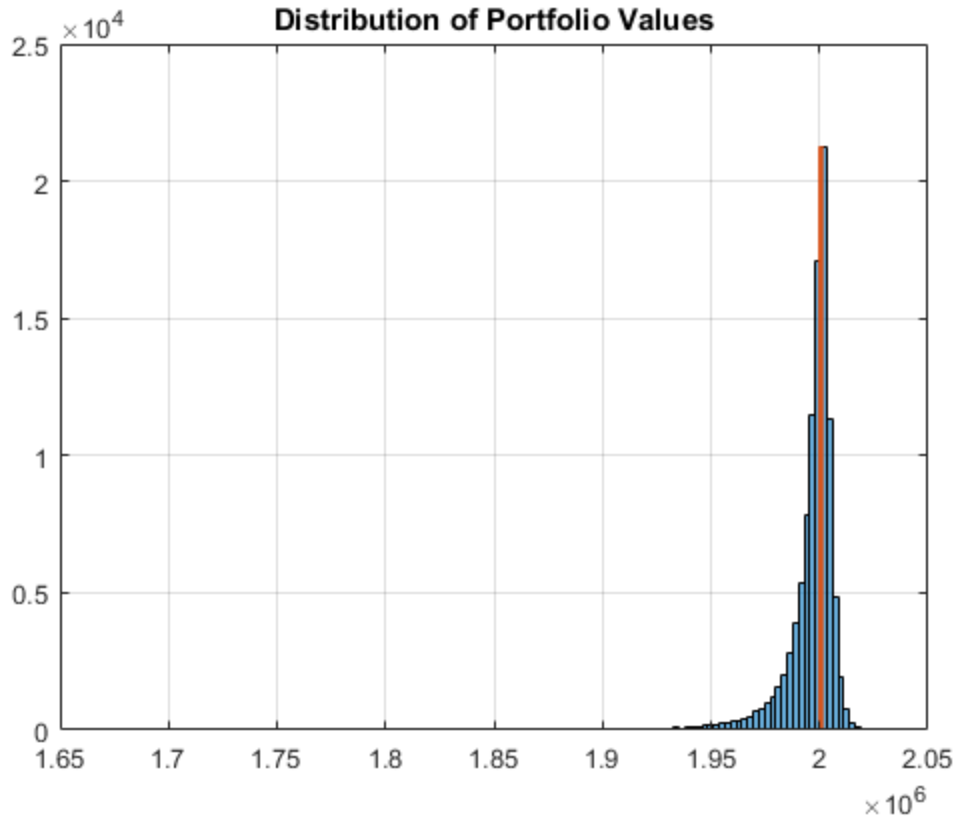
### Step 6. Overlay the value if all counterparties maintain current credit ratings.

Overlay the value that the portfolio object (cmc) takes if all counterparties maintain their current credit ratings.

```
CurrentRatingValue = portRisk.EL + mean(cmc.PortfolioValues);
```

```
hold on
```

```
plot([CurrentRatingValue CurrentRatingValue],[0 max(h.Values)], 'LineWidth', 2);
grid on
```



### Step 7. Generate a risk contributions report.

Use the `riskContribution` function to display the risk contribution. The risk contributions, EL and CVaR, are additive. If you sum each of these two metrics over all the counterparties, you get the values reported for the entire portfolio in the `portfolioRisk` table.

```
rc = riskContribution(cmc);
disp(rc(1:10,:))
```

ID	EL	CVaR
1	16.397	254.12
2	9.1179	134.31
3	5.7873	236.84
4	6.4235	338.23
5	22.739	544.69
6	10.776	704.29
7	2.9046	551.4
8	12.152	265.97
9	2.1567	26.112
10	1.7495	15.933

### Step 8. Simulate the risk exposure with a *t* copula.

To use a *t* copula with 10 degrees of freedom, use the `simulate` function with optional input arguments. Save the results to a new `creditMigrationCopula` object (`cmct`).

```
cmct = simulate(cmc,1e5,'Copula','t','DegreesOfFreedom',10)
```

```
cmct =
```

```
creditMigrationCopula with properties:
```

```
Portfolio: [250x5 table]  
FactorCorrelation: [4x4 double]  
RatingLabels: [8x1 string]  
TransitionMatrix: [8x8 double]  
VaRLevel: 0.9900  
PortfolioValues: [1x100000 double]
```

### Step 9. Generate a report for the portfolio risk for the *t* copula.

Use the `portfolioRisk` function to obtain a report for risk measures and confidence intervals for EL, Std, VaR, and CVaR.

```
[portRisk2,RiskConfidenceInterval2] = portfolioRisk(cmct)
```

```
portRisk2 =
```

1×4 table

EL	Std	VaR	CVaR
4553.6	17158	72689	1.2545e+05

RiskConfidenceInterval2 =

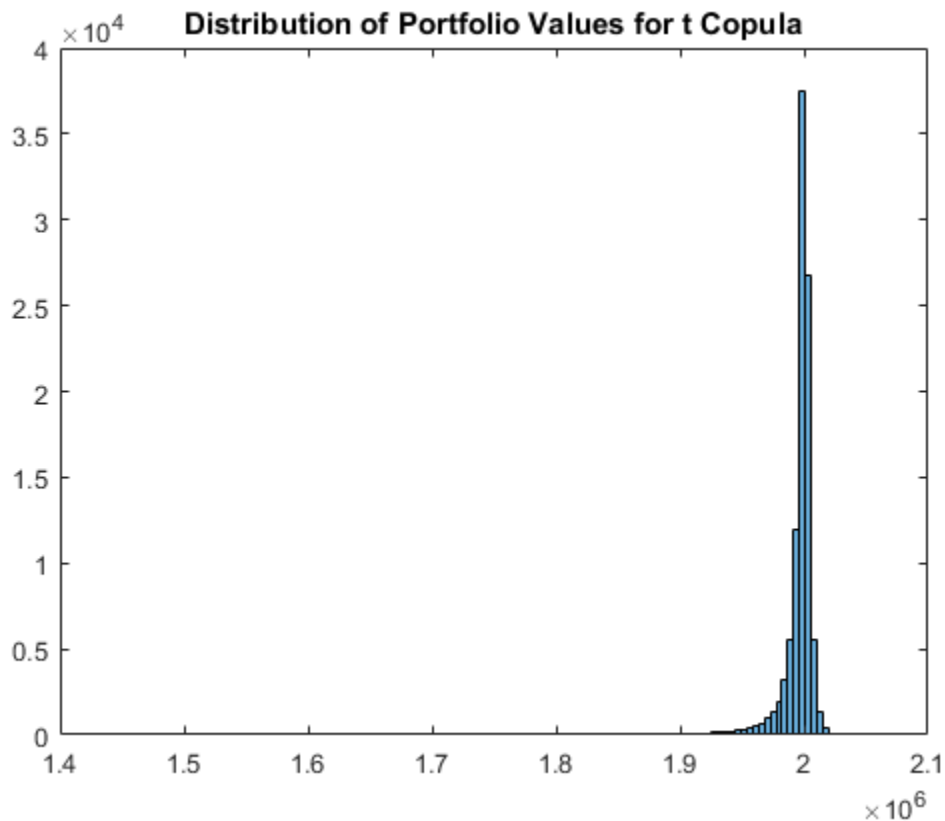
1×4 table

EL		Std		VaR		CVaR	
4447.2	4659.9	17083	17233	70834	75063	1.2144e+05	1.2947e+05

### Step 10. Visualize the distribution for the t copula.

View a histogram of the portfolio values.

```
figure
h = histogram(cmct.PortfolioValues,125);
title('Distribution of Portfolio Values for t Copula');
```



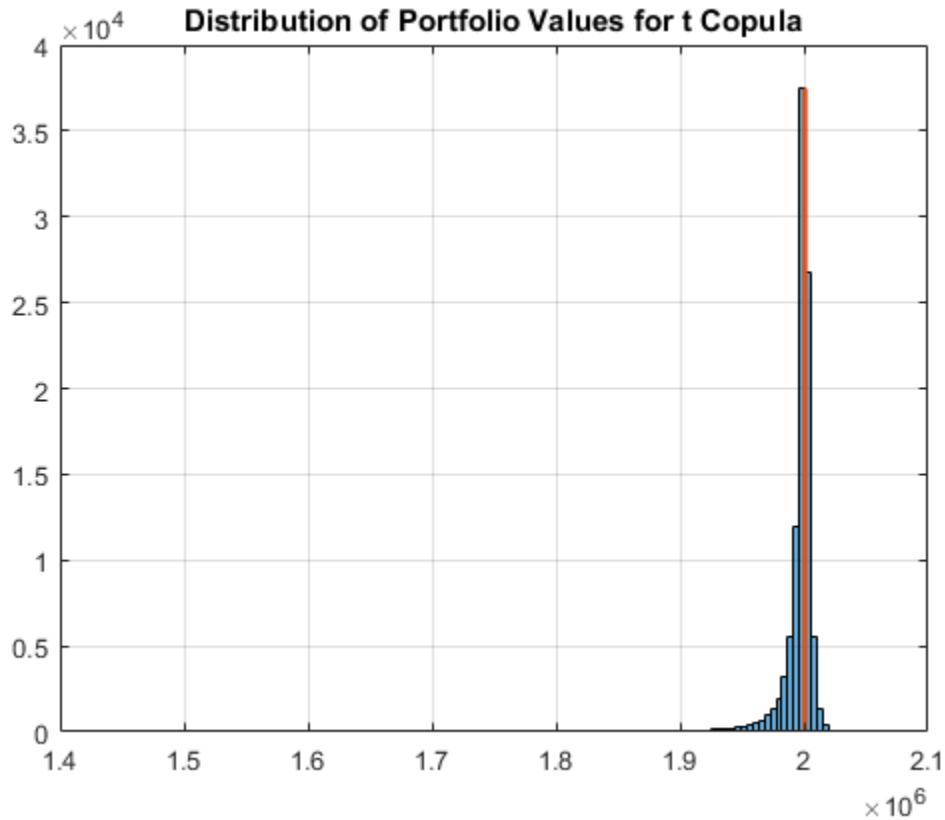
**Step 11. Overlay the value if all counterparties maintain current credit ratings for t copula.**

Overlay the value that the portfolio object (cmct) takes if all counterparties maintain their current credit ratings.

```
CurrentRatingValue2 = portRisk2.EL + mean(cmct.PortfolioValues);
```

```
hold on
plot([CurrentRatingValue2 CurrentRatingValue2],[0 max(h.Values)], 'LineWidth',2);
grid on
```





## See Also

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate`

## Related Examples

- “Credit Simulation Using Copulas” on page 4-2
- “creditDefaultCopula Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21

### **More About**

- “Credit Rating Migration Risk” on page 1-8
- `creditMigrationCopula`

## Modeling Correlated Defaults with Copulas

This example explores how to simulate correlated counterparty defaults using a multifactor copula model.

Potential losses are estimated for a portfolio of counterparties, given their exposure at default, default probability, and loss given default information. A `CreditDefaultCopula` object is used to model each obligor's credit worthiness with latent variables. Latent variables are composed of a series of weighted underlying credit factors, as well as, each obligor's idiosyncratic credit factor. The latent variables are mapped to an obligor's default or nondefault state for each scenario based on their probability of default. Portfolio risk measures, risk contributions at a counterparty level, and simulation convergence information are supported in the `CreditDefaultCopula` object.

This example also explores the sensitivity of the risk measures to the type of copula (Gaussian copula versus  $t$  copula) used for the simulation.

### Load and Examine Portfolio Data

The portfolio contains 100 counterparties and their associated credit exposures at default (EAD), probability of default (PD), and loss given default (LGD). Using a `CreditDefaultCopula` object, you can simulate defaults and losses over some fixed time period (for example, one year). The EAD, PD, and LGD inputs must be specific to a particular time horizon.

In this example, each counterparty is mapped onto two underlying credit factors with a set of weights. The `Weights2F` variable is a `NumCounterparties-by-3` matrix, where each row contains the weights for a single counterparty. The first two columns are the weights for the two credit factors and the last column is the idiosyncratic weights for each counterparty. A correlation matrix for the two underlying factors is also provided in this example (`FactorCorr2F`).

```
load CreditPortfolioData.mat
whos EAD PD LGD Weights2F FactorCorr2F
```

Name	Size	Bytes	Class	Attributes
EAD	100x1	800	double	
FactorCorr2F	2x2	32	double	
LGD	100x1	800	double	
PD	100x1	800	double	
Weights2F	100x3	2400	double	

Initialize the `creditDefaultCopula` object with the portfolio information and the factor correlation.

```
rng('default');
cc = creditDefaultCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F);
```

*% Change the VaR level to 99%.*

```
cc.VaRLevel = 0.99;
```

```
disp(cc)
```

```
creditDefaultCopula with properties:
```

```
Portfolio: [100x5 table]
FactorCorrelation: [2x2 double]
VaRLevel: 0.9900
PortfolioLosses: []
```

```
cc.Portfolio(1:5,:)
```

```
ans =
```

```
5x5 table
```

ID	EAD	PD	LGD	Weights		
1	21.627	0.0050092	0.35	0.35	0	0.65
2	3.2595	0.060185	0.35	0	0.45	0.55
3	20.391	0.11015	0.55	0.15	0	0.85
4	3.7534	0.0020125	0.35	0.25	0	0.75
5	5.7193	0.060185	0.35	0.35	0	0.65

### Simulate the Model and Plot Potential Losses

Simulate the multifactor model using the `simulate` function. By default, a Gaussian copula is used. This function internally maps realized latent variables to default states and computes the corresponding losses. After the simulation, the `creditDefaultCopula` object populates the `PortfolioLosses` and `CounterpartyLosses` properties with the simulation results.

```
cc = simulate(cc,1e5);
disp(cc)

creditDefaultCopula with properties:

    Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
    VaRLevel: 0.9900
PortfolioLosses: [1×100000 double]
```

The `portfolioRisk` function returns risk measures for the total portfolio loss distribution, and optionally, their respective confidence intervals. The value-at-risk (VaR) and conditional value-at-risk (CVaR) are reported at the level set in the `VaRLevel` property for the `creditDefaultCopula` object.

```
[pr,pr_ci] = portfolioRisk(cc);

fprintf('Portfolio risk measures:\n');
disp(pr)

fprintf('\n\nConfidence intervals for the risk measures:\n');
disp(pr_ci)
```

```
Portfolio risk measures:
      EL      Std      VaR      CVaR
-----
24.774   23.693   101.57   120.22

Confidence intervals for the risk measures:
      EL      Std      VaR      CVaR
-----
24.627   24.92   23.589   23.797   100.65   102.82   119.1   121.35
```

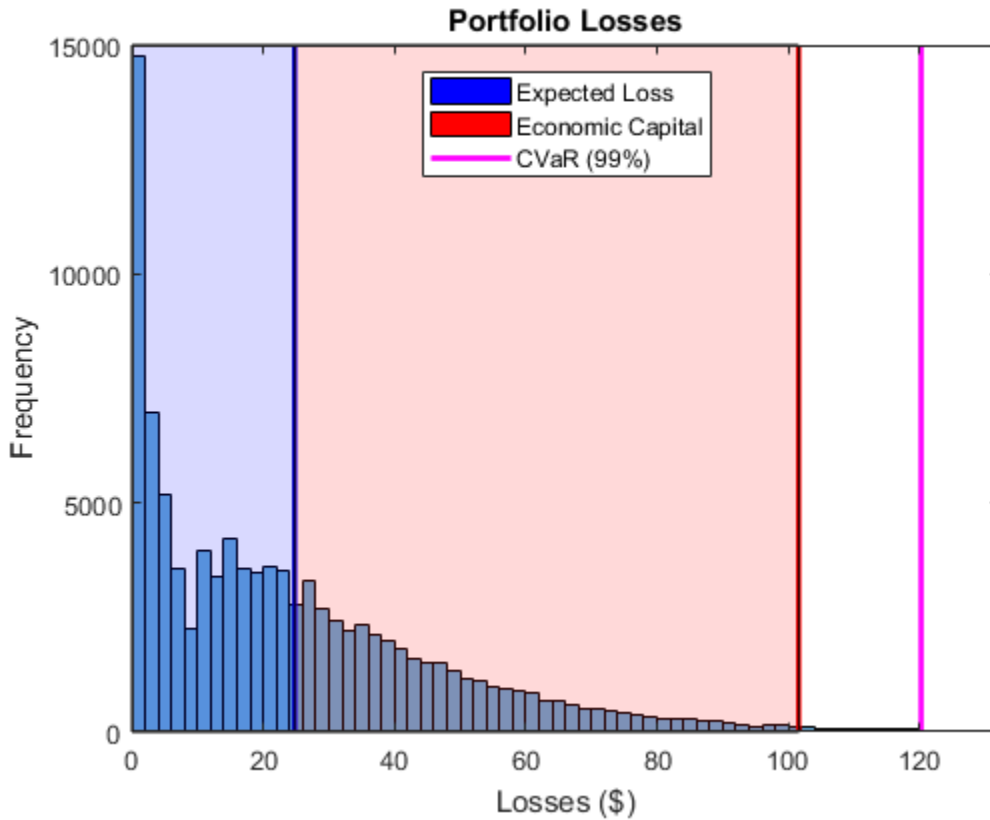
Look at the distribution of portfolio losses. The expected loss (EL), VaR, and CVaR are marked as the vertical lines. The economic capital, given by the difference between the VaR and the EL, is shown as the shaded area between the EL and the VaR.

```
histogram(cc.PortfolioLosses)
title('Portfolio Losses');
```

```
xlabel('Losses ($)')
ylabel('Frequency')
hold on

% Overlay the risk measures on the histogram.
xlim([0 1.1 * pr.CVaR])
plotline = @(x,color) plot([x x],ylim,'LineWidth',2,'Color',color);
plotline(pr.EL,'b');
plotline(pr.VaR,'r');
cvarline = plotline(pr.CVaR,'m');

% Shade the areas of expected loss and economic capital.
plotband = @(x,color) patch([x fliplr(x)],[0 0 repmat(max(ylim),1,2)],...
    color,'FaceAlpha',0.15);
elband = plotband([0 pr.EL],'blue');
ulband = plotband([pr.EL pr.VaR],'red');
legend([elband,ulband,cvarline],...
    {'Expected Loss','Economic Capital','CVaR (99%)'},...
    'Location','north');
```



### Find Concentration Risk for Counterparties

Find the concentration risk in the portfolio using the `riskContribution` function. `riskContribution` returns the contribution of each counterparty to the portfolio EL and CVaR. These additive contributions sum to the corresponding total portfolio risk measure.

```
rc = riskContribution(cc);
```

```
% Risk contributions are reported for EL and CVaR.  
rc(1:5,:)
```

```
ans =
```

```
5×3 table
```

ID	EL	CVaR
1	0.038604	0.12868
2	0.067068	0.24527
3	1.2527	2.3103
4	0.0023253	0.0026274
5	0.11766	0.26223

Find the riskiest counterparties by their CVaR contributions.

```
[rc_sorted,idx] = sortrows(rc, 'CVaR', 'descend');
rc_sorted(1:5,:)
```

```
ans =
```

```
5×3 table
```

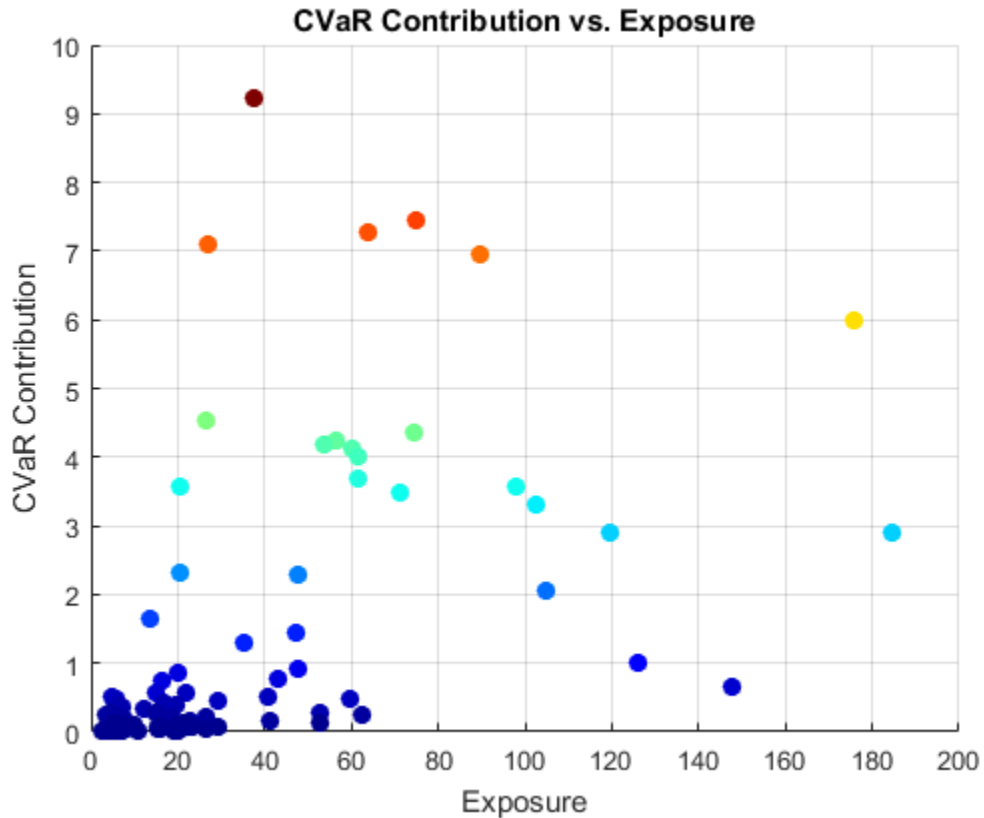
ID	EL	CVaR
89	2.261	9.2257
22	1.5672	7.4602
66	0.85227	7.2691
16	1.6236	7.1083
96	1.3331	6.9669

Plot the counterparty exposures and CVaR contributions. The counterparties with the highest CVaR contributions are plotted in red and orange.

```
figure;
pointSize = 50;
colorVector = rc_sorted.CVaR;
scatter(cc.Portfolio(idx,:).EAD, rc_sorted.CVaR,...
        pointSize,colorVector,'filled')
colormap('jet')
title('CVaR Contribution vs. Exposure')
xlabel('Exposure')
```



```
ylabel('CVaR Contribution')
grid on
```



### Investigate Simulation Convergence with Confidence Bands

Use the `confidenceBands` function to investigate the convergence of the simulation. By default, the CVaR confidence bands are reported, but confidence bands for all risk measures are supported using the optional `RiskMeasure` argument.

```
cb = confidenceBands(cc);
```

```
% The confidence bands are stored in a table.
```

```
cb(1:5,:)
```

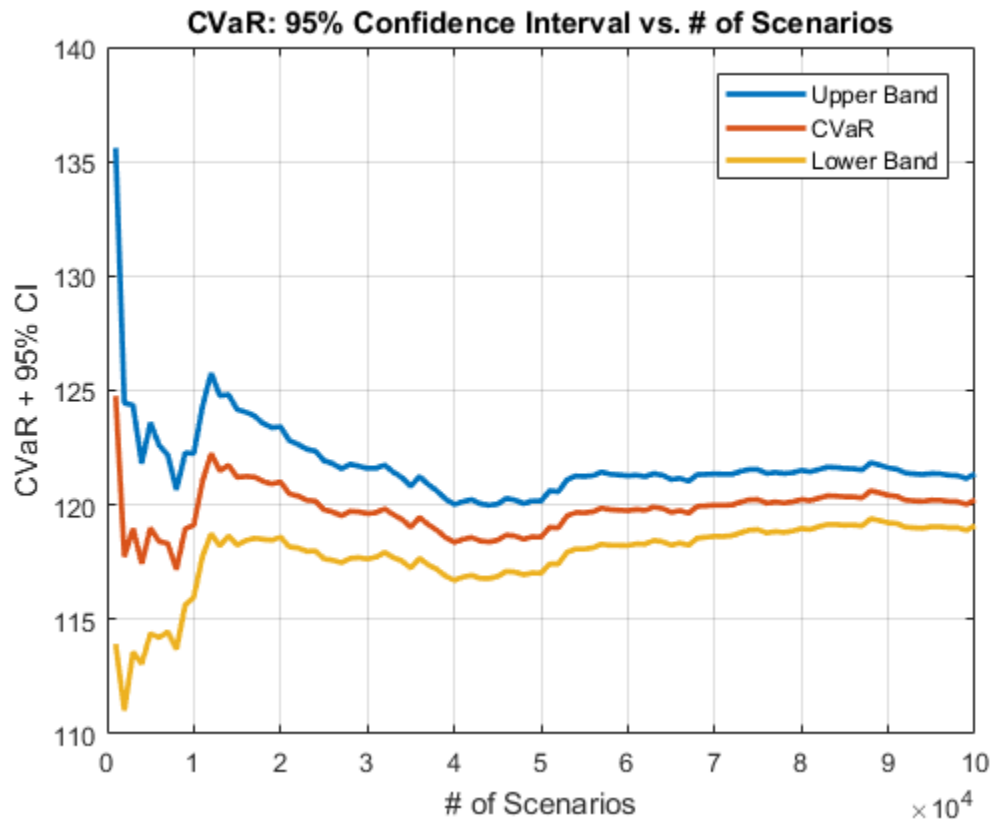
```
ans =
```

```
5×4 table
```

NumScenarios	Lower	CVaR	Upper
1000	113.92	124.76	135.59
2000	111.02	117.74	124.45
3000	113.58	118.97	124.36
4000	113.06	117.44	121.81
5000	114.38	118.99	123.6

Plot the confidence bands to see how quickly the estimates converge.

```
figure;  
plot(...  
    cb.NumScenarios,...  
    cb{:},{ 'Upper' 'CVaR' 'Lower' }},...  
    'LineWidth',2);  
  
title('CVaR: 95% Confidence Interval vs. # of Scenarios');  
xlabel('# of Scenarios');  
ylabel('CVaR + 95% CI');  
legend('Upper Band', 'CVaR', 'Lower Band');  
grid on
```



Find the necessary number of scenarios to achieve a particular width of the confidence bands.

```
width = (cb.Upper - cb.Lower) ./ cb.CVaR;
```

```
figure;
plot(cb.NumScenarios,width * 100,'LineWidth',2);
title('CVaR: 95% Confidence Interval Width vs. # of Scenarios');
xlabel('# of Scenarios');
ylabel('Width of CI as %ile of Value')
grid on
```

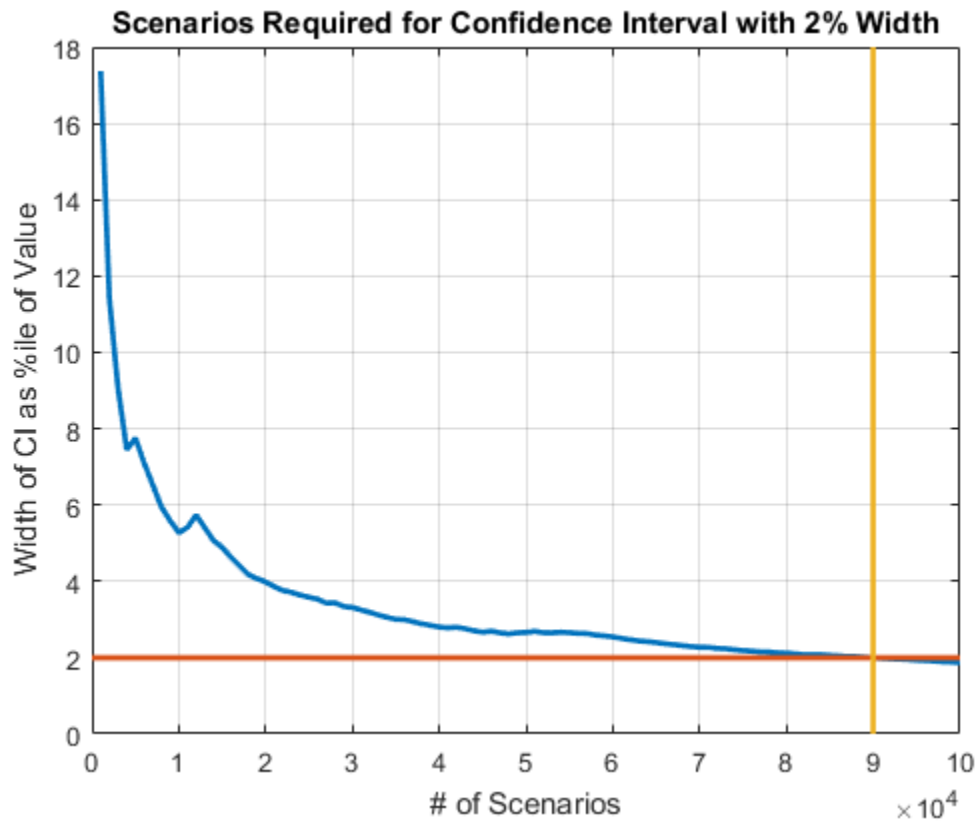
```
% Find point at which the confidence bands are within 1% (two sided) of the
% CVaR.
```

```

thresh = 0.02;

scenIdx = find(width <= thresh,1,'first');
scenValue = cb.NumScenarios(scenIdx);
widthValue = width(scenIdx);
hold on
plot(xlim,100 * [widthValue widthValue],...
     [scenValue scenValue], ylim,...
     'LineWidth',2);
title('Scenarios Required for Confidence Interval with 2% Width');

```



### Compare Tail Risk for Gaussian and $t$ Copulas

Switching to a  $t$  copula increases the default correlation between counterparties. This results in a fatter tail distribution of portfolio losses, and in higher potential losses in stressed scenarios.

Rerun the simulation using a  $t$  copula and compute the new portfolio risk measures. The default degrees of freedom (dof) for the  $t$  copula is five.

```
cc_t = simulate(cc,1e5,'Copula','t');
pr_t = portfolioRisk(cc_t);
```

See how the portfolio risk changes with the  $t$  copula.

```
fprintf('Portfolio risk with Gaussian copula:\n');
disp(pr)
```

```
fprintf('\n\nPortfolio risk with t copula (dof = 5):\n');
disp(pr_t)
```

```
Portfolio risk with Gaussian copula:
      EL      Std      VaR      CVaR
-----
24.774  23.693  101.57  120.22
```

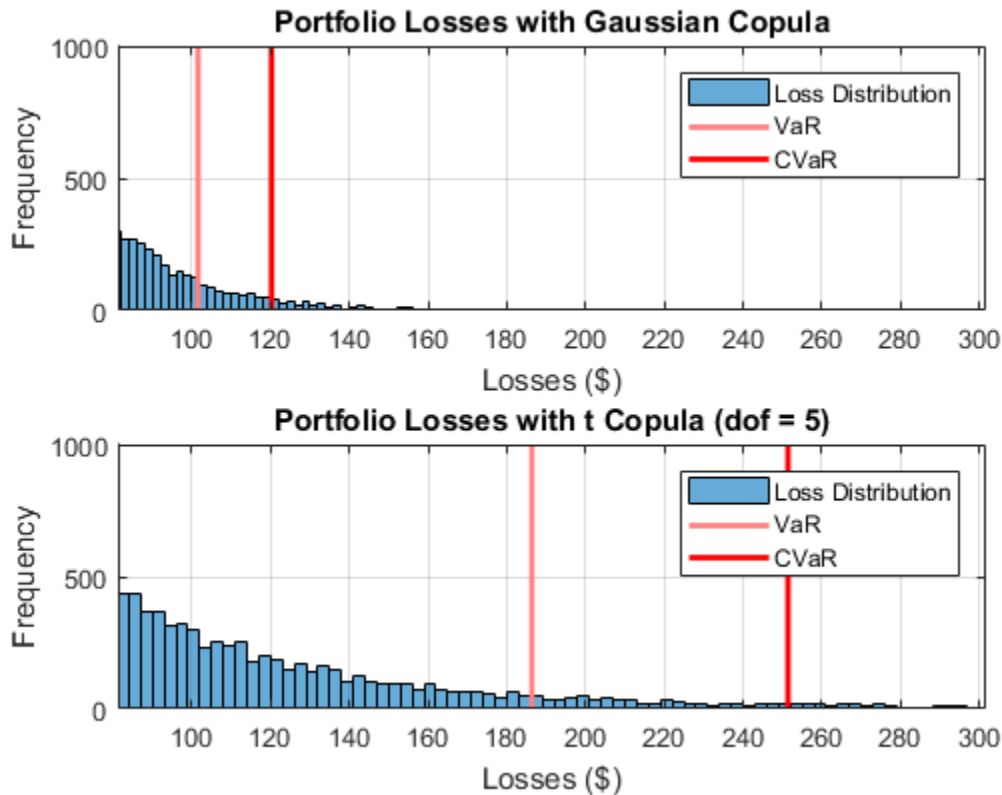
```
Portfolio risk with t copula (dof = 5):
      EL      Std      VaR      CVaR
-----
24.924  38.982  186.33  251.38
```

Compare the tail losses of each model.

```
% Plot the Gaussian copula tail.
figure;
subplot(2,1,1)
p1 = histogram(cc.PortfolioLosses);
hold on
plotline(pr.VaR,[1 0.5 0.5])
plotline(pr.CVaR,[1 0 0])
xlim([0.8 * pr.VaR 1.2 * pr_t.CVaR]);
ylim([0 1000]);
```

```
grid on
legend('Loss Distribution','VaR','CVaR')
title('Portfolio Losses with Gaussian Copula');
xlabel('Losses ($)');
ylabel('Frequency');

% Plot the t copula tail.
subplot(2,1,2)
p2 = histogram(cc_t.PortfolioLosses);
hold on
plotline(pr_t.VaR,[1 0.5 0.5])
plotline(pr_t.CVaR,[1 0 0])
xlim([0.8 * pr.VaR 1.2 * pr_t.CVaR]);
ylim([0 1000]);
grid on
legend('Loss Distribution','VaR','CVaR');
title('Portfolio Losses with t Copula (dof = 5)');
xlabel('Losses ($)');
ylabel('Frequency');
```



The tail risk measures VaR and CVaR are significantly higher using the  $t$  copula with five degrees of freedom. The default correlations are higher with  $t$  copulas, therefore there are more scenarios where multiple counterparties default. The number of degrees of freedom plays a significant role. For very high degrees of freedom, the results with the  $t$  copula are similar to the results with the Gaussian copula. Five is a very low number of degrees of freedom and, consequentially, the results show striking differences. Furthermore, these results highlight that the potential for extreme losses are very sensitive to the choice of copula and the number of degrees of freedom.

