

# Risk Management Toolbox™

## User's Guide



# MATLAB®

R2020a



# 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.  
1 Apple Hill Drive  
Natick, MA 01760-2098

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

© COPYRIGHT 2016 - 2020 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)
September 2017	Online only	Revised for Version 1.2 (Release 2017b)
March 2018	Online only	Revised for Version 1.3 (Release 2018a)
September 2018	Online only	Revised for Version 1.4 (Release 2018b)
March 2019	Online only	Revised for Version 1.5 (Release 2019a)
September 2019	Online only	Revised for Version 1.6 (Release 2019b)
March 2020	Online only	Revised for Version 1.7 (Release 2020a)

## Getting Started

### 1

<b>Risk Management Toolbox Product Description</b> .....	<b>1-2</b>
Key Features .....	<b>1-2</b>
<b>Risk Modeling with Risk Management Toolbox</b> .....	<b>1-3</b>
Consumer Credit Risk .....	<b>1-3</b>
Corporate Credit Risk .....	<b>1-3</b>
Market Risk .....	<b>1-5</b>
<b>Credit Rating Migration Risk</b> .....	<b>1-7</b>
<b>Default Probability by Using the Merton Model for Structural Credit Risk</b> .....	<b>1-10</b>
<b>Concentration Indices</b> .....	<b>1-12</b>

## Market Risk Measurements Using VaR BackTesting Tools

### 2

<b>Overview of VaR Backtesting</b> .....	<b>2-2</b>
Binomial Test .....	<b>2-2</b>
Traffic Light Test .....	<b>2-3</b>
Kupiec's POF and TUFF Tests .....	<b>2-3</b>
Christoffersen's Interval Forecast Tests .....	<b>2-4</b>
Haas's Time Between Failures or Mixed Kupiec's Test .....	<b>2-4</b>
<b>VaR Backtesting Workflow</b> .....	<b>2-6</b>
<b>Value-at-Risk Estimation and Backtesting</b> .....	<b>2-10</b>
<b>Overview of Expected Shortfall Backtesting</b> .....	<b>2-21</b>
Conditional Test by Acerbi and Szekely .....	<b>2-22</b>
Unconditional Test by Acerbi and Szekely .....	<b>2-23</b>
Quantile Test by Acerbi and Szekely .....	<b>2-23</b>
ES Backtest Using Du-Escanciano Method .....	<b>2-24</b>
Comparison of ES Backtesting Methods .....	<b>2-26</b>
<b>Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information</b> .....	<b>2-29</b>
<b>Expected Shortfall (ES) Backtesting Workflow Using Simulation</b> .....	<b>2-33</b>

## 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 .....	3-6
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 .....	3-14
Change Bin Boundaries for Multiple Predictors .....	3-15
Set Options for Display .....	3-16
Export and Save the Binning .....	3-17
Troubleshoot the Binning .....	3-17
<b>Binning Explorer Case Study Example</b> .....	<b>3-21</b>
<b>Stress Testing of Consumer Credit Default Probabilities Using Panel Data</b> .....	<b>3-34</b>

## Corporate Credit Risk Simulations for Portfolios

### 4

<b>Credit Simulation Using Copulas</b> .....	<b>4-2</b>
Factor Models .....	4-2
Supported Simulations .....	4-3
<b>creditDefaultCopula Simulation Workflow</b> .....	<b>4-5</b>
<b>creditMigrationCopula Simulation Workflow</b> .....	<b>4-10</b>
<b>Modeling Correlated Defaults with Copulas</b> .....	<b>4-18</b>
<b>Analyze the Sensitivity of Concentration to a Given Exposure</b> .....	<b>4-28</b>
<b>Compare Concentration Indices for Random Portfolios</b> .....	<b>4-30</b>
<b>Comparison of the Merton Model Single-Point Approach to the Time- Series Approach</b> .....	<b>4-33</b>
<b>Calculating Regulatory Capital with the ASRF Model</b> .....	<b>4-38</b>





# 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 1-7
- “Default Probability by Using the Merton Model for Structural Credit Risk” on page 1-10
- “Concentration Indices” on page 1-12

## **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) and expected shortfall (ES).

### **Key Features**

- Binning Explorer app for developing credit scorecards
- Credit risk simulation using copulas
- Probability of Default (PD) estimation using Merton model
- Concentration risk indices for identifying and controlling large exposure
- Capital calculations using the ASRF model
- Value-at-Risk (VaR) and expected shortfall (ES) 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 you 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 1-7.

Related to the `creditDefaultCopula` and `creditMigrationCopula` objects, Risk Management Toolbox provides an analytical model known as the Asymptotic Single Risk Factor (ASRF) model. The

ASRF model is useful because the Basel II documents propose this model as the standard for certain types of capital requirements. ASRF is not a Monte-Carlo model, so you can quickly compute the capital requirements for large credit portfolios. You can use the ASRF model to perform a quick sensitivity analysis and exploring "what-if" scenarios more easily than rerunning large simulations. For more information, see `asrf`.

Risk Management Toolbox also provides tools for portfolio concentration analysis, see "Concentration Indices" on page 1-12.

## 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.

VaR 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.

Expected Shortfall (ES) Backtesting gives an estimate of the loss in those very bad days when the VaR is violated. ES is the expected loss on days when there is a VaR failure. If the VaR is 10 million and the ES is 12 million, you know that the expected loss tomorrow, if it happens to be a very bad day, is about 20% higher than the VaR.

Risk Management Toolbox provides the following table-based tests for expected shortfall based on the `esbacktest` object:

- `unconditionalNormal`
- `unconditionalT`

The following tools support expected shortfall simulation-based tests for the `esbacktestbysim` object:

- `conditional`

- unconditional
- quantile

For information on the different tests, see “Overview of Expected Shortfall Backtesting” on page 2-21.

## See Also

asrf | concentrationIndices | creditDefaultCopula | creditMigrationCopula | esbacktest | esbacktestbysim | mertonByTimeSeries | mertonmodel | varbacktest

## Related Examples

- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-21
- “creditMigrationCopula Simulation Workflow” on page 4-10
- “creditDefaultCopula Simulation Workflow” on page 4-5
- “Modeling Correlated Defaults with Copulas” on page 4-18
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-34
- “VaR Backtesting Workflow” on page 2-6
- “Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29
- “Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33
- “Expected Shortfall Estimation and Backtesting”
- “Value-at-Risk Estimation and Backtesting” on page 2-10

## More About

- “Credit Simulation Using Copulas” on page 4-2
- “Credit Rating Migration Risk” on page 1-7
- “Default Probability by Using the Merton Model for Structural Credit Risk” on page 1-10
- “Concentration Indices” on page 1-12
- “Traffic Light Test” on page 2-3
- “Binomial Test” on page 2-2
- “Kupiec’s POF and TUFF Tests” on page 2-3
- “Christoffersen’s Interval Forecast Tests” on page 2-4
- “Haas’s Time Between Failures or Mixed Kupiec’s Test” on page 2-4
- “Overview of Expected Shortfall Backtesting” on page 2-21

## External Websites

- Introduction to Risk Management Toolbox (26 min 24 sec)
- Credit Scorecard Modeling Using the Binning Explorer App (6 min 17 sec)
- Credit Risk Modeling with MATLAB (53 min 09 sec)
- Forecasting Corporate Default Rates with MATLAB (54 min 36 sec)

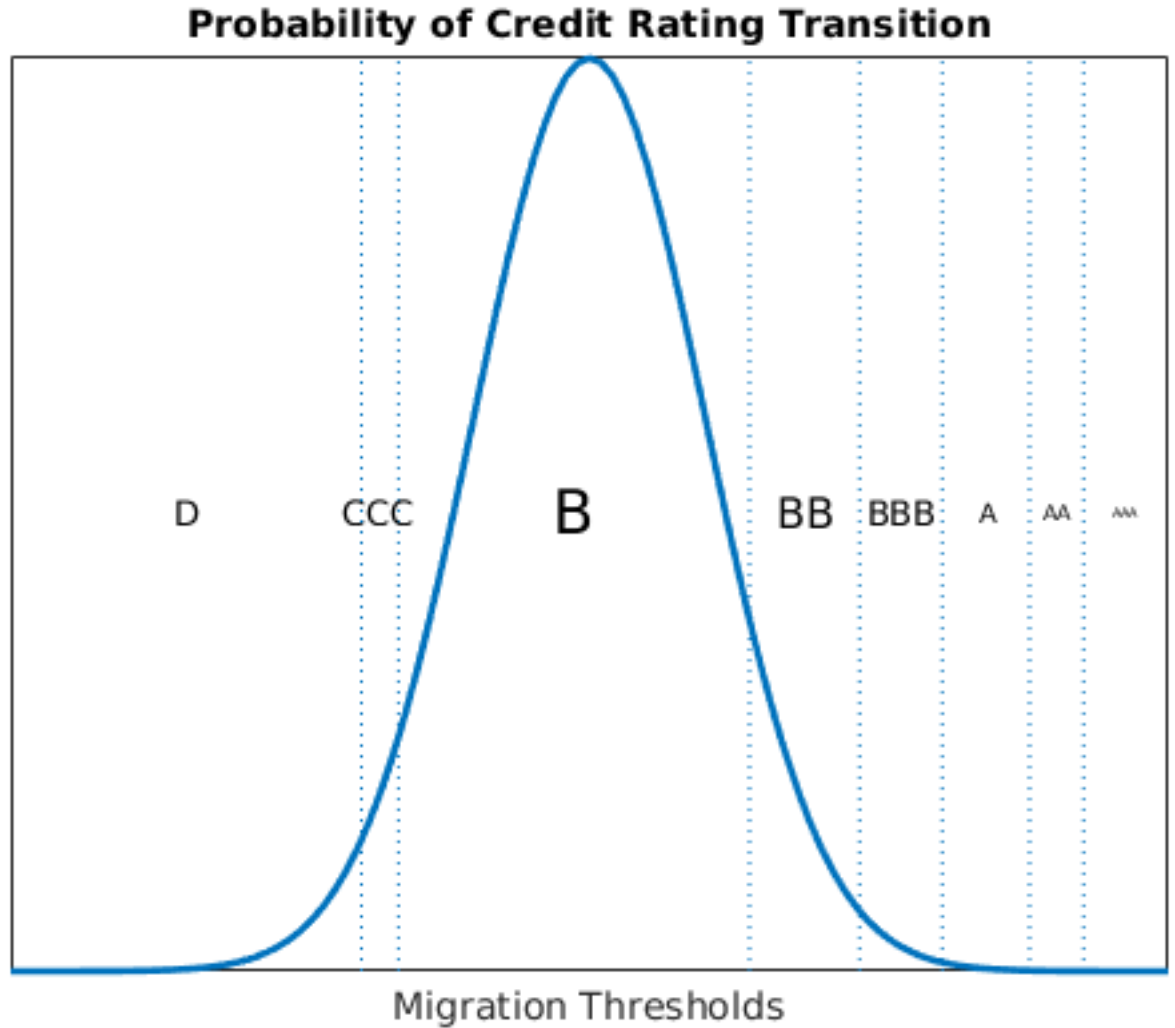
## Credit Rating Migration Risk

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.

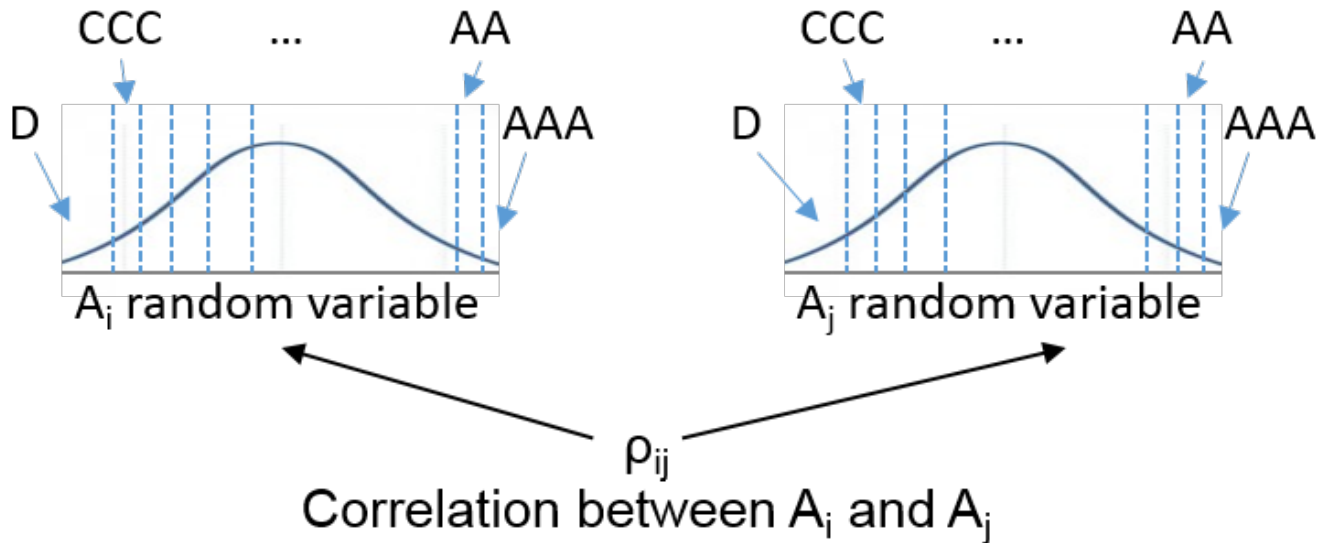
Given enough historical data, the likelihood is calculated that a company at a particular rating will migrate to a different rating over some time period. For example, this table shows the probabilities that a company with credit rating "B" will transition to each other rating.

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 4-10

## 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. The Merton model uses the Black-Scholes-Merton option pricing methods and is structural because it provides a relationship between the default risk and the asset (capital) structure of the firm.

A company balance sheet records book values—the value of a firm's equity  $E$ , its total assets  $A$ , and its total liabilities  $L$ . The relationship between these values is defined by the equation

$$A = E + L$$

These book values for  $E$ ,  $A$ , and  $L$  are all observable because they are recorded on a firm's balance sheet. However, the book values are reported infrequently. Alternatively, only the equity's market value is observable, and is 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.

The Merton model relates the market values of equity, assets, and liabilities in an option pricing framework. The Merton model assumes a single liability  $L$  with maturity  $T$ , usually a period of one year or less. At time  $T$ , the firm's value to the shareholders equals the difference  $A - L$  when the asset value  $A$  is greater than the liabilities  $L$ . However, if the liabilities  $L$  exceed 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 to a value 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  $T$ :

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

In this equation,  $r$  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 this equation using one of two approaches:

- The mertonmodel approach uses single-point calibration and requires values for the 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 aforementioned option pricing formula. The second equation relates the unobservable volatility of assets  $\sigma_A$  to the given equity volatility  $\sigma_E$ :

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

- The `mertonByTimeSeries` approach requires 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:

$$E_1 = A_1 N(d_1) - L_1 e^{-r_1 T_1} N(d_2)$$

...

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

The function directly computes the volatility of assets  $\sigma_A$  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.

After computing the values of  $A$  and  $\sigma_A$ , the function computes 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 of the asset value falling 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 4-33

## 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 4-28
- “Compare Concentration Indices for Random Portfolios” on page 4-30

# Market Risk Measurements Using VaR BackTesting Tools

---

- “Overview of VaR Backtesting” on page 2-2
- “VaR Backtesting Workflow” on page 2-6
- “Value-at-Risk Estimation and Backtesting” on page 2-10
- “Overview of Expected Shortfall Backtesting” on page 2-21
- “Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29
- “Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33

## Overview of VaR Backtesting

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 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.

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 “References” on page 2-5 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 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 is a 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 “References” on page 2-5 for Basel Committee on Banking Supervision and `tl`.

## 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 1 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 TUFF test extends the TUFF approach to include all the failures. See `tbfi`.

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 1 degree of freedom. For more information, see “References” on page 2-5 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

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

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 1 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 2 degrees of freedom.

For more information, see “References” on page 2-5 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.

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

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:

$$LR_{TBF} = LR_{POF} + LR_{TBF1}$$

This test is asymptotically distributed as a chi-square variable with  $\chi+1$  degrees of freedom. For more information, see "References" on page 2-5 for Haas, `tbf`, and `tbfi`.

## References

- [1] 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, <https://www.bis.org/publ/bcbs22.htm>.
- [2] Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841-862.
- [3] Cogneau, P. "Backtesting Value-at-Risk: how good is the model?" *Intelligent Risk*, PRMIA, July, 2015.
- [4] Haas, M. "New Methods in Backtesting." *Financial Engineering*, Research Center Caesar, Bonn, 2001.
- [5] Jorion, P. *Financial Risk Manager Handbook. 6th Edition*, Wiley Finance, 2011.
- [6] Kupiec, P. "Techniques for Verifying the Accuracy of Risk Management Models." *Journal of Derivatives*. Vol. 3, 1995, pp. 73-84.
- [7] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management*. Princeton University Press, 2005.
- [8] 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` | `tl` | `tuff` | `varbacktest`

## Related Examples

- "Value-at-Risk Estimation and Backtesting" on page 2-10

## More About

- "Risk Modeling with Risk Management Toolbox" on page 1-3
- "Overview of VaR Backtesting" on page 2-2

## 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” on page 2-10.

### 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`, `Normal95`, and `Normal99` 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” on page 2-10 for an example on how to generate this VaR data.

```
load('VaRBacktestData')
disp([EquityIndex(1:5) Normal95(1:5) Normal99(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
```

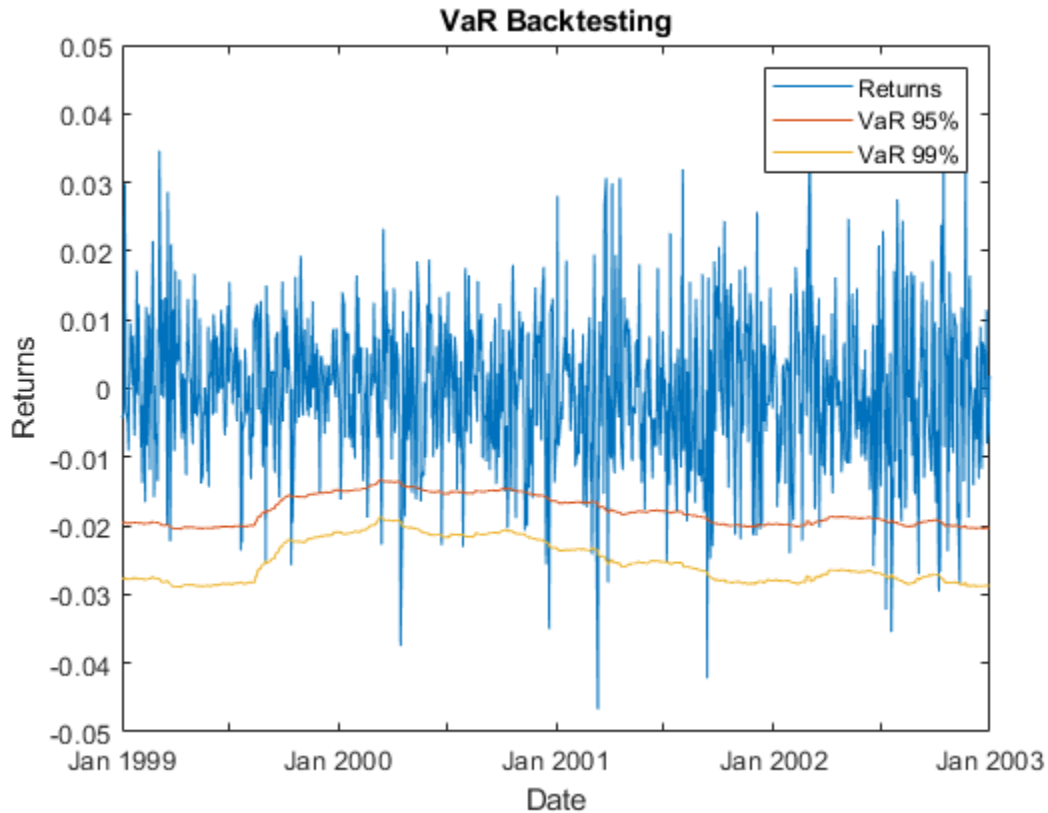
The first column shows three losses in the first three days, but none of these losses exceeds the corresponding VaR (columns 2 and 3). The VaR model fails whenever the loss (negative of returns) exceeds the VaR.

### 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 -Normal95 -Normal99])
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,[Normal95 Normal99],...
    'PortfolioID','S&P', ...
    'VaRID',{'Normal95' 'Normal99'}, ...
    'VaRLevel',[0.95 0.99]);
disp(vbt)
```

`varbacktest` with properties:

```
PortfolioData: [1043x1 double]
VaRData: [1043x2 double]
PortfolioID: "S&P"
VaRID: ["Normal95" "Normal99"]
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 Expect
```

"S&P"	"Normal95"	0.95	0.94535	1043	57	52.15
"S&P"	"Normal99"	0.99	0.9837	1043	17	10.43

**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	CC	
"S&P"	"Normal95"	0.95	green	accept	accept	accept	accept	a
"S&P"	"Normal99"	0.99	yellow	reject	accept	accept	accept	a

**Step 6. Run individual tests.**

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

```
tl(vbt)
```

```
ans=2x9 table
```

PortfolioID	VaRID	VaRLevel	TL	Probability	TypeI	Increase	Obs
"S&P"	"Normal95"	0.95	green	0.77913	0.26396	0	
"S&P"	"Normal99"	0.99	yellow	0.97991	0.03686	0.26582	

**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),[Normal95(Ind1,:) Normal99(Ind1:)],...
    'PortfolioID','S&P, 1999-2000',...
    'VaRID',{'Normal95' 'Normal99'},...
    'VaRLevel',[0.95 0.99]);

vbt2 = varbacktest(EquityIndex(Ind2),[Normal95(Ind2,:) Normal99(Ind2:)],...
    'PortfolioID','S&P, 2001-2002',...
    'VaRID',{'Normal95' 'Normal99'},...
    '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	Failures	Ex
"S&P, 1999-2000"	"Normal95"	0.95	0.94626	521	28	
"S&P, 1999-2000"	"Normal99"	0.99	0.98464	521	8	
"S&P, 2001-2002"	"Normal95"	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	CC
"S&P, 1999-2000"	"Normal95"	0.95	green	accept	accept	accept	accept
"S&P, 1999-2000"	"Normal99"	0.99	green	accept	accept	accept	accept
"S&P, 2001-2002"	"Normal95"	0.95	green	accept	accept	accept	accept
"S&P, 2001-2002"	"Normal99"	0.99	yellow	accept	accept	accept	accept

### See Also

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

### Related Examples

- "Value-at-Risk Estimation and Backtesting" on page 2-10

### More About

- "Traffic Light Test" on page 2-3
- "Binomial Test" on page 2-2
- "Kupiec's POF and TUFF Tests" on page 2-3
- "Christoffersen's Interval Forecast Tests" on page 2-4
- "Haas's Time Between Failures or Mixed Kupiec's Test" on page 2-4

## Value-at-Risk Estimation and Backtesting

This example shows how to estimate the value-at-risk (VaR) using three methods and 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 from 1993 through 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);
```

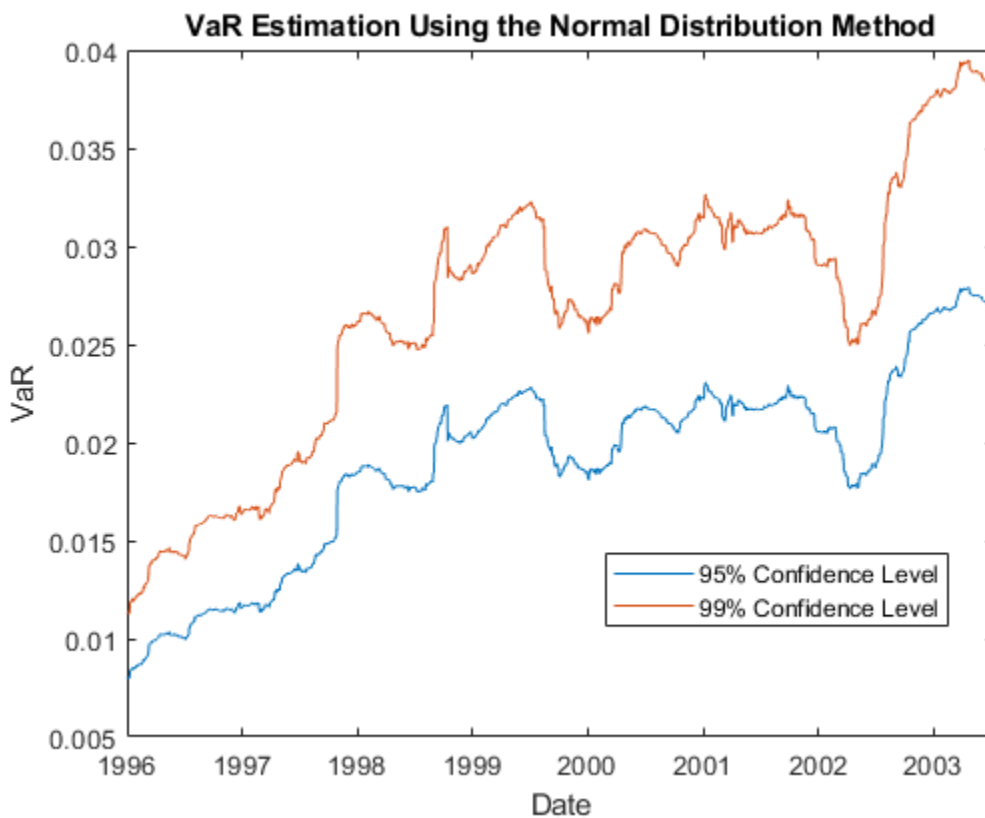
```

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."

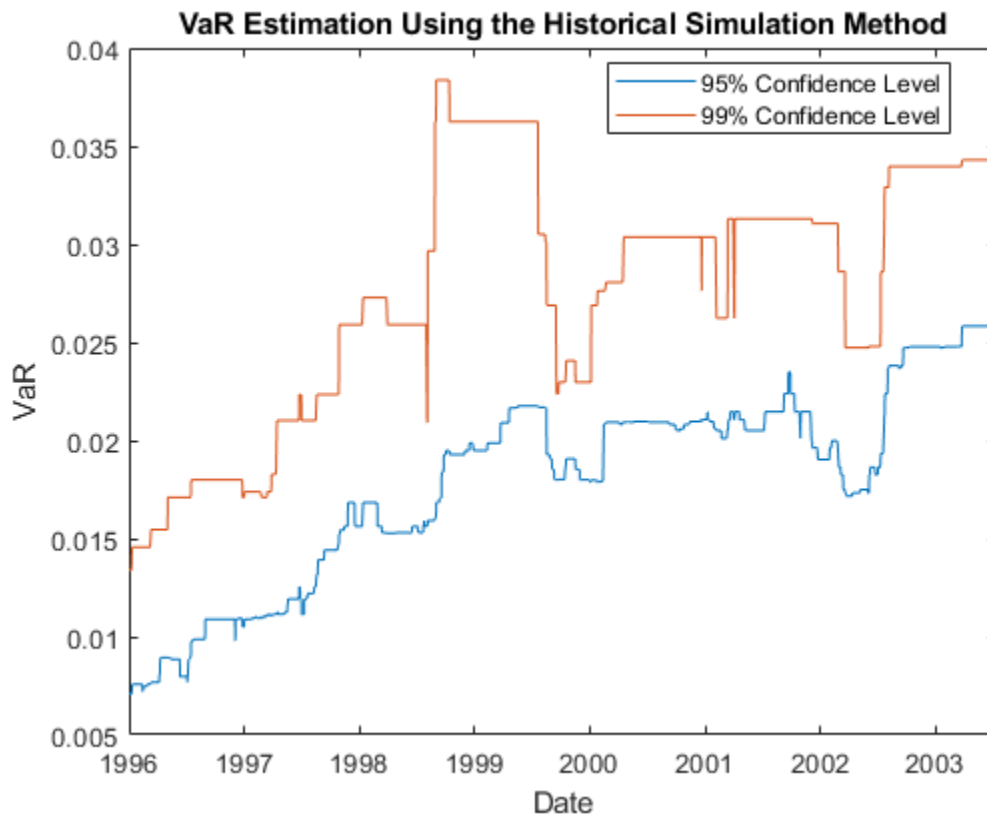
```

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;

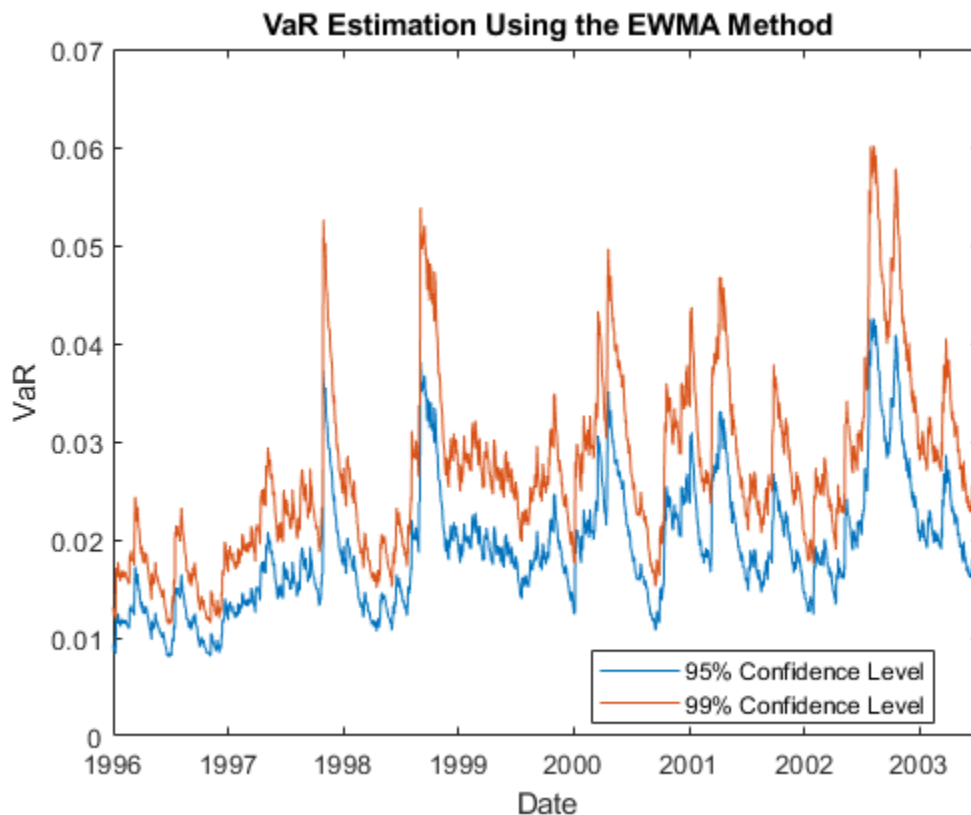
for i = 2 : (TestWindowStart-1)
    Sigma2(i) = (1-Lambda) *>Returns(i-1)^2 + Lambda * Sigma2(i-1);
end
```

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(Date>Returns(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.

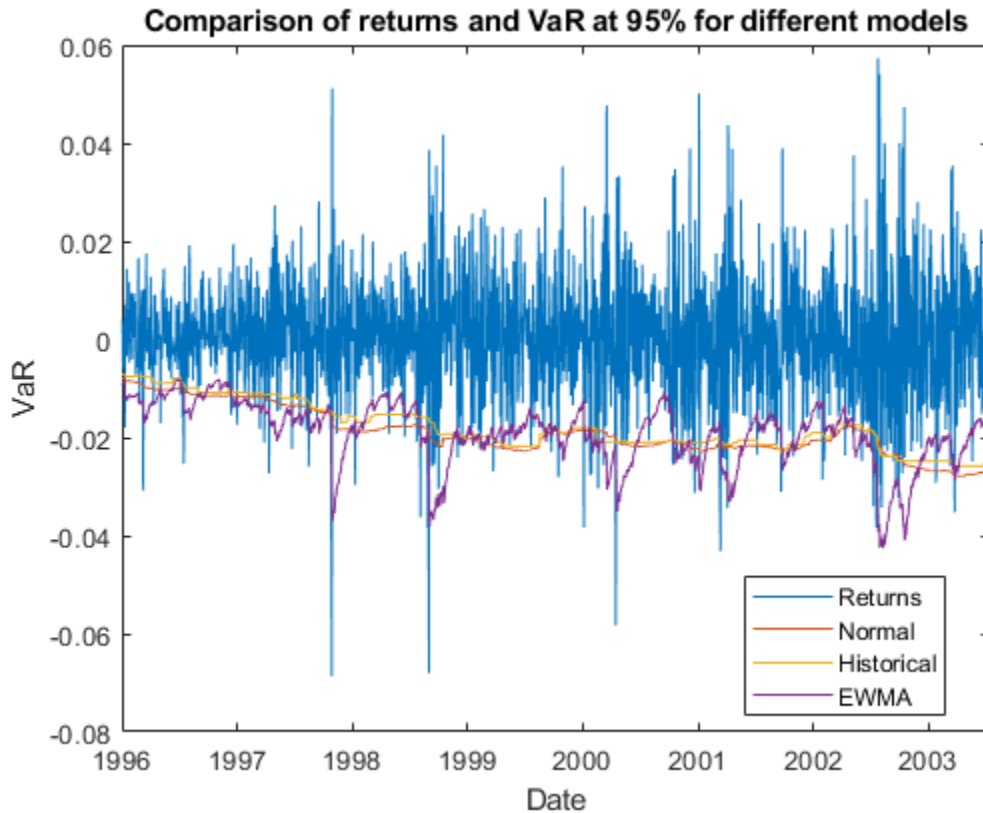
### VaR Backtesting

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')
```

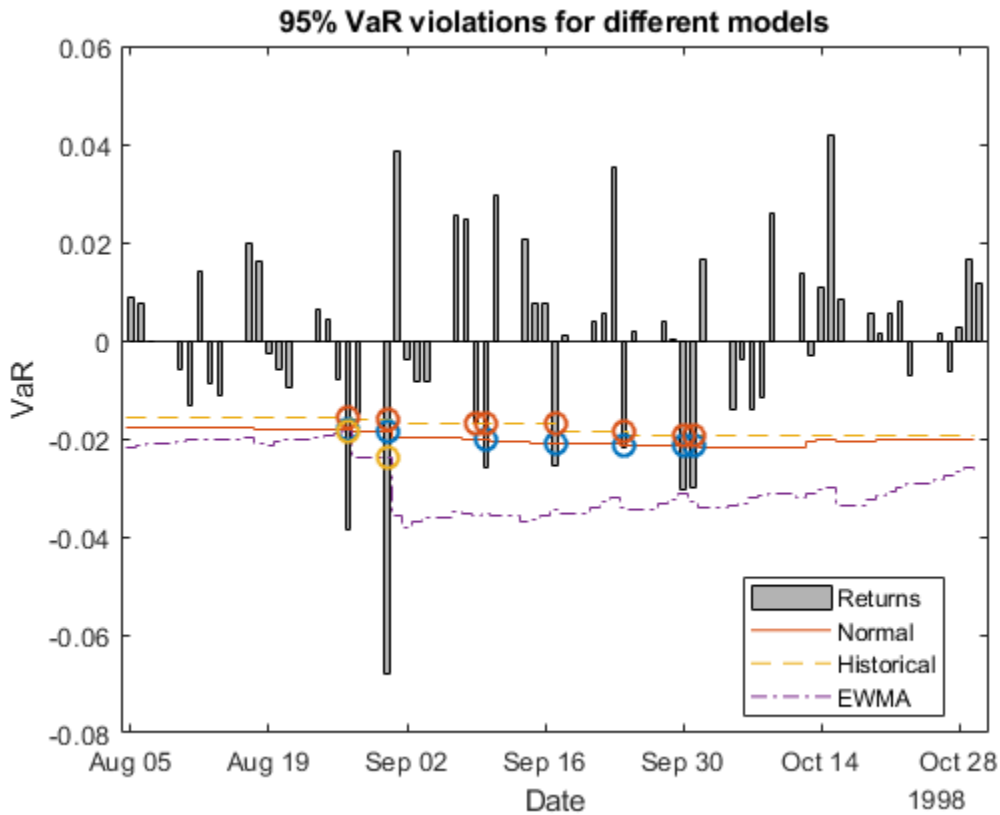




To highlight how the different approaches react differently to changing market conditions, you can zoom in on the time series where there is a large and sudden change in the value of returns. For example, around August 1998:

```
ZoomInd = (DatesTest >= datestr('5-Aug-1998','local')) & (DatesTest <= datestr('31-Oct-1998','local'));
VaRData = [-Normal95(ZoomInd) -Historical95(ZoomInd) -EWMA95(ZoomInd)];
VaRFormat = {'-', '--', '-.'};
D = DatesTest(ZoomInd);
R = ReturnsTest(ZoomInd);
N = Normal95(ZoomInd);
H = Historical95(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 27 to August 31 shows a significant dip in the returns. On the dates starting from August 27 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', ...
    {'Normal95', 'Normal99'}, 'VaRLevel', [0.95 0.99]);
summary(vbt)
```

```
ans =
    2x10 table
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Failures	Expected
"S&P"	"Normal95"	0.95	0.94863	1966	101	98.3
"S&P"	"Normal99"	0.99	0.98372	1966	32	19.66

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 Normal95 VaR level is within range, whereas the Normal99 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	CC	CCI	TBFI
"S&P"	"Normal95"	0.95	green	accept	accept	accept	accept	accept	accept
"S&P"	"Normal99"	0.99	yellow	reject	reject	accept	reject	reject	reject

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,[Normal95 Historical95 EWMA95 Normal99 Historical99 ...
EWMA99], 'PortfolioID', 'S&P', 'VaRID', {'Normal95', 'Historical95', 'EWMA95', ...
'Normal99', 'Historical99', 'EWMA99'}, 'VaRLevel', [0.95 0.95 0.95 0.99 0.99 0.99]);
runtests(vbt)
```

```
ans =
```

```
6x11 table
```

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF	CC	CCI	TBFI
"S&P"	"Normal95"	0.95	green	accept	accept	accept	accept	accept	accept
"S&P"	"Historical95"	0.95	yellow	accept	accept	accept	accept	accept	accept
"S&P"	"EWMA95"	0.95	green	accept	accept	accept	accept	accept	accept
"S&P"	"Normal99"	0.99	yellow	reject	reject	accept	reject	reject	reject
"S&P"	"Historical99"	0.99	yellow	reject	reject	accept	reject	reject	reject

```
"S&P"      "EWMA99"      0.99      red      reject      reject      accept      reject
```

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),[Normal95(Ind2000) Historical95(Ind2000) EWMA95(Ind2000)
'PortfolioID', 'S&P, 2000', 'VaRID', {'Normal', 'Historical', 'EWMA'});
runtests(vbt2000)
```

ans =

3x11 table

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

```
Ind2002 = (year(DatesTest) == 2002);
vbt2002 = varbacktest>ReturnsTest(Ind2002),[Normal95(Ind2002) Historical95(Ind2002) EWMA95(Ind2002)
'PortfolioID', 'S&P, 2002', 'VaRID', {'Normal', 'Historical', 'EWMA'});
runtests(vbt2002)
```

ans =

3x11 table

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

3x13 table

PortfolioID	VaRID	VaRLevel	CCI	LRatioCCI	PValueCCI	Observations
"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	Observations
"S&P, 2002"	"Normal"	0.95	reject	53.936	0.00010087	261
"S&P, 2002"	"Historical"	0.95	reject	45.274	0.0010127	261
"S&P, 2002"	"EWMA"	0.95	reject	25.756	0.027796	261

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](#) | [tbfi](#) | [tl](#) | [tuff](#) | [varbacktest](#)

### Related Examples

- “VaR Backtesting Workflow” on page 2-6

### More About

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

## Overview of Expected Shortfall Backtesting

Expected Shortfall (ES) is the expected loss on days when there is a Value-at-Risk (VaR) failure. If the VaR is 10 million and the ES is 12 million, we know the expected loss tomorrow; if it happens to be a very bad day, it is 20% higher than the VaR. ES is sometimes called Conditional Value-at-Risk (CVaR), Tail Value-at-Risk (TVaR), Tail Conditional Expectation (TCE), or Conditional Tail Expectation (CTE).

There are many approaches to estimating VaR and ES, and they may lead to different VaR and ES estimates. How can one determine if models are accurately estimating the risk on a daily basis? How can one evaluate which model performs better? The `varbacktest` tools help validate the performance of VaR models with regards to estimated VaR values. The `esbacktest`, `esbacktestbysim`, and `esbacktestbyde` tools extend these capabilities to evaluate VaR models with regards to estimated ES values.

For VaR backtesting, the possibilities every day are two: either there is a VaR failure or not. If the VaR confidence level is 95%, VaR failures should happen approximately 5% of the time. To backtest VaR, you only need to know whether the VaR was exceeded (VaR failure) or not on each day of the test window and the VaR confidence level. Risk Management Toolbox VaR backtesting tools support “frequency” (assess the proportion of failures) and “independence” (assess independence across time) tests, and these tests work with the binary sequence of “failure” or “no-failure” results over the test window.

For expected shortfall (ES), the possibilities every day are infinite: The VaR may be exceeded by 1%, or by 10%, or by 150%, and so on. For example, there are three VaR failures in the following example:

Failure Data				Severity Ratio	
Date	Return	VaR	ES	Observed (-Return/VaR)	Expected (ES/VaR)
26-Feb-96	-1.308	1.078	1.364	1.21	1.27
8-Mar-96	-2.051	1.110	1.404	1.85	1.27
10-Apr-96	-1.353	1.218	1.541	1.11	1.27
Average				1.39	1.27

On failure days, the VaR is exceeded on average by 39%, but the estimated ES exceeds VaR by an average of 27%. How can you tell if 39% is significantly larger than 27%? Knowing the VaR confidence level is not enough, you must also know how likely are the different exceedances over the VaR according to the VaR model. In other words, you need some distribution information about what happens beyond the VaR according to your model assumptions. For thin-tail VaR models, 39% vs. 27% may be a large difference. However, for a heavy-tail VaR model where a severity of twice the VaR has a non-trivial probability of happening, then 39% vs. 27% over the three failure dates may not be a red flag.

A key difference between VaR backtesting and ES backtesting is that most ES backtesting methods require information about the distribution of the returns on each day, or at least the distribution of the tails beyond the VaR. One exception is the “unconditional” test (see `unconditionalNormal` and `unconditionalT`) where you can get approximate test results without providing the distribution information. This is important in practice, because the “unconditional” test is much simpler to use and can be used in principle for any VaR or ES model. The trade-off is that the approximate results may be inaccurate, especially in borderline accept, or reject cases, or for certain types of distributions.

The toolbox supports the following tests for expected shortfall backtesting for table-based tests for the unconditional Acerbi-Szekely test using the `esbacktest` object:

- `unconditionalNormal`
- `unconditionalT`

The toolbox supports the following Acerbi-Szekely simulation-based tests for expected shortfall backtesting using the `esbacktestbysim` object:

- `conditional`
- `unconditional`
- `quantile`

For the Acerbi-Szekely simulation-based tests, you must provide the model distribution information as part of the inputs to `esbacktestbysim`.

The toolbox also supports the following Du and Escanciano tests for expected shortfall backtesting using the `esbacktestbyde` object:

- `unconditionalDE`
- `conditionalDE`

For the Du and Escanciano simulation-based tests, you must provide the model distribution information as part of the inputs to `esbacktestbyde`.

## Conditional Test by Acerbi and Szekely

The conditional test statistic by Acerbi and Szekely is based on the conditional relationship

$$ES_t = -E_t[X_t | X_t < -VaR_t]$$

where

$X_t$  is the portfolio outcome, that is, the portfolio return or portfolio profit and loss for period  $t$ .

$VaR_t$  is the estimated VaR for period  $t$ .

$ES_t$  is the estimated expected shortfall for period  $t$ .

The number of failures is defined as

$$NumFailures = \sum_{t=1}^N I_t$$

where

$N$  is the number of periods in the test window ( $t = 1, \dots, N$ ).

$I_t$  is the VaR failure indicator on period  $t$  with a value of 1 if  $X_t < -VaR_t$ , and 0 otherwise.

The conditional test statistic is defined as

$$Z_{cond} = \frac{1}{NumFailures} \sum_{t=1}^N \frac{X_t I_t}{ES_t} + 1$$



The conditional test has two parts. A VaR backtest must be run for the number of failures (`NumFailures`), and a standalone conditional test is performed for the conditional test statistic  $Z_{\text{cond}}$ . The conditional test accepts the model only when both the VaR test and the standalone conditional test accept the model. For more information, see `conditional`.

## Unconditional Test by Acerbi and Szekely

The unconditional test statistic by Acerbi and Szekely is based on the unconditional relationship,

$$ES_t = -E_t \left[ \frac{X_t I_t}{P_{VaR}} \right]$$

where

$X_t$  is the portfolio outcome, that is, the portfolio return or portfolio profit and loss for period  $t$ .

$P_{VaR}$  is the probability of VaR failure defined as 1-VaR level.

$ES_t$  is the estimated expected shortfall for period  $t$ .

$I_t$  is the VaR failure indicator on period  $t$  with a value of 1 if  $X_t < -VaR$ , and 0 otherwise.

The unconditional test statistic is defined as

$$Z_{\text{uncond}} = \frac{1}{N P_{VaR}} \sum_{t=1}^N \frac{X_t I_t}{ES_t} + 1$$

The critical values for the unconditional test statistic are stable across a range of distributions, which is the basis for the table-based tests. The `esbacktest` class runs the unconditional test against precomputed critical values under two distributional assumptions, namely, normal distribution (thin tails, see `unconditionalNormal`), and  $t$  distribution with 3 degrees of freedom (heavy tails, see `unconditionalT`).

## Quantile Test by Acerbi and Szekely

A sample estimator of the expected shortfall for a sample  $Y_1, \dots, Y_N$  is:

$$\widehat{ES}(Y) = - \frac{1}{[N P_{VaR}]} \sum_{i=1}^{[N P_{VaR}]} Y_{[i]}$$

where

$N$  is the number of periods in the test window ( $t = 1, \dots, N$ ).

$P_{VaR}$  is the probability of VaR failure defined as 1-VaR level.

$Y_1, \dots, Y_N$  are the sorted sample values (from smallest to largest), and  $[N P_{VaR}]$  is the largest integer less than or equal to  $N P_{VaR}$ .

To compute the quantile test statistic, a sample of size  $N$  is created at each time  $t$  as follows. First, convert the portfolio outcomes to  $X_t$  to ranks  $U_1 = P_1(X_1), \dots, U_N = P_N(X_N)$  using the cumulative distribution function  $P_t$ . If the distribution assumptions are correct, the rank values  $U_1, \dots, U_N$  are uniformly distributed in the interval (0,1). Then at each time  $t$ :

- 1 Invert the ranks  $U = (U_1, \dots, U_N)$  to get  $N$  quantiles  $P_t^{-1}(U) = (P_t^{-1}(U_1), \dots, P_t^{-1}(U_N))$ .
- 2 Compute the sample estimator  $\widehat{ES}(P_t^{-1}(U))$ .
- 3 Compute the expected value of the sample estimator  $E[\widehat{ES}(P_t^{-1}(V))]$

where  $V = (V_1, \dots, V_N)$  is a sample of  $N$  independent uniform random variables in the interval  $(0,1)$ . This can be computed analytically.

The quantile test statistic by Acerbi and Szekely is defined as

$$Z_{quantile} = -\frac{1}{N} \sum_{t=1}^N \frac{\widehat{ES}(P_t^{-1}(U))}{E[\widehat{ES}(P_t^{-1}(V))]} + 1$$

The denominator inside the sum can be computed analytically as

$$E[\widehat{ES}(P_t^{-1}(V))] = -\frac{N}{[N_{pVaR}]} \int_0^1 I_{1-p}(N - [N_{pVaR}], [N_{pVaR}]) P_t^{-1}(p) dp$$

where  $I_x(z,w)$  is the regularized incomplete beta function. For more information, see `betainc` and `quantile`.

## ES Backtest Using Du-Escanciano Method

For each day, the Du-Escanciano model assumes a distribution for the returns. For example, if you have a normal distribution with a conditional variance of 1.5%, there is a corresponding cumulative distribution function  $P_t$ . By mapping the returns  $X_t$  with the distribution  $P_t$ , you get the “mapped returns” series  $U_t$ , also known as the “ranks” series, which by construction has values between 0 and 1 (see column 2 in the following table). Let  $\alpha$  be the complement of the VaR level — for example, if the VaR level is 95%,  $\alpha$  is 5%. If the mapped return  $U_t$  is smaller than  $\alpha$ , then there is a VaR “violation” or VaR “failure.” This is equivalent to observing a return  $X_t$  smaller than the negative of the VaR value for that day, since, by construction, the negative of the VaR value gets mapped to  $\alpha$ . Therefore, you can compare  $U_t$  against  $\alpha$  without even knowing the VaR value. The series of VaR failures is denoted by  $h_t$  and it is a series of 0's and 1's stored in column 3 in the following table. Finally, column 4 in the following table contains the “cumulative violations” series, denoted by  $H_t$ . This is the severity of the mapped VaR violations on days on which the VaR is violated. For example, if the mapped return  $U_t$  is 1% and  $\alpha$  is 5%,  $H_t$  is 4%.  $H_t$  is defined as zero if there are no VaR violations.

$X_t$	$U_t = P_t(X_t)$	$h_t = U_t < \alpha$	$H_t = (\alpha - U_t) * h_t$
0.00208	0.5799	0	0
-0.01073	0.1554	0	0
-0.00825	0.2159	0	0
-0.02967	0.0073	1	0.0427
0.01242	0.8745	0	0
...	...	...	...

Given the violations series  $h_t$  and the cumulative violations series  $H_t$ , the Du-Escanciano (DE) tests are summarized as:

Du-Escanciano Test	VaR Test	ES Test
Unconditional	Mean of $h_t$	Mean of $H_t$
Conditional	Autocorrelation of $h_t$	Autocorrelation of $H_t$

The DE VaR tests assess the mean value and the autocorrelation of the  $h_t$  series, and the resulting tests overlap with known VaR tests. For example, the mean of  $h_t$  is expected to match  $\alpha$ . In other words, the proportion of time the VaR is violated is expected to match the confidence level. This test is supported in the `varbacktest` class with the proportion of failures (`po f`) test (finite sample) and the binomial (`bin`) test (large-sample approximation). In turn, the conditional VaR test measures if there is a time pattern in the sequence of VaR failures (back-to-back failures, and so on). The conditional coverage independence (`cci`) test in the `varbacktest` class tests for one-lag independence. The time between failures independence (`tbfi`) test in the `varbacktest` class also assesses time independence for VaR models.

The `esbacktestbyde` class supports the DE ES tests. The DE ES tests assess the mean value and the autocorrelation of the  $H_t$  series. For the unconditional test (`unconditionalDE`), the expected value is  $\alpha/2$  — for example, the average value in the bottom 5% of a uniform (0,1) distribution is 2.5%. The conditional test (`conditionalDE`) assesses not only if a failure occurs but also if the failure severity is correlated to previous failure occurrences and their severities.

The test statistic for the unconditional DE ES test is

$$U_{ES} = \frac{1}{N} \sum_{t=1}^N H_t$$

If the number of observations is large, the test statistic is distributed as

$$U_{ES} \xrightarrow{dist} N\left(\frac{\alpha}{2}, \frac{\alpha(1/3 - \alpha/4)}{N}\right) = P_U$$

where  $N(\mu, \sigma^2)$  is the normal distribution with mean  $\mu$  and variance  $\sigma^2$ .

The unconditional DE ES test is a two-sided test that checks if the test statistic is close to the expected value of  $\alpha/2$ . From the limiting distribution, a confidence level is derived. Finite-sample confidence intervals are estimated through simulation.

The test statistic for the conditional DE ES test is derived in several steps. First, define the autocovariance for lag  $j$ :

$$\gamma_j = \frac{1}{N-j} \sum_{t=j+1}^N (H_t - \alpha/2)(H_{t-j} - \alpha/2)$$

The autocorrelation for lag  $j$  is then

$$\rho_j = \frac{\gamma_j}{\gamma_0}$$

The test statistic for  $m$  lags is then

$$C_{ES}(m) = N \sum_{j=1}^m \rho_j^2$$

If the number of observations is large, the test statistic is distributed as a chi-square distribution with  $m$  degrees of freedom:

$$C_{ES}(m) \xrightarrow{dist} \chi_m^2$$

The conditional DE ES test is a one-sided test to determine if the conditional DE ES test statistic is much larger than zero. If so, there is evidence of autocorrelation. The limiting distribution computes large-sample critical values. Finite-sample critical values are estimated through simulation.

### Comparison of ES Backtesting Methods

The backtesting tools supported by Risk Management Toolbox have the following requirements and features.

Backtesting Tool	Portfolio Data Required	VarData Required	ESData Required	VaRLevel Required <sup>a</sup>	Portfolio ID and VaRID Supported	Distribution Information Required	Supports Multiple Models <sup>b</sup>	Supports Multiple VaRLevels
varbacktest	Yes	Yes	No	Yes	Yes	No	Yes	Yes
esbacktest	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
esbacktestby sim	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
esbacktestby de	Yes	No	No	Yes	Yes	Yes	No	Yes

- a. VaRLevel is an optional name-value pair argument with a default value of 95%. It is recommended to set the VaRLevel when creating the backtesting object.
- b. For example, you can backtest a normal and a t model in the same object with varbacktest, but you need two separate instances of the esbacktestby de class to backtest them.

Risk Management Toolbox supports the following backtesting tools and their associated tests.

Test Type	Test Name	Tests for	Risk Measure	Critical Value Computation	Use Object	Use Function
Basel	Traffic light	Frequency	VaR	Exact finite-sample (binomial)	varbacktest	tl
Various	Binomial	Frequency	VaR	Large-sample normal approximation	varbacktest	bin
Kupiec	Proportion of failures	Frequency	VaR	Exact finite-sample (log likelihood)	varbacktest	pof

Test Type	Test Name	Tests for	Risk Measure	Critical Value Computation	Use Object	Use Function
Kupiec	Time until first failure	Independence	VaR	Exact finite-sample (log likelihood)	varbacktest	tuff
Christofferson	Conditional coverage, mixed	Frequency and independence	VaR	Exact finite-sample (log likelihood)	varbacktest	cc
Christofferson	Conditional coverage, independence	Independence	VaR	Exact finite-sample (log likelihood)	varbacktest	cci
Haas	Mixed Kupiec test	Frequency and independence	VaR	Exact finite-sample (log likelihood)	varbacktest	tbf
Haas	Independence (time between failures)	Independence	VaR	Exact finite-sample (log likelihood)	varbacktest	tbfi
Acerbi-Szekely	"Test 2" or unconditional	Severity	ES	Tables of presimulated critical values, under normal and <i>t</i> distribution	esbacktest	unconditionalNormal unconditionalT
Acerbi-Szekely	"Test 1" or conditional	Severity	ES	Finite-sample simulation	esbacktest bysim	conditional
Acerbi-Szekely	"Test 2" or unconditional	Severity	ES	Finite-sample simulation	esbacktest bysim	unconditional
Acerbi-Szekely	"Test 1" or ranks (quantile)	Severity	ES	Finite-sample simulation	esbacktest bysim	quantile
Du-Escanciano	Unconditional	Severity	ES	Large-sample approximation and finite-sample simulation	esbacktest byde	unconditionalDE

Test Type	Test Name	Tests for	Risk Measure	Critical Value Computation	Use Object	Use Function
Du-Escanciano	Conditional	Independence	ES	Large-sample approximation and finite-sample simulation	esbacktestbyde	conditionallDE

## References

- [1] 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. <https://www.bis.org/publ/bcbs22.htm>.
- [2] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December 2014.
- [3] Du, Z., and J. C. Escanciano. "Backtesting Expected Shortfall: Accounting for Tail Risk." *Management Science*. Vol. 63, Issue 4, April 2017.

## See Also

esbacktest | esbacktestbyde | esbacktestbysim | varbacktest

## Related Examples

- “VaR Backtesting Workflow” on page 2-6
- “Value-at-Risk Estimation and Backtesting” on page 2-10
- “Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29
- “Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33
- “Expected Shortfall Estimation and Backtesting”
- “Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano”
- “Rolling Windows and Multiple Models for Expected Shortfall (ES) Backtesting by Du and Escanciano”

## Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information

This example shows an expected shortfall (ES) backtesting workflow and the use of ES backtesting tools. The `esbacktest` class supports two tests -- unconditional normal and unconditional  $t$  -- which are based on Acerbi-Szekely's unconditional test statistic (also known as the Acerbi-Szekely second test). These tests use presimulated critical values for the unconditional test statistic, with an assumption of normal distribution for the normal case and a  $t$  distribution with 3 degrees of freedom for the  $t$  case.

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

Use the `ESBacktestData.mat` file to load the data into the workspace. This example works with the `Returns` numeric array. This array represents the equity returns, `VaRModel1`, `VaRModel2`, and `VaRModel3`, and the corresponding VaR data at 97.5% confidence levels, generated with three different models. The expected shortfall data is contained in `ESModel1`, `ESModel2`, and `ESModel3`. The three model distributions used to generate the expected shortfall data in this example are normal (model 1),  $t$  with 10 degrees of freedom (model 2), and  $t$  with 5 degrees of freedom (model 3). However, this distribution information is not needed in this example because the `esbacktest` object does not require it.

```
load('ESBacktestData')
whos
```

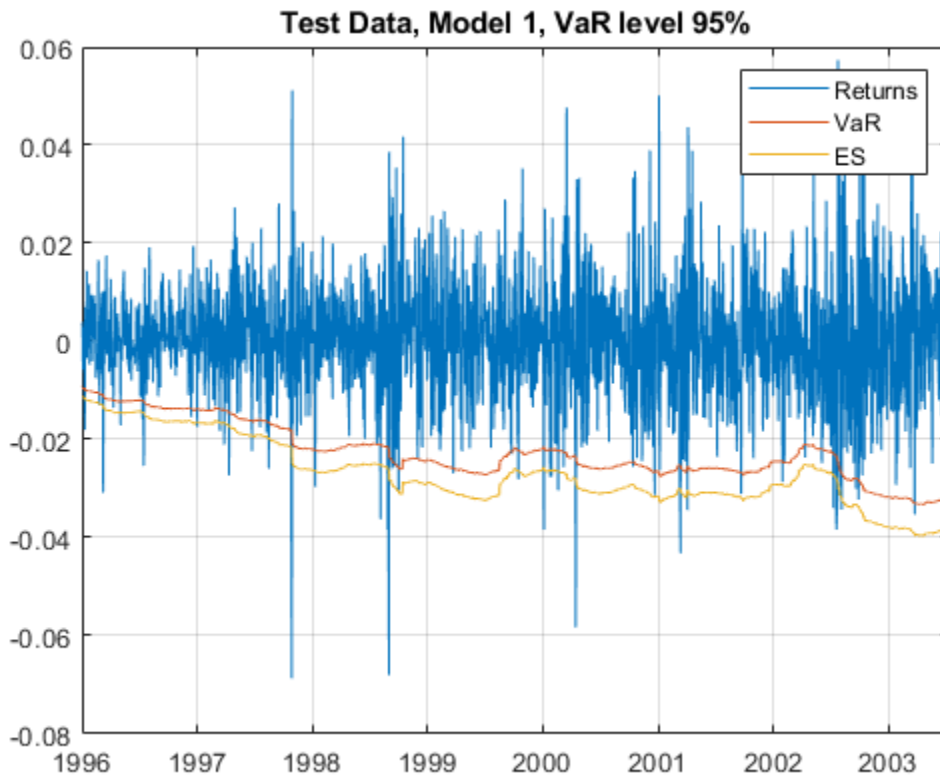
Name	Size	Bytes	Class	Attributes
Data	1966x13	223946	timetable	
Dates	1966x1	15728	datetime	
ESModel1	1966x1	15728	double	
ESModel2	1966x1	15728	double	
ESModel3	1966x1	15728	double	
Returns	1966x1	15728	double	
VaRLevel	1x1	8	double	
VaRModel1	1966x1	15728	double	
VaRModel2	1966x1	15728	double	
VaRModel3	1966x1	15728	double	

### Step 2. Generate an ES backtesting plot.

Use the `plot` function to visualize the ES backtesting data. This type of visualization is a common first step when performing an ES backtesting analysis. For illustration purposes only, visualize the returns, together with VaR and ES, for a particular model.

The resulting plot shows some large violations in 1997, 1998, and 2000. The violations in 1996 look smaller in absolute terms, however relative to the volatility of that period, those violations are also significant. For the unconditional test, the magnitude of the violations and the number of violations make a difference, because the test statistic averages over the *expected* number of failures. If the expected number is small, but there are several violations, the effective severity for the test is larger. The year 2002 is an example of a year with small, but many VaR failures.

```
figure;
plot(Dates>Returns,Dates,-VaRModel1,Dates,-ESModel1)
legend('Returns','VaR','ES')
title('Test Data, Model 1, VaR level 95%')
grid on
```



**Step 3. Create an esbacktest object.**

Create an esbacktest object using esbacktest.

```
load ESBacktestData
ebt = esbacktest>Returns,[VaRModel1 VaRModel2 VaRModel3],[ESModel1 ESModel2 ESModel3],...
    'PortfolioID','S&P','VaRID',['Model1','Model2','Model3'],'VaRLevel',VaRLevel)
```

```
ebt =
    esbacktest with properties:

    PortfolioData: [1966x1 double]
    VaRData: [1966x3 double]
    ESData: [1966x3 double]
    PortfolioID: "S&P"
    VaRID: ["Model1" "Model2" "Model3"]
    VaRLevel: [0.9750 0.9750 0.9750]
```

**Step 4. Generate the ES summary report.**

Generate the ES summary report. The ObservedSeverity column shows the average ratio of loss to VaR on periods when the VaR is violated. The ExpectedSeverity column shows the average ratio of ES to VaR for the VaR violation periods.

```
S = summary(ebt);
disp(S)
```



PortfolioID	VaRID	VaRLevel	ObservedLevel	ExpectedSeverity	ObservedSeverity
"S&P"	"Model1"	0.975	0.97101	1.1928	1.4221
"S&P"	"Model2"	0.975	0.97202	1.2652	1.4134
"S&P"	"Model3"	0.975	0.97202	1.37	1.4146

**Step 5. Run a report for all tests.**

Run all tests and generate a report only on the accept or reject results.

```
t = runtests(ebt);
disp(t)
```

PortfolioID	VaRID	VaRLevel	UnconditionalNormal	UnconditionalT
"S&P"	"Model1"	0.975	reject	reject
"S&P"	"Model2"	0.975	reject	accept
"S&P"	"Model3"	0.975	accept	accept

**Step 6. Run the unconditional normal test.**

Run the individual test for the unconditional normal test.

```
t = unconditionalNormal(ebt);
disp(t)
```

PortfolioID	VaRID	VaRLevel	UnconditionalNormal	PValue	TestStatistic
"S&P"	"Model1"	0.975	reject	0.0054099	-0.38265
"S&P"	"Model2"	0.975	reject	0.044967	-0.25011
"S&P"	"Model3"	0.975	accept	0.149	-0.15551

**Step 7. Run the unconditional t test.**

Run the individual test for the unconditional t test.

```
t = unconditionalT(ebt);
disp(t)
```

PortfolioID	VaRID	VaRLevel	UnconditionalT	PValue	TestStatistic	Critical
"S&P"	"Model1"	0.975	reject	0.018566	-0.38265	-0.25011
"S&P"	"Model2"	0.975	accept	0.073292	-0.25011	-0.25011
"S&P"	"Model3"	0.975	accept	0.17932	-0.15551	-0.25011

**Step 8. Run ES backtests for a particular year.**

Select a particular calendar year and run the tests for that year only by creating an `esbacktest` object and passing only the data of interest.