### See Also

`confidenceBands` | `creditDefaultCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate`

### **Related Examples**

- “Credit Simulation Using Copulas” on page 4-2
- “creditDefaultCopula Simulation Workflow” on page 4-6

### **More About**

- “Risk Modeling with Risk Management Toolbox” on page 1-3
- creditDefaultCopula



## Analyze the Sensitivity of Concentration to a Given Exposure

This example shows how to sweep through a range of values for an existing exposure from 0 to double the current value and plot the corresponding values. This could be used as one criterion (among others) for assessing portfolio limits.

Load credit portfolio data and use exposure at default (EAD) as the portfolio values. Compute current values of concentration indices.

```
load CreditPortfolioData.mat
P = EAD;
CurrentConcentration = concentrationIndices(P)
```

```
CurrentConcentration =
```

```
1×8 table
```

ID	CR	Deciles	Gini	HH	HK	H
"Portfolio"	0.058745	[1x11 double]	0.55751	0.023919	0.013363	0.02

Choose an index of interest. For instance, select a loan with maximum exposure.

```
[~,IndMax] = max(P);
CurrentExposure = P(IndMax);
```

Sweep through a range of multipliers for the selected exposure and get the corresponding concentration measures.

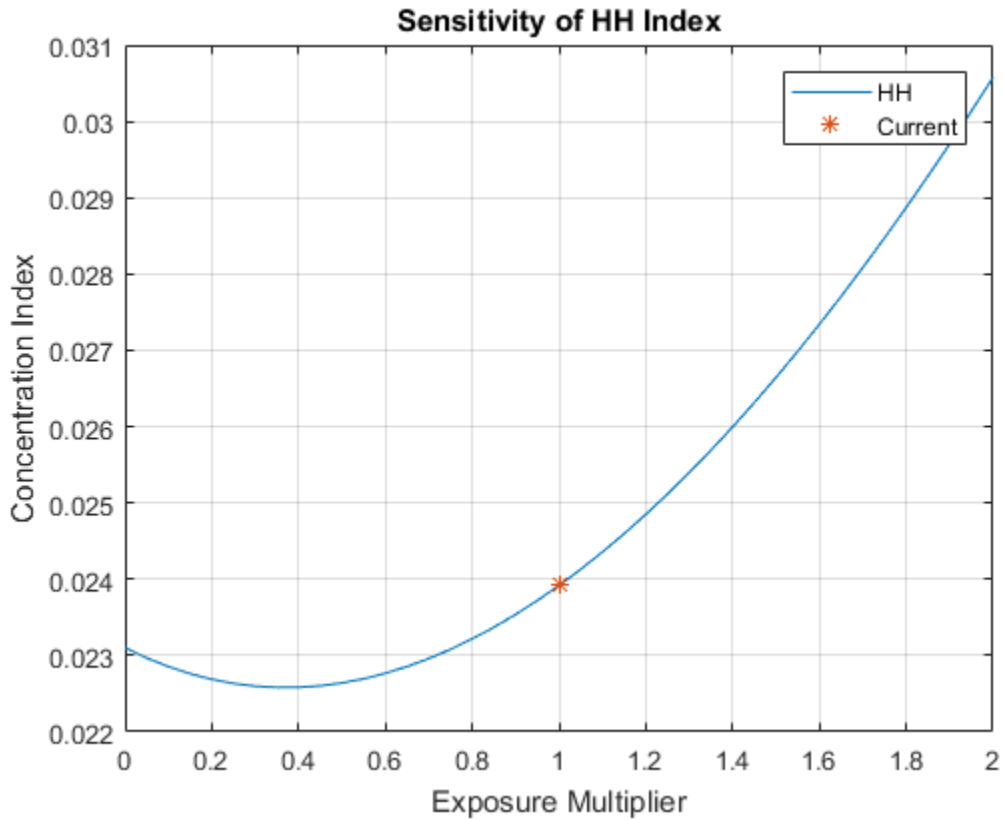
```
Multiplier = 0.0:0.05:2;
% Compute concentration with selected exposure removed from portfolio
P(IndMax) = 0;
ciSensitivity = concentrationIndices(P, 'ID', 'Multiplier 0.0');
ciSensitivity = repmat(ciSensitivity, length(Multiplier), 1);
for ii=2:length(Multiplier)
    P(IndMax) = CurrentExposure*Multiplier(ii);
    ci = concentrationIndices(P, 'ID', ['Multiplier ' num2str(Multiplier(ii))]);
    ciSensitivity(ii,:) = ci;
end
% Display first five rows
```

```
disp(ciSensitivity(1:5,:))
```

ID	CR	Deciles	Gini	HH	HK
"Multiplier 0.0"	0.059442	[1x11 double]	0.55051	0.023102	0.013314
"Multiplier 0.05"	0.059257	[1x11 double]	0.5467	0.022968	0.013185
"Multiplier 0.1"	0.059074	[1x11 double]	0.54456	0.022855	0.013156
"Multiplier 0.15"	0.058891	[1x11 double]	0.54355	0.022762	0.013143
"Multiplier 0.2"	0.058709	[1x11 double]	0.54313	0.022688	0.013139

Plot the sensitivity to changes in exposure for a particular index.

```
IndexID = 'HH';
figure;
plot(Multiplier',ciSensitivity.(IndexID))
hold on
plot(1,CurrentConcentration.(IndexID),'*')
hold off
title(['Sensitivity of ' IndexID ' Index'])
xlabel('Exposure Multiplier')
ylabel('Concentration Index')
legend(IndexID,'Current')
grid on
```



**See Also**

concentrationIndices

**Related Examples**

- “Compare Concentration Indices for Random Portfolios” on page 4-38

**More About**

- “Concentration Indices” on page 1-14

# Compare Concentration Indices for Random Portfolios

This example shows how to simulate random portfolios with different distributions and compare their concentration indices. For illustration purposes, a lognormal and Weibull distribution are used. The distribution parameters are chosen arbitrarily to get a similar range of values for both random portfolios.

Generate random portfolios with different distributions.

```
rng('default'); % for reproducibility
PLgn = lognrnd(1,1,1,300);
PWbl = wblrnd(2,0.5,1,300);
```

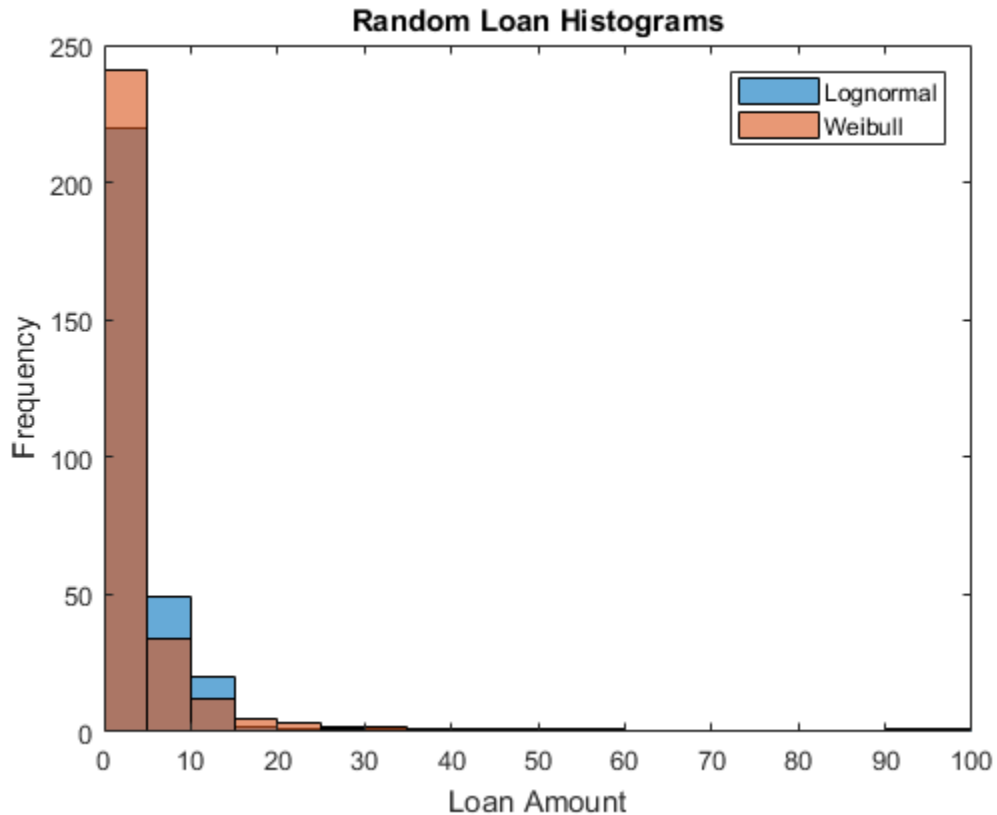
Display largest simulated loan value.

```
fprintf('\nLargest loan Lognormal: %g\n',max(PLgn));
fprintf('Largest loan Weibull: %g\n',max(PWbl));
```

```
Largest loan Lognormal: 97.3582
Largest loan Weibull: 91.5866
```

Plot the portfolio histograms.

```
figure;
histogram(PLgn,0:5:100)
hold on
histogram(PWbl,0:5:100)
hold off
title('Random Loan Histograms')
xlabel('Loan Amount')
ylabel('Frequency')
legend('Lognormal','Weibull')
```



Compute and display the concentration measures.

```
ciLgn = concentrationIndices(PLgn,'ID','Lognormal');
ciWbl = concentrationIndices(PWbl,'ID','Weibull');
disp([ciLgn;ciWbl])
```

```
ProportionLoans = 0:0.1:1;
```

```
figure;
```

```
area(ProportionLoans',[ciWbl.Deciles; ciLgn.Deciles-ciWbl.Deciles; ProportionLoans-ciLgn.Deciles]);
```

```
axis([0 1 0 1])
```

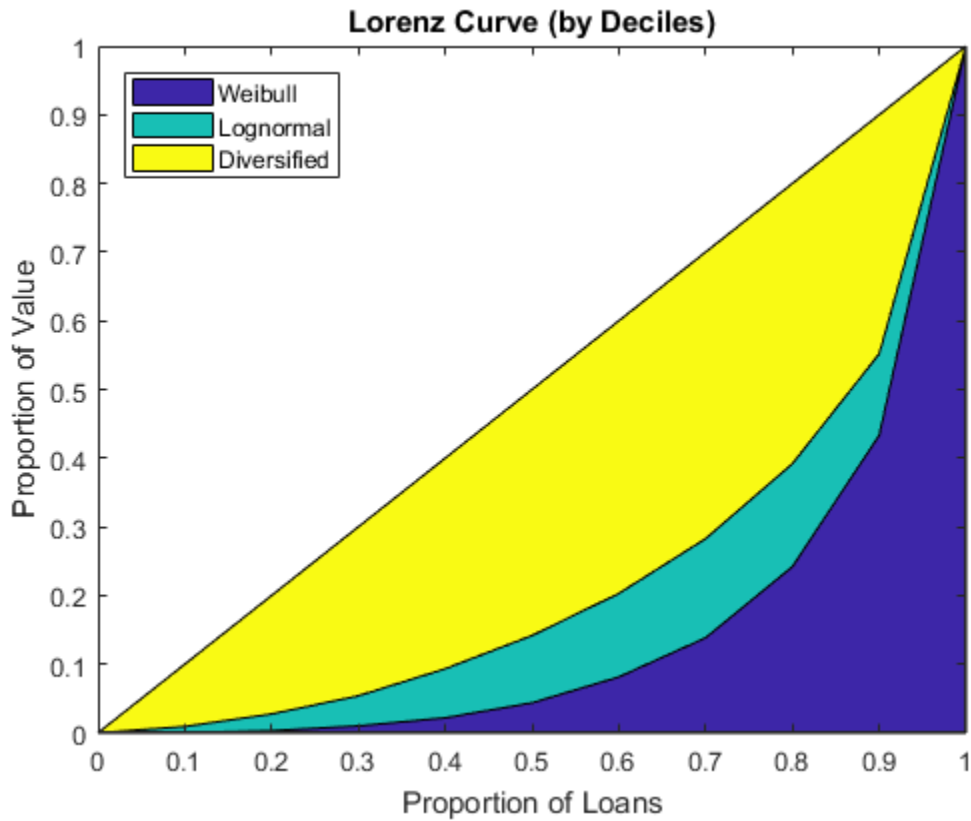
```
legend('Weibull','Lognormal','Diversified','Location','NorthWest')
```

```
title('Lorenz Curve (by Deciles)')
```

```
xlabel('Proportion of Loans')
```

```
ylabel('Proportion of Value')
```

ID	CR	Deciles	Gini	HH	HK	
"Lognormal"	0.066363	[1x11 double]	0.5686	0.013298	0.0045765	0.0
"Weibull"	0.090152	[1x11 double]	0.72876	0.020197	0.0062594	0



**See Also**

concentrationIndices

**Related Examples**

- “Analyze the Sensitivity of Concentration to a Given Exposure” on page 4-35

## **More About**

- “Concentration Indices” on page 1-14

## Comparison of the Merton Model Single-Point Approach to the Time-Series Approach

This example shows how to compare the Merton model approach, where equity volatility is provided, to the time series approach.

Load the data from MertonData.mat.

```
load MertonData.mat
Dates      = MertonDataTS.Dates;
Equity     = MertonDataTS.Equity;
Liability  = MertonDataTS.Liability;
Rate      = MertonDataTS.Rate;
```

For a given data point in the returns, the corresponding equity volatility is computed from the last preceding 30 days.

```
Returns      = tick2ret(Equity);
DateReturns  = Dates(2:end);
SampleSize   = length>Returns);

EstimationWindowSize = 30;
TestWindowStart      = EstimationWindowSize+1;
TestWindow           = (TestWindowStart : SampleSize)';

EquityVol = zeros(length(TestWindow),1);

for i = 1 : length(TestWindow)
    t = TestWindow(i);
    EstimationWindow = t-EstimationWindowSize:t-1;
    EquityVol(i) = sqrt(250)*std>Returns(EstimationWindow));
end
```

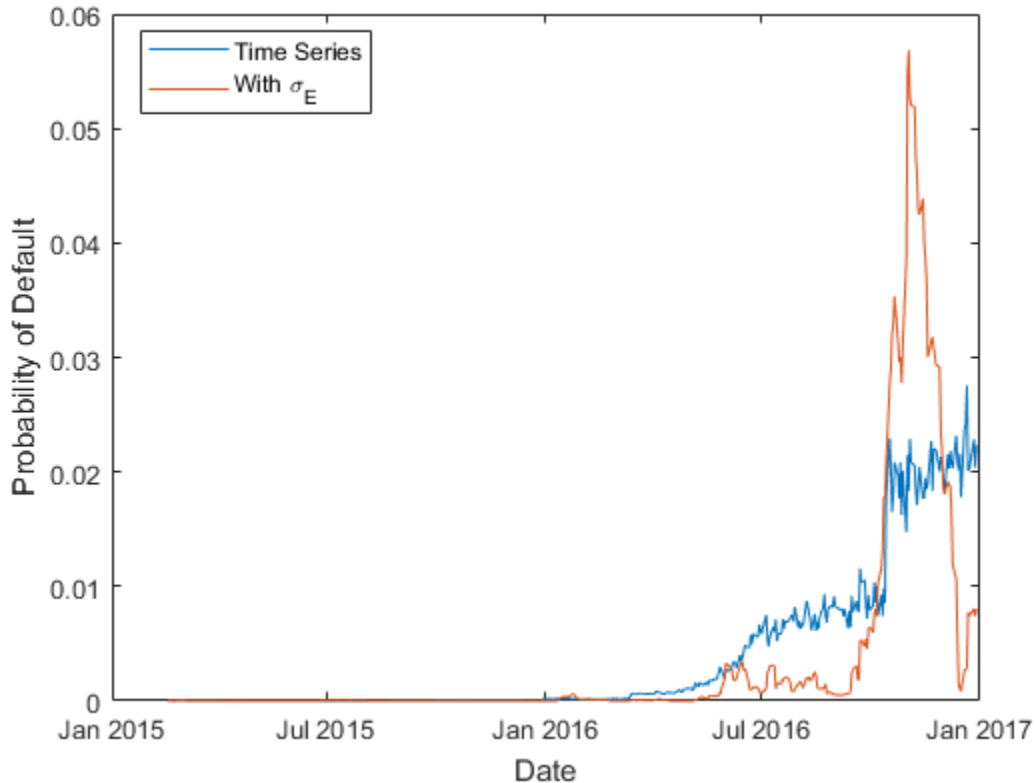
Compare the probabilities of default and the estimated asset and asset volatility values using the test window only.

```
[PDTS,DDTS,ATS,SaTS] = mertonByTimeSeries(Equity(TestWindow),Liability(TestWindow),Rate
[PDh,DDh,Ah,Sah] = mertonmodel(Equity(TestWindow),EquityVol,Liability(TestWindow),Rate

figure
plot(Dates(TestWindow),PDTS,Dates(TestWindow),PDh)
xlabel('Date')
ylabel('Probability of Default')
```



```
legend({'Time Series', 'With \sigma_E'}, 'Location', 'best')
```

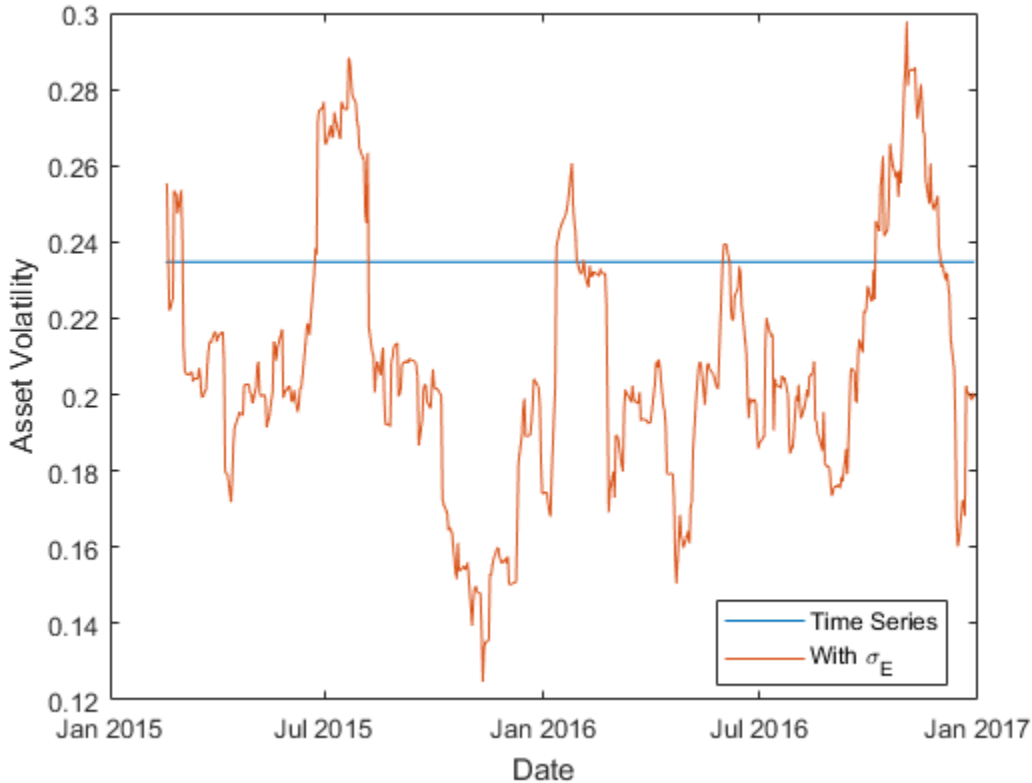


The probabilities of default are essentially zero up to early 2016. At that point, both models start predicting positive default probabilities, but we observe some differences between the two models.

Both models calibrate asset values and asset volatilities. The asset values for both approaches match. However, the time-series method, by design, computes a single asset volatility for the entire time window, and the single-point version of the Merton model computes one volatility for each time period, as shown in the following figure.

```
figure
plot(Dates(TestWindow), SaTS*ones(size(TestWindow)), Dates(TestWindow), Sah)
xlabel('Date')
```

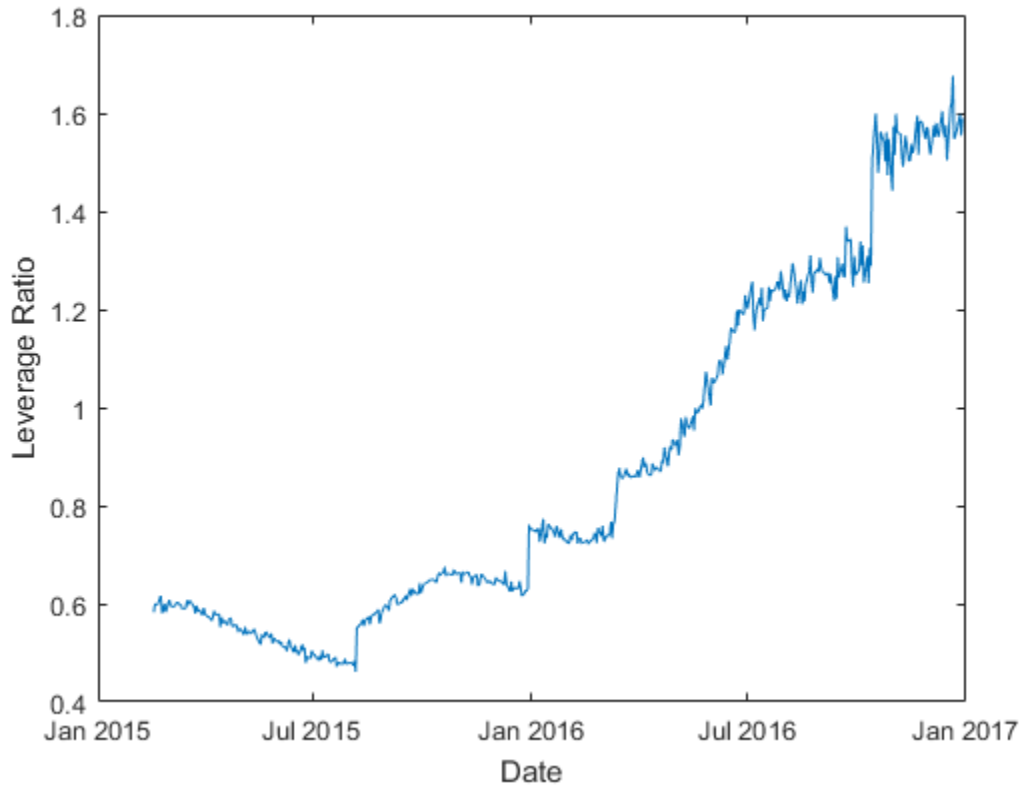
```
ylabel('Asset Volatility')
legend({'Time Series', 'With \sigma_E'}, 'Location', 'best')
```



Towards the end of the time window, the single-point probability of default is above the time-series probability of default when the single-point asset volatility is also above the time-series probability (and vice versa). However, before 2016 the volatility has no effect on the default probability. This means other factors must influence the sensitivity of the default probability to the asset volatility and the overall default probability level.

The firm's *leverage ratio*, defined as the ratio of liabilities to equity, is a key factor in understanding the default probability values in this example. Earlier in the time window, the leverage ratio is low. However, in the second half of the time window, the leverage ratio grows significantly as shown in the following figure.

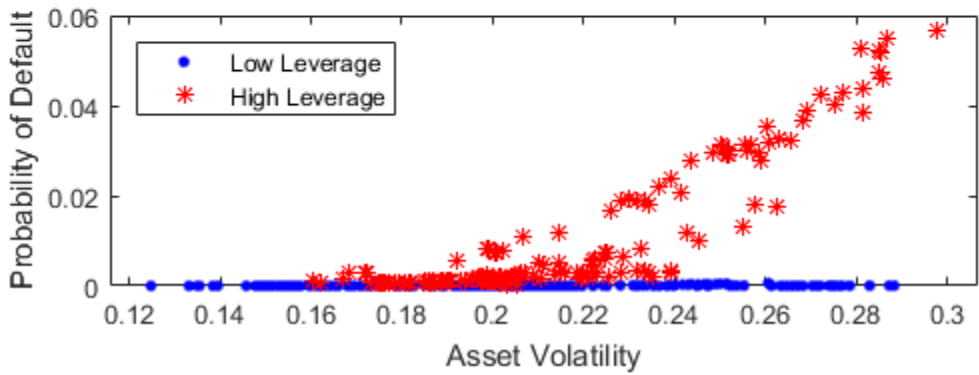
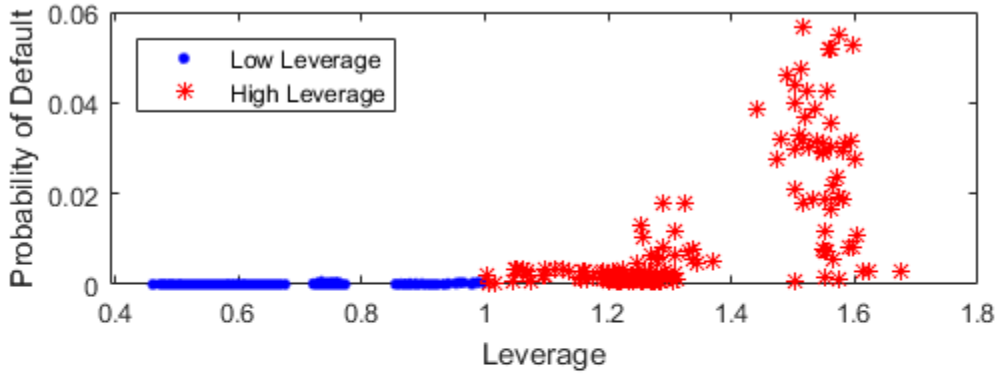
```
Leverage = Liability(TestWindow) ./ Equity(TestWindow);  
  
figure  
plot(Dates(TestWindow), Leverage)  
xlabel('Date')  
ylabel('Leverage Ratio')
```



The following plot shows the default probability against the asset volatility for low and high leverage ratios. The leverage ratio is used to divide the points into two groups, depending on whether the leverage ratio is greater or smaller than a cut off value. In this example, a cut off value of 1 works well.

For low leverage, the probability of default is essentially zero, independently of the asset volatilities. For high leverage situations, such as the end of the time window, the probability of default is highly correlated with the asset volatility.

```
figure
subplot(2,1,1)
gscatter(Leverage,PDh,Leverage>1,'br','.*')
xlabel('Leverage')
ylabel('Probability of Default')
legend('Low Leverage','High Leverage','Location','northwest')
subplot(2,1,2)
gscatter(Sah,PDh,Leverage>1,'br','.*')
xlabel('Asset Volatility')
ylabel('Probability of Default')
legend('Low Leverage','High Leverage','Location','northwest')
```



## See Also

[mertonByTimeSeries](#) | [mertonmodel](#)

## More About

- “Default Probability by Using the Merton Model for Structural Credit Risk” on page 1-11



# Functions — Alphabetical List

---

## Binning Explorer

Bin data and export into a `creditscorecard` object

### Description

The **Binning Explorer** app enables you to manage binning categories for a `creditscorecard` object. After creating a table of data in your MATLAB workspace, or after using `creditscorecard` to create a `creditscorecard` object, use the **Binning Explorer** to:

- Select an automatic binning algorithm.
- Shift bin boundaries.
- Split bins.
- Merge bins.
- Save and export a `creditscorecard` object.

### Open the Binning Explorer App

- MATLAB toolstrip: On the **Apps** tab, under **Computational Finance**, click the app icon.
- MATLAB command prompt: Enter `binningExplorer`.

### Examples

- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-26
- “Case Study for a Credit Scorecard Analysis” (Financial Toolbox)
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-44



## See Also

### See Also

#### Functions

creditscorecard

#### Topics

“Common Binning Explorer Tasks” on page 3-4

“Binning Explorer Case Study Example” on page 3-26

“Case Study for a Credit Scorecard Analysis” (Financial Toolbox)

“Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-44

“Overview of Binning Explorer” on page 3-2

“Credit Scorecard Modeling Workflow” (Financial Toolbox)

### External Websites

Credit Scorecard Modeling Using the Binning Explorer App (6 min 30 sec)

**Introduced in R2016b**

## concentrationIndices

Compute ad-hoc concentration indices for a portfolio

### Syntax

```
ci = concentrationIndices(PortfolioData)
[ci,Lorenz] = concentrationIndices( ____,Name,Value)
```

### Description

`ci = concentrationIndices(PortfolioData)` computes multiple ad-hoc concentration indices for a given portfolio. The `concentrationIndices` function supports the following indices:

- CR — Concentration ratio
- Deciles — Deciles of the portfolio weights distribution
- Gini — Gini coefficient
- HH — Herfindahl-Hirschman index
- HK — Hannah-Kay index
- HT — Hall-Tideman index
- TE — Theil entropy index

`[ci,Lorenz] = concentrationIndices( ____,Name,Value)` adds optional name-value pair arguments.

### Examples

#### Compute Concentration Indices for a Credit Portfolio

Compute the concentration indices for a credit portfolio using a portfolio that is described by its exposures. The exposures at default are stored in the `EAD` array.

Load the `CreditPortfolioData.mat` file that contains EAD used for the `PortfolioData` input argument.

```
load CreditPortfolioData.mat
ci = concentrationIndices(EAD)
```

```
ci =
```

```
1×8 table
```

ID	CR	Deciles	Gini	HH	HK	H
"Portfolio"	0.058745	[1x11 double]	0.55751	0.023919	0.013363	0.02

### Compute Multiple Concentration Ratios

Use the `CRIndex` optional input to obtain the concentration ratios for the tenth and twentieth largest exposures. In the output, the `CR` column becomes a vector, with one value for each requested index.

Load the `CreditPortfolioData.mat` file that contains the EAD used for the `PortfolioData` input argument.

```
load CreditPortfolioData.mat
ci = concentrationIndices(EAD, 'CRIndex', [10 20])
```

```
ci =
```

```
1×8 table
```

ID	CR	Deciles	Gini	HH	HK	H
"Portfolio"	0.38942    0.58836	[1x11 double]	0.55751	0.023919	0.013363	0.02

### Modify the Alpha Parameter of the Hannah-Kay Index

Use the `HKAlpha` optional input to set the alpha parameter for the Hannah-Kay (HK) index. Use a vector of alpha values to compute the HK index for multiple parameter

values. In the output, the HK column becomes a vector, with one value for each requested alpha value.

Load the `CreditPortfolioData.mat` file that contains EAD used for the `PortfolioData` input argument.

```
load CreditPortfolioData.mat
ci = concentrationIndices(EAD, 'HKAlpha', [0.5 3])
```

```
ci =
```

```
1×8 table
```

ID	CR	Deciles	Gini	HH	HK	HT	TE
"Portfolio"	0.058745	[1x11 double]	0.55751	0.023919	0.013363	0.023919	0.013363

### Create an ID to Compare Concentration Index Results

Compare the concentration measures using an ID optional argument for a fully diversified portfolio and a fully concentrated portfolio.

```
ciD = concentrationIndices([1 1 1 1 1], 'ID', 'Fully diversified');
ciC = concentrationIndices([0 0 0 0 5], 'ID', 'Fully concentrated');
disp([ciD;ciC])
```

ID	CR	Deciles	Gini	HH	HK	HT	TE
"Fully diversified"	0.2	[1x11 double]	0	0.2	0.2	0.2	-2.2204
"Fully concentrated"	1	[1x11 double]	0.8	1	1	1	-

### Apply Scaling to Concentration Indices

Use the `ScaleIndices` optional input to scale the index values of Gini, HH, HK, HT, and TE. The range of `ScaleIndices` is from 0 through 1, independent of the number of loans.

```
ciDU = concentrationIndices([1 1 1 1 1], 'ID', 'Diversified, unscaled');
```

```

ciDS = concentrationIndices([1 1 1 1 1], 'ID', 'Diversified, scaled', 'ScaleIndices', true);
ciCU = concentrationIndices([0 0 0 0 5], 'ID', 'Concentrated, unscaled');
ciCS = concentrationIndices([0 0 0 0 5], 'ID', 'Concentrated, scaled', 'ScaleIndices', true);
disp([ciDU;ciDS;ciCU;ciCS])

```

ID	CR	Deciles	Gini	HH	HK
"Diversified, unscaled"	0.2	[1x11 double]	0	0.2	0
"Diversified, scaled"	0.2	[1x11 double]	0	3.4694e-17	-3.4694e-17
"Concentrated, unscaled"	1	[1x11 double]	0.8	1	1
"Concentrated, scaled"	1	[1x11 double]	1	1	1

### Plot an Approximate Lorenz Curve Using Deciles Information

Load the `CreditPortfolioData.mat` file that contains `EAD` used for the `PortfolioData` input argument.

```

load CreditPortfolioData.mat
P = EAD;
ci = concentrationIndices(P);

```

Visualize an approximate Lorenz curve using the deciles information and also the concentration at the decile level.

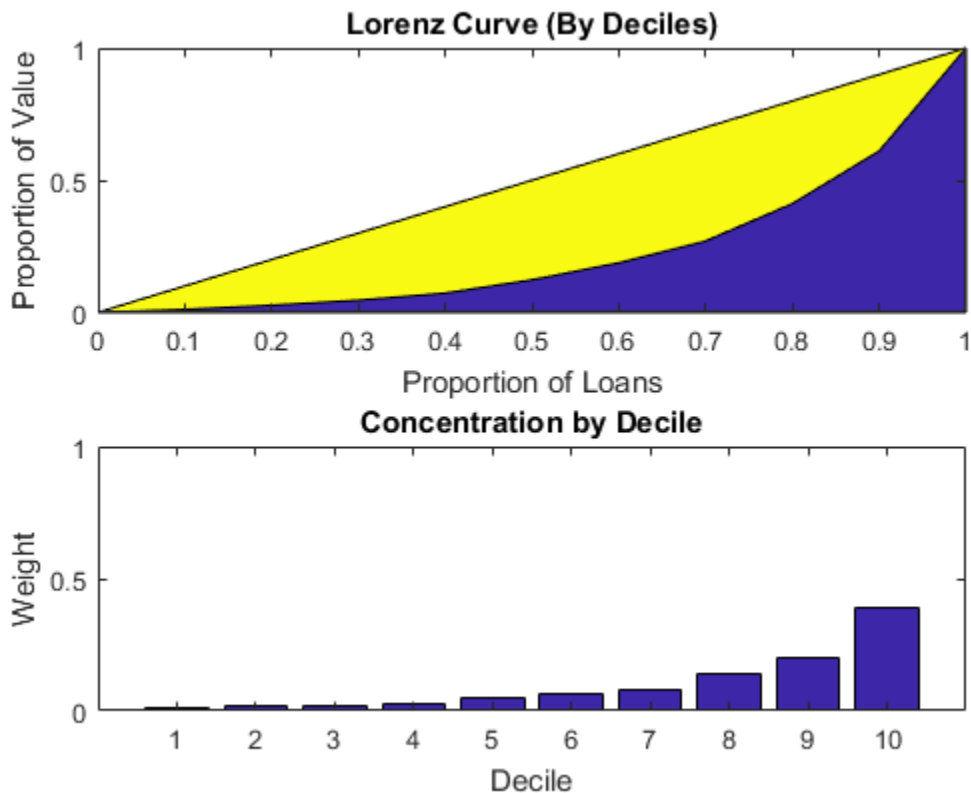
```

Proportion = 0:0.1:1;

figure;
subplot(2,1,1)
area(Proportion', [ci.Deciles' Proportion' -ci.Deciles'])
axis([0 1 0 1])
title('Lorenz Curve (By Deciles)')
xlabel('Proportion of Loans')
ylabel('Proportion of Value')

subplot(2,1,2)
bar(diff(ci.Deciles))
axis([0 11 0 1])
title('Concentration by Decile')
xlabel('Decile')
ylabel('Weight')

```



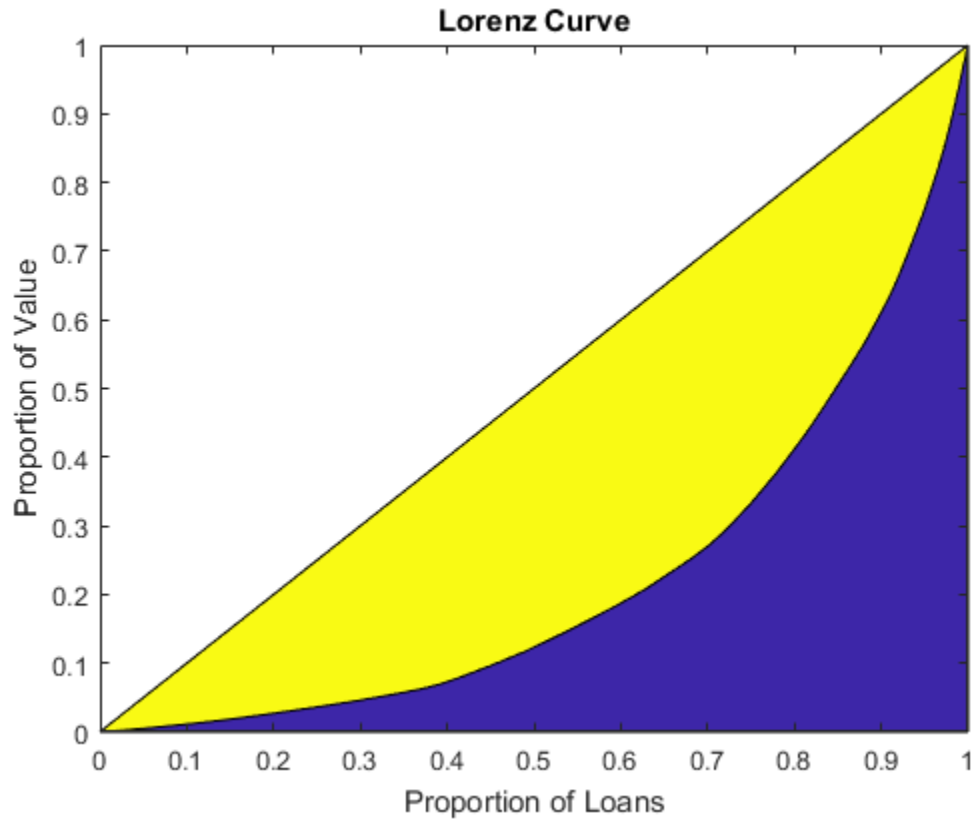
### Plot an Exact Lorenz Curve Using the Optional Lorenz Output

Load the `CreditPortfolioData.mat` file that contains `EAD` used for the `PortfolioData` input argument. The optional output `Lorenz` contains the data for the exact Lorenz curve.

```
load CreditPortfolioData.mat
P = EAD;
[~,Lorenz] = concentrationIndices(P);
```

```
figure;
area(Lorenz.ProportionLoans,[Lorenz.ProportionValue Lorenz.ProportionLoans-Lorenz.ProportionValue]);
axis([0 1 0 1])
title('Lorenz Curve')
```

```
xlabel('Proportion of Loans')  
ylabel('Proportion of Value')
```



- “Analyze the Sensitivity of Concentration to a Given Exposure” on page 4-35
- “Compare Concentration Indices for Random Portfolios” on page 4-38

## Input Arguments

**PortfolioData** — Nonnegative portfolio positions in  $N$  assets  
numeric array

Nonnegative portfolio positions in  $N$  assets, specified as an  $N$ -by-1 (or 1-by- $N$ ) numeric array.

Data Types: `double`

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

Example: `[ci,Lorenz] = concentrationIndices(PortfolioData,'CRIndex',100)`

#### 'CRIndex' — Index of interest for concentration ratio

1 (default) | nonnegative integer

Index of interest for the concentration ratio, specified as the comma-separated pair consisting of 'CRIndex' and an integer value between 1 and  $N$ , where  $N$  is the number of assets in the portfolio. The default value for CRIndex is 1 (the default CR is the largest portfolio weight). If CRIndex is a vector, the concentration ratio is computed for the index value in the given order.

Data Types: `double`

#### 'HKA1pha' — Alpha parameter for Hannah-Kay index

0.5 (default) | nonnegative numeric

Alpha parameter for Hannah-Kay index, specified as the comma-separated pair consisting of 'HKA1pha', and a positive number that cannot be equal to 1. If HKA1pha is a vector, the Hannah-Kay index is computed for each alpha value in the given order.

Data Types: `double`

#### 'ID' — User-defined ID for portfolio

"Portfolio" (default) | character vector | string object

User-defined ID for the portfolio, specified as the comma-separated pair consisting of 'ID' and a scalar string object or character vector.

Data Types: `char` | `string`



**'ScaleIndices' — Flag to indicate whether to scale concentration indices**

false (no scaling) (default) | logical

Flag to indicate whether to scale concentration indices, specified as the comma-separated pair consisting of `'ScaleIndices'` and a logical scalar. When the `ScaleIndices` is set to true, the value of the Gini, HH, HK, HT, and TE indices are scaled so that all these indices have a minimum value of 0 (full diversification) and a maximum value of 1 (full concentration).

---

**Note:** Scaling is applied only for portfolios with at least two assets. Otherwise, the scaling capability is undefined.

---

Data Types: logical

## Output Arguments

**ci — Concentration indices information for given portfolio**

table

Concentration indices information for the given portfolio, returned as a table with the following columns:

- **ID** — Portfolio ID string. Use the ID name-value pair argument to set it.
- **CR** — Concentration ratio. By default, the concentration ratio for the first index (largest portfolio weight) is reported. Use the `CRIndex` name-value pair argument to choose a different index. If `CRIndex` is a vector of length  $m$ , then `CR` is a row vector of size 1-by- $m$ . For more information, see “Definitions” on page 5-12.
- **Deciles** — Deciles of the portfolio weights distribution is a 1-by-11 row vector containing the values 0, the nine decile cut points, and 1. For more information, see “Definitions” on page 5-12.
- **Gini** — Gini coefficient. For more information, see “Definitions” on page 5-12.
- **HH** — Herfindahl-Hirschman index. For more information, see “Definitions” on page 5-12.
- **HK** — Hannah-Kay index (reciprocal). By default, the `'alpha'` parameter is set to 0.5. Use the `HKAlpha` name-value pair argument to choose a different value. If

HKAlpha is a vector of length  $m$ , then HK is a row vector of size 1-by- $m$ . For more information, see “Definitions” on page 5-12.

- HT — Hall-Tideman index. For more information, see “Definitions” on page 5-12.
- TE — Theil entropy index. For more information, see “Definitions” on page 5-12.

### **Lorenz — Lorenz curve data**

table

Lorenz curve data, returned as a table with the following columns:

- **ProportionLoans** — (N+1)-by-1 numeric array containing the values 0, 1/N, 2/N, ... N/N = 1. This is the data for the horizontal axis of the Lorenz curve.
- **ProportionValue** — (N+1)-by-1 numeric array containing the proportion of portfolio value accumulated up to the corresponding proportion of loans in the **ProportionLoans** column. This is the data for the vertical axis of the Lorenz curve.

## **Definitions**

### **Portfolio Notation**

All the concentration indices for **concentrationIndices** assume a credit portfolio with an exposure to counterparties.

Let  $P$  be a given credit portfolio with exposure to  $N$  counterparties. Let  $x_1, \dots, x_N$  represent the exposures to each counterparty, with  $x_i \geq 0$  for all  $i = 1, \dots, N$ . And, let  $x$  be the total portfolio exposure

$$x = \sum_{i=1}^N x_i$$

Assume that  $x > 0$ , that is, at least one exposure is nonzero. The portfolio weights are given by  $w_1, \dots, w_N$  with

$$w_i = \frac{x_i}{x}$$

The weights are sorted in nondecreasing order. The following standard notation uses brackets around the indices to denote ordered values.

$$w_{[1]} \leq w_{[2]} \leq \dots \leq w_{[N]}$$

## Concentration Ratio

The *concentration ratio* (CR) answers the question “what proportion of the total exposure is accumulated in the largest  $k$  loans?”

The formula for the concentration ratio (CR) is:

$$CR_k = \sum_{i=1}^k w_{[N-i+1]}$$

For example, if  $k=1$ ,  $CR_1$  is a sum of the one term  $w_{[N-1+1]} = w_{[N]}$ , that is, it is the largest weight. For any  $k$ , the CR index takes values from 0 through 1.

## Lorenz Curve

The *Lorenz curve* is a visualization of the cumulative proportion of portfolio value (or cumulative portfolio weights) against the cumulative proportion of loans.

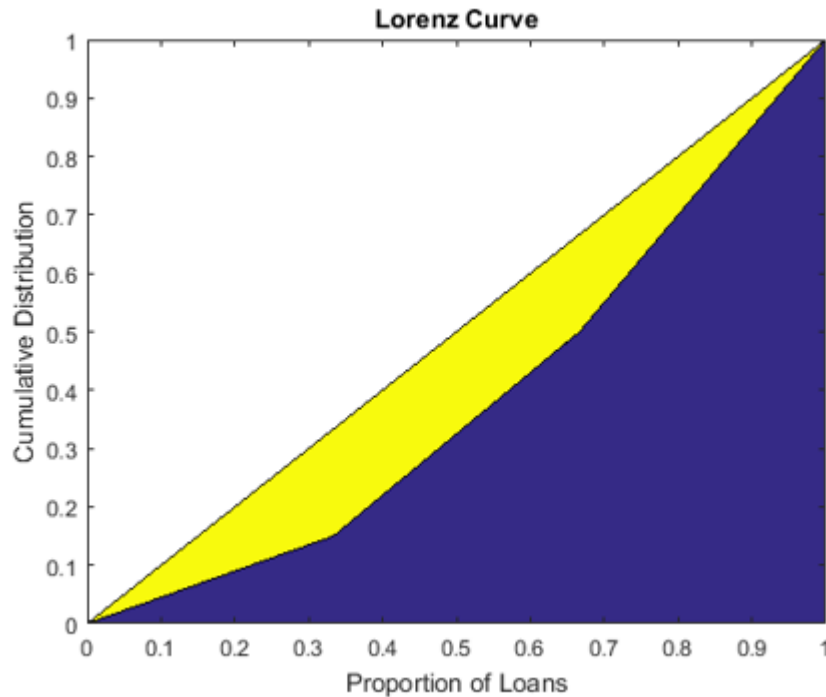
The cumulative proportion of loans ( $p$ ) is defined by:

$$p_0 = 0, p_1 = \frac{1}{N}, p_2 = \frac{2}{N}, \dots, p_N = \frac{N}{N} = 1$$

The cumulative proportion of portfolio value  $L$  is defined as:

$$L_0 = 0, L_k = \sum_{i=1}^k w_{[i]}$$

The Lorenz curve is a plot of  $L$  versus  $p$ , or the cumulative proportion of portfolio value versus cumulative proportion of the number of loans (sorted from smallest to largest).



The diagonal line is indicated in the same plot because it represents the curve for the portfolio with the least possible concentration (all loans with the same weight). The area between the diagonal and the Lorenz curve is a visual representation of the Gini coefficient, which is another concentration measure.

## Deciles

*Deciles* are commonly used in the context of income inequality.

If you sort individuals by their income level, what proportion of the total income is earned by the lowest 10% and the lowest 20% of the population? In a credit portfolio, loans can be sorted by exposure. The first decile corresponds to the proportion of the portfolio value that is accumulated by the smallest 10% loans, and so on. Deciles are proportions, therefore they always take values from 0 through 1.

Defining the cumulative proportion of loans ( $p$ ) and the cumulative proportion of values  $L$  as in “Lorenz Curve” on page 5-13, the deciles are a subset of the proportion of value array. Given indices  $d1, d2, \dots, d9$  such that the proportion of loans matches exactly these values:

$$p_{d1} = 0.1, p_{d2} = 0.2, \dots, p_{d9} = 0.9$$

The deciles  $D_0, D_1, \dots, D_9, D_{10}$  are defined as the corresponding proportion of values:

$$D_0 = L_0 = 0, D_1 = L_{d1}, D_2 = L_{d2}, \dots, D_9 = L_{d9}, D_{10} = L_N = 1$$

When the total number of loans  $N$  is not divisible by 10, no indices match the exact proportion of loans 0.1, 0.2, and so on. In that case, the decile values are linearly interpolated from the Lorenz curve data (that is, from the  $p$  and  $L$  arrays). With this definition, there are 11 values in the deciles information because the end points 0% and 100% are included.

## Gini Index

The *Gini index* (or coefficient) is visualized on a Lorenz curve plot as the area between the diagonal and the Lorenz curve.

Technically, the Gini index is the ratio of that area to the area of the full triangle under the diagonal on the Lorenz curve plot. The Gini index is also defined equivalently as the average absolute difference between all the weights in the portfolio normalized by the average weight.

Using the proportion of values that array  $L$  defined in the Lorenz curve section, the Gini index is given by the formula:

$$Gini = 1 - \frac{1}{N} \sum_{i=1}^N (L_{i-1} + L_i)$$

Equivalently, the Gini index can be computed from the sorted weights directly with the formula:

$$Gini = \frac{1}{N} \sum_{i=1}^N (2i - 1)w_{[i]} - 1$$

The Gini coefficient values are always between 0 (full diversification) and  $1 - 1/N$  (full concentration).

## Herfindahl-Hirschman Index

The *Herfindahl-Hirschman index* is commonly used as a measure of market concentration.

The formula for the Herfindahl-Hirschman index is:

$$HH = \sum_{i=1}^N w_i^2$$

The Herfindahl-Hirschman index takes values between  $1/N$  (full diversification) and 1 (full concentration).

## Hannah-Kay Index

The *Hannah-Kay index* is a generalization of the Herfindahl-Hirschman index.

The formula for the Hannah-Kay depends on a parameter  $\alpha > 0$ ,  $\alpha \neq 1$ , as follows:

$$HK_{\alpha} = \left( \sum_{i=1}^N w_i^{\alpha} \right)^{1/(\alpha-1)}$$

This formula is the reciprocal of the original Hannah-Kay index, which is defined with  $1/(1-\alpha)$  in the exponent. For concentration analysis, the reciprocal formula is the standard because it increases as the concentration increases. This is the formula implemented in `concentrationIndices`. The Hannah-Kay index takes values between  $1/N$  (full diversification) and 1 (full concentration).

## Hall-Tideman Index

The *Hall-Tideman index* is a measure commonly used for market concentration.

The formula for the Hall-Tideman index is:

$$HT = \frac{1}{2 \sum_{i=1}^N (N-i+1)w_{[i]} - 1}$$

The Hall-Tideman index takes values between  $1/N$  (full diversification) and 1 (full concentration).

## Theil Entropy Index

The *Theil entropy index*, based on a traditional entropy measure (for example, Shannon entropy), is adjusted so that it increases as concentration increases (entropy moves in the opposite direction), and shifted to make it positive.

The formula for the Theil entropy index is:

$$TE = \sum_{i=1}^N w_i \log(w_i) + \log(N)$$

The Theil entropy index takes values between 0 (full diversification) and  $\log(N)$  (full concentration).

## References

Basel Committee on Banking Supervision. "*Studies on Credit Risk Concentration*". Working paper no. 15. November, 2006.

Calabrese, R., and F. Porro. "Single-name concentration risk in credit portfolios: a comparison of concentration indices." working paper 201214, Geary Institute, University College, Dublin, May, 2012.

Lütkebohmert, E. *Concentration Risk in Credit Portfolios*. Springer, 2009.

## See Also

### Topics

"Analyze the Sensitivity of Concentration to a Given Exposure" on page 4-35

"Compare Concentration Indices for Random Portfolios" on page 4-38

"Concentration Indices" on page 1-14

### Introduced in R2017a

# creditCopula

(Not recommended) Simulate and analyze multifactor credit default model

## Compatibility

`creditCopula` is not recommended. Use `creditDefaultCopula` instead.

## Description

The `creditCopula` class simulates portfolio losses due to counterparty defaults using a multifactor model. The inputs to the model describe the credit-sensitive portfolio of exposures:

- `EAD` — Exposure at default
- `PD` — Probability of default
- `LGD` — Loss given default ( $1 - Recovery$ )
- `Weights` — Factor and idiosyncratic model weights

After the `creditCopula` object is created using the `creditCopula` function, use the `simulate` function to simulate credit defaults using the multifactor model. The results are stored in the form of a distribution of losses at the portfolio and counterparty level. Several risk measures at the portfolio level are calculated, as well as the risk contributions from individual obligors. The model calculates:

- The full simulated distribution of portfolio losses across scenarios
- The losses on each counterparty across scenarios
- Several risk measures (VaR, CVaR, EL, Std) with confidence intervals
- Risk contributions per counterparty (for EL and CVaR)

## Create Object

To create a `creditCopula` object, use the `creditCopula` function.



## Properties

### Portfolio — Details of credit portfolio

table

Details of credit portfolio, specified as a MATLAB table that contains all the portfolio data that was passed as input into the `creditCopula` constructor.

The `Portfolio` table has a column for each of the constructor inputs (EAD, PD, LGD, Weights, and ID). Each row of the table represents one counterparty.

For example:

ID	EAD	PD	LGD	Weights
1	122.43	0.064853	0.68024	0.3 0.7
2	70.386	0.073957	0.59256	0.3 0.7
3	79.281	0.066235	0.52383	0.3 0.7
4	113.42	0.01466	0.43977	0.3 0.7
5	100.46	0.0042036	0.41838	0.3 0.7

Data Types: table

### FactorCorrelation — Correlation matrix for credit factors

matrix

Correlation matrix for credit factors, specified as a NumFactors-by-NumFactors matrix. Specify the correlation matrix using the optional name-value pair argument `FactorCorrelation` with the `creditCopula` constructor.

Data Types: double

### VaRLevel1 — Value at Risk Level

numeric between 0 and 1

Value at risk level used when reporting VaR and CVaR, specified using an optional name-value pair argument `'VaRLevel'` with the `creditCopula` constructor.

Data Types: double

### PortfolioLosses — Total portfolio losses

vector

Total portfolio losses, specified as a NumScenarios-by-1 vector. The `PortfolioLosses` property is empty after construction with the `creditCopula` constructor. After the

`simulate` function is invoked, the `PortfolioLosses` property is populated with the vector of portfolio losses.

Data Types: `double`

### **CounterpartyLosses — Individual counterparty losses across all scenarios**

sparse matrix

Individual counterparty losses across all scenarios, specified as a `NumCounterparties-by-NumScenarios` sparse matrix of counterparty losses. The `CounterpartyLosses` property is empty after construction with the `creditCopula` constructor. After the `simulate` function is invoked, it is populated with the losses for each counterparty over each scenario. This matrix can become large, and for this reason it is stored as a sparse matrix.

Data Types: `double`

## **Object Functions**

<code>creditCopula</code>	(Not recommended) Create <code>creditCopula</code> object
<code>simulate</code>	(Not recommended) Simulate credit defaults using a <code>creditCopula</code> object
<code>portfolioRisk</code>	(Not recommended) Generate portfolio-level risk measurements
<code>riskContribution</code>	(Not recommended) Generate risk contributions for each counterparty in portfolio
<code>confidenceBands</code>	(Not recommended) Confidence interval bands

## **Examples**

### **Create a `creditCopula` Object and Simulate Credit Portfolio Losses**

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F, 'FactorCorrelation',FactorCorr2F)
```

Warning: creditCopula will be removed in a future release. Use creditDefaultCopula instead.

```
cc =
```

```
creditCopula with properties:
```

```
    Portfolio: [100×5 table]
  FactorCorrelation: [2×2 double]
    VaRLevel: 0.9500
  PortfolioLosses: []
CounterpartyLosses: []
```

Set the VaRLevel to 99%.

```
cc.VaRLevel = 0.99;
```

Simulate 100,000 scenarios and view the portfolio risk measures.

```
cc = simulate(cc,1e5)
portRisk = portfolioRisk(cc)
```

```
cc =
```

```
creditCopula with properties:
```

```
    Portfolio: [100×5 table]
  FactorCorrelation: [2×2 double]
    VaRLevel: 0.9900
  PortfolioLosses: [1×100000 double]
CounterpartyLosses: [100×100000 double]
```

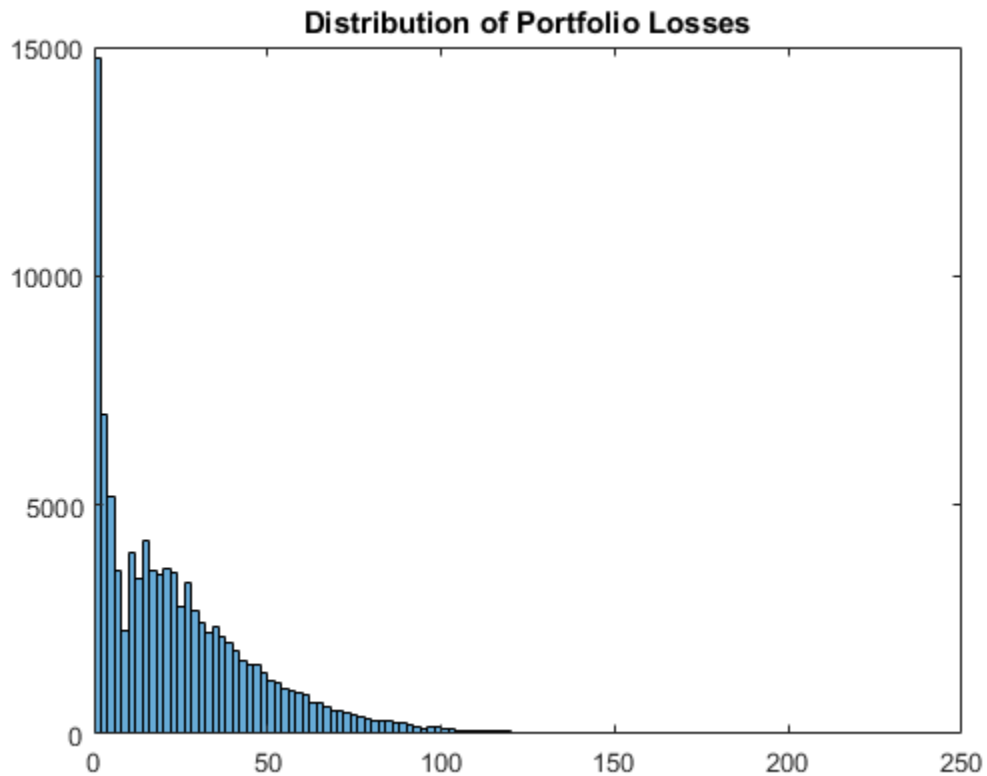
```
portRisk =
```

```
1×4 table
```

EL	Std	VaR	CVaR
24.774	23.693	101.57	120.22

View a histogram of the portfolio losses.

```
histogram(cc.PortfolioLosses);  
title('Distribution of Portfolio Losses');
```



For further analysis, use the `simulate`, `portfolioRisk`, and `riskContribution` functions with the `creditCopula` object.

### References

Crouhy, M., Galai, D., and Mark, R. "A Comparative Analysis of Current Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditCopula` | `creditDefaultCopula` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

**Introduced in R2016b**

# creditCopula

(Not recommended) Create creditCopula object

## Compatibility

creditCopula is not recommended. Use creditDefaultCopula instead.

## Syntax

```
cc = creditCopula(EAD,PD,LGD,Weights)
cc = creditCopula( ____,Name,Value)
```

## Description

cc = creditCopula(EAD,PD,LGD,Weights) creates a creditCopula object. For more information on using a creditCopula object, see creditCopula.

cc = creditCopula( \_\_\_\_,Name,Value) adds optional name-value pair arguments. For more information on using a creditCopula object, see creditCopula.

## Examples

### Create a creditCopula Object and Simulate Credit Portfolio Losses

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a creditCopula object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F, 'FactorCorrelation',FactorCorr2F)
```

Warning: creditCopula will be removed in a future release. Use creditDefaultCopula instead.

```
cc =
  creditCopula with properties:
      Portfolio: [100×5 table]
      FactorCorrelation: [2×2 double]
      VaRLevel: 0.9500
      PortfolioLosses: []
      CounterpartyLosses: []
```

Set the VaRLevel to 99%.

```
cc.VaRLevel = 0.99;
```

Simulate 100,000 scenarios and view the portfolio risk measures.

```
cc = simulate(cc,1e5)
portRisk = portfolioRisk(cc)
```

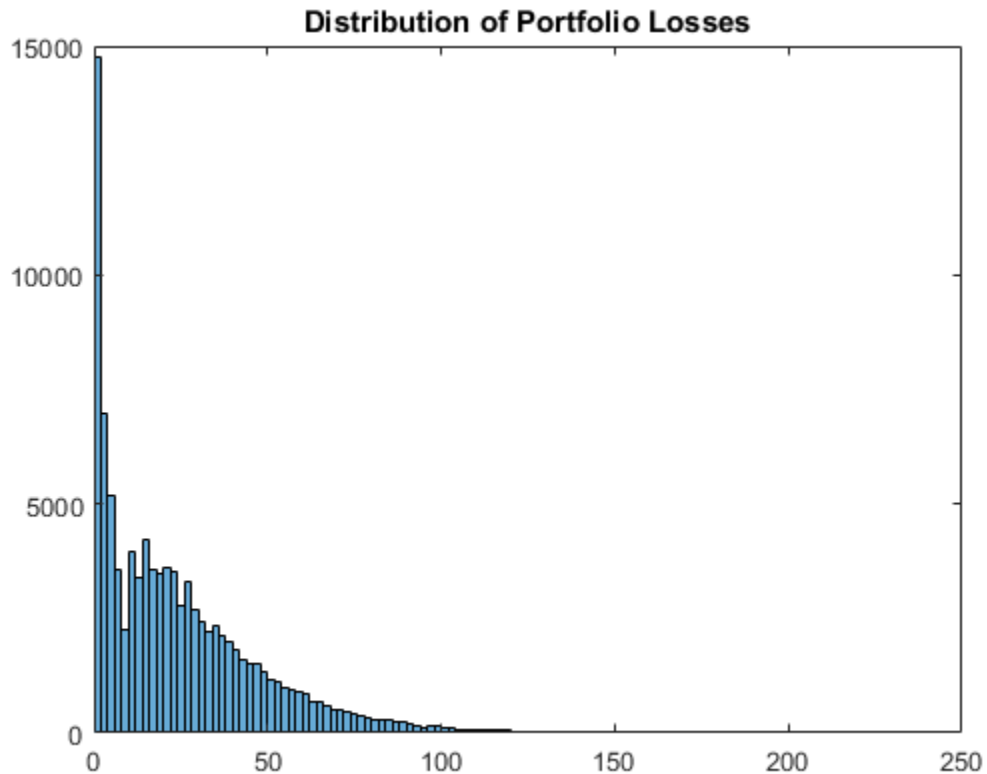
```
cc =
  creditCopula with properties:
      Portfolio: [100×5 table]
      FactorCorrelation: [2×2 double]
      VaRLevel: 0.9900
      PortfolioLosses: [1×100000 double]
      CounterpartyLosses: [100×100000 double]
```

```
portRisk =
  1×4 table
      EL      Std      VaR      CVaR
      _____
      24.774   23.693   101.57   120.22
```

View a histogram of the portfolio losses.

```
histogram(cc.PortfolioLosses);
```

```
title('Distribution of Portfolio Losses');
```



For further analysis, use the `simulate`, `portfolioRisk`, and `riskContribution` functions with the `creditCopula` object.

## Input Arguments

### **EAD** — Exposure at default

numeric vector

Exposure at default, specified as a `NumCounterparties-by-1` vector of credit exposures.



---

**Note:** The `creditCopula` model simulates defaults and losses over some fixed time period (for example, one year). The counterparty exposures (EAD) and default probabilities (PD) must both be specific to a particular time.

---

Data Types: `double`

### **PD — Probability of default**

numeric vector with elements between 0 and 1

Probability of default, specified as a `NumCounterparties-by-1` numeric vector with elements between 0 and 1, representing the default probabilities for the counterparties.

---

**Note:** The `creditCopula` model simulates defaults and losses over some fixed time period (for example, one year). The counterparty exposures (EAD) and default probabilities (PD) must both be specific to a particular time.

---

Data Types: `double`

### **LGD — Loss given default**

numeric vector with elements between 0 and 1

Loss given default, specified as a `NumCounterparties-by-1` numeric vector with elements between 0 and 1, representing the fraction of exposure that is lost when a counterparty defaults. LGD is defined as  $(1 - Recovery)$ . For example, a LGD of 0.6 implies a 40% recovery rate in the event of a default.

Data Types: `double`

### **Weights — Factor and idiosyncratic weights**

array of factor and idiosyncratic weights

Factor and idiosyncratic weights, specified as a `NumCounterparties-by-(NumFactors + 1)` array. Each row contains the factor weights for a particular counterparty. Each column contains the weights for an underlying risk factor. The last column in `Weights` contains the idiosyncratic risk weight for each counterparty. The idiosyncratic weight represents the company-specific credit risk. The total of the weights for each counterparty (that is, each row) must sum to 1.

For example, if a counterparty's creditworthiness was composed of 60% US, 20% European, and 20% idiosyncratic, then the `Weights` vector would be `[0.6 0.2 0.2]`.

Data Types: double

### Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

Example: `cc = creditCopula(EAD,PD,LGD,Weights,'VaRLevel',0.99)`

#### 'ID' — User-defined IDs for counterparties

ID is numeric vector (default) | vector

User-defined IDs for counterparties, specified as a `NumCounterparties-by-1` vector of IDs for each counterparty. ID is used to identify exposures in the `Portfolio` table and the risk contribution table. ID must be a numeric, a string array, or a cell array of character vectors. If unspecified, ID defaults to a numeric vector.

Data Types: double | string | cell

#### 'VaRLevel' — Value at risk level

0.95 (default) | decimal

Value at risk level (used for reporting VaR and CVaR) specified as a decimal.

Data Types: double

#### 'FactorCorrelation' — Factor correlation matrix

identity matrix (default) | correlation matrix

Factor correlation matrix, specified as a `NumFactors-by-NumFactors` matrix that defines the correlation between the risk factors. If not specified, the factor correlation matrix defaults to an identity matrix, meaning that factors are not correlated.

Data Types: double

## Output Arguments

### `cc` — `creditCopula` object

object

`creditCopula` object contains the provided portfolio data, that is stored in the `Portfolio` property.

The `Portfolio` property is a table that has a column for each of the constructor inputs (EAD, PD, LGD, Weights, and ID). Each row of the `Portfolio` table represents one counterparty.

The `creditCopula` object has the following properties:

- **Portfolio:**

A table with the following variables:

- **ID** — An ID to identify each counterparty
- **EAD** — Exposure at default
- **PD** — Probability of default
- **LGD** — Loss given default
- **Weights** — Factor and idiosyncratic weights for counterparties

- **FactorCorrelation:**

Factor correlation matrix, a `NumFactors`-by-`NumFactors` matrix that defines the correlation between the risk factors.

- **VaRLevel:**

The value-at-risk level, used when reporting VaR and CVaR.

- **PortfolioLosses**

Portfolio losses, a `NumScenarios`-by-1 vector of portfolio losses. This property is empty until the `simulate` function is used.

- **CounterpartyLosses**

Counterparty losses, a `NumCounterparties`-by-`NumScenarios` sparse matrix of individual counterparty losses across the scenarios. This property is empty until the `simulate` function is used.

For more information on `creditCopula` objects, see `creditCopula`.

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditDefaultCopula` | `portfolioRisk` |  
`riskContribution` | `simulate` | `table`

**Introduced in R2016b**

# confidenceBands

(Not recommended) Confidence interval bands

## Compatibility

`creditCopula` is not recommended. Use `creditDefaultCopula` instead.

## Syntax

```
cbTable = confidenceBands(cc)  
cbTable = confidenceBands(cc,Name,Value)
```

## Description

`cbTable = confidenceBands(cc)` returns a table of the requested risk measure and its associated confidence bands. `confidenceBands` is used to investigate how the values of a risk measure and its associated confidence interval converge as the number of scenarios increases. The `simulate` function must be run before `confidenceBands` is used. For more information on using a `creditCopula` object, see `creditCopula`.

`cbTable = confidenceBands(cc,Name,Value)` adds optional name-value pair arguments. The `simulate` function must be run before `confidenceBands` is used. For more information on using a `creditCopula` object, see `creditCopula`.

## Examples

### Generate a Table of the Associated Confidence Bands for a Requested Risk Measure for a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F, 'FactorCorrelation',FactorCorr2F)
```

Warning: creditCopula will be removed in a future release. Use creditDefaultCopula instead.