```
Year = 1996;
Ind = year(Dates)==Year;
PortID = ['S&P, ' num2str(Year)];
PortfolioData = Returns(Ind);
VaRData = [VaRModel1(Ind) VaRModel2(Ind) VaRModel3(Ind)];
```

```
ESData = [ESModel1(Ind) ESModel2(Ind) ESModel3(Ind)];  
ebt = esbacktest(PortfolioData, VaRData, ESData, ...  
    'PortfolioID', PortID, 'VaRID', ["Model1", "Model2", "Model3"], 'VaRLevel', VaRLevel);  
disp(ebt)
```

esbacktest with properties:

```
PortfolioData: [262x1 double]  
VaRData: [262x3 double]  
ESData: [262x3 double]  
PortfolioID: "S&P, 1996"  
VaRID: ["Model1" "Model2" "Model3"]  
VaRLevel: [0.9750 0.9750 0.9750]
```

```
tt = runtests(ebt);  
disp(tt)
```

PortfolioID	VaRID	VaRLevel	UnconditionalNormal	UnconditionalT
"S&P, 1996"	"Model1"	0.975	reject	reject
"S&P, 1996"	"Model2"	0.975	reject	reject
"S&P, 1996"	"Model3"	0.975	reject	accept

### See Also

[esbacktest](#) | [runtests](#) | [summary](#) | [unconditionalNormal](#) | [unconditionalT](#)

### Related Examples

- “Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33
- “Expected Shortfall Estimation and Backtesting”

### More About

- “Overview of Expected Shortfall Backtesting” on page 2-21

## Expected Shortfall (ES) Backtesting Workflow Using Simulation

This example shows an expected shortfall (ES) backtesting workflow using the `esbacktestbysim` object. The tests supported in the `esbacktestbysim` object require as inputs not only the test data (Portfolio, VaR, and ES data), but also the distribution information of the model being tested.

The `esbacktestbysim` class supports three tests -- conditional, unconditional, and quantile -- which are based on Acerbi-Szekely (2014). These tests use the distributional assumptions to simulate return scenarios, assuming the distributional assumptions are correct (null hypothesis). The simulated scenarios find the distribution of typical values for the test statistics and the significance of the tests. `esbacktestbysim` supports *normal* and *t* location-scale distributions (with a fixed number of degrees of freedom throughout the test window).

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

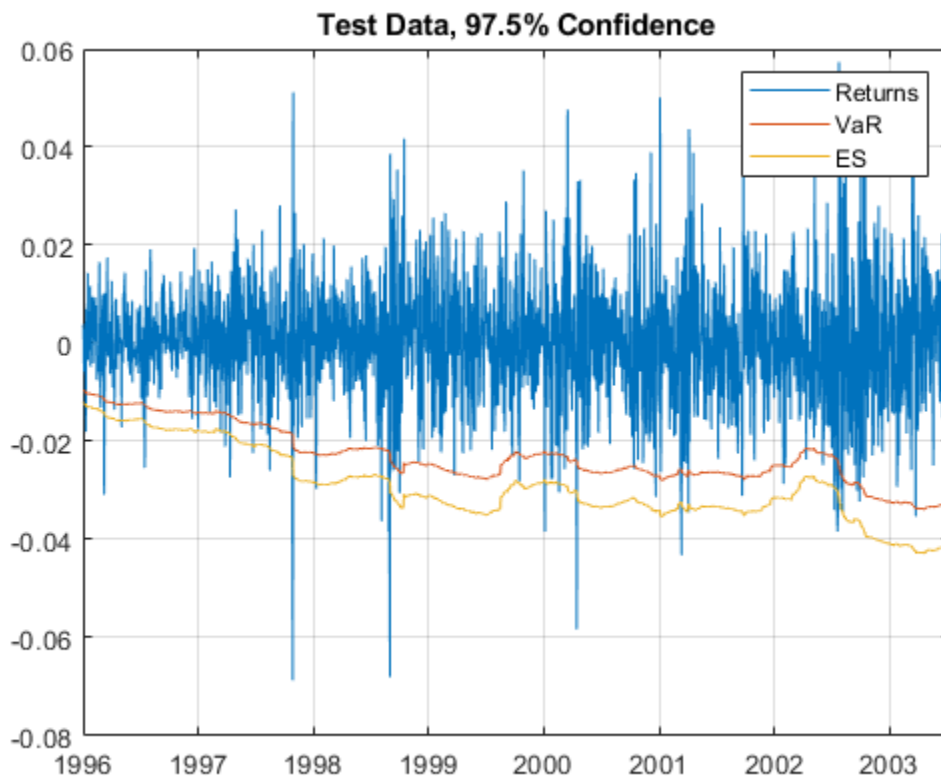
Use the `ESBacktestBySimData.mat` file to load the data into the workspace. This example works with the `Returns` numeric array. This array represents the equity returns. The corresponding VaR data and VaR confidence levels are in `VaR` and `VaRLevel`. The expected shortfall data is contained in `ES`.

```
load ESBacktestBySimData
```

### Step 2. Generate an ES backtesting plot.

Use the `plot` function to visualize the ES backtesting data. This type of visualization is a common first step when performing an ES backtesting analysis. This plot displays the returns data against the VaR and ES data.

```
VaRInd = 2;
figure;
plot(Dates,Returns,Dates,-VaR(:,VaRInd),Dates,-ES(:,VaRInd))
legend('Returns','VaR','ES')
title(['Test Data, ' num2str(VaRLevel(VaRInd)*100) '% Confidence'])
grid on
```



### Step 3. Create an esbacktestbysim object.

Create an `esbacktestbysim` object using `esbacktestbysim`. The Distribution information is used to simulate returns to estimate the significance of the tests. The simulation to estimate the significance is run by default when you create the `esbacktestbysim` object. Therefore, the test results are available when you create the object. You can set the optional name-value pair input argument `'Simulate'` to `false` to avoid the simulation, in which case you can use the `simulate` function before querying for test results.

```
rng('default'); % for reproducibility
IDs = ["t(dof) 95%", "t(dof) 97.5%", "t(dof) 99%"];
IDs = strrep(IDs, "dof", num2str(DoF));
ebts = esbacktestbysim>Returns, VaR, ES, Distribution, ...
    'DegreesOfFreedom', DoF, ...
    'Location', Mu, ...
    'Scale', Sigma, ...
    'PortfolioID', "S&P", ...
    'VaRID', IDs, ...
    'VaRLevel', VaRLevel);
disp(ebts)
```

esbacktestbysim with properties:

```
PortfolioData: [1966x1 double]
VaRData: [1966x3 double]
ESData: [1966x3 double]
Distribution: [1x1 struct]
```

```
PortfolioID: "S&P"
  VaRID: ["t(10) 95%" "t(10) 97.5%" "t(10) 99%"]
  VaRLevel: [0.9500 0.9750 0.9900]
```

disp(ebts.Distribution) % distribution information stored in the 'Distribution' property

```
Name: "t"
DegreesOfFreedom: 10
Location: 0
Scale: [1966x1 double]
```

**Step 4. Generate the ES summary report.**

The ES summary report provides information about the severity of the violations, that is, how large the loss is compared to the VaR on days when the VaR was violated. The **ObservedSeverity** (or observed average severity ratio) column is the ratio of loss to VaR over days when the VaR is violated. The **ExpectedSeverity** (or expected average severity ratio) column shows the average of the ratio of ES to VaR on the days when the VaR is violated.

```
S = summary(ebts);
disp(S)
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	ExpectedSeverity	ObservedSev
"S&P"	"t(10) 95%"	0.95	0.94812	1.3288	1.4515
"S&P"	"t(10) 97.5%"	0.975	0.97202	1.2652	1.4134
"S&P"	"t(10) 99%"	0.99	0.98627	1.2169	1.3947

**Step 5. Run a report for all tests.**

Run all tests and generate a report on only the accept or reject results.

```
t = runtests(ebts);
disp(t)
```

PortfolioID	VaRID	VaRLevel	Conditional	Unconditional	Quantile
"S&P"	"t(10) 95%"	0.95	reject	accept	reject
"S&P"	"t(10) 97.5%"	0.975	reject	reject	reject
"S&P"	"t(10) 99%"	0.99	reject	reject	reject

**Step 6. Run the conditional test.**

Run the individual test for the conditional test (also known as the first Acerbi-Szekely test). The second output (s) contains simulated test statistic values, assuming the distributional assumptions are correct. Each row of the s output matches the VaRID in the corresponding row of the t output. Use these simulated statistics to determine the significance of the tests.

```
[t,s] = conditional(ebts);
disp(t)
```

PortfolioID	VaRID	VaRLevel	Conditional	ConditionalOnly	PValue	TestS
"S&P"	"t(10) 95%"	0.95	reject	reject	0	-0
"S&P"	"t(10) 97.5%"	0.975	reject	reject	0.001	-0
"S&P"	"t(10) 99%"	0.99	reject	reject	0.003	-0

whos s

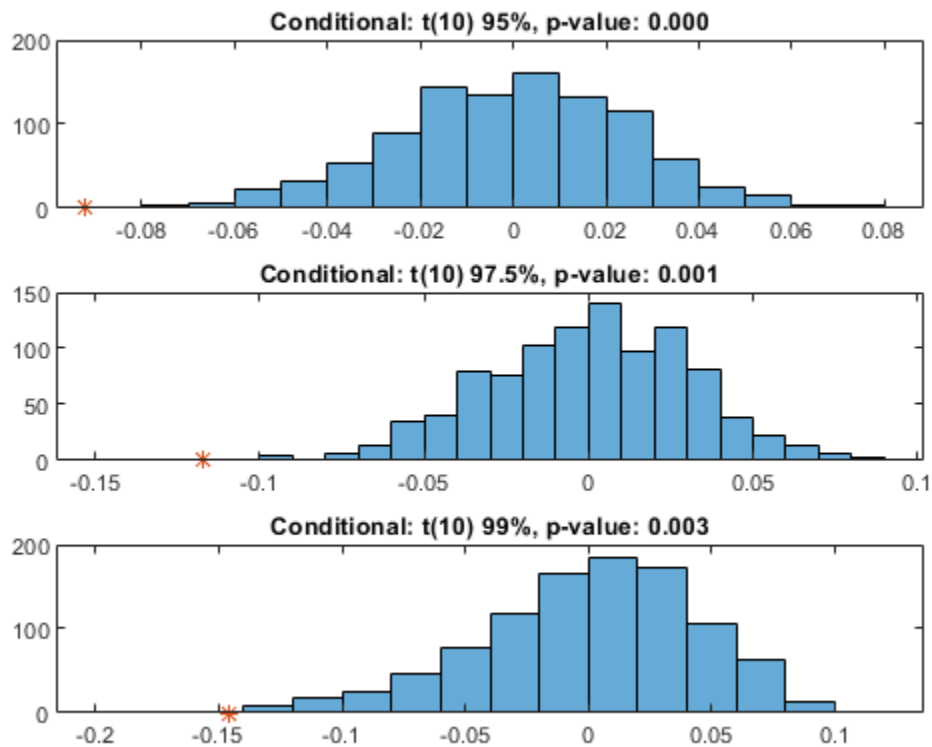
Name	Size	Bytes	Class	Attributes
s	3x1000	24000	double	

**Step 7. Visualize the significance of the conditional test.**

Visualize the significance of the conditional test using histograms to show the distribution of typical values (simulation results). In the histograms, the asterisk shows the value of the test statistic observed for the actual returns. This is a visualization of the standalone conditional test. The final conditional test result also depends on a preliminary VaR backtest, as shown in the conditional test output.

```

NumVaRs = height(t);
figure;
for VaRInd = 1:NumVaRs
    subplot(NumVaRs,1,VaRInd)
    histogram(s(VaRInd,:));
    hold on;
    plot(t.TestStatistic(VaRInd),0,'*');
    hold off;
    Title = sprintf('Conditional: %s, p-value: %4.3f',t.VaRID(VaRInd),t.PValue(VaRInd));
    title(Title)
end
    
```



**Step 8. Run the unconditional test.**

Run the individual test for the unconditional test (also known as the second Acerbi-Szekely test).

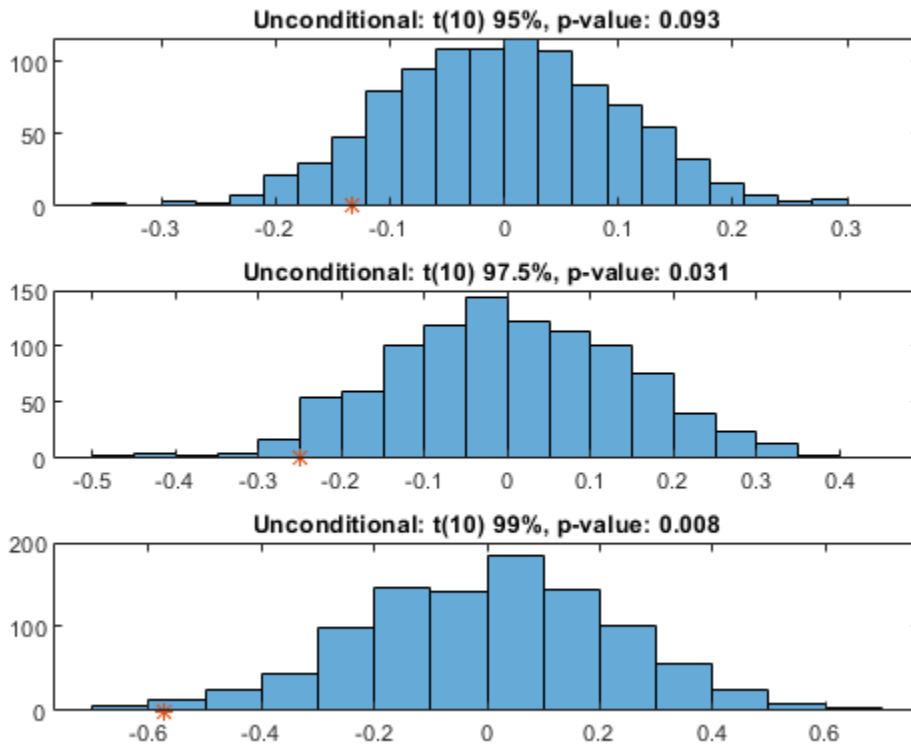
```
[t,s] = unconditional(ebts);
disp(t)
```

PortfolioID	VaRID	VaRLevel	Unconditional	PValue	TestStatistic	Crit
"S&P"	"t(10) 95%"	0.95	accept	0.093	-0.13342	-0
"S&P"	"t(10) 97.5%"	0.975	reject	0.031	-0.25011	-0
"S&P"	"t(10) 99%"	0.99	reject	0.008	-0.57396	-0

**Step 9. Visualize the significance of the unconditional test.**

Visualize the significance of the unconditional test using histograms to show the distribution of typical values (simulation results). In the histograms, the asterisk shows the value of the test statistic observed for the actual returns.

```
NumVaRs = height(t);
figure;
for VaRInd = 1:NumVaRs
    subplot(NumVaRs,1,VaRInd)
    histogram(s(VaRInd,:));
    hold on;
    plot(t.TestStatistic(VaRInd),0,'*');
    hold off;
    Title = sprintf('Unconditional: %s, p-value: %4.3f',t.VaRID(VaRInd),t.PValue(VaRInd));
    title(Title)
end
```



**Step 10. Run the quantile test.**

Run the individual test for the quantile test (also known as the third Acerbi-Szekely test).

```
[t,s] = quantile(ebts);
disp(t)
```

PortfolioID	VaRID	VaRLevel	Quantile	PValue	TestStatistic	CriticalVaR
"S&P"	"t(10) 95%"	0.95	reject	0.002	-0.10602	-0.0557
"S&P"	"t(10) 97.5%"	0.975	reject	0	-0.15697	-0.0735
"S&P"	"t(10) 99%"	0.99	reject	0	-0.26561	-0.101

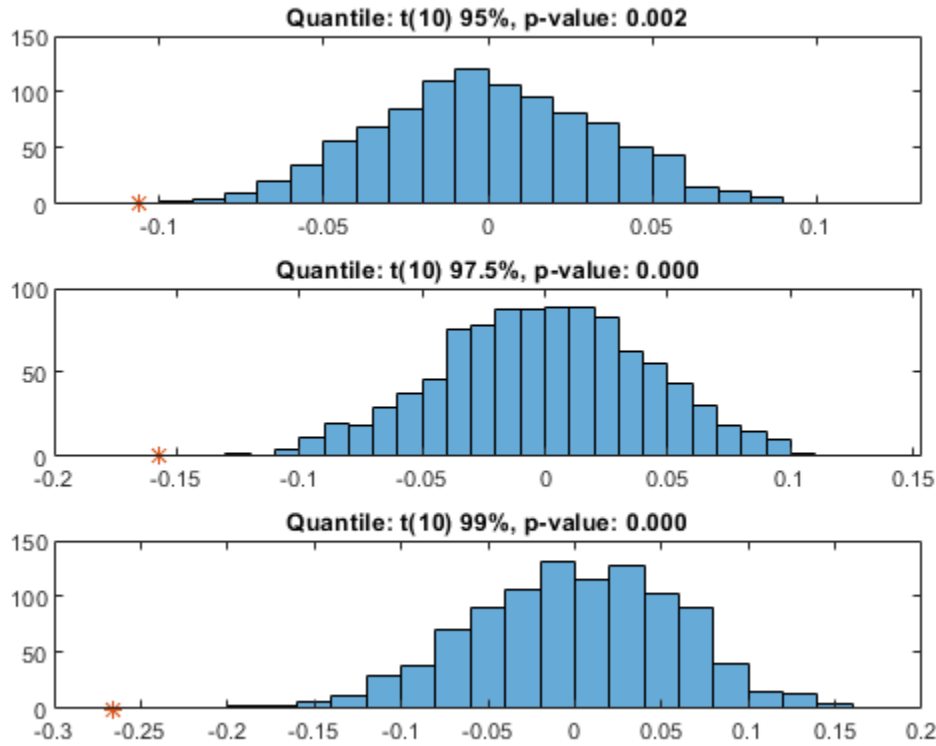
**Step 11. Visualize the significance of the quantile test.**

Visualize the significance of the quantile test using histograms to show the distribution of typical values (simulation results). In the histograms, the asterisk shows the value of the test statistic observed for the actual returns.

```
NumVaRs = height(t);
figure;
for VaRInd = 1:NumVaRs
    subplot(NumVaRs,1,VaRInd)
        histogram(s(VaRInd,:));
        hold on;
        plot(t.TestStatistic(VaRInd),0,'*');
        hold off;
```



```
Title = sprintf('Quantile: %s, p-value: %4.3f',t.VaRID(VaRInd),t.PValue(VaRInd));
title>Title)
end
```



**Step 12. Run a new simulation to estimate the significance of the tests.**

Run the simulation again using 5000 scenarios to generate a new set of test results. If the initial test results for one of the tests are borderline, using a larger simulation can help clarify the test results.

```
ebts = simulate(ebts,'NumScenarios',5000);
t = unconditional(ebts); % new results for unconditional test
disp(t)
```

PortfolioID	VaRID	VaRLevel	Unconditional	PValue	TestStatistic	Crit
"S&P"	"t(10) 95%"	0.95	accept	0.0984	-0.13342	-0
"S&P"	"t(10) 97.5%"	0.975	reject	0.0456	-0.25011	-0
"S&P"	"t(10) 99%"	0.99	reject	0.0104	-0.57396	-0

**See Also**

[conditional](#) | [quantile](#) | [runtests](#) | [simulate](#) | [summary](#) | [unconditional](#)

**Related Examples**

- "Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information" on page 2-29

- “Expected Shortfall Estimation and Backtesting”

**More About**

- “Overview of Expected Shortfall Backtesting” on page 2-21

# 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 3-21
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-34

## 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 with an option to bin missing data. (For more information on algorithms for automatic binning, see `autobinning`.)
- Shift bin boundaries.
- Split bins.
- Merge bins.
- Save and export a `creditscorecard` object.

---

**Note** When using the **Binning Explorer** app with MATLAB Online:

- The App toolbar is not available for MATLAB Online. To access Help, from the MATLAB® command prompt, enter `doc binningExplorer`.
  - MATLAB Online does not display predictor information using three panels (**Overview**, **Bin Information**, and **Predictor Information**) in the Binning Explorer window. Instead, MATLAB Online displays these panels as tabs labelled **Overview**, **Bin Information**, and **Predictor Information**.
  - When performing manual binning, selected predictors are displayed in a tab in the Binning Explorer window. When you close the tab for a predictor, you do not return to the **Overview** panel. To return to the **Overview** panel, click the **Overview** tab.
- 

**Binning Explorer** complements the overall workflow for developing a credit scorecard model. Use `screenpredictors` to pare down a potentially large set of predictors to a subset that is most predictive of the credit score card response variable. You can then use this subset of predictors when using **Binning Explorer** to create the `creditscorecard` object.

Using Binning Explorer:	
1.	Open the <b>Binning Explorer</b> app. <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:               <ul style="list-style-type: none"> <li>• Enter <code>binningExplorer</code> to open the <b>Binning Explorer</b> app.</li> <li>• Enter <code>binningExplorer(data)</code> or <code>binningExplorer(data,Name,Value)</code> to open a table in the <b>Binning Explorer</b> app by specifying a table (<code>data</code>) as input.</li> <li>• Enter <code>binningExplorer(sc)</code> to open a <code>creditscorecard</code> object in the <b>Binning Explorer</b> app by specifying a <code>creditscorecard</code> object (<code>sc</code>) as input.</li> </ul> </li> </ul>
2.	Import the data into the app. <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.	Use <b>Binning Explorer</b> to work interactively with the binning assignments for a scorecard.

<b>Using Binning Explorer:</b>	
4.	Export the scorecard to a new <code>creditscorecard</code> object.  Continue the workflow from the MATLAB command line using <code>creditscorecard</code> object functions from Financial Toolbox. For more information, see <code>creditscorecard</code> .
<b>Using <code>creditscorecard</code> Object Functions in Financial Toolbox:</b>	
5.	Fit a logistic regression model.
6.	Review and format the credit scorecard points.
7.	Score the data.
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 3-21.

## See Also

### Apps

**Binning Explorer**

### Classes

`creditscorecard`

## Related Examples

- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-21
- “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 17 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 3-6

“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 3-14

“Change Bin Boundaries for Multiple Predictors” on page 3-15

“Set Options for Display” on page 3-16

“Export and Save the Binning” on page 3-17

“Troubleshoot the Binning” on page 3-17

## 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:

- 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. If the input MATLAB table contains a column for weights, from the **Step 2** pane, using the **Variable Type** column, click the drop-down to select **Weights**. If the data contains missing values, from the **Step 2** pane, set **Bin missing data:** to **Yes**. For more information on working with missing data, see “Credit Scorecard Modeling with Missing Values” (Financial Toolbox).

- 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 creditscorecard Object

To start using an existing creditscorecard object:

- Place the creditscorecard object in your MATLAB workspace. Create the creditscorecard object either by using creditscorecard or by clicking **Export** in the **Binning Explorer** to export and save a creditscorecard object to the MATLAB workspace.
- Open Binning Explorer from the MATLAB toolstrip: On the **Apps** tab, under **Computational Finance**, click the app icon.
- From **Step 1** pane of the Import Data window, select the creditscorecard object.
- From the **Step 3** pane, select a binning algorithm. When using an existing creditscorecard 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.

### Start from MATLAB Command Line Using Data or an Existing creditscorecard Object

To start **Binning Explorer** from the MATLAB command line:

- Place the credit scorecard data or existing creditscorecard object in your MATLAB workspace.
- At the MATLAB command prompt:
  - Enter `binningExplorer(data)` or `binningExplorer(data,Name,Value)` to open a table in the **Binning Explorer** app by specifying a table (data) as input.
  - Enter `binningExplorer(sc)` to open an existing creditscorecard object in the **Binning Explorer** app by specifying a creditscorecard object (sc) as input.

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 creditscorecard object into **Binning Explorer**, you can change the predictor type.

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

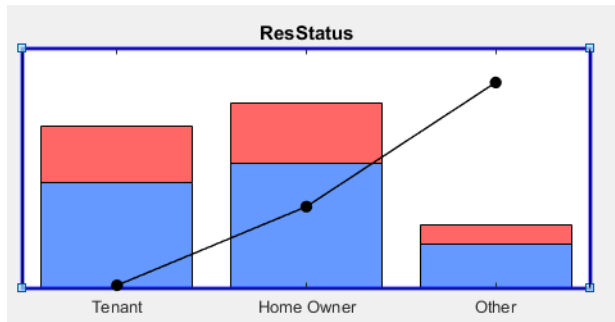
On the **Binning Explorer** toolstrip, the predictor type for the selected predictor displays under **Predictor Type**.

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

## Change Binning Algorithm for One or More Predictors

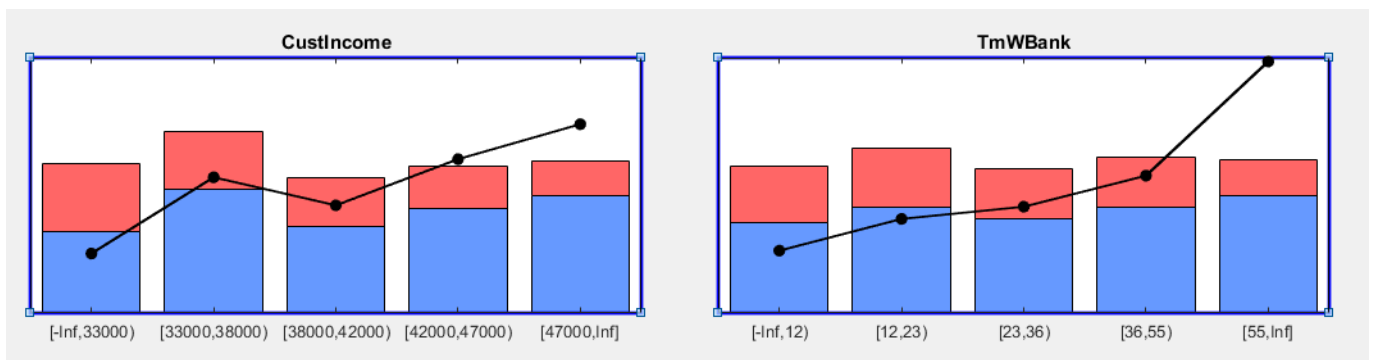
After you import data or a `creditscorecard` object into **Binning Explorer**, you can change the binning algorithm for an individual predictor or for multiple predictors.

- 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** toolstrip, click **Apply Monotone** and select **Monotone**, **Split**, **Merge**, **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.



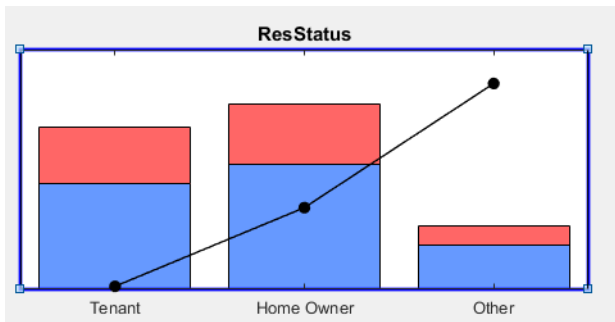
- 4 Click **Apply Monotone** and select **Monotone**, **Split**, **Merge**, **Equal Frequency**, or **Equal Width**. All the selected predictor plots are updated for a change of algorithm.

## Change Algorithm Options for Binning Algorithms

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

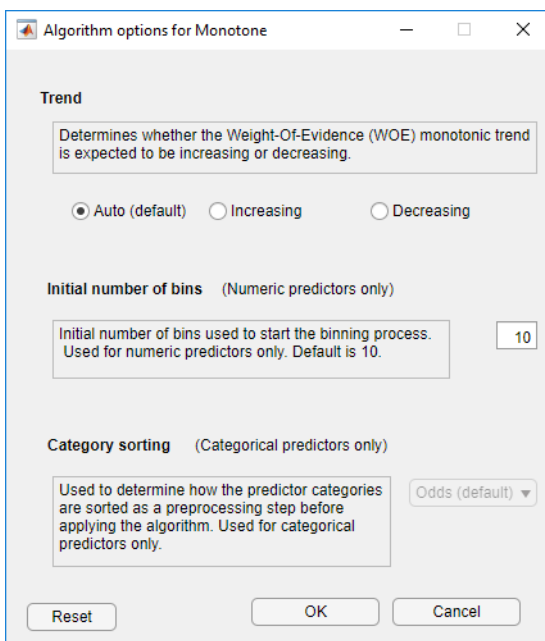
- 1 Click any predictor plot. The predictor plot displays with a blue outline.





**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** toolstrip, click **Algorithm Options** to open the Algorithm Options dialog box.



- 3 From the associated Algorithm options dialog box:

- **Monotone**

- 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 **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 **Category Sorting**, used for categorical predictors only, 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.

For more information, see Sort Categories (Financial Toolbox)

- **Split**
  - For **Measure**, select one of the following: **Gini** (default), **Chi2**, **InfoValue**, or **Entropy**.
  - For **Tolerance**, specify a tolerance value above which the gain in the information value has to be for the split to be accepted. The default is  $1e-4$ .
  - For **Significance**, only for the **Chi2** measure, specify a significance level threshold for the chi-square statistic, above which splitting happens. Values are in the interval  $[0, 1]$ . Default is 0.9 (90% significance level).
  - For **Bin distribution**, specify values for
    - **MinBad** — Specifies the minimum number  $n$  ( $n \geq 0$ ) of Bads per bin. The default value is 1, to avoid pure bins.
    - **MaxBad** — Specifies the maximum number  $n$  ( $n \geq 0$ ) of Bads per bin. The default value is Inf.
    - **MinGood** — Specifies the minimum number  $n$  ( $n \geq 0$ ) of Goods per bin. The default value is 1, to avoid pure bins.
    - **MaxGood** — Specifies the maximum number  $n$  ( $n \geq 0$ ) of Goods per bin. The default value is Inf.
    - **MinCount** — Specifies the minimum number  $n$  ( $n \geq 0$ ) of observations per bin. The default value is 1, to avoid empty bins.
    - **MaxCount** — Specifies the maximum number  $n$  ( $n \geq 0$ ) of observations per bin. The default value is Inf.
    - **MaxNumBins** — Specifies the maximum number  $n$  ( $n \geq 2$ ) of bins resulting from the splitting. The default value is 5.
  - For **Initial number bins**, specify an integer that determines the number ( $n > 0$ ) of bins that the predictor is initially binned into before splitting. Valid for numeric predictors only. Default is 50.
  - For **Category sorting**, used for categorical predictors only, select a value:

- **Goods** — The categories are sorted by order of increasing values of “Good.”
- **Bads** — The categories are sorted by order of increasing values of “Bad.”
- **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.
- **Totals** — The categories are sorted by order of increasing values of total number of observations (“Good” plus “Bad”).
- **None** — No sorting is applied. The existing order of the categories is unchanged before applying the algorithm. (The existing order of the categories can be seen in the category grouping optional output from `bininfo`.)

For more information, see Sort Categories (Financial Toolbox)

- **Merge**

- For **Measure**, select one of the following: **Chi2** (default), **Gini**, **InfoValue**, or **Entropy**.
- For **Tolerance**, specify the minimum threshold below which merging happens for the information value and entropy statistics. Valid values are in the interval  $(0, 1)$ . Default is  $1e-3$ .
- For **Significance**, specify the significance level threshold for the chi-square statistic, below which merging happens. Values are in the interval  $[0, 1]$ . Default is  $0.9$  (90% significance level).
- For **Bin distribution**, specify the following:
  - **MinNumBins** — Specifies the minimum number  $n$  ( $n \geq 2$ ) of bins that result from merging. The default value is 2.
  - **MaxNumBins** — Specifies the maximum number  $n$  ( $n \geq 2$ ) of bins that result from merging. The default value is 5.
- For **Initial number of bins**, specify an integer that determines the number ( $n > 0$ ) of bins that the predictor is initially binned into before merging. Valid for numeric predictors only. Default is 50.
- For **Category sorting**, used for categorical predictors only. Select a value:
  - **Goods** — The categories are sorted by order of increasing values of “Good.”
  - **Bads** — The categories are sorted by order of increasing values of “Bad.”
  - **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.
  - **Totals** — The categories are sorted by order of increasing values of total number of observations (“Good” plus “Bad”).
  - **None** — No sorting is applied. The existing order of the categories is unchanged before applying the algorithm. (The existing order of the categories can be seen in the category grouping optional output from `bininfo`.)

For more information, see Sort Categories (Financial Toolbox)

- **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.

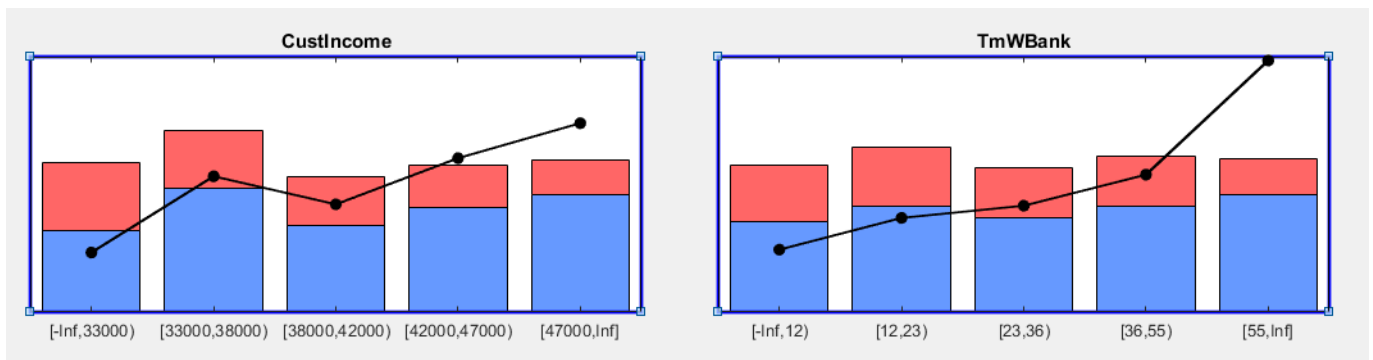
---

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

---

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

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

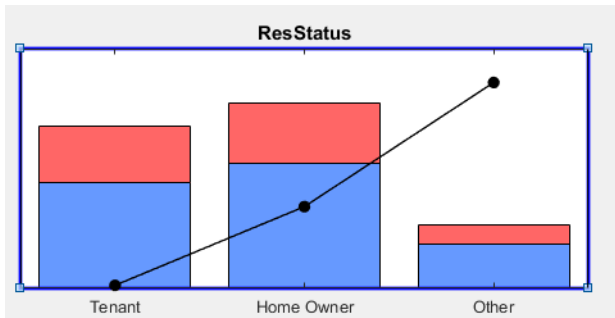


- 5 On the **Binning Explorer** toolstrip, click **Algorithm Options** to open the Algorithm Options dialog box. Make your selection from the Algorithm Options dialog box and click **OK**. The selected predictor plots are updated for the change of algorithm.

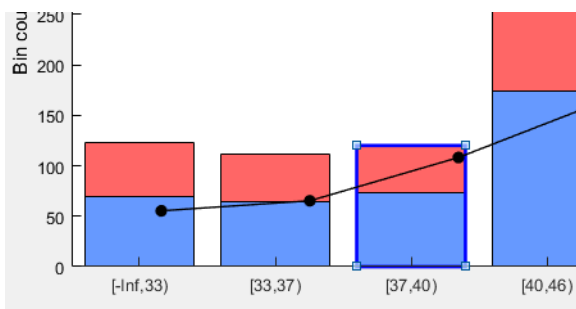
## Split Bins for a Numeric Predictor

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.

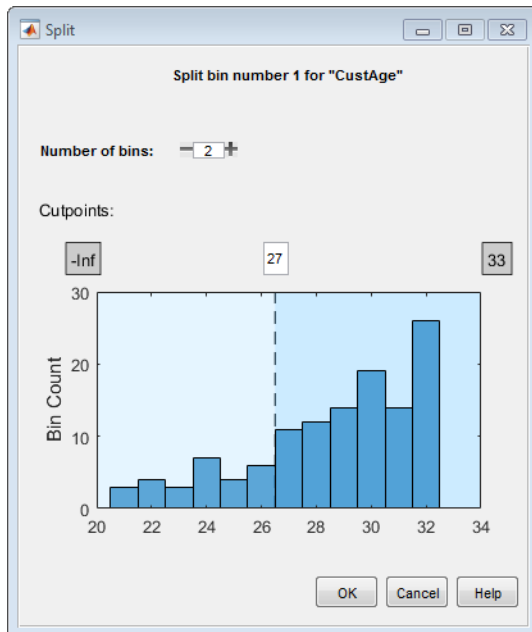



---

**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.



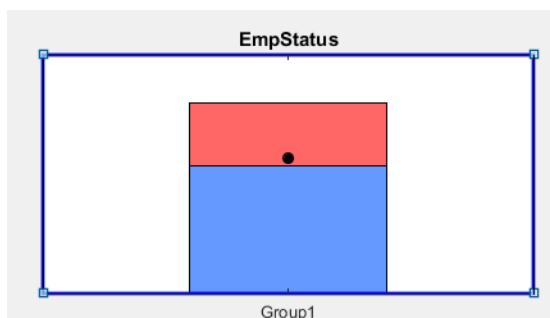
- 5 Use the **Number of bins** control to split the selected bin into multiple bins. Click **OK** to complete the split operation.

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.

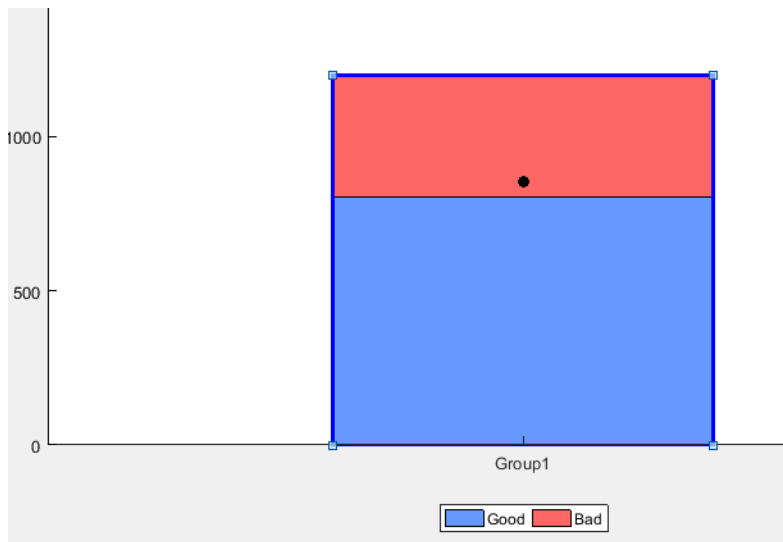
## Split Bins for a Categorical Predictor

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

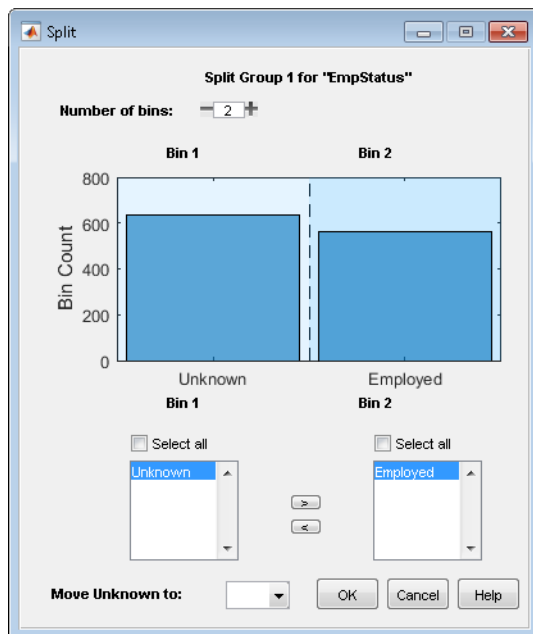
- 1 Click any categorical predictor plot. The predictor plot displays with a blue outline.



- 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.

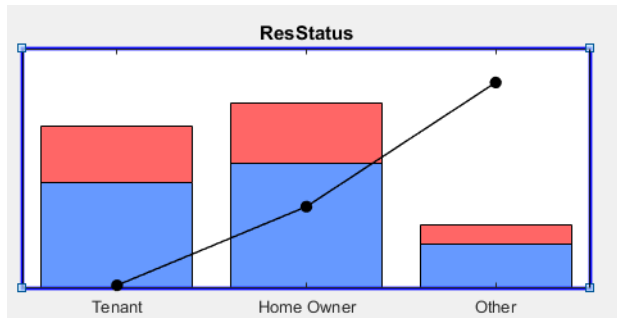
- 4 Click **OK** to complete the split operation.

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.



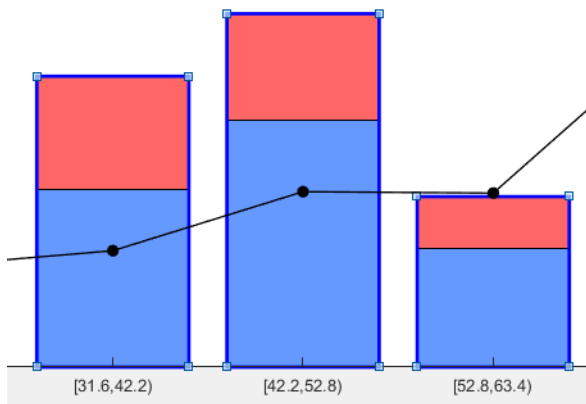
- 2 On the **Binning Explorer** toolstrip, click **Manual Binning** to open the selected predictor in a new tabbed window.

---

**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** toolstrip display the values for the edges of the selected bins to merge.

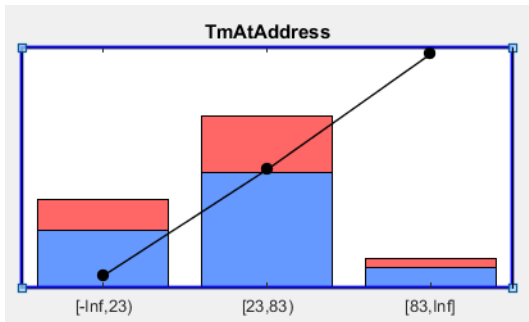
- 4 Click **Merge** to complete the merge operation. The plot for the selected predictor is updated with the new bin information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

## Change Bin Boundaries for a Single Predictor

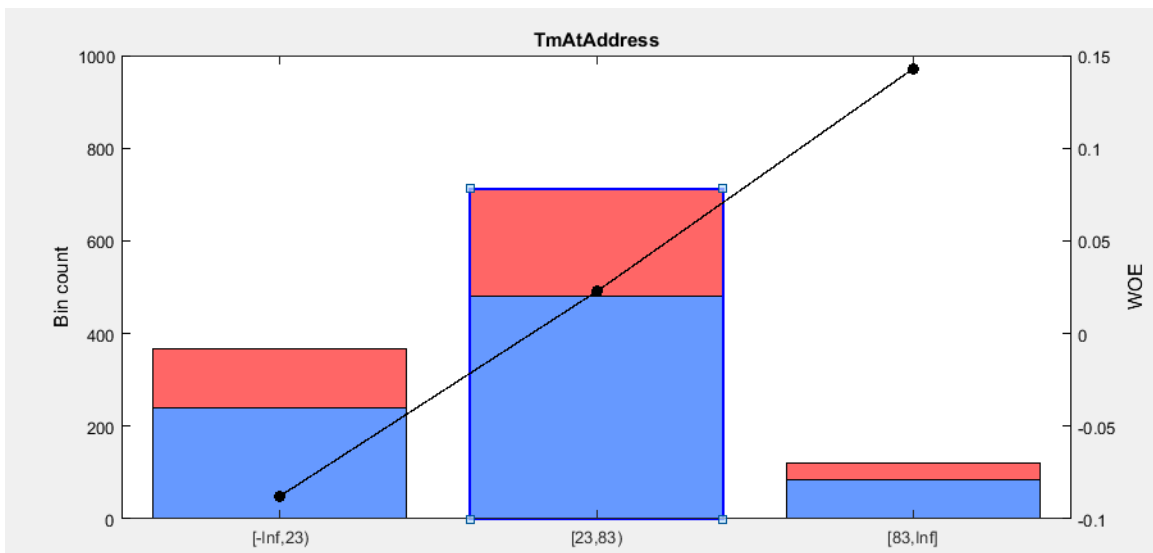
After you import data or a `creditscorecard` object into **Binning Explorer**, you can change the bin boundaries for a single predictor.



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



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

Edges:

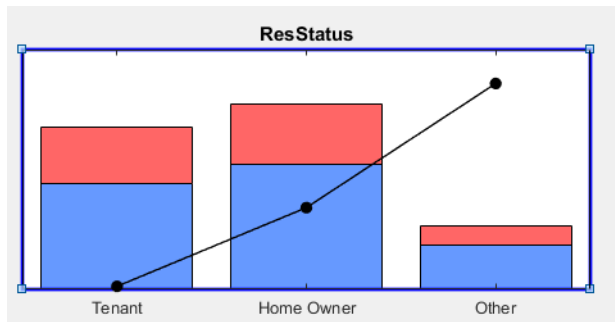
Edit the values in the **Edges** text boxes to change the selected bin's dimensions.

- 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.

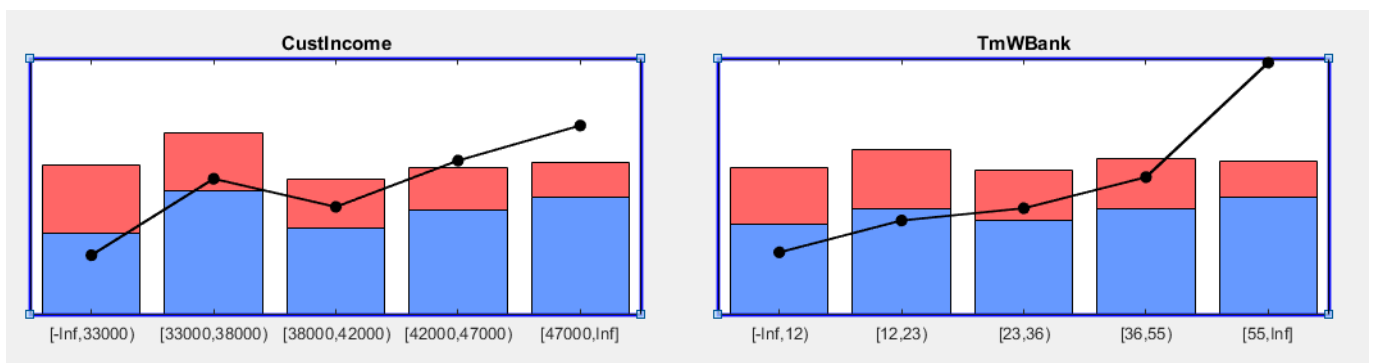
## Change Bin Boundaries for Multiple Predictors

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 The Algorithm Options dialog box displays the options for the current binning algorithm. Depending on which is the current algorithm, you can change bin boundaries:
  - If your current algorithm for the selected bins is **EqualWidth** or **EqualFrequency**, enter a number in the **Number of bins** box. Optionally, for **EqualWidth** and **EqualFrequency** options, under **Category Sorting**, specify the type of sorting.
  - If your current algorithm for the selected bins is **Monotone**, **Split**, or **Merge** the default of 10 for **Monotone** or 50 for **Split** and **Merge** is used for the **Initial number of bins**. Optionally, for **Monotone**, 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** toolstrip item for **Table Columns**, select any of the following options:
  - **Odds**
  - **WOE**
  - **InfoValue**
  - **Entropy**
  - **Chi2**
  - **Gini**
  - **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 3-18
- “Predictor Plot Appears Distorted” on page 3-19

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 3-21.

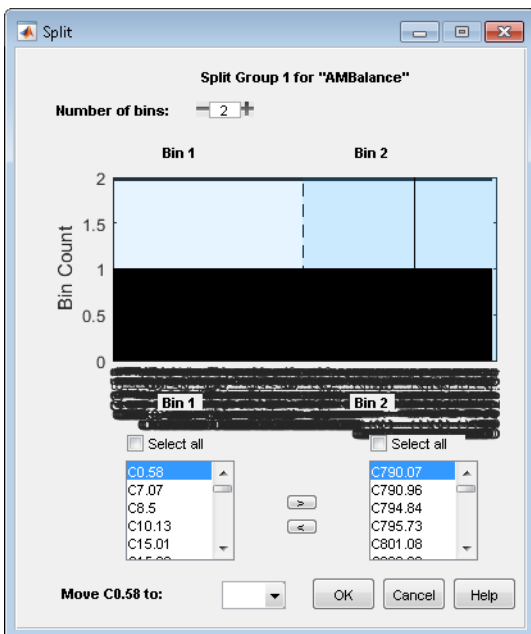
**Numeric Predictor Converted to Categorical Predictor Does Not Display Split Data Properly**

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`

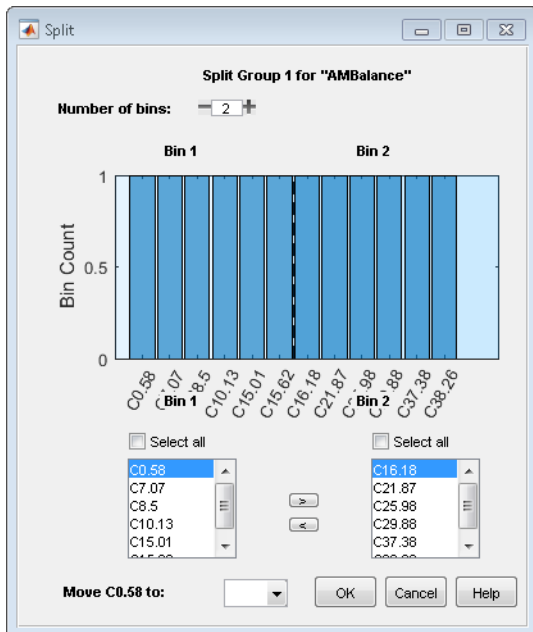
Open the **Binning Explorer** and select the numeric predictor **AMBalance**. From the Binning Explorer toolbar, change the predictor type to **Categorical**.

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.

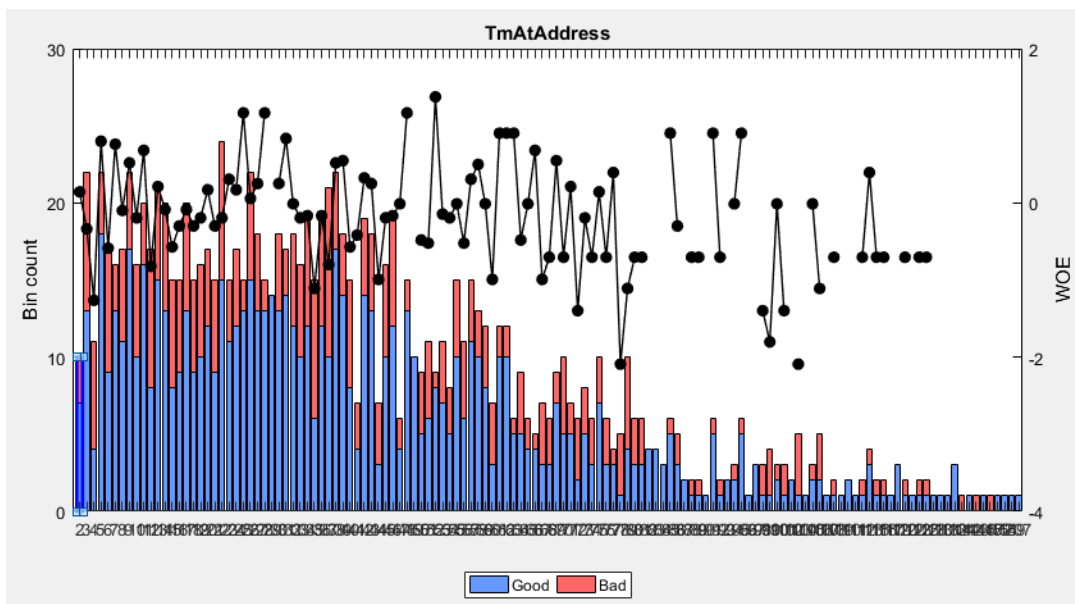
**Solution:** If you have a categorical predictor with a large number of categories, use the **Algorithm Options** to change the binning algorithm for that predictor to **Equal Frequency**, with the **Number of bins** set to 100 (or another smaller value). The Split dialog box then displays properly.



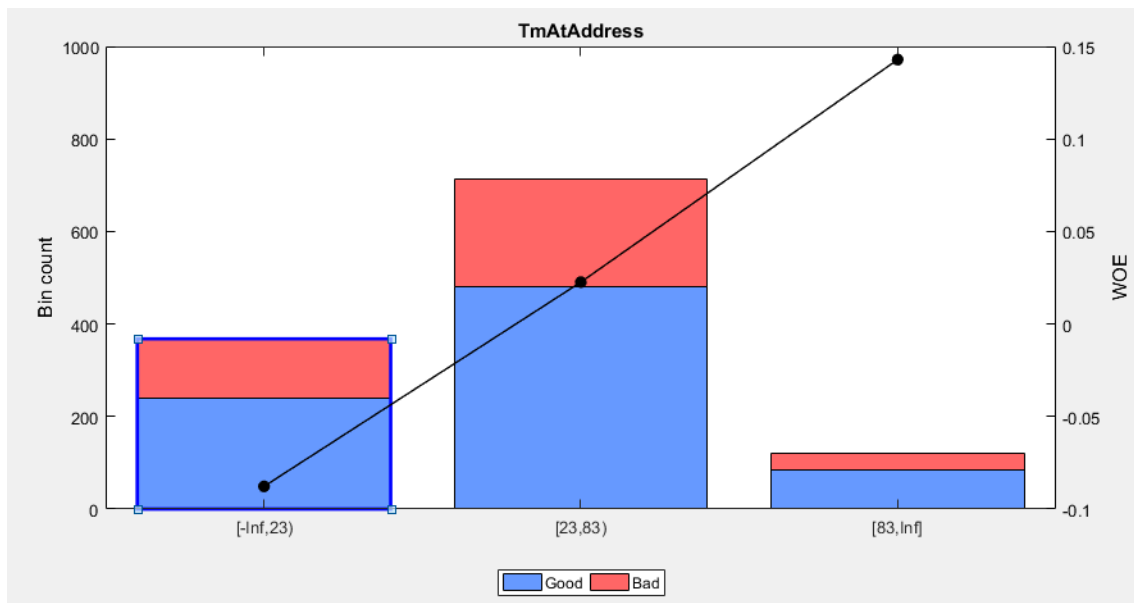
### Predictor Plot Appears Distorted

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

### Classes

creditscorecard

## Related Examples

- "Binning Explorer Case Study Example" on page 3-21
- "Case Study for a Credit Scorecard Analysis" (Financial Toolbox)
- "Credit Scorecard Modeling with Missing Values" (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 17 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).

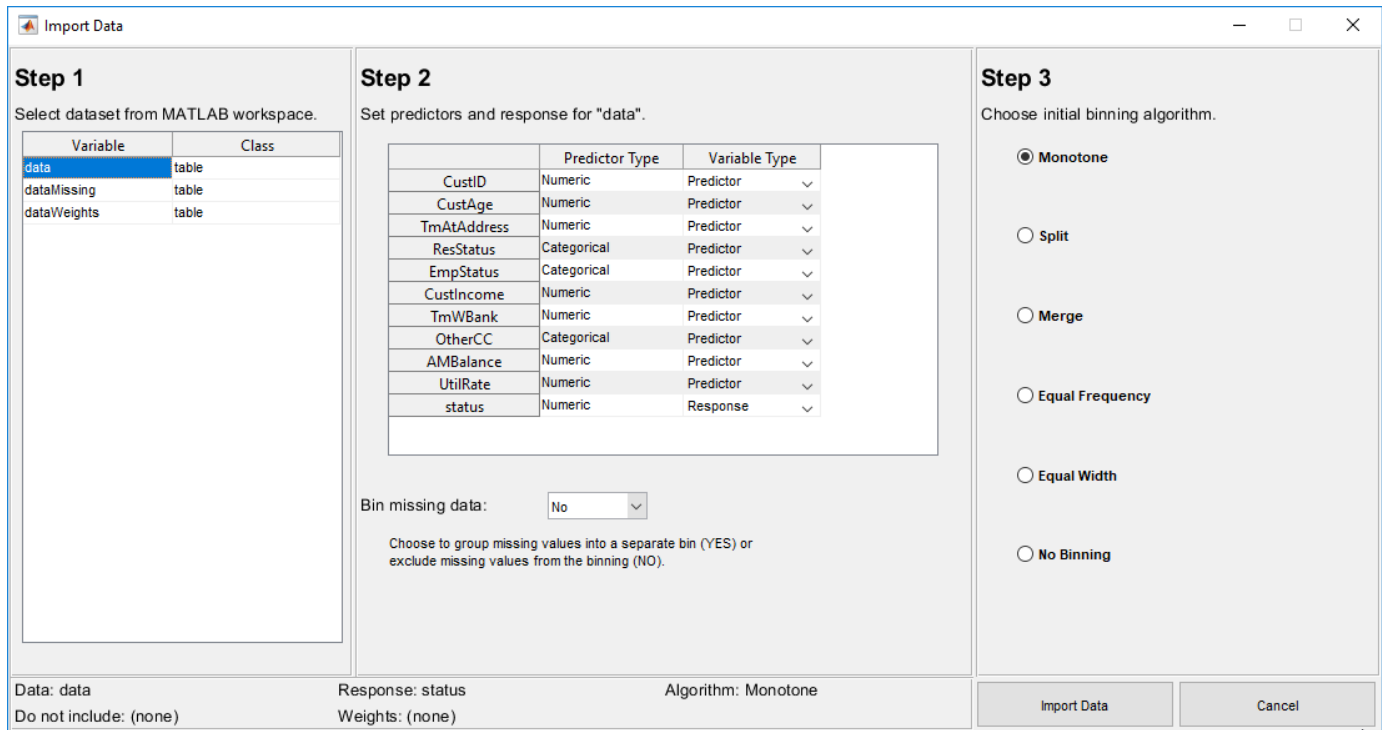
```
load CreditCardData
disp(data(1:10,:))
```

CustID	CustAge	TmAtAddress	ResStatus	EmpStatus	CustIncome	TmWBank	OtherCC	AMBalance	UtilRate	status
1	53	62	Tenant	Unknown	50000	55	Yes	1055.9	0.22	0
2	61	22	Home Owner	Employed	52000	25	Yes	1161.6	0.24	0
3	47	30	Tenant	Employed	37000	61	No	877.23	0.29	0
4	50	75	Home Owner	Employed	53000	20	Yes	157.37	0.08	0
5	68	56	Home Owner	Employed	53000	14	Yes	561.84	0.11	0
6	65	13	Home Owner	Employed	48000	59	Yes	968.18	0.15	0
7	34	32	Home Owner	Unknown	32000	26	Yes	717.82	0.02	1
8	50	57	Other	Employed	51000	33	No	3041.2	0.13	0
9	50	10	Tenant	Unknown	52000	25	Yes	115.56	0.02	1
10	49	30	Home Owner	Unknown	53000	23	Yes	718.5	0.17	1

### 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 MATLAB command line. For more information on starting the **Binning Explorer** from the command line, see “Start from MATLAB Command Line Using Data or an Existing `creditscorecard` Object” on page 3-5.

From the **Binning Explorer** toolstrip, 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**.

**Note** If the input MATLAB table contains a column for weights, from the **Step 2** pane, using the **Variable Type** column, click the drop-down to select **Weights**. For more information on using observation weights with a `creditscorecard` object, see “Credit Scorecard Modeling Using Observation Weights” (Financial Toolbox).

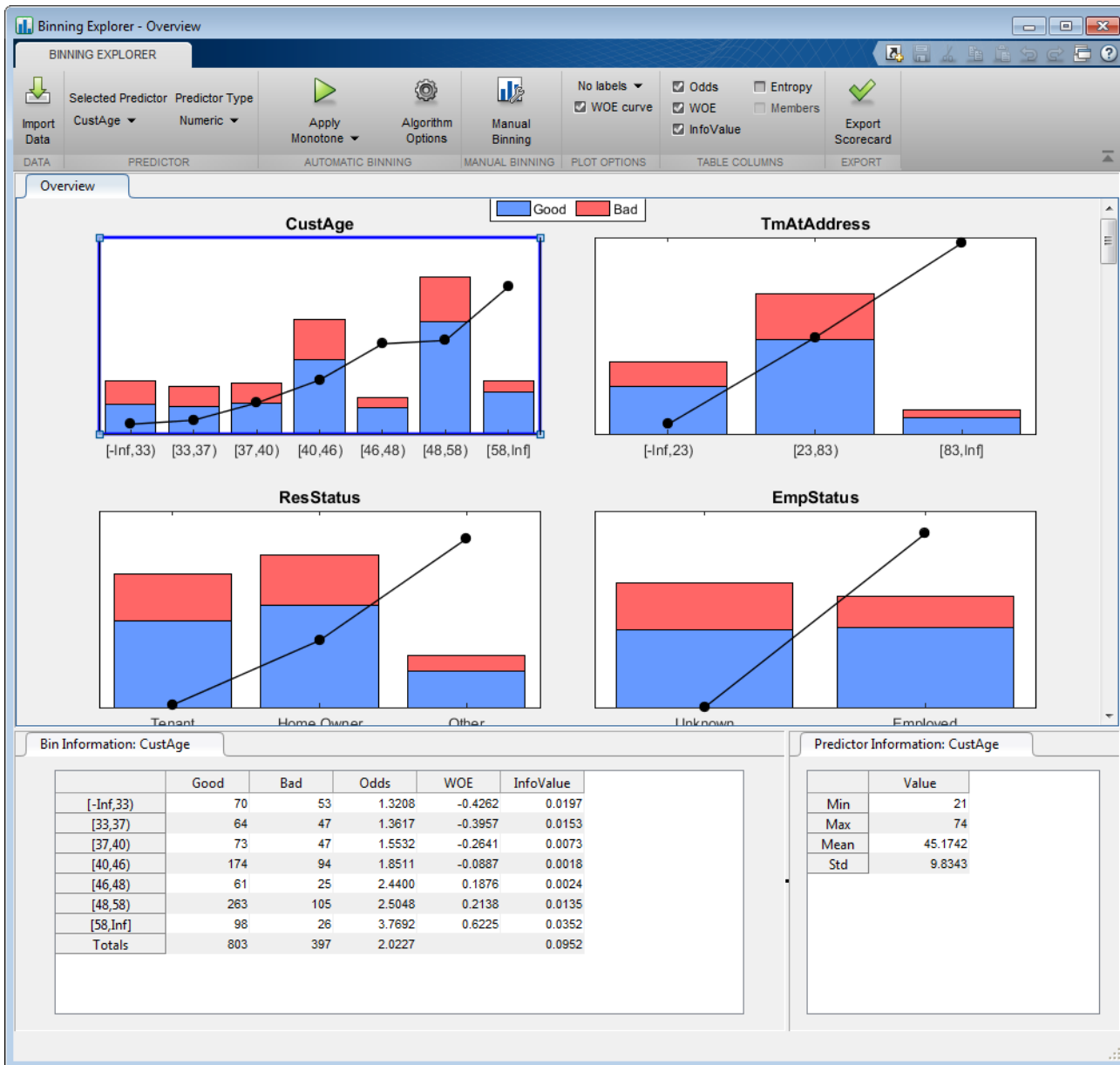
If the data contains missing values, from the **Step 2** pane, set **Bin missing data:** to **Yes**. For more information on working with missing data, see “Credit Scorecard Modeling with Missing Values” (Financial Toolbox).

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.





**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).

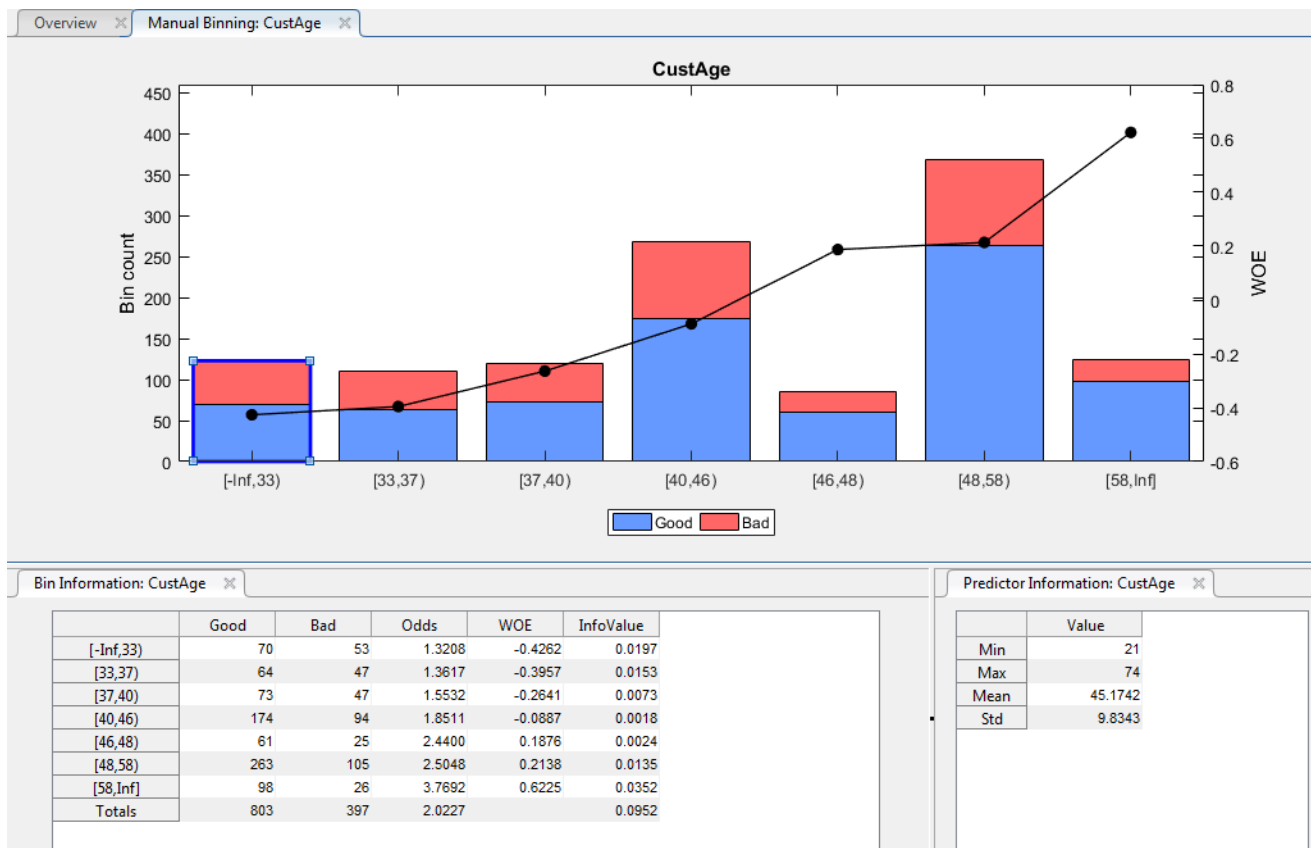
Bin Information: ResStatus					
	Good	Bad	Odds	WOE	InfoValue
Tenant	307	167	1.8383	-0.0956	0.0037
Home Owner	365	177	2.0621	0.0193	1.6820e-04
Other	131	53	2.4717	0.2005	0.0059
Totals	803	397	2.0227		0.0098

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

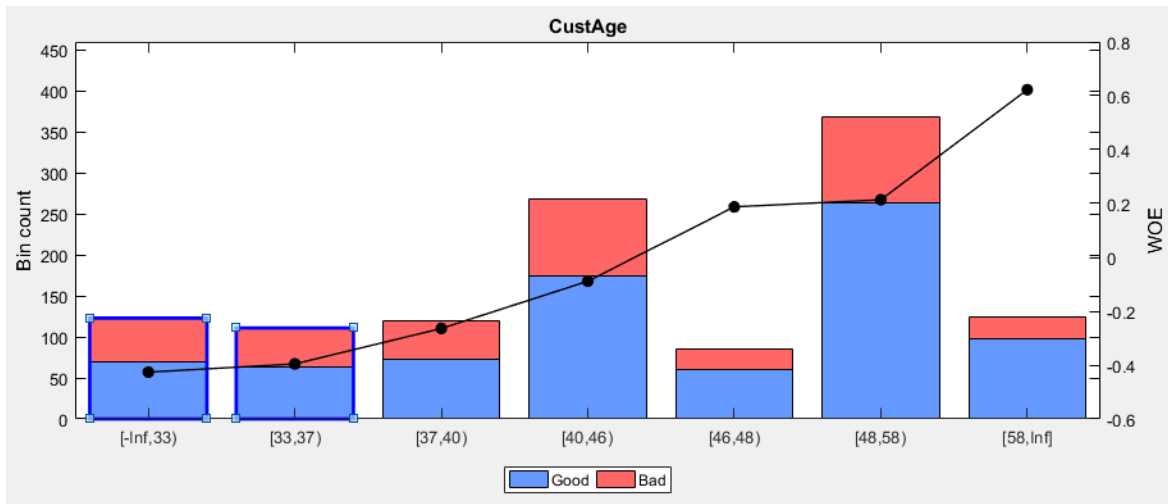
### Step 3. Fine-tune the bins using manual binning in Binning Explorer.

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

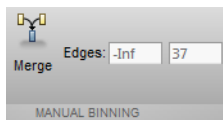


To merge bins 1 and 2, from the **Binning Explorer** toolstrip, click **Manual Binning** to open the selected predictor in a new tabbed window. Alternatively, double-click the predictor plot to open the

**Manual Binning** tab. Select bin 1 and 2 for merging by using **Ctrl + click** to multiselect these bins to display with blue outlines.



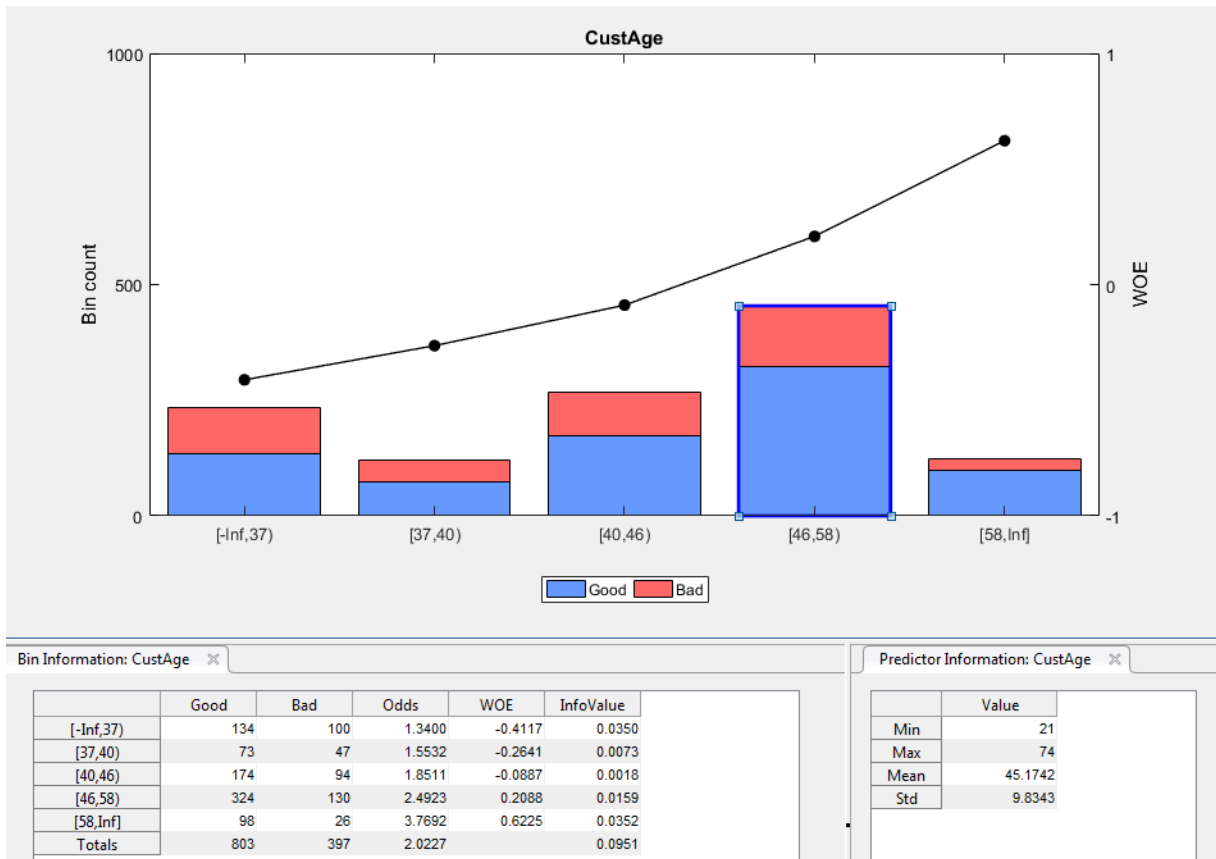
On the **Binning Explorer** toolbar, the **Edges** text boxes display values for the edges of the selected bins to merge.



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.



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



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 creditcorecard object from Binning Explorer.

After you complete your binning assignments, using **Binning Explorer**, click **Export** and provide a creditcorecard object name. The creditcorecard 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<sup>2</sup>-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 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
```



```

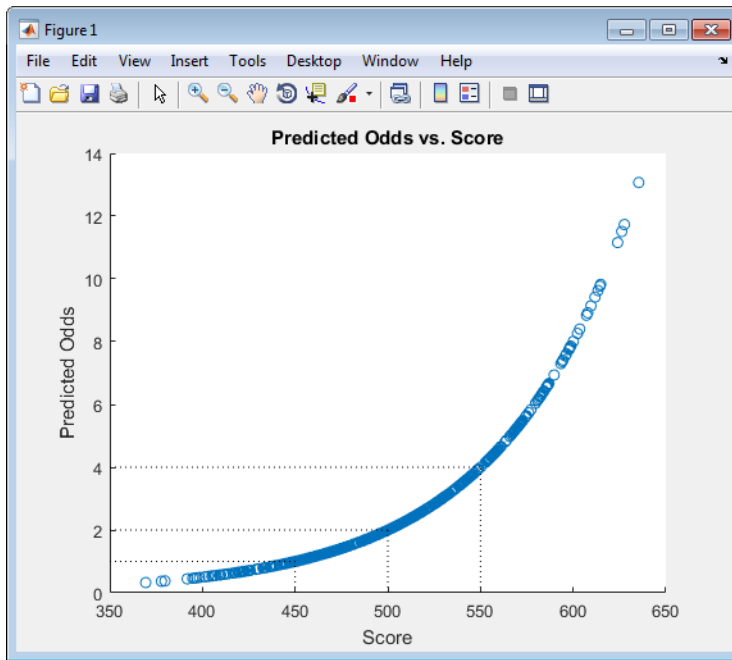
xLimits = xlim;
yLimits = ylim;

% 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`.

```

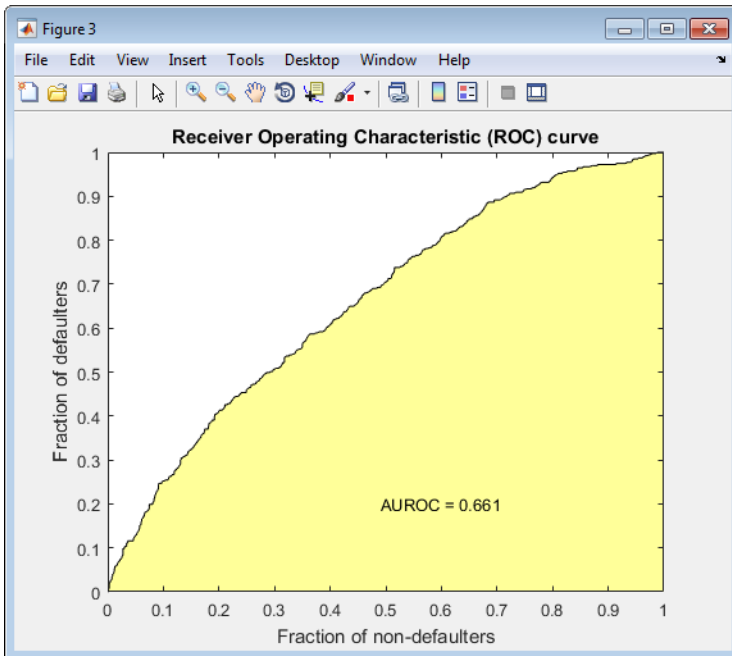
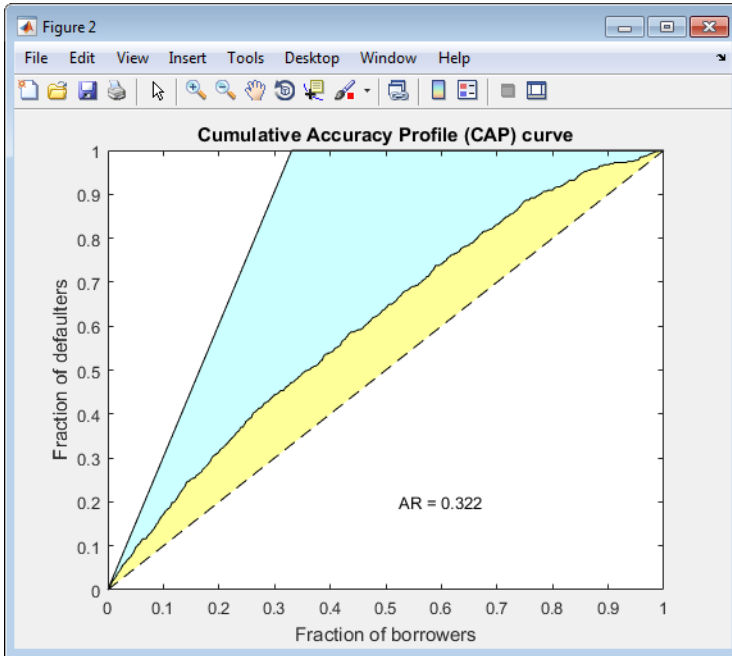
[Stats,T] = validatemodel(sc,'Plot',{'CAP','ROC','KS'});
disp(Stats)
disp(T(1:15,:))
    
```

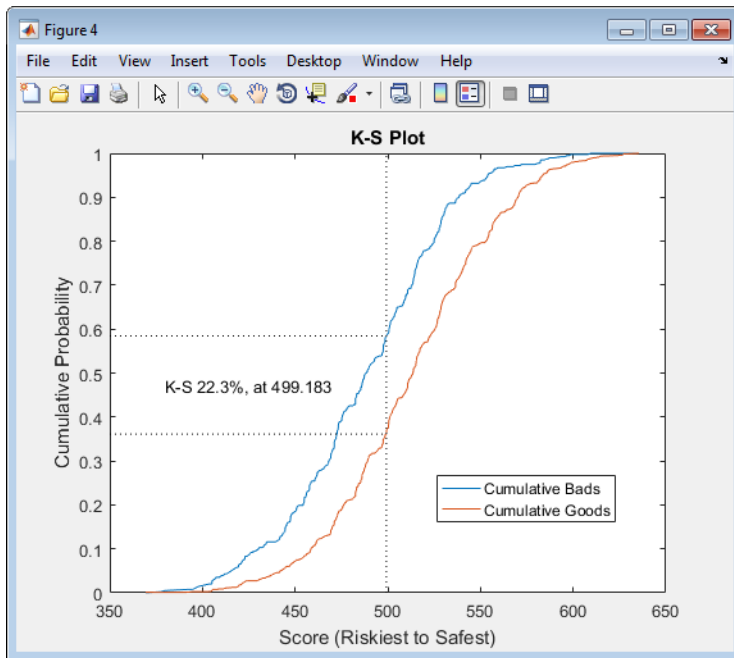
Measure		Value	
'Accuracy Ratio'		0.32225	
'Area under ROC curve'		0.66113	
'KS statistic'		0.22324	
'KS score'		499.18	

Scores	ProbDefault	TrueBads	FalseBads	TrueGoods	FalseGoods	Sensitivity	FalseAlarm	PctObs
369.4	0.7535	0	1	802	397	0	0.0012453	0.00083333
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](#) | [screenpredictors](#) | [setmodel](#) | [validatemodel](#)

## Related Examples

- “Common Binning Explorer Tasks” on page 3-4
- “Credit Scorecard Modeling with Missing Values” (Financial Toolbox)
- “Feature Screening with screenpredictors”
- “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 3-34

## More About

- “Overview of Binning Explorer” on page 3-2
- “About Credit Scorecards” (Financial Toolbox)
- “Credit Scorecard Modeling Workflow” (Financial Toolbox)
- Monotone Adjacent Pooling Algorithm (MAPA) (Financial Toolbox)
- “Credit Scorecard Modeling Using Observation Weights” (Financial Toolbox)

## External Websites

- Credit Scorecard Modeling Using the Binning Explorer App (6 min 17 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.

For additional information, refer to the example “Modeling Probabilities of Default with Cox Proportional Hazards”, which follows the same workflow but uses Cox regression instead of logistic regression, and also has additional information on the computation of lifetime PD and lifetime Expected Credit Loss (ECL).

### 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')
```

First ten rows:

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

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

```
fprintf('\nLast ten rows:\n')
```

Last ten rows:

```
disp(data(end-9:end,:))
```

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

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

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

Total number of IDs: 96820

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

```
Total number of rows: 646724
```

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

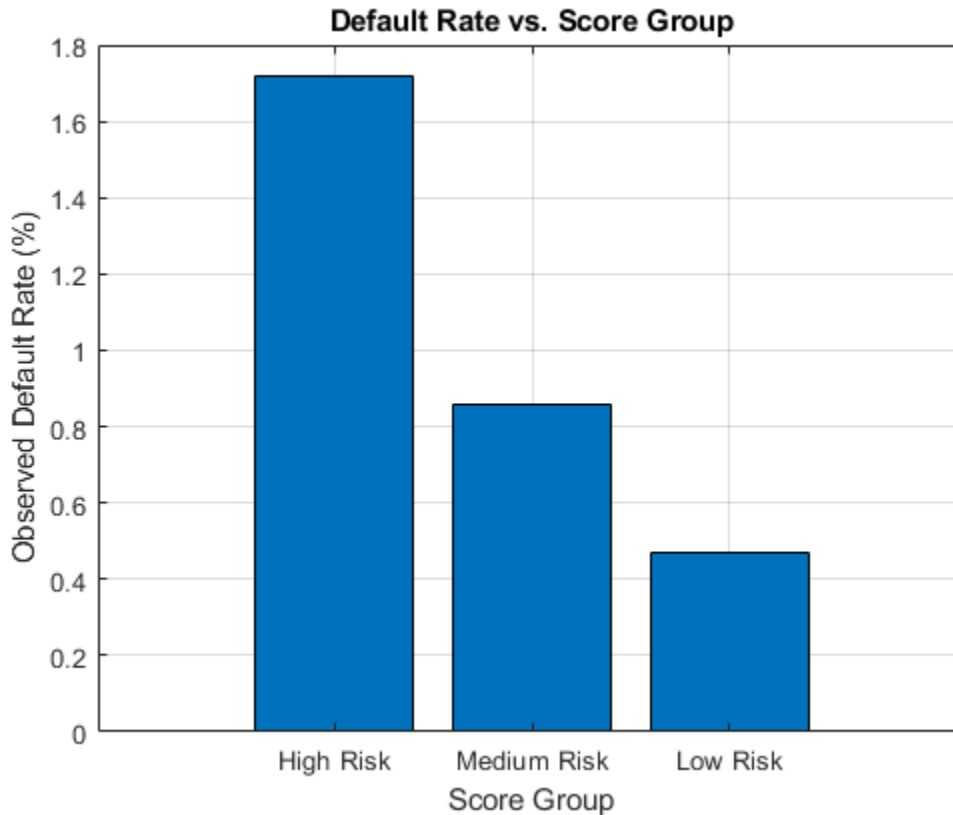
Use the credit score group as a grouping variable to compute the observed default rate for each score group. For this, use the `groupsummary` function to compute the mean of the `Default` variable, grouping by the `ScoreGroup` variable. Plot the results on a bar chart. As expected, the default rate goes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

```
DefRateByScore = groupsummary(data, 'ScoreGroup', 'mean', 'Default');  
NumScoreGroups = height(DefRateByScore);
```

```
disp(DefRateByScore)
```

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

```
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
```



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.

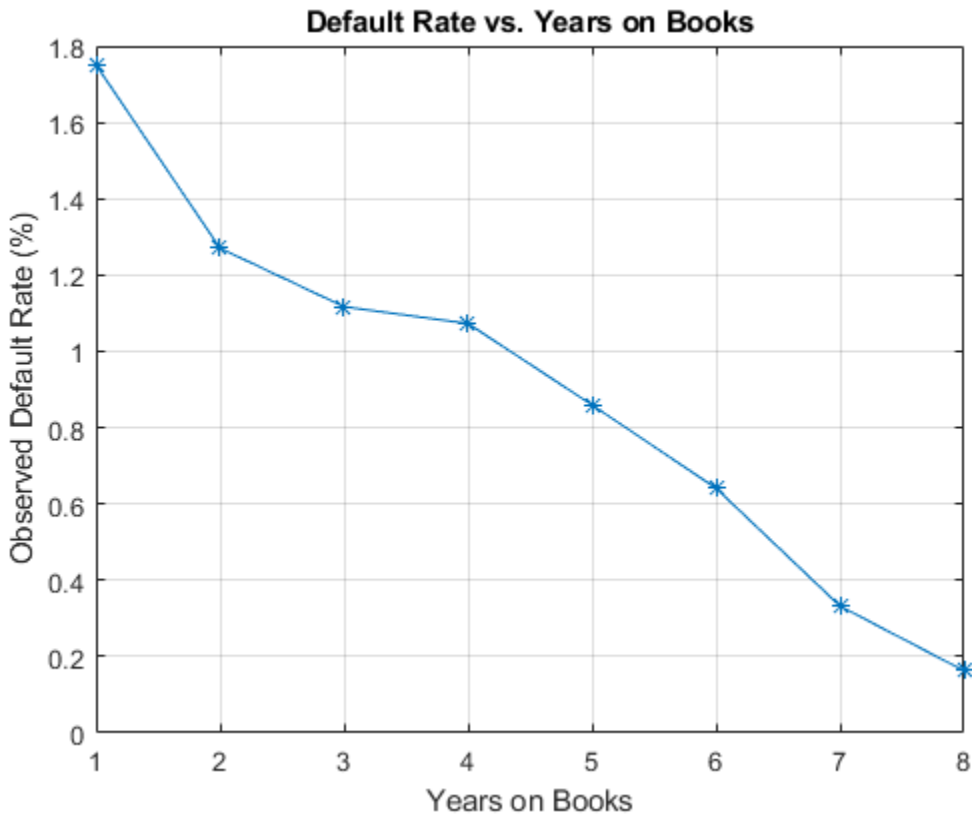
Plot the results. There is a clear downward trend, with default rates going down as the number of years on books increases. Years three and four have similar default rates. However, it is unclear from this plot whether this is a characteristic of the loan product or an effect of the macroeconomic environment.

```
DefRateByYOB = groupsummary(data, 'YOB', 'mean', 'Default');
NumYOB = height(DefRateByYOB);
```

```
disp(DefRateByYOB)
```

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

```
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
```



Now, group both by the score group and number of years on books and then 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 = groupsummary(data,{'ScoreGroup','YOB'},'mean','Default');
% Display output table to show the way it is structured
% Display only the first 10 rows, for brevity
disp(DefRateByScoreYOB(1:10,:))
```

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



```

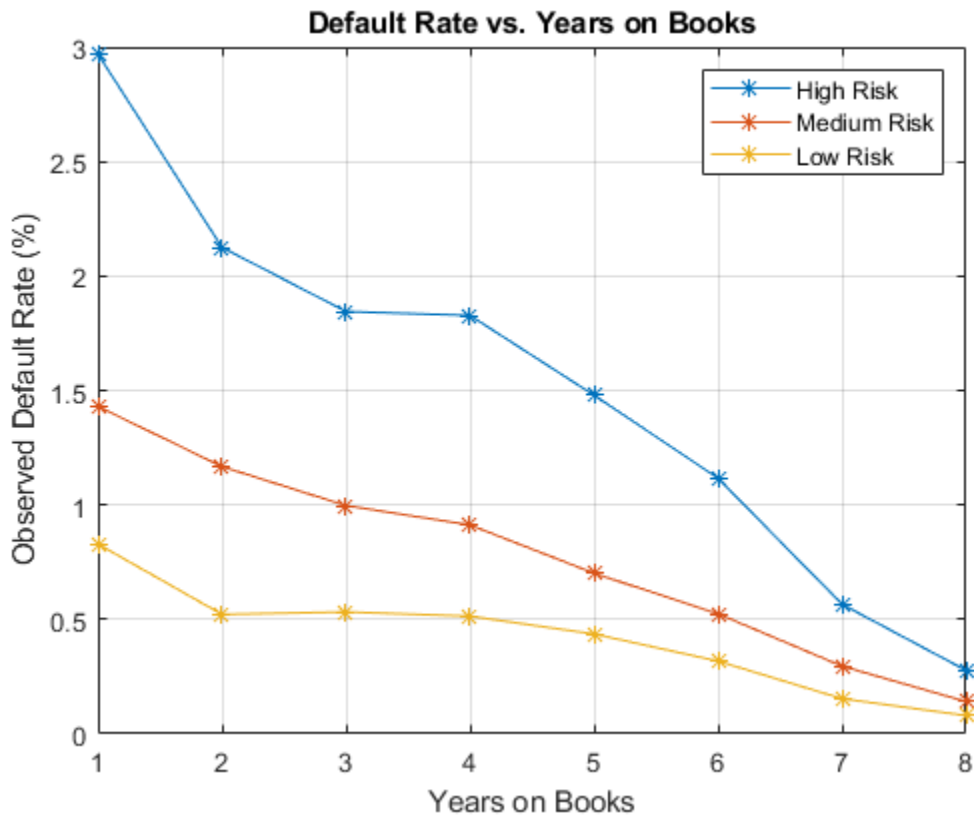
High Risk      7      19606      0.0056615
High Risk      8      10094      0.0027739
Medium Risk    1      32373      0.014302
Medium Risk    2      31775      0.011676
    
```

```
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
```



### 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.

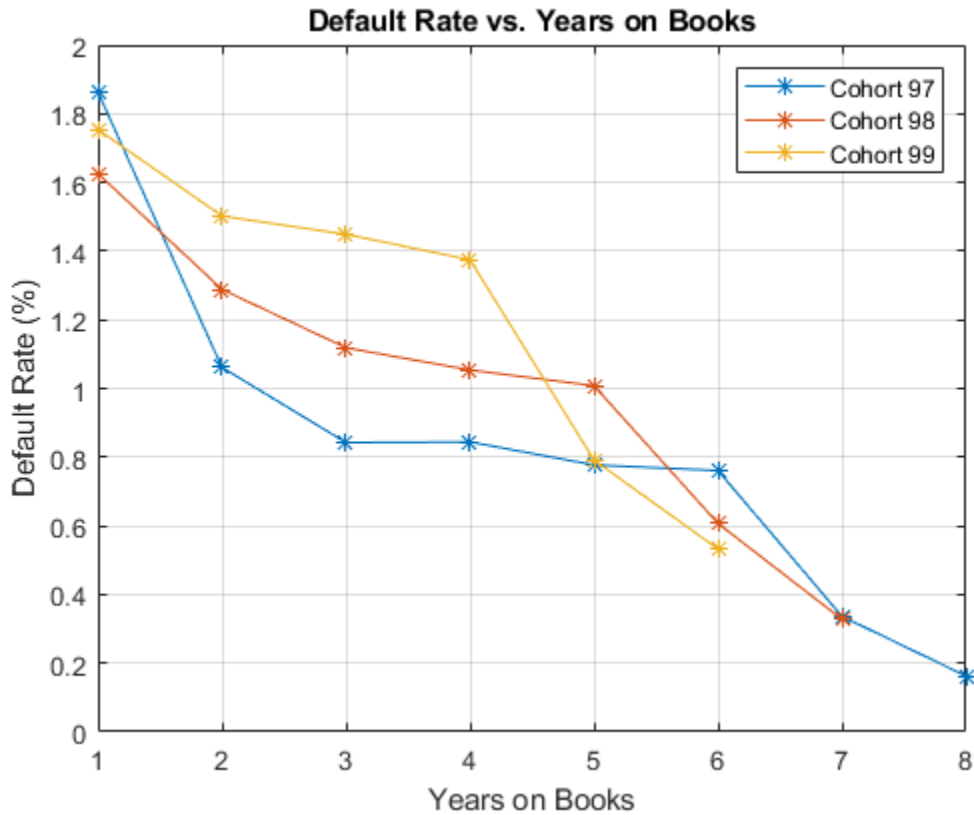
```
% Get IDs of 1997, 1998, and 1999 cohorts
IDs1997 = data.ID(data.YOB==1&data.Year==1997);
IDs1998 = data.ID(data.YOB==1&data.Year==1998);
IDs1999 = data.ID(data.YOB==1&data.Year==1999);
% IDs2000AndUp is unused, it is only computed to show that this is empty,
% no loans started after 1999
IDs2000AndUp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

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

ObsDefRate1998 = groupsummary(data(ismember(data.ID,IDs1998),:),...
    'YOB','mean','Default');

ObsDefRate1999 = groupsummary(data(ismember(data.ID,IDs1999),:),...
    'YOB','mean','Default');

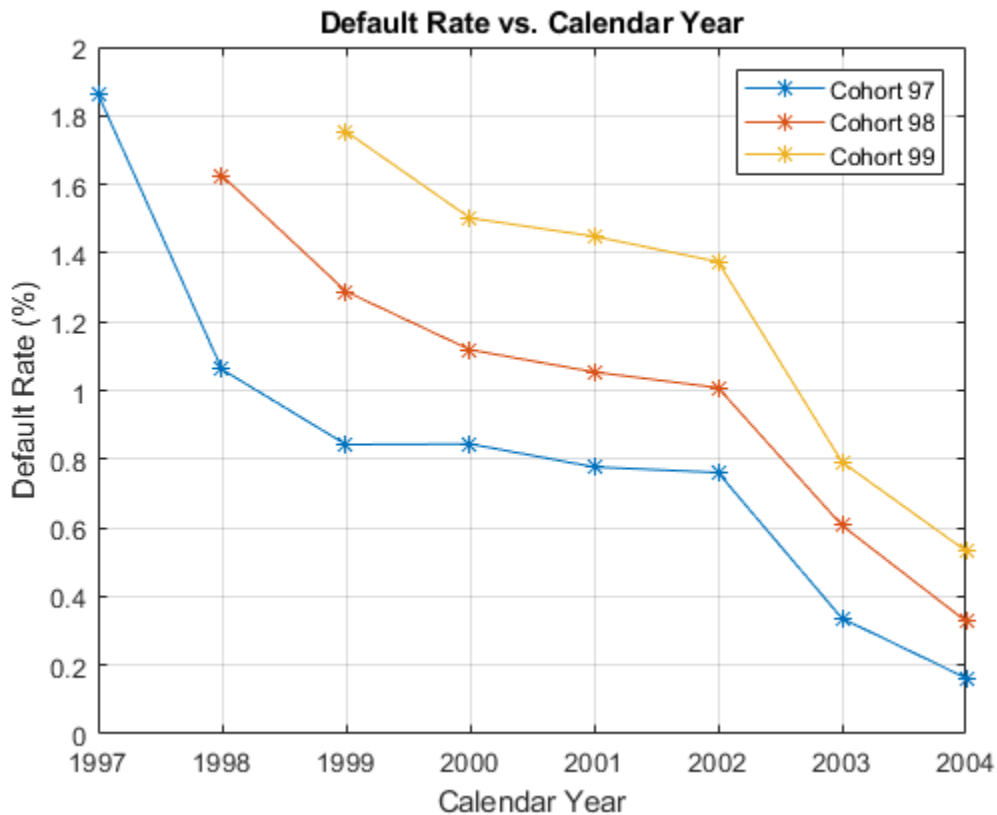
% Plot against the years on books
figure;
plot(ObsDefRate1997.YOB,ObsDefRate1997.mean_Default*100,'-*')
hold on
plot(ObsDefRate1998.YOB,ObsDefRate1998.mean_Default*100,'-*')
plot(ObsDefRate1999.YOB,ObsDefRate1999.mean_Default*100,'-*')
hold off
title('Default Rate vs. Years on Books')
xlabel('Years on 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  $a_H$  term), and the `ScoreGroup_Medium Risk` and `ScoreGroup_Low Risk` coefficients are the adjustments for Medium Risk and Low Risk levels (the  $a_M$  and  $a_L$  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 `fitglme` function from Statistics and Machine Learning Toolbox™. Although this model is not fitted in this example, the code is very similar:

```
ModelNoMacro = fitglme(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.

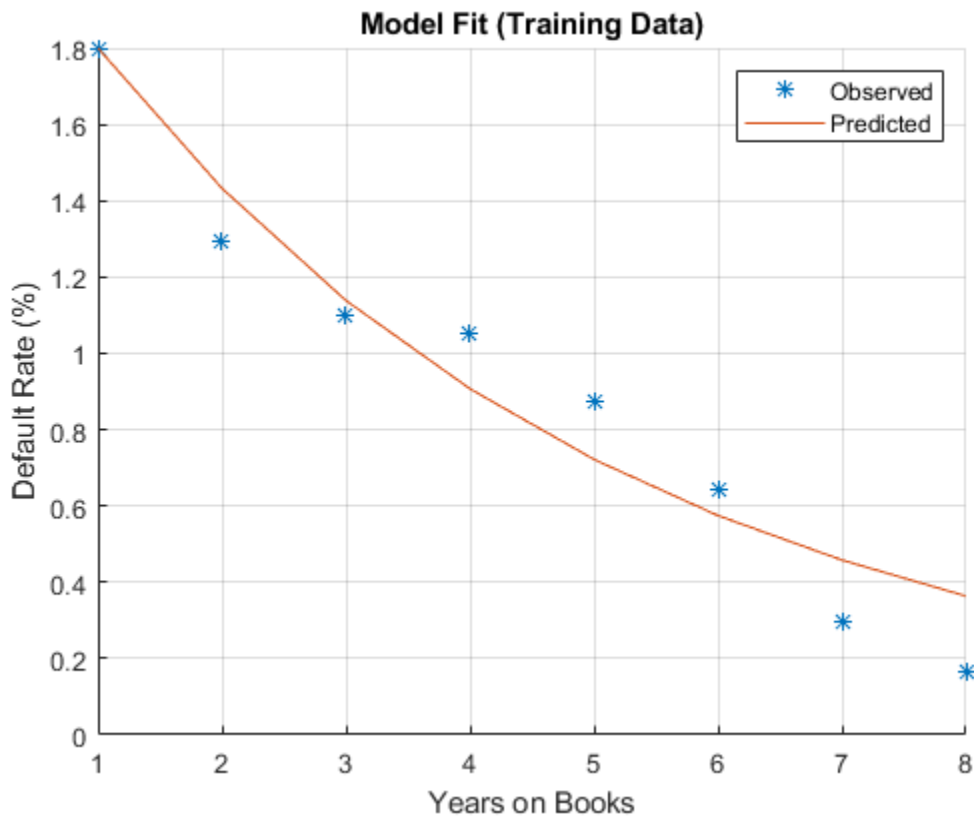
```
data.PDNoMacro = zeros(height(data),1);

% 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 = groupsummary(data(TrainDataInd,:), 'YOB', 'mean', ...
    {'Default', 'PDNoMacro'});

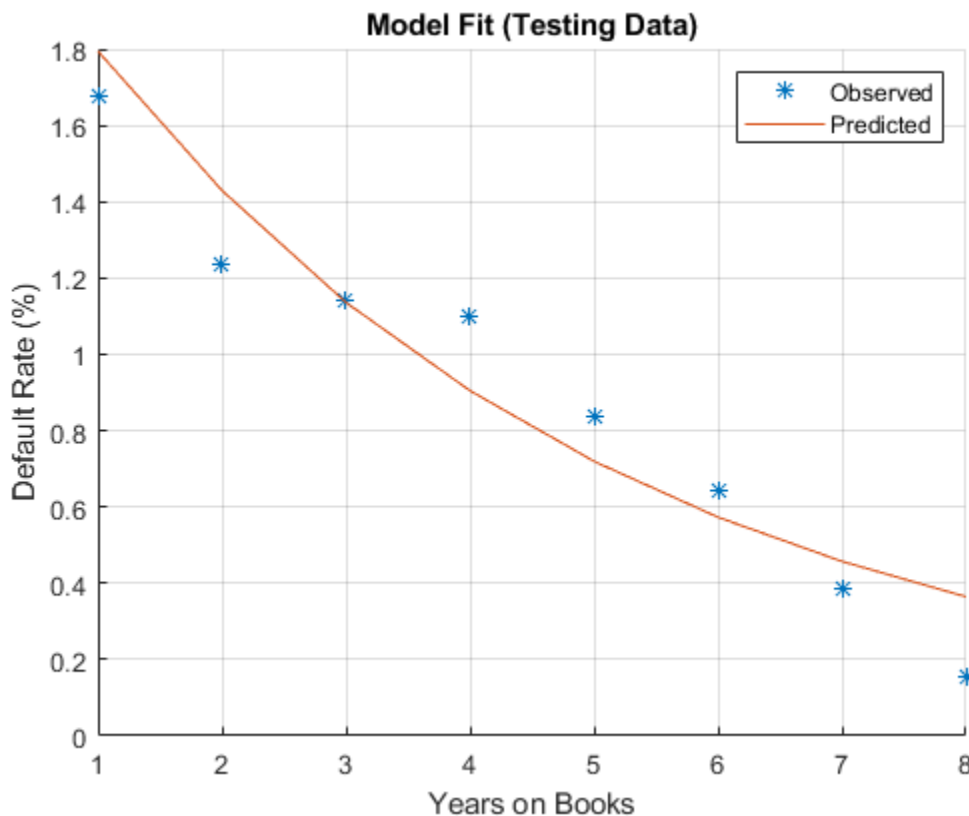
figure;
scatter(PredPDTrainYOB.YOB,PredPDTrainYOB.mean_Default*100, '*');
hold on
plot(PredPDTrainYOB.YOB,PredPDTrainYOB.mean_PDNoMacro*100);
hold off
xlabel('Years on Books')
ylabel('Default Rate (%)')
legend('Observed','Predicted')
title('Model Fit (Training Data)')
grid on
```



Visualize the out-of-sample fit.

```
PredPDTestYOB = groupsummary(data(TestDataInd,:), 'YOB', 'mean', ...
    {'Default', 'PDNoMacro'});
```

```
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.

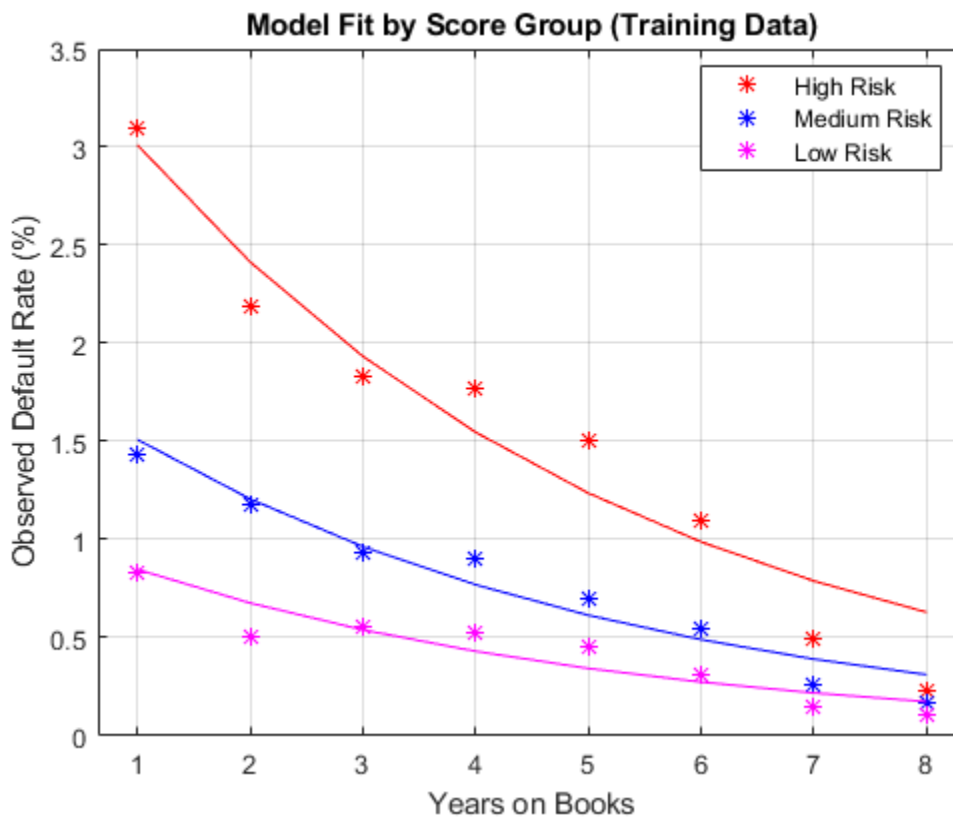
```
PredPDTrainScoreYOB = groupsummary(data(TrainDataInd,:), ...
    {'ScoreGroup', 'YOB'}, 'mean', {'Default', 'PDNoMacro'});
```

```
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.

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

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,:));
```

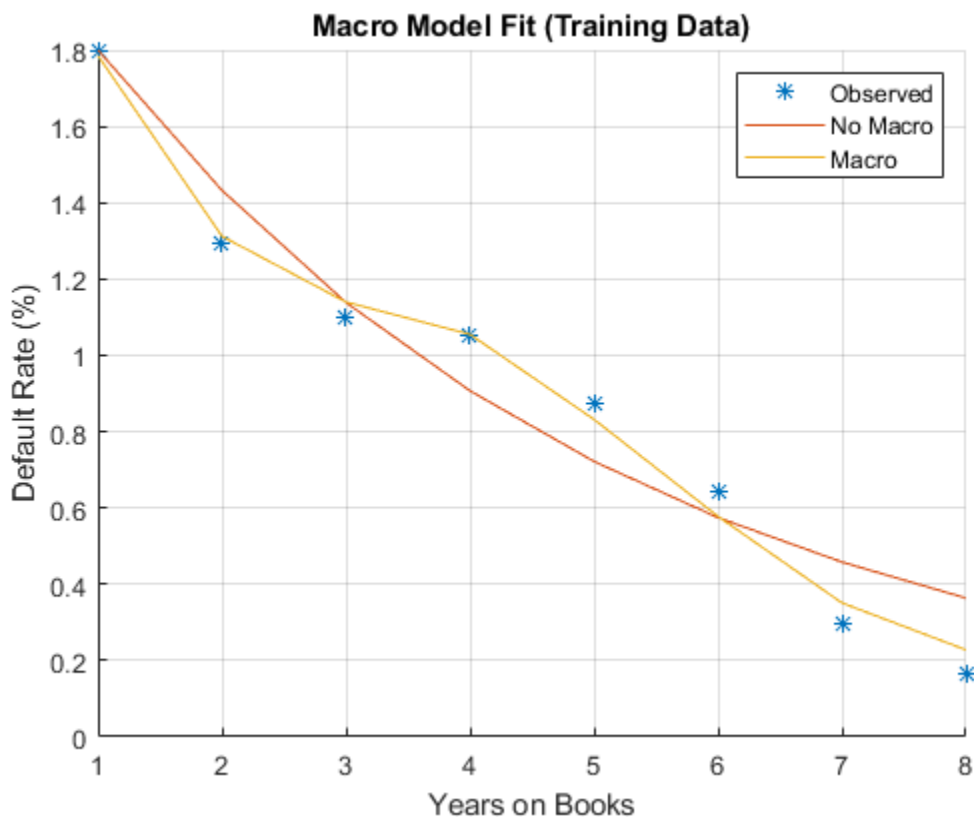
Visualize the in-sample fit. As desired, the model including macroeconomic variables, or macro model, deviates from the smooth trend predicted by the previous model. The rates predicted with the macro model match more closely with the observed default rates.

```

PredPDTrainYOBMacro = groupsummary(data(TrainDataInd,:), 'YOB', 'mean', ...
    {'Default', 'PDMacro'});

figure;
scatter(PredPDTrainYOBMacro.YOB, PredPDTrainYOBMacro.mean_Default*100, '*');
hold on
plot(PredPDTrainYOB.YOB, PredPDTrainYOB.mean_PDNoMacro*100); % No Macro
plot(PredPDTrainYOBMacro.YOB, PredPDTrainYOBMacro.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 = groupsummary(data(TestDataInd,:), 'YOB', 'mean', ...
    {'Default', 'PDMacro'});

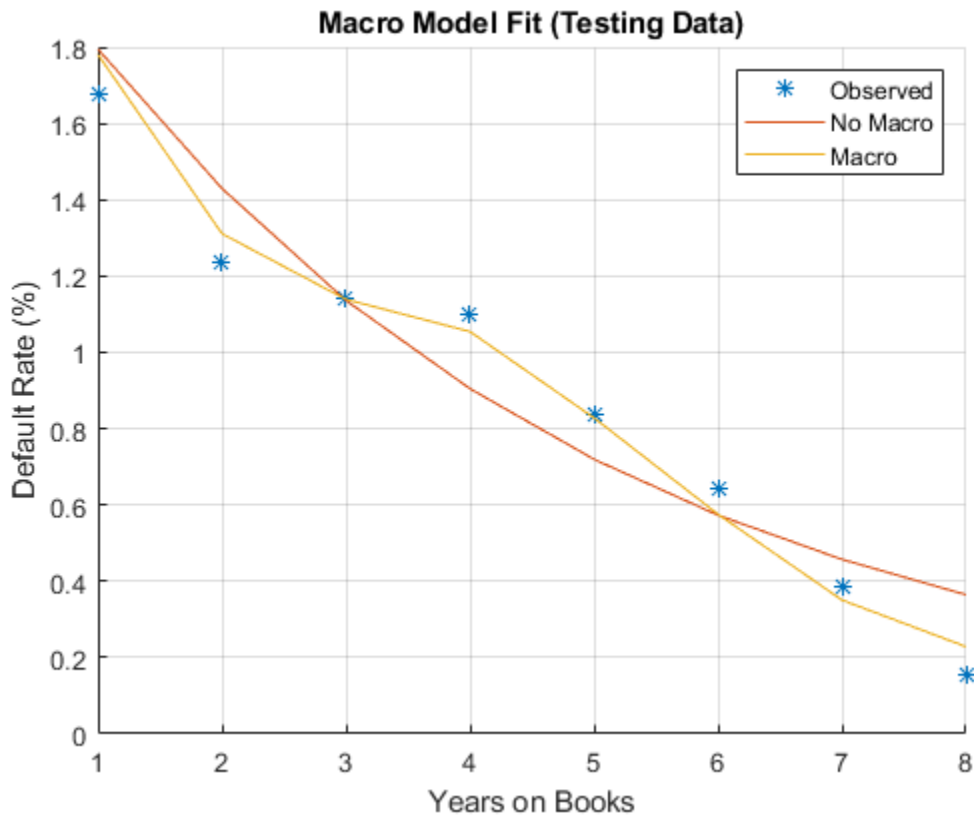
figure;
scatter(PredPDTestYOBMacro.YOB, PredPDTestYOBMacro.mean_Default*100, '*');
hold on
plot(PredPDTestYOB.YOB, PredPDTestYOB.mean_PDNoMacro*100); % No Macro
plot(PredPDTestYOBMacro.YOB, PredPDTestYOBMacro.mean_PDMacro*100); % Macro
hold off
xlabel('Years on Books')
ylabel('Default Rate (%)')

```

```

legend('Observed', 'No Macro', 'Macro')
title('Macro Model Fit (Testing Data)')
grid on

```



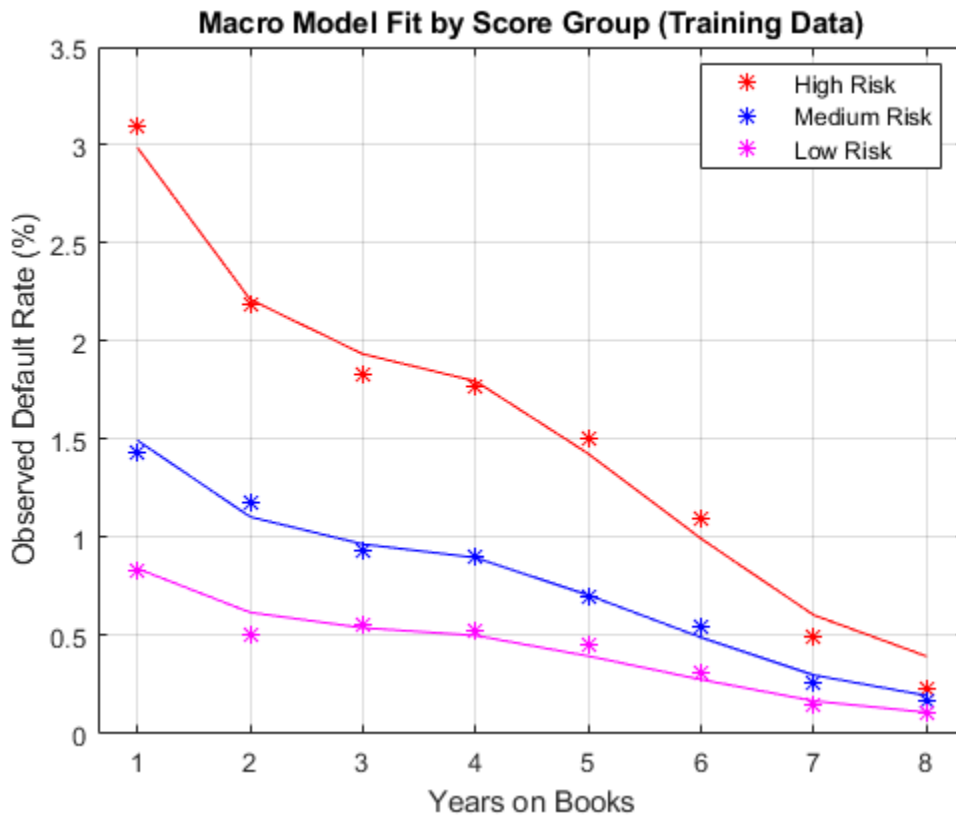
Visualize the in-sample fit for all score groups.

```

PredPDTrainScoreYOBBMacro = groupsummary(data(TrainDataInd,:), ...
    {'ScoreGroup', 'YOB'}, 'mean', {'Default', 'PDMacro'});

figure;
hs = gscatter(PredPDTrainScoreYOBBMacro.YOB, ...
    PredPDTrainScoreYOBBMacro.mean_Default*100, ...
    PredPDTrainScoreYOBBMacro.ScoreGroup, 'rbmgk', '*');
mean_PDMacroMat = reshape(PredPDTrainScoreYOBBMacro.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);

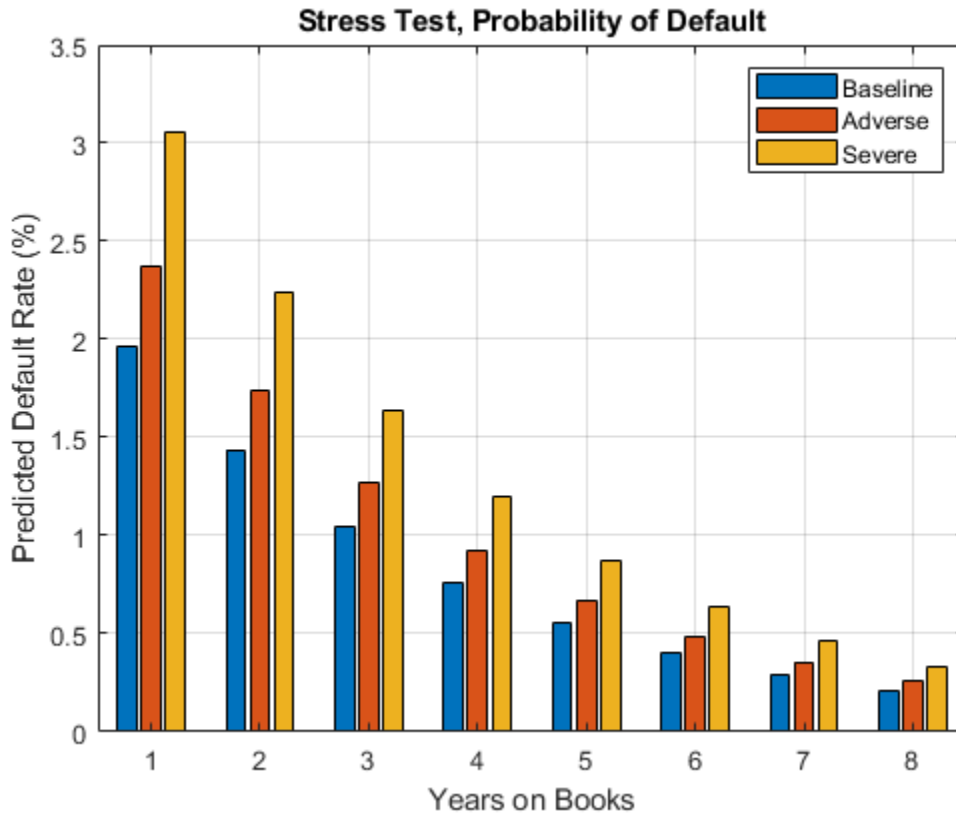
% Predict the probabilities of default in the adverse scenario
dataAdverse = dataBaseline;
dataAdverse.GDP(:) = dataMacroStress.GDP('Adverse');
dataAdverse.Market(:) = dataMacroStress.Market('Adverse');
dataAdverse.PD = predict(ModelMacro,dataAdverse);

% 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: <https://www.mathworks.com/help/stats/generalized-linear-regression.html>.
- 2 Generalized Linear Mixed Effects Models documentation: <https://www.mathworks.com/help/stats/generalized-linear-mixed-effects-models.html>.
- 3 Federal Reserve, Comprehensive Capital Analysis and Review (CCAR): <https://www.federalreserve.gov/bankinfo/ccar.htm>.
- 4 Bank of England, Stress Testing: <https://www.bankofengland.co.uk/financial-stability>
- 5 European Banking Authority, EU-Wide Stress Testing: <https://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)
- “Credit Scorecard Modeling with Missing Values” (Financial 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 4-5
- “creditMigrationCopula Simulation Workflow” on page 4-10
- “Modeling Correlated Defaults with Copulas” on page 4-18
- “Analyze the Sensitivity of Concentration to a Given Exposure” on page 4-28
- “Compare Concentration Indices for Random Portfolios” on page 4-30
- “Comparison of the Merton Model Single-Point Approach to the Time-Series Approach” on page 4-33
- “Calculating Regulatory Capital with the ASRF Model” on page 4-38

## Credit Simulation Using Copulas

### In this section...

“Factor Models” on page 4-2

“Supported Simulations” on page 4-3

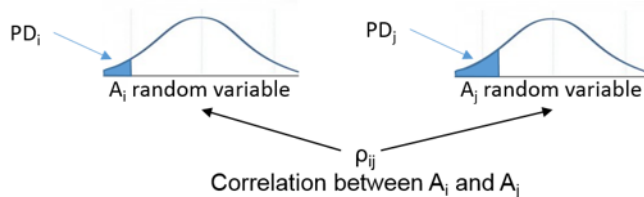
Predicting the credit losses for a counterparty depends on three main elements:

- 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  $PD \times EAD \times LGD$ . 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.

$$A_i = w_{i1}Z_1 + \dots + w_{iK}Z_K + w_{i\epsilon}\epsilon_i$$

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

`asrf` | `creditDefaultCopula` | `creditMigrationCopula`

### Related Examples

- “`creditDefaultCopula` Simulation Workflow”
- “`creditMigrationCopula` Simulation Workflow” on page 4-10
- “Modeling Correlated Defaults with Copulas” on page 4-18
- “One-Factor Model Calibration”

### More About

- “Corporate Credit Risk” on page 1-3
- “Credit Rating Migration Risk” on page 1-7

## 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” on page 4-18.

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

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

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

```
creditDefaultCopula with properties:
```

```
Portfolio: [100x5 table]
FactorCorrelation: [2x2 double]
VaRLevel: 0.9500
UseParallel: 0
PortfolioLosses: []
```

```
disp(cdc.Portfolio(1:10:100,:))
```

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: [100x5 table]
```

```
FactorCorrelation: [2x2 double]
    VaRLevel: 0.9900
    UseParallel: 0
PortfolioLosses: [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(cdc)
```

```
portRisk=1x4 table
```

EL	Std	VaR	CVaR
24.876	23.778	102.4	121.28

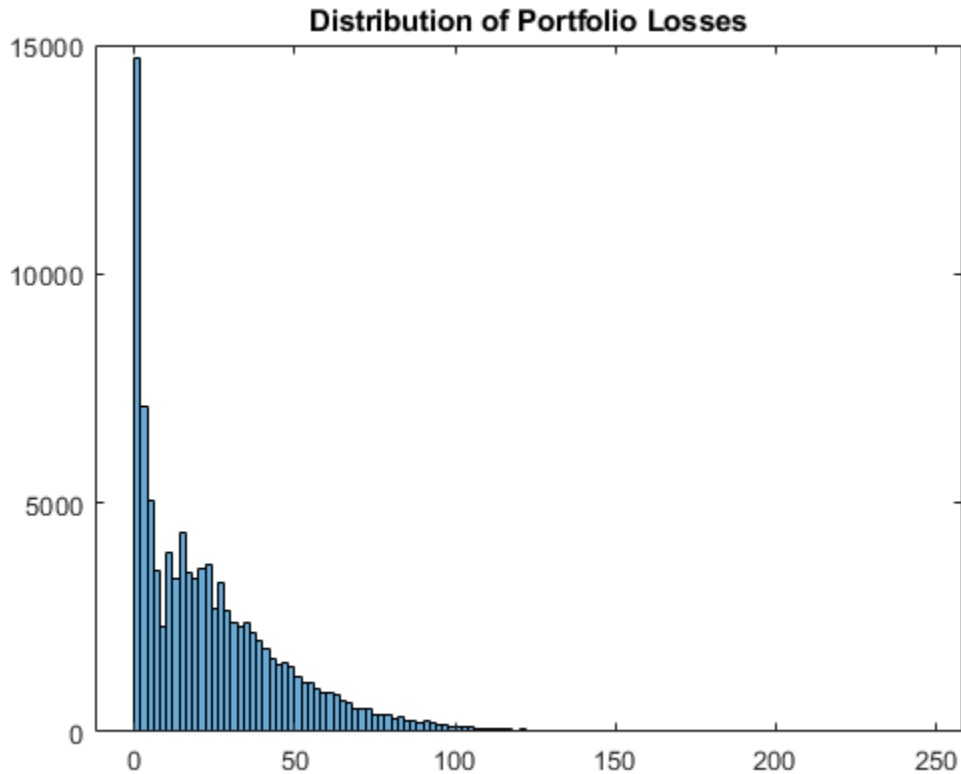
```
RiskConfidenceInterval=1x4 table
```

EL		Std	VaR		CVaR		
24.729	25.023	23.674	23.883	101.19	103.5	120.13	122.42

**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	Std	VaR	CVaR
1	0.036031	0.022762	0.083828	0.13625
2	0.068357	0.039295	0.23373	0.24984
3	1.2228	0.60699	2.3184	2.3775
4	0.002877	0.00079014	0.0024248	0.0013137
5	0.12127	0.037144	0.18474	0.24622
6	0.12638	0.078506	0.39779	0.48334
7	0.84284	0.3541	1.6221	1.8183
8	0.00090088	0.00011379	0.0016463	0.00089197
9	0.93117	0.87638	3.3868	3.9936
10	0.26054	0.37918	1.7399	2.3042

### 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
      UseParallel: 0
  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)
```

```
ans=10x4 table
  NumScenarios  Lower  Std  Upper
  _____  _____  _____  _____
      10000      23.525  23.799  24.079
      20000      23.564  23.758  23.955
      30000      23.543  23.701  23.861
      40000      23.621  23.758  23.897
      50000      23.565  23.687  23.811
      60000      23.604  23.716  23.829
      70000      23.688  23.792  23.897
      80000      23.663  23.76  23.858
      90000      23.639  23.73  23.823
     1e+05      23.691  23.778  23.866
```

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

```
ans=10x4 table
  NumScenarios  Lower  Std  Upper
  _____  _____  _____  _____
      10000      31.923  32.294  32.675
      20000      31.775  32.036  32.302
      30000      31.759  31.972  32.188
      40000      31.922  32.107  32.295
      50000      32.012  32.179  32.347
      60000      31.911  32.062  32.216
      70000      31.879  32.019  32.161
      80000      31.909  32.04  32.173
      90000      31.866  31.99  32.114
     1e+05      31.933  32.05  32.169
```

### See Also

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



## **Related Examples**

- “Credit Simulation Using Copulas” on page 4-2
- “creditMigrationCopula Simulation Workflow” on page 4-10
- “Modeling Correlated Defaults with Copulas” on page 4-18
- “One-Factor Model Calibration”

## **More About**

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

## 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 `creditMigrationCopula`.

```
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
    UseParallel: 0
    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. Display the `Portfolio` property for information about migration values, ratings, LGDs, and weights.

Display the `Portfolio` property containing information about migration values, ratings, LGDs, and weights. The columns in the migration values are in the same order of the ratings, with the default rating in the last column.

```
head(cmc.Portfolio)
```

```
ans=8x5 table
   ID  MigrationValues  Rating  LGD  Weights
   --  -
```

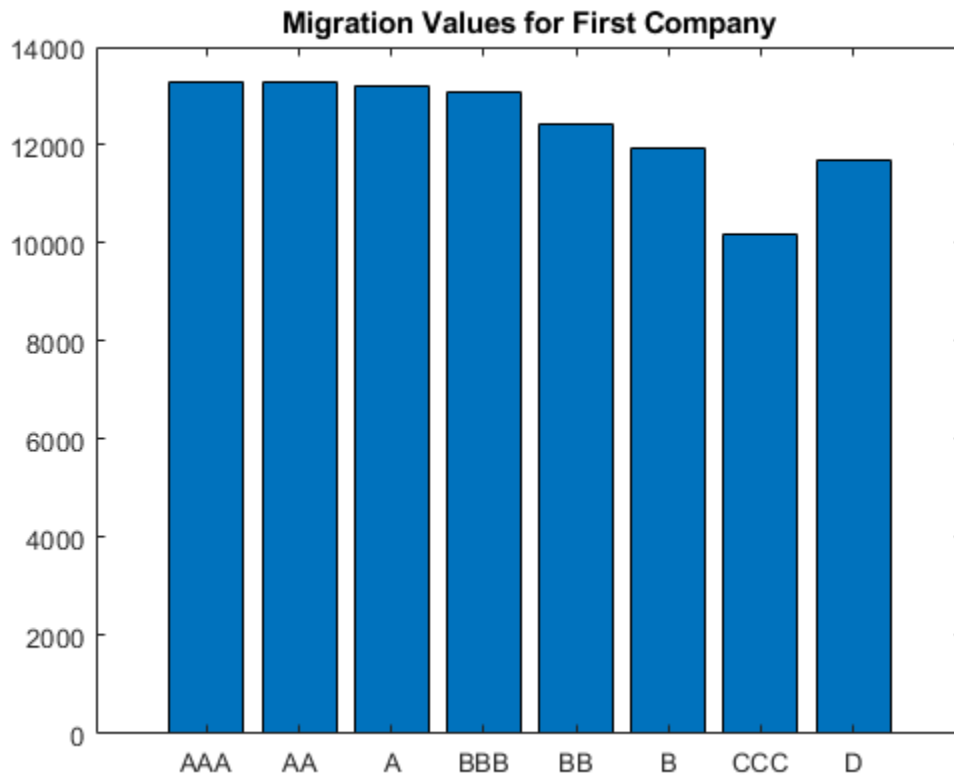
ID	MigrationValues	Rating	LGD	Weights
1	[1x8 double]	"A"	0.6509	0 0 0 0.5 0.5
2	[1x8 double]	"BBB"	0.8283	0 0.55 0 0 0.45
3	[1x8 double]	"AA"	0.6041	0 0.7 0 0 0.3
4	[1x8 double]	"BB"	0.6509	0 0.55 0 0 0.45
5	[1x8 double]	"BBB"	0.4966	0 0 0.75 0 0.25
6	[1x8 double]	"BB"	0.8283	0 0 0 0.65 0.35
7	[1x8 double]	"BB"	0.6041	0 0 0 0.65 0.35

```
8 [1x8 double] "BB" 0.4873 0.5 0 0 0 0.5
```

#### Step 4. Display migration values for a counterparty.

For example, you can display the migration values for the first counterparty. Note that the value for default is higher than some of the non-default ratings. This is because the migration value for the default rating is a reference value (for example, face value, forward value at current rating, or other) that is multiplied by the recovery rate during the simulation to get the value of the asset in the event of default. The recovery rate is  $1 - \text{LGD}$  when the LGD input to `creditMigrationCopula` is a constant LGD value (the LGD input has one column). The recovery rate is a random quantity when the LGD input to `creditMigrationCopula` is specified as a mean and standard deviation for a beta distribution (the LGD input has two columns).