```
cc =
```

```
creditCopula with properties:
```

```
Portfolio: [100x5 table]
FactorCorrelation: [2x2 double]
VaRLevel: 0.9500
PortfolioLosses: []
CounterpartyLosses: []
```

Set the VaRLevel to 99%.

```
cc.VaRLevel = 0.99;
```

Use the simulate function before running confidenceBands. Use confidenceBands with the creditCopula object to generate the cbTable.

```
cc = simulate(cc,1e5);
cbTable = confidenceBands(cc, 'RiskMeasure', 'Std', 'ConfidenceIntervalLevel',0.9);
cbTable(1:10,:)
```

```
ans =
```

```
10x4 table
```

NumScenarios	Lower	Std	Upper
1000	22.796	23.633	24.538
2000	22.62	23.207	23.828
3000	23.082	23.572	24.084
4000	23.125	23.549	23.991
5000	23.228	23.61	24.005
6000	23.372	23.723	24.085
7000	23.378	23.702	24.037
8000	23.268	23.57	23.881
9000	23.419	23.706	24.001
10000	23.467	23.739	24.019

## Input Arguments

### **cc** — **creditCopula** object

object

`creditCopula` object, specified using the updated `creditCopula` object obtained from running the `simulate` function, which must be run before `confidenceBands` is used.

For more information on `creditCopula` objects, see `creditCopula`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, . . . , `NameN`, `ValueN`.

Example: `cbTable =`

```
confidenceBands(cc, 'RiskMeasure', 'Std', 'ConfidenceIntervalLevel', 0.92, 'NumPoints')
```

### 'RiskMeasure' — Risk measure to investigate

CVaR (default) | character vector with values 'EL', 'Std', 'VaR', or 'CVaR'

Risk measure to investigate, specified as character vector with possible values:

- 'EL' — Expected loss, the mean of portfolio losses
- 'Std' — Standard deviation of the losses
- 'VaR' — Value at risk at the threshold specified by the `VarLevel` property of the `creditCopula` object
- 'CVaR' — Conditional VaR at the threshold specified by the `VarLevel` property of the `creditCopula` object

Data Types: char

### 'ConfidenceIntervalLevel' — Confidence interval level

0.95 (default) | decimal

Confidence interval level, specified as a decimal. For example, if you specify 0.95, a 95% confidence interval is reported in the output table (`cbTable`).

Data Types: `double`

### 'NumPoints' — Number of scenario samples to report

100 (default) | nonnegative integer

Number of scenario samples to report, specified as a nonnegative integer. The default is 100, meaning confidence bands are reported at 100 evenly spaced points of increasing sample size ranging from 0 to the total number of simulated scenarios.

---

**Note:** `NumPoints` must be a numeric scalar greater than 1, and is typically much smaller than total number of scenarios simulated. `confidenceBands` should be used to obtain a qualitative idea of how fast a risk measure and its confidence interval are converging. Specifying a large value for `NumPoints` is not recommended and could cause performance issues with `confidenceBands`.

---

Data Types: `double`

## Output Arguments

### `cbTable` — Requested risk measure and associated confidence bands

table

Requested risk measure and associated confidence bands at each of the `NumPoints` scenario sample sizes, returned as a table containing the following columns:

- `NumScenarios` — Number of scenarios at the sample point
- `Lower` — Lower confidence band
- *RiskMeasure* — Requested risk measure where the column takes its name from whatever risk measure is requested with the optional input `RiskMeasure`
- `Upper` — Upper confidence band

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.



Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`creditCopula` | `creditDefaultCopula` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

**Introduced in R2016b**

## portfolioRisk

(Not recommended) Generate portfolio-level risk measurements

### Compatibility

`creditCopula` is not recommended. Use `creditDefaultCopula` instead.

### Syntax

```
[riskMeasures,confidenceIntervals] = portfolioRisk(cc)  
[riskMeasures,confidenceIntervals] = portfolioRisk(cc,Name,Value)
```

### Description

`[riskMeasures,confidenceIntervals] = portfolioRisk(cc)` returns tables of risk measurements for the portfolio losses. The `simulate` function must be run before `portfolioRisk` is used. For more information on using a `creditCopula` object, see `creditCopula`.

`[riskMeasures,confidenceIntervals] = portfolioRisk(cc,Name,Value)` adds an optional name-value pair argument for `ConfidenceIntervalLevel`. The `simulate` function must be run before `portfolioRisk` is used.

### Examples

#### Generate Tables for Risk Measure and Confidence Intervals for a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F, 'FactorCorrelation',FactorCorr2F)
```

Warning: creditCopula will be removed in a future release. Use creditDefaultCopula instead.

```
cc =
```

```
creditCopula with properties:
```

```
Portfolio: [100x5 table]
FactorCorrelation: [2x2 double]
VaRLevel: 0.9500
PortfolioLosses: []
CounterpartyLosses: []
```

Set the VaRLevel to 99%.

```
cc.VaRLevel = 0.99;
```

Use the `simulate` function before running `portfolioRisk`. Then use `portfolioRisk` with the `creditCopula` object to generate the `riskMeasure` and `ConfidenceIntervals` tables.

```
cc = simulate(cc,1e5);
[riskMeasure,confidenceIntervals] = portfolioRisk(cc,'ConfidenceIntervalLevel',0.9)
```

```
riskMeasure =
```

```
1x4 table
```

EL	Std	VaR	CVaR
24.774	23.693	101.57	120.22

```
confidenceIntervals =
```

```
1x4 table
```

EL	Std	VaR	CVaR
----	-----	-----	------

24.65    24.897    23.606    23.78    100.83    102.57    119.28    121.17

## Input Arguments

### **cc** — **creditCopula** object

object

`creditCopula` object, specified as an updated `creditCopula` object obtained by running the `simulate` function, which must be run before `portfolioRisk` is used.

For more information on `creditCopula` objects, see `creditCopula`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

```
Example: [riskMeasure, confidenceIntervals] =  
portfolioRisk(cc, 'ConfidenceIntervalLevel', 0.92)
```

### '**ConfidenceIntervalLevel**' — Confidence interval level

0.95 (default) | decimal

Confidence interval level, specified as a decimal. For example, if you specify `0.95`, a 95% confidence interval is reported in the output table (`riskMeasures`).

Data Types: `double`

## Output Arguments

### **riskMeasures** — Risk measures

table

Risk measures, returned as a table containing the following columns:

- `EL` — Expected loss, the mean of portfolio losses
- `Std` — Standard deviation of the losses

- VaR — Value at risk at the threshold specified by the `VarLevel` property of the `creditCopula` object
- CVaR — Conditional VaR at the threshold specified by the `VarLevel` property of the `creditCopula` object

### **confidenceIntervals — Confidence intervals**

table

Confidence intervals, returned as a table of confidence intervals corresponding to the portfolio risk measures reported in the `riskMeasures` table. Confidence intervals are reported at the level specified by the `ConfidenceIntervalLevel` parameter.

## **References**

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## **See Also**

### **See Also**

`confidenceBands` | `creditCopula` | `creditDefaultCopula` | `riskContribution`  
| `simulate` | `table`

**Introduced in R2016b**

# riskContribution

(Not recommended) Generate risk contributions for each counterparty in portfolio

## Compatibility

`creditCopula` is not recommended. Use `creditDefaultCopula` instead.

## Syntax

```
riskContributions = riskContribution(cc)
```

## Description

`riskContributions = riskContribution(cc)` returns a table of risk contributions for each counterparty in the portfolio. The `riskContributions` table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

For more information on using a `creditCopula` object, see `creditCopula`.

## Examples

### Determine the Risk Contribution for Each Counterparty for a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

Warning: `creditCopula` will be removed in a future release. Use `creditDefaultCopula` instead.

```
cc =
```

```
creditCopula with properties:
```

```

        Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
        VaRLevel: 0.9500
PortfolioLosses: []
CounterpartyLosses: []

```

Set the `VaRLevel` to 99%.

```
cc.VaRLevel = 0.99;
```

Use the `simulate` function before running `riskContribution`. Then use `riskContribution` with the `creditCopula` object to generate the `riskContributions`.

```
cc = simulate(cc,1e5);
riskContributions = riskContribution(cc);
riskContributions(1:10,:)
```

```
ans =
```

```
10×3 table
```

ID	EL	CVaR
1	0.038604	0.12868
2	0.067068	0.24527
3	1.2527	2.3103
4	0.0023253	0.0026274
5	0.11766	0.26223
6	0.12437	0.47915
7	0.82913	1.6516
8	0.00085629	0.00089197
9	0.91406	4.009
10	0.24352	2.2781

## Input Arguments

**cc** — **creditCopula** object  
object

`creditCopula` object, specified as an updated `creditCopula` object obtained by running the `simulate` function, which must be run before `riskContribution` is used.

For more information on `creditCopula` objects, see `creditCopula`.

## Output Arguments

### **riskContributions** — Risk contributions

table

Risk contributions, returned as a table containing the following risk contributions for each counterparty:

- **EL** — The expected loss for the particular counterparty over the scenarios
- **CVaR** — The conditional value at risk for the particular counterparty over the scenarios

The `riskContributions` table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Glasserman, P. “Measuring Marginal Risk Contributions in Credit Portfolios.” *Journal of Computational Finance*. Vol. 9, No. 2, Winter 2005/2006.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Kalkbrenner, M., Lotter, H., and Overbeck, L. “Sensible and Efficient Capital Allocation for Credit Portfolios.” *Risk*. 17, 2004, pp. S19–S24.



Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditCopula` | `creditDefaultCopula` | `portfolioRisk` | `simulate` | `table`

**Introduced in R2016b**

# simulate

(Not recommended) Simulate credit defaults using a `creditCopula` object

## Compatibility

`creditCopula` is not recommended. Use `creditDefaultCopula` instead.

## Syntax

```
cc = simulate(cc,NumScenarios)
cc = simulate( __ ,Name,Value)
```

## Description

`cc = simulate(cc,NumScenarios)` performs the full simulation of credit scenarios and computes defaults and losses for the portfolio defined in the `creditCopula` object.

For more information on using a `creditCopula` object, see `creditCopula`.

`cc = simulate( __ ,Name,Value)` adds optional name-value pair arguments for (Copula, DegreesOfFreedom, and BlockSize).

## Examples

### Run a Simulation Using a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

Warning: creditCopula will be removed in a future release. Use creditDefaultCopula instead.

```
cc =
    creditCopula with properties:
        Portfolio: [100×5 table]
        FactorCorrelation: [2×2 double]
        VaRLevel: 0.9500
        PortfolioLosses: []
        CounterpartyLosses: []
```

Set the VaRLevel to 99%.

```
cc.VaRLevel = 0.99;
```

Use the `simulate` function with the `creditCopula` object. After using `simulate`, you can then use the `portfolioRisk`, `riskContribution`, and `confidenceBands` functions with the updated `creditCopula` object.

```
cc = simulate(cc,1e5,'Copula','t','DegreesOfFreedom',10)
```

```
cc =
    creditCopula with properties:
        Portfolio: [100×5 table]
        FactorCorrelation: [2×2 double]
        VaRLevel: 0.9900
        PortfolioLosses: [1×100000 double]
        CounterpartyLosses: [100×100000 double]
```

For instance, you can use `riskContribution` with the `creditCopula` object to generate the `riskContributions`.

```
riskContributions = riskContribution(cc);
riskContributions(1:10,:)
```

```
ans =
```

10×3 table

ID	EL	CVaR
1	0.034744	0.55257
2	0.07008	0.40727
3	1.2406	3.3644
4	0.002785	0.056489
5	0.11933	0.47042
6	0.12519	0.73861
7	0.83928	2.591
8	0.000999	0.026759
9	0.93426	5.7206
10	0.25294	3.9015

## Input Arguments

### **cc** — **creditCopula** object

object

`creditCopula` object, obtained by running the `creditCopula` constructor.

For more information on a `creditCopula` object, see `creditCopula`.

### **NumScenarios** — **Number of scenarios to simulate**

nonnegative integer

Number of scenarios to simulate, specified as a nonnegative integer. Scenarios are processed in blocks to conserve machine resources.

Data Types: `double`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

Example: `obj=`  
`simulate(obj,NumScenarios,'Copula','t','DegreesOfFreedom',5)`

**'Copula' — Type of copula**

'Gaussian' (default) | character vector with values 'Gaussian' or 't'

Type of copula, specified as a character vector with a value of:

- 'Gaussian' — A Gaussian copula
- 't' — A  $t$  copula with degrees of freedom specified using `DegreesOfFreedom`.

Data Types: char

**'DegreesOfFreedom' — Degrees of freedom for t copula**

5 degrees of freedom (default) | nonnegative numeric value

Degrees of freedom for a  $t$  copula, specified as a nonnegative numeric value. If `Copula` is set to 'Gaussian', the `DegreesOfFreedom` parameter is ignored.

Data Types: double

**'BlockSize' — Number of scenarios to process in each iteration**

1e5/Number of counterparties (default) | nonnegative numeric value

Number of scenarios to process in each iteration, specified as a nonnegative numeric value.

Data Types: double

## Output Arguments

**cc — Updated creditCopula object**

object

Updated `creditCopula` object. The object is populated with the simulated `PortfolioLosses` and `CounterpartyLosses`.

For more information on a `creditCopula` object, see `creditCopula`.

---

**Note:** In the `simulate` function, the `Weights` are transformed to ensure that the latent variables have a mean of 0 and a variance of 1.

---

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditCopula` | `creditDefaultCopula` | `portfolioRisk` | `riskContribution` | `table`

**Introduced in R2016b**

# creditDefaultCopula

Simulate and analyze multifactor credit default model

## Description

The `creditDefaultCopula` class simulates portfolio losses due to counterparty defaults using a multifactor model. `creditDefaultCopula` associates each counterparty with a random variable, called a latent variable, which is mapped to default/non-default outcomes for each scenario such that defaults occur with probability PD. In the event of default, a loss for that scenario is recorded equal to  $EAD * LGD$  for the counterparty. These latent variables are simulated using a multi-factor model, where systemic credit fluctuations are modeled with a series of risk factors. These factors can be based on industry sectors (such as financial, aerospace), geographical regions (such as USA, Eurozone), or any other underlying driver of credit risk. Each counterparty is assigned a series of weights which determine their sensitivity to each underlying credit factors.

The inputs to the model describe the credit-sensitive portfolio of exposures:

- **EAD** — Exposure at default
- **PD** — Probability of default
- **LGD** — Loss given default ( $1 - Recovery$ )
- **Weights** — Factor and idiosyncratic model weights

After the `creditDefaultCopula` object is created using the `creditDefaultCopula` function, use the `simulate` function to simulate credit defaults using the multifactor model. The results are stored in the form of a distribution of losses at the portfolio and counterparty level. Several risk measures at the portfolio level are calculated, as well as the risk contributions from individual obligors. The model calculates:

- Full simulated distribution of portfolio losses across scenarios
- Losses on each counterparty across scenarios
- Several risk measures (VaR, CVaR, EL, Std) with confidence intervals
- Risk contributions per counterparty (for EL and CVaR)

## Create Object

To create a `creditDefaultCopula` object, use the `creditDefaultCopula` function.

## Properties

### **Portfolio** — Details of credit portfolio

table

Details of credit portfolio, specified as a MATLAB table that contains all the portfolio data that was passed as input into the `creditDefaultCopula` constructor.

The `Portfolio` table has a column for each of the constructor inputs (EAD, PD, LGD, Weights, and ID). Each row of the table represents one counterparty.

For example:

ID	EAD	PD	LGD	Weights
1	122.43	0.064853	0.68024	0.3 0.7
2	70.386	0.073957	0.59256	0.3 0.7
3	79.281	0.066235	0.52383	0.3 0.7
4	113.42	0.01466	0.43977	0.3 0.7
5	100.46	0.0042036	0.41838	0.3 0.7

Data Types: table

### **FactorCorrelation** — Correlation matrix for credit factors

matrix

Correlation matrix for credit factors, specified as a `NumFactors`-by-`NumFactors` matrix. Specify the correlation matrix using the optional name-value pair argument `FactorCorrelation` with the `creditDefaultCopula` constructor.

Data Types: double

### **VaRLevel1** — Value at Risk Level

numeric between 0 and 1

Value at risk level used when reporting VaR and CVaR, specified using an optional name-value pair argument `'VaRLevel1'` with the `creditDefaultCopula` constructor.



Data Types: double

### **PortfolioLosses** — Total portfolio losses

vector

Total portfolio losses, specified as a 1-by-NumScenarios vector. The `PortfolioLosses` property is empty after construction with the `creditDefaultCopula` constructor. After the `simulate` function is invoked, the `PortfolioLosses` property is populated with the vector of portfolio losses.

Data Types: double

## **Object Functions**

`creditDefaultCopula`  
`simulate`

`portfolioRisk`  
`riskContribution`

`confidenceBands`  
`getScenarios`

Create `creditDefaultCopula` object  
Simulate credit defaults using a  
`creditDefaultCopula` object  
Generate portfolio-level risk measurements  
Generate risk contributions for each  
counterparty in portfolio  
Confidence interval bands  
Counterparty scenarios

## **Examples**

### **Create a creditDefaultCopula Object and Simulate Credit Portfolio Losses**

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditDefaultCopula` object with a two-factor model.

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```
Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
VaRLevel: 0.9500
PortfolioLosses: []
```

Set the `VaRLevel` to 99%.

```
cdc.VaRLevel = 0.99;
```

Simulate 100,000 scenarios, and view the portfolio risk measures.

```
cdc = simulate(cdc,1e5)
portRisk = portfolioRisk(cdc)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```
Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
VaRLevel: 0.9900
PortfolioLosses: [1×100000 double]
```

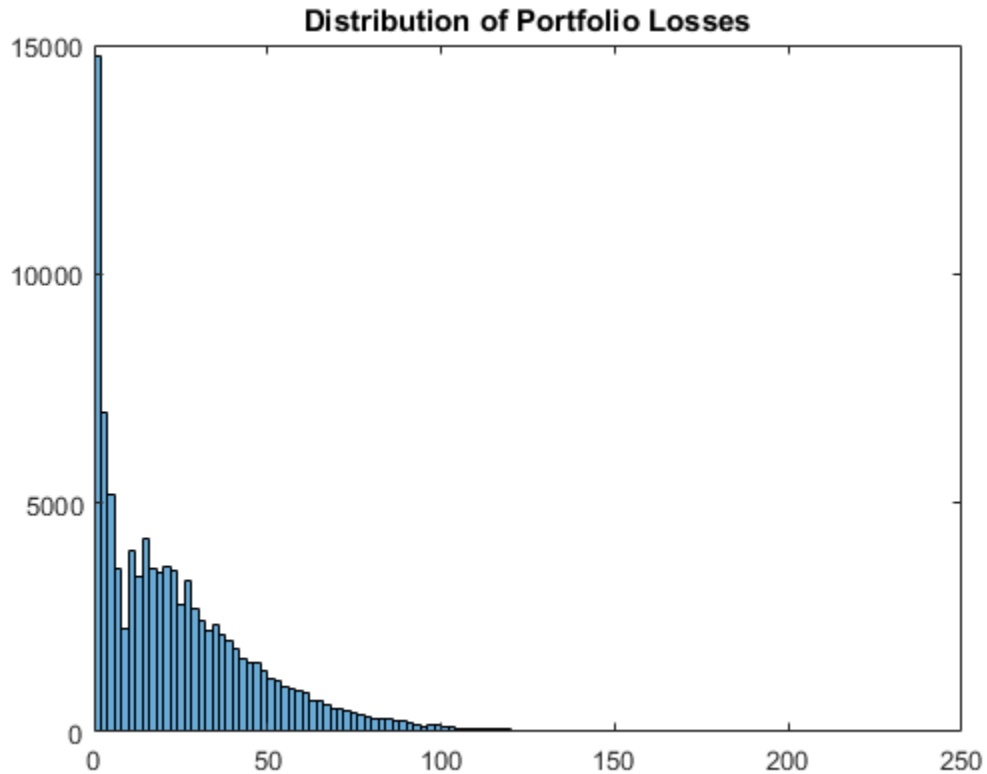
```
portRisk =
```

```
1×4 table
```

EL	Std	VaR	CVaR
24.774	23.693	101.57	120.22

View a histogram of the portfolio losses.

```
histogram(cdc.PortfolioLosses);
title('Distribution of Portfolio Losses');
```



For further analysis, use the `simulate`, `portfolioRisk`, `riskContribution`, and `getScenarios` functions with the `creditDefaultCopula` object.

- “Credit Simulation Using Copulas” on page 4-2
- “`creditDefaultCopula` Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `getscenarios` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

### Topics

“Credit Simulation Using Copulas” on page 4-2

“`creditDefaultCopula` Simulation Workflow” on page 4-6

“Modeling Correlated Defaults with Copulas” on page 4-21

“Corporate Credit Risk” on page 1-3

“Credit Simulation Using Copulas” on page 4-2

### Introduced in R2017a

# creditDefaultCopula

Create `creditDefaultCopula` object

## Syntax

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights)
cdc = creditDefaultCopula( ____,Name,Value)
```

## Description

`cdc = creditDefaultCopula(EAD,PD,LGD,Weights)` creates a `creditDefaultCopula` object. For more information on using a `creditDefaultCopula` object, see `creditDefaultCopula`.

`cdc = creditDefaultCopula( ____,Name,Value)` adds optional name-value pair arguments.

## Examples

### Create a `creditDefaultCopula` Object, and Simulate Credit Portfolio Losses

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditDefaultCopula` object with a two-factor model.

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights2F, 'FactorCorrelation',FactorCorr2F)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```
Portfolio: [100x5 table]
```

```
FactorCorrelation: [2×2 double]
    VaRLevel: 0.9500
PortfolioLosses: []
```

Set the `VaRLevel` to 99%.

```
cdc.VaRLevel = 0.99;
```

Simulate 100,000 scenarios, and view the portfolio risk measures.

```
cdc = simulate(cdc,1e5)
portRisk = portfolioRisk(cdc)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```
    Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
    VaRLevel: 0.9900
PortfolioLosses: [1×100000 double]
```

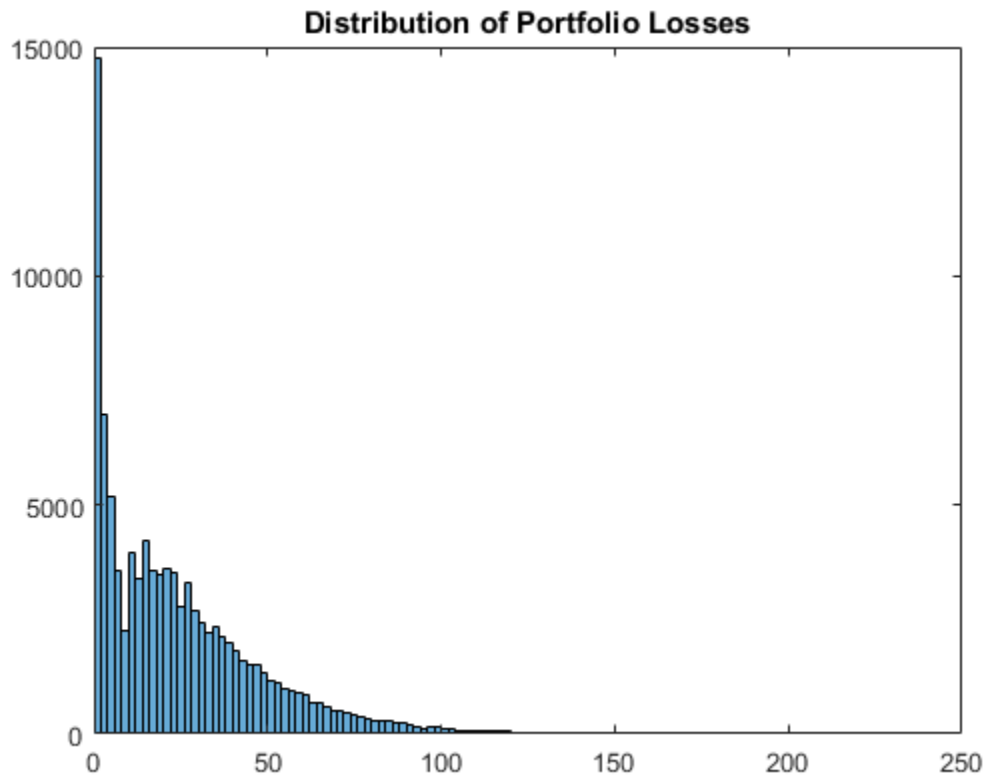
```
portRisk =
```

```
1×4 table
```

EL	Std	VaR	CVaR
24.774	23.693	101.57	120.22

View a histogram of the portfolio losses.

```
histogram(cdc.PortfolioLosses);
title('Distribution of Portfolio Losses');
```



For further analysis, use the `simulate`, `portfolioRisk`, `riskContribution`, `confidenceBands`, and `getScenarios` functions with the `creditDefaultCopula` object.

- “Credit Simulation Using Copulas” on page 4-2
- “creditDefaultCopula Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21

## Input Arguments

### **EAD — Exposure at default**

numeric vector

Exposure at default, specified as a `NumCounterparties-by-1` vector of credit exposures.

---

**Note:** The `creditDefaultCopula` model simulates defaults and losses over some fixed time period (for example, one year). The counterparty exposures (EAD) and default probabilities (PD) must both be specific to a particular time.

---

Data Types: `double`

### **PD — Probability of default**

numeric vector with elements from 0 through 1

Probability of default, specified as a `NumCounterparties-by-1` numeric vector with elements from 0 through 1, representing the default probabilities for the counterparties.

---

**Note:** The `creditDefaultCopula` model simulates defaults and losses over a fixed time period (for example, one year). The counterparty exposures (EAD) and default probabilities (PD) must both be specific to a particular time.

---

Data Types: `double`

### **LGD — Loss given default**

numeric vector with elements from 0 through 1

Loss given default, specified as a `NumCounterparties-by-1` numeric vector with elements from 0 through 1, representing the fraction of exposure that is lost when a counterparty defaults. LGD is defined as  $(1 - Recovery)$ . For example, an LGD of 0.6 implies a 40% recovery rate in the event of a default.

Data Types: `double`

### **Weights — Factor and idiosyncratic weights**

array of factor and idiosyncratic weights



Factor and idiosyncratic weights, specified as a `NumCounterparties-by-(NumFactors + 1)` array. Each row contains the factor weights for a particular counterparty. Each column contains the weights for an underlying risk factor. The last column in `Weights` contains the idiosyncratic risk weight for each counterparty. The idiosyncratic weight represents the company-specific credit risk. The total of the weights for each counterparty (that is, each row) must sum to 1.

For example, if a counterparty's creditworthiness was composed of 60% US, 20% European, and 20% idiosyncratic, then the `Weights` vector could be `[0.6 0.2 0.2]`.

Data Types: `double`

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `cdc = creditDefaultCopula(EAD,PD,LGD,Weights,'VaRLevel',0.99)`

### 'ID' — User-defined IDs for counterparties

`1:NumCounterparties` (default) | vector

User-defined IDs for counterparties, specified as the comma-separated pair consisting of `'ID'` and a `NumCounterparties-by-1` vector of IDs for each counterparty. `ID` is used to identify exposures in the `Portfolio` table and the risk contribution table. `ID` must be a numeric, a string array, or a cell array of character vectors. If unspecified, `ID` defaults to a numeric vector `1:NumCounterparties`.

Data Types: `double` | `string` | `cell`

### 'VaRLevel' — Value at risk level

`0.95` (default) | numeric between 0 and 1

Value at risk level (used for reporting VaR and CVaR), specified as the comma-separated pair consisting of `'VaRLevel'` and a numeric between 0 and 1.

Data Types: `double`

### 'FactorCorrelation' — Factor correlation matrix

identity matrix (default) | correlation matrix

Factor correlation matrix, specified as the comma-separated pair consisting of 'FactorCorrelation' and a NumFactors-by-NumFactors matrix that defines the correlation between the risk factors. If not specified, the factor correlation matrix defaults to an identity matrix, meaning that factors are not correlated.

Data Types: double

## Output Arguments

### **cdc — creditDefaultCopula object**

object

creditDefaultCopula object, returned as an object that contains the provided portfolio data that is stored in the Portfolio property.

The Portfolio property is a table that has a column for each of the constructor inputs (EAD, PD, LGD, Weights, and ID). Each row of the Portfolio table represents one counterparty.

The creditDefaultCopula object has the following properties:

- **Portfolio:**

A table with the following variables (each row of the table represents one counterparty):

- ID — ID to identify each counterparty
- EAD — Exposure at default
- PD — Probability of default
- LGD — Loss given default
- Weights — Factor and idiosyncratic weights for counterparties

- **FactorCorrelation:**

Factor correlation matrix, a NumFactors-by-NumFactors matrix that defines the correlation between the risk factors.

- **VaRLevel:**

The value-at-risk level, used when reporting VaR and CVaR.

- **PortfolioLosses**

Portfolio losses, a NumScenarios-by-1 vector of portfolio losses. This property is empty until the `simulate` function is used.

For more information on `creditDefaultCopula` objects, see `creditDefaultCopula`.

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

### Topics

“Credit Simulation Using Copulas” on page 4-2

“`creditDefaultCopula` Simulation Workflow” on page 4-6

“Modeling Correlated Defaults with Copulas” on page 4-21

“Corporate Credit Risk” on page 1-3

“Credit Simulation Using Copulas” on page 4-2

### Introduced in R2017a

## confidenceBands

Confidence interval bands

### Syntax

```
cbTable = confidenceBands(cdc)  
cbTable = confidenceBands(cdc,Name,Value)
```

### Description

`cbTable = confidenceBands(cdc)` returns a table of the requested risk measure and its associated confidence bands. `confidenceBands` is used to investigate how the values of a risk measure and its associated confidence interval converge as the number of scenarios increases. The `simulate` function must be run before `confidenceBands` is used. For more information on using a `creditDefaultCopula` object, see `creditDefaultCopula`.

`cbTable = confidenceBands(cdc,Name,Value)` adds optional name-value pair arguments.

### Examples

#### Generate a Table of the Associated Confidence Bands for a Requested Risk Measure for a `creditDefaultCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditDefaultCopula` object with a two-factor model.

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```

cdc =

  creditDefaultCopula with properties:

      Portfolio: [100×5 table]
  FactorCorrelation: [2×2 double]
      VaRLevel: 0.9500
  PortfolioLosses: []

```

Set the `VaRLevel` to 99%.

```
cdc.VaRLevel = 0.99;
```

Use the `simulate` function before running `confidenceBands`. Use `confidenceBands` with the `creditDefaultCopula` object to generate the `cbTable`.

```

cdc = simulate(cdc,1e5);
cbTable = confidenceBands(cdc, 'RiskMeasure', 'Std', 'ConfidenceIntervalLevel',0.9);
cbTable(1:10,:)

```

```
ans =
```

```
10×4 table
```

NumScenarios	Lower	Std	Upper
1000	22.796	23.633	24.538
2000	22.62	23.207	23.828
3000	23.082	23.572	24.084
4000	23.125	23.549	23.991
5000	23.228	23.61	24.005
6000	23.372	23.723	24.085
7000	23.378	23.702	24.037
8000	23.268	23.57	23.881
9000	23.419	23.706	24.001
10000	23.467	23.739	24.019

- “Credit Simulation Using Copulas” on page 4-2
- “creditDefaultCopula Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21

## Input Arguments

### **cdc** — **creditDefaultCopula** object

object

creditDefaultCopula object obtained after running the `simulate` function.

For more information on `creditDefaultCopula` objects, see `creditDefaultCopula`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( `' '`). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

Example: `cbTable =`

```
confidenceBands(cdc, 'RiskMeasure', 'Std', 'ConfidenceIntervalLevel', 0.9, 'NumPoin
```

### **'RiskMeasure'** — Risk measure to investigate

`'CVaR'` (default) | character vector or string with values `'EL'`, `'Std'`, `'VaR'`, or `'CVaR'`

Risk measure to investigate, specified as the comma-separated pair consisting of `'RiskMeasure'` and a character vector or string. Possible values are:

- `'EL'` — Expected loss, the mean of portfolio losses
- `'Std'` — Standard deviation of the losses
- `'VaR'` — Value at risk at the threshold specified by the `VarLevel` property of the `creditDefaultCopula` object
- `'CVaR'` — Conditional VaR at the threshold specified by the `VarLevel` property of the `creditDefaultCopula` object

Data Types: `char` | `string`

### **'ConfidenceIntervalLevel'** — Confidence interval level

0.95 (default) | numeric between 0 and 1

Confidence interval level, specified as the comma-separated pair consisting of `'ConfidenceIntervalLevel'` and a numeric between 0 and 1. For example, if you specify `0.95`, a 95% confidence interval is reported in the output table (`cbTable`).

Data Types: double

**'NumPoints' — Number of scenario samples to report**

100 (default) | nonnegative integer

Number of scenario samples to report, specified as the comma-separated pair consisting of 'NumPoints' and a nonnegative integer. The default is 100, meaning confidence bands are reported at 100 evenly spaced points of increasing sample size ranging from 0 to the total number of simulated scenarios.

---

**Note:** NumPoints must be a numeric scalar greater than 1, and is typically much smaller than total number of scenarios simulated. confidenceBands can be used to obtain a qualitative idea of how fast a risk measure and its confidence interval are converging. Specifying a large value for NumPoints is not recommended and could cause performance issues with confidenceBands.

---

Data Types: double

## Output Arguments

**cbTable — Requested risk measure and associated confidence bands**

table

Requested risk measure and associated confidence bands at each of the NumPoints scenario sample sizes, returned as a table containing the following columns:

- NumScenarios — Number of scenarios at the sample point
- Lower — Lower confidence band
- RiskMeasure — Requested risk measure where the column takes its name from whatever risk measure is requested with the optional input RiskMeasure
- Upper — Upper confidence band

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

### Topics

“Credit Simulation Using Copulas” on page 4-2

“`creditDefaultCopula` Simulation Workflow” on page 4-6

“Modeling Correlated Defaults with Copulas” on page 4-21

“Corporate Credit Risk” on page 1-3

“Credit Simulation Using Copulas” on page 4-2

### Introduced in R2017a



# getScenarios

Counterparty scenarios

## Syntax

```
scenarios = getScenarios(cdc,scenarioIndices)
```

## Description

`scenarios = getScenarios(cdc,scenarioIndices)` returns counterparty scenario details as a matrix of individual losses for each counterparty for the scenarios requested in `scenarioIndices`.