```
bar(cmc.Portfolio.MigrationValues(1,:))
xtickLabels(cmc.RatingLabels)
title('Migration Values for First Company')
```



#### Step 5. 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
UseParallel: 0
PortfolioValues: [1x100000 double]
```

**Step 6. 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)
```

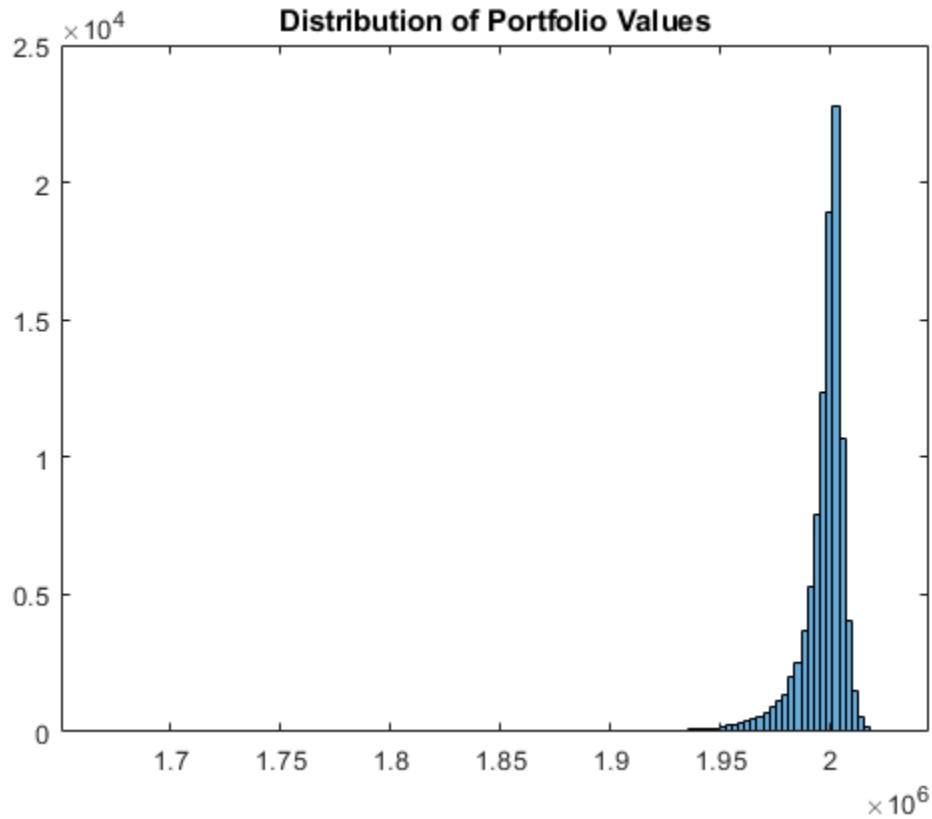
```
portRisk=1x4 table
      EL      Std      VaR      CVaR
-----
4515.9  12963  57176  83975
```

```
RiskConfidenceInterval=1x4 table
      EL      Std      VaR      CVaR
-----
4435.6  4596.3  12907  13021  55739  58541  82137  85812
```

**Step 7. Visualize the distribution.**

View a histogram of the portfolio values.

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

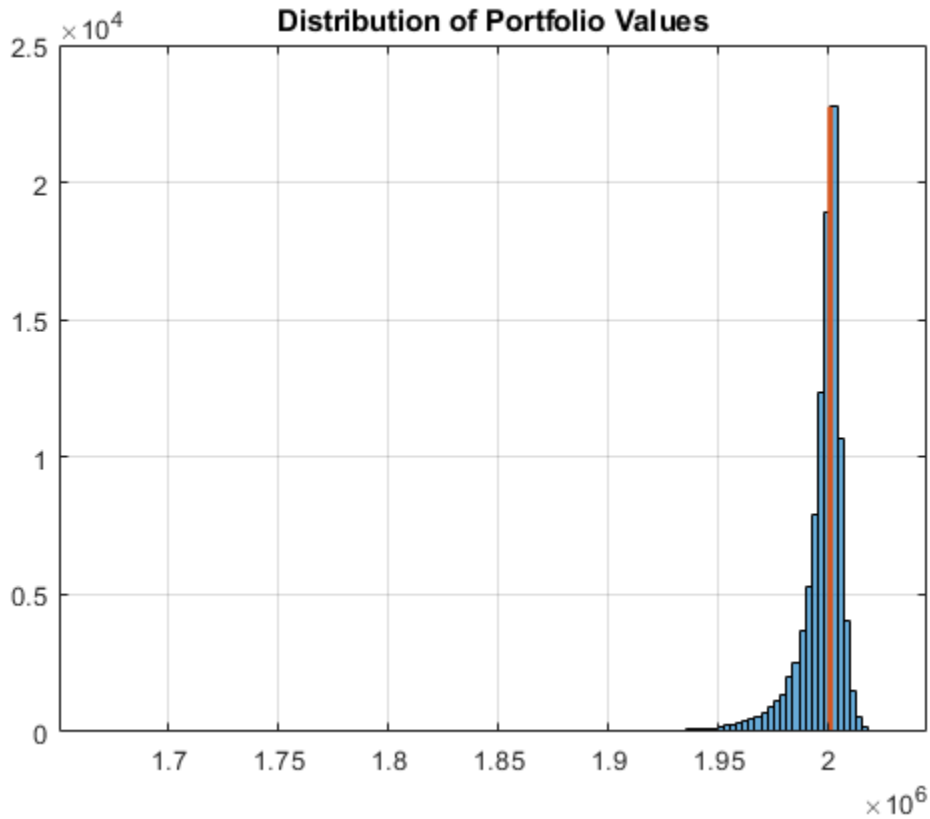


**Step 8. Overlay the value if all counterparties maintain current credit ratings.**

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 9. 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	Std	VaR	CVaR
1	15.521	41.153	238.72	279.18
2	8.49	18.838	92.074	122.19
3	6.0937	20.069	113.22	181.53
4	6.6964	55.885	272.23	313.25
5	23.583	73.905	360.32	573.39
6	10.722	114.97	445.94	728.38
7	1.8393	84.754	262.32	490.39
8	11.711	39.768	175.84	253.29
9	2.2154	4.4038	22.797	31.039
10	1.7453	2.5545	9.8801	17.603

**Step 10. 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
    UseParallel: 0
  PortfolioValues: [1x100000 double]
```

### Step 11. 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=1x4 table
```

EL	Std	VaR	CVaR
4544	17034	72270	1.2391e+05

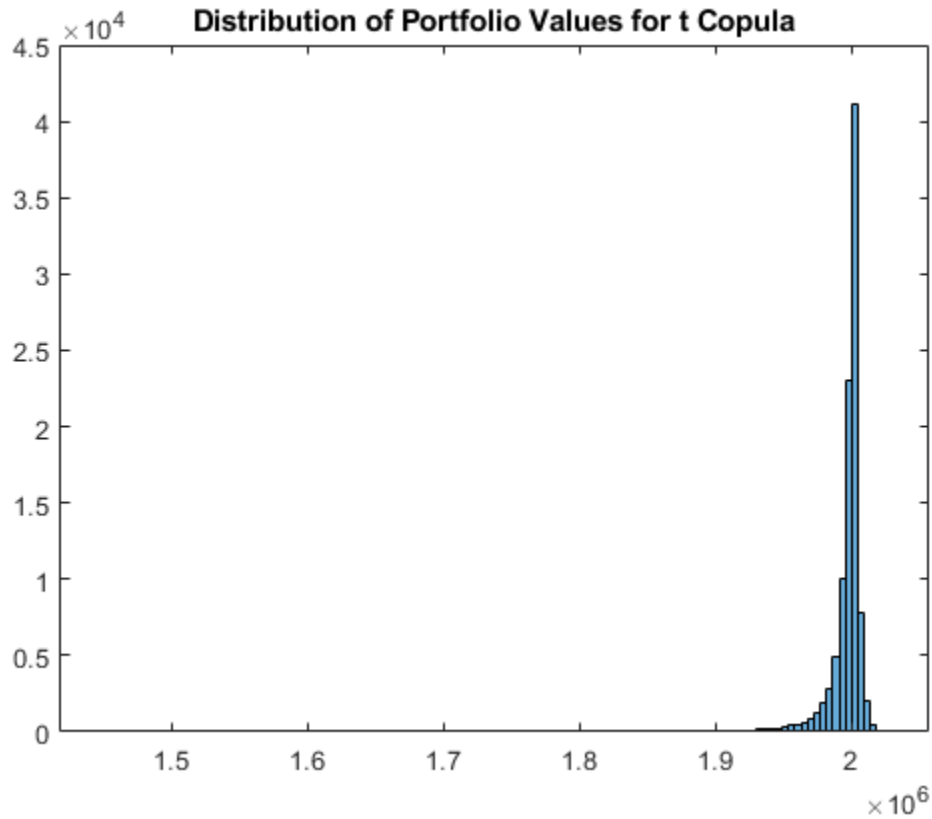
```
RiskConfidenceInterval2=1x4 table
```

EL		Std		VaR		CVaR	
4438.5	4649.6	16960	17109	69769	75382	1.1991e+05	1.2791e+05

### Step 12. 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');
```



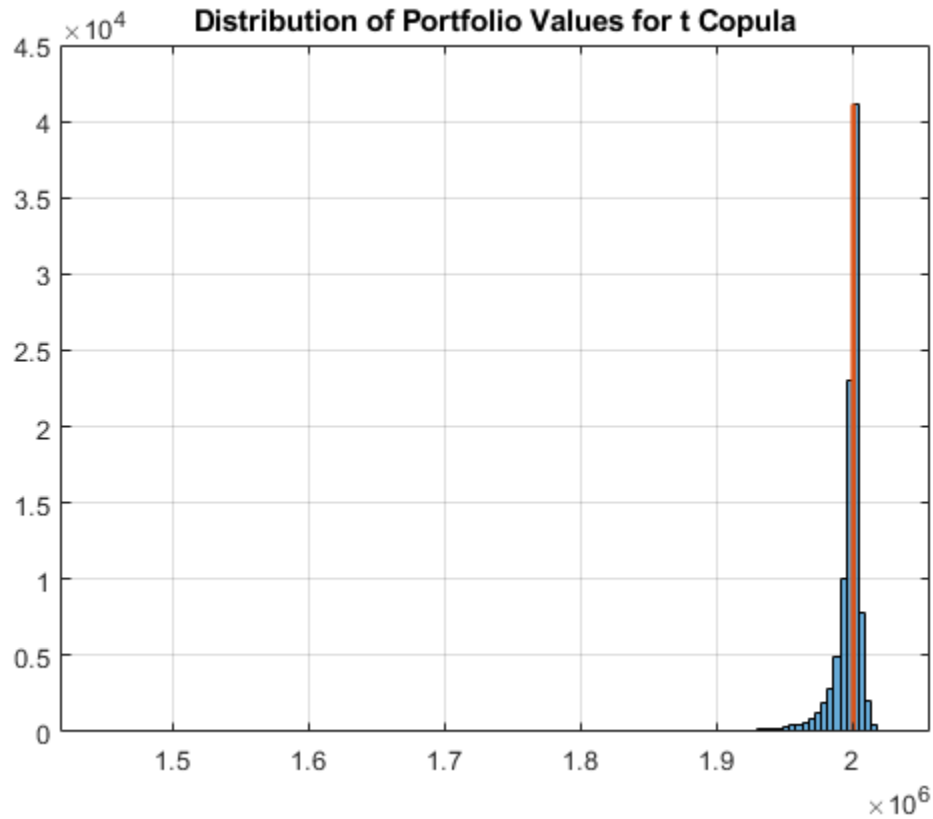
**Step 13. Overlay the value if all counterparties maintain current credit ratings for t copula.**

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

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

### Related Examples

- "Credit Simulation Using Copulas" on page 4-2
- "creditDefaultCopula Simulation Workflow" on page 4-5
- "Modeling Correlated Defaults with Copulas" on page 4-18
- "One-Factor Model Calibration"

### More About

- "Credit Rating Migration Risk" on page 1-7

## 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
```

```

        UseParallel: 0
        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: [100x5 table]
  FactorCorrelation: [2x2 double]
    VaRLevel: 0.9900
    UseParallel: 0
  PortfolioLosses: [1x100000 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
      ---  ---  ---  ---
    24.876  23.778  102.4  121.28

```

Confidence intervals for the risk measures:

EL		Std		VaR		CVaR	
24.729	25.023	23.674	23.883	101.19	103.5	120.13	122.42

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.

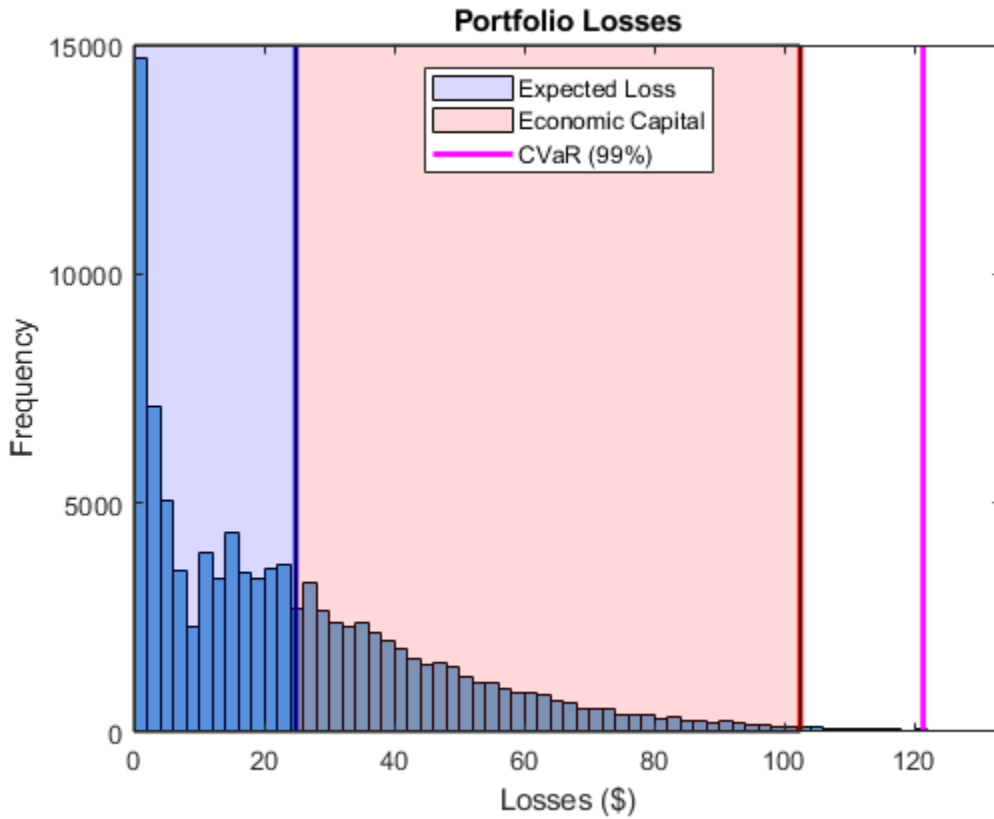
```

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 =

5x5 table

ID	EL	Std	VaR	CVaR
1	0.036031	0.022762	0.083828	0.13625
2	0.068357	0.039295	0.23373	0.24984
3	1.2228	0.60699	2.3184	2.3775
4	0.002877	0.00079014	0.0024248	0.0013137
5	0.12127	0.037144	0.18474	0.24622

Find the riskiest counterparties by their CVaR contributions.

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

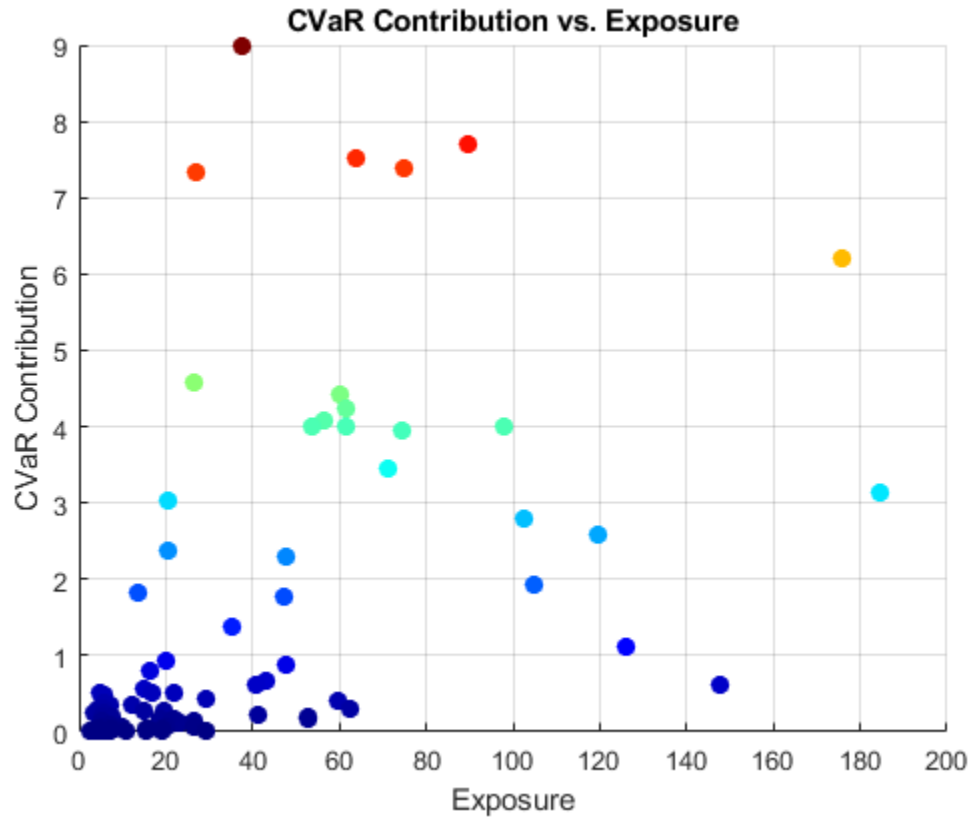
```
ans =
```

```
5x5 table
```

ID	EL	Std	VaR	CVaR
89	2.2647	2.2063	8.2676	8.9997
96	1.3515	1.6514	6.6157	7.7062
66	0.90459	1.474	6.4168	7.5149
22	1.5745	1.8663	6.0121	7.3814
16	1.6352	1.5288	6.3404	7.3462

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 =
```

```
5x4 table
```

NumScenarios	Lower	CVaR	Upper
1000	106.7	121.99	137.28
2000	109.18	117.28	125.38
3000	114.68	121.63	128.58
4000	114.02	120.06	126.11
5000	114.77	120.36	125.94

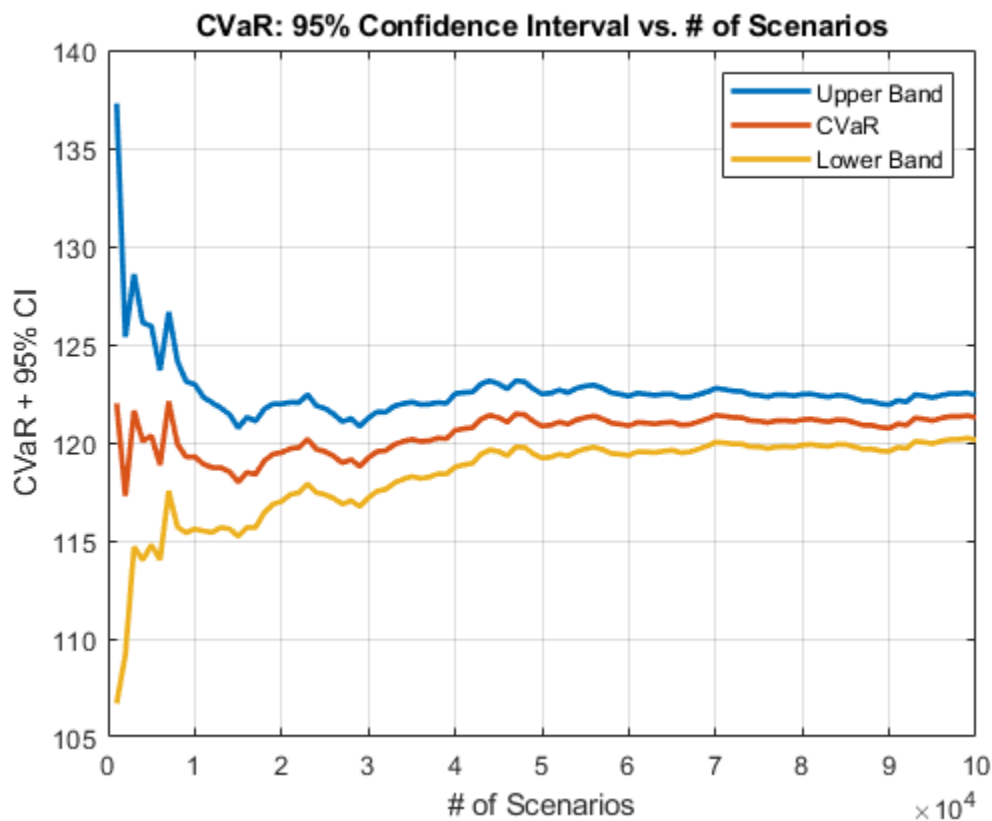
Plot the confidence bands to see how quickly the estimates converge.

```

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;
```

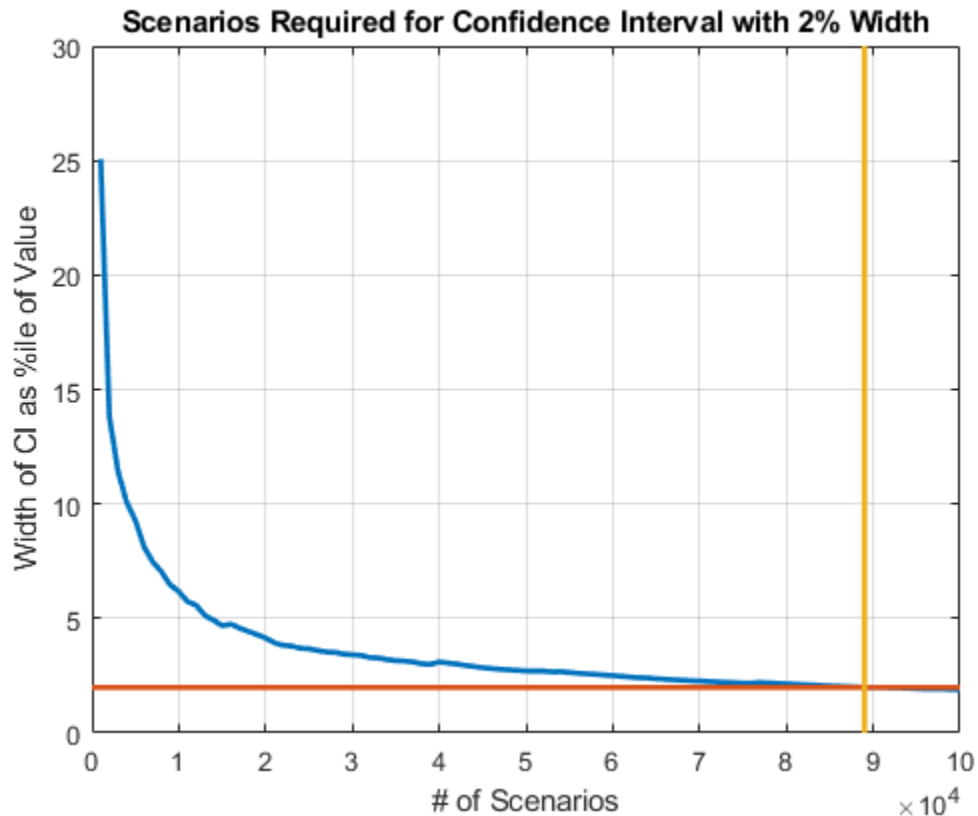
```
scenIdx = find(width <= thresh,1,'first');
```



```

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

```

24.876	23.778	102.4	121.28
--------	--------	-------	--------

Portfolio risk with t copula (dof = 5):			
EL	Std	VaR	CVaR
24.808	38.749	186.08	250.59

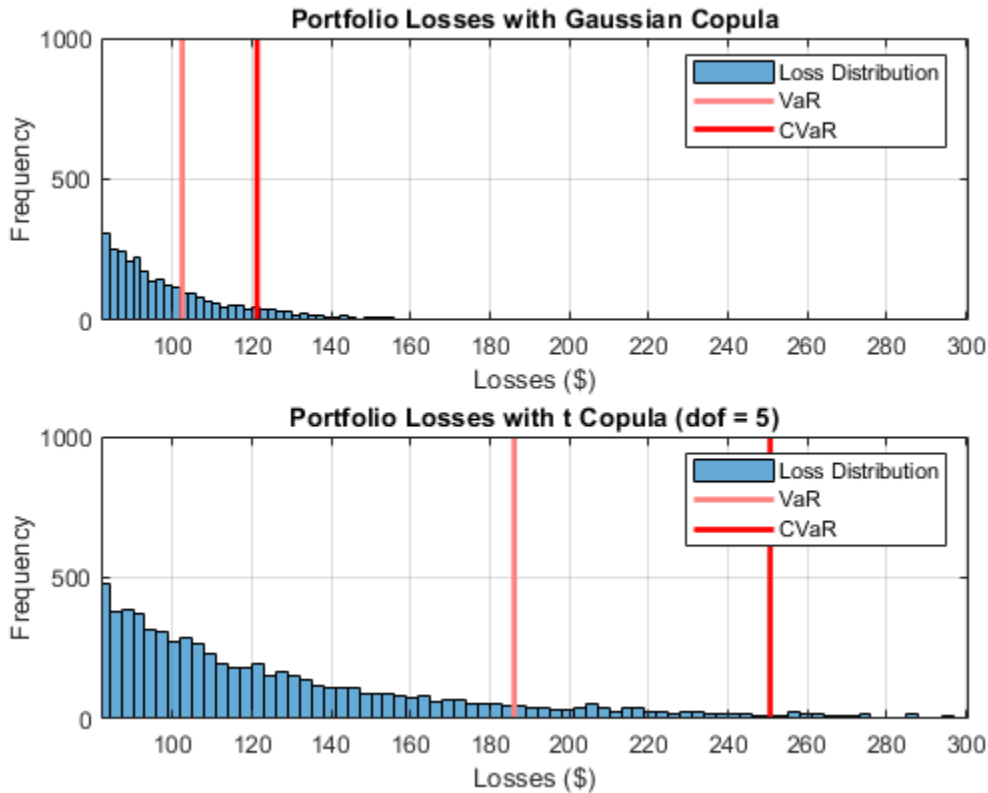
Compare the tail losses of each model.

```

% 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 4-5
- “One-Factor Model Calibration”

## More About

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

## 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=1x8 table
      ID          CR          Deciles          Gini          HH          HK          HT
-----
"Portfolio"  0.058745  [1x11 double]  0.55751  0.023919  0.013363  0.022599  0
```

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          HT
-----
"Multiplier 0.0"  0.059442  [1x11 double]  0.55051  0.023102  0.013314  0.02224
"Multiplier 0.05"  0.059257  [1x11 double]  0.5467  0.022968  0.013185  0.02206
"Multiplier 0.1"  0.059074  [1x11 double]  0.54456  0.022855  0.013156  0.02195
"Multiplier 0.15"  0.058891  [1x11 double]  0.54355  0.022762  0.013143  0.02190
"Multiplier 0.2"  0.058709  [1x11 double]  0.54313  0.022688  0.013139  0.02188
```

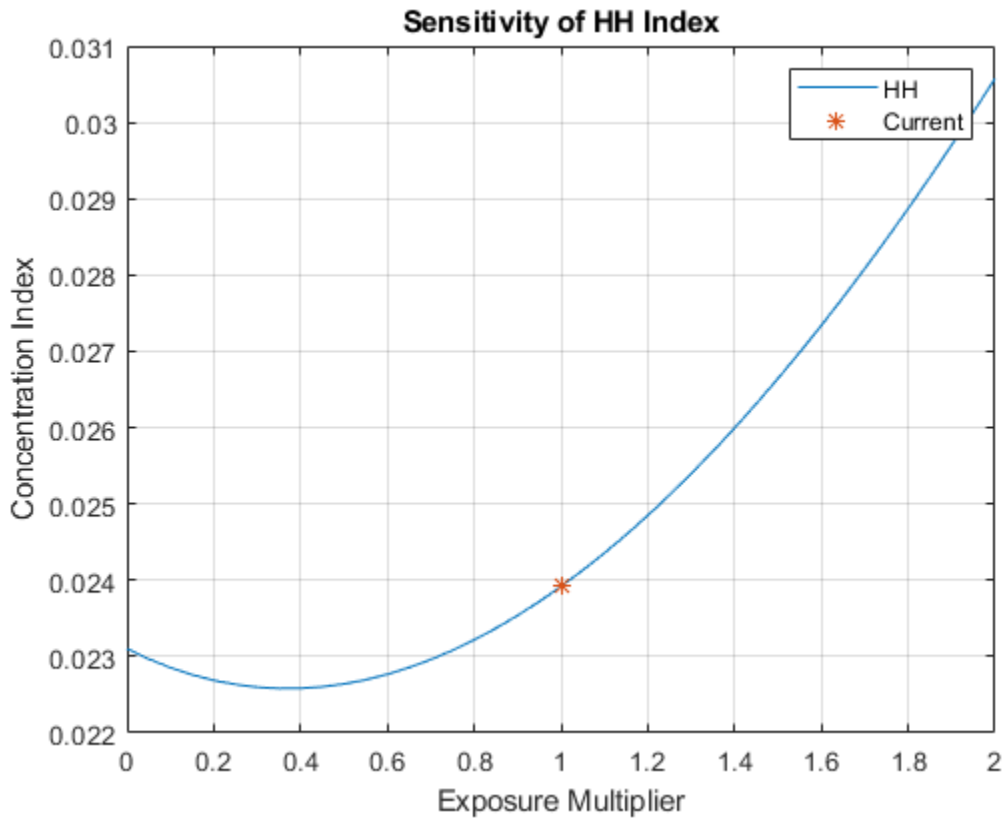
Plot the sensitivity to changes in exposure for a particular index.

```
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 4-30

## More About

- “Concentration Indices” on page 1-12

## 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));
```

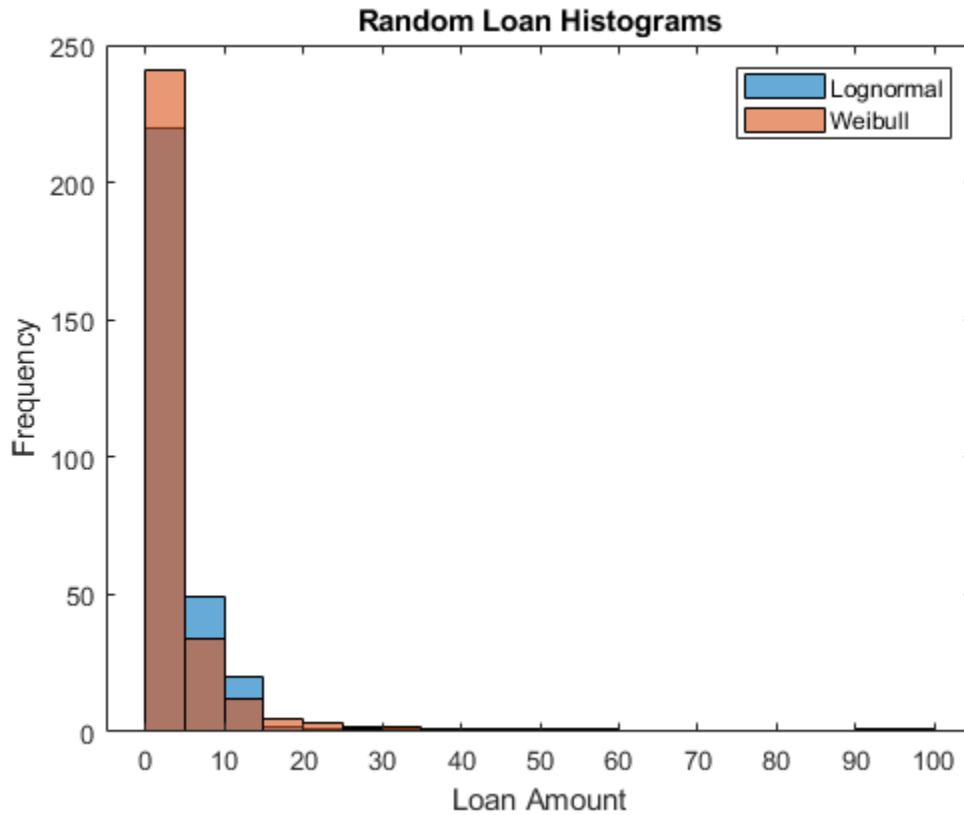
```
Largest loan Lognormal: 97.3582
```

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

```
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')
```

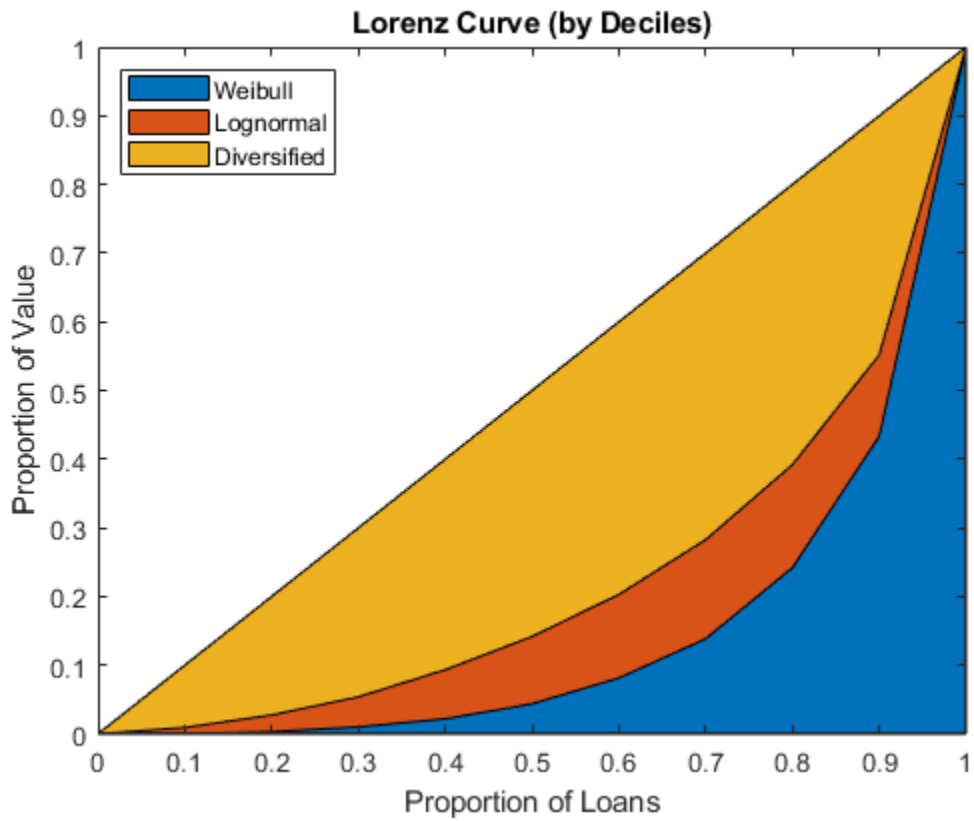


Compute and display the concentration measures.

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

ID	CR	Deciles	Gini	HH	HK	HT
"Lognormal"	0.066363	[1x11 double]	0.5686	0.013298	0.0045765	0.0077267
"Weibull"	0.090152	[1x11 double]	0.72876	0.020197	0.0062594	0.012289

```
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')
```



**See Also**

concentrationIndices

**Related Examples**

- "Analyze the Sensitivity of Concentration to a Given Exposure" on page 4-28

**More About**

- "Concentration Indices" on page 1-12



## 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)';

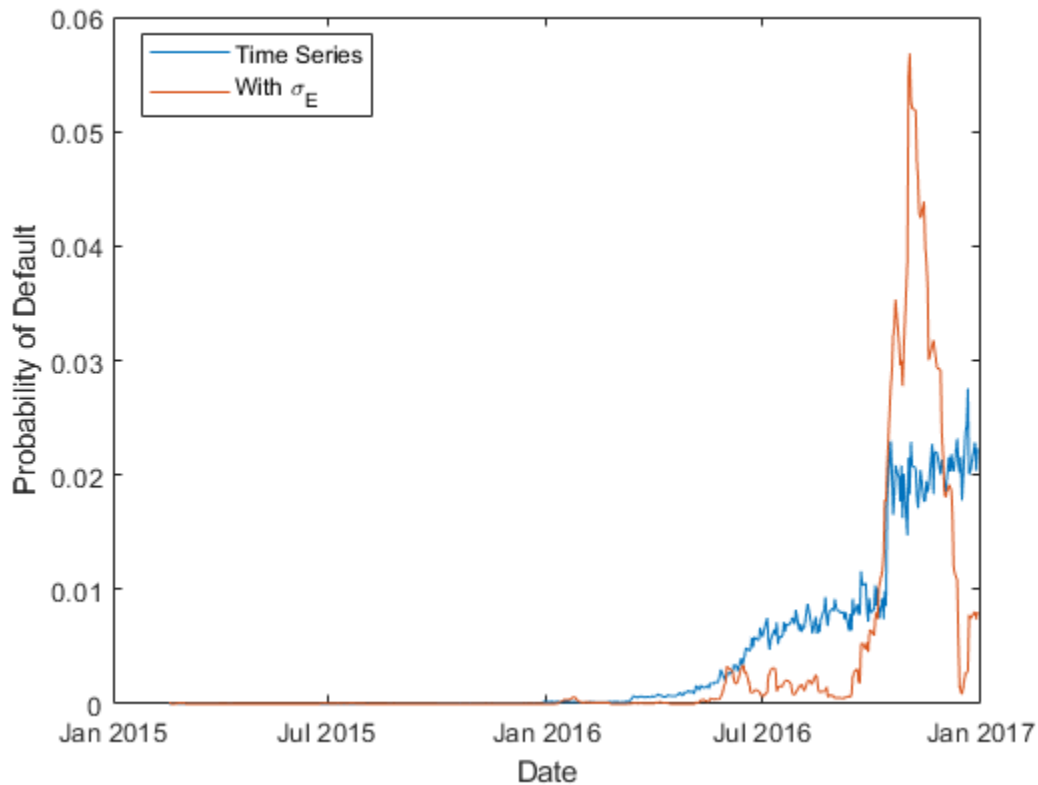
EquityVol = zeros(length(TestWindow),1);

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(TestWindow));
[PDh,DDh,Ah,Sah] = mertonmodel(Equity(TestWindow),EquityVol,Liability(TestWindow),Rate(TestWindow));

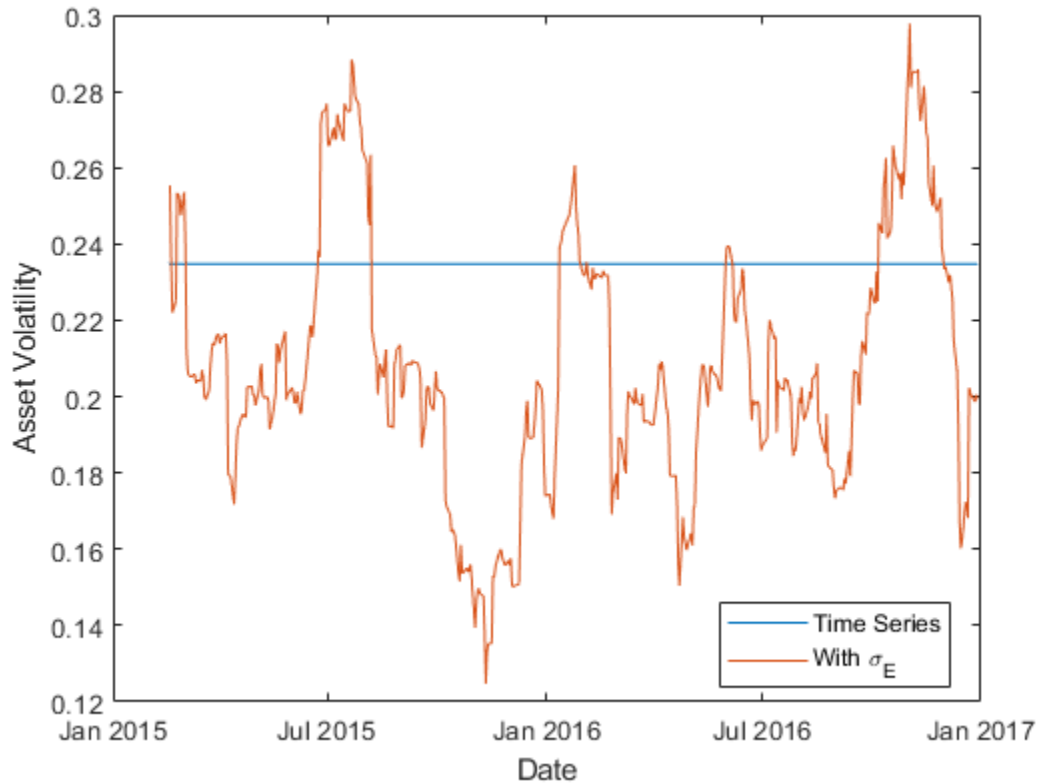
figure
plot(Dates(TestWindow),PDTS,Dates(TestWindow),PDh)
xlabel('Date')
ylabel('Probability of Default')
legend({'Time Series','With \sigma_E'},'Location','best')
```



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')
```

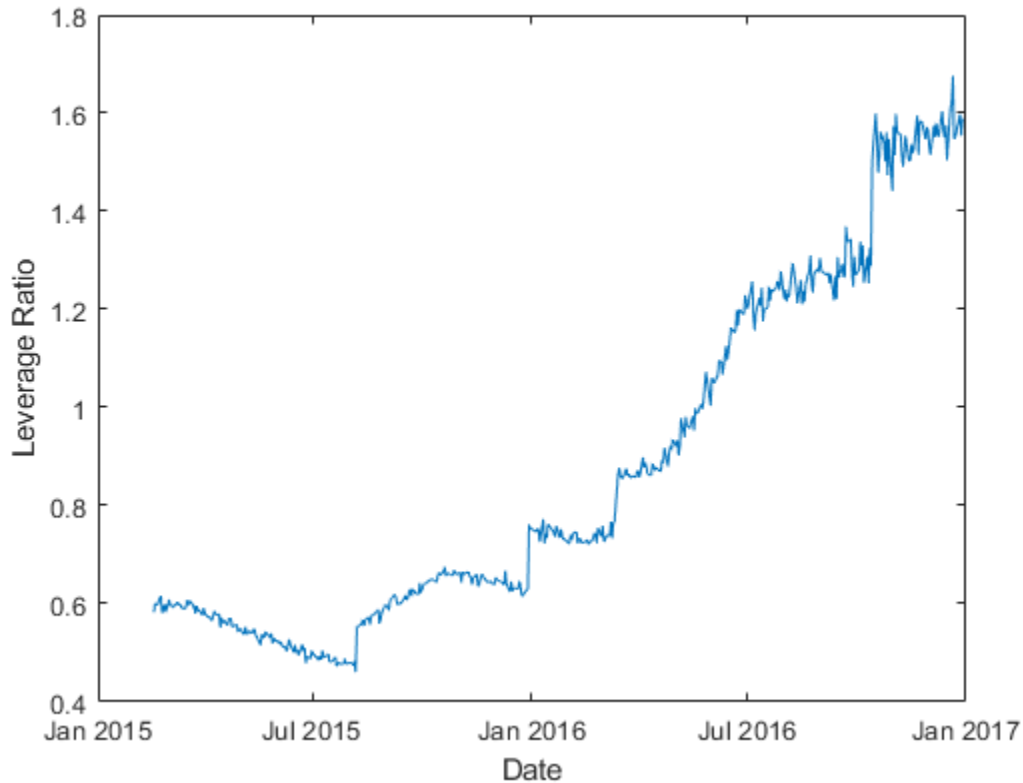


Towards the end of the time window, the single-point probability of default is above the time-series probability of default when the single-point asset volatility is also above the time-series probability (and vice versa). However, before 2016 the volatility has no effect on the default probability. 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);
```

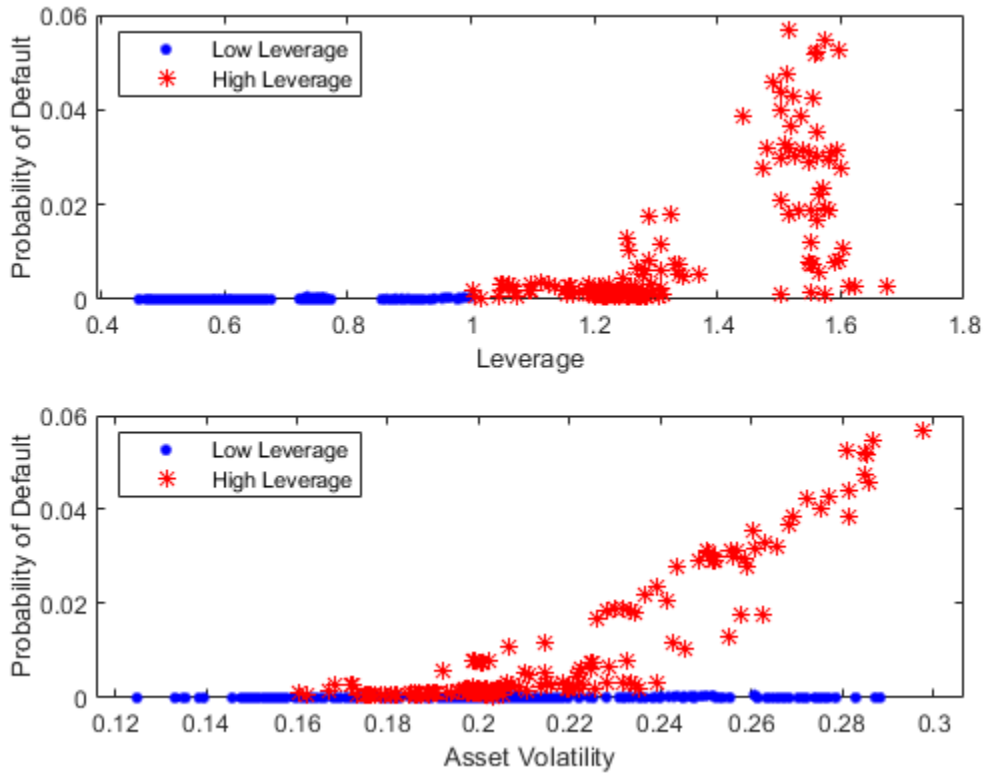
```
figure
plot(Dates(TestWindow),Leverage)
xlabel('Date')
ylabel('Leverage Ratio')
```



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 1-10

## Calculating Regulatory Capital with the ASRF Model

This example shows how to calculate capital requirements and value-at-risk (VaR) for a credit sensitive portfolio of exposures using the asymptotic single risk factor (ASRF) model. This example also shows how to compute Basel capital requirements using an ASRF model.

### The ASRF Model

The ASRF model defines capital as the credit value at risk (VaR) in excess of the expected loss (EL).

$$capital = VaR - EL$$

where the EL for a given counterparty is the exposure at default (EAD) multiplied by the probability of default (PD) and the loss given default (LGD).

$$EL = EAD \cdot PD \cdot LGD$$

To compute the credit VaR, the ASRF model assumes that obligor credit quality is modeled with a latent variable ( $A$ ) using a one factor model where the single common factor ( $Z$ ) represents systemic credit risk in the market.

$$A_i = \sqrt{\rho_i} \cdot Z + \sqrt{1 - \rho_i} \cdot \epsilon$$

Under this model, default losses for a particular scenario are calculated as:

$$L = EAD \cdot I \cdot LGD$$

where  $I$  is the default indicator, and has a value of 1 if  $A_i < \Phi_A^{-1}(PD_i)$  (meaning the latent variable has fallen below the threshold for default), and a value of 0 otherwise. The expected value of the default indicator conditional on the common factor is given by:

$$E(I_i | Z) = \Phi_e\left(\frac{\Phi_A^{-1}(PD_i) - \sqrt{\rho_i}Z}{\sqrt{1 - \rho_i}}\right)$$

For well diversified and perfectly granular portfolios, the expected loss conditional on a value of the common factor is:

$$L | Z = \sum_i EAD_i \cdot LGD_i \cdot \Phi_e\left(\frac{\Phi_A^{-1}(PD_i) - \sqrt{\rho_i}Z}{\sqrt{1 - \rho_i}}\right)$$

You can then directly compute particular percentiles of the distribution of losses using the cumulative distribution function of the common factor. This is the credit VaR, which we compute at the  $\alpha$  confidence level:

$$creditVaR(\alpha) = \sum_i EAD_i \cdot LGD_i \cdot \Phi_e\left(\frac{\Phi_A^{-1}(PD_i) - \sqrt{\rho_i}\Phi_Z^{-1}(1 - \alpha)}{\sqrt{1 - \rho_i}}\right)$$

It follows that the capital for a given level of confidence,  $\alpha$ , is:

$$capital(\alpha) = \sum_i EAD_i \cdot LGD_i \cdot \left[\Phi_e\left(\frac{\Phi_A^{-1}(PD_i) - \sqrt{\rho_i}\Phi_Z^{-1}(1 - \alpha)}{\sqrt{1 - \rho_i}}\right) - PD_i\right]$$

## Basic ASRF

The portfolio contains 100 credit sensitive contracts and information about their exposure. This is simulated data.

```
load asrfPortfolio.mat
```

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

ID	EAD	PD	LGD	AssetClass	Sales	Maturity
1	2.945e+05	0.013644	0.5	"Bank"	NaN	02-Jun-2023
2	1.3349e+05	0.0017519	0.5	"Bank"	NaN	05-Jul-2021
3	3.1723e+05	0.01694	0.4	"Bank"	NaN	07-Oct-2018
4	2.8719e+05	0.013624	0.35	"Bank"	NaN	27-Apr-2022
5	2.9965e+05	0.013191	0.45	"Bank"	NaN	07-Dec-2022

The asset correlations ( $\rho$ ) in the ASRF model define the correlation between similar assets. The square root of this value,  $\sqrt{\rho}$ , specifies the correlation between a counterparty's latent variable (A) and the systemic credit factor (Z). Asset correlations can be calibrated by observing correlations in the market or from historical default data. Correlations can also be set using regulatory guidelines (see Basel Capital Requirements section).

Because the ASRF model is a fast, analytical formula, it is convenient to perform sensitivity analysis for a counterparty by varying the exposure parameters and observing how the capital and VaR change.

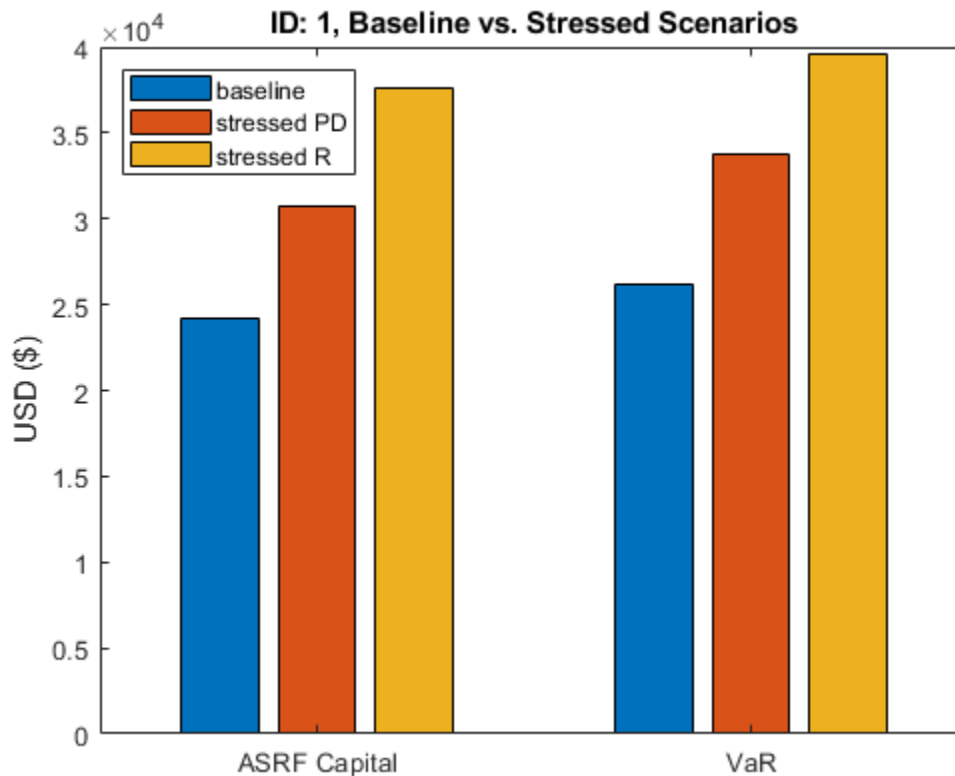
The following plot shows the sensitivity to PD and asset correlation. The LGD and EAD parameters are scaling factors in the ASRF formula so the sensitivity is straightforward.

```
% Counterparty ID
id = 1;

% Set default asset correlation to 0.2 as baseline
R = 0.2;

% Compute baseline capital and VaR
[capital0, var0] = asrf(portfolio.PD(id),portfolio.LGD(id),R,'EAD',portfolio.EAD(id));
% Stressed PD by 50%
[capital1, var1] = asrf(portfolio.PD(id) * 1.5,portfolio.LGD(id),R,'EAD',portfolio.EAD(id));
% Stressed Correlation by 50%
[capital2, var2] = asrf(portfolio.PD(id),portfolio.LGD(id),R * 1.5,'EAD',portfolio.EAD(id));

c = categorical({'ASRF Capital','VaR'});
bar(c,[capital0 capital1 capital2; var0 var1 var2]);
legend({'baseline','stressed PD','stressed R'},'Location','northwest')
title(sprintf('ID: %d, Baseline vs. Stressed Scenarios',id));
ylabel('USD ($)');
```



### Basel Capital Requirements

When computing regulatory capital, the Basel documents have additional model specifications on top of the basic ASRF model. In particular, Basel II/III defines specific formulas for computing the asset correlation for exposures in various asset classes as a function of the default probability.

To set up the vector of correlations according to the definitions established in Basel II/III:

```
R = zeros(height(portfolio),1);

% Compute correlations for corporate, sovereign, and bank exposures
idx = portfolio.AssetClass == "Corporate" |...
     portfolio.AssetClass == "Sovereign" |...
     portfolio.AssetClass == "Bank";

R(idx) = 0.12 * (1-exp(-50*portfolio.PD(idx))) / (1-exp(-50)) +...
        0.24 * (1 - (1-exp(-50*portfolio.PD(idx))) / (1-exp(-50)));

% Compute correlations for small and medium entities
idx = portfolio.AssetClass == "Small Entity" |...
     portfolio.AssetClass == "Medium Entity";

R(idx) = 0.12 * (1-exp(-50*portfolio.PD(idx))) / (1-exp(-50)) +...
        0.24 * (1 - (1-exp(-50*portfolio.PD(idx))) / (1-exp(-50))) -...
        0.04 * (1 - (portfolio.Sales(idx)/1e6 - 5) / 45);

% Compute correlations for unregulated financial institutions
```



```
idx = portfolio.AssetClass == "Unregulated Financial";
```

```
R(idx) = 1.25 * 0.12 * (1-exp(-50*portfolio.PD(idx))) / (1-exp(-50)) + ...
    0.24 * (1 - (1-exp(-50*portfolio.PD(idx))) / (1-exp(-50)));
```

Find the basic ASRF capital using the Basel-defined asset correlations. The default value for the VaR level is 99.9%.

```
asrfCapital = asrf(portfolio.PD,portfolio.LGD,R,'EAD',portfolio.EAD);
```

Additionally, the Basel documents specify a maturity adjustment to be added to each capital calculation. Here we compute the maturity adjustment and update the capital requirements.

```
maturityYears = years(portfolio.Maturity - settle);
```

```
b = (0.11852 - 0.05478 * log(portfolio.PD)).^2;
maturityAdj = (1 + (maturityYears - 2.5) .* b) ./ (1 - 1.5 .* b);
```

```
regulatoryCapital = asrfCapital .* maturityAdj;
```

```
fprintf('Portfolio Regulatory Capital : $%.2f\n',sum(regulatoryCapital));
```

```
Portfolio Regulatory Capital : $2310819.05
```

Risk weighted assets (RWA) are calculated as capital \* 12.5.

```
RWA = regulatoryCapital * 12.5;
```

```
results = table(portfolio.ID,portfolio.AssetClass,RWA,regulatoryCapital,'VariableNames',...
    {'ID','AssetClass','RWA','Capital'});
```

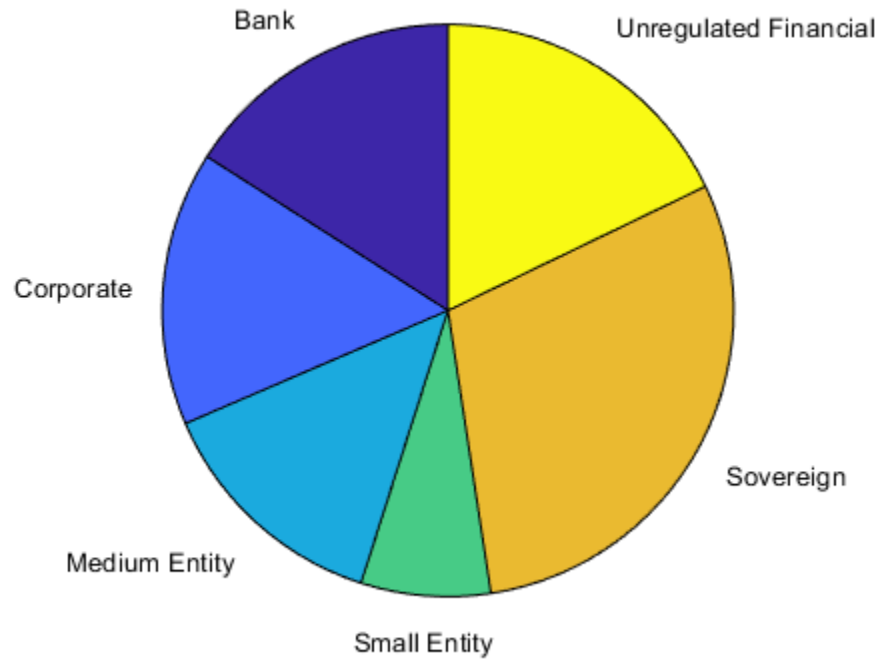
```
% Results table
disp(results(1:5,:))
```

ID	AssetClass	RWA	Capital
1	"Bank"	4.7766e+05	38213
2	"Bank"	79985	6398.8
3	"Bank"	2.6313e+05	21050
4	"Bank"	2.9449e+05	23560
5	"Bank"	4.1544e+05	33235

Aggregate the regulatory capital by asset class.

```
assetClasses = unique(results.AssetClass);
assetClassCapital = zeros(numel(assetClasses),1);
for i = 1:numel(assetClasses)
    assetClassCapital(i) = sum(results.Capital(results.AssetClass == assetClasses(i)));
end
pie(assetClassCapital,cellstr(assetClasses))
title('Regulatory Capital by Asset Class');
```

**Regulatory Capital by Asset Class**



```
capitalTable = table(assetClasses, assetClassCapital, 'VariableNames', {'AssetClass', 'Capital'});
disp(capitalTable);
```

AssetClass	Capital
"Bank"	3.6894e+05
"Corporate"	3.5811e+05
"Medium Entity"	3.1466e+05
"Small Entity"	1.693e+05
"Sovereign"	6.8711e+05
"Unregulated Financial"	4.127e+05

**See Also**

asrf

# Functions

---

## 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. Use `screenpredictors` to pare down a potentially large set of predictors to a subset that is most predictive of the credit score card response variable. You can then use this subset of predictors when creating a MATLAB table of data. 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 with an option to bin missing data. (For more information on algorithms for automatic binning, see `autobinning`.)
- 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` to open the **Binning Explorer** app.
  - Enter `binningExplorer(data)` or `binningExplorer(data,Name,Value)` to open a table in the **Binning Explorer** app by specifying a table (`data`) as input.
  - Enter `binningExplorer(sc)` to open a `creditscorecard` object in the **Binning Explorer** app by specifying a `creditscorecard` object (`sc`) as input.

To access Help for the App, click the Help icon on the toolbar.

---

**Note** When using the **Binning Explorer** app with MATLAB Online:

- The App toolbar is not available for MATLAB Online. To access Help, from the MATLAB command prompt, enter `doc binningExplorer`.
  - MATLAB Online does not display predictor information using three panels (**Overview**, **Bin Information**, and **Predictor Information**) in the Binning Explorer window. Instead, MATLAB Online displays these panels as tabs labeled **Overview**, **Bin Information**, and **Predictor Information**.
  - When performing manual binning, selected predictors are displayed in a tab in the Binning Explorer window. When you close the tab for a predictor, you do not return to the **Overview** panel. To return to the **Overview** panel, click the **Overview** tab.
-

## Examples

- “Overview of Binning Explorer” on page 3-2
- “Feature Screening with screenpredictors”
- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-21
- “Case Study for a Credit Scorecard Analysis” (Financial Toolbox)
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-34

## See Also

### Functions

autobinning | creditscorecard | screenpredictors

### Topics

“Overview of Binning Explorer” on page 3-2

“Feature Screening with screenpredictors”

“Common Binning Explorer Tasks” on page 3-4

“Binning Explorer Case Study Example” on page 3-21

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

“Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-34

“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 17 sec)

### Introduced in R2016b

## asrf

Asymptotic Single Risk Factor (ASRF) capital

### Syntax

```
[capital,VaR] = asrf(PD,LGD,R)
[capital,VaR] = asrf( ___,Name,Value)
```

### Description

[capital,VaR] = asrf(PD,LGD,R) computes regulatory capital and value-at-risk using an ASRF model.

[capital,VaR] = asrf( \_\_\_,Name,Value) adds optional name-value pair arguments.

### Examples

#### Compute Necessary Capital Using an ASRF Model

Load saved portfolio data.

```
load CreditPortfolioData.mat
```

Compute asset correlation for corporate, sovereign, and bank exposures.

```
R = 0.12 * (1-exp(-50*PD)) / (1-exp(-50)) + ...
    0.24 * (1 - (1-exp(-50*PD)) / (1-exp(-50)));
```

Compute the asymptotic single risk factor capital. By specifying the name-value pair argument for EAD, the capital is returned in terms of currency.

```
capital = asrf(PD,LGD,R, 'EAD',EAD);
```

Apply a maturity adjustment.

```
b = (0.11852 - 0.05478 * log(PD)).^2;
matAdj = (1 + (Maturity - 2.5) .* b) ./ (1 - 1.5 * b);
adjustedCapital = capital .* matAdj;
```

```
portfolioCapital = sum(adjustedCapital)
```

```
portfolioCapital = 175.7865
```

### Input Arguments

#### PD — Probability of default

numeric vector with elements from 0 to 1

Probability of default, specified as a NumCounterparties-by-1 numeric vector with elements from 0 to 1, representing the default probabilities for the counterparties.

Data Types: double

### **LGD — Loss given default**

numeric vector with elements from 0 to 1

Loss given default, specified as a NumCounterparties-by-1 numeric vector with elements from 0 to 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

### **R — Asset correlation**

numeric vector

Asset correlation, specified as a NumCounterparties-by-1 numeric vector.

The asset correlations, R, have values from 0 to 1 and specify the correlation between assets in the same asset class.

---

**Note** The correlation between an asset value and the underlying single risk factor is  $\sqrt{R}$ . This value,  $\sqrt{R}$ , corresponds to the `Weights` input argument to the `creditDefaultCopula` and `creditMigrationCopula` classes for one-factor models.

---

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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: `capital = asrf(PD,LGD,R,'EAD',EAD)`

### **EAD — Exposure at default**

1 (default) | numeric vector

Exposure at default, specified as the comma-separated pair consisting of 'EAD' and a NumCounterparties-by-1 numeric vector of credit exposures.

If EAD is not specified, the default EAD is 1, meaning that `capital` and `VaR` results are reported as a percentage of the counterparty's exposure. If EAD is specified, then `capital` and `VaR` are returned in units of currency.

Data Types: double

### **VaRLevel — Value at risk level**

0.999 (99.9%) (default) | decimal value between 0 and 1

Value at risk level used when calculating the capital requirement, specified as the comma-separated pair consisting of 'VaRLevel' and a decimal value between 0 and 1.

Data Types: double

## Output Arguments

### capital — Capital for each element in portfolio

vector

Capital for each element in the portfolio, returned as a NumCounterparties-by-1 vector. If the optional input EAD is specified, then capital is in units of currency. Otherwise, capital is reported as a percentage of each exposure.

### VaR — Value-at-risk for each exposure

vector

Value-at-risk for each exposure, returned as a NumCounterparties-by-1 vector. If the optional input EAD is specified, then VaR is in units of currency. Otherwise, VaR is reported as a percentage of each exposure.

## More About

### ASRF Model Capital

In the ASRF model, capital is defined as the loss in excess of the expected loss (EL) at a high confidence level.

The formula for capital is

$$\text{capital} = \text{VaR} - \text{EL}$$

## Algorithms

The capital requirement formula for exposures is defined as

$$\text{VaR} = \text{EAD} * \text{LGD} * \Phi \left( \frac{\Phi^{-1}(\text{PD}) - \sqrt{R} \Phi^{-1}(1 - \text{VaRLevel})}{\sqrt{1 - R}} \right)$$

$$\text{capital} = \text{VaR} - \text{EAD} * \text{LGD} * \text{PD}$$

where

$\Phi$  is the normal CDF.

$\Phi^{-1}$  is the inverse normal CDF.

R is asset correlation.

EAD is exposure at default.

PD is probability of default.

LGD is loss given default.

## References

[1] Gordy, M.B. "A risk-factor model foundation for ratings-based bank capital rule." *Journal of Financial Intermediation*. Vol. 12, pp. 199-232, 2003.



## **See Also**

creditDefaultCopula | creditMigrationCopula

## **Topics**

“Calculating Regulatory Capital with the ASRF Model”

**Introduced in R2017b**

## 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=1x8 table
      ID          CR          Deciles          Gini          HH          HK          HT
-----
"Portfolio"  0.058745  [1x11 double]  0.55751  0.023919  0.013363  0.022599  0
```

### 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=1x8 table
      ID              CR              Deciles      Gini      HH      HK      TE
-----
"Portfolio"  0.38942  0.58836  [1x11 double]  0.55751  0.023919  0.013363  0.029344
```

### 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=1x8 table
      ID              CR              Deciles      Gini      HH      HK      TE
-----
"Portfolio"  0.058745  [1x11 double]  0.55751  0.023919  0.013363  0.029344  0.029344
```

### 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  -2.2204e-16
"Fully concentrated"  1  [1x11 double]  0.8  1  1  1  1.6094
```

### 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	HT	TE
"Diversified, unscaled"	0.2	[1x11 double]	0	0.2	0.2	0.2	0.2
"Diversified, scaled"	0.2	[1x11 double]	0	3.4694e-17	-3.4694e-17	-6.9388e-17	-6.9388e-17
"Concentrated, unscaled"	1	[1x11 double]	0.8	1	1	1	1
"Concentrated, scaled"	1	[1x11 double]	1	1	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.

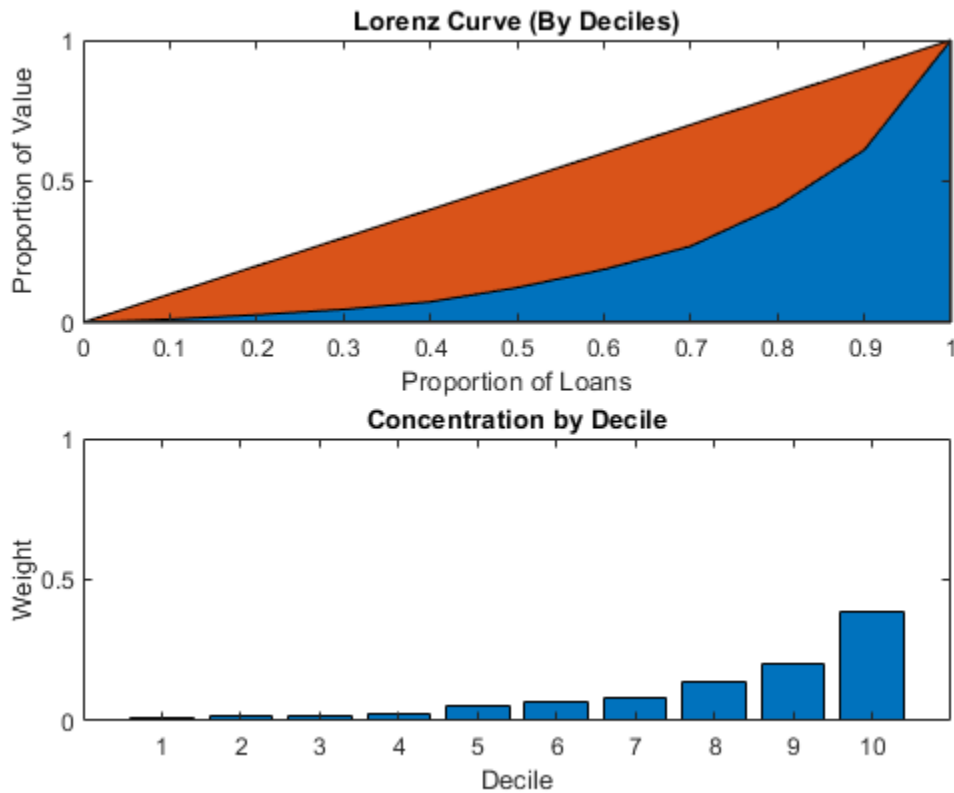
```
load CreditPortfolioData.mat
P = EAD;
ci = concentrationIndices(P);
```

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')
```

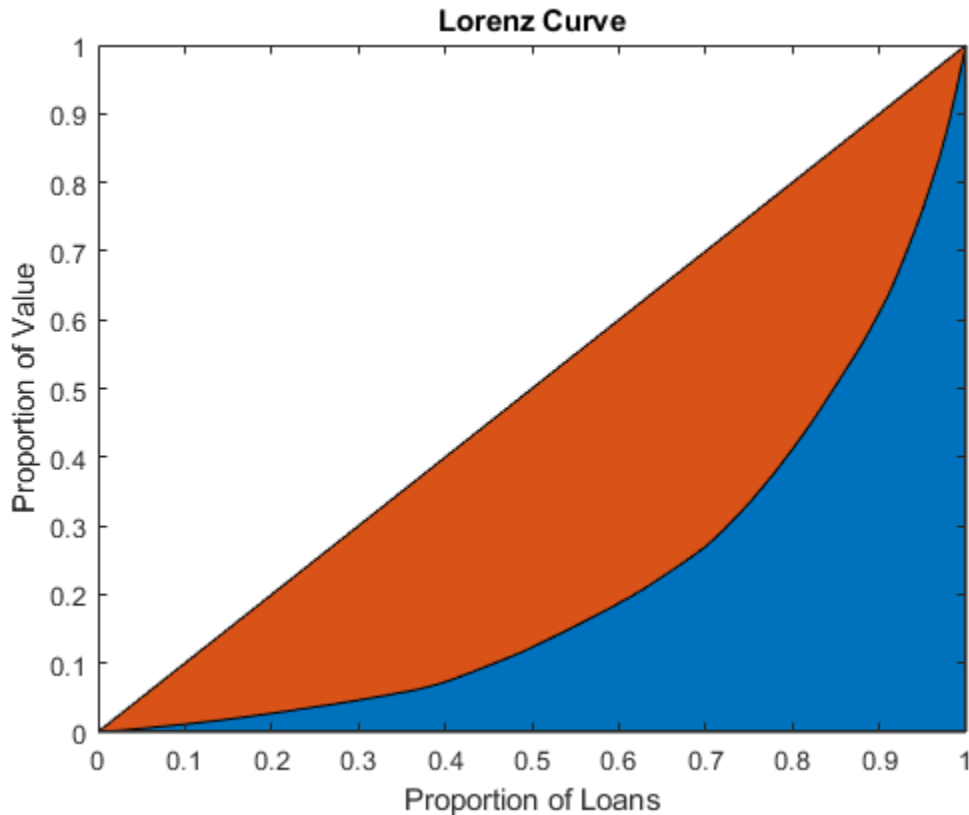


### Plot an Exact Lorenz Curve Using the Optional Lorenz Output

Load the `CreditPortfolioData.mat` file that contains the 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')
```



## 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 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

**HKAlpha — 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 'HKAlpha', and a positive number that cannot be equal to 1. If HKAlpha 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 “More About” on page 5-14.
- **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 “More About” on page 5-14.
- **Gini** — Gini coefficient. For more information, see “More About” on page 5-14.
- **HH** — Herfindahl-Hirschman index. For more information, see “More About” on page 5-14.
- **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 “More About” on page 5-14.

- HT — Hall-Tideman index. For more information, see “More About” on page 5-14.
- TE — Theil entropy index. For more information, see “More About” on page 5-14.

### 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.

## More About

### 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  $x_i > 0$  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 non-decreasing 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.

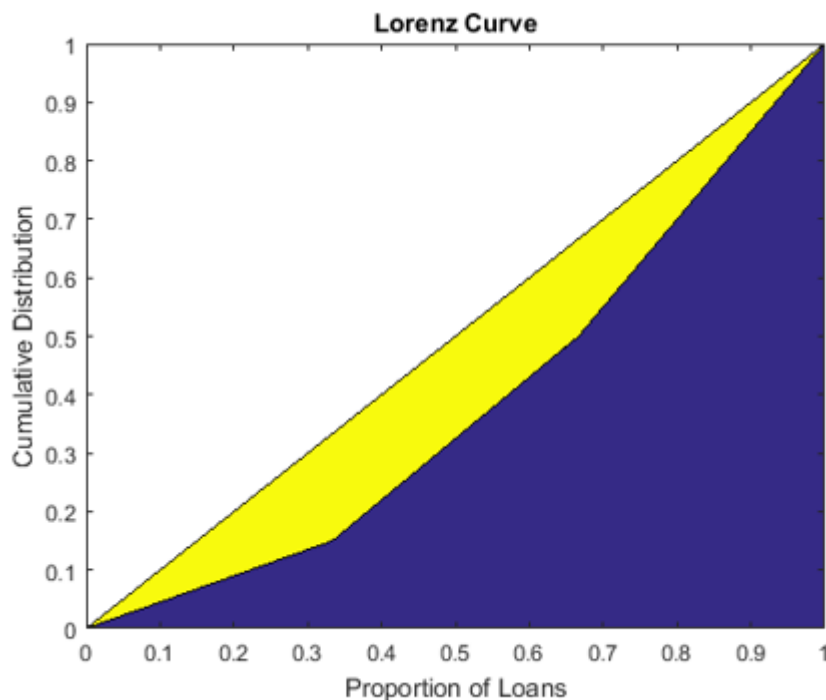
The cumulative proportion of loans ( $p$ ) is defined by:

$$p_0 = 0, p_1 = \frac{1}{N}, p_2 = \frac{2}{N}, \dots, p_N = \frac{N}{N} = 1$$

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 5-15, the deciles are a subset of the proportion of value array. Given indices  $d_1, d_2, \dots, d_9$  such that the proportion of loans matches exactly these values:

$$p_{d_1} = 0.1, p_{d_2} = 0.2, \dots, p_{d_9} = 0.9$$

The deciles  $D_0, D_1, \dots, D_9, D_{10}$  are defined as the corresponding proportion of values:

$$D_0 = L_0 = 0, D_1 = L_{d_1}, D_2 = L_{d_2}, \dots, D_9 = L_{d_9}, D_{10} = L_N = 1$$

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

- [1] Basel Committee on Banking Supervision. "*Studies on Credit Risk Concentration*". Working paper no. 15. November, 2006.
- [2] 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.
- [3] Lütkebohmert, E. *Concentration Risk in Credit Portfolios*. Springer, 2009.

## See Also

### Topics

- "Analyze the Sensitivity of Concentration to a Given Exposure" on page 4-28
- "Compare Concentration Indices for Random Portfolios" on page 4-30
- "Concentration Indices" on page 1-12

### Introduced in R2017a

## creditDefaultCopula

Create `creditDefaultCopula` object to 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 (see “Create `creditDefaultCopula`” on page 5-18 and “Properties” on page 5-21), 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, and 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)

### Creation

#### Syntax

```
cdc = creditDefaultCopula(EAD, PD, LGD, Weights)
cdc = creditDefaultCopula( ____, Name, Value)
```

#### Description

`cdc = creditDefaultCopula(EAD, PD, LGD, Weights)` creates a `creditDefaultCopula` object. The `creditDefaultCopula` object has the following properties:

- Portfolio on page 5-0 :

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 on page 5-0 :

Factor correlation matrix, a NumFactors-by-NumFactors matrix that defines the correlation between the risk factors.

- VaRLevel on page 5-0 :

The value-at-risk level, used when reporting VaR and CVaR.

- PortfolioLosses on page 5-0

Portfolio losses, a NumScenarios-by-1 vector of portfolio losses. This property is empty until the simulate function is used.

`cdc = creditDefaultCopula( ____, Name, Value)` sets Properties on page 5-21 using name-value pairs and any of the arguments in the previous syntax. For example, `cdc = creditDefaultCopula(EAD, PD, LGD, Weights, 'VaRLevel', 0.99)`. You can specify multiple name-value pairs as optional name-value pair arguments.

## Input Arguments

### EAD — Exposure at default

numeric vector

Exposure at default, specified as a NumCounterparties-by-1 vector of credit exposures. The EAD input sets the Portfolio on page 5-0 property.

---

**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. The PD input sets the Portfolio on page 5-0 property.

---

**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. The LGD input sets the `Portfolio` on page 5-0 property.

LGD can alternatively be specified as a `NumCounterparties-by-2` matrix, where the first column holds the LGD mean values and the 2nd column holds the LGD standard deviations. Valid open intervals for LGD mean and standard deviation are:

- For the first column, the mean values are between 0 and 1.
- For the second column, the LGD standard deviations are between 0 and  $\sqrt{m*(1-m)}$ .

Then, in the case of default, LGD values are drawn randomly from a beta distribution with provided parameters for the defaulting counterparty.

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. The `Weights` input sets the `Portfolio` on page 5-0 property.

For example, if a counterparty's creditworthiness is 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 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. The `ID` name-value pair argument sets the `Portfolio` on page 5-0 property.

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. The VaRLevel name-value pair argument sets the VaRLevel on page 5-0 property.

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. The FactorCorrelation name-value pair argument sets the FactorCorrelation on page 5-0 property.

If not specified, the factor correlation matrix defaults to an identity matrix, meaning that factors are not correlated.

Data Types: double

**UseParallel – Flag to use parallel processing for simulations**

false (default) | logical with value of true or false

Flag to use parallel processing for simulations, specified as the comma-separated pair consisting of 'UseParallel' and a scalar value of true or false. The UseParallel name-value pair argument sets the UseParallel on page 5-0 property.

---

**Note** The 'UseParallel' property can only be set when creating a creditDefaultCopula object if you have Parallel Computing Toolbox™. Once the 'UseParallel' property is set, parallel processing is used with riskContribution or simulate.

---

Data Types: logical

**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 creditDefaultCopula.

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' when you create a creditDefaultCopula object.

Data Types: double

### VaRLevel — 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' when you create a creditDefaultCopula object.

Data Types: double

### PortfolioLosses — Total portfolio losses

vector

Total portfolio losses, specified as a 1-by-NumScenarios vector. The PortfolioLosses property is empty after you create a creditDefaultCopula object. After the simulate function is invoked, the PortfolioLosses property is populated with the vector of portfolio losses.

Data Types: double

### UseParallel — Flag to use parallel processing for simulations

false (default) | logical with value of true or false

Flag to use parallel processing for simulations, specified using an optional name-value pair argument 'UseParallel' when you create a creditDefaultCopula object. The UseParallel name-value pair argument sets the UseParallel property.

---

**Note** The 'UseParallel' property can only be set when creating a creditDefaultCopula object if you have Parallel Computing Toolbox. Once the 'UseParallel' property is set, parallel processing is used with riskContribution or simulate.

---

Data Types: logical

## Object Functions

simulate	Simulate credit defaults using a creditDefaultCopula object
portfolioRisk	Generate portfolio-level risk measurements
riskContribution	Generate risk contributions for each counterparty in portfolio
confidenceBands	Confidence interval bands
getScenarios	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: [100x5 table]  
FactorCorrelation: [2x2 double]  
VaRLevel: 0.9500  
UseParallel: 0  
PortfolioLosses: []
```

Set the VaRLevel to 99%.

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

Simulate 100,000 scenarios, and view the portfolio risk measures.

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

```
cdc =  
creditDefaultCopula with properties:
```

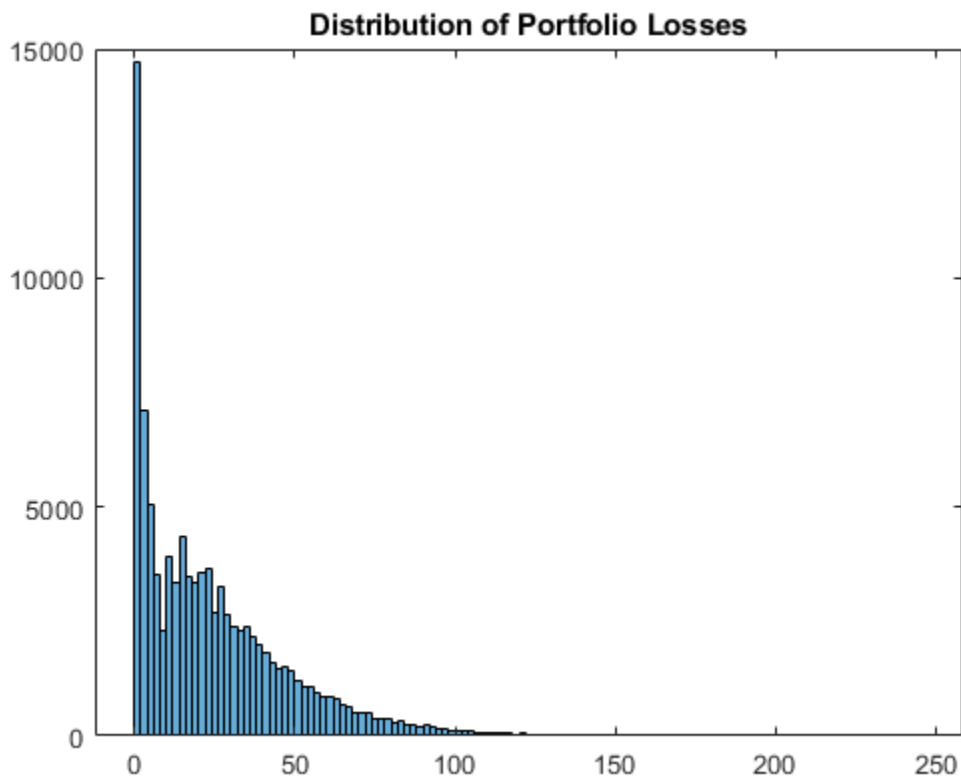
```
Portfolio: [100x5 table]  
FactorCorrelation: [2x2 double]  
VaRLevel: 0.9900  
UseParallel: 0  
PortfolioLosses: [1x100000 double]
```

```
portRisk = portfolioRisk(cdc)
```

```
portRisk=1x4 table  
EL      Std      VaR      CVaR  
-----  
24.876  23.778  102.4    121.28
```

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.

## References

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. "*CreditMetrics - Technical Document*." J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

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

**Topics**

"creditDefaultCopula Simulation Workflow"

"creditDefaultCopula Simulation Workflow" on page 4-5

"Modeling Correlated Defaults with Copulas" on page 4-18

"One-Factor Model Calibration"

"Corporate Credit Risk" on page 1-3

"Credit Simulation Using Copulas" on page 4-2

**External Websites**

Parallel Computing with MATLAB (53 min 27 sec)

**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: [100x5 table]
  FactorCorrelation: [2x2 double]
        VaRLevel: 0.9500
    UseParallel: 0
  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=10x4 table

NumScenarios	Lower	Std	Upper
1000	23.38	24.237	25.166
2000	23.255	23.859	24.497
3000	23.617	24.117	24.642
4000	23.44	23.871	24.319
5000	23.504	23.891	24.291
6000	23.582	23.935	24.301
7000	23.756	24.086	24.426
8000	23.587	23.893	24.208
9000	23.582	23.871	24.167
10000	23.525	23.799	24.079

## 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 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, 'NumPoints', 50)
```

### 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**

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. "CreditMetrics - Technical Document." J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

**See Also**

`creditDefaultCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

**Topics**

“Credit Simulation Using Copulas” on page 4-2

“creditDefaultCopula Simulation Workflow” on page 4-5

“Modeling Correlated Defaults with Copulas” on page 4-18

“One-Factor Model Calibration”

“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: [100x5 table]  
FactorCorrelation: [2x2 double]  
    VaRLevel: 0.9500  
    UseParallel: 0  
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 = 1x2
```



0.0369    0.0329

## 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

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. "*CreditMetrics - Technical Document*." J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

confidenceBands | creditDefaultCopula | portfolioRisk | riskContribution | simulate

## Topics

"Credit Simulation Using Copulas" on page 4-2

“Modeling Correlated Defaults with Copulas” on page 4-18

“One-Factor Model Calibration”

“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: [100x5 table]
    FactorCorrelation: [2x2 double]
    VaRLevel: 0.9500
    UseParallel: 0
    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
24.876	23.778	102.4	121.28

confidenceIntervals=1×4 table

EL	Std	VaR		CVaR			
24.752	25	23.691	23.866	101.35	103.35	120.32	122.24

## 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 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**

- [1] 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.
- [2] Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics - Technical Document*.” J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

**See Also**

`confidenceBands` | `creditDefaultCopula` | `getScenarios` | `riskContribution` | `simulate` | `table`

**Topics**

- “Credit Simulation Using Copulas” on page 4-2
- “`creditDefaultCopula` Simulation Workflow” on page 4-5
- “Modeling Correlated Defaults with Copulas” on page 4-18
- “One-Factor Model Calibration”
- “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)
Contributions = riskContribution(cdc,Name,Value)
```

### 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`.

---

**Note** When creating a `creditDefaultCopula` object, you can set the 'UseParallel' property if you have Parallel Computing Toolbox. Once the 'UseParallel' property is set, parallel processing is used to compute `riskContribution`.

---

The `simulate` function must be run before `riskContribution` is used. For more information on using a `creditDefaultCopula` object, see `creditDefaultCopula`.

`Contributions = riskContribution(cdc,Name,Value)` adds an optional name-value pair argument for `VaRWindow`.

### 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: [100x5 table]
    FactorCorrelation: [2x2 double]
    VaRLevel: 0.9500
    UseParallel: 0
    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=10x5 table
   ID      EL      Std      VaR      CVaR
   ---  ---  ---  ---  ---
   1    0.036031    0.022762    0.083828    0.13625
   2    0.068357    0.039295    0.23373    0.24984
   3    1.2228    0.60699    2.3184    2.3775
   4    0.002877    0.00079014    0.0024248    0.0013137
   5    0.12127    0.037144    0.18474    0.24622
   6    0.12638    0.078506    0.39779    0.48334
   7    0.84284    0.3541    1.6221    1.8183
   8    0.00090088    0.00011379    0.0016463    0.00089197
   9    0.93117    0.87638    3.3868    3.9936
  10    0.26054    0.37918    1.7399    2.3042
```

Note: Due to simulation noise or numerical error, the VaR contribution can sometimes be greater than the CVaR contribution.

## 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `Contributions = riskContribution(cdc, 'VaRWindow', 0.3)`

### **VaRWindow** — Size of the window used to compute VaR contributions

0.05 (default) | numeric between 0 and 1

Size of the window used to compute VaR contributions, specified as the comma-separated pair consisting of `'VaRWindow'` and a scalar numeric with a percent value. Scenarios in the VaR scenario set are used to calculate the individual counterparty VaR contributions.

The default is 0.05, meaning that all scenarios with portfolio losses within 5 percent of the VaR are included when computing counterparty VaR contributions.

Data Types: double

## 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
- Std — Standard deviation of loss for the particular counterparty over the scenarios
- VaR — Value at risk 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.

## More About

### Risk Contributions

The riskContribution function reports the individual counterparty contributions to the total portfolio risk measures using four risk measures: expected loss (EL), standard deviation (Std), VaR, and CVaR.

- EL is the expected loss for each counterparty and is the mean of the counterparty's losses across all scenarios.
- Std is the standard deviation for counterparty  $i$ :

$$StdCont_i = Std_i \frac{\sum_j Std_j \rho_{ij}}{Std_\rho}$$

where

$Std_i$  is the standard deviation of losses from counterparty  $i$ .

$Std_\rho$  is the standard deviation of portfolio losses.

$\rho_{ij}$  is the correlation of the losses between counterparties  $i$  and  $j$ .

- VaR contribution is the mean of a counterparty's losses across all scenarios in which the total portfolio loss is within some small neighborhood around the Portfolio VaR. The default of the 'VaRWindow' parameter is 0.05 meaning that all scenarios in which the total portfolio loss is within 5% of the portfolio VaR are included in VaR neighborhood.
- CVaR is the mean of the counterparty's losses in the set of scenarios in which the total portfolio losses exceed the portfolio VaR.

## References

- [1] Glasserman, P. "Measuring Marginal Risk Contributions in Credit Portfolios." *Journal of Computational Finance*. Vol. 9, No. 2, Winter 2005/2006.



[2] Gupton, G., Finger, C., and Bhatia, M. *CreditMetrics - Technical Document.* J. P. Morgan, New York, 1997.

## See Also

`confidenceBands` | `creditDefaultCopula` | `getScenarios` | `portfolioRisk` | `simulate` | `table`

## Topics

“Credit Simulation Using Copulas” on page 4-2

“creditDefaultCopula Simulation Workflow” on page 4-5

“Modeling Correlated Defaults with Copulas” on page 4-18

“One-Factor Model Calibration”

“Corporate Credit Risk” on page 1-3

“Credit Simulation Using Copulas” on page 4-2

## External Websites

Parallel Computing with MATLAB (53 min 27 sec)

## 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`.

---

**Note** When creating a `creditDefaultCopula` object, you can set the 'UseParallel' property if you have Parallel Computing Toolbox. Once the 'UseParallel' property is set, parallel processing is used to compute `simulate`.

---

`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: [100x5 table]
FactorCorrelation: [2x2 double]
    VaRLevel: 0.9500
    UseParallel: 0
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: [100x5 table]
  FactorCorrelation: [2x2 double]
      VaRLevel: 0.9900
  UseParallel: 0
  PortfolioLosses: [1x100000 double]
```

You can use `riskContribution` with the `creditDefaultCopula` object to generate the risk Contributions table.

```
Contributions = riskContribution(cdc);
Contributions(1:10,:)
```

```
ans=10x5 table
   ID      EL      Std      VaR      CVaR
   ---  ---  ---  ---  ---
   1    0.036031    0.022762    0.083828    0.13625
   2    0.068357    0.039295    0.23373    0.24984
   3     1.2228     0.60699     2.3184     2.3775
   4    0.002877    0.00079014    0.0024248    0.0013137
   5     0.12127     0.037144     0.18474     0.24622
   6     0.12638     0.078506     0.39779     0.48334
   7     0.84284     0.3541     1.6221     1.8183
   8    0.00090088    0.00011379    0.0016463    0.00089197
   9     0.93117     0.87638     3.3868     3.9936
  10     0.26054     0.37918     1.7399     2.3042
```

## Input Arguments

### **cdc** — `creditDefaultCopula` object

object

`creditDefaultCopula` object, obtained from `creditDefaultCopula`.

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 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  $1,000,000 / (\text{Number-of-counterparties})$ . 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 `creditDefaultCopula`) are transformed to ensure that the latent variables have a mean of 0 and a variance of 1.

---

## References

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. "*CreditMetrics - Technical Document*." J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

`confidenceBands` | `creditDefaultCopula` | `getScenarios` | `portfolioRisk` | `riskContribution` | `table`

## Topics

- "Credit Simulation Using Copulas" on page 4-2
- "creditDefaultCopula Simulation Workflow" on page 4-5
- "Modeling Correlated Defaults with Copulas" on page 4-18
- "One-Factor Model Calibration"
- "Corporate Credit Risk" on page 1-3
- "Credit Simulation Using Copulas" on page 4-2

## External Websites

Parallel Computing with MATLAB (53 min 27 sec)

## 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 (see “Create `creditMigrationCopula`” on page 5-44 and “Properties” on page 5-48), 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`.

### Creation

#### Syntax

```
cmc = creditMigrationCopula(migrationValues, ratings, transitionMatrix, LGD,
Weights)
cmc = creditMigrationCopula( ____, Name, Value)
```

#### Description

`cmc = creditMigrationCopula(migrationValues, ratings, transitionMatrix, LGD, Weights)` creates a `creditMigrationCopula` object. The `creditMigrationCopula` object has the following properties:

- Portfolio on page 5-0 :

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 on page 5-0 :

Factor correlation matrix, a NumFactors-by-NumFactors matrix that defines the correlation between the risk factors.

- RatingLabels on page 5-0 :

The set of all possible credit ratings.

- TransitionMatrix on page 5-0 :

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 remains 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 on page 5-0 :

The value-at-risk level, used when reporting VaR and CVaR.

- PortfolioValues on page 5-0 :

A NumScenarios-by-1 vector of portfolio values. This property is empty until you use the simulate function.

`cmc = creditMigrationCopula( ____, Name, Value)` sets Properties on page 5-48 using name-value pairs and any of the arguments in the previous syntax. For example, `cmc = creditMigrationCopula(migrationValues, ratings, transitionMatrix, LGD, Weights, 'VaRLevel', 0.99)`. You can specify multiple name-value pairs as optional name-value pair arguments.

## 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 migrationValues input sets the Portfolio on page 5-0 property.

The migration value for the default rating (the last column of migrationValues input) is pre-recovery. This is a reference value (for example, face value, forward value at current rating, or other)

that is multiplied by the recovery rate during the simulation to get the value of the asset in the event of default. The recovery rate is defined as  $1 - \text{LGD}$ , where LGD is specified using the LGD input argument. The LGD is either a constant or a random number drawn from a beta distribution (see the description of the LGD input).

---

**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. The `ratings` input sets the `Portfolio` on page 5-0 property.

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 `transitionMatrix` input sets the `TransitionMatrix` on page 5-0 property.

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 remains 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. The LGD input sets the Portfolio on page 5-0 property.

LGD can alternatively be specified as a `NumCounterparties-by-2` matrix, where the first column holds the LGD mean values and the 2nd column holds the LGD standard deviations. Then, in the case of default, LGD values are drawn randomly from a beta distribution with provided parameters for the defaulting counterparty.

Valid open intervals for LGD mean and standard deviation are:

- For the first column, the mean values are between 0 and 1.
- For the second column, the LGD standard deviations are between 0 and  $\sqrt{m*(1-m)}$ .

Data Types: `double`

### Weights — Weights variable name

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. The `Weights` input sets the Portfolio on page 5-0 property.

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 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, '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. The ID name-value pair argument sets the Portfolio on page 5-0 property.

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. The VaRLevel name-value pair argument sets the VaRLevel on page 5-0 property.

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. The FactorCorrelation name-value pair argument sets the FactorCorrelation on page 5-0 property.

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. The RatingLabels name-value pair argument sets the RatingLabels on page 5-0 property.

Data Types: cell | double | string

**UseParallel — Flag to use parallel processing for simulations**

false (default) | logical with value of true or false

Flag to use parallel processing for simulations, specified as the comma-separated pair consisting of 'UseParallel' and a scalar value of true or false. The UseParallel name-value pair argument sets the UseParallel on page 5-0 property.

---

**Note** The 'UseParallel' property can only be set when creating a `creditMigrationCopula` object if you have Parallel Computing Toolbox. Once the 'UseParallel' property is set, parallel processing is used with `riskContribution` or `simulate`.

---

Data Types: logical

**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` object.

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'` when you create the `creditMigrationCopula` object.

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'` when you create the `creditMigrationCopula` object.

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'` when you create the `creditMigrationCopula` object. 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 remains 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'` when you create the `creditMigrationCopula` object.

Data Types: double

### PortfolioValues — Portfolio values

vector

Portfolio values, specified as a 1-by-NumScenarios vector. After creating the `creditMigrationCopula` object, the `PortfolioValues` property is empty. After you invoke the `simulate` function, `PortfolioValues` is populated with the portfolio values over each scenario.

Data Types: double

### UseParallel — Flag to use parallel processing for simulations

false (default) | logical with value of true or false

Flag to use parallel processing for simulations, specified using an optional name-value pair argument 'UseParallel' when you create a `creditMigrationCopula` object. The `UseParallel` name-value pair argument sets the `UseParallel` property.

---

**Note** The 'UseParallel' property can only be set when creating a `creditMigrationCopula` object if you have Parallel Computing Toolbox. Once the 'UseParallel' property is set, parallel processing is used with `riskContribution` or `simulate`.

---

Data Types: logical

## Object Functions

<code>simulate</code>	Simulate credit migrations using <code>creditMigrationCopula</code> object
<code>portfolioRisk</code>	Generate portfolio-level risk measurements
<code>riskContribution</code>	Generate risk contributions for each counterparty in portfolio
<code>confidenceBands</code>	Confidence interval bands
<code>getScenarios</code>	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 `creditMigrationCopula`.

```
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
UseParallel: 0
PortfolioValues: []

```

Set the VaRLevel to 99%.

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

The Portfolio property contains information about migration values, ratings, LGDs and weights.

```
head(cmc.Portfolio)
```

```
ans=8x5 table
```

ID	MigrationValues	Rating	LGD	Weights				
1	[1x8 double]	"A"	0.6509	0	0	0	0.5	0.5
2	[1x8 double]	"BBB"	0.8283	0	0.55	0	0	0.45
3	[1x8 double]	"AA"	0.6041	0	0.7	0	0	0.3
4	[1x8 double]	"BB"	0.6509	0	0.55	0	0	0.45
5	[1x8 double]	"BBB"	0.4966	0	0	0.75	0	0.25
6	[1x8 double]	"BB"	0.8283	0	0	0	0.65	0.35
7	[1x8 double]	"BB"	0.6041	0	0	0	0.65	0.35
8	[1x8 double]	"BB"	0.4873	0.5	0	0	0	0.5

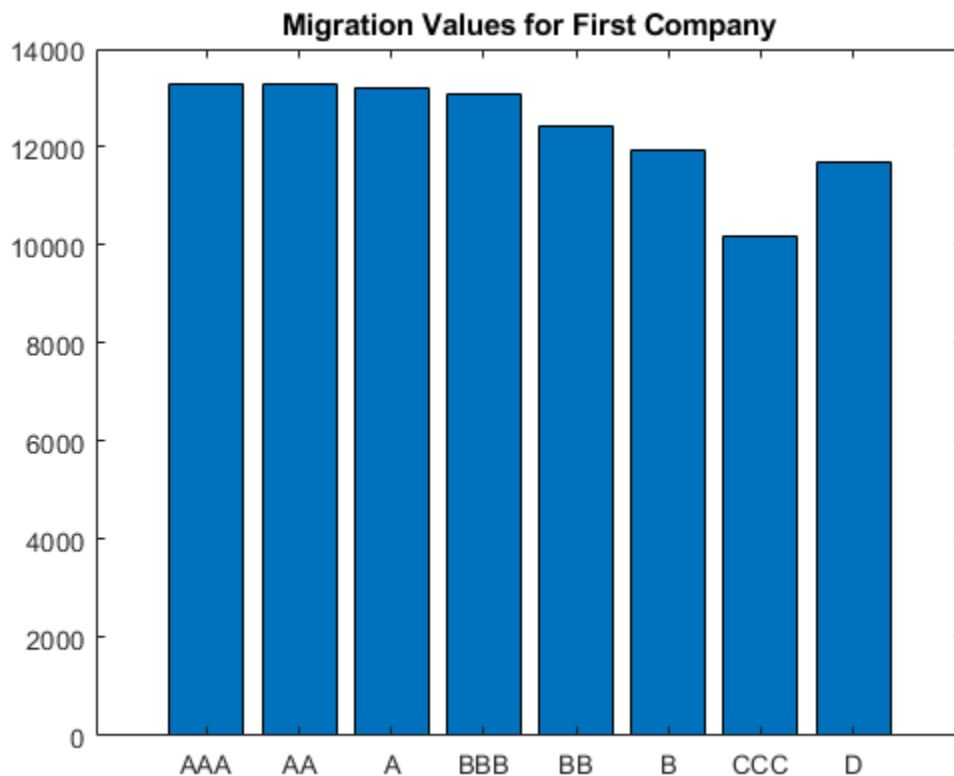
The columns in the migration values are in the same order of the ratings, with the default rating in the last column.

For example, these are the migration values for the first counterparty. Note that the value for default is higher than some of the non-default ratings. This is because the migration value for the default rating is a reference value (for example, face value, forward value at current rating, or other) that is multiplied by the recovery rate during the simulation to get the value of the asset in the event of default. The recovery rate is 1-LGD when the LGD input to `creditMigrationCopula` is a constant LGD value (the LGD input has one column). The recovery rate is a random quantity when the LGD input to `creditMigrationCopula` is specified as a mean and standard deviation for a beta distribution (the LGD input has two columns).

```

bar(cmc.Portfolio.MigrationValues(1,:))
xticklabels(cmc.RatingLabels)
title('Migration Values for First Company')

```



Use the `simulate` function to simulate 100,000 scenarios, and then view portfolio risk measures using the `portfolioRisk` function.

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

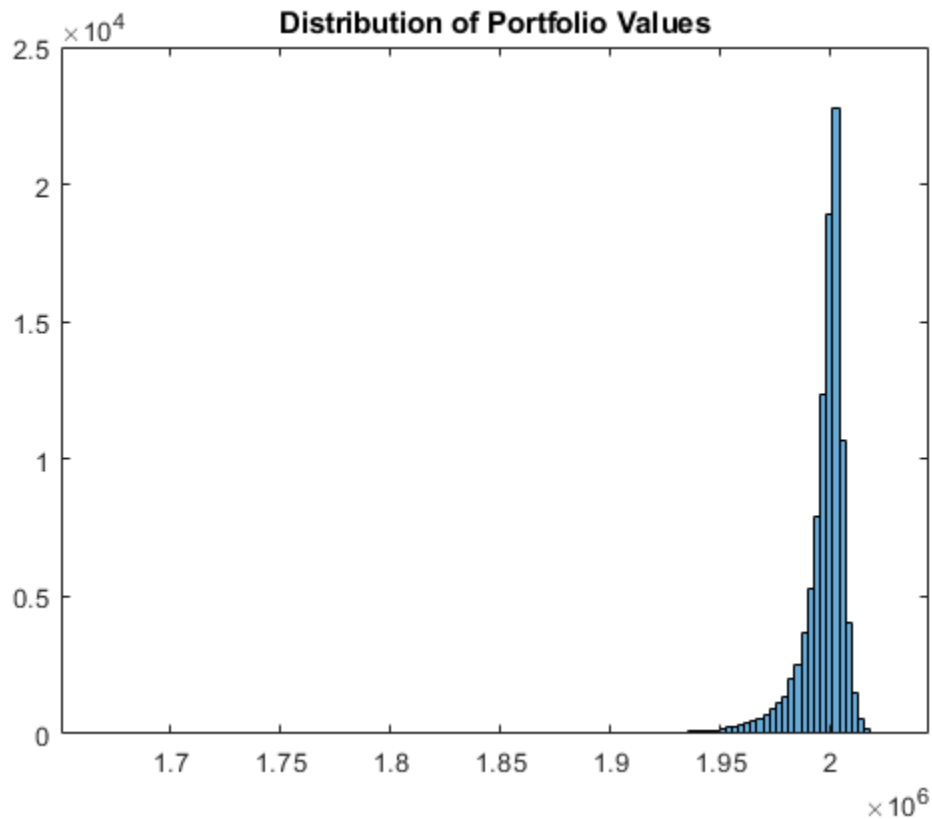
```
cmc =
  creditMigrationCopula with properties:
    Portfolio: [250x5 table]
    FactorCorrelation: [4x4 double]
    RatingLabels: [8x1 string]
    TransitionMatrix: [8x8 double]
    VaRLevel: 0.9900
    UseParallel: 0
    PortfolioValues: [1x100000 double]
```

```
portRisk = portfolioRisk(cmc)
```

```
portRisk=1x4 table
    EL      Std      VaR      CVaR
    ---    ---    ---    ---
    4515.9  12963  57176  83975
```

View a histogram of the portfolio values.

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



### 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 creditMigrationCopula.

```
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
    UseParallel: 0
    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)
```

```

cmc =
    creditMigrationCopula with properties:

        Portfolio: [250x5 table]
        FactorCorrelation: [4x4 double]
        RatingLabels: [8x1 string]
        TransitionMatrix: [8x8 double]
        VaRLevel: 0.9900
        UseParallel: 0
        PortfolioValues: [1x100000 double]

```

```
portRisk = portfolioRisk(cmc)
```

```

portRisk=1x4 table
    EL      Std      VaR      CVaR
    _____
    4515.9   12963   57176   83975

```

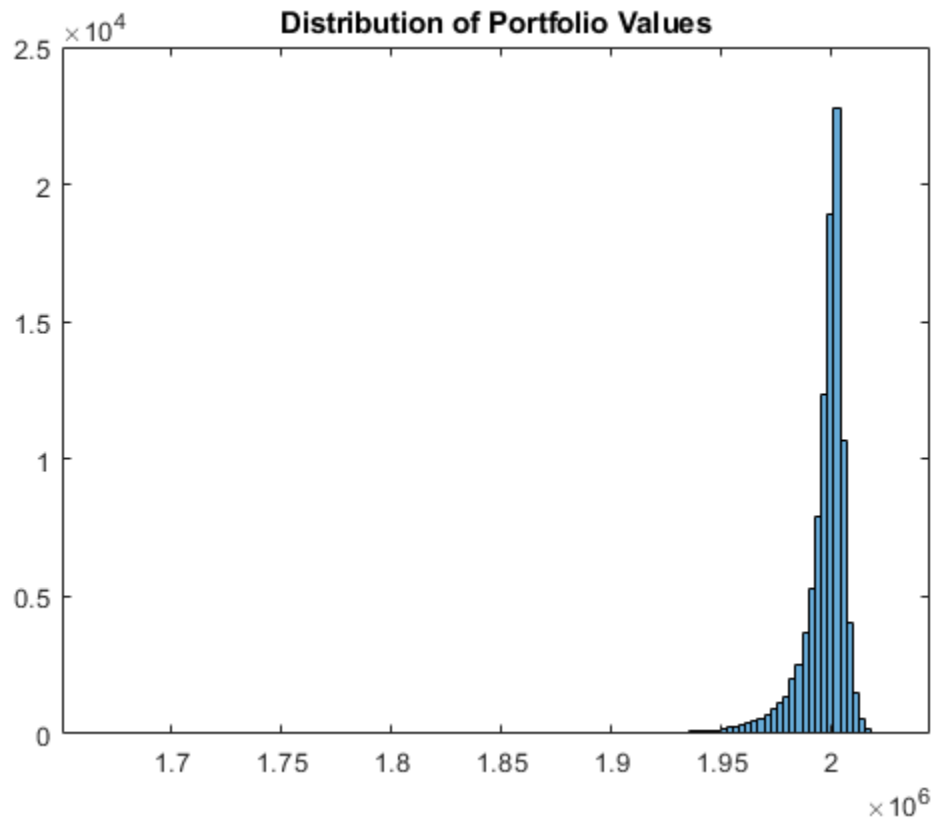
View a histogram of the portfolio values.

```

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

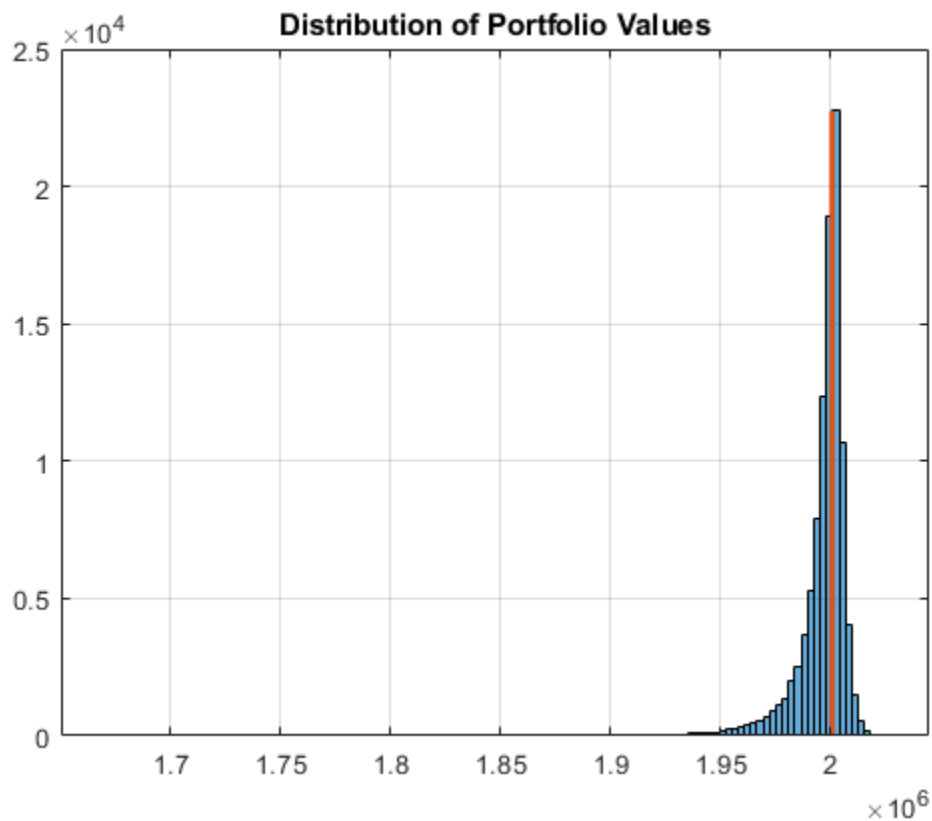
```





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
```



## References

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. "*CreditMetrics - Technical Document*." J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

confidenceBands | creditDefaultCopula | getScenarios | nearcorr | portfolioRisk | riskContribution | simulate | table

## Topics

"creditMigrationCopula Simulation Workflow" on page 4-10

“One-Factor Model Calibration”  
“Credit Rating Migration Risk” on page 1-7

**External Websites**

Parallel Computing with MATLAB (53 min 27 sec)

**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 `creditMigrationCopula`.

```
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
    UseParallel: 0
    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, 'NumPoints',50);
cbTable(1:10,:)
```

```
ans=10x4 table
  NumScenarios  Lower  Std  Upper
  _____  _____  _____  _____
      2000      11996  12308  12637
      4000      12871  13108  13354
      6000      12556  12744  12939
      8000      12830  12997  13168
     10000      12702  12850  13001
     12000      12784  12920  13059
     14000      12895  13022  13151
     16000      12747  12864  12983
     18000      12948  13060  13174
     20000      12971  13077  13186
```

## 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 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, 'NumPoints',50)`

## 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

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.

- [3] Gupton, G., Finger, C., and Bhatia, M. *CreditMetrics - Technical Document.* J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook.* 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA.* Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools.* Princeton University Press, 2005.

## **See Also**

creditMigrationCopula | getScenarios | portfolioRisk | riskContribution | simulate | table

## **Topics**

“creditMigrationCopula Simulation Workflow” on page 4-10

“One-Factor Model Calibration”

“Credit Rating Migration Risk” on page 1-7

## **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 `creditMigrationCopula`.

```
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
    UseParallel: 0
    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 = 10x2
104 ×
```

```
1.3082    1.3216
0.2893    0.2893
0.9788    0.9754
0.4503    0.4503
1.0376    1.0376
0.5795    0.5795
0.5350    0.5350
0.4956    0.4956
0.3537    0.3537
2.3492    2.3492
```

## 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

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.

- [3] Gupton, G., Finger, C., and Bhatia, M. *"CreditMetrics - Technical Document."* J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

**See Also**

confidenceBands | creditMigrationCopula | portfolioRisk | riskContribution | simulate

**Topics**

"creditMigrationCopula Simulation Workflow" on page 4-10  
"One-Factor Model Calibration"  
"Credit Rating Migration Risk" on page 1-7

**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 `creditMigrationCopula`.

```
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
    UseParallel: 0
    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=1×4 table
      EL      Std      VaR      CVaR
-----
4515.9  12963  57176  83975
```

```
confidenceIntervals=1×4 table
      EL      Std      VaR      CVaR
-----
4448.5  4583.4  12916  13011  56012  58278  82433  85517
```

## 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 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**

- [1] 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.
- [2] Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. “*CreditMetrics - Technical Document*.” J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## **See Also**

`confidenceBands` | `creditMigrationCopula` | `getScenarios` | `riskContribution` | `simulate` | `table`

## **Topics**

“creditMigrationCopula Simulation Workflow” on page 4-10  
 “One-Factor Model Calibration”  
 “Credit Rating Migration Risk” on page 1-7

## **Introduced in R2017a**

## riskContribution

Generate risk contributions for each counterparty in portfolio

### Syntax

```
Contributions = riskContribution(cmc)
Contributions = riskContribution(cmc,Name,Value)
```

### 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`.

---

**Note** When creating a `creditMigrationCopula` object, you can set the 'UseParallel' property if you have Parallel Computing Toolbox. Once the 'UseParallel' property is set, parallel processing is used to compute `riskContribution`.

---

Before you use the `riskContribution` function, you must run the `simulate` function. For more information on using a `creditMigrationCopula` object, see `creditMigrationCopula`.

`Contributions = riskContribution(cmc,Name,Value)` adds an optional name-value pair argument for `VaRWindow`.

### 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 `creditMigrationCopula`.

```
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
  UseParallel: 0
  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=10x5 table
   ID      EL      Std      VaR      CVaR
   ---  ---  ---  ---  ---
   1  15.521  41.153  238.72  279.18
   2    8.49  18.838  92.074  122.19
   3  6.0937  20.069  113.22  181.53
   4  6.6964  55.885  272.23  313.25
   5  23.583  73.905  360.32  573.39
   6  10.722  114.97  445.94  728.38
   7  1.8393  84.754  262.32  490.39
   8  11.711  39.768  175.84  253.29
   9  2.2154  4.4038  22.797  31.039
  10  1.7453  2.5545  9.8801  17.603

```

## 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `Contributions = riskContribution(cmc, 'VaRWindow', 0.3)`

**VaRWindow** — **Size of the window used to compute VaR contributions**

0.05 (default) | numeric between 0 and 1

Size of the window used to compute VaR contributions, specified as the comma-separated pair consisting of 'VaRWindow' and a scalar numeric with a percent value. Scenarios in the VaR scenario set are used to calculate the individual counterparty VaR contributions.

The default is 0.05, meaning that all scenarios with portfolio losses within 5 percent of the VaR are included when computing counterparty VaR contributions.

Data Types: double

## 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
- Std — Standard deviation of loss for the particular counterparty over the scenarios
- VaR — Value at risk 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.

## More About

### Risk Contributions

The riskContribution function reports the individual counterparty contributions to the total portfolio risk measures using four risk measures: expected loss (EL), standard deviation (Std), VaR, and CVaR.

- EL is the expected loss for each counterparty and is the mean of the counterparty's losses across all scenarios.
- Std is the standard deviation for counterparty  $i$ :

$$StdCont_i = Std_i \frac{\sum_j Std_j \rho_{ij}}{Std_\rho}$$

where

$Std_i$  is the standard deviation of losses from counterparty  $i$ .

$Std_\rho$  is the standard deviation of portfolio losses.

$\rho_{ij}$  is the correlation of the losses between counterparties  $i$  and  $j$ .

- VaR contribution is the mean of a counterparty's losses across all scenarios in which the total portfolio loss is within some small neighborhood around the Portfolio VaR. The default of the 'VaRWindow' parameter is 0.05 meaning that all scenarios in which the total portfolio loss is within 5% of the portfolio VaR are included in VaR neighborhood.



- CVaR is the mean of the counterparty's losses in the set of scenarios in which the total portfolio losses exceed the portfolio VaR.

## References

- [1] Glasserman, P. "Measuring Marginal Risk Contributions in Credit Portfolios." *Journal of Computational Finance*. Vol. 9, No. 2, Winter 2005/2006.
- [2] Gupton, G., Finger, C., and Bhatia, M. "*CreditMetrics - Technical Document*." J. P. Morgan, New York, 1997.

## See Also

`confidenceBands` | `creditMigrationCopula` | `getScenarios` | `portfolioRisk` | `simulate` | `table`

## Topics

"creditMigrationCopula Simulation Workflow" on page 4-10  
"One-Factor Model Calibration"  
"Credit Rating Migration Risk" on page 1-7

## External Websites

Parallel Computing with MATLAB (53 min 27 sec)

## 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`.

---

**Note** When creating a `creditMigrationCopula` object, you can set the 'UseParallel' property if you have Parallel Computing Toolbox. Once the 'UseParallel' property is set, parallel processing is used to compute `simulate`.

---

`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 `creditMigrationCopula`.

```
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
    UseParallel: 0
```

```
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: [250x5 table]
  FactorCorrelation: [4x4 double]
      RatingLabels: [8x1 string]
  TransitionMatrix: [8x8 double]
      VaRLevel: 0.9900
      UseParallel: 0
  PortfolioValues: [1x100000 double]
```

You can use the `riskContribution` function with the `creditMigrationCopula` object to generate the risk Contributions table.

```
Contributions = riskContribution(cmc);
Contributions(1:10,:)
```

```
ans=10x5 table
   ID      EL      Std      VaR      CVaR
   ---  ---  ---  ---  ---
   1  15.521  41.153  238.72  279.18
   2   8.49  18.838  92.074  122.19
   3  6.0937  20.069  113.22  181.53
   4  6.6964  55.885  272.23  313.25
   5  23.583  73.905  360.32  573.39
   6  10.722  114.97  445.94  728.38
   7   1.8393  84.754  262.32  490.39
   8  11.711  39.768  175.84  253.29
   9   2.2154   4.4038  22.797  31.039
  10   1.7453   2.5545   9.8801  17.603
```

## Input Arguments

**cmc** — **creditMigrationCopula** object  
object

`creditMigrationCopula` object, obtained from `creditMigrationCopula`.

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 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 creditMigrationCopula) are transformed to ensure that the latent variables have a mean of 0 and a variance of 1.

---

## References

- [1] 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.
- [2] Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119-149.
- [3] Gupton, G., Finger, C., and Bhatia, M. "CreditMetrics - Technical Document." J. P. Morgan, New York, 1997.
- [4] Jorion, P. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [5] Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.
- [6] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management: Concepts, Techniques, and Tools*. Princeton University Press, 2005.

## See Also

confidenceBands | creditMigrationCopula | getScenarios | portfolioRisk | riskContribution | table

## Topics

"creditMigrationCopula Simulation Workflow" on page 4-10  
"One-Factor Model Calibration"  
"Credit Rating Migration Risk" on page 1-7

## External Websites

Parallel Computing with MATLAB (53 min 27 sec)

## Introduced in R2017a

## esbacktest

Create `esbacktest` object to run suite of table-based expected shortfall (ES) backtests by Acerbi and Szekely

### Description

The general workflow is:

- 1 Load or generate the data for the ES backtesting analysis.
- 2 Create an `esbacktest` object. For more information, see “Create `esbacktest`” on page 5-76 and “Properties” on page 5-79.
- 3 Use the `summary` function to generate a summary report for the number of observations, expected, and observed average severity ratio.
- 4 Use the `runtests` function to run all tests at once.
- 5 For additional test details, run the following individual tests:
  - `unconditionalNormal` — Unconditional ES backtest assuming returns distribution is normal
  - `unconditionalT` — Unconditional ES backtest assuming returns distribution is t

For more information, see “Overview of Expected Shortfall Backtesting” on page 2-21.

### Creation

#### Syntax

```
ebt = esbacktest(PortfolioData, VaRData, ESData)
ebt = esbacktest( ____, Name, Value)
```

#### Description

`ebt = esbacktest(PortfolioData, VaRData, ESData)` creates an `esbacktest` (`ebt`) object using portfolio outcomes data and corresponding value-at-risk (VaR) and ES data. The `ebt` object has the following properties:

- `PortfolioData` on page 5-0 — `NumRows-by-1` numeric array containing a copy of the `PortfolioData`
- `VaRData` on page 5-0 — `NumRows-by-NumVaRs` numeric array containing a copy of the `VaRData`
- `ESData` on page 5-0 — `NumRows-by-NumVaRs` numeric array containing a copy of the `ESData`
- `PortfolioID` on page 5-0 — String containing the `PortfolioID`
- `VaRID` on page 5-0 — `1-by-NumVaRs` string vector containing the `VaRIDs` for the corresponding columns in `VaRData`
- `VaRLevel` on page 5-0 — `1-by-NumVaRs` numeric array containing the `VaRLevels` for the corresponding columns in `VaRData`

---

**Note**

- The required input arguments for `PortfolioData`, `VaRData`, and `ESData` must all be in the same units. These arguments can be expressed as returns or as profits and losses. There are no validations in the `esbacktest` object regarding the units of these arguments.
  - If there are missing values (NaNs) in `PortfolioData`, `VaRData`, and `ESData`, the row of data is discarded before applying the tests. Therefore, a different number of observations are reported for models with a different number of missing values. The reported number of observations equals the original number of rows minus the number of missing values. To determine if there are discarded rows, use the 'Missing' column of the summary report.
  - Because the critical values are precomputed, only certain numbers of observations, VaR levels, and test levels are supported.
    - The number of observations (number of rows in the data minus the number of missing values) must be from 200 through 5000.
    - The `VaRLevel` input argument must be between 0.90 and 0.99; the default is 0.95.
    - The `TestLevel` (test confidence level) input argument for the `runtests`, `unconditionalNormal`, and `unconditionalT` functions must be between 0.5 and 0.9999; the default is 0.95.
- 

`ebt = esbacktest( ____, Name, Value)` sets Properties on page 5-79 using name-value pairs and any of the arguments in the previous syntax. For example, `ebt = esbacktest(PortfolioData, VaRData, ESData, 'VaRID', 'TotalVaR', 'VaRLevel', .99)`. You can specify multiple name-value pairs as optional name-value pair arguments.

**Input Arguments****PortfolioData — Portfolio outcomes data**

NumRows-by-1 numeric array | NumRows-by-1 numeric columns table | NumRows-by-1 numeric columns timetable

Portfolio outcomes data, specified as a NumRows-by-1 numeric array, NumRows-by-1 numeric columns table, or a NumRows-by-1 timetable with a numeric column containing portfolio outcomes data. The `PortfolioData` input argument sets the `PortfolioData` on page 5-0 property.

---

**Note** `PortfolioData` must be in the same units as `VaRData` and `ESData`. `PortfolioData`, `VaRData`, and `ESData` can be expressed as returns or as profits and losses. There are no validations in the `esbacktest` object regarding the units of portfolio, VaR, and ES data.

---

Data Types: double | table | timetable

**VaRData — Value-at-risk (VaR) data**

NumRows-by-NumVaRs numeric array | NumRows-by-NumVaRs table with numeric columns | NumRows-by-NumVaRs timetable with numeric columns

Value-at-risk (VaR) data, specified as a NumRows-by-NumVaRs numeric array, NumRows-by-NumVaRs numeric columns table, or NumRows-by-NumVaRs timetable with numeric columns. The `VaRData` input argument sets the `VaRData` on page 5-0 property.

Negative `VaRData` values are allowed. However, negative VaR values indicate a highly profitable portfolio that cannot lose money at the given VaR confidence level. The worst-case scenario at the given confidence level is still a profit.

---

**Note** `VaRData` must be in the same units as `PortfolioData` and `ESData`. `VaRData`, `PortfolioData`, and `ESData` can be expressed as returns or as profits and losses. There are no validations in the `esbacktest` object regarding the units of portfolio, VaR, and ES data.

---

Data Types: `double` | `table` | `timetable`

#### **ESData — Expected shortfall data**

`NumRows-by-NumVaRs` positive numeric array | `NumRows-by-NumVaRs` table with positive numeric columns | `NumRows-by-NumVaRs` timetable with positive numeric columns

Expected shortfall data, specified as a `NumRows-by-NumVaRs` positive numeric array, `NumRows-by-NumVaRs` table with positive numeric columns, or `NumRows-by-NumVaRs` timetable with positive numeric columns containing ES data. The `ESData` input argument sets the `ESData` on page 5-0 property.

---

**Note** `ESData` must be in the same units as `PortfolioData` and `VaRData`. `ESData`, `PortfolioData`, and `VaRData` can be expressed as returns or as profits and losses. There are no validations in the `esbacktest` object regarding the units of portfolio, VaR, and ES data.

---

Data Types: `double` | `table` | `timetable`

#### **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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `ebt =`

```
esbacktest(PortfolioData, VaRData, ESData, 'VaRID', 'TotalVaR', 'VaRLevel', .99)
```

#### **PortfolioID — User-defined ID**

character vector | string

User-defined ID for `PortfolioData` input, specified as the comma-separated pair consisting of `'PortfolioID'` and a character vector or string. The `PortfolioID` name-value pair argument sets the `PortfolioID` on page 5-0 property.

If `PortfolioData` is a numeric array, the default value for `PortfolioID` is `'Portfolio'`. If `PortfolioData` is a table, `PortfolioID` is set to the corresponding variable name in the table, by default.

Data Types: `char` | `string`

#### **VaRID — VaR identifier**

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, cell array of character vectors, string, or string array.



Multiple VaRID values are specified using a 1-by-NumVaRs (or NumVaRs-by-1) cell array of character vectors or a string vector with user-defined IDs for the VaRData columns. A single VaRID identifies a VaRData column and the corresponding ESData column. The VaRID name-value pair argument sets the VaRID on page 5-0 property.

If NumVaRs = 1, the default value for VaRID is 'VaR'. If 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 between 0.90 and 0.99

VaR confidence level, specified as the comma-separated pair consisting of 'VaRLevel' and a numeric value between 0.90 and 0.99 or a 1-by-NumVaRs (or NumVaRs-by-1) numeric array. The VaRLevel name-value pair argument sets the VaRLevel on page 5-0 property.

Data Types: double

## **Properties**

#### **PortfolioData — Portfolio data for ES backtesting analysis**

numeric array

Portfolio data for ES backtesting analysis, specified as a NumRows-by-1 numeric array containing a copy of the portfolio data.