The `simulate` function must be run before `getScenarios` is used. For more information on using a `creditDefaultCopula` object, see `creditDefaultCopula`.

## Examples

### Compute Individual Losses for Each Counterparty

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditDefaultCopula` object with a two-factor model.

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```
Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
VaRLevel: 0.9500
PortfolioLosses: []
```

Set the `VarLevel` to 99%.

```
cdc.VarLevel = 0.99;
```

Use the `simulate` function before running `getScenarios`. Use the `getScenarios` function with the `creditDefaultCopula` object to generate the `scenarios` matrix.

```
cdc = simulate(cdc,1e5);
scenarios = getScenarios(cdc,[2,3]);
% expected loss for each scenario
mean(scenarios)
```

```
ans =
```

```
0.1382    1.1461
```

- “Credit Simulation Using Copulas” on page 4-2
- “Modeling Correlated Defaults with Copulas” on page 4-21

## Input Arguments

**cdc** — **creditDefaultCopula** object

object

`creditDefaultCopula` object obtained after running the `simulate` function.

For more information on `creditDefaultCopula` objects, see `creditDefaultCopula`.

**scenarioIndices** — **Specifies which scenarios are returned**

vector

Specifies which scenarios are returned, entered as a vector.

## Output Arguments

**scenarios** — **Counterparty losses**

matrix

Counterparty losses, returned as NumCounterparties-by-N matrix where N is the number of elements in scenarioIndices.

---

**Note:** If the number of scenarios requested is large, then the output matrix, scenarios, could be large and potentially limited by the available machine memory.

---

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

confidenceBands | creditDefaultCopula | creditMigrationCopula | portfolioRisk | riskContribution | simulate

### Topics

“Credit Simulation Using Copulas” on page 4-2

“Modeling Correlated Defaults with Copulas” on page 4-21

“Corporate Credit Risk” on page 1-3

“Credit Simulation Using Copulas” on page 4-2

**Introduced in R2017a**

# portfolioRisk

Generate portfolio-level risk measurements

## Syntax

```
[riskMeasures,confidenceIntervals] = portfolioRisk(cdc)
[riskMeasures,confidenceIntervals] = portfolioRisk(cdc,Name,Value)
```

## Description

`[riskMeasures,confidenceIntervals] = portfolioRisk(cdc)` returns tables of risk measurements for the portfolio losses. The `simulate` function must be run before `portfolioRisk` is used. For more information on using a `creditDefaultCopula` object, see `creditDefaultCopula`.

`[riskMeasures,confidenceIntervals] = portfolioRisk(cdc,Name,Value)` adds an optional name-value pair argument for `ConfidenceIntervalLevel`. The `simulate` function must be run before `portfolioRisk` is used.

## Examples

### Generate Tables for Risk Measure and Confidence Intervals for a `creditDefaultCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditDefaultCopula` object with a two-factor model.

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cdc =
```

creditDefaultCopula with properties:

```

Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
VaRLevel: 0.9500
PortfolioLosses: []

```

Set the VaRLevel to 99%.

```
cdc.VaRLevel = 0.99;
```

Use the `simulate` function before running `portfolioRisk`. Then use `portfolioRisk` with the `creditDefaultCopula` object to generate the `riskMeasure` and `ConfidenceIntervals` tables.

```
cdc = simulate(cdc,1e5);
[riskMeasure,confidenceIntervals] = portfolioRisk(cdc,'ConfidenceIntervalLevel',0.9)
```

riskMeasure =

1×4 table

EL	Std	VaR	CVaR
24.774	23.693	101.57	120.22

confidenceIntervals =

1×4 table

EL		Std	VaR		CVaR		
24.65	24.897	23.606	23.78	100.83	102.57	119.28	121.17

- “Credit Simulation Using Copulas” on page 4-2
- “creditDefaultCopula Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21

## Input Arguments

### **cdc** — creditDefaultCopula object

object

creditDefaultCopula object obtained after running the `simulate` function.

For more information on `creditDefaultCopula` objects, see `creditDefaultCopula`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( `' '`). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

```
Example: [riskMeasure, confidenceIntervals] =
portfolioRisk(cdc, 'ConfidenceIntervalLevel', 0.9)
```

### **'ConfidenceIntervalLevel'** — Confidence interval level

0.95 (default) | numeric between 0 and 1

Confidence interval level, specified as the comma-separated pair consisting of `'ConfidenceIntervalLevel'` and a numeric between 0 and 1. For example, if you specify `0.95`, a 95% confidence interval is reported in the output table (`riskMeasures`).

Data Types: `double`

## Output Arguments

### **riskMeasures** — Risk measures

table

Risk measures, returned as a table containing the following columns:

- `EL` — Expected loss, the mean of portfolio losses
- `Std` — Standard deviation of the losses
- `VaR` — Value at risk at the threshold specified by the `VaRLevel` property of the `creditDefaultCopula` object

- **CVaR** — Conditional VaR at the threshold specified by the `VarLevel` property of the `creditDefaultCopula` object

### **confidenceIntervals** — Confidence intervals

table

Confidence intervals, returned as a table of confidence intervals corresponding to the portfolio risk measures reported in the `riskMeasures` table. Confidence intervals are reported at the level specified by the `ConfidenceIntervalLevel` parameter.

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `riskContribution` | `simulate` | `table`

### Topics

“Credit Simulation Using Copulas” on page 4-2

“`creditDefaultCopula` Simulation Workflow” on page 4-6



“Modeling Correlated Defaults with Copulas” on page 4-21

“Corporate Credit Risk” on page 1-3

“Credit Simulation Using Copulas” on page 4-2

**Introduced in R2017a**

## riskContribution

Generate risk contributions for each counterparty in portfolio

### Syntax

```
Contributions = riskContribution(cdc)
```

### Description

`Contributions = riskContribution(cdc)` returns a table of risk contributions for each counterparty in the portfolio. The risk `Contributions` table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

The `simulate` function must be run before `riskContribution` is used. For more information on using a `creditDefaultCopula` object, see `creditDefaultCopula`.

### Examples

#### Determine the Risk Contribution for Each Counterparty for a `creditDefaultCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditDefaultCopula` object with a two-factor model.

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```
Portfolio: [100×5 table]  
FactorCorrelation: [2×2 double]  
VaRLevel: 0.9500  
PortfolioLosses: []
```

Set the `VarLevel` to 99%.

```
cdc.VarLevel = 0.99;
```

Use the `simulate` function before running `riskContribution`. Then use `riskContribution` with the `creditDefaultCopula` object to generate the risk Contributions table.

```
cdc = simulate(cdc,1e5);
Contributions = riskContribution(cdc);
Contributions(1:10,:)
```

```
ans =
```

```
10×3 table
```

ID	EL	CVaR
1	0.038604	0.12868
2	0.067068	0.24527
3	1.2527	2.3103
4	0.0023253	0.0026274
5	0.11766	0.26223
6	0.12437	0.47915
7	0.82913	1.6516
8	0.00085629	0.00089197
9	0.91406	4.009
10	0.24352	2.2781

- “Credit Simulation Using Copulas” on page 4-2
- “`creditDefaultCopula` Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21

## Input Arguments

**cdc** — `creditDefaultCopula` object  
object

`creditDefaultCopula` object obtained after running the `simulate` function.

For more information on `creditDefaultCopula` objects, see `creditDefaultCopula`.

## Output Arguments

### Contributions — Risk contributions

table

Risk contributions, returned as a table containing the following risk contributions for each counterparty:

- EL — Expected loss for the particular counterparty over the scenarios
- CVaR — Conditional value at risk for the particular counterparty over the scenarios

The risk `Contributions` table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Glasserman, P. “Measuring Marginal Risk Contributions in Credit Portfolios.” *Journal of Computational Finance*. Vol. 9, No. 2, Winter 2005/2006.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Kalkbrener, M., Lotter, H., and Overbeck, L. “Sensible and Efficient Capital Allocation for Credit Portfolios.” *Risk*. 17, 2004, pp. S19–S24.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

[confidenceBands](#) | [creditDefaultCopula](#) | [creditMigrationCopula](#) | [getScenarios](#) | [portfolioRisk](#) | [simulate](#) | [table](#)

### Topics

- “Credit Simulation Using Copulas” on page 4-2
- “[creditDefaultCopula](#) Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21
- “Corporate Credit Risk” on page 1-3
- “Credit Simulation Using Copulas” on page 4-2

**Introduced in R2017a**

## simulate

Simulate credit defaults using a `creditDefaultCopula` object

### Syntax

```
cdc = simulate(cdc,NumScenarios)
cdc = simulate( ___,Name,Value)
```

### Description

`cdc = simulate(cdc,NumScenarios)` performs the full simulation of credit scenarios and computes defaults and losses for the portfolio defined in the `creditDefaultCopula` object.

For more information on using a `creditDefaultCopula` object, see `creditDefaultCopula`.

`cdc = simulate( ___,Name,Value)` adds optional name-value pair arguments for (Copula, DegreesOfFreedom, and BlockSize).

### Examples

#### Run a Simulation Using a `creditDefaultCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditDefaultCopula` object with a two-factor model.

```
cdc = creditDefaultCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```

        Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
        VaRLevel: 0.9500
PortfolioLosses: []

```

Set the `VaRLevel` to 99%.

```
cdc.VaRLevel = 0.99;
```

Use the `simulate` function with the `creditDefaultCopula` object. After using `simulate`, you can then use the `portfolioRisk`, `riskContribution`, `confidenceBands`, and `getScenarios` functions with the updated `creditDefaultCopula` object.

```
cdc = simulate(cdc,1e5)
```

```
cdc =
```

```
creditDefaultCopula with properties:
```

```

        Portfolio: [100×5 table]
FactorCorrelation: [2×2 double]
        VaRLevel: 0.9900
PortfolioLosses: [1×100000 double]

```

You can use `riskContribution` with the `creditDefaultCopula` object to generate the risk Contributions table.

```
Contributions = riskContribution(cdc);
Contributions(1:10,:)
```

```
ans =
```

```
10×3 table
```

ID	EL	CVaR
1	0.038604	0.12868
2	0.067068	0.24527
3	1.2527	2.3103

4	0.0023253	0.0026274
5	0.11766	0.26223
6	0.12437	0.47915
7	0.82913	1.6516
8	0.00085629	0.00089197
9	0.91406	4.009
10	0.24352	2.2781

- “Credit Simulation Using Copulas” on page 4-2
- “creditDefaultCopula Simulation Workflow” on page 4-6
- “Modeling Correlated Defaults with Copulas” on page 4-21

## Input Arguments

### **cdc** — creditDefaultCopula object

object

creditDefaultCopula object, obtained by running the creditDefaultCopula constructor.

For more information on a creditDefaultCopula object, see creditDefaultCopula.

### **NumScenarios** — Number of scenarios to simulate

nonnegative integer

Number of scenarios to simulate, specified as a nonnegative integer. Scenarios are processed in blocks to conserve machine resources.

Data Types: double

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, ..., **NameN**, **ValueN**.

Example: `cdc = simulate(cdc, NumScenarios, 'Copula', 't', 'DegreesOfFreedom', 5)`



**'Copula' — Type of copula**

'Gaussian' (default) | character vector or string with values 'Gaussian' or 't'

Type of copula, specified as the comma-separated pair consisting of 'Copula' and a character vector or string. Possible values are:

- 'Gaussian' — A Gaussian copula
- 't' — A  $t$  copula with degrees of freedom specified using DegreesOfFreedom.

Data Types: char | string

**'DegreesOfFreedom' — Degrees of freedom for t copula**

5 (default) | nonnegative numeric value

Degrees of freedom for a  $t$  copula, specified as the comma-separated pair consisting of 'DegreesOfFreedom' and a nonnegative numeric value. If Copula is set to 'Gaussian', the DegreesOfFreedom parameter is ignored.

Data Types: double

**'BlockSize' — Number of scenarios to process in each iteration**

nonnegative numeric value

Number of scenarios to process in each iteration, specified as the comma-separated pair consisting of 'BlockSize' and a nonnegative numeric value.

If unspecified, BlockSize defaults to a value of approximately 1,000,000 / (Number-of-counterparties). For example, if there are 100 counterparties, the default BlockSize is 10,000 scenarios.

Data Types: double

## Output Arguments

**cdc — Updated creditDefaultCopula object**

object

Updated creditDefaultCopula object. The object is populated with the simulated PortfolioLosses.

For more information on a creditDefaultCopula object, see creditDefaultCopula.

---

**Note:** In the `simulate` function, the `Weights` (specified when using the `creditDefaultCopula` constructor) are transformed to ensure that the latent variables have a mean of 0 and a variance of 1.

---

### References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

### See Also

#### See Also

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `table`

### Topics

“Credit Simulation Using Copulas” on page 4-2

“`creditDefaultCopula` Simulation Workflow” on page 4-6

“Modeling Correlated Defaults with Copulas” on page 4-21

“Corporate Credit Risk” on page 1-3

“Credit Simulation Using Copulas” on page 4-2

### Introduced in R2017a

# creditMigrationCopula

Simulate and analyze multifactor credit migration rating model

## Description

The `creditMigrationCopula` takes as input a portfolio of credit-sensitive positions with a set of counterparties and performs a copula-based, multifactor simulation of credit rating migrations. Counterparty credit rating migrations and subsequent changes in portfolio value are calculated for each scenario and several risk measurements are reported.

`creditMigrationCopula` associates each counterparty with a random variable, called a latent variable, which is mapped to credit ratings based on a rating transition matrix. For each scenario, the value of the position with each counterparty is recomputed based on the realized credit rating of the counterparty. These latent variables are simulated by using a multifactor model, where systemic credit fluctuations are modeled with a series of risk factors. These factors can be based on industry sectors (such as financial or aerospace), geographical regions (such as USA or Eurozone), or any other underlying driver of credit risk. Each counterparty is assigned a series of weights which determine their sensitivity to each underlying credit factors.

The inputs to the model are:

- `migrationValues` — Values of the counterparty positions for each credit rating.
- `ratings` — Current credit rating for each counterparty.
- `transitionMatrix` — Matrix of credit rating transition probabilities.
- `LGD` — Loss given default ( $1 - Recovery$ ).
- `Weights` — Factor and idiosyncratic model weights

After you create `creditMigrationCopula` object by using the `creditMigrationCopula` function, use the `simulate` function to simulate credit migration by using the multifactor model. Then, for detailed reports, use the following functions: `portfolioRisk`, `riskContribution`, `confidenceBands`, and `getScenarios`.

## Create Object

To create a `creditMigrationCopula` object, use the `creditMigrationCopula` function.

## Properties

### Portfolio — Details of credit portfolio

table

Details of credit portfolio, specified as a MATLAB table that contains all the portfolio data that was passed as input into the `creditMigrationCopula` constructor.

The `Portfolio` table has a column for each of the constructor inputs (`MigrationValues`, `Rating`, `LGD`, `Weights`, and `ID`). Each row of the table represents one counterparty.

For example:

ID	MigrationValues	Rating	LGD	Weights	
1	[1x8 double]	"A"	0.6509	0.5	0.5
2	[1x8 double]	"BBB"	0.8283	0.55	0.45
3	[1x8 double]	"AA"	0.6041	0.7	0.3
4	[1x8 double]	"BB"	0.6509	0.55	0.45
5	[1x8 double]	"BBB"	0.4966	0.75	0.25

Data Types: table

### FactorCorrelation — Correlation matrix for credit factors

matrix

Correlation matrix for credit factors, specified as a `NumFactors`-by-`NumFactors` matrix. Specify the correlation matrix by using the optional name-value pair argument `'FactorCorrelation'` with the `creditMigrationCopula` constructor.

Data Types: double

### RatingLabels — Set of all possible credit ratings

cell array of character vectors, string, or numeric vector representing set of credit ratings

Set of all possible credit ratings, specified using an optional name-value input argument for 'RatingLabels' with the `creditMigrationCopula` constructor.

Data Types: `double` | `cell` | `string`

### **TransitionMatrix** — Probabilities counterparty transitions from starting credit rating to final credit rating

matrix

Probabilities that a counterparty transitions from a starting credit rating to a final credit rating, specified using the input argument 'transitionMatrix' with the `creditMigrationCopula` constructor. The rows represent the starting credit ratings and the columns represent the final ratings. The top row corresponds to the highest rating.

The top row holds the probabilities for a counterparty that starts at the highest rating (such as AAA) and the bottom row holds those for a counterparty starting in the default state. The bottom row may be omitted, indicating that a counterparty in default will remain in default. Each row must sum to 1.

The order of rows and columns must match the order of credit ratings defined in the `RatingLabels` parameter. The last column holds the probability of default for each of the ratings. If `RatingLabels` are unspecified, the default rating labels are: "AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D".

Data Types: `double`

### **VaRLevel** — Value at Risk Level

numeric value between 0 and 1

Value at risk level used when reporting VaR and CVaR, specified using an optional name-value pair argument 'VaRLevel' with the `creditMigrationCopula` constructor.

Data Types: `double`

### **PortfolioValues** — Portfolio values

vector

Portfolio values, specified as a 1-by-`NumScenarios` vector. After construction with the `creditMigrationCopula` constructor, the `PortfolioValues` property is empty. After you invoke the `simulate` function, `PortfolioValues` is populated with the portfolio values over each scenario.

Data Types: double

## Object Functions

creditMigrationCopula  
simulate

Create creditMigrationCopula object  
Simulate credit migrations using  
creditMigrationCopula object

portfolioRisk  
riskContribution

Generate portfolio-level risk measurements  
Generate risk contributions for each  
counterparty in portfolio

confidenceBands  
getScenarios

Confidence interval bands  
Counterparty scenarios

## Examples

### Create a creditMigrationCopula Object Using a Four-Factor Model

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a creditMigrationCopula object with a four-factor model using the creditMigrationCopula constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...  
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =
```

```
creditMigrationCopula with properties:
```

```
Portfolio: [250×5 table]  
FactorCorrelation: [4×4 double]  
RatingLabels: [8×1 string]  
TransitionMatrix: [8×8 double]
```

```

    VaRLevel: 0.9500
PortfolioValues: []

```

Set the `VaRLevel` to 99%.

```
cmc.VaRLevel = 0.99;
```

Use the `simulate` function to simulate 100,000 scenarios, and then view portfolio risk measures using the `portfolioRisk` function.

```
cmc = simulate(cmc,1e5)
portRisk = portfolioRisk(cmc)
```

```
cmc =
```

```
creditMigrationCopula with properties:
```

```

    Portfolio: [250×5 table]
FactorCorrelation: [4×4 double]
    RatingLabels: [8×1 string]
TransitionMatrix: [8×8 double]
    VaRLevel: 0.9900
PortfolioValues: [1×100000 double]

```

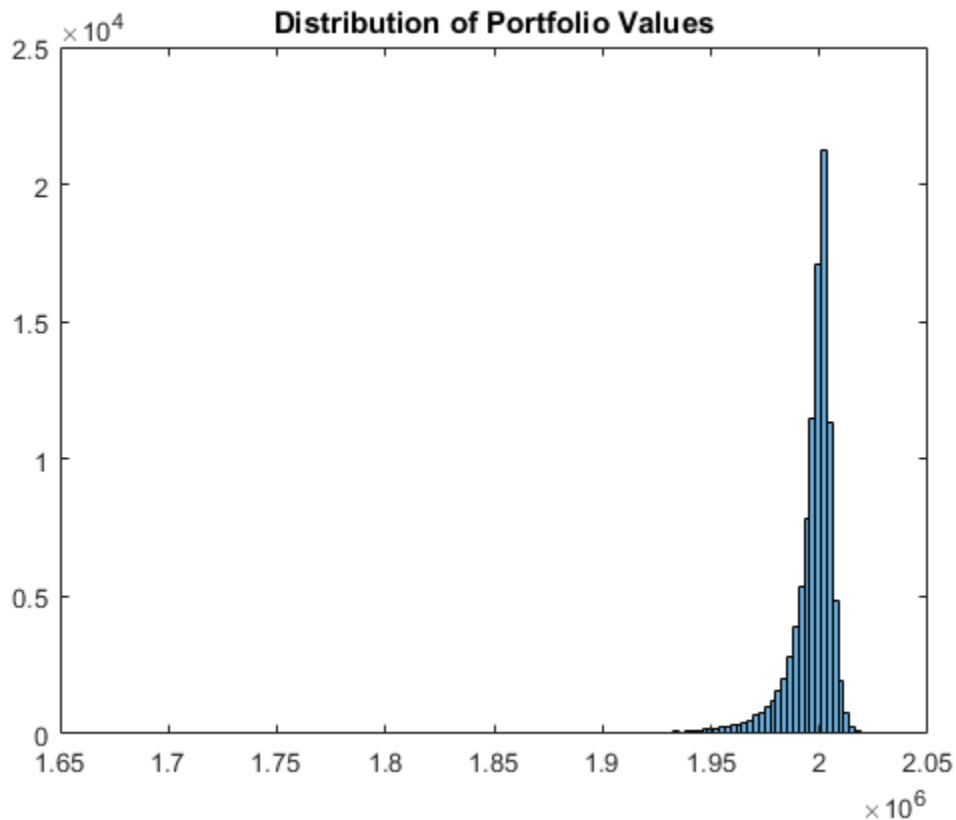
```
portRisk =
```

```
1×4 table
```

EL	Std	VaR	CVaR
4573.9	13039	56515	84463

View a histogram of the portfolio values.

```
h = histogram(cmc.PortfolioValues,125);
title('Distribution of Portfolio Values');
```



- “creditMigrationCopula Simulation Workflow” on page 4-12

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.



Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

### Topics

“creditMigrationCopula Simulation Workflow” on page 4-12

“Credit Rating Migration Risk” on page 1-8

**Introduced in R2017a**

## creditMigrationCopula

Create `creditMigrationCopula` object

### Syntax

```
cmc = creditMigrationCopula(migrationValues,ratings,  
transitionMatrix,LGD,Weights)  
cmc = creditMigrationCopula( ____,Name,Value)
```

### Description

`cmc = creditMigrationCopula(migrationValues,ratings,transitionMatrix,LGD,Weights)` creates a `creditMigrationCopula` object. For more information on using a `creditMigrationCopula` object, see `creditMigrationCopula`.

`cmc = creditMigrationCopula( ____,Name,Value)` adds optional name-value pair arguments.

### Examples

#### Create a creditMigrationCopula Object and Analyze Results

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a `creditMigrationCopula` object with a four-factor model using the `creditMigrationCopula` constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...
```

```
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =
```

```
creditMigrationCopula with properties:
```

```
    Portfolio: [250×5 table]
FactorCorrelation: [4×4 double]
    RatingLabels: [8×1 string]
TransitionMatrix: [8×8 double]
    VaRLevel: 0.9500
PortfolioValues: []
```

Set the `VaRLevel` to 99%.

```
cmc.VaRLevel = 0.99;
```

Use the `simulate` function to simulate 100,000 scenarios, and then view portfolio risk measures by using the `portfolioRisk` function.

```
cmc = simulate(cmc,1e5)
portRisk = portfolioRisk(cmc)
```

```
cmc =
```

```
creditMigrationCopula with properties:
```

```
    Portfolio: [250×5 table]
FactorCorrelation: [4×4 double]
    RatingLabels: [8×1 string]
TransitionMatrix: [8×8 double]
    VaRLevel: 0.9900
PortfolioValues: [1×100000 double]
```

```
portRisk =
```

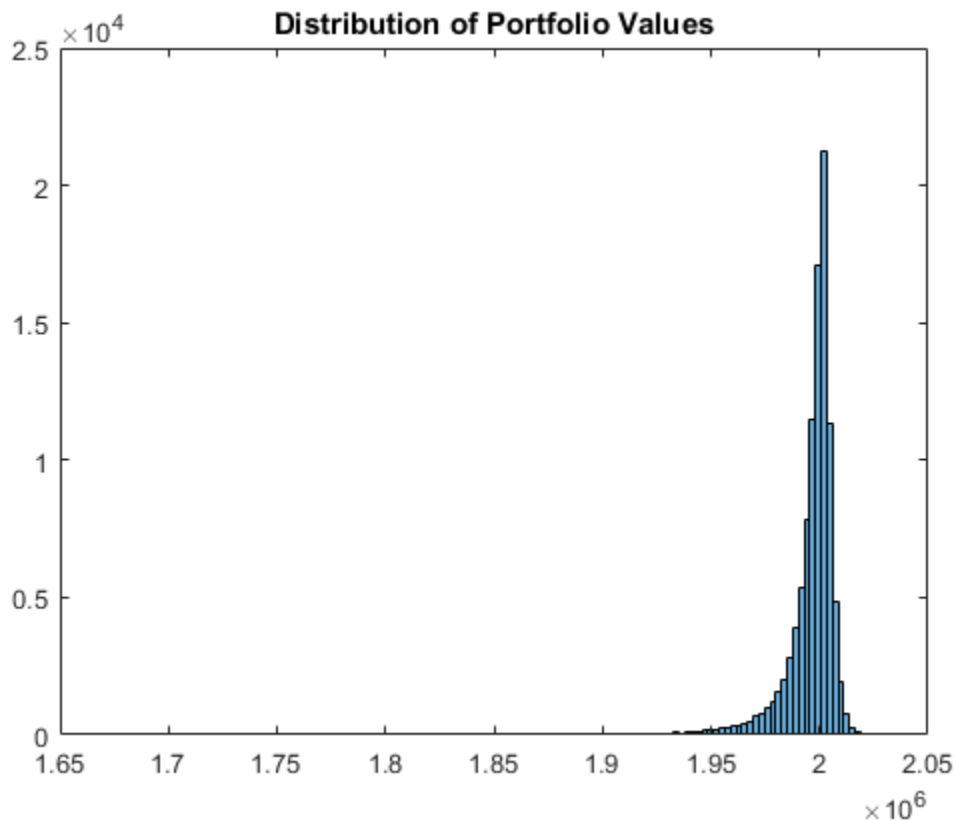
```
1×4 table
```

EL	Std	VaR	CVaR
_____	_____	_____	_____

```
4573.9    13039    56515    84463
```

View a histogram of the portfolio values.

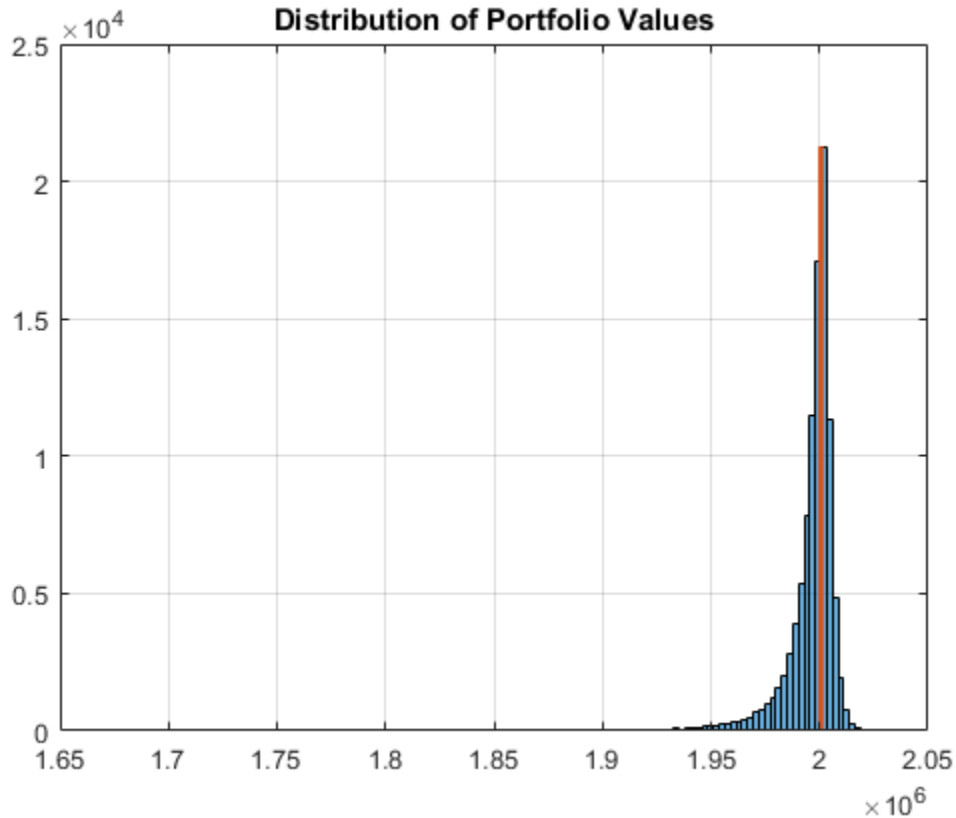
```
h = histogram(cmc.PortfolioValues,125);
title('Distribution of Portfolio Values');
```



Overlay the value that the portfolio takes if all counterparties maintained their current credit ratings.

```
CurrentRatingValue = portRisk.EL + mean(cmc.PortfolioValues);
hold on
plot([CurrentRatingValue CurrentRatingValue],[0 max(h.Values)],...
     'LineWidth',2);
```

grid on



- “creditMigrationCopula Simulation Workflow” on page 4-12

## Input Arguments

**migrationValues** — Values of counterparty positions for each credit rating  
matrix

Values of the counterparty positions for each credit rating, specified as a NumCounterparties-by-NumRatings matrix. Each row holds the possible values of the counterparty position for each credit rating. The last rating must be the default rating. The value for this rating is pre-recovery.

---

**Note:** The `creditMigrationCopula` model simulates the changes in portfolio value over a fixed time period (for example, one year). The `migrationValues` and `transitionMatrix` must be specific to a particular time period.

---

Data Types: `double`

**ratings** — Current credit rating for each counterparty

cell array of character vectors | numeric value | string

Current credit rating for each counterparty, specified as a `NumCounterparties-by-1` vector that represents the initial credit states. The set of all valid credit ratings and their order is defined by using the optional `RatingLabels` parameter. If `RatingLabels` are unspecified, the default rating labels are: "AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D".

Data Types: `double` | `string` | `cell`

**transitionMatrix** — Credit rating transition probabilities

numeric value

Credit rating transition probabilities, specified as a `NumRatings-by-NumRatings` matrix. The matrix contains the probabilities that a counterparty starting at a particular credit rating transitions to every other rating over some fixed time period. Each row holds all the transition probabilities for a particular starting credit rating.

The top row holds the probabilities for a counterparty that starts at the highest rating (such as AAA). The bottom row holds the probabilities for a counterparty starting in the default state. The bottom row may be omitted, indicating that a counterparty in default will remain in default. Each row must sum to 1.

The order of rows and columns must match the order of credit ratings defined in the `RatingLabels` parameter. The last column holds the probability of default for each of the ratings. If `RatingLabels` are unspecified, the default rating labels are: "AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D".

---

**Note:** The `creditMigrationCopula` model simulates the changes in portfolio value over a fixed time period (for example, one year). The `migrationValues` and `transitionMatrix` must be specific to a particular time period.

---

Data Types: `double`

**LGD — Loss given default**

numeric vector with elements from 0 through 1

Loss given default, specified as a `NumCounterparties`-by-1 numeric vector with elements from 0 through 1, representing the fraction of exposure that is lost when a counterparty defaults. LGD is defined as  $(1 - Recovery)$ . For example, an LGD of 0.6 implies a 40% recovery rate in the event of a default.

Data Types: double

**Weights — Factor and idiosyncratic weights**

array of factor and idiosyncratic weights

Factor and idiosyncratic weights, specified as a `NumCounterparties`-by- $(\text{NumFactors} + 1)$  array. Each row contains the factor weights for a particular counterparty. Each column contains the weights for an underlying risk factor. The last column in `Weights` contains the idiosyncratic risk weight for each counterparty. The idiosyncratic weight represents the company-specific credit risk. The total of the weights for each counterparty (that is, each row) must sum to 1.

For example, if a counterparty's creditworthiness was composed of 60% US, 20% European, and 20% idiosyncratic, then the `Weights` vector is `[0.6 0.2 0.2]`.

Data Types: double

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `cmc =`

```
creditMigrationCopula(migrationValues, ratings, transitionMatrix, LGD, Weights, 'Va
```

**'ID' — User-defined IDs for counterparties**

`1:NumCounterparties` (default) | vector

User-defined IDs for counterparties, specified as the comma-separated pair consisting of `'ID'` and a `NumCounterparties`-by-1 vector of IDs for each counterparty. `ID` is used to identify exposures in the `Portfolio` table and the risk contribution table. `ID` must be a numeric, a string array, or a cell array of character vectors. If unspecified, `ID` defaults to a numeric vector (`1:NumCounterparties`).

Data Types: double | string | cell

### 'VaRLevel' — Value at risk level

0.95 (default) | numeric between 0 and 1

Value at risk level (used for reporting VaR and CVaR), specified as the comma-separated pair consisting of 'VaRLevel' and a numeric between 0 and 1.

Data Types: double

### 'FactorCorrelation' — Factor correlation matrix

identity matrix (default) | correlation matrix

Factor correlation matrix, specified as the comma-separated pair consisting of 'FactorCorrelation' and a NumFactors-by-NumFactors matrix that defines the correlation between the risk factors. If not specified, the factor correlation matrix defaults to an identity matrix, meaning that the factors are not correlated.

Data Types: double

### 'RatingLabels' — Set of all possible credit ratings

["AAA", "AA", "A", "BBB", "BB", "B", "CCC", "D"] (default) | cell array of character vectors | numeric | string

Set of all possible credit ratings, specified as the comma-separated pair consisting of 'RatingLabels' and a NumRatings-by-1 vector, where the first element is the highest credit rating and the last element is the default state.

Data Types: cell | double | string

## Output Arguments

### cmc — creditMigrationCopula object

object

creditMigrationCopula object, returned as an object that contains the provided portfolio data that is stored in the Portfolio property.

The Portfolio property is a table that has a column for each of the constructor inputs (migrationValues, ratings, LGD, Weights, and ID). Each row of the Portfolio table represents one counterparty.



The `creditMigrationCopula` object has the following properties:

- **Portfolio:**

A table with the following variables:

- **ID** — ID to identify each counterparty
- **migrationValues** — Values of counterparty positions for each credit rating
- **ratings** — Current credit rating for each counterparty
- **LGD** — Loss given default
- **Weights** — Factor and idiosyncratic weights for counterparties

- **FactorCorrelation:**

Factor correlation matrix, a `NumFactors`-by-`NumFactors` matrix that defines the correlation between the risk factors.

- **RatingLabels:**

The set of all possible credit ratings.

- **TransitionMatrix:**

The matrix of probabilities that a counterparty transitions from a starting credit rating to a final credit rating. The rows represent the starting credit ratings and the columns represent the final ratings. The top row holds the probabilities for a counterparty that starts at the highest rating (for example `AAA`) and the bottom row holds those for a counterparty starting in the default state. The bottom row may be omitted, indicating that a counterparty in default will remain in default. Each row must sum to 1. The order of rows and columns must match the order of credit ratings defined in the `RatingLabels` parameter. The last column holds the probability of default for each of the ratings. If unspecified, the default rating labels are: `"AAA"`, `"AA"`, `"A"`, `"BBB"`, `"BB"`, `"B"`, `"CCC"`, `"D"`.

- **VaRLevel:**

The value-at-risk level, used when reporting VaR and CVaR.

- **PortfolioValues**

A `NumScenarios`-by-1 vector of portfolio values. This property is empty until you use the `simulate` function.

For more information on `creditMigrationCopula` objects, see `creditMigrationCopula`.

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditDefaultCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

### Topics

“`creditMigrationCopula` Simulation Workflow” on page 4-12

“Credit Rating Migration Risk” on page 1-8

`creditMigrationCopula`

### Introduced in R2017a

# confidenceBands

Confidence interval bands

## Syntax

```
cbTable = confidenceBands(cmc)
cbTable = confidenceBands(cmc,Name,Value)
```

## Description

`cbTable = confidenceBands(cmc)` returns a table of the requested risk measure and its associated confidence bands. Use `confidenceBands` to investigate how the values of a risk measure and its associated confidence interval converge as the number of scenarios increases. Before you run the `confidenceBands` function, you must run the `simulate` function. For more information on using a `creditMigrationCopula` object, see `creditMigrationCopula`.

`cbTable = confidenceBands(cmc,Name,Value)` adds optional name-value pair arguments.

## Examples

### Generate a Table of the Associated Confidence Bands for a Requested Risk Measure for a `creditMigrationCopula` Object

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a `creditMigrationCopula` object with a four-factor model using the `creditMigrationCopula` constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =
```

```
creditMigrationCopula with properties:
```

```
Portfolio: [250x5 table]
FactorCorrelation: [4x4 double]
RatingLabels: [8x1 string]
TransitionMatrix: [8x8 double]
VaRLevel: 0.9500
PortfolioValues: []
```

Set the `VaRLevel` to 99%.

```
cmc.VaRLevel = 0.99;
```

Use the `simulate` function to simulate 100,000 scenarios, and then use the `confidenceBands` function to generate the `cbTable`.

```
cmc = simulate(cmc,1e5);
cbTable = confidenceBands(cmc,'RiskMeasure','Std','ConfidenceIntervalLevel',0.9,'NumPo
cbTable(1:10,:)
```

```
ans =
```

```
10x4 table
```

NumScenarios	Lower	Std	Upper
2000	12297	12616	12954
4000	12512	12741	12980
6000	12211	12395	12584
8000	12236	12395	12559
10000	12537	12683	12832
12000	12447	12579	12714
14000	12495	12618	12743
16000	12574	12689	12807
18000	12650	12759	12871
20000	12780	12885	12992

- “creditMigrationCopula Simulation Workflow” on page 4-12

## Input Arguments

### **cmc** — creditMigrationCopula object

object

creditMigrationCopula object obtained after running the `simulate` function.