Data Types: double

#### **VaRData — VaR data for ES backtesting analysis**

numeric array

VaR data for ES backtesting analysis, specified as a NumRows-by-NumVaRs numeric array containing a copy of the VaR data.

Data Types: double

#### **ESData — Expected shortfall data for ES backtesting analysis**

numeric array

Expected shortfall data for ES backtesting analysis, specified as a NumRows-by-NumVaRs numeric array containing a copy of the ESData.

Data Types: double

#### **PortfolioID — Portfolio identifier**

string

Portfolio identifier, specified as a string.

Data Types: string

#### **VaRID — VaR identifier**

string | 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

### VaRLevel – VaR level

numeric array with values between 0.90 and 0.99

VaR level, specified as a 1-by-NumVaRs numeric array with values from 0.90 through 0.99, containing the VaR levels for the corresponding columns in VaRData.

Data Types: double

esbacktest Property	Set or Modify Property from Command Line Using esbacktest	Modify Property Using Dot Notation
PortfolioData	Yes	No
VaRData	Yes	No
ESData	Yes	No
PortfolioID	Yes	Yes
VaRID	Yes	Yes
VaRLevel	Yes	Yes

## Object Functions

summary	Basic expected shortfall (ES) report on failures and severity
runtests	Run all expected shortfall (ES) backtests for esbacktest object
unconditionalNormal	Unconditional expected shortfall (ES) backtest of Acerbi-Szekely with critical values for normal distributions
unconditionalT	Unconditional expected shortfall (ES) backtest of Acerbi-Szekely with critical values for t distributions

## Examples

### Create esbacktest Object and Run ES Backtests for Single VaR at 95%

esbacktest takes in portfolio outcomes data, the corresponding value-at-risk (VaR) data, and the expected shortfall (ES) data and returns an esbacktest object.

Create an esbacktest object.

```
load ESBacktestData
ebt = esbacktest>Returns, VaRModel1, ESModel1, 'VaRLevel', VaRLevel)

ebt =
  esbacktest with properties:

    PortfolioData: [1966x1 double]
    VaRData: [1966x1 double]
    ESData: [1966x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
```

```
VaRLevel: 0.9750
```

`ebt`, the `esbacktest` object, contains a copy of the given portfolio data (`PortfolioData` property), the given VaR data (`VaRData` property), and the given ES data (`ESData`) property. The object also contains all combinations of portfolio ID, VaR ID, and VaR level to be tested (`PortfolioID`, `VaRID`, and `VaRLevel` properties).

Run the tests using the `ebt` object.

```
runtests(ebt)
ans=1x5 table
  PortfolioID   VaRID   VaRLevel   UnconditionalNormal   UnconditionalT
  _____   _____   _____   _____   _____
  "Portfolio"   "VaR"     0.975      reject         reject
```

Change the `PortfolioID` and `VaRID` properties using dot notation. For more information on creating an `esbacktest` object, see `esbacktest`.

```
ebt.PortfolioID = 'S&P';
ebt.VaRID = 'Normal at 97.5%';
disp(ebt)
```

```
esbacktest with properties:
  PortfolioData: [1966x1 double]
  VaRData: [1966x1 double]
  ESData: [1966x1 double]
  PortfolioID: "S&P"
  VaRID: "Normal at 97.5%"
  VaRLevel: 0.9750
```

Run all tests using the updated `esbacktest` object.

```
runtests(ebt)
ans=1x5 table
  PortfolioID   VaRID   VaRLevel   UnconditionalNormal   UnconditionalT
  _____   _____   _____   _____   _____
  "S&P"        "Normal at 97.5%"   0.975      reject         reject
```

## References

- [1] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.
- [2] Basel Committee on Banking Supervision. "Minimum Capital Requirements for Market Risk". January, 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

`esbacktestbysim` | `runtests` | `summary` | `table` | `timetable` | `unconditionalNormal` | `unconditionalT` | `varbacktest`

**Topics**

“Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

**Introduced in R2017b**

## summary

Basic expected shortfall (ES) report on failures and severity

### Syntax

```
S = summary(ebt)
```

### Description

`S = summary(ebt)` returns a basic report on the given `esbacktest` data, including the number of observations, number of failures, observed confidence level, and so on (see `S` for details).

### Examples

#### Generate an ES Summary Report

Create an `esbacktest` object.

```
load ESBacktestData
ebt = esbacktest>Returns, VaRModel1, ESMoel1, 'VaRLevel', VaRLevel)
```

```
ebt =
  esbacktest with properties:
```

```
PortfolioData: [1966x1 double]
VaRData: [1966x1 double]
ESData: [1966x1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9750
```

Generate the ES summary report.

```
S = summary(ebt)
```

```
S=1x11 table
PortfolioID VaRID VaRLevel ObservedLevel ExpectedSeverity ObservedSeverity
-----
"Portfolio" "VaR" 0.975 0.97101 1.1928 1.4221
```

### Input Arguments

**ebt** — `esbacktest` object  
object

`esbacktest` (`ebt`) object, contains a copy of the given data (the `PortfolioData`, `VarData`, and `ESData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an `esbacktest` object, see `esbacktest`.

## 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 the number of periods without failures divided by number of observations
- `'ExpectedSeverity'` — Expected average severity ratio, that is, the average ratio of ES to VaR over the periods with VaR failures
- `'ObservedSeverity'` — Observed average severity ratio, that is, the average ratio of loss to VaR over the periods with VaR failures
- `'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 1 minus the VaR level
- `'Ratio'` — Ratio of number of failures to expected number of failures
- `'Missing'` — Number of periods with missing values removed from the sample

---

**Note** The `'ExpectedSeverity'` and `'ObservedSeverity'` ratios are undefined (NaN) when there are no VaR failures in the data.

---

## See Also

`esbacktest` | `runtests` | `unconditionalNormal` | `unconditionalT`

### Topics

“Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

### Introduced in R2017b

## runtests

Run all expected shortfall (ES) backtests for `esbacktest` object

### Syntax

```
TestResults = runtests(ebt)
TestResults = runtests(ebt,Name,Value)
```

### Description

`TestResults = runtests(ebt)` runs all the tests for the `esbacktest` object. `runtests` reports only the final test result. For test details, such as  $p$ -values, run the individual tests:

- `unconditionalNormal`
- `unconditionalT`

`TestResults = runtests(ebt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

### Examples

#### Run All ES Backtests

Create an `esbacktest` object.

```
load ESBacktestData
ebt = esbacktest>Returns, VaRModel1, ESModel1, 'VaRLevel', VaRLevel)
```

```
ebt =
  esbacktest with properties:

    PortfolioData: [1966x1 double]
      VaRData: [1966x1 double]
      ESData: [1966x1 double]
    PortfolioID: "Portfolio"
      VaRID: "VaR"
    VaRLevel: 0.9750
```

Generate the `TestResults` report for all ES backtests.

```
TestResults = runtests(ebt, 'TestLevel', 0.99)
```

```
TestResults=1x5 table
  PortfolioID  VaRID  VaRLevel  UnconditionalNormal  UnconditionalT
  _____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.975    reject             accept
```

Generate the `TestResults` report for all ES backtests using the name-value argument for `'ShowDetails'` to display the test confidence level.

```
TestResults = runtests(ebt, 'TestLevel', 0.99, 'ShowDetails', true)
```

```
TestResults=1x6 table
  PortfolioID  VaRID  VaRLevel  UnconditionalNormal  UnconditionalT  TestLevel
  _____  _____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.975    reject           accept           0.99
```

## Input Arguments

### **ebt** — esbacktest object

object

esbacktest (ebt) object, which contains a copy of the given data (the `PortfolioData`, `VarData`, and `ESData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktest object, see `esbacktest`.

### **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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `TestResults = runtests(ebt, 'TestLevel', 0.99)`

### **TestLevel** — Test confidence level

0.95 (default) | numeric value between 0.5 and 0.9999

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric value between 0.5 and 0.9999.

Data Types: `double`

### **ShowDetails** — Indicates if the output displays a column showing the test confidence level

false (default) | scalar logical with a value of `true` or `false`

Indicates if the output displays a column showing the test confidence level, specified as the comma-separated pair consisting of `'ShowDetails'` and a scalar logical value.

Data Types: `logical`

## 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
- 'UnconditionalNormal' — Categorical array with categories 'accept' and 'reject' that indicate the result of the unconditional normal test
- 'UnconditionalT' — Categorical array with categories 'accept' and 'reject' that indicate the result of the unconditional  $t$  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

`esbacktest` | `summary` | `unconditionalNormal` | `unconditionalT`

## Topics

“Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

**Introduced in R2017b**

## unconditionalNormal

Unconditional expected shortfall (ES) backtest of Acerbi-Szekely with critical values for normal distributions

### Syntax

```
TestResults = unconditionalNormal(ebt)
TestResults = unconditionalNormal(ebt,Name,Value)
```

### Description

`TestResults = unconditionalNormal(ebt)` runs the unconditional expected shortfall (ES) backtest of Acerbi-Szekely (2014) using precomputed critical values and assuming that the returns distribution is standard normal.

`TestResults = unconditionalNormal(ebt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

### Examples

#### Run an Unconditional ES Backtest

Create an `esbacktest` object.

```
load ESBacktestData
ebt = esbacktest>Returns, VaRModel1, ESMModel1, 'VaRLevel', VaRLevel)
```

```
ebt =
  esbacktest with properties:
```

```
PortfolioData: [1966x1 double]
VaRData: [1966x1 double]
ESData: [1966x1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9750
```

Generate the `TestResults` report for the unconditional ES backtest that assumes the returns distribution is standard normal.

```
TestResults = unconditionalNormal(ebt, 'TestLevel', 0.99)
```

```
TestResults=1x9 table
PortfolioID VaRID VaRLevel UnconditionalNormal PValue TestStatistic Cri
-----
"Portfolio" "VaR" 0.975 reject 0.0054099 -0.38265 -0
```

## Input Arguments

### ebt — esbacktest object

object

esbacktest (ebt) object, which contains a copy of the given data (the PortfolioData, VarData, and ESData properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktest object, see esbacktest.

### 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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: TestResults = unconditionalNormal(ebt, 'TestLevel', 0.99)

### TestLevel — Test confidence level

0.95 (default) | numeric value between 0.5 and 0.9999

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric value between 0.5 and 0.9999.

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.
- 'UnconditionalNormal' — Categorical array with categories 'accept' and 'reject' that indicate the result of the unconditional normal test.
- 'PValue' — P-value of the unconditional normal test, interpolated from the precomputed critical values under the assumption that the returns follow a standard normal distribution.

---

**Note**  $p$ -values  $< 0.0001$  are truncated to the minimum (0.0001) and  $p$ -values  $> 0.5$  are displayed as a maximum (0.5).

---

- 'TestStatistic' — Unconditional normal test statistic.
- 'CriticalValue' — Precomputed critical value for the corresponding test level and number of observations. Critical values are obtained under the assumption that the returns follow a standard normal distribution.
- 'Observations' — Number of observations.
- 'TestLevel' — Test confidence level.

---

**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.

---

## More About

### Unconditional Test by Acerbi and Szekely

The unconditional test (also known as the second Acerbi-Szekely test) scales the losses by the corresponding ES value.

The unconditional test statistic is based on the unconditional relationship

$$ES_t = -E_t \left[ \frac{X_t I_t}{P_{VaR}} \right]$$

where

$X_t$  is the portfolio outcome, that is, the portfolio return or portfolio profit and loss for period  $t$ .

$P_{VaR}$  is the probability of VaR failure defined as  $1 - VaR$  level.

$ES_t$  is the estimated expected shortfall for period  $t$ .

$I_t$  is the VaR failure indicator on period  $t$  with a value of 1 if  $X_t < -VaR$ , and 0 otherwise.

The unconditional test statistic is defined as



The critical values for the unconditional test statistic, which form the basis for table-based tests, are stable across a range of distributions. The `esbacktest` class runs the unconditional test against precomputed critical values under two distributional assumptions: normal distribution (thin tails) using `unconditionalNormal` and  $t$  distribution with 3 degrees of freedom (heavy tails) using `unconditionalT`.

## References

[1] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.

## See Also

`esbacktest` | `runtests` | `summary` | `unconditionalT`

## Topics

“Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

## Introduced in R2017b

# unconditionalT

Unconditional expected shortfall (ES) backtest of Acerbi-Szekely with critical values for  $t$  distributions

## Syntax

```
TestResults = unconditionalT(ebt)
TestResults = unconditionalT(ebt,Name,Value)
```

## Description

`TestResults = unconditionalT(ebt)` runs the unconditional expected shortfall (ES) backtest of Acerbi-Szekely (2014) using precomputed critical values and assuming that the returns distribution is  $t$  with 3 degrees of freedom.

`TestResults = unconditionalT(ebt,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

## Examples

### Run an Unconditional $t$ ES Backtest

Create an `esbacktest` object.

```
load ESBacktestData
ebt = esbacktest>Returns, VaRModel1, ESMModel1, 'VaRLevel', VaRLevel)
```

```
ebt =
  esbacktest with properties:
```

```
PortfolioData: [1966x1 double]
VaRData: [1966x1 double]
ESData: [1966x1 double]
PortfolioID: "Portfolio"
VaRID: "VaR"
VaRLevel: 0.9750
```

Generate the `TestResults` report for the unconditional  $t$  ES backtest that assumes the returns distribution is  $t$  with 3 degrees of freedom.

```
TestResults = unconditionalT(ebt, 'TestLevel', 0.99)
```

```
TestResults=1x9 table
```

PortfolioID	VaRID	VaRLevel	UnconditionalT	PValue	TestStatistic	CriticalValue
"Portfolio"	"VaR"	0.975	accept	0.018566	-0.38265	-0.4298

## Input Arguments

### ebt — esbacktest object

object

esbacktest (ebt) object, which contains a copy of the given data (the `PortfolioData`, `VarData`, and `ESData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktest object, see `esbacktest`.

### 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `TestResults = unconditionalT(ebt, 'TestLevel', 0.99)`

### TestLevel — Test confidence level

0.95 (default) | numeric value between 0.5 and 0.9999

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric value between 0.5 and 0.9999.

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.
- `'UnconditionalT'` — Categorical array with categories 'accept' and 'reject' indicating the result of the unconditional  $t$  test.
- `'PValue'` — P-value of the unconditional  $t$  test, interpolated from the precomputed critical values under the assumption that the returns follow a standard normal distribution.

---

**Note**  $p$ -values  $< 0.0001$  are truncated to the minimum (0.0001) and  $p$ -values  $> 0.5$  are displayed as a maximum (0.5).

---

- `'TestStatistic'` — Unconditional  $t$  test statistic.
- `'CriticalValue'` — Precomputed critical value for the corresponding test level and number of observations. Critical values are obtained under the assumption that the returns follow a  $t$  distribution with 3 degrees of freedom.
- `'Observations'` — Number of observations.
- `'TestLevel'` — Test confidence level.

---

**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.

---

## More About

### Unconditional Test by Acerbi and Szekely

The unconditional test (also known as the second Acerbi-Szekely test) scales the losses by the corresponding ES value.

The unconditional test statistic is based on the unconditional relationship

$$ES_t = -E_t \left[ \frac{X_t I_t}{P_{VaR}} \right]$$

where

$X_t$  is the portfolio outcome, that is, the portfolio return or portfolio profit and loss for period  $t$ .

$P_{VaR}$  is the probability of VaR failure defined as  $1 - VaR$  level.

$ES_t$  is the estimated expected shortfall for period  $t$ .

$I_t$  is the VaR failure indicator on period  $t$  with a value of 1 if  $X_t < -VaR$ , and 0 otherwise.

The unconditional test statistic is defined as:



The critical values for the unconditional test statistic, which form the basis for table-based tests, are stable across a range of distributions. The `esbacktest` class runs the unconditional test against precomputed critical values under two distributional assumptions: normal distribution (thin tails) using `unconditionalNormal` and  $t$  distribution with 3 degrees of freedom (heavy tails) using `unconditionalT`.

## References

[1] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.

## See Also

`esbacktest` | `runtests` | `summary` | `unconditionalNormal`

## Topics

“Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information” on page 2-29

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

## Introduced in R2017b

## esbacktestbysim

Create `esbacktestbysim` object to run simulation-based suite of expected shortfall (ES) backtests by Acerbi and Szekely

### Description

The general workflow is:

- 1 Load or generate the data for the ES backtesting analysis.
- 2 Create an `esbacktestbysim` object. For more information, see “Create `esbacktestbysim`” on page 5-94.
- 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:
  - `conditional` — Conditional test of Acerbi-Szekely (2014)
  - `unconditional` — Unconditional test of Acerbi-Szekely (2014)
  - `quantile` — Quantile test of Acerbi-Szekely (2014)

For more information, see “Overview of Expected Shortfall Backtesting” on page 2-21.

### Creation

#### Syntax

```
ebts = esbacktestbysim(PortfolioData, VaRData, ESData, DistributionName)
ebts = esbacktestbysim( ____, Name, Value)
```

#### Description

`ebts = esbacktestbysim(PortfolioData, VaRData, ESData, DistributionName)` creates an `esbacktestbysim` (`ebts`) object and simulates portfolio outcome scenarios to compute critical values for three tests:

- `conditional`
- `unconditional`
- `quantile`

The `ebts` object has the following properties:

- `PortfolioData` on page 5-0 — `NumRows-by-1` numeric array containing a copy of the `PortfolioData`
- `VaRData` on page 5-0 — `NumRows-by-NumVaRs` numeric array containing a copy of the `VaRData`



- `ESData` on page 5-0 — `NumRows-by-NumVaRs` numeric array containing a copy of the `ESData`
- `Distribution` on page 5-0 — Structure containing the model information, including model distribution name and distribution parameters. For example, for a normal distribution, `Distribution` has fields `'Name'`, `'Mean'`, and `'StandardDeviation'`, with values set to the corresponding inputs.
- `PortfolioID` on page 5-0 — String containing the `PortfolioID`
- `VaRID` on page 5-0 — `1-by-NumVaRs` string vector containing the `VaRIDs` for the corresponding columns in `VaRData`
- `VaRLevel` on page 5-0 — `1-by-NumVaRs` numeric array containing the `VaRLevels` for the corresponding columns in `VaRData`.

---

### Note

- The required input arguments for `PortfolioData`, `VaRData`, and `ESData` must all be in the same units. These arguments can be expressed as returns or as profits and losses. There are no validations in the `esbacktestbysim` object regarding the units of these arguments.
  - If there are missing values (NaNs) in `PortfolioData`, `VaRData`, `ESData`, or `Distribution` parameters data, the row of data is discarded before applying the tests. Therefore, a different number of observations are reported for models with a different number of missing values. The reported number of observations equals the original number of rows minus the number of missing values. To determine if there are discarded rows, use the `'Missing'` column of the summary report.
- 

`ebts = esbacktestbysim( ____, Name, Value)` sets `Properties` on page 5-99 using name-value pairs and any of the arguments in the previous syntax. For example, `ebts = esbacktestbysim(PortfolioData, VaRData, ESData, DistributionName, 'VaRID', 'TotalVaR', 'VaRLevel', .99)`. You can specify multiple name-value pairs.

### Input Arguments

#### **PortfolioData — Portfolio outcomes data**

`NumRows-by-1` numeric array | `NumRows-by-1` numeric columns table | `NumRows-by-1` numeric columns timetable

Portfolio outcomes data, specified as a `NumRows-by-1` numeric array, `NumRows-by-1` table, or a `NumRows-by-1` timetable with a numeric column containing portfolio outcomes data. The `PortfolioData` input argument sets the `PortfolioData` on page 5-0 property.

---

**Note** `PortfolioData` data must be in the same units as `VaRData` and `ESData`. There are no validations in the `esbacktestbysim` object regarding the units of portfolio, VaR, and ES data. `PortfolioData`, `VaRData`, and `ESData` can be expressed as returns or as profits and losses.

---

Data Types: `double` | `table` | `timetable`

#### **VaRData — Value-at-risk (VaR) data**

`NumRows-by-NumVaRs` numeric array | `NumRows-by-NumVaRs` table with numeric columns | `NumRows-by-NumVaRs` timetable with numeric columns

Value-at-risk (VaR) data, specified as a NumRows-by-NumVaRs numeric array, NumRows-by-NumVaRs table, or a NumRows-by-NumVaRs timetable with numeric columns. The VaRData input argument sets the VaRData on page 5-0 property.

Negative VaRData values are allowed. However negative VaR values indicate a highly profitable portfolio that cannot lose money at the given VaR confidence level. The worst-case scenario at the given confidence level is still a profit.

---

**Note** VaRData must be in the same units as PortfolioData and ESData. There are no validations in the esbacktestbysim object regarding the units of portfolio, VaR, and ES data. VaRData, PortfolioData, and ESData can be expressed as returns or as profits and losses.

---

Data Types: double | table | timetable

#### **ESData — Expected shortfall data**

NumRows-by-NumVaRs positive numeric array | NumRows-by-NumVaRs table with positive numeric columns | NumRows-by-NumVaRs timetable with positive numeric columns

Expected shortfall data, specified as a NumRows-by-NumVaRs positive numeric array, NumRows-by-NumVaRs table, or NumRows-by-NumVaRs timetable with positive numeric columns containing ES data. The ESData input argument sets the ESData on page 5-0 property.

---

**Note** ESData data must be in the same units as PortfolioData and VaRData. There are no validations in the esbacktestbysim object regarding the units of portfolio, VaR, and ES data. ESData, PortfolioData, and VaRData can be expressed as returns or as profits and losses.

---

Data Types: double | table | timetable

#### **DistributionName — Distribution name**

string with values normal and t

Distribution name, specified as a string with a value of normal or t. The DistributionName input argument sets the 'Name' field of the Distribution on page 5-0 property.

Data Types: string

#### **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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: ebts =

```
esbacktestbysim(PortfolioData, VaRData, ESData, DistributionName, 'VaRID', 'TotalVaR', 'VaRLevel', .99)
```

#### **PortfolioID — User-defined ID**

character vector | string

User-defined ID for PortfolioData input, specified as the comma-separated pair consisting of 'PortfolioID' and a character vector or string. The PortfolioID name-value pair argument sets the PortfolioID on page 5-0 property.

If `PortfolioData` is a numeric array, the default value for `PortfolioID` is 'Portfolio'. If `PortfolioData` is a table, `PortfolioID` is set to the corresponding variable name in the table, by default.

Data Types: char | string

### **VaRID — VaR identifier**

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, cell array of character vectors, string, or string array. Multiple VaRIDs are specified using a 1-by-NumVaRs (or NumVaRs-by-1) cell array of character vectors, or a string array with user-defined IDs for the `VaRData` columns. A single VaRID identifies a `VaRData` column and the corresponding `ESData` column. The VaRID name-value pair argument sets the VaRID on page 5-0 property.

If `NumVaRs` = 1, the default value for VaRID is 'VaR'. If `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 or numeric array with values between 0 and 1

VaR confidence level, specified as a scalar with the comma-separated pair consisting of 'VaRLevel' and a numeric value between 0 and 1 or a 1-by-NumVaRs (or NumVaRs-by-1) numeric array with a numeric value between 0 and 1. The VaRLevel name-value pair argument sets the VaRLevel on page 5-0 property.

Data Types: double

### **Mean — Means for normal distribution**

0 (default) | numeric | numeric array

Means for the normal distribution, specified as a comma-separated pair consisting of 'Mean' and a numeric value or a NumRows-by-1 numeric array. The Mean name-value pair argument sets the 'Mean' field of the `Distribution` on page 5-0 property.

---

**Note** You set the Mean name-value pair argument only when the `DistributionName` input argument is specified as `normal`.

---

Data Types: double

### **StandardDeviation — Standard deviation for normal distribution**

1 (default) | positive numeric | positive numeric array

Standard deviation for the normal distribution, specified as a comma-separated pair consisting of 'StandardDeviation' and a positive numeric value or a NumRows-by-1 array. The StandardDeviation name-value pair argument sets the 'StandardDeviation' field of the `Distribution` on page 5-0 property.

---

**Note** You set the StandardDeviation name-value pair argument only when the `DistributionName` input argument is specified as `normal`.

---

Data Types: double

**DegreesOfFreedom — Degrees of freedom for t distribution**

integer  $\geq 3$

Degrees of freedom for the t distribution, specified as a comma-separated pair consisting of 'DegreesOfFreedom' and an integer value  $\geq 3$ . The DegreesOfFreedom name-value pair argument sets the 'DegreesOfFreedom' field of the Distribution on page 5-0 property.

---

**Note** The DegreesOfFreedom name-value pair argument is only set when the DistributionName input argument is specified as t. A value for DegreesOfFreedom is *required* when the value of DistributionName is t.

---

Data Types: double

**Location — Location parameters for t distribution**

0 (default) | numeric | numeric array

Location parameters for the t distribution, specified as a comma-separated pair consisting of 'Location' and a numeric value or a NumRows-by-1 array. The Location name-value pair argument sets the 'Location' field of the Distribution on page 5-0 property.

---

**Note** The Location name-value pair argument is only set when the DistributionName input argument is specified as t.

---

Data Types: double

**Scale — Scale parameters for t distribution**

1 (default) | positive numeric

Scale parameters for the t distribution, specified as a comma-separated pair consisting of 'Scale' and a positive numeric value or a NumRows-by-1 array. The Scale name-value pair argument sets the 'Scale' field of the Distribution on page 5-0 property.

---

**Note** The Scale name-value pair argument is only set when the DistributionName input argument is specified as t.

---

Data Types: double

**Simulate — Indicates if simulation for statistical significance is run**

true (default) | values are true or false

Indicates if a simulation for statistical significance is run when you create an esbacktestbysim object, specified as a logical scalar with the comma-separated pair consisting of 'Simulate' and a value of true or false.

Data Types: logical

## Properties

### PortfolioData — Portfolio data for ES backtesting analysis

numeric array

Portfolio data for the ES backtesting analysis, specified as a NumRows-by-1 numeric array containing a copy of the portfolio data.

Data Types: double

### VaRData — VaR data for ES backtesting analysis

numeric array

VaR data for the ES backtesting analysis, specified as a NumRows-by-NumVaRs numeric array containing a copy of the VaR data.

Data Types: double

### ESData — Expected shortfall data

numeric array

Expected shortfall data for ES backtesting analysis, specified as a NumRows-by-NumVaRs numeric array containing a copy of the ESData.

Data Types: double

### Distribution — Distribution information

structure

Distribution information, including distribution name and the associated distribution parameters, specified as a structure.

For a normal distribution, the Distribution structure has fields 'Name' (set to normal), 'Mean', and 'StandardDeviation', with values set to the corresponding inputs.

For a t distribution, the Distribution structure has fields 'Name' (set to t), 'DegreesOfFreedom', 'Location', and 'Scale', with values set to the corresponding inputs.

Data Types: struct

### PortfolioID — Portfolio identifier

string

Portfolio identifier, specified as a string.

Data Types: string

### VaRID — VaR identifier

string | 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

### VaRLevel — VaR level

numeric array with values between 0 and 1

VaR level, specified as a 1-by-NumVaRs numeric array with values between 0 and 1 containing the VaR levels for the corresponding columns in VaRData.

Data Types: double

esbacktestbysim Property	Set or Modify Property from Command Line Using esbacktestbysim	Modify Property Using Dot Notation
PortfolioData	Yes	No
VaRData	Yes	No
ESData	Yes	No
Distribution	Yes	No
PortfolioID	Yes	Yes
VaRID	Yes	Yes
VaRLevel	Yes	Yes

## Object Functions

summary Basic expected shortfall (ES) report on failures and severity  
 runtests Run all expected shortfall backtests (ES) for esbacktestbysim object  
 conditional Conditional expected shortfall (ES) backtest by Acerbi and Szekely  
 unconditional Unconditional expected shortfall backtest by Acerbi and Szekely  
 quantile Quantile expected shortfall (ES) backtest by Acerbi and Szekely  
 simulate Simulate expected shortfall (ES) test statistics

## Examples

### Create esbacktestbysim Object and Run ES Backtests

esbacktestbysim takes in portfolio outcomes data, the corresponding value-at-risk (VaR) data, the expected shortfall (ES) data, and the Distribution information and returns an esbacktestbysim object.

Create an esbacktestbysim object and display the Distribution property.

```
load ESBacktestBySimData
rng('default'); % for reproducibility
ebts = esbacktestbysim>Returns,VaR,ES,"t",...
    'DegreesOfFreedom',10,...
    'Location',Mu,...
    'Scale',Sigma,...
    'PortfolioID',"S&P",...
    'VaRID',["t(10) 95%","t(10) 97.5%","t(10) 99%"],...
    'VaRLevel',VaRLevel)

ebts =
    esbacktestbysim with properties:
        PortfolioData: [1966x1 double]
        VaRData: [1966x3 double]
        ESData: [1966x3 double]
```

```
Distribution: [1x1 struct]
PortfolioID: "S&P"
  VaRID: ["t(10) 95%" "t(10) 97.5%" "t(10) 99%"]
  VaRLevel: [0.9500 0.9750 0.9900]
```

**ebts.Distribution**

```
ans = struct with fields:
    Name: "t"
    DegreesOfFreedom: 10
    Location: 0
    Scale: [1966x1 double]
```

**ebts**, the **esbacktestbysim** object, contains a copy of the given portfolio data (**PortfolioData** property), the given VaR data (**VaRData** property), the given ES data (**ESData**) property, and the given **Distribution** information. The object also contains all combinations of portfolio ID, VaR ID, and VaR level to be tested (**PortfolioID**, **VaRID**, and **VaRLevel** properties).

Run the tests using the **ebts** object.

```
TestResults = runtests(ebts)
```

```
TestResults=3x6 table
PortfolioID      VaRID      VaRLevel      Conditional      Unconditional      Quantile
-----
"S&P"           "t(10) 95%"      0.95           reject           accept           reject
"S&P"           "t(10) 97.5%"    0.975          reject           reject           reject
"S&P"           "t(10) 99%"      0.99           reject           reject           reject
```

Change the **PortfolioID** property using dot notation. For more information on creating an **esbacktestbysim** object, see **esbacktestbysim**.

```
ebts.PortfolioID = 'S&P, 1996-2003'
```

```
ebts =
  esbacktestbysim with properties:

  PortfolioData: [1966x1 double]
  VaRData: [1966x3 double]
  ESData: [1966x3 double]
  Distribution: [1x1 struct]
  PortfolioID: "S&P, 1996-2003"
  VaRID: ["t(10) 95%" "t(10) 97.5%" "t(10) 99%"]
  VaRLevel: [0.9500 0.9750 0.9900]
```

Run all tests using the updated **esbacktestbysim** object.

```
runtests(ebts)
```

```
ans=3x6 table
PortfolioID      VaRID      VaRLevel      Conditional      Unconditional      Quantile
-----
"S&P, 1996-2003"  "t(10) 95%"      0.95           reject           accept           reject
```

"S&P, 1996-2003"	"t(10) 97.5%"	0.975	reject	reject	reject
"S&P, 1996-2003"	"t(10) 99%"	0.99	reject	reject	reject

## References

[1] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.

[2] Basel Committee on Banking Supervision. *Minimum Capital Requirements for Market Risk*. January, 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

`conditional` | `esbacktest` | `esbacktestbyte` | `quantile` | `runtests` | `simulate` | `summary` | `table` | `timetable` | `unconditional` | `varbacktest`

## Topics

“Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

## Introduced in R2017b



## summary

Basic expected shortfall (ES) report on failures and severity

### Syntax

```
S = summary(ebts)
```

### Description

`S = summary(ebts)` returns a basic report on the given `esbacktestbysim` data, including the number of observations, number of failures, observed confidence level, and so on (see `S` for details).

### Examples

#### Generate an ES Summary Report

Create an `esbacktestbysim` object.

```
load ESBacktestBySimData
rng('default'); % for reproducibility
ebts = esbacktestbysim>Returns, VaR, ES, "t", ...
    'DegreesOfFreedom', 10, ...
    'Location', Mu, ...
    'Scale', Sigma, ...
    'PortfolioID', "S&P", ...
    'VaRID', ["t(10) 95%", "t(10) 97.5%", "t(10) 99%"], ...
    'VaRLevel', VaRLevel);
```

Generate the ES summary report.

```
S = summary(ebts)
```

```
S=3x11 table
    PortfolioID      VaRID      VaRLevel      ObservedLevel      ExpectedSeverity      ObservedSev
    _____      _____      _____      _____      _____      _____
    "S&P"           "t(10) 95%"      0.95          0.94812           1.3288              1.4515
    "S&P"           "t(10) 97.5%"   0.975         0.97202           1.2652              1.4134
    "S&P"           "t(10) 99%"     0.99          0.98627           1.2169              1.3947
```

### Input Arguments

#### **ebts** — `esbacktestbysim` object

object

`esbacktestbysim` (`ebts`) object, which contains a copy of the given data (the `PortfolioData`, `VarData`, `ESData`, and `Distribution` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an `esbacktestbysim` object, see `esbacktestbysim`.

## 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 the number of periods without failures divided by number of observations
- 'ExpectedSeverity' — Expected average severity ratio, that is, the average ratio of ES to VaR over the periods with VaR failures
- 'ObservedSeverity' — Observed average severity ratio, that is, the average ratio of loss to VaR over the periods with VaR failures
- '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 1 minus the VaR level
- 'Ratio' — Ratio of number of failures to expected number of failures
- 'Missing' — Number of periods with missing values removed from the sample

---

**Note** The 'ExpectedSeverity' and 'ObservedSeverity' ratios are undefined (NaN) when there are no VaR failures in the data.

---

### See Also

`conditional` | `esbacktestbyde` | `esbacktestbysim` | `quantile` | `runtests` | `simulate` | `unconditional`

### Topics

“Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

### Introduced in R2017b

## runtests

Run all expected shortfall backtests (ES) for `esbacktestbysim` object

### Syntax

```
TestResults = runtests(ebts)
TestResults = runtests(ebts,Name,Value)
```

### Description

`TestResults = runtests(ebts)` runs all the tests for the `esbacktestbysim` object. `runtests` reports only the final test result. For test details, such as  $p$ -values, run the individual tests:

- conditional
- unconditional
- quantile

`TestResults = runtests(ebts,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

### Examples

#### Run All ES Backtests

Create an `esbacktestbysim` object.

```
load ESBacktestBySimData
rng('default'); % for reproducibility
ebts = esbacktestbysim>Returns,VaR,ES,"t",...
    'DegreesOfFreedom',10,...
    'Location',Mu,...
    'Scale',Sigma,...
    'PortfolioID',"S&P",...
    'VaRID',"t(10) 95%","t(10) 97.5%","t(10) 99%",...
    'VaRLevel',VaRLevel);
```

Generate the `TestResults` report for all ES backtests.

```
TestResults = runtests(ebts,'TestLevel',0.99)
```

```
TestResults=3x6 table
    PortfolioID      VaRID      VaRLevel      Conditional      Unconditional      Quantile
    _____      _____      _____      _____      _____      _____
    "S&P"            "t(10) 95%"      0.95          reject           accept           reject
    "S&P"            "t(10) 97.5%"    0.975        reject           accept           reject
    "S&P"            "t(10) 99%"     0.99         reject           reject           reject
```

Generate the `TestResults` report for all ES backtests using the name-value argument for `'ShowDetails'` to display the test confidence level.

```
TestResults = runtests(ebts, 'TestLevel', 0.99, 'ShowDetails', true)
```

TestResults=3×7 table

PortfolioID	VaRID	VaRLevel	Conditional	Unconditional	Quantile	Test
"S&P"	"t(10) 95%"	0.95	reject	accept	reject	0.9
"S&P"	"t(10) 97.5%"	0.975	reject	accept	reject	0.9
"S&P"	"t(10) 99%"	0.99	reject	reject	reject	0.9

## Input Arguments

### ebts — esbacktestbysim object

object

esbacktestbysim (ebts) object, which contains a copy of the given data (the PortfolioData, VarData, ESData, and Distribution properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktestbysim object, see esbacktestbysim.

### 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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: TestResults = runtests(ebts, 'TestLevel', 0.99)

### TestLevel — Test confidence level

0.95 (default) | numeric value between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric value between 0 and 1.

Data Types: double

### ShowDetails — Indicates if the output displays a column showing the test confidence level

false (default) | scalar logical with a value of true or false

Indicates if the output displays a column showing the test confidence level, specified as the comma-separated pair consisting of 'ShowDetails' and a scalar logical value.

Data Types: logical

## 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
- 'Conditional' — Categorical array with categories 'accept' and 'reject' indicating the result of the conditional test
- 'Unconditional' — Categorical array with categories 'accept' and 'reject' indicating the result of the unconditional test
- 'Quantile' — Categorical array with categories 'accept' and 'reject' indicating the result of the quantile 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

`conditional` | `esbacktestbyte` | `esbacktestbysim` | `quantile` | `simulate` | `summary` | `unconditional`

## Topics

“Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

## Introduced in R2017b

## conditional

Conditional expected shortfall (ES) backtest by Acerbi and Szekely

### Syntax

```
TestResults = conditional(ebts)
[TestResults, SimTestStatistic] = conditional(ebts, Name, Value)
```

### Description

`TestResults = conditional(ebts)` runs the conditional ES backtest of Acerbi-Szekely (2014). The conditional test has two underlying tests, a preliminary Value-at-Risk (VaR) backtest that is specified using the name-value pair argument `VaRTest`, and the standalone conditional ES backtest. A 'reject' result on either underlying test produces a 'reject' result on the conditional test.

`[TestResults, SimTestStatistic] = conditional(ebts, Name, Value)` adds optional name-value pair arguments for `TestLevel` and `VaRTest`.

### Examples

#### Run an ES Conditional Test

Create an `esbacktestbysim` object.

```
load ESBBacktestBySimData
rng('default'); % for reproducibility
ebts = esbacktestbysim>Returns, VaR, ES, "t", ...
    'DegreesOfFreedom', 10, ...
    'Location', Mu, ...
    'Scale', Sigma, ...
    'PortfolioID', "S&P", ...
    'VaRID', ["t(10) 95%", "t(10) 97.5%", "t(10) 99%"], ...
    'VaRLevel', VaRLevel);
```

Generate the ES conditional test report.

```
TestResults = conditional(ebts)
```

TestResults=3x14 table

PortfolioID	VaRID	VaRLevel	Conditional	ConditionalOnly	PValue	Test
"S&P"	"t(10) 95%"	0.95	reject	reject	0	-0
"S&P"	"t(10) 97.5%"	0.975	reject	reject	0.001	-0
"S&P"	"t(10) 99%"	0.99	reject	reject	0.003	-0

## Input Arguments

### ebts — esbacktestbysim object

object

esbacktestbysim (ebts) object, which contains a copy of the given data (the `PortfolioData`, `VarData`, `ESData`, and `Distribution` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktestbysim object, see esbacktestbysim.

### 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `[TestResults, SimTestStatistic] = conditional(ebts, 'TestLevel', 0.99)`

### TestLevel — Test confidence level

0.95 (default) | numeric value between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric value between 0 and 1.

Data Types: double

### VaRTest — Indicator for VaR back test

'pof' (default) | character vector with a value of 'tl', 'bin', 'pof', 'tuff', 'cc', 'cci', 'tbf', or 'tbfi' | string array with a value of 'tl', 'bin', 'pof', 'tuff', 'cc', 'cci', 'tbf', or 'tbfi'

Indicator for VaR back test, specified as the comma-separated pair consisting of `'VaRTest'` and a character vector or string array with a value of 'tl', 'bin', 'pof', 'tuff', 'cc', 'cci', 'tbf', or 'tbfi'. For more information on these VaR backtests, see varbacktest.

---

**Note** The specified `VaRTest` is run using the same `TestLevel` value that is specified with the `TestLevel` name-value pair argument in the `conditional` function.

---

Data Types: char | string

## 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.

- 'Conditional' — Categorical array with categories 'accept' and 'reject' indicating the result of the conditional test. This result combines the outcome of the 'ConditionalOnly' column and the VaR test.
- 'ConditionalOnly' — Categorical array with categories 'accept' and 'reject' indicating the result of the standalone conditional test, independent of the VaR test outcome.
- 'PValue' — *P*-value of the standalone conditional test (for the 'ConditionalOnly' column).
- 'TestStatistic' — Conditional test statistic (for the 'ConditionalOnly' column).
- 'CriticalValue' — Critical value for the conditional test.
- 'VaRTest' — String array indicating the selected VaR test as specified by the VaRTest argument.
- 'VaRTestResult' — Categorical array with categories 'accept' and 'reject' indicating the result of the VaR test selected with the 'VaRTest' argument.
- 'VaRTestPValue' — *P*-value for the VaR backtest. If the traffic-light test (tl) is used, this is 1 minus the traffic-light test's 'Probability' column value.
- 'Observations' — Number of observations.
- 'Scenarios' — Number of scenarios simulated to get the *p*-values.
- 'TestLevel' — Test confidence level.

---

**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.

---

### **SimTestStatistic — Simulated values of test statistic**

numeric array

Simulated values of the test statistic, returned as a NumVaRs-by-NumScenarios numeric array.

## **More About**

### **Conditional Test by Acerbi and Szekely**

The conditional test is also known as the first Acerbi-Szekely test.

The conditional test statistic is based on the conditional relationship

$$ES_t = -E_t[X_t | X_t < -VaR_t]$$

where

$X_t$  is the portfolio outcome, that is the portfolio return or portfolio profit and loss for period  $t$ .

$VaR_t$  is the estimated VaR for period  $t$ .

$ES_t$  is the estimated expected shortfall for period  $t$ .

The number of failures is defined as

$$NumFailures = \sum_{t=1}^N I_t$$

where



$N$  is the number of periods in the test window ( $t = 1, \dots, N$ ).

$I_t$  is the VaR failure indicator on period  $t$  with a value of 1 if  $X_t < -\text{VaR}$ , and 0 otherwise.

The conditional test statistic is defined as:



The conditional test has two parts. A VaR backtest, specified by the `VaRTest` name-value pair argument, must be run for the number of failures (`NumFailures`), and a standalone conditional test is performed for the conditional test statistic  $Z_{\text{cond}}$ . The conditional test accepts the model only when both the VaR test and the standalone conditional test accept the model.

### Significance of the Test

Under the assumption that the distributional assumptions are correct, the expected value of the test statistic  $Z_{\text{cond}}$ , assuming at least one VaR failure, is 0.

This is expressed as:

$$E[Z_{\text{cond}} | \text{NumFailures} > 0] = 0$$

Negative values of the test statistic indicate risk underestimation. The conditional test is a one-sided test that rejects when there is evidence that the model underestimates risk (for technical details on the null and alternative hypotheses, see Acerbi-Szekely, 2014). The conditional test rejects the model when the  $p$ -value is less than 1 minus the test confidence level.

For more information on the steps to simulate the test statistics and the details for the computation of the  $p$ -values and critical values, see `simulate`.

### Edge Cases

The conditional test statistic is undefined (NaN) when there are no VaR failures in the data (`NumFailures = 0`).

The  $p$ -value is set to NaN in these cases, and test result is to 'accept', because there is no evidence of risk underestimation.

Likewise, the simulated conditional test statistic is undefined (NaN) for scenarios with no VaR failures. These scenarios are discarded for the estimation of the significance of the test. Under the assumption that the distributional assumptions are correct,  $E[Z_{\text{cond}} | \text{NumFailures} > 0] = 0$ , so the significance is computed over scenarios with at least one failure (`NumFailures > 0`). The number of scenarios reported by the `conditional` test function is the number of scenarios with at least one VaR failure. The number of scenarios reported can be smaller than the total number of scenarios simulated. The critical value is estimated over the scenarios with at least one VaR failure. If the simulated test statistic is NaN for all scenarios, the critical value is set to NaN. Scenarios with no failures are more likely as the expected number of failures  $Np_{\text{VaR}}$  gets smaller.

## References

[1] Acerbi, C. and Szekely, B. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.

## **See Also**

`bin` | `cc` | `cci` | `esbacktestbyde` | `esbacktestbysim` | `pof` | `quantile` | `runtests` | `simulate` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` | `unconditional`

## **Topics**

“Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

## **Introduced in R2017b**

# unconditional

Unconditional expected shortfall backtest by Acerbi and Szekely

## Syntax

```
TestResults = unconditional(ebts)
[TestResults,SimTestStatistic] = unconditional(ebts,Name,Value)
```

## Description

`TestResults = unconditional(ebts)` runs the unconditional expected shortfall (ES) backtest of Acerbi-Szekely (2014).

`[TestResults,SimTestStatistic] = unconditional(ebts,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

## Examples

### Run an ES Unconditional Test

Create an `esbacktestbysim` object.

```
load ESBBacktestBySimData
rng('default'); % for reproducibility
ebts = esbacktestbysim>Returns,VaR,ES,"t",...
    'DegreesOfFreedom',10,...
    'Location',Mu,...
    'Scale',Sigma,...
    'PortfolioID',"S&P",...
    'VaRID',["t(10) 95%","t(10) 97.5%","t(10) 99%"],...
    'VaRLevel',VaRLevel);
```

Generate the ES unconditional test report.

```
TestResults = unconditional(ebts)
```

```
TestResults=3x10 table
PortfolioID      VaRID      VaRLevel      Unconditional      PValue      TestStatistic      Criti
-----
"S&P"           "t(10) 95%"      0.95          accept             0.093       -0.13342          -0
"S&P"           "t(10) 97.5%"    0.975         reject             0.031       -0.25011          -0
"S&P"           "t(10) 99%"      0.99          reject             0.008       -0.57396          -0
```

## Input Arguments

**ebts** — `esbacktestbysim` object  
object

`esbacktestbysim` (`ebts`) object, contains a copy of the given data (the `PortfolioData`, `VarData`, `ESData`, and `Distribution` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an `esbacktestbysim` object, see `esbacktestbysim`.

### 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `[TestResults, SimTestStatistic] = unconditional(ebts, 'TestLevel', 0.99)`

### TestLevel — Test confidence level

0.95 (default) | numeric with values between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric value 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
- `'Unconditional'` — Categorical array with categories 'accept' and 'reject' that indicate the result of the unconditional test
- `'PValue'` — P-value of the unconditional test
- `'TestStatistic'` — Unconditional test statistic
- `'CriticalValue'` — Critical value for the unconditional test
- `'Observations'` — Number of observations
- `'Scenarios'` — Number of scenarios simulated to get the  $p$ -values
- `'TestLevel'` — Test confidence level

### SimTestStatistic — Simulated values of the test statistic

numeric array

Simulated values of the test statistic, returned as a `NumVaRs-by-NumScenarios` numeric array.

## More About

### Unconditional Test by Acerbi and Szekely

The unconditional test is also known as the second Acerbi-Szekely test.

The unconditional test is based on the unconditional relationship

$$ES_t = -E_t \left[ \frac{X_t I_t}{P_{VaR}} \right]$$

where

$X_t$  is the portfolio outcome, that is, the portfolio return or portfolio profit and loss for period  $t$ .

$P_{VaR}$  is the probability of VaR failure defined as 1-VaR level.

$ES_t$  is the estimated expected shortfall for period  $t$ .

$I_t$  is the VaR failure indicator on period  $t$  with a value of 1 if  $X_t < -VaR$ , and 0 otherwise.

The unconditional test statistic is defined as:



### Significance of the Test

Under the assumption that the distributional assumptions are correct, the expected value of the test statistic  $Z_{uncond}$  is 0.

This is expressed as

$$E[Z_{uncond}] = 0$$

Negative values of the test statistic indicate risk underestimation. The unconditional test is a one-sided test that rejects when there is evidence that the model underestimates risk (for technical details on the null and alternative hypotheses, see Acerbi-Szekely, 2014). The unconditional test rejects the model when the  $p$ -value is less than 1 minus the test confidence level.

For more information on the steps to simulate the test statistics and the details for the computation of the  $p$ -values and critical values, see `simulate`.

### Edge Cases

The unconditional test statistic takes a value of 1 when there are no VaR failures in the data or in a simulated scenario.

1 is also the maximum possible value for the test statistic. When the expected number of failures  $Np_{VaR}$  is small, the distribution of the unconditional test statistic has a discrete probability jump at  $Z_{uncond} = 1$ , and the probability that  $Z_{uncond} \leq 1$  is 1. The  $p$ -value is set to 1 in these cases, and the test result is to 'accept', because there is no evidence of risk underestimation. Scenarios with no failures are more likely as the expected number of failures  $Np_{VaR}$  gets smaller.

### References

[1] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.

### See Also

`conditional` | `esbacktestbyde` | `esbacktestbysim` | `quantile` | `runtests` | `simulate` | `summary`

**Topics**

“Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

**Introduced in R2017b**

# quantile

Quantile expected shortfall (ES) backtest by Acerbi and Szekely

## Syntax

```
TestResults = quantile(ebts)
[TestResults,SimTestStatistic] = quantile(ebts,Name,Value)
```

## Description

`TestResults = quantile(ebts)` runs the quantile ES backtest of Acerbi-Szekely (2014).

`[TestResults,SimTestStatistic] = quantile(ebts,Name,Value)` adds an optional name-value pair argument for `TestLevel`.

## Examples

### Run an ES Quantile Test

Create an `esbacktestbysim` object.

```
load ESBacktestBySimData
rng('default'); % for reproducibility
ebts = esbacktestbysim>Returns,VaR,ES,"t",...
    'DegreesOfFreedom',10,...
    'Location',Mu,...
    'Scale',Sigma,...
    'PortfolioID',"S&P",...
    'VaRID',"t(10) 95%","t(10) 97.5%","t(10) 99%",...
    'VaRLevel',VaRLevel);
```

Generate the ES quantile test report.

```
TestResults = quantile(ebts)
```

```
TestResults=3x10 table
    PortfolioID      VaRID      VaRLevel      Quantile      PValue      TestStatistic      CriticalVa
    _____      _____      _____      _____      _____      _____      _____
    "S&P"            "t(10) 95%"      0.95          reject         0.002        -0.10602          -0.05579
    "S&P"            "t(10) 97.5%"    0.975         reject         0             -0.15697          -0.07351
    "S&P"            "t(10) 99%"      0.99          reject         0             -0.26561          -0.10111
```

## Input Arguments

**ebts** — `esbacktestbysim` object  
object

`esbacktestbysim` (`ebts`) object, which contains a copy of the given data (the `PortfolioData`, `VarData`, `ESData`, and `Distribution` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an `esbacktestbysim` object, see `esbacktestbysim`.

### 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `[TestResults, SimTestStatistic] = quantile(ebts, 'TestLevel', 0.99)`

#### TestLevel — Test confidence level

0.95 (default) | numeric with values between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of `'TestLevel'` and a numeric value 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
- `'Quantile'` — Categorical array with categories `'accept'` and `'reject'` indicating the result of the quantile test
- `'PValue'` — *P*-value of the quantile test
- `'TestStatistic'` — Quantile test statistic
- `'CriticalValue'` — Critical value for the quantile test
- `'Observations'` — Number of observations
- `'Scenarios'` — Number of scenarios simulated to get the *p*-values
- `'TestLevel'` — Test confidence level

#### SimTestStatistic — Simulated values of test statistic

numeric array

Simulated values of the test statistic, returned as a `NumVaRs-by-NumScenarios` numeric array.



## More About

### Quantile Test by Acerbi and Szekely

The quantile test (also known as the third Acerbi-Szekely test) uses a sample estimator of the expected shortfall.

The expected shortfall for a sample  $Y_1, \dots, Y_N$  is:

$$\widehat{ES}(Y) = -\frac{1}{[Np_{VaR}]} \sum_{i=1}^{[Np_{VaR}]} Y_{[i]}$$


where

$N$  is the number of periods in the test window ( $t = 1, \dots, N$ ).

$p_{VaR}$  is the probability of VaR failure defined as  $1 - VaR$  level.

$Y_{[1]}, \dots, Y_{[N]}$  are the sorted sample values (from smallest to largest), and  $[Np_{VaR}]$  is the largest integer less than or equal to  $Np_{VaR}$ .

To compute the quantile test statistic, a sample of size  $N$  is created at each time  $t$  as follows. First, convert the portfolio outcomes to  $X_t$  to ranks  $U_1 = P_1(X_1), \dots, U_N = P_N(X_N)$  using the cumulative distribution function  $P_t$ . If the distribution assumptions are correct, the rank values  $U_1, \dots, U_N$  are uniformly distributed in the interval  $(0, 1)$ . Then at each time  $t$ :

- Invert the ranks  $U = (U_1, \dots, U_N)$  to get  $N$  quantiles  $P_t^{-1}(U) = (P_t^{-1}(U_1), \dots, P_t^{-1}(U_N))$ .
- Compute the sample estimator  $\widehat{ES}(P_t^{-1}(U))$ .
- Compute the expected value of the sample estimator 

where  $V = (V_1, \dots, V_N)$  is a sample of  $N$  independent uniform random variables in the interval  $(0, 1)$ . This value can be computed analytically.

Define the quantile test statistic as

$$Z_{quantile} = -\frac{1}{N} \sum_{t=1}^N \frac{\widehat{ES}(P_t^{-1}(U))}{E[\widehat{ES}(P_t^{-1}(V))]} + 1$$

The denominator inside the sum can be computed analytically as

$$E[\widehat{ES}(P_t^{-1}(V))] = -\frac{N}{[Np_{VaR}]} \int_0^1 I_{1-p}(N - [Np_{VaR}], [Np_{VaR}]) P_t^{-1}(p) dp$$

where  $I_x(z, w)$  is the regularized incomplete beta function. For more information, see `betainc`.

### Significance of the Test

Assuming that the distributional assumptions are correct, the expected value of the test statistic  $Z_{quantile}$  is  $0$ .

This is expressed as:

$$E[Z_{quantile}] = 0$$

Negative values of the test statistic indicate risk underestimation. The quantile test is a one-sided test that rejects the model when there is evidence that the model underestimates risk. (For technical details on the null and alternative hypotheses, see Acerbi-Szekely, 2014). The quantile test rejects the model when the  $p$ -value is less than 1 minus the test confidence level.

For more information on simulating the test statistics and computing the  $p$ -values and critical values, see `simulate`.

### Edge Cases

The quantile test statistic is well-defined when there are no VaR failures in the data.

However, when the expected number of failures  $N_{pVaR}$  is small, an adjustment is required. The sample estimator of the expected shortfall takes the average of the smallest  $N_{tail}$  observations in the sample, where  $N_{tail} = \lfloor N_{pVaR} \rfloor$ . If  $N_{pVaR} < 1$ , then  $N_{tail} = 0$ , the sample estimator of the expected shortfall becomes an empty sum, and the quantile test statistic is undefined.

To account for this, whenever  $N_{pVaR} < 1$ , the value of  $N_{tail}$  is set to 1. Thus, the sample estimator of the expected shortfall has a single term and is equal to the minimum value of the sample. With this adjustment, the quantile test statistic is then well-defined and the significance analysis is unchanged.

### References

[1] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.

### See Also

`conditional` | `esbacktestbyde` | `esbacktestbysim` | `runtests` | `simulate` | `summary` | `unconditional`

### Topics

“Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33  
 “Expected Shortfall Estimation and Backtesting”  
 “Overview of Expected Shortfall Backtesting” on page 2-21  
 “Comparison of ES Backtesting Methods” on page 2-26

### Introduced in R2017b

# simulate

Simulate expected shortfall (ES) test statistics

## Syntax

```
ebts = simulate(ebts)
ebts = simulate(ebts,Name,Value)
```

## Description

`ebts = simulate(ebts)` performs a simulation of ES test statistics. The `simulate` function simulates portfolio outcomes according to the distribution assumptions indicated in the `esbacktestbysim` object, and calculates all the supported test statistics under each scenario. The simulated test statistics are used to estimate the significance of the ES backtests.

`ebts = simulate(ebts,Name,Value)` adds optional name-value pair arguments.

## Examples

### Simulate ES Test Statistics

Create an `esbacktestbysim` object and run a simulation of 1000 scenarios.

```
load ESBacktestBySimData
rng('default'); % for reproducibility
ebts = esbacktestbysim>Returns,VaR,ES,"t",...
    'DegreesOfFreedom',10,...
    'Location',Mu,...
    'Scale',Sigma,...
    'PortfolioID',"S&P",...
    'VaRID',"t(10) 95%","t(10) 97.5%","t(10) 99%",...
    'VaRLevel',VaRLevel);
```

The unconditional test reports 1000 scenarios (see the `Scenarios` column in the report).

```
unconditional(ebts)
```

```
ans=3x10 table
PortfolioID      VaRID      VaRLevel      Unconditional      PValue      TestStatistic      Criti
```

PortfolioID	VaRID	VaRLevel	Unconditional	PValue	TestStatistic	Criti
"S&P"	"t(10) 95%"	0.95	accept	0.093	-0.13342	-0
"S&P"	"t(10) 97.5%"	0.975	reject	0.031	-0.25011	-
"S&P"	"t(10) 99%"	0.99	reject	0.008	-0.57396	-0

Run a second simulation with 5000 scenarios using the `simulate` function. Rerun the unconditional test using the updated `esbacktestbysim` object. Notice that the test now shows 5,000 scenarios along with updated *p*-values and critical values.

```
ebts = simulate(ebts, 'NumScenarios', 5000);
unconditional(ebts)
```

ans=3×10 table

PortfolioID	VaRID	VaRLevel	Unconditional	PValue	TestStatistic	Crit
"S&P"	"t(10) 95%"	0.95	accept	0.0984	-0.13342	-0
"S&P"	"t(10) 97.5%"	0.975	reject	0.0456	-0.25011	-0
"S&P"	"t(10) 99%"	0.99	reject	0.0104	-0.57396	-0

## Input Arguments

### ebts — esbacktestbysim object

object

esbacktestbysim (ebts) object, which contains a copy of the given data (the `PortfolioData`, `VarData`, `ESData`, and `Distribution` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktestbysim object, see esbacktestbysim.

### 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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: ebts =

```
simulate(ebts, 'NumScenarios', 1000000, 'BlockSize', 10000, 'TestList', 'conditional')
```

### NumScenarios — Number of scenarios to simulate

1000 (default) | positive integer

Number of scenarios to simulate, specified using the comma-separated pair consisting of 'NumScenarios' and a positive integer.

Data Types: double

### BlockSize — Number of scenarios to simulate in single simulation block

1000 (default) | positive integer

Number of scenarios to simulate in a single simulation block, specified using the comma-separated pair consisting of 'BlockSize' and a positive integer.

Data Types: double

### TestList — Indicator for which test statistics to simulate

["conditional", "unconditional", "quantile"] (default) | character vector or string the value conditional, unconditional, or quantile

Indicator for which test statistics to simulate, specified as the comma-separated pair consisting of 'TestList' and a cell array of character vectors or a string array with the value conditional, unconditional, or quantile.

Data Types: cell | string

## Output Arguments

**ebts** — Updated `esbacktestbysim` object  
object

`esbacktestbysim` (`ebts`), returned as an updated object. After running `simulate`, the updated `esbacktestbysim` object stores the simulated test statistics, which are used to calculate  $p$ -values and generate test results.

For more information on an `esbacktestbysim` object, see `esbacktestbysim`.

## More About

### Simulation of Test Statistics and Significance of the Tests

The VaR and ES models assume that for each period  $t$ , the portfolio outcomes  $X_t$  have a cumulative probability distribution  $P_t$ .

Under the assumption that the distributions  $P_t$  are correct (the null hypothesis), test statistics are simulated by:

- Simulating  $M$  scenarios of  $N$  observations each, for example,  $X^s = (X_1^s, \dots, X_t^s, \dots, X_N^s)$ , with  $X_t^s \sim P_t$ ,  $t = 1, \dots, N$ , and  $s = 1, \dots, M$ .
- For each simulated scenario  $X^s$ , compute the test statistic of interest  $Z^s = Z(X^s)$ ,  $s = 1, \dots, M$ .
- The resulting  $M$  simulated test statistic values  $Z^1, \dots, Z^M$  from a distribution of the test statistic assuming the probability distributions  $P_t$  are correct.

The  $p$ -value is defined as the proportion of scenarios for which the simulated test statistic is smaller than the test statistic evaluated at the observed portfolio outcomes:  $Z^{obs} = Z(X_1, \dots, X_N)$ :

$$P_{value} = \frac{1}{M} \sum_{s=1}^M I(Z^s \leq Z^{obs})$$

where  $I(Z^s \leq Z^{obs})$  is an indicator function with a value of 1 if  $Z^s \leq Z^{obs}$ , and 0 otherwise. If  $P_{test}$  is 1 minus the test confidence level, the test result is to 'reject' if  $P_{value} < P_{test}$ .

The critical value is defined as the minimum simulated test statistic  $Z^{crit}$  with a  $p$ -value greater than or equal to  $P_{test}$ .

## References

[1] Acerbi, C., and B. Szekely. *Backtesting Expected Shortfall*. MSCI Inc. December, 2014.

## See Also

`conditional` | `esbacktestbyde` | `esbacktestbysim` | `quantile` | `runtests` | `summary` | `unconditional`

**Topics**

“Expected Shortfall (ES) Backtesting Workflow Using Simulation” on page 2-33

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“Comparison of ES Backtesting Methods” on page 2-26

**Introduced in R2017b**

# 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

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 = 5×1

```
0.0638
0.0008
0.0000
0.0026
0.0344
```

DD = 5×1

1.5237  
3.1679  
4.4298  
2.7916  
1.8196

A = 5×1  
10<sup>7</sup> ×

6.4210  
6.0109  
7.3063  
5.9906  
6.3231

Sa = 5×1

0.3010  
0.1753  
0.1699  
0.2263  
0.2511

## 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 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.

**More About****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 5-128.

**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**

[1] Zielinski, T. *Merton's and KMV Models In Credit Risk Management*.

[2] Löffler, G. and Posch, P.N. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2011.

[3] Kim, I.J., Byun, S.J, Hwang, S.Y. *An Iterative Method for Implementing Merton*.