For more information on `creditMigrationCopula` objects, see `creditMigrationCopula`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `cbTable =`

```
confidenceBands(cmc, 'RiskMeasure', 'Std', 'ConfidenceIntervalLevel', 0.9, 'NumPoint
```

### **'RiskMeasure'** — Risk measure to investigate

'CVar' (default) | character vector or string with values 'EL', 'Std', 'VaR', or 'CVar'

Risk measure to investigate, specified as the comma-separated pair consisting of 'RiskMeasure' and a character vector or string. Possible values are:

- 'EL' — Expected loss, the mean of portfolio losses
- 'Std' — Standard deviation of the losses
- 'VaR' — Value at risk at the threshold specified by the `VarLevel` property of the `creditMigrationCopula` object
- 'CVar' — Conditional VaR at the threshold specified by the `VarLevel` property of the `creditMigrationCopula` object

Data Types: `char` | `string`

### 'ConfidenceIntervalLevel' — Confidence interval level

0.95 (default) | numeric between 0 and 1

Confidence interval level, specified as the comma-separated pair consisting of 'ConfidenceIntervalLevel' and a numeric between 0 and 1. For example, if you specify 0.95, a 95% confidence interval is reported in the output table (`cbTable`).

Data Types: double

### 'NumPoints' — Number of scenario samples to report

100 (default) | nonnegative integer

Number of scenario samples to report, specified as the comma-separated pair consisting of 'NumPoints' and a nonnegative integer. The default is 100, meaning that confidence bands are reported at 100 evenly spaced points of increasing sample size ranging from 0 to the total number of simulated scenarios.

---

**Note:** `NumPoints` must be a numeric scalar greater than 1. `NumPoints` is typically much smaller than total number of scenarios simulated. You can use `confidenceBands` to obtain a qualitative idea of how fast a risk measure and its confidence interval are converging. Specifying a large value for `NumPoints` is not recommended and can potentially cause performance issues with `confidenceBands`.

---

Data Types: double

## Output Arguments

### `cbTable` — Requested risk measure and associated confidence bands

table

Requested risk measure and associated confidence bands at each of the `NumPoints` scenario sample sizes, returned as a table containing the following columns:

- `NumScenarios` — Number of scenarios at the sample point
- `Lower` — Lower confidence band
- *RiskMeasure* — Requested risk measure, where the column takes its name from whatever risk measure is requested with the optional input `RiskMeasure`
- `Upper` — Upper confidence band

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

### Topics

“`creditMigrationCopula` Simulation Workflow” on page 4-12

“Credit Rating Migration Risk” on page 1-8

### Introduced in R2017a

## getScenarios

Counterparty scenarios

### Syntax

```
scenarios = getScenarios(cmc,scenarioIndices)
```

### Description

`scenarios = getScenarios(cmc,scenarioIndices)` returns counterparty scenario details as a matrix of individual values for each counterparty for the scenarios requested in `scenarioIndices`.

Before you use the `getScenarios` function, you must run the `simulate` function. For more information on using a `creditMigrationCopula` object, see `creditMigrationCopula`.

### Examples

#### Compute Individual Values for Each Counterparty

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a `creditMigrationCopula` object with a four-factor model using the `creditMigrationCopula` constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...  
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =
```



creditMigrationCopula with properties:

```

    Portfolio: [250×5 table]
FactorCorrelation: [4×4 double]
    RatingLabels: [8×1 string]
TransitionMatrix: [8×8 double]
    VaRLevel: 0.9500
PortfolioValues: []

```

Set the `VaRLevel` to 99%.

```
cmc.VaRLevel = 0.99;
```

Use the `simulate` function to simulate 100,000 scenarios, and then use the `getScenarios` function to generate the `scenarios` matrix.

```

cmc = simulate(cmc,1e5);
scenarios = getScenarios(cmc,[2,3]);
scenarios(1:10,:)

```

ans =

```

1.0e+04 *
    1.3216    1.3216
    0.2924    0.2737
    0.9788    0.9681
    0.4697    0.1536
    1.0376    1.0376
    0.5795    0.5795
    0.5641    0.5138
    0.5039    0.4956
    0.3537    0.3537
    2.3492    2.3477

```

- “creditMigrationCopula Simulation Workflow” on page 4-12

## Input Arguments

**cmc** — `creditMigrationCopula` object  
object

creditMigrationCopula object obtained after running the `simulate` function.

For more information on `creditMigrationCopula` objects, see `creditMigrationCopula`.

### **scenarioIndices** — Specifies which scenarios are returned

vector

Specifies which scenarios are returned, entered as a vector.

## Output Arguments

### **scenarios** — Counterparty values

matrix

Counterparty values, returned as NumCounterparties-by-N matrix, where N is the number of elements in `scenarioIndices`.

---

**Note:** If the number of scenarios requested is very large, then the output matrix, `scenarios`, could be very large, and potentially limited by the available machine memory.

---

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `portfolioRisk` | `riskContribution` | `simulate`

### Topics

“creditMigrationCopula Simulation Workflow” on page 4-12

“Credit Rating Migration Risk” on page 1-8

**Introduced in R2017a**

## portfolioRisk

Generate portfolio-level risk measurements

### Syntax

```
[riskMeasures,confidenceIntervals] = portfolioRisk(cmc)
[riskMeasures,confidenceIntervals] = portfolioRisk(cmc,Name,Value)
```

### Description

`[riskMeasures,confidenceIntervals] = portfolioRisk(cmc)` returns tables of risk measurements for the portfolio losses. Before you use the `portfolioRisk` function, run the `simulate` function. For more information on using a `creditMigrationCopula` object, see `creditMigrationCopula`.

`[riskMeasures,confidenceIntervals] = portfolioRisk(cmc,Name,Value)` adds an optional name-value pair argument for `ConfidenceIntervalLevel`.

### Examples

#### Generate Tables for Risk Measure and Confidence Intervals for a `creditMigrationCopula` Object

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a `creditMigrationCopula` object with a four-factor model using the `creditMigrationCopula` constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =
    creditMigrationCopula with properties:
        Portfolio: [250x5 table]
        FactorCorrelation: [4x4 double]
        RatingLabels: [8x1 string]
        TransitionMatrix: [8x8 double]
        VaRLevel: 0.9500
        PortfolioValues: []
```

Set the `VaRLevel` to 99%.

```
cmc.VaRLevel = 0.99;
```

Use the `simulate` function to simulate 100,000 scenarios, and then use the `portfolioRisk` function to generate the `riskMeasure` and `ConfidenceIntervals` tables.

```
cmc = simulate(cmc,1e5);
[riskMeasure,confidenceIntervals] = portfolioRisk(cmc,'ConfidenceIntervalLevel',0.9)
```

```
riskMeasure =
```

```
1x4 table
```

EL	Std	VaR	CVaR
4573.9	13039	56515	84463

```
confidenceIntervals =
```

```
1x4 table
```

EL		Std		VaR		CVaR	
4506.1	4641.8	12991	13087	55358	57812	82803	86123

- “creditMigrationCopula Simulation Workflow” on page 4-12

## Input Arguments

**cmc** — **creditMigrationCopula** object  
object

creditMigrationCopula object obtained after running the `simulate` function.

For more information on `creditMigrationCopula` objects, see `creditMigrationCopula`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

```
Example: [riskMeasure, confidenceIntervals] =  
portfolioRisk(cmc, 'ConfidenceIntervalLevel', 0.9)
```

**'ConfidenceIntervalLevel'** — **Confidence interval level**  
0.95 (default) | numeric between 0 and 1

Confidence interval level, specified as the comma-separated pair consisting of `'ConfidenceIntervalLevel'` and a numeric between 0 and 1. For example, if you specify `0.95`, a 95% confidence interval is reported in the output table (`riskMeasures`).

Data Types: `double`

## Output Arguments

**riskMeasures** — **Risk measures**  
table

Risk measures, returned as a table containing the following columns:

- `EL` — Expected loss, the mean of portfolio losses
- `Std` — Standard deviation of the losses
- `VaR` — Value at risk at the threshold specified by the `VaRLevel` property of the `creditMigrationCopula` object

- **CVaR** — Conditional VaR at the threshold specified by the `VarLevel` property of the `creditMigrationCopula` object

### **confidenceIntervals** — Confidence intervals

table

Confidence intervals, returned as a table of confidence intervals corresponding to the portfolio risk measures reported in the `riskMeasures` table. Confidence intervals are reported at the level specified by the `ConfidenceIntervalLevel` parameter.

## **References**

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## **See Also**

### **See Also**

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `riskContribution` | `simulate` | `table`

### **Topics**

“`creditMigrationCopula` Simulation Workflow” on page 4-12

“Credit Rating Migration Risk” on page 1-8

**Introduced in R2017a**



# riskContribution

Generate risk contributions for each counterparty in portfolio

## Syntax

```
Contributions = riskContribution(cmc)
```

## Description

`Contributions = riskContribution(cmc)` returns a table of risk contributions for each counterparty in the portfolio. The risk `Contributions` table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

Before you use the `riskContribution` function, you must run the `simulate` function. For more information on using a `creditMigrationCopula` object, see `creditMigrationCopula`.

## Examples

### Determine the Risk Contribution for Each Counterparty for a `creditMigrationCopula` Object

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a `creditMigrationCopula` object with a four-factor model using the `creditMigrationCopula` constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...  
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =  
  
    creditMigrationCopula with properties:  
  
        Portfolio: [250×5 table]  
        FactorCorrelation: [4×4 double]  
        RatingLabels: [8×1 string]  
        TransitionMatrix: [8×8 double]  
        VaRLevel: 0.9500  
        PortfolioValues: []
```

Set the `VaRLevel` to 99%.

```
cmc.VaRLevel = 0.99;
```

Use the `simulate` function to simulate 100,000 scenarios, and then use the `riskContribution` function to generate the `Contributions` table.

```
cmc = simulate(cmc,1e5);  
Contributions = riskContribution(cmc);  
Contributions(1:10,:)
```

```
ans =
```

```
10×3 table
```

ID	EL	CVaR
1	16.397	254.12
2	9.1179	134.31
3	5.7873	236.84
4	6.4235	338.23
5	22.739	544.69
6	10.776	704.29
7	2.9046	551.4
8	12.152	265.97
9	2.1567	26.112
10	1.7495	15.933

- “creditMigrationCopula Simulation Workflow” on page 4-12

## Input Arguments

**cmc** — **creditMigrationCopula** object  
object

creditMigrationCopula object obtained after running the `simulate` function.

For more information on creditMigrationCopula objects, see creditMigrationCopula.

## Output Arguments

**Contributions** — **Risk contributions**  
table

Risk contributions, returned as a table containing the following risk contributions for each counterparty:

- **EL** — Expected loss for the particular counterparty over the scenarios
- **CVaR** — Conditional value at risk for the particular counterparty over the scenarios

The risk **Contributions** table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Glasserman, P. “Measuring Marginal Risk Contributions in Credit Portfolios.” *Journal of Computational Finance*. Vol. 9, No. 2, Winter 2005/2006.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Kalkbrenner, M., Lotter, H., and Overbeck, L. “Sensible and Efficient Capital Allocation for Credit Portfolios.” *Risk*. 17, 2004, pp. S19–S24.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

`confidenceBands` | `creditDefaultCopula` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `simulate` | `table`

### Topics

“`creditMigrationCopula` Simulation Workflow” on page 4-12

“Credit Rating Migration Risk” on page 1-8

### Introduced in R2017a

# simulate

Simulate credit migrations using `creditMigrationCopula` object

## Syntax

```
cmc = simulate(cmc,NumScenarios)
cmc = simulate( ____,Name,Value)
```

## Description

`cmc = simulate(cmc,NumScenarios)` performs the full simulation of credit scenarios and computes changes in value due to credit rating changes for the portfolio defined in the `creditMigrationCopula` object.

For more information on using a `creditMigrationCopula` object, see `creditMigrationCopula`.

`cmc = simulate( ____,Name,Value)` adds optional name-value pair arguments for (Copula, DegreesOfFreedom, and BlockSize).

## Examples

### Run a Simulation Using a `creditMigrationCopula` Object

Load the saved portfolio data.

```
load CreditMigrationData.mat;
```

Scale the bond prices for portfolio positions for each bond.

```
migrationValues = migrationPrices .* numBonds;
```

Create a `creditMigrationCopula` object with a four-factor model using the `creditMigrationCopula` constructor.

```
cmc = creditMigrationCopula(migrationValues,ratings,transMat,...
lgd,weights,'FactorCorrelation',factorCorr)
```

```
cmc =  
  
    creditMigrationCopula with properties:  
  
        Portfolio: [250×5 table]  
        FactorCorrelation: [4×4 double]  
        RatingLabels: [8×1 string]  
        TransitionMatrix: [8×8 double]  
        VaRLevel: 0.9500  
        PortfolioValues: []
```

Set the `VaRLevel` to 99%.

```
cmc.VaRLevel = 0.99;
```

Use the `simulate` function to simulate 100,000 scenarios. After using `simulate`, you can then use the `portfolioRisk`, `riskContribution`, `confidenceBands`, and `getScenarios` with the updated `creditMigrationCopula` object.

```
cmc = simulate(cmc,1e5)
```

```
cmc =  
  
    creditMigrationCopula with properties:  
  
        Portfolio: [250×5 table]  
        FactorCorrelation: [4×4 double]  
        RatingLabels: [8×1 string]  
        TransitionMatrix: [8×8 double]  
        VaRLevel: 0.9900  
        PortfolioValues: [1×100000 double]
```

You can use the `riskContribution` function with the `creditMigrationCopula` object to generate the risk Contributions table.

```
Contributions = riskContribution(cmc);  
Contributions(1:10,:)
```

```
ans =  
  
    10×3 table
```

ID	EL	CVaR
1	16.397	254.12
2	9.1179	134.31
3	5.7873	236.84
4	6.4235	338.23
5	22.739	544.69
6	10.776	704.29
7	2.9046	551.4
8	12.152	265.97
9	2.1567	26.112
10	1.7495	15.933

- “creditMigrationCopula Simulation Workflow” on page 4-12

## Input Arguments

### **cmc** — creditMigrationCopula object

object

creditMigrationCopula object, obtained by running the creditMigrationCopula constructor.

For more information on a creditMigrationCopula object, see creditMigrationCopula.

### **NumScenarios** — Number of scenarios to simulate

nonnegative integer

Number of scenarios to simulate, specified as a nonnegative integer. Scenarios are processed in blocks to conserve machine resources.

Data Types: double

## Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: `cmc = simulate(cmc, NumScenarios, 'Copula', 't', 'DegreesOfFreedom', 5, 'BlockSize', 1000)`

### 'Copula' — Type of copula

'Gaussian' (default) | character vector or string with values 'Gaussian' or 't'

Type of copula, specified as the comma-separated pair consisting of 'Copula' and a character vector or string. Possible values are:

- 'Gaussian' — Gaussian copula
- 't' —  $t$  copula with degrees of freedom specified by using `DegreesOfFreedom`.

Data Types: `char` | `string`

### 'DegreesOfFreedom' — Degrees of freedom for t copula

5 (default) | nonnegative numeric value

Degrees of freedom for a  $t$  copula, specified as the comma-separated pair consisting of 'DegreesOfFreedom' and a nonnegative numeric value. If `Copula` is set to 'Gaussian', the `DegreesOfFreedom` parameter is ignored.

Data Types: `double`

### 'BlockSize' — Number of scenarios to process in each iteration

nonnegative numeric value

Number of scenarios to process in each iteration, specified as the comma-separated pair consisting of 'BlockSize' and a nonnegative numeric value. Adjust `BlockSize` for performance, especially when executing large simulations.

If unspecified, `BlockSize` defaults to a value of approximately  $1,000,000 / (\text{Number-of-counterparties})$ . For example, if there are 100 counterparties, the default `BlockSize` is 10,000 scenarios.

Data Types: `double`

## Output Arguments

**cmc** — Updated `creditMigrationCopula` object  
object



creditMigrationCopula object, returned as an updated object that is populated with the simulated PortfolioValues.

For more information on a creditMigrationCopula object, see creditMigrationCopula.

---

**Note:** In the simulate function, the Weights (specified when using the creditMigrationCopula constructor) are transformed to ensure that the latent variables have a mean of 0 and a variance of 1.

---

## References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics – Technical Document*.” J. P. Morgan, New York, 1997.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

### See Also

confidenceBands | creditDefaultCopula | creditMigrationCopula | getScenarios | portfolioRisk | riskContribution | table

## Topics

“creditMigrationCopula Simulation Workflow” on page 4-12

“Credit Rating Migration Risk” on page 1-8

**Introduced in R2017a**

# mertonmodel

Estimates probability of default using Merton model

## Syntax

```
[PD,DD,A,Sa] = mertonmodel(Equity,EquityVol,Liability,Rate)
[PD,DD,A,Sa] = mertonmodel( ____,Name,Value)
```

## Description

[PD,DD,A,Sa] = mertonmodel(Equity,EquityVol,Liability,Rate) estimates the default probability of a firm by using the Merton model.

[PD,DD,A,Sa] = mertonmodel( \_\_\_\_,Name,Value) adds optional name-value pair arguments.

## Examples

### Compute the Probability of Default Using the Single-Point Approach to the Merton Model

Load the data from MertonData.mat.

```
load MertonData.mat
Equity      = MertonData.Equity;
EquityVol   = MertonData.EquityVol;
Liability   = MertonData.Liability;
Drift       = MertonData.Drift;
Rate        = MertonData.Rate;
MertonData
```

```
MertonData =
```

```
5×6 table array
```

ID	Equity	EquityVol	Liability	Rate	Drift
_____	_____	_____	_____	_____	_____

'Firm 1'	2.6406e+07	0.7103	4e+07	0.05	0.0306
'Firm 2'	2.6817e+07	0.3929	3.5e+07	0.05	0.03
'Firm 3'	3.977e+07	0.3121	3.5e+07	0.05	0.031
'Firm 4'	2.947e+07	0.4595	3.2e+07	0.05	0.0302
'Firm 5'	2.528e+07	0.6181	4e+07	0.05	0.0305

Compute the default probability using the single-point approach to the Merton model.

```
[PD,DD,A,Sa] = mertonmodel(Equity,EquityVol,Liability,Rate,'Drift',Drift)
```

PD =

```
0.0638
0.0008
0.0000
0.0026
0.0344
```

DD =

```
1.5237
3.1679
4.4298
2.7916
1.8196
```

A =

```
1.0e+07 *
6.4210
6.0109
7.3063
5.9906
6.3231
```

Sa =

```
0.3010
0.1753
```

0.1699  
0.2263  
0.2511

- “Comparison of the Merton Model Single-Point Approach to the Time-Series Approach” on page 4-42

## Input Arguments

### **Equity** — Current market value of firm’s equity

positive numeric value

Current market value of firm’s equity, specified as a positive value.

Data Types: double

### **EquityVol** — Volatility of firm's equity

positive numeric value

Volatility of the firm's equity, specified as a positive annualized standard deviation.

Data Types: double

### **Liability** — Liability threshold of firm

positive numeric value

Liability threshold of firm, specified as a positive value. The liability threshold is often referred to as the default point.

Data Types: double

### **Rate** — Annualized risk-free interest rate

numeric value

Annualized risk-free interest rate, specified as a numeric value.

Data Types: double

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single

quotes ( ' ' ). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `[PD,DD,A,Sa] = mertonmodel(Equity,EquityVol,Liability,Rate,'Maturity',4,'Drift',0.22)`

### **'Maturity' — Time to maturity corresponding to liability threshold**

1 year (default) | positive numeric value

Time to maturity corresponding to the liability threshold, specified as the comma-separated pair consisting of 'Maturity' and a positive value.

Data Types: double

### **'Drift' — Annualized drift rate**

risk-free interest rate defined in Rate (default) | numeric value

Annualized drift rate (expected rate of return of the firm's assets), specified as the comma-separated pair consisting of 'Drift' and a numeric value.

Data Types: double

### **'Tolerance' — Tolerance for convergence of the solver**

1e-6 (default) | positive scalar

Tolerance for convergence of the solver, specified as the comma-separated pair consisting of 'Tolerance' and a positive scalar value.

Data Types: double

### **'MaxIterations' — Maximum number of iterations allowed**

500 (default) | positive integer

Maximum number of iterations allowed, specified as the comma-separated pair consisting of 'MaxIterations' and a positive integer.

Data Types: double

## Output Arguments

### **PD — Probability of default of firm at maturity**

numeric value

Probability of default of the firm at maturity, returned as a numeric value.

**DD — Distance-to-default**

numeric value

Distance-to-default, defined as the number of standard deviations between the mean of the asset distribution at maturity and the liability threshold (default point), returned as a numeric value.

**A — Current value of firm's assets**

numeric value

Current value of firm's assets, returned as a numeric value.

**Sa — Annualized firm's asset volatility**

numeric value

Annualized firm's asset volatility, returned as a numeric value.

## Definitions

### Merton Model Using Single-Point Calibration

In the Merton model, the value of a company's equity is treated as a call option on its assets and the liability is taken as a strike price.

`mertonmodel` accepts inputs for the firm's equity, equity volatility, liability threshold, and risk-free interest rate. The `mertonmodel` function solves a 2-by-2 nonlinear system of equations whose unknowns are the firm's assets and asset volatility. You compute the probability of default and distance-to-default by using the formulae in “Algorithms” on page 5-129.

## Algorithms

Unlike the time series method (see `mertonByTimeSeries`), when using `mertonmodel`, the equity volatility ( $\sigma_E$ ) is provided. Given equity ( $E$ ), liability ( $L$ ), risk-free interest rate ( $r$ ), asset drift ( $\mu_A$ ), and maturity ( $T$ ), you use a 2-by-2 nonlinear system of equations. `mertonmodel` solves for the asset value ( $A$ ) and asset volatility ( $\sigma_A$ ) as follows:

$$E = AN(d_1) - Le^{-rT}N(d_2)$$

$$\sigma_E = \frac{A}{E} N(d_1) \sigma_A$$

where  $N$  is the cumulative normal distribution,  $d_1$  and  $d_2$  are defined as:

$$d_1 = \frac{\ln\left(\frac{A}{L}\right) + (r + 0.5\sigma_A^2)T}{\sigma_A\sqrt{T}}$$

$$d_2 = d_1 - \sigma_A\sqrt{T}$$

The formulae for the distance-to-default ( $DD$ ) and default probability ( $PD$ ) are:

$$DD = \frac{\ln\left(\frac{A}{L}\right) + (\mu_A - 0.5\sigma_A^2)T}{\sigma_A\sqrt{T}}$$

$$PD = 1 - N(DD)$$

## References

Zielinski, T. *Merton's and KMV Models In Credit Risk Management*.

Löffler, G. and Posch, P.N. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2011.

Kim, I.J., Byun, S.J, Hwang, S.Y. *An Iterative Method for Implementing Merton*.

Merton, R. C. "On the Pricing of Corporate Debt: The Risk Structure of Interest Rates." *Journal of Finance*. Vol. 29. pp. 449–470.

## See Also

### See Also

mertonByTimeSeries



## **Topics**

“Comparison of the Merton Model Single-Point Approach to the Time-Series Approach”  
on page 4-42

“Default Probability by Using the Merton Model for Structural Credit Risk” on page 1-11

**Introduced in R2017a**

## mertonByTimeSeries

Estimate default probability using time-series version of Merton model

### Syntax

```
[PD,DD,A,Sa] = mertonByTimeSeries(Equity,Liability,Rate)
[PD,DD,A,Sa] = mertonByTimeSeries( ____,Name,Value)
```

### Description

`[PD,DD,A,Sa] = mertonByTimeSeries(Equity,Liability,Rate)` estimates the default probability of a firm by using the Merton model.

`[PD,DD,A,Sa] = mertonByTimeSeries( ____,Name,Value)` adds optional name-value pair arguments.

### Examples

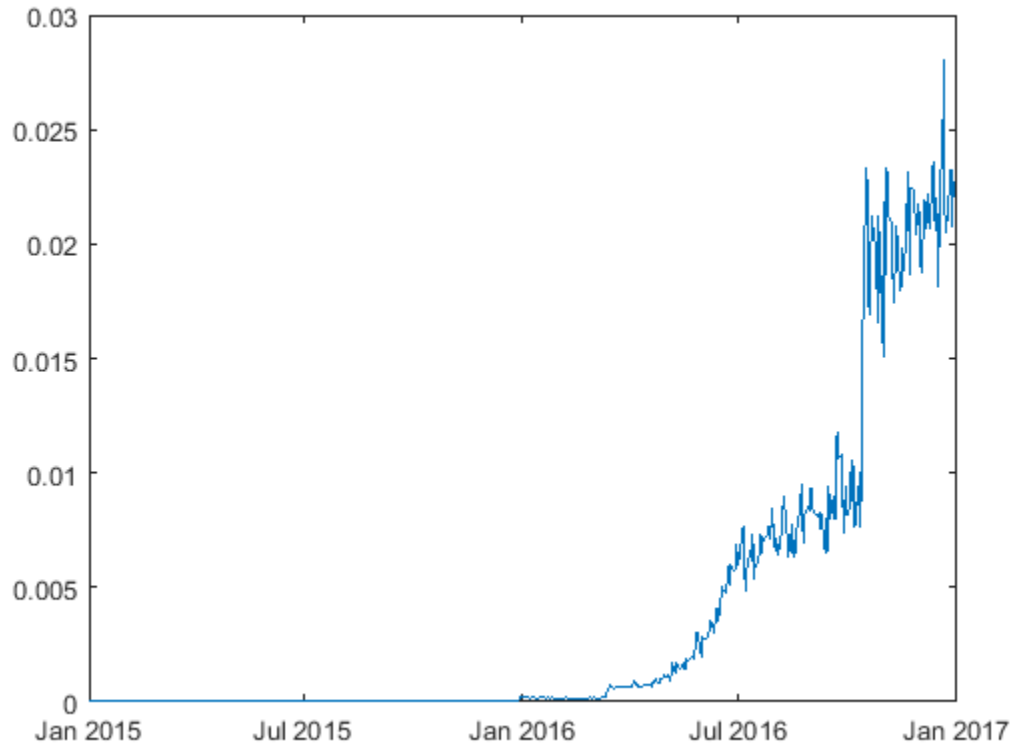
#### Compute Probability of Default Using the Time-Series Approach to the Merton Model

Load the data from `MertonData.mat`.

```
load MertonData.mat
Dates      = MertonDataTS.Dates;
Equity     = MertonDataTS.Equity;
Liability  = MertonDataTS.Liability;
Rate      = MertonDataTS.Rate;
```

Compute the default probability by using the time-series approach of Merton's model.

```
[PD,DD,A,Sa] = mertonByTimeSeries(Equity,Liability,Rate);
plot(Dates,PD)
```



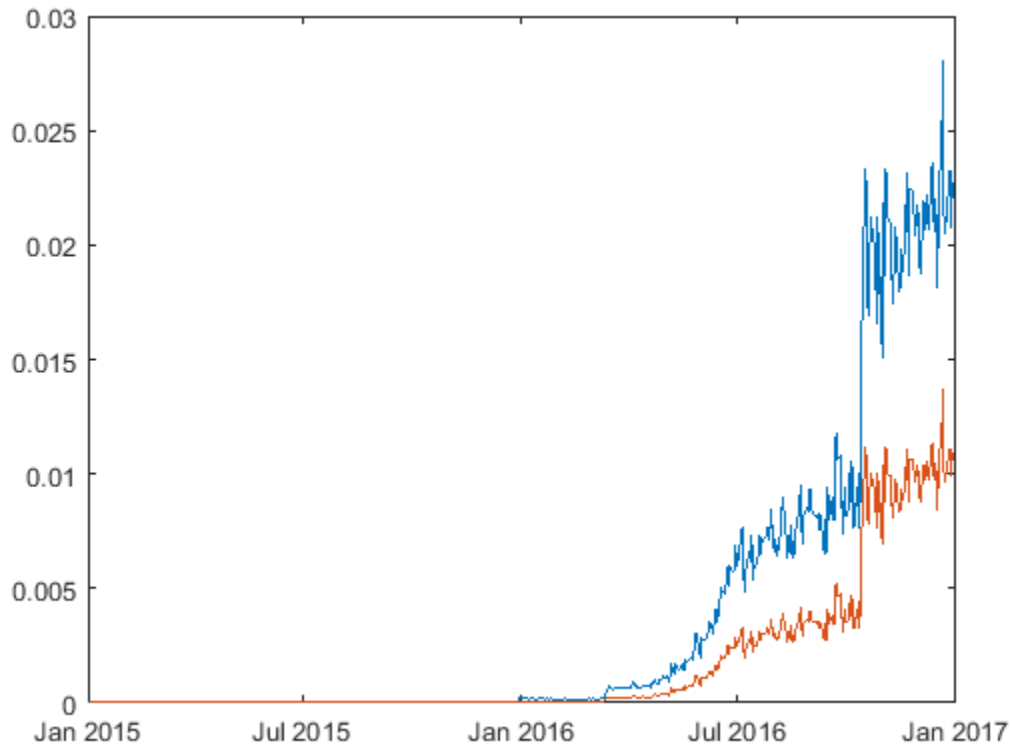
### Compute Probability of Default Using the Time-Series Approach to the Merton Model With Drift

Load the data from `MertonData.mat`.

```
load MertonData.mat
Dates      = MertonDataTS.Dates;
Equity     = MertonDataTS.Equity;
Liability  = MertonDataTS.Liability;
Rate      = MertonDataTS.Rate;
```

Compute the plot for the default probability values by using the time-series approach of Merton's model. You compute the PDO (blue line) by using the default values. You compute the PD1 (red line) by specifying an optional `Drift` value.