[4] 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**

asrf | mertonByTimeSeries

## **Topics**

"Comparison of the Merton Model Single-Point Approach to the Time-Series Approach" on page 4-33

"Default Probability by Using the Merton Model for Structural Credit Risk" on page 1-10

## **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

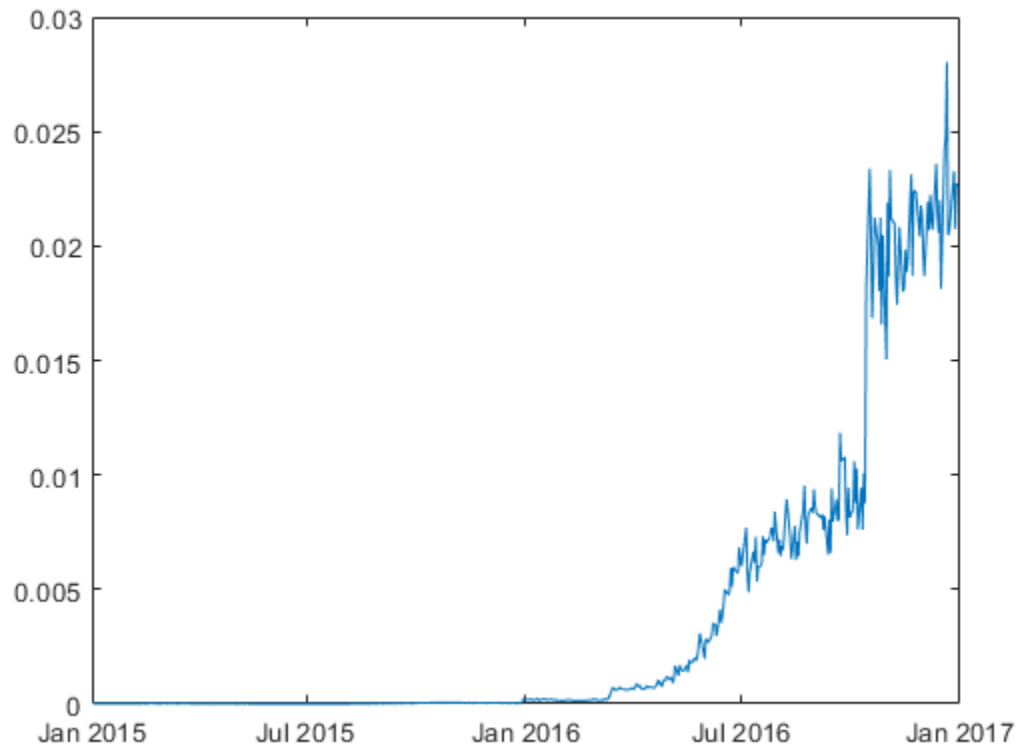
#### Compute Probability of Default Using the Time-Series Approach to the Merton Model

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)
```



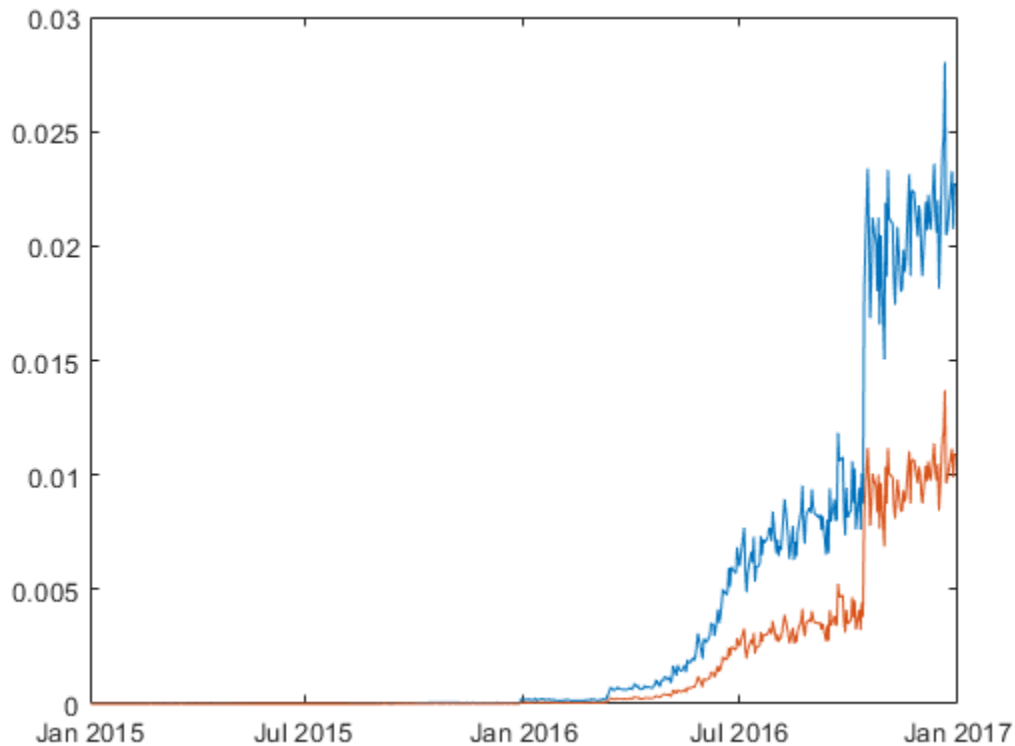
### Compute Probability of Default Using the Time-Series Approach to the Merton Model With Drift

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 PD0 (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)
```



## 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 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', 1e-5, 'NumPeriods', 12)
```

### 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.

**More About****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 5-134.

**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

- [1] Zielinski, T. *Merton's and KMV Models In Credit Risk Management*.
- [2] Loffler, G. and Posch, P.N. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2011.
- [3] Kim, I.J., Byun, S.J, Hwang, S.Y. *An Iterative Method for Implementing Merton*.
- [4] 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

asrf | mertonmodel

## Topics

"Comparison of the Merton Model Single-Point Approach to the Time-Series Approach" on page 4-33  
"Default Probability by Using the Merton Model for Structural Credit Risk" on page 1-10

## Introduced in R2017a

## varbacktest

Create `varbacktest` object to run suite of value-at-risk (VaR) backtests

### Description

The general workflow is:

- 1 Load or generate the data for the VaR backtesting analysis.
- 2 Create a `varbacktest` object. For more information, see “Create `varbacktest`” on page 5-136.
- 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:
  - `tl` — 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
  - `tbfi` — Time between failures independence

For more information, see “VaR Backtesting Workflow” on page 2-6.

### Creation

#### Syntax

```
vbt = varbacktest(PortfolioData, VaRData)
vbt = varbacktest( ____, Name, Value)
```

#### Description

`vbt = varbacktest(PortfolioData, VaRData)` creates a `varbacktest` (`vbt`) object using portfolio outcomes data and corresponding value-at-risk (VaR) data. The `vbt` object has the following properties:

- `PortfolioData` on page 5-0 — `NumRows-by-1` numeric array containing a copy of the `PortfolioData`
- `VaRData` on page 5-0 — `NumRows-by-NumVaRs` numeric array containing a copy of the `VaRData`
- `PortfolioID` on page 5-0 — String containing the `PortfolioID`

- `VaRID` on page 5-0 — 1-by-NumVaRs string vector containing the VaRIDs for the corresponding columns in `VaRData`
- `VaRLevel` on page 5-0 — 1-by-NumVaRs numeric array containing the VaRLevels for the corresponding columns in `VaRData`.

---

### Note

- The required input arguments for `PortfolioData` and `VaRData` must all be in the same units. These arguments can be expressed as returns or as profits and losses. There are no validations in the `varbacktest` object regarding the units of these arguments.
  - If there are missing values (NaNs) in the data for `PortfolioData` or `VaRData`, the row of data is discarded before applying the tests. Therefore, a different number of observations are reported for models with different number of missing values. The reported number of observations equals the original number of rows minus the number of missing values. To determine if there are discarded rows, use the 'Missing' column of the summary report.
- 

`vbt = varbacktest( ____, Name, Value)` sets Properties on page 5-139 using name-value pairs and any of the arguments in the previous syntax. For example, `vbt = varbacktest(PortfolioData, VaRData, 'PortfolioID', 'Equity100', 'VaRID', 'TotalVaR', 'VaRLevel', .99)`. You can specify multiple name-value pairs as optional name-value pair arguments.

### Input Arguments

#### PortfolioData — Portfolio outcomes data

NumRows-by-1 numeric array | NumRows-by-1 numeric columns table | NumRows-by-1 numeric columns timetable

Portfolio outcomes data, specified as a NumRows-by-1 numeric array, NumRows-by-1 table, or a NumRows-by-1 timetable with a numeric column containing portfolio outcomes data. `PortfolioData` input sets the `PortfolioData` on page 5-0 property.

---

**Note** The required input arguments for `PortfolioData` and `VaRData` must all be in the same units. These arguments can be expressed as returns or as profits and losses. There are no validations in the `varbacktest` object regarding the units of these arguments.

---

Data Types: double | table | timetable

#### VaRData — Value-at-risk (VaR) data

NumRows-by-NumVaRs numeric array | NumRows-by-NumVaRs table with numeric columns | NumRows-by-NumVaRs timetable with numeric columns

Value-at-risk (VaR) data, specified using a NumRows-by-NumVaRs numeric array, NumRows-by-NumVaRs table, or a NumRows-by-NumVaRs timetable with numeric columns. `VaRData` input sets the `VaRData` on page 5-0 property.

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

```
-PortfolioData > VaRData
```

For example, a VaR of 1,000,000 (positive) is violated whenever there is an outcome worse than a 1,000,000 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.

---

**Note** The required input arguments for `PortfolioData` and `VaRData` must all be in the same units. These arguments can be expressed as returns or as profits and losses. There are no validations in the `varbacktest` object regarding the units of these arguments.

---

Data Types: `double` | `table` | `timetable`

### 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 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', 'VaRLevel', .99)
```

### 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. The `PortfolioID` name-value pair argument sets the `PortfolioID` on page 5-0 property.

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. The `VaRID` name-value pair argument sets the `VaRID` on page 5-0 property.

If `NumVaRs = 1`, the default value for `VaRID` is `'VaR'`. If `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. The VaRLevel name-value pair argument sets the VaRLevel on page 5-0 property.

Data Types: double

**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

**VaRLevel** — 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

varbacktest Property	Set or Modify Property from Command Line Using varbacktest	Modify Property Using Dot Notation
PortfolioData	Yes	No

varbacktest Property	Set or Modify Property from Command Line Using varbacktest	Modify Property Using Dot Notation
VaRData	Yes	No
PortfolioID	Yes	Yes
VaRID	Yes	Yes
VaRLevel	Yes	Yes

## Object Functions

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
tbfI	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%

varbacktest 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,Normal95)

vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
    VaRData: [1043x1 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=1x11 table
  PortfolioID  VaRID  VaRLevel  TL  Bin  POF  TUFF  CC  CCI
```

"Portfolio"	"VaR"	0.95	green	accept	accept	accept	accept	accept
-------------	-------	------	-------	--------	--------	--------	--------	--------

Change the PortfolioID and VaRID properties using dot notation.

```
vbt.PortfolioID = 'S&P'
```

```
vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
      VaRData: [1043x1 double]
    PortfolioID: "S&P"
      VaRID: "VaR"
    VaRLevel: 0.9500
```

```
vbt.VaRID = 'Normal at 95%'
```

```
vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
      VaRData: [1043x1 double]
    PortfolioID: "S&P"
      VaRID: "Normal at 95%"
    VaRLevel: 0.9500
```

Run all tests using the updated varbacktest object.

```
runtests(vbt)
```

ans=1x11 table

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF	CC
"S&P"	"Normal at 95%"	0.95	green	accept	accept	accept	accept

### Run VaR Backtests for a Single VaR at 95%

Create a varbacktest object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)

vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
      VaRData: [1043x1 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=1x11 table
  PortfolioID  VaRID  VaRLevel  TL  Bin  POF  TUFF  CC  CCI
  _____  _____  _____  ____  ____  ____  ____  ____  ____
  "Portfolio"  "VaR"    0.95     green accept accept accept accept accept
```

Change the `PortfolioID` and `VaRID` properties using dot notation.

```
vbt.PortfolioID = 'S&P'
```

```
vbt =
  varbacktest with properties:

  PortfolioData: [1043x1 double]
  VaRData: [1043x1 double]
  PortfolioID: "S&P"
  VaRID: "VaR"
  VaRLevel: 0.9500
```

```
vbt.VaRID = 'Normal at 95%'
```

```
vbt =
  varbacktest with properties:

  PortfolioData: [1043x1 double]
  VaRData: [1043x1 double]
  PortfolioID: "S&P"
  VaRID: "Normal at 95%"
  VaRLevel: 0.9500
```

Run all tests using the updated `varbacktest` object.

```
runtests(vbt)

ans=1x11 table
  PortfolioID  VaRID  VaRLevel  TL  Bin  POF  TUFF  CC
  _____  _____  _____  ____  ____  ____  ____  ____
  "S&P"    "Normal at 95%"  0.95     green accept 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=6x10 table

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Failures	Exp
"Equity"	"Normal95"	0.95	0.94535	1043	57	57
"Equity"	"Normal99"	0.99	0.9837	1043	17	10
"Equity"	"Historical95"	0.95	0.94343	1043	59	57
"Equity"	"Historical99"	0.99	0.98849	1043	12	10
"Equity"	"EWMA95"	0.95	0.94343	1043	59	57
"Equity"	"EWMA99"	0.99	0.97891	1043	22	10

Run all tests using the `varbacktest` object.

```
runtests(vbt)
```

ans=6x11 table

PortfolioID	VaRID	VaRLevel	TL	Bin	POF	TUFF	CC
"Equity"	"Normal95"	0.95	green	accept	accept	accept	accept
"Equity"	"Normal99"	0.99	yellow	reject	accept	accept	accept
"Equity"	"Historical95"	0.95	green	accept	accept	accept	accept
"Equity"	"Historical99"	0.99	green	accept	accept	accept	accept
"Equity"	"EWMA95"	0.95	green	accept	accept	accept	accept
"Equity"	"EWMA99"	0.99	yellow	reject	reject	accept	reject

Run the traffic light test (`tl`) using the `varbacktest` object.

```
tl(vbt)
```

ans=6x9 table

PortfolioID	VaRID	VaRLevel	TL	Probability	TypeI	Increase
"Equity"	"Normal95"	0.95	green	0.77913	0.26396	0
"Equity"	"Normal99"	0.99	yellow	0.97991	0.03686	0.26582
"Equity"	"Historical95"	0.95	green	0.85155	0.18232	0
"Equity"	"Historical99"	0.99	green	0.74996	0.35269	0
"Equity"	"EWMA95"	0.95	green	0.85155	0.18232	0
"Equity"	"EWMA99"	0.99	yellow	0.99952	0.0011122	0.43511

## Run VaR Backtests for Multiple Portfolios and Concatenate Results

Use `varbacktest` with table inputs and name-value pair arguments to create two `varbacktest` objects and run the concatenated summary report. `varbacktest` 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=4x10 table
  PortfolioID      VaRID      VaRLevel  ObservedLevel  Observations  Failures
  _____  _____  _____  _____  _____  _____
  "Equity"      "VaREquity95"      0.95      0.94343      1043      59
  "Equity"      "VaREquity99"      0.99      0.97891      1043      22
  "Derivatives"  "VaRDerivatives95"  0.95      0.95014      1043      52
  "Derivatives"  "VaRDerivatives99"  0.99      0.97028      1043      31
```

Run all the tests and concatenate the results.

```
[runtests(vbte); runtests(vbtD)]
```

```
ans=4x11 table
  PortfolioID      VaRID      VaRLevel  TL      Bin      POF      TUFF
  _____  _____  _____  _____  _____  _____  _____
  "Equity"      "VaREquity95"      0.95      green  accept  accept  accept
  "Equity"      "VaREquity99"      0.99      yellow reject  reject  accept
  "Derivatives"  "VaRDerivatives95"  0.95      green  accept  accept  accept
  "Derivatives"  "VaRDerivatives99"  0.99      red    reject  reject  accept
```

Run the pof test and concatenate the results.

```
[pof(vbte); pof(vbtD)]
```

```
ans=4x9 table
  PortfolioID      VaRID      VaRLevel  POF      LRatioPOF  PValuePOF  Obs
  _____  _____  _____  _____  _____  _____  _____
  "Equity"      "VaREquity95"      0.95      accept  0.91023    0.34005
  "Equity"      "VaREquity99"      0.99      reject  9.8298     0.0017171
  "Derivatives"  "VaRDerivatives95"  0.95      accept  0.00045457 0.98299
  "Derivatives"  "VaRDerivatives99"  0.99      reject  26.809     2.2457e-07
```

## References

- [1] 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, <https://www.bis.org/publ/bcbs22.htm>.

- [2] Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841-862.
- [3] Cogneau, Ph. "Backtesting Value-at-Risk: How Good is the Model?" *Intelligent Risk*, PRMIA, July, 2015.
- [4] Haas, M. "New Methods in Backtesting." *Financial Engineering*, Research Center Caesar, Bonn, 2001.
- [5] Jorion, Ph. *Financial Risk Manager Handbook*. 6th Edition. Wiley Finance, 2011.
- [6] Kupiec, P. "Techniques for Verifying the Accuracy of Risk Management Models." *Journal of Derivatives*. Vol. 3, 1995, pp. 73-84.
- [7] McNeil, A., Frey, R., and Embrechts, P. *Quantitative Risk Management*. Princeton University Press, 2005.
- [8] Nieppola, O. "Backtesting Value-at-Risk Models." Master's Thesis, Helsinki School of Economics, 2009.

## See Also

[bin](#) | [cc](#) | [cci](#) | [esbacktest](#) | [esbacktestbysim](#) | [pof](#) | [runtests](#) | [summary](#) | [table](#) | [tbf](#) | [tbfi](#) | [tl](#) | [tuff](#)

## Topics

"VaR Backtesting Workflow" on page 2-6  
"Value-at-Risk Estimation and Backtesting" on page 2-10  
"Overview of VaR Backtesting" on page 2-2  
"Comparison of ES Backtesting Methods" on page 2-26

## 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,Normal95)
```

```
vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
      VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
      VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the bin test results.

```
TestResults = bin(vbt)
```

```
TestResults=1x9 table
  PortfolioID  VaRID  VaRLevel  Bin  ZScoreBin  PValueBin  Observations  Failu
  _____  _____  _____  _____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.95      accept  0.68905   0.49079   1043          5
```

#### 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,...
    [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: [1043x1 double]
    VaRData: [1043x6 double]
    PortfolioID: "Equity"
    VaRID: [1x6 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=6x9 table

PortfolioID	VaRID	VaRLevel	Bin	ZScoreBin	PValueBin	Observations
"Equity"	"Normal95"	0.95	accept	0.68905	0.49079	1043
"Equity"	"Normal99"	0.99	reject	2.0446	0.040896	1043
"Equity"	"Historical95"	0.95	accept	0.9732	0.33045	1043
"Equity"	"Historical99"	0.99	accept	0.48858	0.62514	1043
"Equity"	"EWMA95"	0.95	accept	0.9732	0.33045	1043
"Equity"	"EWMA99"	0.99	reject	3.6006	0.0003175	1043

## Input Arguments

### vbt — varbacktest object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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 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.

---

## More About

### 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 tail probability of the `bin` test is the probability that a standard normal distribution exceeds the absolute value of the z-score

$$TailProbability = 1 - F(|ZScoreBin|)$$

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.

The  $p$ -value of the `bin` test is defined as two times the tail probability. This is because the binomial test is a two-sided test. If  $alpha$  is defined as 1 minus the test confidence level, the test rejects if the tail probability is less than one half of  $alpha$ , or equivalently if

$$PValueBin = 2 * TailProbability < alpha$$

## References

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

## See Also

`cc` | `cci` | `pof` | `runtests` | `summary` | `tbfi` | `tbl` | `tl` | `tuff` | `varbacktest`

## Topics

“VaR Backtesting Workflow” on page 2-6

“Value-at-Risk Estimation and Backtesting” on page 2-10

“Overview of VaR Backtesting” on page 2-2

“Binomial Test” on page 2-2

“Comparison of ES Backtesting Methods” on page 2-26

## 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,Normal95)
```

```
vbt =
  varbacktest with properties:
    PortfolioData: [1043x1 double]
    VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the cc test results.

```
TestResults = cc(vbt)
```

```
TestResults=1x19 table
  PortfolioID  VaRID  VaRLevel  CC  LRatioCC  PValueCC  POF  LRatioPOF
  _____  _____  _____  ____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.95     accept  0.72013  0.69763  accept  0.46147
```

#### 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,...
    [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: [1043x1 double]
    VaRData: [1043x6 double]
    PortfolioID: "Equity"
    VaRID: [1x6 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=6x19 table

PortfolioID	VaRID	VaRLevel	CC	LRatioCC	PValueCC	POF	LR
"Equity"	"Normal95"	0.95	accept	0.72013	0.69763	accept	0
"Equity"	"Normal99"	0.99	accept	4.0757	0.13031	reject	3
"Equity"	"Historical95"	0.95	accept	1.0487	0.59194	accept	0
"Equity"	"Historical99"	0.99	accept	0.5073	0.77597	accept	0
"Equity"	"EWMA95"	0.95	accept	0.95051	0.62173	accept	0
"Equity"	"EWMA99"	0.99	reject	10.779	0.0045645	reject	9

## Input Arguments

### vbt — varbacktest object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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 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.

---

## More About

### 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 2 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 2 degrees of freedom exceeds the likelihood ratio  $LRatioCC$ ,

$$PValueCC = 1 - F(LRatioCC)$$

where  $F$  is the cumulative distribution of a chi-square variable with 2 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 2 degrees of freedom.

## References

[1] Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841-862.

## See Also

`bin` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` | `varbacktest`

## Topics

"VaR Backtesting Workflow" on page 2-6  
 "Value-at-Risk Estimation and Backtesting" on page 2-10  
 "Overview of VaR Backtesting" on page 2-2  
 "Christoffersen's Interval Forecast Tests" on page 2-4  
 "Comparison of ES Backtesting Methods" on page 2-26

**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,Normal95)
```

```
vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
    VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the `cci` test results.

```
TestResults = cci(vbt)
```

```
TestResults=1x13 table
  PortfolioID  VaRID  VaRLevel  CCI  LRatioCCI  PValueCCI  Observations  Failure
  _____  _____  _____  _____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.95     accept  0.25866   0.61104   1043         5
```

#### 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: [1043x1 double]
    VaRData: [1043x6 double]
    PortfolioID: "Equity"
    VaRID: [1x6 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=6x13 table

PortfolioID	VaRID	VaRLevel	CCI	LRatioCCI	PValueCCI	Observations
"Equity"	"Normal95"	0.95	accept	0.25866	0.61104	1043
"Equity"	"Normal99"	0.99	accept	0.56393	0.45268	1043
"Equity"	"Historical95"	0.95	accept	0.13847	0.70981	1043
"Equity"	"Historical99"	0.99	accept	0.27962	0.59695	1043
"Equity"	"EWMA95"	0.95	accept	0.040277	0.84094	1043
"Equity"	"EWMA99"	0.99	accept	0.94909	0.32995	1043

## Input Arguments

### vbt — varbacktest object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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 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.

---

## More About

### 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

$$\begin{aligned} LRatioCCI &= -2 \log \left( \frac{(1 - p_{UC})^{N_{00} + N_{10}} p_{UC}^{N_{01} + N_{11}}}{(1 - p_{01})^{N_{00}} p_{01}^{N_{01}} (1 - p_{11})^{N_{10}} p_{11}^{N_{11}}} \right) \\ &= -2((N_{00} + N_{10}) \log(1 - p_{UC}) + (N_{01} + N_{11}) \log(p_{UC}) - N_{00} \log(1 - p_{01}) - N_{01} \log(p_{01}) - N_{10} \log(1 - p_{11}) - N_{11} \log(p_{11})) \end{aligned}$$

which is asymptotically distributed as a chi-square distribution with 1 degree of freedom.

The  $p$ -value of the CCI test is the probability that a chi-square distribution with 1 degree of freedom exceeds the likelihood ratio  $LRatioCCI$ ,

$$PValueCCI = 1 - F(LRatioCCI)$$

where  $F$  is the cumulative distribution of a chi-square variable with 1 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 1 degree of freedom.

If one or more of the quantities  $N_{00}$ ,  $N_{10}$ ,  $N_{01}$ , or  $N_{11}$  are zero, the likelihood ratio is handled differently. The likelihood ratio as defined above is composed of three likelihood functions of the form

$$L = (1 - p)^{n_1} \times p^{n_2}$$

For example, in the numerator of the likelihood ratio, there is a likelihood function of the form  $L$  with  $p = p_{UC}$ ,  $n_1 = N_{00} + N_{10}$ , and  $n_2 = N_{01} + N_{11}$ . There are two such likelihood functions in the denominator of the likelihood ratio.

It can be shown that whenever  $n_1 = 0$  or  $n_2 = 0$ , the likelihood function  $L$  is replaced by the constant value 1. Therefore, whenever  $N_{00}$ ,  $N_{10}$ ,  $N_{01}$ , or  $N_{11}$  is zero, replace the corresponding likelihood functions by 1 in the likelihood ratio, and the likelihood ratio is well-defined.

## References

[1] Christoffersen, P. "Evaluating Interval Forecasts." *International Economic Review*. Vol. 39, 1998, pp. 841-862.

## See Also

`bin` | `cc` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` | `varbacktest`

## Topics

"VaR Backtesting Workflow" on page 2-6

"Value-at-Risk Estimation and Backtesting" on page 2-10

"Overview of VaR Backtesting" on page 2-2

"Christoffersen's Interval Forecast Tests" on page 2-4

"Comparison of ES Backtesting Methods" on page 2-26

## 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,Normal95)
```

```
vbt =
  varbacktest with properties:
    PortfolioData: [1043x1 double]
    VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the `pof` test results.

```
TestResults = pof(vbt,'TestLevel',0.99)
```

```
TestResults=1x9 table
  PortfolioID  VaRID  VaRLevel  POF  LRatioPOF  PValuePOF  Observations  Failure
  _____  _____  _____  _____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.95     accept  0.46147   0.49694   1043          5
```

#### 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,...
  [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: [1043x1 double]
  VaRData: [1043x6 double]
  PortfolioID: "Equity"
  VaRID: [1x6 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=6x9 table

PortfolioID	VaRID	VaRLevel	POF	LRatioPOF	PValuePOF	Observations
"Equity"	"Normal95"	0.95	accept	0.46147	0.49694	1043
"Equity"	"Normal99"	0.99	reject	3.5118	0.060933	1043
"Equity"	"Historical95"	0.95	accept	0.91023	0.34005	1043
"Equity"	"Historical99"	0.99	accept	0.22768	0.63325	1043
"Equity"	"EWMA95"	0.95	accept	0.91023	0.34005	1043
"Equity"	"EWMA99"	0.99	reject	9.8298	0.0017171	1043

## Input Arguments

### vbt — varbacktest object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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 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.

---

## More About

### 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 1 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 1 degree of freedom exceeds the likelihood ratio  $L\text{RatioPOF}$

$$P\text{ValuePOF} = 1 - F(L\text{RatioPOF})$$

where  $F$  is the cumulative distribution of a chi-square variable with 1 degree of freedom.

The result of the test is to accept if

$$P\text{ValuePOF} < F(\text{TestLevel})$$

and reject otherwise, where  $F$  is the cumulative distribution of a chi-square variable with 1 degree of freedom.

## References

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

## See Also

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

## Topics

- "VaR Backtesting Workflow" on page 2-6
- "Value-at-Risk Estimation and Backtesting" on page 2-10
- "Overview of VaR Backtesting" on page 2-2
- "Kupiec's POF and TUFF Tests" on page 2-3
- "Comparison of ES Backtesting Methods" on page 2-26

## 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:

- `tl` — 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

`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,Normal95)

vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
    VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the `TestResults` report for all VaR backtests.

```
TestResults = runtests(vbt,'TestLevel',0.99)
```

```

TestResults=1x11 table
  PortfolioID  VaRID  VaRLevel  TL  Bin  POF  TUFF  CC  CCI
  -----
  "Portfolio"  "VaR"  0.95  green  accept  accept  accept  accept  accept

```

Generate the `TestResults` report for all VaR backtests using the name-value argument for `'ShowDetails'` to display the test confidence level.

```
TestResults = runtests(vbt, 'TestLevel', 0.99, "ShowDetails", true)
```

```

TestResults=1x12 table
  PortfolioID  VaRID  VaRLevel  TL  Bin  POF  TUFF  CC  CCI
  -----
  "Portfolio"  "VaR"  0.95  green  accept  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=6x11 table
  PortfolioID  VaRID  VaRLevel  TL  Bin  POF  TUFF  CC
  -----
  "Equity"  "Normal95"  0.95  green  accept  accept  accept  accept
  "Equity"  "Normal99"  0.99  yellow  reject  accept  accept  accept
  "Equity"  "Historical95"  0.95  green  accept  accept  accept  accept
  "Equity"  "Historical99"  0.99  green  accept  accept  accept  accept
  "Equity"  "EWMA95"  0.95  green  accept  accept  accept  accept
  "Equity"  "EWMA99"  0.99  yellow  reject  reject  accept  reject

```

## Input Arguments

### `vbt` — `varbacktest` object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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 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`

### ShowDetails — Indicates if the output displays a column showing the test confidence level

false (default) | scalar logical with a value of `true` or `false`

Indicates if the output displays a column showing the test confidence level, specified as the comma-separated pair consisting of `'ShowDetails'` and a scalar logical value.

Data Types: `logical`

## 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**

cc | cci | pof | summary | tbf | tbfi | tl | tuff | varbacktest

**Topics**

“VaR Backtesting Workflow” on page 2-6

“Value-at-Risk Estimation and Backtesting” on page 2-10

“Overview of VaR Backtesting” on page 2-2

“Comparison of ES Backtesting Methods” on page 2-26

**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,Normal95)
```

```
vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
      VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
      VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the summary report.

```
S = summary(vbt)
```

```
S=1x10 table
  PortfolioID  VaRID  VaRLevel  ObservedLevel  Observations  Failures  Expected
  _____  _____  _____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.95      0.94535      1043        57        52.15
```

#### 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=6x10 table

PortfolioID	VaRID	VaRLevel	ObservedLevel	Observations	Failures	Exp
"Equity"	"Normal95"	0.95	0.94535	1043	57	52
"Equity"	"Normal99"	0.99	0.9837	1043	17	10
"Equity"	"Historical95"	0.95	0.94343	1043	59	52
"Equity"	"Historical99"	0.99	0.98849	1043	12	10
"Equity"	"EWMA95"	0.95	0.94343	1043	59	52
"Equity"	"EWMA99"	0.99	0.97891	1043	22	10

## Input Arguments

### vbt — varbacktest object

object

varbacktest (vbt) 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 varbacktest 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**

cc | cci | pof | runtests | tbf | tbfi | tl | tuff | varbacktest

## **Topics**

“VaR Backtesting Workflow” on page 2-6

“Value-at-Risk Estimation and Backtesting” on page 2-10

“Overview of VaR Backtesting” on page 2-2

“Comparison of ES Backtesting Methods” on page 2-26

## **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,Normal95)
```

```
vbt =
  varbacktest with properties:
    PortfolioData: [1043x1 double]
    VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the `tbf` test results.

```
TestResults = tbf(vbt)
```

```
TestResults=1x20 table
  PortfolioID  VaRID  VaRLevel  TBF  LRatioTBF  PValueTBF  POF  LRatioPOF
  _____  _____  _____  _____  _____  _____  _____  _____
  "Portfolio"  "VaR"    0.95     reject  88.952    0.0055565  accept  0.46147
```

#### 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,...
    [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: [1043x1 double]
VaRData: [1043x6 double]
PortfolioID: "Equity"
VaRID: [1x6 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=6x20 table

PortfolioID	VaRID	VaRLevel	TBF	LRatioTBF	PValueTBF	POF	LR
"Equity"	"Normal95"	0.95	reject	88.952	0.0055565	accept	0
"Equity"	"Normal99"	0.99	reject	26.441	0.090095	reject	0
"Equity"	"Historical95"	0.95	reject	83.63	0.023609	accept	0
"Equity"	"Historical99"	0.99	accept	16.456	0.22539	accept	0
"Equity"	"EWMA95"	0.95	accept	72.545	0.12844	accept	0
"Equity"	"EWMA99"	0.99	reject	41.66	0.0099428	reject	0

## Input Arguments

### vbt — varbacktest object

object

varbacktest (vbt) 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 varbacktest 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 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 — tbf test results

table

tbf 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
- 'TBF' — Categorical array with categories `accept` and `reject` that indicate the result of the tbf test
- 'LRatioTBF' — Likelihood ratio of the tbf test
- 'PValueTBF' — P-value of the tbf 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
- '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 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.

---

## More About

### 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 (TBFI) 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 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

- [1] Haas, M. "New Methods in Backtesting." Financial Engineering, Research Center Caesar, Bonn, 2001.

## See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbfi` | `tl` | `tuff` | `varbacktest`

## Topics

"VaR Backtesting Workflow" on page 2-6

"Value-at-Risk Estimation and Backtesting" on page 2-10

"Overview of VaR Backtesting" on page 2-2

"Haas's Time Between Failures or Mixed Kupiec's Test" on page 2-4

"Comparison of ES Backtesting Methods" on page 2-26

## 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,Normal95)
```

```
vbt =
  varbacktest with properties:
    PortfolioData: [1043x1 double]
    VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the `tbfi` test results.

```
TestResults = tbfi(vbt)
```

```
TestResults=1x14 table
  PortfolioID  VaRID  VaRLevel  TBFI  LRatioTBFI  PValueTBFI  Observations  Fa
  _____  _____  _____  _____  _____  _____  _____  _____
  "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: [1043x1 double]
    VaRData: [1043x6 double]
    PortfolioID: "Equity"
    VaRID: [1x6 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=6x14 table

PortfolioID	VaRID	VaRLevel	TBFI	LRatioTBFI	PValueTBFI	Observati
"Equity"	"Normal95"	0.95	reject	88.491	0.0047475	1043
"Equity"	"Normal99"	0.99	accept	22.929	0.15157	1043
"Equity"	"Historical95"	0.95	reject	82.719	0.022513	1043
"Equity"	"Historical99"	0.99	accept	16.228	0.18101	1043
"Equity"	"EWMA95"	0.95	accept	71.635	0.12517	1043
"Equity"	"EWMA99"	0.99	reject	31.83	0.080339	1043

## Input Arguments

### vbt — varbacktest object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `TestResults = tbfi(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
- '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 `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.

---

## More About

### 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 `tbf` 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

$$\begin{aligned}
 LRatioTBFI_i = LRatioTUFF(n_i) &= -2 \sum_{i=1}^x \log \left( \frac{pVaR(1-pVaR)^{n_i-1}}{\left(\frac{1}{n_i}\right)\left(1-\frac{1}{n_i}\right)^{n_i-1}} \right) \\
 &= -2(\log(pVaR) + (n_i-1)\log(1-pVaR) + n_i\log(n_i) - (n_i-1)\log(n_i-1))
 \end{aligned}$$

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

- [1] Haas, M. "New Methods in Backtesting." Financial Engineering, Research Center Caesar, Bonn, 2001.

## See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbfi` | `tl` | `tuff` | `varbacktest`

## Topics

- "VaR Backtesting Workflow" on page 2-6
- "Value-at-Risk Estimation and Backtesting" on page 2-10
- "Overview of VaR Backtesting" on page 2-2
- "Haas's Time Between Failures or Mixed Kupiec's Test" on page 2-4
- "Comparison of ES Backtesting Methods" on page 2-26

**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,Normal95)
```

```
vbt =
  varbacktest with properties:

    PortfolioData: [1043x1 double]
      VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
      VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the `tl` test results.

```
TestResults = tl(vbt)
```

```
TestResults=1x9 table
  PortfolioID  VaRID  VaRLevel  TL  Probability  TypeI  Increase  Observati
```

PortfolioID	VaRID	VaRLevel	TL	Probability	TypeI	Increase	Observati
"Portfolio"	"VaR"	0.95	green	0.77913	0.26396	0	1043

#### 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: [1043x1 double]
    VaRData: [1043x6 double]
    PortfolioID: "Equity"
    VaRID: [1x6 string]
    VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]

```

Generate the `tl` test results.

```
TestResults = tl(vbt)
```

```
TestResults=6x9 table
  PortfolioID      VaRID      VaRLevel      TL      Probability      TypeI      Increase
  _____      _____      _____      _____      _____      _____      _____
  "Equity"         "Normal95"         0.95      green      0.77913          0.26396          0
  "Equity"         "Normal99"         0.99      yellow     0.97991          0.03686          0.26582
  "Equity"         "Historical95"     0.95      green      0.85155          0.18232          0
  "Equity"         "Historical99"     0.99      green      0.74996          0.35269          0
  "Equity"         "EWMA95"           0.95      green      0.85155          0.18232          0
  "Equity"         "EWMA99"           0.99      yellow     0.99952          0.0011122        0.43511

```

## Input Arguments

### **vbt** — varbacktest object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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

## More About

### 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 5-180.

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, and so on) 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 "References" on page 5-181) and is explained in the "Algorithms" on page 5-180 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  $p = 1 - VaRLevel$ . 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  $TypeI = TypeI(x|N, p) = 1 - F(X \geq x|N, p)$ .

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

$$Increase = Baseline \times \left( \frac{zAssumed}{zObserved} - 1 \right)$$

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 "References" on page 5-181). The `tl` function does not apply any ad-hoc adjustments.

## References

- [1] 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, <https://www.bis.org/publ/bcbs22.htm>.

## See Also

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

## Topics

- "VaR Backtesting Workflow" on page 2-6
- "Value-at-Risk Estimation and Backtesting" on page 2-10
- "Overview of VaR Backtesting" on page 2-2
- "Traffic Light Test" on page 2-3
- "Comparison of ES Backtesting Methods" on page 2-26

## 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,Normal95)
```

```
vbt =
  varbacktest with properties:
    PortfolioData: [1043x1 double]
    VaRData: [1043x1 double]
    PortfolioID: "Portfolio"
    VaRID: "VaR"
    VaRLevel: 0.9500
```

Generate the `tuff` test results.

```
TestResults = tuff(vbt)
```

```
TestResults=1x9 table
  PortfolioID  VaRID  VaRLevel  TUFF  LRatioTUFF  PValueTUFF  FirstFailure  Obs
-----
  "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,...
    [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: [1043x1 double]
    VaRData: [1043x6 double]
    PortfolioID: "Equity"
    VaRID: [1x6 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	FirstFail
"Equity"	"Normal95"	0.95	accept	1.7354	0.18773	58
"Equity"	"Normal99"	0.99	accept	0.36686	0.54472	173
"Equity"	"Historical95"	0.95	accept	1.5348	0.2154	55
"Equity"	"Historical99"	0.99	accept	0.36686	0.54472	173
"Equity"	"EWMA95"	0.95	accept	0.13304	0.7153	28
"Equity"	"EWMA99"	0.99	accept	0.14596	0.70243	143

## Input Arguments

### vbt — varbacktest object

object

`varbacktest` (`vbt`) 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 `varbacktest` 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 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.

---

## More About

### 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 1 degree of freedom.

The `p`-value of the `tuff` test is the probability that a chi-square distribution with 1 degree of freedom exceeds the likelihood ratio  $LRatioTUFF$

$$PValueTUFF = 1 - F(LRatioTUFF)$$

where  $F$  is the cumulative distribution of a chi-square variable with 1 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

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

## See Also

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

## Topics

- "VaR Backtesting Workflow" on page 2-6
- "Value-at-Risk Estimation and Backtesting" on page 2-10
- "Overview of VaR Backtesting" on page 2-2
- "Kupiec's POF and TUFF Tests" on page 2-3
- "Comparison of ES Backtesting Methods" on page 2-26

## Introduced in R2016b

## compactCreditScorecard

Create compactCreditScorecard object for a credit scorecard model

### Description

Build a compact credit scorecard model by creating a compactCreditScorecard object from an existing creditScorecard object.

After creating a compactCreditScorecard object, you can use the associated object functions to display points (displaypoints), calculate the probability of default (probdefault), or compute scores (score).

---

**Note** You cannot directly modify a compactCreditScorecard object. To change a compactCreditScorecard object, you must modify the existing creditScorecard object that you used to create the compactCreditScorecard object. You must then use compactCreditScorecard to create a new compactCreditScorecard object.

---

### Creation

#### Syntax

```
csc = compactCreditScorecard(sc)
```

#### Description

csc = compactCreditScorecard(sc) creates a compactCreditScorecard object from an existing creditScorecard. You can then use the compactCreditScorecard object with the displaypoints, score, and probdefault functions.

---

**Note** You cannot use a compactCreditScorecard object with the **Binning Explorer** app.

---

#### Input Arguments

##### sc — creditScorecard object

object

creditScorecard object, specified using an existing creditScorecard object.

---

**Note** To use a creditScorecard object for input, you must first process the object using the autobinning and fitmodel functions. Optionally, you can also use formatpoints for processing.

---

Data Types: object

## Properties

### PredictorVars — Names of predictor variables

cell array of character vectors

Names of the predictor variables used in the input `creditscorecard` object, returned as a cell array of character vectors. The `PredictorVars` property includes only the predictor variable names in the fitted `creditscorecard` object.

Data Types: `cell`

### NumericPredictors — Numeric predictors

cell array of character vectors

Numeric predictors in the input `creditscorecard` object, returned as a cell array of character vectors. The `NumericPredictors` property includes only the numeric predictors in the fitted `creditscorecard` object.

Data Types: `cell`

### CategoricalPredictors — Names of categorical predictors

cell array of character vectors

Names of categorical predictors used in the input `creditscorecard` object, returned as a cell array of character vectors. The `CategoricalPredictors` property includes only the categorical predictors in the fitted `creditscorecard` object.

Data Types: `cell`

### Description — User-defined description

character vector | string

User-defined description, returned as a character vector or string.

Data Types: `char` | `string`

## Object Functions

<code>displaypoints</code>	Return points per predictor per bin for a <code>compactCreditScorecard</code> object
<code>score</code>	Compute credit scores for given dataset for a <code>compactCreditScorecard</code> object
<code>probdefault</code>	Likelihood of default for given dataset for a <code>compactCreditScorecard</code> object
<code>validatemodel</code>	Validate quality of compact credit scorecard model

## Examples

### Create a compactCreditScorecard Object

To create a `compactCreditScorecard` object, first create a `creditscorecard` object using the `CreditCardData.mat` file to load the data (using a dataset from Refaat 2011).

```
load CreditCardData.mat
sc = creditscorecard(data)

sc =
    creditscorecard with properties:
```

```

    GoodLabel: 0
    ResponseVar: 'status'
    WeightsVar: ''
    VarNames: {1x11 cell}
    NumericPredictors: {1x7 cell}
    CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
    BinMissingData: 0
    IDVar: ''
    PredictorVars: {1x10 cell}
    Data: [1200x11 table]

```

Before creating a `compactCreditScorecard` object, you must use `autobinning` and `fitmodel` with the `creditscorecard` object.

```

sc = autobinning(sc);
sc = fitmodel(sc);

```

1. Adding CustIncome, Deviance = 1490.8527, Chi2Stat = 32.588614, PValue = 1.1387992e-08
2. Adding TmWBank, Deviance = 1467.1415, Chi2Stat = 23.711203, PValue = 1.1192909e-06
3. Adding AMBalance, Deviance = 1455.5715, Chi2Stat = 11.569967, PValue = 0.00067025601
4. Adding EmpStatus, Deviance = 1447.3451, Chi2Stat = 8.2264038, PValue = 0.0041285257
5. Adding CustAge, Deviance = 1441.994, Chi2Stat = 5.3511754, PValue = 0.020708306
6. Adding ResStatus, Deviance = 1437.8756, Chi2Stat = 4.118404, PValue = 0.042419078
7. Adding OtherCC, Deviance = 1433.707, Chi2Stat = 4.1686018, PValue = 0.041179769

Generalized linear regression model:

```

status ~ [Linear formula with 8 terms in 7 predictors]
Distribution = Binomial

```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	0.70239	0.064001	10.975	5.0538e-28
CustAge	0.60833	0.24932	2.44	0.014687
ResStatus	1.377	0.65272	2.1097	0.034888
EmpStatus	0.88565	0.293	3.0227	0.0025055
CustIncome	0.70164	0.21844	3.2121	0.0013179
TmWBank	1.1074	0.23271	4.7589	1.9464e-06
OtherCC	1.0883	0.52912	2.0569	0.039696
AMBalance	1.045	0.32214	3.2439	0.0011792

1200 observations, 1192 error degrees of freedom

Dispersion: 1

Chi<sup>2</sup>-statistic vs. constant model: 89.7, p-value = 1.4e-16

Use the `creditscorecard` object with `compactCreditScorecard` to create a `compactCreditScorecard` object.

```

csc = compactCreditScorecard(sc)

```

```

csc =
compactCreditScorecard with properties:

```

```

    Description: ''

```

```

    GoodLabel: 0
    ResponseVar: 'status'
    WeightsVar: ''
    NumericPredictors: {'CustAge' 'CustIncome' 'TmWBANK' 'AMBALANCE'}
    CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
    PredictorVars: {1x7 cell}

```

You can then use `displaypoints`, `score`, and `probdefault` with the `compactCreditScorecard` object.

## References

- [1] Anderson, R. *The Credit Scoring Toolkit*. Oxford University Press, 2007.
- [2] Refaat, M. *Data Preparation for Data Mining Using SAS*. Morgan Kaufmann, 2006.
- [3] Refaat, M. *Credit Risk Scorecards: Development and Implementation Using SAS*. lulu.com, 2011.

## See Also

### Functions

`displaypoints` | `probdefault` | `score` | `validatemodel`

### Apps

**Binning Explorer**

### Topics

“compactCreditScorecard Object Workflow”  
 “Case Study for a Credit Scorecard Analysis” (Financial Toolbox)  
 “Credit Scorecard Modeling Workflow” (Financial Toolbox)  
 “About Credit Scorecards” (Financial Toolbox)

### External Websites

Credit Risk Modeling with MATLAB (53 min 10 sec)

### Introduced in R2019a

## displaypoints

Return points per predictor per bin for a `compactCreditScorecard` object

### Syntax

```
PointsInfo = displaypoints(csc)
[PointsInfo,MinScore,MaxScore] = displaypoints(csc)
[PointsInfo,MinScore,MaxScore] = displaypoints( ____,Name,Value)
```

### Description

`PointsInfo = displaypoints(csc)` returns a table of points for all bins of all predictor variables used in the `compactCreditScorecard` object. The `PointsInfo` table displays information on the predictor name, bin labels, and the corresponding points per bin.

`[PointsInfo,MinScore,MaxScore] = displaypoints(csc)` returns a table of points for all bins of all predictor variables used in the `compactCreditScorecard` object. The `PointsInfo` table displays information on the predictor name, bin labels, and the corresponding points per bin and `displaypoints`. In addition, the optional `MinScore` and `MaxScore` values are returned.

`[PointsInfo,MinScore,MaxScore] = displaypoints( ____,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the previous syntax.

### Examples

#### Display Points for a compactCreditScorecard Object

To create a `compactCreditScorecard` object, first create a `creditscorecard` object using the `CreditCardData.mat` file to load the data (using a dataset from Refaat 2011).

```
load CreditCardData.mat
sc = creditscorecard(data)

sc =
    creditscorecard with properties:

        GoodLabel: 0
        ResponseVar: 'status'
        WeightsVar: ''
        VarNames: {1x11 cell}
        NumericPredictors: {1x7 cell}
        CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
        BinMissingData: 0
        IDVar: ''
        PredictorVars: {1x10 cell}
        Data: [1200x11 table]
```

Before creating a `compactCreditScorecard` object, you must use `autobinning` and `fitmodel` with the `creditscorecard` object.



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

1. Adding CustIncome, Deviance = 1490.8527, Chi2Stat = 32.588614, PValue = 1.1387992e-08
2. Adding TmWBank, Deviance = 1467.1415, Chi2Stat = 23.711203, PValue = 1.1192909e-06
3. Adding AMBalance, Deviance = 1455.5715, Chi2Stat = 11.569967, PValue = 0.00067025601
4. Adding EmpStatus, Deviance = 1447.3451, Chi2Stat = 8.2264038, PValue = 0.0041285257
5. Adding CustAge, Deviance = 1441.994, Chi2Stat = 5.3511754, PValue = 0.020708306
6. Adding ResStatus, Deviance = 1437.8756, Chi2Stat = 4.118404, PValue = 0.042419078
7. Adding OtherCC, Deviance = 1433.707, Chi2Stat = 4.1686018, PValue = 0.041179769

Generalized linear regression model:

```
status ~ [Linear formula with 8 terms in 7 predictors]
Distribution = Binomial
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	0.70239	0.064001	10.975	5.0538e-28
CustAge	0.60833	0.24932	2.44	0.014687
ResStatus	1.377	0.65272	2.1097	0.034888
EmpStatus	0.88565	0.293	3.0227	0.0025055
CustIncome	0.70164	0.21844	3.2121	0.0013179
TmWBank	1.1074	0.23271	4.7589	1.9464e-06
OtherCC	1.0883	0.52912	2.0569	0.039696
AMBalance	1.045	0.32214	3.2439	0.0011792

1200 observations, 1192 error degrees of freedom

Dispersion: 1

Chi<sup>2</sup>-statistic vs. constant model: 89.7, p-value = 1.4e-16

Use the `creditscorecard` object with `compactCreditScorecard` to create a `compactCreditScorecard` object.

```
csc = compactCreditScorecard(sc)
```

```
csc =
```

```
compactCreditScorecard with properties:
```

```

Description: ''
GoodLabel: 0
ResponseVar: 'status'
WeightsVar: ''
NumericPredictors: {'CustAge' 'CustIncome' 'TmWBank' 'AMBalance'}
CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
PredictorVars: {1x7 cell}
```

Then use `displaypoints` with the `compactCreditScorecard` object to return a table of points for all bins of all predictor variables used in the `compactCreditScorecard` object.

```
[PointsInfo,MinScore,MaxScore] = displaypoints(csc)
```

```
PointsInfo=37x3 table
```

Predictors	Bin	Points
_____	_____	_____

```

{'CustAge' } {'[-Inf,33)'} } -0.15894
{'CustAge' } {'[33,37)'} } -0.14036
{'CustAge' } {'[37,40)'} } -0.060323
{'CustAge' } {'[40,46)'} } 0.046408
{'CustAge' } {'[46,48)'} } 0.21445
{'CustAge' } {'[48,58)'} } 0.23039
{'CustAge' } {'[58,Inf]'} } 0.479
{'CustAge' } {'<missing>'} } NaN
{'ResStatus' } {'Tenant' } -0.031252
{'ResStatus' } {'Home Owner' } 0.12696
{'ResStatus' } {'Other' } 0.37641
{'ResStatus' } {'<missing>'} } NaN
{'EmpStatus' } {'Unknown' } -0.076317
{'EmpStatus' } {'Employed' } 0.31449
{'EmpStatus' } {'<missing>'} } NaN
{'CustIncome' } {'[-Inf,29000)'} } -0.45716
:

```

```
MinScore = -1.3100
```

```
MaxScore = 3.0726
```

`displaypoints` always displays a '<missing>' bin for each predictor. The value of the '<missing>' bin comes from the initial `creditscorecard` object, and the '<missing>' bin is set to NaN whenever the scorecard model has no information on how to assign points to missing data.

To configure the points for the '<missing>' bin, you must use the initial `creditscorecard` object. For predictors that have missing values in the training set, the points for the '<missing>' bin are estimated from the data if the `'BinMissingData'` name-value pair argument is set to `true` using `creditscorecard`. When the `'BinMissingData'` parameter is set to `false`, or when the data contains no missing values in the training set, use the `'Missing'` name-value pair argument in `formatpoints` to indicate how to assign points to the missing data. Then, rebuild the `compactCreditScorecard` object and rerun `displaypoints`. Here is an example of this workflow:

```

sc = formatpoints(sc,'Missing','minpoints');
csc = compactCreditScorecard(sc);
[PointsInfo,MinScore,MaxScore] = displaypoints(csc)

```

```
PointsInfo=37x3 table
```

Predictors	Bin	Points
{'CustAge' }	{'[-Inf,33)'} }	-0.15894
{'CustAge' }	{'[33,37)'} }	-0.14036
{'CustAge' }	{'[37,40)'} }	-0.060323
{'CustAge' }	{'[40,46)'} }	0.046408
{'CustAge' }	{'[46,48)'} }	0.21445
{'CustAge' }	{'[48,58)'} }	0.23039
{'CustAge' }	{'[58,Inf]'} }	0.479
{'CustAge' }	{'<missing>'} }	-0.15894
{'ResStatus' }	{'Tenant' }	-0.031252
{'ResStatus' }	{'Home Owner' }	0.12696
{'ResStatus' }	{'Other' }	0.37641
{'ResStatus' }	{'<missing>'} }	-0.031252
{'EmpStatus' }	{'Unknown' }	-0.076317
{'EmpStatus' }	{'Employed' }	0.31449
{'EmpStatus' }	{'<missing>'} }	-0.076317

```

{'CustIncome'}    {'[-Inf,29000)'}    -0.45716
:

```

```
MinScore = -1.3100
```

```
MaxScore = 3.0726
```

### Display Points for a compactCreditScorecard Object That Contains Missing Data

To create a compactCreditScorecard object, first create a creditScorecard object using the CreditCardData.mat file to load the data (using a dataset from Refaat 2011). Using the dataMissing dataset, set the 'BinMissingData' indicator to true.

```
load CreditCardData.mat
sc = creditScorecard(dataMissing, 'BinMissingData', true);
```

Before creating a compactCreditScorecard object, you must use autobinning and fitmodel with the creditScorecard object. First, use autobinning with the creditScorecard object.

```
sc = autobinning(sc);
```

The binning map or rules for categorical data are summarized in a "category grouping" table, returned as an optional output. By default, each category is placed in a separate bin. Here is the information for the predictor ResStatus.

```
[bi,cg] = bininfo(sc, 'ResStatus')
```

```
bi=5x6 table
```

Bin	Good	Bad	Odds	WOE	InfoValue
{'Tenant' }	296	161	1.8385	-0.095463	0.0035249
{'Home Owner'}	352	171	2.0585	0.017549	0.00013382
{'Other' }	128	52	2.4615	0.19637	0.0055808
{'<missing>' }	27	13	2.0769	0.026469	2.3248e-05
{'Totals' }	803	397	2.0227	NaN	0.0092627

```
cg=3x2 table
```

Category	BinNumber
{'Tenant' }	1
{'Home Owner'}	2
{'Other' }	3

To group categories 'Tenant' and 'Other', modify the category grouping table cg, so the bin number for 'Other' is the same as the bin number for 'Tenant'. Then use modifybins to update the creditScorecard object.

```
cg.BinNumber(3) = 2;
sc = modifybins(sc, 'ResStatus', 'Catg', cg);
```

Display the updated bin information using `bininfo`. Note that the bin labels has been updated and that the bin membership information is contained in the category grouping `cg`.

```
[bi,cg] = bininfo(sc, 'ResStatus')
```

```
bi=4x6 table
      Bin      Good      Bad      Odds      WOE      InfoValue
-----
{'Group1' }    296     161     1.8385   -0.095463   0.0035249
{'Group2' }    480     223     2.1525    0.062196   0.0022419
{'<missing>' }  27      13     2.0769    0.026469   2.3248e-05
{'Totals'  }   803     397     2.0227         NaN         0.00579
```

```
cg=3x2 table
      Category      BinNumber
-----
{'Tenant'   }          1
{'Home Owner'}         2
{'Other'    }          2
```

Use `formatpoints` with the 'Missing' name-value pair argument to indicate that missing data is assigned 'maxpoints'.

```
sc = formatpoints(sc, 'BasePoints', true, 'Missing', 'maxpoints', 'WorstAndBest', [300 800]);
```

Use `fitmodel` to fit the model.

```
sc = fitmodel(sc, 'VariableSelection', 'fullmodel', 'Display', 'Off');
```

Use the `creditscorecard` object with `compactCreditScorecard` to create a `compactCreditScorecard` object.

```
csc = compactCreditScorecard(sc)
```

```
csc =
compactCreditScorecard with properties:
    Description: ''
    GoodLabel: 0
    ResponseVar: 'status'
    WeightsVar: ''
    NumericPredictors: {1x7 cell}
    CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
    PredictorVars: {1x10 cell}
```

Then use `displaypoints` with the `compactCreditScorecard` object to return a table of points for all bins of all predictor variables used in the `compactCreditScorecard` object. By setting the `displaypoints` name-value pair argument for 'ShowCategoricalMembers' to true, all the members contained in each individual group are displayed.

```
[PointsInfo,MinScore,MaxScore] = displaypoints(csc, 'ShowCategoricalMembers', true)
```

```
PointsInfo=51x3 table
      Predictors      Bin      Points
```

{'BasePoints' }	{'BasePoints' }	535.25
{'CustID' }	{' [-Inf,121)' }	12.085
{'CustID' }	{' [121,241)' }	5.4738
{'CustID' }	{' [241,1081)' }	-1.4061
{'CustID' }	{' [1081,Inf]' }	-7.2217
{'CustID' }	{' <missing>' }	12.085
{'CustAge' }	{' [-Inf,33)' }	-25.973
{'CustAge' }	{' [33,37)' }	-22.67
{'CustAge' }	{' [37,40)' }	-17.122
{'CustAge' }	{' [40,46)' }	-2.8071
{'CustAge' }	{' [46,48)' }	9.5034
{'CustAge' }	{' [48,51)' }	10.913
{'CustAge' }	{' [51,58)' }	13.844
{'CustAge' }	{' [58,Inf]' }	37.541
{'CustAge' }	{' <missing>' }	-9.7271
{'TmAtAddress' }	{' [-Inf,23)' }	-9.3683
:		

MinScore = 300

MaxScore = 800.0000

## Input Arguments

### **csc** — Compact credit scorecard model

compactCreditScorecard object

Compact credit scorecard model, specified as a compactCreditScorecard object.

To create a compactCreditScorecard object, use compactCreditScorecard or compact from Financial Toolbox.

### **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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: [PointsInfo,MinScore,MaxScore] =  
displaypoints(csc,'ShowCategoricalMembers',true)

### **ShowCategoricalMembers** — Indicator for how to display bins labels of categories that were grouped together

false (default) | true or false

Indicator for how to display bins labels of categories that were grouped together, specified as the comma-separated pair consisting of 'ShowCategoricalMembers' and a logical scalar with a value of true or false.

By default, when 'ShowCategoricalMembers' is false, bin labels are displayed as Group1, Group2,...,Groupn, or if the bin labels were modified in creditscorecard, then the user-defined bin label names are displayed.

If 'ShowCategoricalMembers' is true, all the members contained in each individual group are displayed.

Data Types: `logical`

## Output Arguments

### PointsInfo — One row per bin, per predictor, with the corresponding points

table

One row per bin, per predictor, with the corresponding points, returned as a table. For example:

Predictors	Bin	Points
Predictor_1	Bin_11	Points_11
Predictor_1	Bin_12	Points_12
Predictor_1	Bin_13	Points_13
	...	...
Predictor_1	'<missing>'	NaN (Default)
Predictor_2	Bin_21	Points_21
Predictor_2	Bin_22	Points_22
Predictor_2	Bin_23	Points_23
	...	...
Predictor_2	'<missing>'	NaN (Default)
Predictor_j	Bin_ji	Points_ji
	...	...
Predictor_j	'<missing>'	NaN (Default)

`displaypoints` always displays a '<missing>' bin for each predictor. The value of the '<missing>' bin comes from the initial `creditscorecard` object, and the '<missing>' bin is set to NaN whenever the scorecard model has no information on how to assign points to missing data.

To configure the points for the '<missing>' bin, you must use the initial `creditscorecard` object. For predictors that have missing values in the training set, the points for the '<missing>' bin are estimated from the data if the 'BinMissingData' name-value pair argument for is set to `true` using `creditscorecard`. When the 'BinMissingData' parameter is set to `false`, or when the data contains no missing values in the training set, use the 'Missing' name-value pair argument in `formatpoints` to indicate how to assign points to the missing data. Then rebuild the `compactCreditScorecard` object and rerun `displaypoints`.

When base points are reported separately (see `formatpoints`), the first row of the returned `PointsInfo` table contains the base points.

### MinScore — Minimum possible total score

scalar

Minimum possible total score, returned as a scalar.

---

**Note** Minimum score is the lowest possible total score in the mathematical sense, independently of whether a low score means high risk or low risk.

---

### MaxScore — Maximum possible total score

scalar

Maximum possible total score, returned as a scalar.

---

**Note** Maximum score is the highest possible total score in the mathematical sense, independently of whether a high score means high risk or low risk.

---

## Algorithms

The points for predictor  $j$  and bin  $i$  are, by default, given by

$$\text{Points}_{ji} = (\text{Shift} + \text{Slope} * b_0) / p + \text{Slope} * (b_j * \text{WOE}_j(i))$$

where  $b_j$  is the model coefficient of predictor  $j$ ,  $p$  is the number of predictors in the model, and  $\text{WOE}_j(i)$  is the Weight of Evidence (WOE) value for the  $i$ -th bin corresponding to the  $j$ -th model predictor. `Shift` and `Slope` are scaling constants.

When the base points are reported separately (see the `formatpoints` name-value pair argument `BasePoints`), the base points are given by

$$\text{Base Points} = \text{Shift} + \text{Slope} * b_0,$$

and the points for the  $j$ -th predictor,  $i$ -th row are given by

$$\text{Points}_{ji} = \text{Slope} * (b_j * \text{WOE}_j(i)).$$

By default, the base points are not reported separately.

The minimum and maximum scores are:

$$\begin{aligned} \text{MinScore} &= \text{Shift} + \text{Slope} * b_0 + \min(\text{Slope} * b_1 * \text{WOE}_1) + \dots + \min(\text{Slope} * b_p * \text{WOE}_p), \\ \text{MaxScore} &= \text{Shift} + \text{Slope} * b_0 + \max(\text{Slope} * b_1 * \text{WOE}_1) + \dots + \max(\text{Slope} * b_p * \text{WOE}_p). \end{aligned}$$

Use `formatpoints` to control the way points are scaled, rounded, and whether the base points are reported separately. See `formatpoints` for more information on format parameters and for details and formulas on these formatting options.

## References

[1] Anderson, R. *The Credit Scoring Toolkit*. Oxford University Press, 2007.

[2] Refaat, M. *Credit Risk Scorecards: Development and Implementation Using SAS*. lulu.com, 2011.

## See Also

`compactCreditScorecard` | `probdefault` | `score` | `validatemodel`

## Topics

“compactCreditScorecard Object Workflow”

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

“Credit Scorecard Modeling with Missing Values” (Financial Toolbox)

“Credit Scorecard Modeling Workflow” (Financial Toolbox)

“About Credit Scorecards” (Financial Toolbox)

**Introduced in R2019a**



## probdefault

Likelihood of default for given dataset for a compactCreditScorecard object

### Syntax

```
pd = probdefault(csc,data)
```

### Description

pd = probdefault(csc,data) computes the probability of default for the compactCreditScorecard (csc) based on the data.

### Examples

#### Calculate the Probability of Default for a compactCreditScorecard Object with New Data

To create a compactCreditScorecard object, first create a creditScorecard object using the CreditCardData.mat file to load the data (using a dataset from Refaat 2011).

```
load CreditCardData.mat
sc = creditScorecard(data)
```

```
sc =
  creditScorecard with properties:
      GoodLabel: 0
      ResponseVar: 'status'
      WeightsVar: ''
      VarNames: {1x11 cell}
      NumericPredictors: {1x7 cell}
      CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
      BinMissingData: 0
      IDVar: ''
      PredictorVars: {1x10 cell}
      Data: [1200x11 table]
```

Before creating a compactCreditScorecard object, you must use autobinning and fitmodel with the creditScorecard object.

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

1. Adding CustIncome, Deviance = 1490.8527, Chi2Stat = 32.588614, PValue = 1.1387992e-08
2. Adding TmWBank, Deviance = 1467.1415, Chi2Stat = 23.711203, PValue = 1.1192909e-06
3. Adding AMBalance, Deviance = 1455.5715, Chi2Stat = 11.569967, PValue = 0.00067025601
4. Adding EmpStatus, Deviance = 1447.3451, Chi2Stat = 8.2264038, PValue = 0.0041285257
5. Adding CustAge, Deviance = 1441.994, Chi2Stat = 5.3511754, PValue = 0.020708306
6. Adding ResStatus, Deviance = 1437.8756, Chi2Stat = 4.118404, PValue = 0.042419078
7. Adding OtherCC, Deviance = 1433.707, Chi2Stat = 4.1686018, PValue = 0.041179769

```
Generalized linear regression model:
  status ~ [Linear formula with 8 terms in 7 predictors]
Distribution = Binomial
```

```
Estimated Coefficients:
              Estimate      SE      tStat      pValue
-----
(Intercept)  0.70239    0.064001  10.975  5.0538e-28
CustAge      0.60833    0.24932   2.44   0.014687
ResStatus    1.377      0.65272   2.1097  0.034888
EmpStatus    0.88565     0.293    3.0227  0.0025055
CustIncome   0.70164    0.21844   3.2121  0.0013179
TmWBank      1.1074     0.23271   4.7589  1.9464e-06
OtherCC      1.0883     0.52912   2.0569  0.039696
AMBalance    1.045     0.32214   3.2439  0.0011792
```

```
1200 observations, 1192 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 89.7, p-value = 1.4e-16
```

Use the `creditscorecard` object with `compactCreditScorecard` to create a `compactCreditScorecard` object.

```
csc = compactCreditScorecard(sc)
```

```
csc =
compactCreditScorecard with properties:

    Description: ''
    GoodLabel: 0
    ResponseVar: 'status'
    WeightsVar: ''
    NumericPredictors: {'CustAge' 'CustIncome' 'TmWBank' 'AMBalance'}
    CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
    PredictorVars: {1x7 cell}
```

Then use `probdefault` with the `compactCreditScorecard` object. For the purpose of illustration, suppose that a few rows from the original data are our "new" data. Use the `data` input argument in the `probdefault` function to obtain the probability of default using the `newdata`.