```
PD0 = mertonByTimeSeries(Equity, Liability, Rate);  
PD1 = mertonByTimeSeries(Equity, Liability, Rate, 'Drift', 0.10);  
plot(Dates, PD0, Dates, PD1)
```



- “Comparison of the Merton Model Single-Point Approach to the Time-Series Approach” on page 4-42

## Input Arguments

**Equity** — Market value of firm’s equity  
positive numeric value

Market value of the firm's equity, specified as a positive value.

Data Types: double

### **Liability** – Liability threshold of firm

positive numeric value

Liability threshold of the firm, specified as a positive value. The liability threshold is often referred to as the default point.

Data Types: double

### **Rate** – Annualized risk-free interest rate

numeric value

Annualized risk-free interest rate, specified as a numeric value.

Data Types: double

## **Name-Value Pair Arguments**

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, . . . , **NameN**, **ValueN**.

Example: [PD,DD,A,Sa] =

```
mertonByTimeSeries(Equity, Liability, Rate, 'Maturity', 4, 'Drift', 0.22, 'Tolerance'
```

### **'Maturity'** – Time to maturity corresponding to liability threshold

1 year (default) | positive numeric value

Time to maturity corresponding to the liability threshold, specified as the comma-separated pair consisting of 'Maturity' and a positive value.

Data Types: double

### **'Drift'** – Annualized drift rate

risk-free interest rate defined in **Rate** (default) | numeric value

Annualized drift rate, expected rate of return of the firm's assets, specified as the comma-separated pair consisting of 'Drift' and a numeric value.

Data Types: double

### 'NumPeriods' — Number of periods per year

250 trading days per year (default) | positive integer

Number of periods per year, specified as the comma-separated pair consisting of 'NumPeriods' and a positive integer. Typical values are 250 (yearly), 12 (monthly), or 4 (quarterly).

Data Types: double

### 'Tolerance' — Tolerance for convergence of the solver

1e-6 (default) | positive scalar

Tolerance for convergence of the solver, specified as the comma-separated pair consisting of 'Tolerance' and a positive scalar value.

Data Types: double

### 'MaxIterations' — Maximum number of iterations allowed

500 (default) | positive integer

Maximum number of iterations allowed, specified as the comma-separated pair consisting of 'MaxIterations' and a positive integer.

Data Types: double

## Output Arguments

### PD — Probability of default of firm at maturity

numeric value

Probability of default of the firm at maturity, returned as a numeric.

### DD — Distance-to-default

numeric value

Distance-to-default, defined as the number of standard deviations between the mean of the asset distribution at maturity and the liability threshold (default point), returned as a numeric.

### A — Value of firm's assets

numeric value

Value of firm's assets, returned as a numeric value.

**Sa — Annualized firm's asset volatility**

numeric value

Annualized firm's asset volatility, returned as a numeric value.

## Definitions

### Merton Model for Time Series

In the Merton model, the value of a company's equity is treated as a call option on its assets, and the liability is taken as a strike price.

Given a time series of observed equity values and liability thresholds for a company, `mertonByTimeSeries` calibrates corresponding asset values, the volatility of the assets in the sample's time span, and computes the probability of default for each observation. Unlike `mertonmodel`, no equity volatility input is required for the time-series version of the Merton model. You compute the probability of default and distance-to-default by using the formulae in “Algorithms” on page 5-137.

## Algorithms

Given the time series for equity ( $E$ ), liability ( $L$ ), risk-free interest rate ( $r$ ), asset drift ( $\mu A$ ), and maturity ( $T$ ), `mertonByTimeSeries` sets up the following system of nonlinear equations and solves for a time series asset values ( $A$ ), and a single asset volatility ( $\sigma_A$ ).

At each time period  $t$ , where  $t = 1 \dots n$ :

$$A_1 = \left( \frac{E_1 + L_1 e^{-r_1 T_1} N(d_2)}{N(d_1)} \right)$$

$$A_t = \left( \frac{E_t + L_t e^{-r_t T_t} N(d_2)}{N(d_1)} \right)$$

...

$$A_n = \left( \frac{E_n + L_n e^{-r_n T_n} N(d_2)}{N(d_1)} \right)$$

where  $N$  is the cumulative normal distribution. To simplify the notation, the time subscript is omitted for  $d_1$  and  $d_2$ . At each time period,  $d_1$ , and  $d_2$  are defined as:

$$d_1 = \frac{\ln\left(\frac{A}{L}\right) + (r + 0.5\sigma_A^2)T}{\sigma_A\sqrt{T}}$$

$$d_2 = d_1 - \sigma_A\sqrt{T}$$

The formulae for the distance-to-default ( $DD$ ) and default probability ( $PD$ ) at each time period are:

$$DD = \frac{\ln\left(\frac{A}{L}\right) + (\mu_A - 0.5\sigma_A^2)T}{\sigma_A\sqrt{T}}$$

$$PD = 1 - N(DD)$$

## References

Zielinski, T. *Merton's and KMV Models In Credit Risk Management*.

Loffler, G. and Posch, P.N. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2011.

Kim, I.J., Byun, S.J, Hwang, S.Y. *An Iterative Method for Implementing Merton*.

Merton, R. C. "On the Pricing of Corporate Debt: The Risk Structure of Interest Rates." *Journal of Finance*. Vol. 29. pp. 449–470.

## See Also

### See Also

mertonmodel



## **Topics**

“Comparison of the Merton Model Single-Point Approach to the Time-Series Approach”  
on page 4-42

“Default Probability by Using the Merton Model for Structural Credit Risk” on page 1-11

**Introduced in R2017a**

# varbacktest

Suite of value-at-risk (VaR) back tests

## Description

The general workflow is:

- 1 Load or generate the data for the VaR backtesting analysis.
- 2 Create a `varbacktest` object using the `varbacktest` constructor.
- 3 Use the `summary` function to generate a summary report for the given data on the number of observations and the number of failures.
- 4 Use the `runtests` function to run all tests at once.
- 5 For additional test details, run the following individual tests:
  - `t1` — Traffic light test
  - `bin` — Binomial test
  - `pof` — Proportion of failures
  - `tuff` — Time until first failure
  - `cc` — Conditional coverage mixed
  - `cci` — Conditional coverage independence
  - `tbf` — Time between failures mixed
  - `tbf i` — Time between failures independence

For more information, see “VaR Backtesting Workflow” on page 2-8.

## Create Object

To create a `varbacktest` (`vbt`) object, use the `varbacktest` function.

## Properties

**PortfolioData** — Portfolio data for VaR backtesting analysis

numeric array

Portfolio data for the VaR backtesting analysis, specified as a NumRows-by-1 numeric array containing a copy of the portfolio data.

Data Types: double

### **VaRData** — VaR data for VaR backtesting analysis

numeric array

VaR data for the VaR backtesting analysis, specified as a NumRows-by-NumVaRS numeric array containing a copy of the VaR data.

Data Types: double

### **PortfolioID** — Portfolio identifier

string

Portfolio identifier, specified as a string.

Data Types: string

### **VaRID** — VaR identifier

string array

VaR identifier, specified as a 1-by-NumVaRS string array containing the VaR IDs for the corresponding columns in VaRData.

Data Types: string

### **VaRLevel1** — VaR level

numeric array with values between 0 and 1

VaR level, specified as a 1-by-NumVaRS numeric array containing the VaR levels for the corresponding columns in VaRData.

Data Types: double

<b>varbacktest Property</b>	<b>Set or Modify Property from Command Line Using varbacktest Function</b>	<b>Modify Property Using Dot Notation</b>
PortfolioData	Yes	No
VaRData	Yes	No
PortfolioID	Yes	Yes

varbacktest Property	Set or Modify Property from Command Line Using varbacktest Function	Modify Property Using Dot Notation
VaRID	Yes	Yes
VaRLevel	Yes	Yes

## Object Functions

varbacktest	Create varbacktest object using portfolio outcomes data and corresponding value-at-risk (VaR) data
tl	Traffic light test for value-at-risk (VaR) backtesting
bin	Binomial test for value-at-risk (VaR) backtesting
pof	Proportion of failures test for value-at-risk (VaR) backtesting
tuff	Time until first failure test for value-at-risk (VaR) backtesting
cc	Conditional coverage mixed test for value-at-risk (VaR) backtesting
cci	Conditional coverage independence test for value-at-risk (VaR) backtesting
tbf	Time between failures mixed test for value-at-risk (VaR) backtesting
tbf <sub>i</sub>	Time between failures independence test for value-at-risk (VaR) backtesting
summary	Report on varbacktest data
runtests	Run all tests in varbacktest

## Examples

### Create varbacktest Object and Run VaR Backtests for Single VaR at 95%

The `varbacktest` constructor takes in portfolio outcomes data and corresponding value-at-risk (VaR) data and returns a `varbacktest` object.

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

`vbt`, the `varbacktest` object, contains a copy of the given portfolio data (`PortfolioData` property), the given VaR data (`VaRData` property) and all combinations of portfolio ID, VaR ID, and VaR level to be tested (`PortfolioID`, `VaRID`, and `VaRLevel` properties).

Run the tests using the `vbt` object.

```
runtests(vbt)
```

```
ans =
```

```
1×11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF	CC
"Portfolio"	"VaR"	0.95	green	accept	accept	accept	accept

Change the `PortfolioID` and `VaRID` properties using dot notation. For more information on creating a `varbacktest` object, see `varbacktest`.

```
vbt.PortfolioID = 'S&P'
vbt.VaRID = 'Normal at 95%'
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
  VaRData: [1043×1 double]
PortfolioID: "S&P"
  VaRID: "VaR"
  VaRLevel: 0.9500
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
  VaRData: [1043×1 double]
PortfolioID: "S&P"
  VaRID: "Normal at 95%"
  VaRLevel: 0.9500
```

Run all tests using the updated `varbacktest` object.

```
runtests(vbt)
```

```
ans =
```

```
1×11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"S&P"	"Normal at 95%"	0.95	green	accept	accept	accept

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## References

Basel Committee on Banking Supervision, *Supervisory Framework for the Use of 'Backtesting' in Conjunction with the Internal Models Approach to Market Risk Capital Requirements*. January, 1996, <http://www.bis.org/publ/bcbs22.htm>.

Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841–862.

Cogneau, Ph. "*Backtesting Value-at-Risk: How Good is the Model?*" Intelligent Risk, PRMIA, July, 2015.

Haas, M. "*New Methods in Backtesting*." Financial Engineering, Research Center Caesar, Bonn, 2001.

Jorion, Ph. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Kupiec, P. "Techniques for Verifying the Accuracy of Risk Management Models." *Journal of Derivatives*. Vol. 3, 1995, pp. 73–84.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management*. Princeton University Press, 2005.

Nieppola, O. "Backtesting Value-at-Risk Models." Master's Thesis, Helsinki School of Economics, 2009.

## See Also

### See Also

varbacktest

### Topics

"VaR Backtesting Workflow" on page 2-8

"Value-at-Risk Estimation and Backtesting" on page 2-13

"Overview of VaR Backtesting" on page 2-2

### Introduced in R2016b

## varbacktest

Create `varbacktest` object using portfolio outcomes data and corresponding value-at-risk (VaR) data

### Syntax

```
vbt = varbacktest(PortfolioData, VaRData)
vbt = varbacktest( ____, Name, Value)
```

### Description

`vbt = varbacktest(PortfolioData, VaRData)` creates a `varbacktest` object using portfolio outcomes data and corresponding value-at-risk (VaR) data. For more information on using a `vbt` object, see `varbacktest`.

`vbt = varbacktest( ____, Name, Value)` adds optional name-value pair arguments. For more information on using a `vbt` object, see `varbacktest`.

## Examples

### Run VaR Backtests for a Single VaR at 95%

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex, Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
```



```
VaRLevel: 0.9500
```

`vbt`, the `varbacktest` object, contains a copy of the given portfolio data (`PortfolioData` property), the given VaR data (`VaRData` property) and all combinations of portfolio ID, VaR ID, and VaR level to be tested (`PortfolioID`, `VaRID`, and `VaRLevel` properties).

Run the tests using the `varbacktest` object.

```
runtests(vbt)
```

```
ans =
```

```
1×11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF	CC
"Portfolio"	"VaR"	0.95	green	accept	accept	accept	accept

Change the `PortfolioID` and `VaRID` properties using dot notation. For more information on a `varbacktest` object, see `varbacktest`.

```
vbt.PortfolioID = 'S&P'
vbt.VaRID = 'Normal at 95%'
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "S&P"
VaRID: "VaR"
VaRLevel: 0.9500
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
  VaRData: [1043×1 double]
PortfolioID: "S&P"
  VaRID: "Normal at 95%"
  VaRLevel: 0.9500
```

Run all tests using the updated `varbacktest` object.

```
runtests(vbt)
```

```
ans =
```

```
1×11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"S&P"	"Normal at 95%"	0.95	green	accept	accept	accept

### Run VaR Backtests for Multiple VaRs at Different Confidence Levels

Create a `varbacktest` object that has multiple VaR identifiers with different confidence levels.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
  [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
  'PortfolioID', 'Equity',...
  'VaRID', {'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'},
  'VaRLevel', [0.95 0.99 0.95 0.99 0.95 0.99]);
```

Run the summary report for the `varbacktest` object.

```
summary(vbt)
```

```
ans =
```

```
6×10 table
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Fail
-------------	-------	----------	---------------	--------------	------

"Equity"	"Normal195"	0.95	0.94535	1043	57
"Equity"	"Normal199"	0.99	0.9837	1043	17
"Equity"	"Historical195"	0.95	0.94343	1043	59
"Equity"	"Historical199"	0.99	0.98849	1043	12
"Equity"	"EWMA95"	0.95	0.94343	1043	59
"Equity"	"EWMA99"	0.99	0.97891	1043	22

Run all tests using the `varbacktest` object.

```
runtests(vbt)
```

```
ans =
```

```
6x11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"Equity"	"Normal195"	0.95	green	accept	accept	accept
"Equity"	"Normal199"	0.99	yellow	reject	accept	accept
"Equity"	"Historical195"	0.95	green	accept	accept	accept
"Equity"	"Historical199"	0.99	green	accept	accept	accept
"Equity"	"EWMA95"	0.95	green	accept	accept	accept
"Equity"	"EWMA99"	0.99	yellow	reject	reject	accept

Run the traffic light test (`tl`) using the `varbacktest` object.

```
tl(vbt)
```

```
ans =
```

```
6x9 table
```

PortfolioID	VaRID	VaRLevel	TL	Probability	TypeI
"Equity"	"Normal195"	0.95	green	0.77913	0.26396
"Equity"	"Normal199"	0.99	yellow	0.97991	0.03686
"Equity"	"Historical195"	0.95	green	0.85155	0.18232
"Equity"	"Historical199"	0.99	green	0.74996	0.35269

```
"Equity"      "EWMA95"      0.95      green      0.85155      0.18232
"Equity"      "EWMA99"      0.99      yellow     0.99952      0.0011122
```

### Run VaR Backtests for Multiple Portfolios and Concatenate Results

Use the `varbacktest` constructor with table inputs and name-value pair arguments to create two `varbacktest` objects and run the concatenated summary report. The `varbacktest` constructor uses the variable names in the table inputs as `PortfolioID` and `VarID`.

```
load VaRBacktestData
vbtE = varbacktest(DataTable(:,2),DataTable(:,3:4), 'VaRLevel', [0.95 0.99]);
vbtD = varbacktest(DataTable(:,5),DataTable(:,6:7), 'VaRLevel', [0.95 0.99]);
[summary(vbtE); summary(vbtD)]
```

ans =

4×10 table

PortfolioID	VarID	VaRLevel	ObservedLevel	Observations
"Equity"	"VaREquity95"	0.95	0.94343	1043
"Equity"	"VaREquity99"	0.99	0.97891	1043
"Derivatives"	"VaRDerivatives95"	0.95	0.95014	1043
"Derivatives"	"VaRDerivatives99"	0.99	0.97028	1043

Run all the tests and concatenate the results.

```
[runtests(vbtE); runtests(vbtD)]
```

ans =

4×11 table

PortfolioID	VarID	VaRLevel	TL	Bin	POF	
"Equity"	"VaREquity95"	0.95	green	accept	accept	ac
"Equity"	"VaREquity99"	0.99	yellow	reject	reject	ac
"Derivatives"	"VaRDerivatives95"	0.95	green	accept	accept	ac

```
"Derivatives" "VaRDerivatives99" 0.99 red reject reject a
```

Run the pof test and concatenate the results.

```
[pof(vbtE); pof(vbtD)]
```

```
ans =
```

```
4×9 table
```

PortfolioID	VaRID	VaRLevel	POF	LRatioPOF	PValuePOF
"Equity"	"VaREquity95"	0.95	accept	0.91023	0.34
"Equity"	"VaREquity99"	0.99	reject	9.8298	0.0017
"Derivatives"	"VaRDerivatives95"	0.95	accept	0.00045457	0.98
"Derivatives"	"VaRDerivatives99"	0.99	reject	26.809	2.2457e

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

### PortfolioData — Portfolio outcomes data

NumRows-by-1 numeric array | NumRows-by-1 numeric columns table

Portfolio outcomes data, specified as a NumRows-by-1 numeric array or NumRows-by-1 table with a numeric column containing portfolio outcomes data. PortfolioData can be expressed as returns, or alternatively as profits and losses. There are no validations in the tool regarding the units of portfolio or VaR data. It is the user's responsibility to provide the portfolio and VaR data in the same units.

Missing values (NaNs in the PortfolioData) are discarded before applying the tests. Therefore, a different number of observations is reported (NumObs) for series with different number of missing values. The reported number of observations (NumObs) equals the original number of rows in the data (NumRows) minus the number of periods with missing values.

Data Types: double | table

### **VaRData — Value-at-risk (VaR) data**

NumRows-by-NumVaRs numeric array | NumRows-by-NumVaRs table with numeric columns

Value-at-risk (VaR) data, specified using a NumRows-by-NumVaRs numeric array, or NumRows-by-NumVaRs table with numeric columns.

VaRData data must be in the same units PortfolioData. There are no validations in the tool regarding the units of portfolio or VaR data. These may be expressed as returns, or alternatively as profits and losses. It is the user's responsibility to provide the portfolio and VaR data in the same units.

If VaRData has more than one column (NumVaRs > 1), the PortfolioData is tested against each column in VaRData. By default, a 0.95 VaR confidence level is used for all columns in VaRData. (Use VaRLevel to specify different VaR confidence levels.)

The convention is that VaR is a positive amount. Therefore, a failure is recorded when the loss (the negative of the portfolio data) exceeds the VaR, that is, when

$$-\text{PortfolioData} > \text{VaRData}$$

For example, a VaR of one million (positive) is violated whenever there is an outcome worse than a one-million loss (the negative of the portfolio outcome, or loss, is larger than the VaR).

Negative VaRData values are allowed, however negative VaR values indicate a highly profitable portfolio that cannot lose money at the given VaR confidence level. That is, the worst-case scenario at the given confidence level is still a profit.

Missing values (NaNs in VaRData) are discarded before applying the tests. Therefore, a different number of observations is reported (NumObs) for series with different number of missing values. The reported number of observations (NumObs) equals the original number of rows in the data (NumRows) minus the number of periods with missing values.

Data Types: double | table

### **Name-Value Pair Arguments**

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

Example: `vbt = varbacktest(PortfolioData, VaRData, 'PortfolioID', 'Equity100', 'VaRID', 'TotalVaR`

### **'PortfolioID' — User-defined ID for PortfolioData input**

character vector | string

User-defined ID for `PortfolioData` input, specified as the comma-separated pair consisting of `'PortfolioID'` and a character vector or string.

If `PortfolioData` is a numeric array, the default value for `PortfolioID` is `'Portfolio'`. If `PortfolioData` is a table, `PortfolioID` is set by default to the corresponding variable name in the table.

Data Types: `char` | `string`

### **'VaRID' — VaR identifier for VaRData columns**

character vector | cell array of character vectors | string | string array

VaR identifier for `VaRData` columns, specified as the comma-separated pair consisting of `'VaRID'` and a character vector or string. Multiple `VaRIDs` are specified using a 1-by-`NumVaRs` (or `NumVaRs`-by-1) cell array of character vectors or string vector with user-defined IDs for the `VaRData` columns.

If `VaRData` is a numeric array, the default value for `VaRID` is `'VaR'`. If `NumVaRs` = 1 or `NumVaRs` > 1, the default value is `'VaR1'`, `'VaR2'`, and so on. If `VaRData` is a table, `'VaRID'` is set by default to the corresponding variable names in the table

Data Types: `char` | `cell` | `string`

### **'VaRLevel' — VaR confidence level**

0.95 (default) | numeric with values between 0 and 1 | numeric array with values between 0 and 1

VaR confidence level, specified as the comma-separated pair consisting of `'VaRLevel'` and a numeric between 0 and 1 or a 1-by-`NumVaRs` numeric array with values between 0 and 1 for the corresponding columns in `VaRData`.

Data Types: `double`

## Output Arguments

### **vbt — varbacktest object**

object

`varbacktest` (`vbt`) object, contains a copy of the given portfolio data and VaR data (the `PortfolioData` and `VaRData` properties) and all combinations of portfolio ID, VaR ID, and VaR level to be tested (the `PortfolioID`, `VaRID`, and `VarLevel` properties). For more information on a `vbt` object, see `varbacktest`.

The `vbt` object has the following properties:

- `PortfolioData` — NumRows-by-1 numeric array containing a copy of the `PortfolioData`
- `VaRData` — NumRows-by-NumVaRs numeric array containing a copy of the `VaRData`
- `PortfolioID` — String containing the `PortfolioID`
- `VaRID` — 1-by-NumVaRs string vector containing the `VaRIDs` for the corresponding columns in `VaRData`
- `VarLevel` — 1-by-NumVaRs numeric array containing the `VarLevels` for the corresponding columns in `VaRData`.

## References

Basel Committee on Banking Supervision, *Supervisory Framework for the Use of 'Backtesting' in Conjunction with the Internal Models Approach to Market Risk Capital Requirements*. January, 1996, <http://www.bis.org/publ/bcbs22.htm>.

Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841–862.

Cogneau, P. "Backtesting Value-at-Risk: How Good is the Model?" *Intelligent Risk*, PRMIA, July, 2015.

Haas, M. "New Methods in Backtesting." *Financial Engineering*, Research Center Caesar, Bonn, 2001.

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

Kupiec, P. "Techniques for Verifying the Accuracy of Risk Management Models." *Journal of Derivatives*. Vol. 3, 1995, pp. 73–84.

McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management*. Princeton University Press, 2005.

Nieppola, O. "Backtesting Value-at-Risk Models." Master's Thesis, Helsinki School of Economics, 2009.



## See Also

### See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `table` | `tbf` | `tbfi` | `t1` | `tuff`

### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

`varbacktest`

**Introduced in R2016b**

## bin

Binomial test for value-at-risk (VaR) backtesting

### Syntax

```
TestResults = bin(vbt)
TestResults = bin(vbt,Name,Value)
```

### Description

`TestResults = bin(vbt)` generates the binomial test results for value-at-risk (VaR) backtesting.

`TestResults = bin(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

### Examples

#### Generate Bin Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the `bin` test results.

```
TestResults = bin(vbt)
```

```
TestResults =
```

```
1×9 table
```

PortfolioID	VaRID	VaRLevel	Bin	ZScoreBin	PValueBin	Observation
"Portfolio"	"VaR"	0.95	accept	0.68905	0.24539	1043

### Run Bin Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
    [Normal195 Normal199 Historical195 Historical199 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{ 'Normal195' 'Normal199' 'Historical195' 'Historical199' 'EWMA95' 'EWMA99' },
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `bin` test results using the `TestLevel` optional argument.

```
TestResults = bin(vbt, 'TestLevel',0.90)
```

```
TestResults =
```

6×9 table

PortfolioID	VaRID	VaRLevel	Bin	ZScoreBin	PValueBin
"Equity"	"Normal195"	0.95	accept	0.68905	0.24539
"Equity"	"Normal199"	0.99	reject	2.0446	0.020448
"Equity"	"Historical195"	0.95	accept	0.9732	0.16523
"Equity"	"Historical199"	0.99	accept	0.48858	0.31257
"Equity"	"EWMA95"	0.95	accept	0.9732	0.16523
"Equity"	"EWMA99"	0.99	reject	3.6006	0.00015875

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

### **vbt** — varbacktest object

object

varbacktest object, contains a copy of the given data (the `PortfolioData` and `VarData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a `vbt` object, see `varbacktest`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

Example: `TestResults = bin(vbt, 'TestLevel', 0.99)`

### **'TestLevel'** — Test confidence level

0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric between 0 and 1.

Data Types: double

## Output Arguments

### TestResults — Bin test results

table

Bin test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for corresponding VaR data column
- 'Bin' — Categorical array with categories `accept` and `reject` that indicate the result of the `bin` test
- 'ZScoreBin' — Z-score of the number of failures
- 'PValueBin' — P-value of the `bin` test
- 'Observations' — Number of observations
- 'Failures' — Number of failures.
- 'TestLevel' — Test confidence level.

---

**Note:** For `bin` test results, the terms `accept` and `reject` are used for convenience, technically a `bin` test does not accept a model. Rather, the test fails to reject it.

---

## Definitions

### Binomial Test (Bin)

The `bin` function performs a binomial test to assess if the number of failures is consistent with the VaR confidence level.

The binomial test is based on a normal approximation to the binomial distribution.

## Algorithms

The result of the binomial test is based on a normal approximation to a binomial distribution. Suppose:

- $N$  is the number of observations.
- $p = 1 - \text{VaRLevel}$  is the probability of observing a failure if the model is correct.
- $x$  is the number of failures.

If the failures are independent, then the number of failures is distributed as a binomial distribution with parameters  $N$  and  $p$ . The expected number of failures is  $N * p$ , and the standard deviation of the number of failures is

$$\sqrt{Np(1-p)}$$

The test statistic for the `bin` test is the z-score, defined as:

$$ZScoreBin = \frac{(x - Np)}{\sqrt{Np(1-p)}}$$

The z-score approximately follows a standard normal distribution. This approximation is not reliable for small values of  $N$  or small values of  $p$ , but for typical uses in VaR backtesting analyses ( $N = 250$  or much larger,  $p$  in the range 1–10%) the approximation gives results in line with other tests.

The  $p$ -value of the `bin` test is the probability that a standard normal distribution exceeds the absolute value of the z-score

$$PValueBin = 1 - F(|ZScoreBin|)$$

where  $F$  is the standard normal cumulative distribution. When too few failures are observed, relative to the expected failures,  $PValueBin$  is (approximately) the probability of observing that many failures or fewer. For too many failures, this is (approximately) the probability of observing that many failures or more.

## References

Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.

## See Also

### See Also

cc | cci | pof | runtests | summary | tbf | tbfi | t1 | tuff | varbacktest

### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

“Binomial Test” on page 2-3

varbacktest

**Introduced in R2016b**

## cc

Conditional coverage mixed test for value-at-risk (VaR) backtesting

## Syntax

```
TestResults = cc(vbt)
TestResults = cc(vbt,Name,Value)
```

## Description

`TestResults = cc(vbt)` generates the conditional coverage (CC) mixed test for value-at-risk (VaR) backtesting.

`TestResults = cc(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

## Examples

### Generate CC Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```



Generate the CC test results.

```
TestResults = cc(vbt)
```

```
TestResults =
```

```
1×19 table
```

PortfolioID	VaRID	VaRLevel	CC	LRatioCC	PValueCC	POF	LR
"Portfolio"	"VaR"	0.95	accept	0.72013	0.69763	accept	0.4

### Run the CC Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
    [Normal195 Normal199 Historical195 Historical199 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{ 'Normal195' 'Normal199' 'Historical195' 'Historical199' 'EWMA95' 'EWMA99' },...
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the CC test results using the `TestLevel` optional input.

```
TestResults = cc(vbt,'TestLevel',0.90)
```

```
TestResults =
```

6×19 table

PortfolioID	VaRID	VaRLevel	CC	LRatioCC	PValueCC	PO
"Equity"	"Normal195"	0.95	accept	0.72013	0.69763	ac
"Equity"	"Normal199"	0.99	accept	4.0757	0.13031	re
"Equity"	"Historical195"	0.95	accept	1.0487	0.59194	ac
"Equity"	"Historical199"	0.99	accept	0.5073	0.77597	ac
"Equity"	"EWMA95"	0.95	accept	0.95051	0.62173	ac
"Equity"	"EWMA99"	0.99	reject	10.779	0.0045645	re

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

**vbt** — **varbacktest object**  
object

**varbacktest** object, contains a copy of the given data (the **PortfolioData** and **VarData** properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a **vbt** object, see **varbacktest**.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of **Name**, **Value** arguments. **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1, Value1, . . . , NameN, ValueN**.

Example: `TestResults = cc(vbt, 'TestLevel', 0.99)`

**'TestLevel'** — **Test confidence level**  
0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of **'TestLevel'** and a numeric between 0 and 1.

Data Types: double

## Output Arguments

### TestResults — cc test results

table

cc test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for corresponding VaR data column
- 'CC' — Categorical array with the categories `accept` and `reject` that indicate the result of the cc test
- 'LRatioCC' — Likelihood ratio of the cc test
- 'PValueCC' — P-value of the cc test
- 'POF' — Categorical array with the categories `accept` and `reject` that indicate the result of the pof test
- 'LRatioPOF' — Likelihood ratio of the pof test
- 'PValuePOF' — P-value of the pof test
- 'CCI' — Categorical array with categories 'accept' and 'reject' that indicate the result of the cci test
- 'LRatioCCI' — Likelihood ratio of the cci test
- 'PValueCCI' — P-value of the cci test
- 'Observations' — Number of observations
- 'Failures' — Number of failures
- 'N00' — Number of periods with no failures followed by a period with no failures
- 'N10' — Number of periods with failures followed by a period with no failures
- 'N01' — Number of periods with no failures followed by a period with failures
- 'N11' — Number of periods with failures followed by a period with failures
- 'TestLevel' — Test confidence level

---

**Note:** For **cc** test results, the terms **accept** and **reject** are used for convenience, technically a **cc** test does not accept a model. Rather, the test fails to reject it.

---

## Definitions

### Conditional Coverage (CC) Mixed Test

The **cc** function performs the conditional coverage mixed test, also known as Christoffersen's interval forecasts method.

'Mixed' means that it combines a frequency and an independence test. The frequency test is Kupiec's proportion of failures test, implemented by the **pof** function. The independence test is the conditional coverage independence test implemented by the **cci** function. This is a likelihood ratio test proposed by Christoffersen (1998) to assess the independence of failures on consecutive time periods. The **CC** test combines the **POF** test and the **CCI** test.

## Algorithms

The likelihood ratio (test statistic) of the **cc** test is the sum of the likelihood ratios of the **pof** and **cci** tests,

$$LRatioCC = LRatioPOF + LRatioCCI$$

which is asymptotically distributed as a chi-square distribution with two degrees of freedom. See the Algorithms section in **pof** and **cci** for the definition of their likelihood ratios.

The *p*-value of the **cc** test is the probability that a chi-square distribution with two degrees of freedom exceeds the likelihood ratio *LRatioCC*,

$$PValueCC = 1 - F(LRatioCC)$$

where *F* is the cumulative distribution of a chi-square variable with two degrees of freedom.

The result of the **cc** test is to accept if

$$F(LRatioCC) < F(TestLevel)$$

and reject otherwise, where  $F$  is the cumulative distribution of a chi-square variable with two degrees of freedom.

## References

Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841–862.

## See Also

### See Also

[bin](#) | [cci](#) | [pof](#) | [runtests](#) | [summary](#) | [tbf](#) | [tbfi](#) | [t1](#) | [tuff](#) | [varbacktest](#)

## Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

“Christoffersen’s Interval Forecast Tests” on page 2-5

[varbacktest](#)

**Introduced in R2016b**

## cci

Conditional coverage independence test for value-at-risk (VaR) backtesting

## Syntax

```
TestResults = cci(vbt)
TestResults = cci(vbt,Name,Value)
```

## Description

`TestResults = cci(vbt)` generates the conditional coverage independence (CCI) for value-at-risk (VaR) backtesting.

`TestResults = cci(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

## Examples

### Generate CCI Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the cci test results.

```
TestResults = cci(vbt)
```

```
TestResults =
```

```
1×13 table
```

PortfolioID	VaRID	VaRLevel	CCI	LRatioCCI	PValueCCI	Observation
"Portfolio"	"VaR"	0.95	accept	0.25866	0.61104	1043

### Run the CCI Test for VaR Backtests for Multiple VaR's at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
```

```
vbt = varbacktest(EquityIndex,...
    [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'},
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the cci test results using the `TestLevel` optional input.

```
TestResults = cci(vbt,'TestLevel',0.90)
```

```
TestResults =
```

6×13 table

PortfolioID	VaRID	VaRLevel	CCI	LRatioCCI	PValueCCI	OR
"Equity"	"Normal195"	0.95	accept	0.25866	0.61104	10
"Equity"	"Normal199"	0.99	accept	0.56393	0.45268	10
"Equity"	"Historical195"	0.95	accept	0.13847	0.70981	10
"Equity"	"Historical199"	0.99	accept	0.27962	0.59695	10
"Equity"	"EWMA95"	0.95	accept	0.040277	0.84094	10
"Equity"	"EWMA99"	0.99	accept	0.94909	0.32995	10

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

### **vbt** — varbacktest object

object

varbacktest object, contains a copy of the given data (the `PortfolioData` and `VarData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a `vbt` object, see `varbacktest`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = cci(vbt,'TestLevel',0.99)`

### **'TestLevel'** — Test confidence level

0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric between 0 and 1.

Data Types: double



## Output Arguments

### TestResults — cci test results

table

cci test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for the corresponding VaR data column
- 'CCI' — Categorical array with the categories `accept` and `reject` that indicate the result of the cci test
- 'LRatioCCI' — Likelihood ratio of the cci test
- 'PValueCCI' — P-value of the cci test
- 'Observations' — Number of observations
- 'Failures' — Number of failures
- 'N00' — Number of periods with no failures followed by a period with no failures
- 'N10' — Number of periods with failures followed by a period with no failures
- 'N01' — Number of periods with no failures followed by a period with failures
- 'N11' — Number of periods with failures followed by a period with failures
- 'TestLevel' — Test confidence level

---

**Note:** For cci test results, the terms `accept` and `reject` are used for convenience, technically a cci test does not accept a model. Rather, the test fails to reject it.

---

## Definitions

### Conditional Coverage Independence (CCI) Test

The cci function performs the conditional coverage independence test.

This is a likelihood ratio test proposed by Christoffersen (1998) to assess the independence of failures on consecutive time periods. For the conditional coverage mixed test, see the `cc` function.

## Algorithms

To define the likelihood ratio (test statistic) of the `cc` test, first define the following quantities:

- 'N00' — Number of periods with no failures followed by a period with no failures
- 'N10' — Number of periods with failures followed by a period with no failures
- 'N01' — Number of periods with no failures followed by a period with failures
- 'N11' — Number of periods with failures followed by a period with failures

Then define the following conditional probability estimates:

- $p_{01}$  = Probability of having a failure on period  $t$ , given that there was no failure on period  $t - 1$

$$p_{01} = \frac{N_{01}}{(N_{00} + N_{01})}$$

- $p_{11}$  = Probability of having a failure on period  $t$ , given that there was a failure on period  $t - 1$

$$p_{11} = \frac{N_{11}}{(N_{10} + N_{11})}$$

Define also the unconditional probability estimate of observing a failure:

$p_{UC}$  = Probability of having a failure on period  $t$

$$p_{UC} = \frac{(N_{01} + N_{11})}{(N_{00} + N_{01} + N_{10} + N_{11})}$$

The likelihood ratio of the CCI test is then given by

$$LRatioCCI = -2 \log \left( \frac{(1 - pUC)^{N00+N10} pUC^{N01+N11}}{(1 - p01)^{N00} p01^{N01} (1 - p11)^{N10} p11^{N11}} \right)$$

$$= -2((N00+N10)\log(1 - pUC) + (N01+N11)\log(pUC) - N00 \log(1 - p01) - N01 \log(p01) - N10 \log(1 - p11) - N11 \log(p11))$$

which is asymptotically distributed as a chi-square distribution with one degree of freedom.

The  $p$ -value of the CCI test is the probability that a chi-square distribution with one degree of freedom exceeds the likelihood ratio  $LRatioCCI$ ,

$$PValueCCI = 1 - F(LRatioCCI)$$

where  $F$  is the cumulative distribution of a chi-square variable with one degree of freedom.

The result of the test is to accept if

$$F(LRatioCCI) < F(TestLevel)$$

and reject otherwise, where  $F$  is the cumulative distribution of a chi-square variable with one degree of freedom.

If one or more of the quantities  $N00$ ,  $N10$ ,  $N01$ , or  $N11$  are zero, the likelihood ratio is handled differently. The likelihood ratio as defined above is composed of three likelihood functions of the form

$$L = (1 - p)^{n1} \times p^{n2}$$

For example, in the numerator of the likelihood ratio, there is a likelihood function of the form  $L$  with  $p = pUC$ ,  $n1 = N00 + N10$ , and  $n2 = N01 + N11$ . There are two such likelihood functions in the denominator of the likelihood ratio.

It can be shown that whenever  $n1 = 0$  or  $n2 = 0$ , the likelihood function  $L$  can be replaced by the constant value 1. Therefore, whenever  $N00$ ,  $N10$ ,  $N01$ , or  $N11$  is zero, replace the corresponding likelihood functions by 1 in the likelihood ratio, and the likelihood ratio is well defined.

## References

Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841–862.

## See Also

### See Also

[bin](#) | [cc](#) | [pof](#) | [runtests](#) | [summary](#) | [tbf](#) | [tbfi](#) | [t1](#) | [tuff](#) | [varbacktest](#)

## Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

“Christoffersen’s Interval Forecast Tests” on page 2-5

[varbacktest](#)

**Introduced in R2016b**

# pof

Proportion of failures test for value-at-risk (VaR) backtesting

## Syntax

```
TestResults = pof(vbt)
TestResults = pof(vbt,Name,Value)
```

## Description

`TestResults = pof(vbt)` generates the proportion of failures (POF) test for value-at-risk (VaR) backtesting.

`TestResults = pof(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

## Examples

### Generate POF Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the `pof` test results.

```
TestResults = pof(vbt, 'TestLevel',0.99)
```

```
TestResults =
```

```
1×9 table
```

PortfolioID	VaRID	VaRLevel	POF	LRatioPOF	PValuePOF	Observation
"Portfolio"	"VaR"	0.95	accept	0.46147	0.49694	1043

### Run the POF Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
    [Normal195 Normal199 Historical195 Historical199 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{ 'Normal195' 'Normal199' 'Historical195' 'Historical199' 'EWMA95' 'EWMA99' },
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `pof` test results using the `TestLevel` optional input.

```
TestResults = pof(vbt, 'TestLevel',0.90)
```

```
TestResults =
```

6×9 table

PortfolioID	VaRID	VaRLevel	POF	LRatioPOF	PValuePOF	OB
"Equity"	"Normal195"	0.95	accept	0.46147	0.49694	10
"Equity"	"Normal199"	0.99	reject	3.5118	0.060933	10
"Equity"	"Historical195"	0.95	accept	0.91023	0.34005	10
"Equity"	"Historical199"	0.99	accept	0.22768	0.63325	10
"Equity"	"EWMA95"	0.95	accept	0.91023	0.34005	10
"Equity"	"EWMA99"	0.99	reject	9.8298	0.0017171	10

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

**vbt** — **varbacktest object**  
object

varbacktest object, contains a copy of the given data (the PortfolioData and VarData properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a vbt object, see varbacktest.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

Example: TestResults = pof(vbt, 'TestLevel', 0.99)

**'TestLevel'** — **Test confidence level**  
0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric between 0 and 1.

Data Types: `double`

## Output Arguments

### **TestResults** — **pof** test results

table

`pof` test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR level to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for the corresponding VaR data column
- 'POF' — Categorical array with the categories `accept` and `reject` that indicate the result of the `pof` test
- 'LRatioPOF' — Likelihood ratio of the `pof` test
- 'PValuePOF' — P-value of the `pof` test
- 'Observations' — Number of observations
- 'Failures' — Number of failures
- 'TestLevel' — Test confidence level

---

**Note:** For `pof` test results, the terms `accept` and `reject` are used for convenience, technically a `pof` test does not accept a model. Rather, the test fails to reject it.

---

## Definitions

### **Proportion of Failures (POF) Test**

The `pof` function performs Kupiec's proportion of failures test.

The POF test is a likelihood ratio test proposed by Kupiec (1995) to assess if the proportion of failures (number of failures divided by number of observations) is consistent with the VaR confidence level.



## Algorithms

The likelihood ratio (test statistic) of the pof test is given by

$$LRatioPOF = -2 \log \left( \frac{(1 - pVaR)^{N-x} pVaR^x}{\left(1 - \frac{x}{N}\right)^{N-x} \left(\frac{x}{N}\right)^x} \right) = -2 \left[ (N-x) \log \left( \frac{N(1 - pVaR)}{N-x} \right) + x \log \left( \frac{NpVaR}{x} \right) \right]$$

where  $N$  is the number of observations,  $x$  is the number of failures, and  $pVaR = 1 - VaRLevel$ . This test statistic is asymptotically distributed as a chi-square distribution with one degree of freedom. By the properties of the logarithm,

$$LRatioPOF = -2N \log(1 - pVar) \quad \text{if } x = 0.$$

and

$$LRatioPOF = -2N \log(pVar) \quad \text{if } x = N.$$

The  $p$ -value of the POF test is the probability that a chi-square distribution with one degree of freedom exceeds the likelihood ratio  $LRatioPOF$

$$PValuePOF = 1 - F(LRatioPOF)$$

where  $F$  is the cumulative distribution of a chi-square variable with one degree of freedom.

The result of the test is to accept if

$$PValuePOF < F(TestLevel)$$

and reject otherwise, where  $F$  is the cumulative distribution of a chi-square variable with one degree of freedom.

## References

Kupiec, P. "Techniques for Verifying the Accuracy of Risk Management Models." *Journal of Derivatives*. Vol. 3, 1995, pp. 73–84.

## See Also

### See Also

`bin` | `cc` | `cci` | `runtests` | `summary` | `tbf` | `tbfi` | `t1` | `tuff` | `varbacktest`

### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

“Kupiec’s POF and TUFF Tests” on page 2-4

`varbacktest`

**Introduced in R2016b**

## runtests

Run all tests in varbacktest

### Syntax

```
TestResults = runtests(vbt)
TestResults = runtests(vbt,Name,Value)
```

### Description

`TestResults = runtests(vbt)` runs all the tests in the `varbacktest` object. `runtests` reports only the final test result. For test details such as likelihood ratios, run individual tests.

`TestResults = runtests(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

### Examples

#### Run All VaR Backtests

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the `TestResults` report for all VaR backtests.

```
TestResults = runtests(vbt, 'TestLevel', 0.99)
```

```
TestResults =
```

```
1×11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF	CC
"Portfolio"	"VaR"	0.95	green	accept	accept	accept	accept

### Run All VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object and run all tests.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
    [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{ 'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99' },
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99]);
runtests(vbt)
```

```
ans =
```

```
6×11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF
"Equity"	"Normal95"	0.95	green	accept	accept	accept
"Equity"	"Normal99"	0.99	yellow	reject	accept	accept
"Equity"	"Historical95"	0.95	green	accept	accept	accept
"Equity"	"Historical99"	0.99	green	accept	accept	accept
"Equity"	"EWMA95"	0.95	green	accept	accept	accept
"Equity"	"EWMA99"	0.99	yellow	reject	reject	accept

- “VaR Backtesting Workflow” on page 2-8

- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

### **vbt** — varbacktest object

object

varbacktest object, contains a copy of the given data (the `PortfolioData` and `VarData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a `vbt` object, see `varbacktest`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = runtests(vbt,'TestLevel',0.99)`

### **'TestLevel'** — Test confidence level

0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric between 0 and 1.

Data Types: `double`

## Output Arguments

### **TestResults** — Results

table

Results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- `'PortfolioID'` — Portfolio ID for the given data
- `'VaRID'` — VaR ID for each of the VaR data columns provided

- 'VaRLevel' — VaR level for the corresponding VaR data column
- 'TL' — Categorical (ordinal) array with categories **green**, **yellow**, and **red** that indicate the result of the **traffic light (tl)** test
- 'Bin' — Categorical array with categories **accept** and **reject** that indicate the result of the **bin** test
- 'POF' — Categorical array with the categories **accept** and **reject** that indicate the result of the **pof** test.
- 'TUFF' — Categorical array with the categories **accept** and **reject** that indicate the result of the **tuff** test
- 'CC' — Categorical array with the categories **accept** and **reject** that indicate the result of the **cc** test
- 'CCI' — Categorical array with the categories **accept** and **reject** that indicate the result of the **cci** test
- 'TBF' — Categorical array with the categories **accept** and **reject** that indicate the result of the **tbf** test
- 'TBFi' — Categorical array with the categories **accept** and **reject** that indicate the result of the **tbfi** test

---

**Note:** For the test results, the terms **accept** and **reject** are used for convenience, technically a test does not accept a model. Rather, a test fails to reject it.

---

## See Also

### See Also

[cc](#) | [cci](#) | [pof](#) | [summary](#) | [tbf](#) | [tbfi](#) | [tl](#) | [tuff](#) | [varbacktest](#)

### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

[varbacktest](#)

**Introduced in R2016b**

## summary

Report on varbacktest data

### Syntax

```
S = summary(vbt)
```

### Description

`S = summary(vbt)` returns a basic report on the given varbacktest data, including the number of observations, the number of failures, the observed confidence level, and so on (see `S` for details).

### Examples

#### Generate a Summary Report

Create a varbacktest object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the summary report.

```
S = summary(vbt)
```

```
S =
```

```
1×10 table
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Failures	Ex
"Portfolio"	"VaR"	0.95	0.94535	1043	57	52

### Run a Summary Report for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object and generate a summary report.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
    [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'},
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99]);
S = summary(vbt)
```

```
S =
```

```
6×10 table
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Failures	Ex
"Equity"	"Normal95"	0.95	0.94535	1043	57	52
"Equity"	"Normal99"	0.99	0.9837	1043	17	17
"Equity"	"Historical95"	0.95	0.94343	1043	59	59
"Equity"	"Historical99"	0.99	0.98849	1043	12	12
"Equity"	"EWMA95"	0.95	0.94343	1043	59	59
"Equity"	"EWMA99"	0.99	0.97891	1043	22	22

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13



## Input Arguments

### **vbt** — varbacktest object

object

varbacktest object, contains a copy of the given data (the `PortfolioData` and `VarData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a `vbt` object, see `varbacktest`.

## Output Arguments

### **S** — Summary report

table

Summary report, returned as a table. The table rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for the corresponding VaR data column
- 'ObservedLevel' — Observed confidence level, defined as number of periods without failures divided by number of observations
- 'Observations' — Number of observations, where missing values are removed from the data
- 'Failures' — Number of failures, where a failure occurs whenever the loss (negative of portfolio data) exceeds the VaR
- 'Expected' — Expected number of failures, defined as the number of observations multiplied by one minus the VaR level
- 'Ratio' — Ratio of the number of failures to expected number of failures
- 'FirstFailure' — Number of periods until first failure
- 'Missing' — Number of periods with missing values removed from the sample

## See Also

### See Also

`cc` | `cci` | `pof` | `runtests` | `tbf` | `tbfi` | `t1` | `tuff` | `varbacktest`

### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

`varbacktest`

**Introduced in R2016b**

## tbf

Time between failures mixed test for value-at-risk (VaR) backtesting

### Syntax

```
TestResults = tbf(vbt)
TestResults = tbf(vbt,Name,Value)
```

### Description

`TestResults = tbf(vbt)` generates the time between failures mixed test (TBF) for value-at-risk (VaR) backtesting.

`TestResults = tbf(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

### Examples

#### Generate TBF Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the `tbf` test results.

```
TestResults = tbf(vbt)
```

```
TestResults =
```

```
1×20 table
```

PortfolioID	VaRID	VaRLevel	TBF	LRatioTBF	PValueTBF	POF
"Portfolio"	"VaR"	0.95	reject	88.952	0.0055565	accept

### Run the TBF Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
    [Normal195 Normal199 Historical195 Historical199 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{ 'Normal195' 'Normal199' 'Historical195' 'Historical199' 'EWMA95' 'EWMA99' },
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tbf` test results using the `TestLevel` optional input.

```
TestResults = tbf(vbt, 'TestLevel',0.90)
```

```
TestResults =
```

6×20 table

PortfolioID	VaRID	VaRLevel	TBF	LRatioTBF	PValueTBF
"Equity"	"Normal195"	0.95	reject	88.952	0.0055565
"Equity"	"Normal199"	0.99	reject	26.441	0.090095
"Equity"	"Historical195"	0.95	reject	83.63	0.023609
"Equity"	"Historical199"	0.99	accept	16.456	0.22539
"Equity"	"EWMA95"	0.95	accept	72.545	0.12844
"Equity"	"EWMA99"	0.99	reject	41.66	0.0099428

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

**vbt** — **varbacktest object**  
object

varbacktest object, contains a copy of the given data (the PortfolioData and VarData properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a vbt object, see varbacktest.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

Example: TestResults = tbf(vbt, 'TestLevel', 0.99)

**'TestLevel'** — **Test confidence level**  
0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric between 0 and 1.

Data Types: `double`

## Output Arguments

### TestResults — `tbftest` results

table

`tbftest` results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- `'PortfolioID'` — Portfolio ID for the given data
- `'VaRID'` — VaR ID for each of the VaR data columns provided
- `'VaRLevel'` — VaR level for the corresponding VaR data column
- `'TBF'` — Categorical array with categories `accept` and `reject` that indicate the result of the `tbftest`
- `'LRatioTBF'` — Likelihood ratio of the `tbftest`
- `'PValueTBF'` — P-value of the `tbftest`
- `'POF'` — Categorical array with the categories `accept` and `reject` that indicate the result of the POF test
- `'LRatioPOF'` — Likelihood ratio of the `pof` test
- `'PValuePOF'` — P-value of the `pof` test
- `'TBFi'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `tbftest`
- `'LRatioTBFi'` — Likelihood ratio of the `tbftest`
- `'PValueTBFi'` — P-value of the `tbftest`
- `'Observations'` — Number of observations
- `'Failures'` — Number of failures
- `'TBFMin'` — Minimum value of observed times between failures
- `'TBFQ1'` — First quartile of observed times between failures
- `'TBFQ2'` — Second quartile of observed times between failures
- `'TBFQ3'` — Third quartile of observed times between failures
- `'TBFMax'` — Maximum value of observed times between failures

- 'TestLevel' — Test confidence level

---

**Note:** For `tbf` test results, the terms `accept` and `reject` are used for convenience, technically a `tbf` test does not accept a model. Rather, the test fails to reject it.

---

## Definitions

### Time Between Failures (TBF) Mixed Test

The `tbf` function performs the time between failures mixed test, also known as the Haas mixed Kupiec test.

'Mixed' means that it combines a frequency and an independence test. The frequency test is Kupiec's proportion of failures (POF) test. The independence test is the time between failures independence (TBF<sub>I</sub>) test. The TBF test is an extension of Kupiec's time until first failure (TUFF) test, proposed by Haas (2001), to take into account not only the time until the first failure, but the time between all failures. The `tbf` function combines the `pof` test and the `tbfi` test.

## Algorithms

The likelihood ratio (test statistic) of the TBF test is the sum of the likelihood ratios of the POF and TBF<sub>I</sub> tests

$$LRatioTBF = LRatioPOF + LRatioTBF_I$$

which is asymptotically distributed as a chi-square distribution with  $x+1$  degrees of freedom, where  $x$  is the number of failures. See the Algorithms sections for `pof` and `tbfi` for the definitions of their likelihood ratios.

The  $p$ -value of the `tbf` test is the probability that a chi-square distribution with  $x+1$  degrees of freedom exceeds the likelihood ratio  $LRatioTBF$

$$PValueTBF = 1 - F(LRatioTBF)$$

where  $F$  is the cumulative distribution of a chi-square variable with  $x+1$  degrees of freedom and  $x$  is the number of failures.

The result of the test is to accept if

$$F(LRatioTBF) < F(TestLevel)$$

and reject otherwise, where  $F$  is the cumulative distribution of a chi-square variable with  $x+1$  degrees of freedom and  $x$  is the number of failures. If the likelihood ratio ( $LRatioTBF$ ) is undefined, that is, with no failures yet, the TBF result is to accept only when both POF and TBF tests accept.

### References

Haas, M. *"New Methods in Backtesting."* Financial Engineering, Research Center Caesar, Bonn, 2001.

### See Also

#### See Also

[bin](#) | [cc](#) | [cci](#) | [pof](#) | [runtests](#) | [summary](#) | [tbfi](#) | [tl](#) | [tuff](#) | [varbacktest](#)

### Topics

["VaR Backtesting Workflow"](#) on page 2-8

["Value-at-Risk Estimation and Backtesting"](#) on page 2-13

["Haas's Time Between Failures or Mixed Kupiec's Test"](#) on page 2-6

[varbacktest](#)

**Introduced in R2016b**



## tbfi

Time between failures independence test for value-at-risk (VaR) backtesting

### Syntax

```
TestResults = tbfi(vbt)
TestResults = tbfi(vbt,Name,Value)
```

### Description

`TestResults = tbfi(vbt)` generates the time between failures independence (TBFI) test for value-at-risk (VaR) backtesting.

`TestResults = tbfi(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

### Examples

#### Generate TBFI Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the `tbfi` test results.

```
TestResults = tbfi(vbt)
```

```
TestResults =
```

```
1×14 table
```

PortfolioID	VaRID	VaRLevel	TBFI	LRatioTBFI	PValueTBFI	Observat
"Portfolio"	"VaR"	0.95	reject	88.491	0.0047475	1043

### Run the TBFI Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
```

```
vbt = varbacktest(EquityIndex,...
    [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'},
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tbfi` test results using the `TestLevel` optional input.

```
TestResults = tbfi(vbt,'TestLevel',0.90)
```

```
TestResults =
```

6×14 table

PortfolioID	VaRID	VaRLevel	TBFI	LRatioTBFI	PValueTBFI
"Equity"	"Normal195"	0.95	reject	88.491	0.0047475
"Equity"	"Normal199"	0.99	accept	22.929	0.15157
"Equity"	"Historical195"	0.95	reject	82.719	0.022513
"Equity"	"Historical199"	0.99	accept	16.228	0.18101
"Equity"	"EWMA95"	0.95	accept	71.635	0.12517
"Equity"	"EWMA99"	0.99	reject	31.83	0.080339

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

**vbt** — varbacktest object  
object

varbacktest object, contains a copy of the given data (the `PortfolioData` and `VarData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a `vbt` object, see `varbacktest`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = tbf(vbt,'TestLevel',0.99)`

**'TestLevel'** — Test confidence level  
0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric between 0 and 1.

Data Types: double

## Output Arguments

### TestResults — `tbfi` test results

table

`tbfi` test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for the corresponding VaR data column
- 'TBFI' — Categorical array with the categories `accept` and `reject` that indicate the result of the `tbfi` test
- 'LRatioTBFI' — Likelihood ratio of the `tbfi` test
- 'PValueTBFI' — P-value of the `tbfi` test
- 'Observations' — Number of observations
- 'Failures' — Number of failures
- 'TBMin' — Minimum value of observed times between failures
- 'TBFQ1' — First quartile of observed times between failures
- 'TBFQ2' — Second quartile of observed times between failures
- 'TBFQ3' — Third quartile of observed times between failures
- 'TBMax' — Maximum value of observed times between failures
- 'TestLevel' — Test confidence level

---

**Note:** For `tbfi` test results, the terms `accept` and `reject` are used for convenience, technically a `tbfi` test does not accept a model. Rather, the test fails to reject it.

---

## Definitions

### Time Between Failures Independence (TBIF) Test

The `tbfi` function performs the time between failures independence test. This test is an extension of Kupiec's time until first failure (TUFF) test.

TBFI was proposed by Haas (2001) to test for independence. It takes into account not only the time until the first failure, but also the time between all failures. For the time between failures mixed test, see the `tbfi` function.

## Algorithms

The likelihood ratio (test statistic) of the TBFI test is the sum of TUFF likelihood ratios for each time between failures. If  $x$  is the number of failures, and  $n_1$  is the number of periods until the first failure,  $n_2$  the number of periods between the first and the second failure, and, in general,  $n_i$  is the number of periods between failure  $i - 1$  and failure  $i$ , then a likelihood ratio  $LRatioTBFI_i$  for each  $n_i$  is based on the TUFF formula

$$LRatioTBFI_i = LRatioTUFF(n_i) = -2 \sum_{i=1}^x \log \left( \frac{pVaR (1 - pVaR)^{n_i - 1}}{\left(\frac{1}{n_i}\right) \left(1 - \frac{1}{n_i}\right)^{n_i - 1}} \right)$$

$$= -2(\log(pVaR) + (n_i - 1)\log(1 - pVaR) + n_i \log(n_i) - (n_i - 1)\log(n_i - 1))$$

As with the `tuff` test,  $LRatioTBFI_i = -2\log(pVaR)$  if  $n_i = 1$ .

The TBFI likelihood ratio  $LRatioTBFI$  is then the sum of the individual likelihood ratios for all times between failures

$$LRatioTBFI = \sum_{i=1}^x LRatioTBFI_i$$

which is asymptotically distributed as a chi-square distribution with  $x$  degrees of freedom, where  $x$  is the number of failures.

The  $p$ -value of the `tbfi` test is the probability that a chi-square distribution with  $x$  degrees of freedom exceeds the likelihood ratio  $LRatioTBFI$

$$PValueTBFI = 1 - F(LRatioTBFI)$$

where  $F$  is the cumulative distribution of a chi-square variable with  $x$  degrees of freedom and  $x$  is the number of failures.

The result of the test is to accept if

$$F(LRatioTBFI) < F(TestLevel)$$

and reject otherwise, where  $F$  is the cumulative distribution of a chi-square variable with  $x$  degrees of freedom and  $x$  is the number of failures.

If there are no failures in the sample, the test statistic is not defined. This is handled the same as a TUFF test with no failures. For more information, see `tuff`.

### References

Haas, M. *"New Methods in Backtesting."* Financial Engineering, Research Center Caesar, Bonn, 2001.

### See Also

#### See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `t1` | `tuff` | `varbacktest`

### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

“Haas’s Time Between Failures or Mixed Kupiec’s Test” on page 2-6

`varbacktest`

**Introduced in R2016b**

## tl

Traffic light test for value-at-risk (VaR) backtesting

## Syntax

```
TestResults = tl(vbt)
```

## Description

`TestResults = tl(vbt)` generates the traffic light (TL) test for value-at-risk (VaR) backtesting.

## Examples

### Generate Traffic Light Test Results

Create a `varbacktest` object.

```
load VarBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the `tl` test results.

```
TestResults = tl(vbt)
```

```
TestResults =
```

```
1×9 table
```

PortfolioID	VaRID	VaRLevel	TL	Probability	TypeI	Increase
"Portfolio"	"VaR"	0.95	green	0.77913	0.26396	0

### Run the TL Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
```

```
vbt = varbacktest(EquityIndex,...
    [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{ 'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99' },
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tl` test results.

```
TestResults = tl(vbt)
```

```
TestResults =
```

```
6×9 table
```

PortfolioID	VaRID	VaRLevel	TL	Probability	TypeI	Increase
-------------	-------	----------	----	-------------	-------	----------



"Equity"	"Normal195"	0.95	green	0.77913	0.26396
"Equity"	"Normal199"	0.99	yellow	0.97991	0.03686
"Equity"	"Historical195"	0.95	green	0.85155	0.18232
"Equity"	"Historical199"	0.99	green	0.74996	0.35269
"Equity"	"EWMA95"	0.95	green	0.85155	0.18232
"Equity"	"EWMA99"	0.99	yellow	0.99952	0.0011122

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

### **vbt** — varbacktest object

object

varbacktest object, contains a copy of the given data (the `PortfolioData` and `VarData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a `vbt` object, see `varbacktest`.

## Output Arguments

### **TestResults** — tl test results

table

tl test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for the corresponding VaR data column
- 'TL' — Categorical (ordinal) array with the categories `green`, `yellow`, and `red` that indicate the result of the traffic light `tl` test
- 'Probability' — Cumulative probability of observing up to the corresponding number of failures

- 'TypeI' — Probability of observing the corresponding number of failures or more if the model is correct
- 'Increase' — Increase in the scaling factor
- 'Observations' — Number of observations
- 'Failures' — Number of failures

## Definitions

### Traffic Light Test

The `tl` function performs Basel's traffic light test, also known as three-zone test. Basel's methodology can be applied to any number of time periods and VaR confidence levels, as explained in "Algorithms" on page 5-205.

The Basel Committee reports, as an example, a table of the three zones for 250 time periods and a VaR confidence level of 0.99. The increase in scaling factor in the table reported by Basel has some ad-hoc adjustments (rounding, etc.) not explicitly described in the Basel document. The following table compares the increase in scaling factor reported in the Basel document for the case of 250 periods and 0.99% VaR confidence level, and the increase in the factors reported by the TL test.

Failures	Zone	Increase Basel	Increase TL
0	Green	0	0
1	Green	0	0
2	Green	0	0
3	Green	0	0
4	Green	0	0
5	Yellow	0.40	0.3982
6	Yellow	0.50	0.5295
7	Yellow	0.65	0.6520
8	Yellow	0.75	0.7680
9	Yellow	0.85	0.8791
10	Red	1	1

The **tl** function computes the scaling factor following the methodology described in the Basel document (see Bibliography) and is explained in the “Algorithms” on page 5-205 section. The **tl** function does not apply any ad-hoc adjustments.

## Algorithms

The traffic light test is based on a binomial distribution. Suppose **N** is the number of observations,  $p = 1 - \text{VaRLevel}$  is the probability of observing a failure if the model is correct, and  $x$  is the number of failures.

The test computes the cumulative probability of observing up to  $x$  failures, reported in the 'Probability' column,

$$\text{Probability} = \text{Probability}(X \leq x | N, p) = F(x | N, p)$$

where  $F(x | N, p)$  is the cumulative distribution of a binomial variable with parameters  $N$  and  $p$ , with  $p = 1 - \text{VaRLevel}$ . The three zones are defined based on this cumulative probability:

- Green:  $F(x | N, p) \leq 0.95$
- Yellow:  $0.95 < F(x | N, p) \leq 0.9999$
- Red:  $0.9999 < F(x | N, p)$

The probability of a Type-I error, reported in the 'TypeI' column, is

$$\text{TypeI} = \text{TypeI}(x | N, p) = 1 - F(X \geq x | N, p)$$

This probability corresponds to the probability of mistakenly rejecting the model if the model were correct. *Probability* and *TypeI* do not sum up to 1, they exceed 1 by exactly the probability of having  $x$  failures.

The increase in scaling factor, reported in the 'Increase' column, is always 0 for the green zone and always 1 for the red zone. For the yellow zone, it is an adjustment based on the relative difference between the assumed VaR confidence level (*VaRLevel*) and the observed confidence level ( $x / N$ ), where **N** is the number of observations and  $x$  is the number of failures. To find the increase under the assumption of a normal distribution, compute the critical values *zAssumed* and *zObserved*.

The increase to the baseline scaling factor is given by

$$Increase = Baseline \times \left( \frac{z_{Assumed}}{z_{Observed}} - 1 \right)$$

with the restriction that the increase cannot be negative or greater than 1. The baseline scaling factor in the Basel rules is 3.

The `tl` function computes the scaling factor following this methodology, which is also described in the Basel document (see Bibliography). The `tl` function does not apply any ad-hoc adjustments.

### References

Basel Committee on Banking Supervision, *Supervisory Framework for the Use of 'Backtesting' in Conjunction with the Internal Models Approach to Market Risk Capital Requirements*. January, 1996, <http://www.bis.org/publ/bcbs22.htm>.

### See Also

#### See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `tuff` | `varbacktest`

#### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

“Traffic Light Test” on page 2-3

`varbacktest`

#### Introduced in R2016b

# tuff

Time until first failure test for value-at-risk (VaR) backtesting

## Syntax

```
TestResults = tuff(vbt)
TestResults = tuff(vbt,Name,Value)
```

## Description

`TestResults = tuff(vbt)` generates the time until first failure (TUFF) test for value-at-risk (VaR) backtesting.

`TestResults = tuff(vbt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

## Examples

### Generate TUFF Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal195)
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9500
```

Generate the `tuff` test results.

```
TestResults = tuff(vbt)
```

```
TestResults =
```

```
1×9 table
```

PortfolioID	VaRID	VaRLevel	TUFF	LRatioTUFF	PValueTUFF	FirstFail
"Portfolio"	"VaR"	0.95	accept	1.7354	0.18773	58

### Run the TUFF Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,...
    [Normal195 Normal199 Historical195 Historical199 EWMA95 EWMA99],...
    'PortfolioID','Equity',...
    'VaRID',{ 'Normal195' 'Normal199' 'Historical195' 'Historical199' 'EWMA95' 'EWMA99' },
    'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =
```

```
varbacktest with properties:
```

```
PortfolioData: [1043×1 double]
VaRData: [1043×6 double]
PortfolioID: "Equity"
VaRID: [1×6 string]
VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tuff` test results using the `TestLevel` optional input.

```
TestResults = tuff(vbt, 'TestLevel', 0.90)
```

```
TestResults =
```

6×9 table

PortfolioID	VaRID	VaRLevel	TUFF	LRatioTUFF	PValueTUFF
"Equity"	"Normal195"	0.95	accept	1.7354	0.18773
"Equity"	"Normal199"	0.99	accept	0.36686	0.54472
"Equity"	"Historical195"	0.95	accept	1.5348	0.2154
"Equity"	"Historical199"	0.99	accept	0.36686	0.54472
"Equity"	"EWMA95"	0.95	accept	0.13304	0.7153
"Equity"	"EWMA99"	0.99	accept	0.14596	0.70243

- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

## Input Arguments

**vbt** — **varbacktest object**  
object

varbacktest object, contains a copy of the given data (the PortfolioData and VarData properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating a vbt object, see varbacktest.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

Example: TestResults = tuff(vbt, 'TestLevel', 0.99)

**'TestLevel'** — **Test confidence level**  
0.95 (default) | numeric between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric between 0 and 1.

Data Types: `double`

## Output Arguments

### TestResults — `tuff` test results

table

`tuff` test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- `'PortfolioID'` — Portfolio ID for the given data
- `'VaRID'` — VaR ID for each of the VaR data columns provided
- `'VaRLevel'` — VaR level for the corresponding VaR data column
- `'TUFF'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `tuff` test
- `'LRatioTUFF'` — Likelihood ratio of the `tuff` test
- `'PValueTUFF'` — P-value of the `tuff` test
- `'FirstFailure'` — Number of periods until the first failure
- `'Observations'` — Number of observations
- `'TestLevel'` — Test confidence level

---

**Note:** For `tuff` test results, the terms `accept` and `reject` are used for convenience, technically a `tuff` test does not accept a model. Rather, the test fails to reject it.

---

## Definitions

### Time Until First Failure (TUFF) Test

The `tuff` function performs Kupiec's time until first failure test.

The TUFF test is a likelihood ratio test proposed by Kupiec (1995) to assess if the number of periods until the first failure is consistent with the VaR confidence level.



## Algorithms

The likelihood ratio (test statistic) of the **tuff** test is given by

$$LRatioTUFF = -2 \log \left( \frac{pVaR (1 - pVaR)^{n-1}}{\left(\frac{1}{n}\right) \left(1 - \frac{1}{n}\right)^{n-1}} \right) = -2(\log(pVaR) + (n-1)\log(1 - pVaR) + n \log(n) - (n-1) \log(n-1))$$

where  $n$  is the number of periods until the first failure and  $pVaR = 1 - VaRLevel$ . By the properties of the logarithm (if  $n = 1$ ),

$$LRatioTUFF = -2 \log(pVaR)$$

This is asymptotically distributed as a chi-square distribution with one degree of freedom.

The  $p$ -value of the **tuff** test is the probability that a chi-square distribution with one degree of freedom exceeds the likelihood ratio  $LRatioTUFF$

$$PValueTUFF = 1 - F(LRatioTUFF)$$

where  $F$  is the cumulative distribution of a chi-square variable with one degree of freedom.

The result of the test is to accept if

$$F(LRatioTUFF) < F(TestLevel)$$

and reject otherwise, where  $F$  is the cumulative distribution of a chi-square variable with 1 degree of freedom.

If the sample has no failures, the test statistic is not defined. However, there are two cases distinguished here:

- If the number of observations is large enough that no matter when the first failure occurred it would be too late to pass the test, then the model is rejected. Technically, this happens if the number of observations  $N$  is larger than  $1/pVaR$  (large enough

relative to the VaR confidence level) and if the test fails when  $n = N + 1$  (the earliest observation for the first VaR failure). In this case, the likelihood ratio is reported for  $n = N + 1$ , and the corresponding  $p$ -value.

- In all other cases, it is not possible to tell with certainty whether the result of the test would eventually be to accept or reject the model. There are ranges of possible first failure values that would result in accepting or rejecting the model. In these cases, the `tuff` function accepts the model and reports undefined (NaN) values for the likelihood ratio and  $p$ -value.

### References

Kupiec, P. "Techniques for Verifying the Accuracy of Risk Management Models." *Journal of Derivatives*. Vol. 3, 1995, pp. 73–84.

### See Also

#### See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `t1` | `varbacktest`

#### Topics

“VaR Backtesting Workflow” on page 2-8

“Value-at-Risk Estimation and Backtesting” on page 2-13

“Kupiec’s POF and TUFF Tests” on page 2-4

`varbacktest`

**Introduced in R2016b**