```
newdata = data(10:20,:);
pd = probdefault(csc,newdata)
```

```
pd = 11x1

    0.3047
    0.3418
    0.2237
    0.2793
    0.3615
    0.1653
    0.3799
    0.4055
    0.4269
    0.1915
```

⋮

## Input Arguments

### **csc** — Compact credit scorecard model

`compactCreditScorecard` object

Credit scorecard model, specified as a `compactCreditScorecard` object.

To create a `compactCreditScorecard` object, use `compactCreditScorecard` or `compact` from Financial Toolbox.

### **data** — Dataset to apply probability of default rules

table

Dataset to apply probability of default rules, specified as a MATLAB table, where each row corresponds to individual observations. The data must contain columns for each of the predictors in the `compactCreditScorecard` object.

Data Types: table

## Output Arguments

### **pd** — Probability of default

array

Probability of default, returned as a NumObs-by-1 numerical array of default probabilities.

## More About

### Default Probability

After the unscaled scores are computed (see “Algorithms for Computing and Scaling Scores” (Financial Toolbox)), the probability of the points being “Good” is represented by the following formula:

$$\text{ProbGood} = 1./(1 + \exp(-\text{UnscaledScores}))$$

Thus, the probability of default is

$$\text{pd} = 1 - \text{ProbGood}$$

## References

[1] Refaat, M. *Credit Risk Scorecards: Development and Implementation Using SAS*. lulu.com, 2011.

## See Also

`compactCreditScorecard` | `displaypoints` | `score` | `validatemodel`

### Topics

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

“Credit Scorecard Modeling with Missing Values” (Financial Toolbox)

“Credit Scorecard Modeling Workflow” (Financial Toolbox)

“About Credit Scorecards” (Financial Toolbox)

**Introduced in R2019a**

## score

Compute credit scores for given dataset for a `compactCreditScorecard` object

### Syntax

```
[Scores,Points] = score(csc,data)
```

### Description

`[Scores,Points] = score(csc,data)` computes the credit scores and points for the `compactCreditScorecard` object (`csc`) based on the data. Missing data translates into NaN values for the corresponding points.

### Examples

#### Obtain a Score for a compactCreditScorecard Object with New Data

To create a `compactCreditScorecard` object, first create a `creditscorecard` object using the `CreditCardData.mat` file to load the data (using a dataset from Refaat 2011).

```
load CreditCardData.mat
sc = creditscorecard(data)
```

```
sc =
  creditscorecard with properties:

      GoodLabel: 0
      ResponseVar: 'status'
      WeightsVar: ''
      VarNames: {'CustID' 'CustAge' 'TmAtAddress' 'ResStatus' 'EmpStatus' 'CustI
      NumericPredictors: {'CustID' 'CustAge' 'TmAtAddress' 'CustIncome' 'TmWBank' 'AMBalan
      CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
      BinMissingData: 0
      IDVar: ''
      PredictorVars: {'CustID' 'CustAge' 'TmAtAddress' 'ResStatus' 'EmpStatus' 'CustI
      Data: [1200x11 table]
```

Before creating a `compactCreditScorecard` object, you must use `autobinning` and `fitmodel` with the `creditscorecard` object.

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

1. Adding `CustIncome`, Deviance = 1490.8527, Chi2Stat = 32.588614, PValue = 1.1387992e-08
2. Adding `TmWBank`, Deviance = 1467.1415, Chi2Stat = 23.711203, PValue = 1.1192909e-06
3. Adding `AMBBalance`, Deviance = 1455.5715, Chi2Stat = 11.569967, PValue = 0.00067025601
4. Adding `EmpStatus`, Deviance = 1447.3451, Chi2Stat = 8.2264038, PValue = 0.0041285257
5. Adding `CustAge`, Deviance = 1441.994, Chi2Stat = 5.3511754, PValue = 0.020708306
6. Adding `ResStatus`, Deviance = 1437.8756, Chi2Stat = 4.118404, PValue = 0.042419078
7. Adding `OtherCC`, Deviance = 1433.707, Chi2Stat = 4.1686018, PValue = 0.041179769

```
Generalized linear regression model:
status ~ [Linear formula with 8 terms in 7 predictors]
Distribution = Binomial
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	0.70239	0.064001	10.975	5.0538e-28
CustAge	0.60833	0.24932	2.44	0.014687
ResStatus	1.377	0.65272	2.1097	0.034888
EmpStatus	0.88565	0.293	3.0227	0.0025055
CustIncome	0.70164	0.21844	3.2121	0.0013179
TmWBank	1.1074	0.23271	4.7589	1.9464e-06
OtherCC	1.0883	0.52912	2.0569	0.039696
AMBalance	1.045	0.32214	3.2439	0.0011792

```
1200 observations, 1192 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 89.7, p-value = 1.4e-16
```

Use the `creditscorecard` object with `compactCreditScorecard` to create a `compactCreditScorecard` object.

```
csc = compactCreditScorecard(sc)
```

```
csc =
```

```
compactCreditScorecard with properties:
```

```

Description: ''
NumericPredictors: {'CustAge' 'CustIncome' 'TmWBank' 'AMBalance'}
CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
PredictorVars: {'CustAge' 'ResStatus' 'EmpStatus' 'CustIncome' 'TmWBank' 'OtherCC'}
```

Then use `score` with the `compactCreditScorecard` object. For the purpose of illustration, suppose that a few rows from the original data are our "new" data. Use the `data` input argument in the `score` function to obtain the scores for the newdata.

```
newdata = data(10:20,:);
[Scores,Points] = score(csc,newdata)
```

```
Scores = 11x1
```

```

0.8252
0.6553
1.2443
0.9478
0.5690
1.6192
0.4899
0.3824
0.2945
1.4401
:
```

Points=11×7 table

CustAge	ResStatus	EmpStatus	CustIncome	TmWBank	OtherCC	AMBalance
0.23039	0.12696	-0.076317	0.43693	-0.033752	0.15842	-0.017472
0.23039	-0.031252	-0.076317	0.052329	-0.033752	0.15842	0.35551
0.23039	0.37641	-0.076317	0.24473	-0.044811	0.15842	0.35551
0.479	0.12696	-0.076317	0.43693	-0.18257	-0.19168	0.35551
0.046408	0.37641	-0.076317	0.092433	-0.033752	-0.19168	0.35551
0.21445	0.37641	0.31449	0.24473	-0.044811	0.15842	0.35551
-0.14036	0.12696	0.31449	0.081611	-0.033752	0.15842	-0.017472
-0.060323	-0.031252	0.31449	0.052329	-0.033752	0.15842	-0.017472
-0.15894	0.12696	0.31449	-0.45716	-0.044811	0.15842	0.35551
0.23039	0.12696	0.31449	0.43693	-0.18257	0.15842	0.35551
0.23039	0.37641	-0.076317	0.24473	-0.044811	0.15842	-0.064636

## Input Arguments

### **csc** — Compact credit scorecard model

compactCreditScorecard object

Compact credit scorecard model, specified as a compactCreditScorecard object.

To create a compactCreditScorecard object, use compactCreditScorecard or compact from Financial Toolbox.

### **data** — Dataset to be scored

table

Dataset to be scored, specified as a MATLAB table where each row corresponds to individual observations. The data must contain columns for each of the predictors in the compactCreditScorecard object.

## Output Arguments

### **Scores** — Scores for each observation

vector

Scores for each observation, returned as a vector.

### **Points** — Points per predictor for each observation

table

Points per predictor for each observation, returned as a table.

## Algorithms

The score of an individual  $i$  is given by the formula

$$\text{Score}(i) = \text{Shift} + \text{Slope} * (b0 + b1 * \text{WOE1}(i) + b2 * \text{WOE2}(i) + \dots + bp * \text{WOEp}(i))$$

where  $b_j$  is the coefficient of the  $j$ -th variable in the model, and  $WOE_j(i)$  is the Weight of Evidence (WOE) value for the  $i$ -th individual corresponding to the  $j$ -th model variable. `Shift` and `Slope` are scaling constants that can be controlled with `formatpoints`.

If the data for individual  $i$  is in the  $i$ -th row of a given dataset, to compute a score, the  $data(i,j)$  is binned using existing binning maps, and converted into a corresponding Weight of Evidence value  $WOE_j(i)$ . Using the model coefficients, the unscaled score is computed as

$$s = b_0 + b_1 * WOE_1(i) + \dots + b_p * WOE_p(i).$$

For simplicity, assume in the description above that the  $j$ -th variable in the model is the  $j$ -th column in the data input, although, in general, the order of variables in a given dataset does not have to match the order of variables in the model, and the dataset could have additional variables that are not used in the model.

The formatting options can be controlled using `formatpoints`.

## References

[1] Anderson, R. *The Credit Scoring Toolkit*. Oxford University Press, 2007.

[2] Refaat, M. *Credit Risk Scorecards: Development and Implementation Using SAS*. lulu.com, 2011.

## See Also

`compactCreditScorecard` | `displaypoints` | `probdefault` | `validatemodel`

## Topics

“compactCreditScorecard Object Workflow”

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

“Credit Scorecard Modeling with Missing Values” (Financial Toolbox)

“Credit Scorecard Modeling Workflow” (Financial Toolbox)

“About Credit Scorecards” (Financial Toolbox)

## Introduced in R2019a



# validatemodel

Validate quality of compact credit scorecard model

## Syntax

```
Stats = validatemodel(csc,data)
[Stats,T] = validatemodel(___,Name,Value)
[Stats,T,hf] = validatemodel(___,Name,Value)
```

## Description

`Stats = validatemodel(csc,data)` validates the quality of the `compactCreditScorecard` model for the data set specified using the argument `data`.

`[Stats,T] = validatemodel(___,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the previous syntax and returns the outputs `Stats` and `T`.

`[Stats,T,hf] = validatemodel(___,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the previous syntax and returns the outputs `Stats` and `T` and the figure handle `hf` to the CAP, ROC, and KS plots.

## Examples

### Validate a Compact Credit Scorecard Model

Compute model validation statistics for a compact credit scorecard model.

To create a `compactCreditScorecard` object, you must first develop a credit scorecard model using a `creditscorecard` object.

Create a `creditscorecard` object using the `CreditCardData.mat` file to load the data (using a dataset from Refaat 2011).

```
load CreditCardData
sc = creditscorecard(data, 'IDVar','CustID')

sc =
    creditscorecard with properties:

        GoodLabel: 0
        ResponseVar: 'status'
        WeightsVar: ''
        VarNames: {1x11 cell}
        NumericPredictors: {1x6 cell}
        CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
        BinMissingData: 0
        IDVar: 'CustID'
        PredictorVars: {1x9 cell}
        Data: [1200x11 table]
```

Perform automatic binning using the default options. By default, autobinning uses the Monotone algorithm.

```
sc = autobinning(sc);
```

Fit the model.

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

1. Adding CustIncome, Deviance = 1490.8527, Chi2Stat = 32.588614, PValue = 1.1387992e-08
2. Adding TmWBank, Deviance = 1467.1415, Chi2Stat = 23.711203, PValue = 1.1192909e-06
3. Adding AMBalance, Deviance = 1455.5715, Chi2Stat = 11.569967, PValue = 0.00067025601
4. Adding EmpStatus, Deviance = 1447.3451, Chi2Stat = 8.2264038, PValue = 0.0041285257
5. Adding CustAge, Deviance = 1441.994, Chi2Stat = 5.3511754, PValue = 0.020708306
6. Adding ResStatus, Deviance = 1437.8756, Chi2Stat = 4.118404, PValue = 0.042419078
7. Adding OtherCC, Deviance = 1433.707, Chi2Stat = 4.1686018, PValue = 0.041179769

Generalized linear regression model:

```
status ~ [Linear formula with 8 terms in 7 predictors]
Distribution = Binomial
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	0.70239	0.064001	10.975	5.0538e-28
CustAge	0.60833	0.24932	2.44	0.014687
ResStatus	1.377	0.65272	2.1097	0.034888
EmpStatus	0.88565	0.293	3.0227	0.0025055
CustIncome	0.70164	0.21844	3.2121	0.0013179
TmWBank	1.1074	0.23271	4.7589	1.9464e-06
OtherCC	1.0883	0.52912	2.0569	0.039696
AMBalance	1.045	0.32214	3.2439	0.0011792

1200 observations, 1192 error degrees of freedom

Dispersion: 1

Chi<sup>2</sup>-statistic vs. constant model: 89.7, p-value = 1.4e-16

Format the unscaled points.

```
sc = formatpoints(sc, 'PointsOddsAndPDO', [500,2,50]);
```

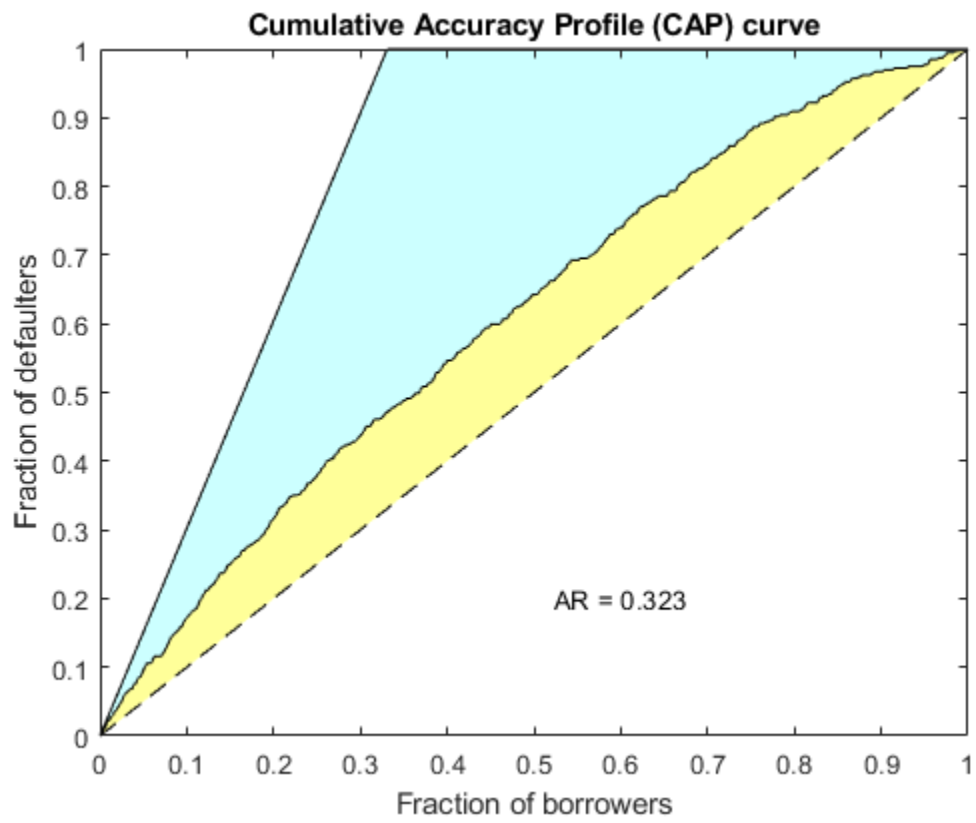
Convert the `creditscorecard` object into a `compactCreditScorecard` object. A `compactCreditScorecard` object is a lightweight version of a `creditscorecard` object that is used for deployment purposes.

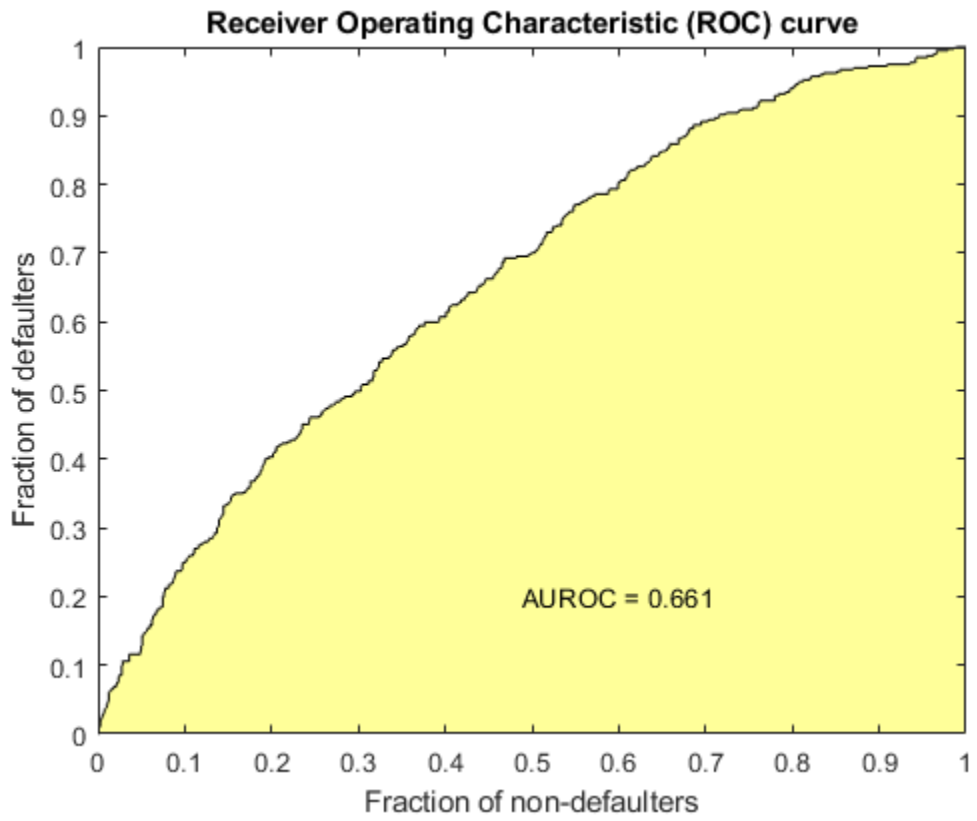
```
csc = compactCreditScorecard(sc);
```

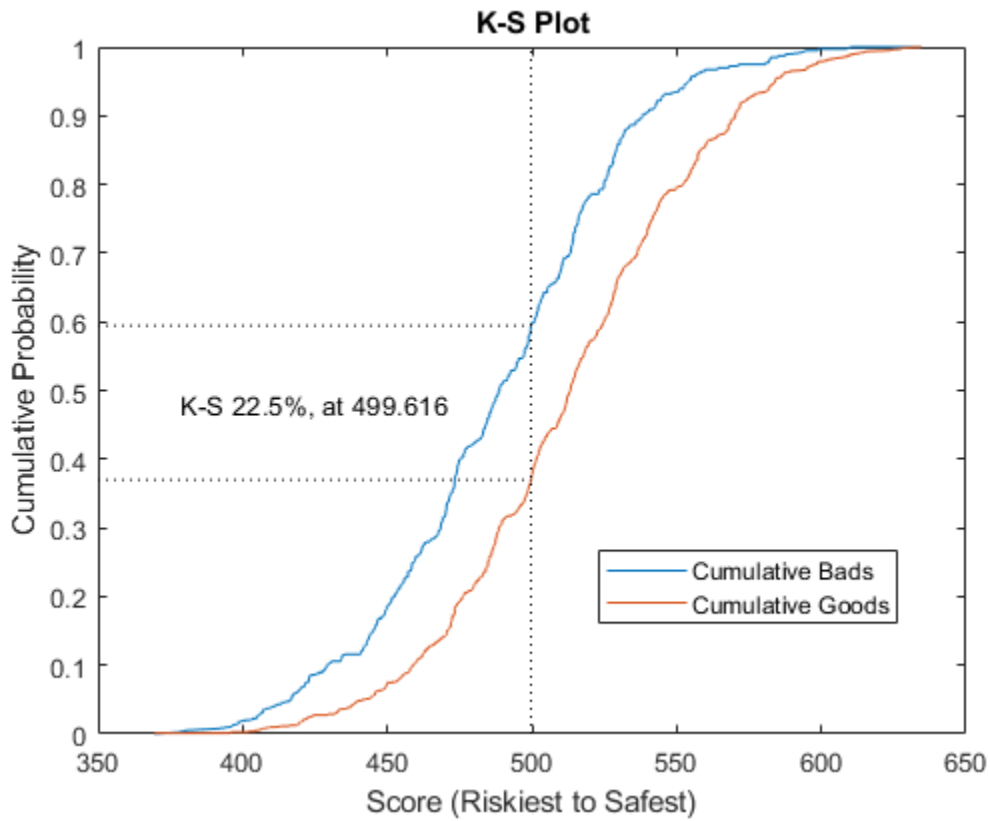
Validate the compact credit scorecard model by generating the CAP, ROC, and KS plots. This example uses the training data. However, you can use any validation data, as long as:

- The data has the same predictor names and predictor types as the data used to create the initial `creditscorecard` object.
- The data has a response column with the same name as the 'ResponseVar' property in the initial `creditscorecard` object.
- The data has a weights column (if weights were used to train the model) with the same name as 'WeightsVar' property in the initial `creditscorecard` object.

```
[Stats,T] = validatemodel(csc,data,'Plot',{'CAP','ROC','KS'});
```







disp(Stats)

Measure	Value
{'Accuracy Ratio' }	0.32258
{'Area under ROC curve' }	0.66129
{'KS statistic' }	0.2246
{'KS score' }	499.62

disp(T(1:15,:))

Scores	ProbDefault	TrueBads	FalseBads	TrueGoods	FalseGoods	Sensitivity
369.54	0.75313	0	1	802	397	0
378.19	0.73016	1	1	802	396	0.0025189
380.28	0.72444	2	1	802	395	0.0050378
391.49	0.69234	3	1	802	394	0.0075567
395.57	0.68017	4	1	802	393	0.010076
396.14	0.67846	4	2	801	393	0.010076
396.45	0.67752	5	2	801	392	0.012594
398.61	0.67094	6	2	801	391	0.015113
398.68	0.67072	7	2	801	390	0.017632
401.33	0.66255	8	2	801	389	0.020151
402.66	0.65842	8	3	800	389	0.020151
404.25	0.65346	9	3	800	388	0.02267
404.73	0.65193	9	4	799	388	0.02267

405.53	0.64941	11	4	799	386	0.027708
405.7	0.64887	11	5	798	386	0.027708

### Validate a Compact Credit Scorecard Model with Weights

Compute model validation statistics for a compact credit scorecard model with weights.

To create a `compactCreditScorecard` object, you must first develop a credit scorecard model using a `creditscorecard` object.

Use the `CreditCardData.mat` file to load the data (`dataWeights`) that contains a column (`RowWeights`) for the weights (using a dataset from Refaat 2011).

```
load CreditCardData
```

Create a `creditscorecard` object using the optional name-value pair argument `'WeightsVar'`.

```
sc = creditscorecard(dataWeights, 'IDVar', 'CustID', 'WeightsVar', 'RowWeights')
```

```
sc =
  creditscorecard with properties:

    GoodLabel: 0
    ResponseVar: 'status'
    WeightsVar: 'RowWeights'
    VarNames: {1x12 cell}
    NumericPredictors: {1x6 cell}
    CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
    BinMissingData: 0
    IDVar: 'CustID'
    PredictorVars: {1x9 cell}
    Data: [1200x12 table]
```

Perform automatic binning. By default, `autobinning` uses the Monotone algorithm.

```
sc = autobinning(sc)
```

```
sc =
  creditscorecard with properties:

    GoodLabel: 0
    ResponseVar: 'status'
    WeightsVar: 'RowWeights'
    VarNames: {1x12 cell}
    NumericPredictors: {1x6 cell}
    CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
    BinMissingData: 0
    IDVar: 'CustID'
    PredictorVars: {1x9 cell}
    Data: [1200x12 table]
```

Fit the model.

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

1. Adding CustIncome, Deviance = 764.3187, Chi2Stat = 15.81927, PValue = 6.968927e-05
2. Adding TmWBank, Deviance = 751.0215, Chi2Stat = 13.29726, PValue = 0.0002657942
3. Adding AMBalance, Deviance = 743.7581, Chi2Stat = 7.263384, PValue = 0.007037455

Generalized linear regression model:

```
logit(status) ~ 1 + CustIncome + TmWBank + AMBalance
Distribution = Binomial
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	0.70642	0.088702	7.964	1.6653e-15
CustIncome	1.0268	0.25758	3.9862	6.7132e-05
TmWBank	1.0973	0.31294	3.5063	0.0004543
AMBalance	1.0039	0.37576	2.6717	0.0075464

1200 observations, 1196 error degrees of freedom

Dispersion: 1

Chi<sup>2</sup>-statistic vs. constant model: 36.4, p-value = 6.22e-08

Format the unscaled points.

```
sc = formatpoints(sc, 'PointsOddsAndPDO', [500,2,50]);
```

Convert the `creditscorecard` object into a `compactCreditScorecard` object. A `compactCreditScorecard` object is a lightweight version of a `creditscorecard` object that is used for deployment purposes.

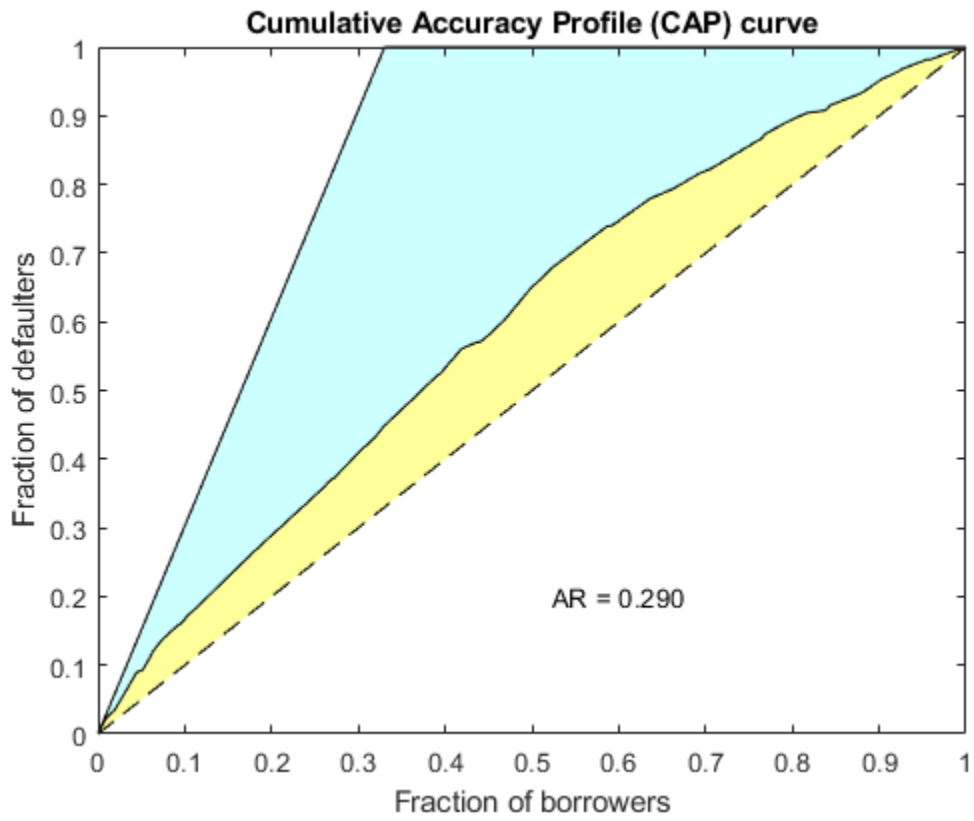
```
csc = compactCreditScorecard(sc);
```

Validate the compact credit scorecard model by generating the CAP, ROC, and KS plots. When you use the optional name-value pair argument `'WeightsVar'` to specify observation (sample) weights in the original `creditscorecard` object, the T table for `validatemodel` uses statistics, sums, and cumulative sums that are weighted counts.

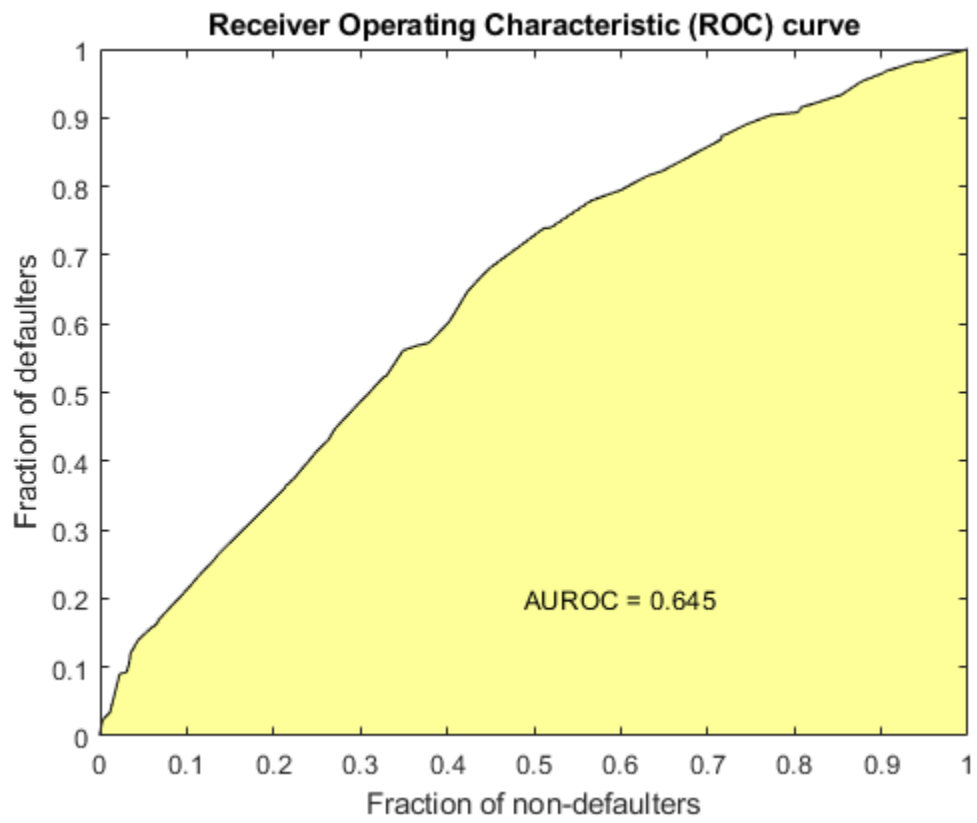
This example uses the training data (`dataWeights`). However, you can use any validation data, as long as:

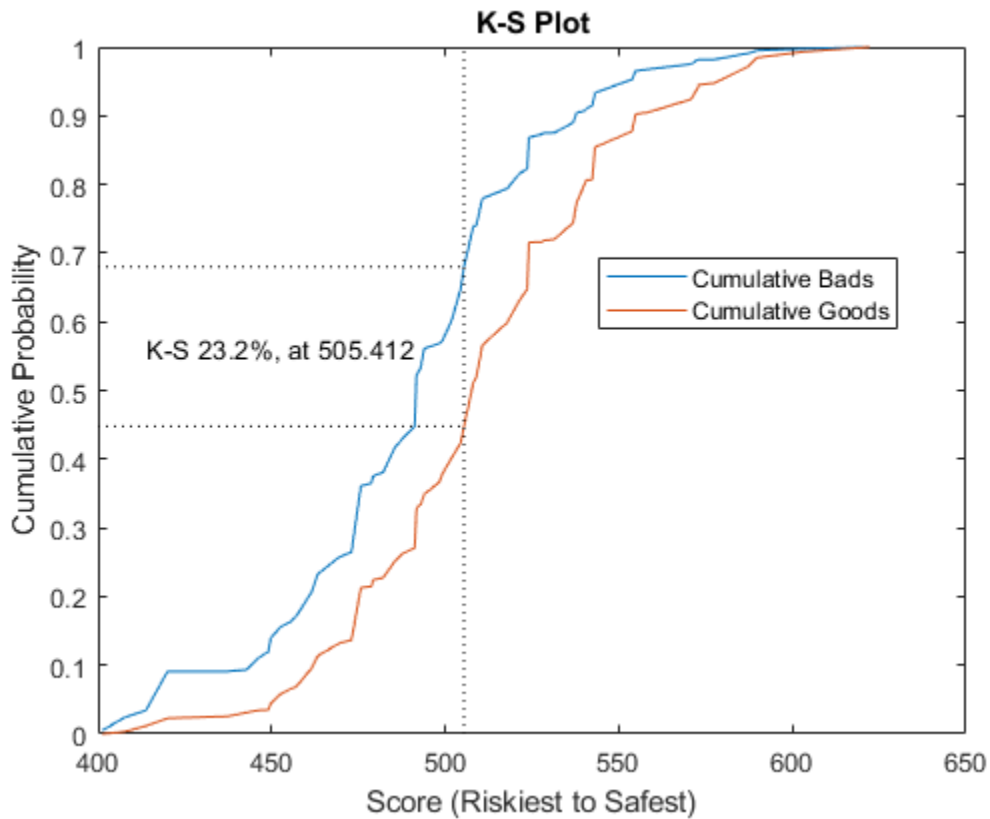
- The data has the same predictor names and predictor types as the data used to create the initial `creditscorecard` object.
- The data has a response column with the same name as the `'ResponseVar'` property in the initial `creditscorecard` object.
- The data has a weights column (if weights were used to train the model) with the same name as the `'WeightsVar'` property in the initial `creditscorecard` object.

```
[Stats,T] = validatemodel(csc,dataWeights,'Plot',{'CAP','ROC','KS'});
```









Stats

Stats=4x2 table

Measure	Value
{'Accuracy Ratio' }	0.28972
{'Area under ROC curve' }	0.64486
{'KS statistic' }	0.23215
{'KS score' }	505.41

T(1:10, :)

ans=10x9 table

Scores	ProbDefault	TrueBads	FalseBads	TrueGoods	FalseGoods	Sensitivity
401.34	0.66253	1.0788	0	411.95	201.95	0.0053135
407.59	0.64289	4.8363	1.2768	410.67	198.19	0.023821
413.79	0.62292	6.9469	4.6942	407.25	196.08	0.034216
420.04	0.60236	18.459	9.3899	402.56	184.57	0.090918
437.27	0.544	18.459	10.514	401.43	184.57	0.090918
442.83	0.52481	18.973	12.794	399.15	184.06	0.093448
446.19	0.51319	22.396	14.15	397.8	180.64	0.11031
449.08	0.50317	24.325	14.405	397.54	178.71	0.11981
449.73	0.50095	28.246	18.049	393.9	174.78	0.13912

452.44      0.49153      31.511      23.565      388.38      171.52      0.1552

### Validate a Compact Credit Score Card Model When Using the 'BinMissingData' Option

Compute model validation statistics and assign points for missing data when using the 'BinMissingData' option.

- Predictors in a `creditscorecard` object that have missing data in the training set have an explicit bin for `<missing>` with corresponding points in the final scorecard. These points are computed from the Weight-of-Evidence (WOE) value for the `<missing>` bin and the logistic model coefficients. For scoring purposes, these points are assigned to missing values and to out-of-range values, and after you convert the `creditscorecard` object to a `compactCreditScorecard` object, you can use the final score to compute model validation statistics with `validatemodel`.
- Predictors in a `creditscorecard` object with no missing data in the training set have no `<missing>` bin, so no WOE can be estimated from the training data. By default, the points for missing and out-of-range values are set to `NaN` resulting in a score of `NaN` when running `score`. For predictors in a `creditscorecard` object that have no explicit `<missing>` bin, use the name-value argument 'Missing' in `formatpoints` to specify how the function treats missing data for scoring purposes. After converting the `creditscorecard` object to a `compactCreditScorecard` object, you can use the final score to compute model validation statistics with `validatemodel`.

To create a `compactCreditScorecard` object, you must first develop a credit scorecard model using a `creditscorecard` object.

Create a `creditscorecard` object using the `CreditCardData.mat` file to load `dataMissing`, a table that contains missing values.

```
load CreditCardData.mat
head(dataMissing,5)
```

ans=5x11 table

CustID	CustAge	TmAtAddress	ResStatus	EmpStatus	CustIncome	TmWBank	Other
1	53	62	<undefined>	Unknown	50000	55	Ye
2	61	22	Home Owner	Employed	52000	25	Ye
3	47	30	Tenant	Employed	37000	61	No
4	NaN	75	Home Owner	Employed	53000	20	Ye
5	68	56	Home Owner	Employed	53000	14	Ye

Use `creditscorecard` with the name-value argument 'BinMissingData' set to true to bin the missing numeric or categorical data in a separate bin. Apply automatic binning.

```
sc = creditscorecard(dataMissing,'IDVar','CustID','BinMissingData',true);
sc = autobinning(sc);
```

```
disp(sc)
```

creditscorecard with properties:

GoodLabel: 0

```

ResponseVar: 'status'
WeightsVar: ''
VarNames: {1x11 cell}
NumericPredictors: {1x6 cell}
CategoricalPredictors: {'ResStatus' 'EmpStatus' 'OtherCC'}
BinMissingData: 1
IDVar: 'CustID'
PredictorVars: {1x9 cell}
Data: [1200x11 table]

```

To make any negative age or income information invalid or "out of range," set a minimum value of zero for 'CustAge' and 'CustIncome'. For scoring and probability-of-default computations, out-of-range values are given the same points as missing values.

```

sc = modifybins(sc, 'CustAge', 'MinValue', 0);
sc = modifybins(sc, 'CustIncome', 'MinValue', 0);

```

Display bin information for numeric data for 'CustAge' that includes missing data in a separate bin labelled <missing>.

```

bi = bininfo(sc, 'CustAge');
disp(bi)

```

Bin	Good	Bad	Odds	WOE	InfoValue
{ '[0,33)' }	69	52	1.3269	-0.42156	0.018993
{ '[33,37)' }	63	45	1.4	-0.36795	0.012839
{ '[37,40)' }	72	47	1.5319	-0.2779	0.0079824
{ '[40,46)' }	172	89	1.9326	-0.04556	0.0004549
{ '[46,48)' }	59	25	2.36	0.15424	0.0016199
{ '[48,51)' }	99	41	2.4146	0.17713	0.0035449
{ '[51,58)' }	157	62	2.5323	0.22469	0.0088407
{ '[58,Inf]' }	93	25	3.72	0.60931	0.032198
{ '<missing>' }	19	11	1.7273	-0.15787	0.00063885
{ 'Totals' }	803	397	2.0227	NaN	0.087112

Display bin information for categorical data for 'ResStatus' that includes missing data in a separate bin labelled <missing>.

```

bi = bininfo(sc, 'ResStatus');
disp(bi)

```

Bin	Good	Bad	Odds	WOE	InfoValue
{ 'Tenant' }	296	161	1.8385	-0.095463	0.0035249
{ 'Home Owner' }	352	171	2.0585	0.017549	0.00013382
{ 'Other' }	128	52	2.4615	0.19637	0.0055808
{ '<missing>' }	27	13	2.0769	0.026469	2.3248e-05
{ 'Totals' }	803	397	2.0227	NaN	0.0092627

For the 'CustAge' and 'ResStatus' predictors, the training data contains missing data (NaNs and <undefined> values). For missing data in these predictors, the binning process estimates WOE values of -0.15787 and 0.026469, respectively.

Because the training data contains no missing values for the 'EmpStatus' and 'CustIncome' predictors, neither predictor has an explicit bin for missing values.

```
bi = bininfo(sc, 'EmpStatus');
disp(bi)
```

Bin	Good	Bad	Odds	WOE	InfoValue
{'Unknown' }	396	239	1.6569	-0.19947	0.021715
{'Employed' }	407	158	2.5759	0.2418	0.026323
{'Totals' }	803	397	2.0227	NaN	0.048038

```
bi = bininfo(sc, 'CustIncome');
disp(bi)
```

Bin	Good	Bad	Odds	WOE	InfoValue
{'[0,29000)' }	53	58	0.91379	-0.79457	0.06364
{'[29000,33000)' }	74	49	1.5102	-0.29217	0.0091366
{'[33000,35000)' }	68	36	1.8889	-0.06843	0.00041042
{'[35000,40000)' }	193	98	1.9694	-0.026696	0.00017359
{'[40000,42000)' }	68	34	2	-0.011271	1.0819e-05
{'[42000,47000)' }	164	66	2.4848	0.20579	0.0078175
{'[47000,Inf]' }	183	56	3.2679	0.47972	0.041657
{'Totals' }	803	397	2.0227	NaN	0.12285

Use `fitmodel` to fit a logistic regression model using Weight of Evidence (WOE) data. `fitmodel` internally transforms all the predictor variables into WOE values by using the bins found in the automatic binning process. `fitmodel` then fits a logistic regression model using a stepwise method (by default). For predictors that have missing data, there is an explicit `<missing>` bin, with a corresponding WOE value computed from the data. When you use `fitmodel`, the function applies the corresponding WOE value for the `<missing>` bin when performing the WOE transformation.

```
[sc,mdl] = fitmodel(sc);
```

1. Adding CustIncome, Deviance = 1490.8527, Chi2Stat = 32.588614, PValue = 1.1387992e-08
2. Adding TmWBank, Deviance = 1467.1415, Chi2Stat = 23.711203, PValue = 1.1192909e-06
3. Adding AMBalance, Deviance = 1455.5715, Chi2Stat = 11.569967, PValue = 0.00067025601
4. Adding EmpStatus, Deviance = 1447.3451, Chi2Stat = 8.2264038, PValue = 0.0041285257
5. Adding CustAge, Deviance = 1442.8477, Chi2Stat = 4.4974731, PValue = 0.033944979
6. Adding ResStatus, Deviance = 1438.9783, Chi2Stat = 3.86941, PValue = 0.049173805
7. Adding OtherCC, Deviance = 1434.9751, Chi2Stat = 4.0031966, PValue = 0.045414057

Generalized linear regression model:  
 status ~ [Linear formula with 8 terms in 7 predictors]  
 Distribution = Binomial

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	0.70229	0.063959	10.98	4.7498e-28
CustAge	0.57421	0.25708	2.2335	0.025513
ResStatus	1.3629	0.66952	2.0356	0.04179
EmpStatus	0.88373	0.2929	3.0172	0.002551
CustIncome	0.73535	0.2159	3.406	0.00065929
TmWBank	1.1065	0.23267	4.7556	1.9783e-06
OtherCC	1.0648	0.52826	2.0156	0.043841
AMBalance	1.0446	0.32197	3.2443	0.0011775

```
1200 observations, 1192 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 88.5, p-value = 2.55e-16
```

Scale the scorecard points by the "points, odds, and points to double the odds (PDO)" method using the 'PointsOddsAndPDO' argument of `formatpoints`. Suppose that you want a score of 500 points to have odds of 2 (twice as likely to be good than to be bad) and that the odds double every 50 points (so that 550 points would have odds of 4).

Display the scorecard showing the scaled points for predictors retained in the fitting model.

```
sc = formatpoints(sc, 'PointsOddsAndPDO', [500 2 50]);
PointsInfo = displaypoints(sc)
```

```
PointsInfo=38x3 table
   Predictors      Bin      Points
   _____  _____  _____
   {'CustAge' } {' [0,33) ' } 54.062
   {'CustAge' } {' [33,37) ' } 56.282
   {'CustAge' } {' [37,40) ' } 60.012
   {'CustAge' } {' [40,46) ' } 69.636
   {'CustAge' } {' [46,48) ' } 77.912
   {'CustAge' } {' [48,51) ' } 78.86
   {'CustAge' } {' [51,58) ' } 80.83
   {'CustAge' } {' [58,Inf] ' } 96.76
   {'CustAge' } {' <missing> ' } 64.984
   {'ResStatus'} {'Tenant' } 62.138
   {'ResStatus'} {'Home Owner'} 73.248
   {'ResStatus'} {'Other' } 90.828
   {'ResStatus'} {'<missing>' } 74.125
   {'EmpStatus'} {'Unknown' } 58.807
   {'EmpStatus'} {'Employed' } 86.937
   {'EmpStatus'} {'<missing>' } NaN
   :
```

Notice that points for the <missing> bin for 'CustAge' and 'ResStatus' are explicitly shown (as 64.9836 and 74.1250, respectively). The function computes these points from the WOE value for the <missing> bin and the logistic model coefficients.

For predictors that have no missing data in the training set, there is no explicit <missing> bin during the training of the model. By default, `displaypoints` reports the points as NaN for missing data resulting in a score of NaN when you use `score`. For these predictors, use the name-value pair argument 'Missing' in `formatpoints` to indicate how missing data should be treated for scoring purposes.

Use `compactCreditScorecard` to convert the `creditscorecard` object into a `compactCreditScorecard` object. A `compactCreditScorecard` object is a lightweight version of a `creditscorecard` object that is used for deployment purposes.

```
csc = compactCreditScorecard(sc);
```

For the purpose of illustration, take a few rows from the original data as test data and introduce some missing data. Also introduce some invalid, or out-of-range, values. For numeric data, values below the

minimum (or above the maximum) are considered invalid, such as a negative value for age (recall that in a previous step, you set 'MinValue' to 0 for 'CustAge' and 'CustIncome'). For categorical data, invalid values are categories not explicitly included in the scorecard, for example, a residential status not previously mapped to scorecard categories, such as "House", or a meaningless string such as "abc123."

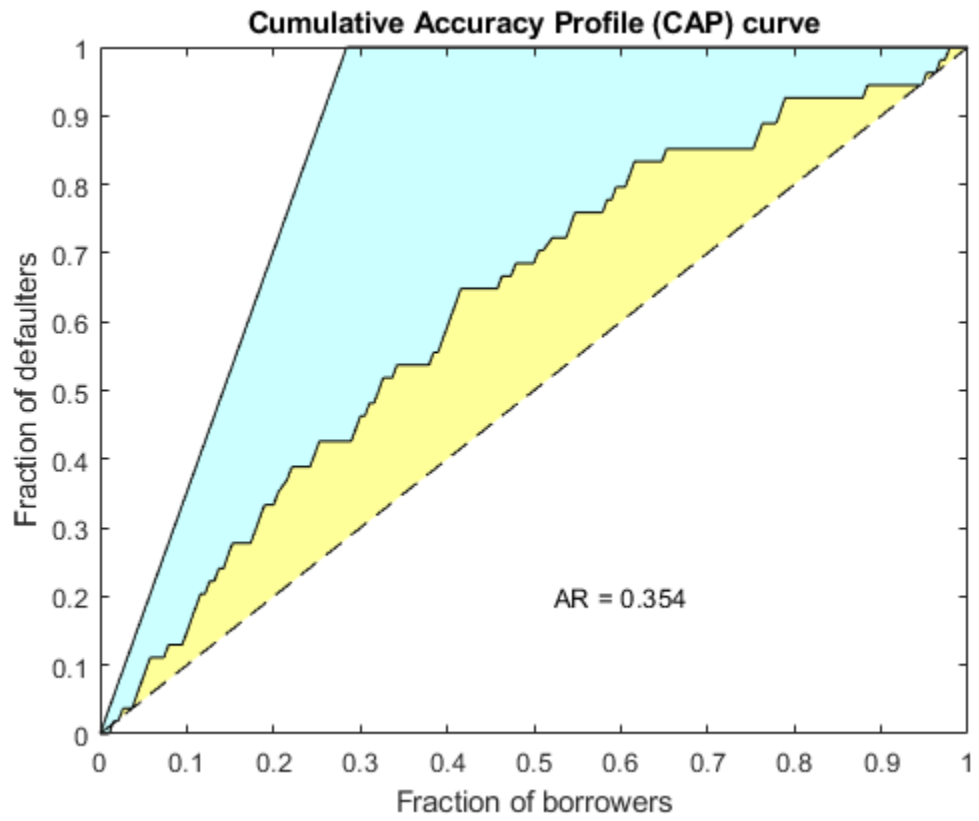
This example uses a very small validation data set only to illustrate the scoring of rows with missing and out-of-range values and the relationship between scoring and model validation.

```
tdata = dataMissing(11:200,mdl.PredictorNames); % Keep only the predictors retained in the model
tdata.status = dataMissing.status(11:200); % Copy the response variable value, needed for validation
% Set some missing values
tdata.CustAge(1) = NaN;
tdata.ResStatus(2) = '<undefined>';
tdata.EmpStatus(3) = '<undefined>';
tdata.CustIncome(4) = NaN;
% Set some invalid values
tdata.CustAge(5) = -100;
tdata.ResStatus(6) = 'House';
tdata.EmpStatus(7) = 'Freelancer';
tdata.CustIncome(8) = -1;
disp(tdata(1:10,:))
```

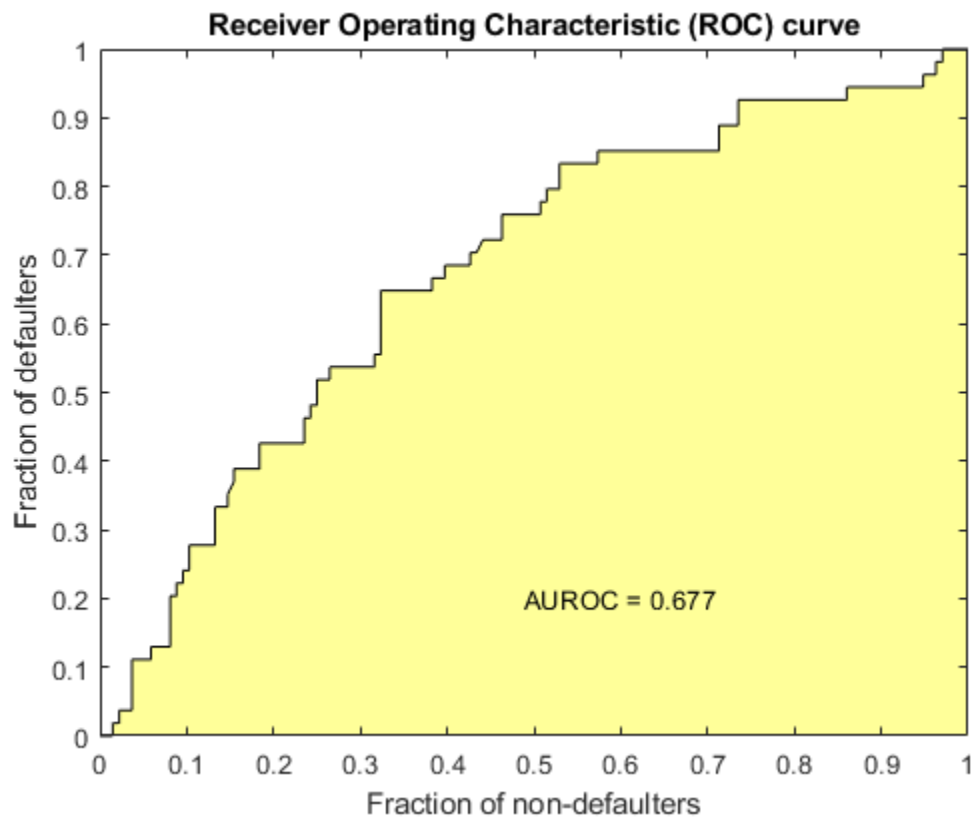
CustAge	ResStatus	EmpStatus	CustIncome	TmWBank	OtherCC	AMBalance	sta
NaN	Tenant	Unknown	34000	44	Yes	119.8	
48	<undefined>	Unknown	44000	14	Yes	403.62	
65	Home Owner	<undefined>	48000	6	No	111.88	
44	Other	Unknown	NaN	35	No	436.41	
-100	Other	Employed	46000	16	Yes	162.21	
33	House	Employed	36000	36	Yes	845.02	
39	Tenant	Freelancer	34000	40	Yes	756.26	
24	Home Owner	Employed	-1	19	Yes	449.61	
NaN	Home Owner	Employed	51000	11	Yes	519.46	
52	Other	Unknown	42000	12	Yes	1269.2	

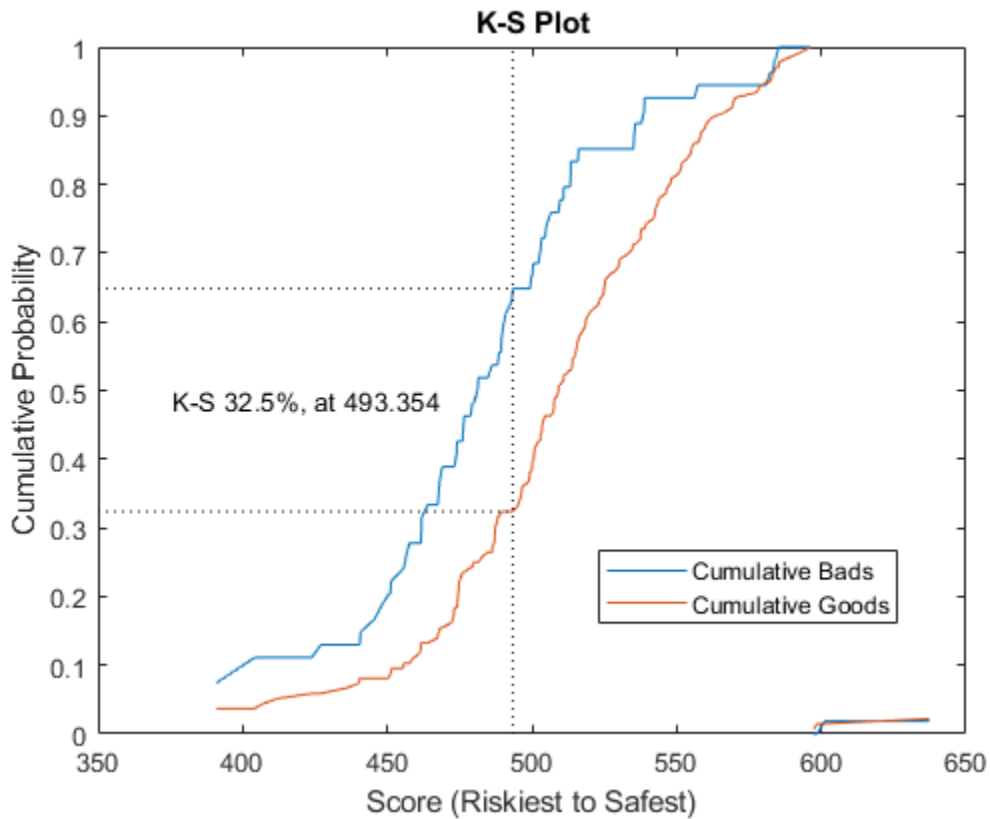
Use `validatemodel` for a `compactCreditScorecard` object with the validation data set (`tdata`).

```
[ValStats,ValTable] = validatemodel(csc,tdata,'Plot',{'CAP','ROC','KS'});
```









disp(ValStats)

Measure	Value
{'Accuracy Ratio' }	0.35376
{'Area under ROC curve' }	0.67688
{'KS statistic' }	0.32462
{'KS score' }	493.35

disp(ValTable(1:10,:))

Scores	ProbDefault	TrueBads	FalseBads	TrueGoods	FalseGoods	Sensitivity
597.33	NaN	0	1	135	54	0
598.54	NaN	0	2	134	54	0
601.18	NaN	1	2	134	53	0.018519
637.3	NaN	1	3	133	53	0.018519
NaN	0.69421	2	3	133	52	0.037037
NaN	0.65394	2	4	132	52	0.037037
NaN	0.64441	2	5	131	52	0.037037
NaN	0.62799	3	5	131	51	0.055556
390.86	0.58964	4	5	131	50	0.074074
404.09	0.57902	6	5	131	48	0.11111

## Input Arguments

### **csc** — Compact credit scorecard model

`compactCreditScorecard` object

Compact credit scorecard model, specified as a `compactCreditScorecard` object.

To create a `compactCreditScorecard` object, use `compactCreditScorecard` or `compact` from Financial Toolbox.

### **data** — Validation data

table

Validation data, specified as a MATLAB table, where each table row corresponds to individual observations. The `data` must contain columns for each of the predictors in the credit scorecard model. The columns of data can be any one of the following data types:

- Numeric
- Logical
- Cell array of character vectors
- Character array
- Categorical
- String
- String array

In addition, the table must contain a binary response variable and the name of this column must match the name of the `ResponseVar` property in the `compactCreditScorecard` object. (The `ResponseVar` property in the `compactCreditScorecard` is copied from the `ResponseVar` property of the original `creditscorecard` object.)

---

**Note** If a different validation data set is provided using the optional `data` input, observation weights for the validation data must be included in a column whose name matches `WeightsVar` from the original `creditscorecard` object, otherwise unit weights are used for the validation data. For more information, see “Using `validatemodel` with Weights” (Financial Toolbox).

---

Data Types: `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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `csc = validatemodel(csc,data,'Plot','CAP')`

### **Plot** — Type of plot

'None' (default) | character vector with values 'None', 'CAP', 'ROC', 'KS' | cell array of character vectors with values 'None', 'CAP', 'ROC', 'KS'

Type of plot, specified as the comma-separated pair consisting of 'Plot' and a character vector with one of the following values:

- 'None' — No plot is displayed.
- 'CAP' — Cumulative Accuracy Profile. Plots the fraction of borrowers up to score “s” against the fraction of defaulters up to score “s” ('PctObs' against 'Sensitivity' columns of T optional output argument). For details, see “Cumulative Accuracy Profile (CAP)” (Financial Toolbox).
- 'ROC' — Receiver Operating Characteristic. Plots the fraction of non-defaulters up to score “s” against the fraction of defaulters up to score “s” ('FalseAlarm' against 'Sensitivity' columns of T optional output argument). For details, see “Receiver Operating Characteristic (ROC)” (Financial Toolbox).
- 'KS' — Kolmogorov-Smirnov. Plots each score “s” against the fraction of defaulters up to score “s,” and also against the fraction of nondefaulters up to score “s” ('Scores' against both 'Sensitivity' and 'FalseAlarm' columns of the optional output argument T). For details, see “Kolmogorov-Smirnov statistic (KS)” (Financial Toolbox).

---

**Tip** For the Kolmogorov-Smirnov statistic option, you can enter either 'KS' or 'K-S'.

---

Data Types: char | cell

## Output Arguments

### Stats — Validation measures

table

Validation measures, returned as a 4-by-2 table. The first column, 'Measure', contains the names of the following measures:

- Accuracy ratio (AR)
- Area under the ROC curve (AUROC)
- The KS statistic
- KS score

The second column, 'Value', contains the values corresponding to these measures.

### T — Validation statistics data

array

Validation statistics data, returned as an N-by-9 table of validation statistics data, sorted by score from riskiest to safest. N is equal to the total number of unique scores, that is, scores without duplicates.

The table T contains the following nine columns, in this order:

- 'Scores' — Scores sorted from riskiest to safest. The data in this row corresponds to all observations up to and including the score in this row.
- 'ProbDefault' — Probability of default for observations in this row. For deciles, the average probability of default for all observations in the given decile is reported.
- 'TrueBads' — Cumulative number of “bads” up to and including the corresponding score.
- 'FalseBads' — Cumulative number of “goods” up to and including the corresponding score.
- 'TrueGoods' — Cumulative number of “goods” above the corresponding score.
- 'FalseGoods' — Cumulative number of “bads” above the corresponding score.

- 'Sensitivity' — Fraction of defaulters (or the cumulative number of “bads” divided by total number of “bads”). This is the distribution of “bads” up to and including the corresponding score.
- 'FalseAlarm' — Fraction of nondefaulters (or the cumulative number of “goods” divided by total number of “goods”). This is the distribution of “goods” up to and including the corresponding score.
- 'PctObs' — Fraction of borrowers, or the cumulative number of observations, divided by total number of observations up to and including the corresponding score.

---

**Note** When creating the `creditscorecard` object with `creditscorecard`, if the optional name-value pair argument `WeightsVar` was used to specify observation (sample) weights, then the T table uses statistics, sums, and cumulative sums that are weighted counts.

---

### hf — Handle to the plotted measures

figure handle

Figure handle to plotted measures, returned as a figure handle or array of handles. When `Plot` is set to 'None', `hf` is an empty array.

## More About

### Cumulative Accuracy Profile (CAP)

CAP is generally a concave curve and is also known as the Gini curve, Power curve, or Lorenz curve.

The scores of given observations are sorted from riskiest to safest. For a given fraction  $M$  (0% to 100%) of the total borrowers, the height of the CAP curve is the fraction of defaulters whose scores are less than or equal to the maximum score of the fraction  $M$ . This fraction of defaulters is also known as the “Sensitivity.”

The area under the CAP curve, known as the AUCAP, is then compared to that of the perfect or “ideal” model, leading to the definition of a summary index known as the accuracy ratio ( $AR$ ) or the Gini coefficient:

$$AR = \frac{A_R}{A_P}$$

where  $A_R$  is the area between the CAP curve and the diagonal, and  $A_P$  is the area between the perfect model and the diagonal. This represents a “random” model, where scores are assigned randomly and therefore the proportion of defaulters and nondefaulters is independent of the score. The perfect model is the model for which all defaulters are assigned the lowest scores, and therefore perfectly discriminates between defaulters and nondefaulters. Thus, the closer to unity  $AR$  is, the better the scoring model.

### Receiver Operating Characteristic (ROC)

To find the receiver operating characteristic (ROC) curve, the proportion of defaulters up to a given score “s,” or “Sensitivity,” is computed.

This proportion is known as the true positive rate (TPR). Also, the proportion of nondefaulters up to score “s,” or “False Alarm Rate,” is also computed. This proportion is also known as the false positive rate (FPR). The ROC curve is the plot of the “Sensitivity” vs. the “False Alarm Rate.” Computing the ROC curve is similar to computing the equivalent of a confusion matrix at each score level.

Similar to the CAP, the ROC has a summary statistic known as the area under the ROC curve (AUROC). The closer to unity, the better the scoring model. The accuracy ratio (AR) is related to the area under the curve by the following formula:

$$AR = 2(AUROC) - 1$$

### Kolmogorov-Smirnov Statistic (KS)

The Kolmogorov-Smirnov (KS) plot, also known as the fish-eye graph, is a common statistic for measuring the predictive power of scorecards.

The KS plot shows the distribution of defaulters and the distribution of nondefaulters on the same plot. For the distribution of defaulters, each score "s" is plotted against the proportion of defaulters up to "s," or "Sensitivity." For the distribution of non-defaulters, each score "s" is plotted against the proportion of nondefaulters up to "s," or "False Alarm." The statistic of interest is called the KS statistic and is the maximum difference between these two distributions ("Sensitivity" minus "False Alarm"). The score at which this maximum is attained is also of interest.

### Use `validateModel` with Weights

If you provide observation weights, the `validateModel` function incorporates the observation weights when calculating model validation statistics.

If you do not provide weights, the validation statistics are based on how many good and bad observations fall below a particular score. If you do provide weights, the weight (not the count) is accumulated for the good and the bad observations that fall below a particular score.

When you define observation weights using the optional `WeightsVar` name-value pair argument when creating a `creditscorecard` object, the weights stored in the `WeightsVar` column are used when validating the model on the training data. When a different validation data set is provided using the optional `data` input, observation weights for the validation data must be included in a column whose name matches `WeightsVar`. Otherwise, the unit weights are used for the validation data set.

The observation weights of the training data affect not only the validation statistics but also the credit scorecard scores themselves. For more information, see "Using `fitModel` with Weights" (Financial Toolbox) and "Credit Scorecard Modeling Using Observation Weights" (Financial Toolbox).

## References

- [1] "Basel Committee on Banking Supervision: Studies on the Validation of Internal Rating Systems." Working Paper No. 14, February 2005.
- [2] Refaat, M. *Credit Risk Scorecards: Development and Implementation Using SAS*. lulu.com, 2011.
- [3] Loeffler, G. and P. N. Posch. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

## See Also

`compactCreditScorecard` | `displayPoints` | `probDefault` | `score`

## Topics

"compactCreditScorecard Object Workflow"

"Case Study for a Credit Scorecard Analysis" (Financial Toolbox)

"Credit Scorecard Modeling with Missing Values" (Financial Toolbox)

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

**Introduced in R2019b**

## screenpredictors

Screen credit scorecard predictors for predictive value

### Syntax

```
metric_table = screenpredictors(data)
metric_table = screenpredictors( ___,Name,Value)
```

### Description

`metric_table = screenpredictors(data)` returns the output variable, `metric_table`, a MATLAB table containing the calculated values for several measures of predictive power for each predictor variable in the data. Use the `screenpredictors` function as a preprocessing step in the “Credit Scorecard Modeling Workflow” (Financial Toolbox) to reduce the number of predictor variables before you create the credit scorecard using the `creditscorecard` function from Financial Toolbox.

`metric_table = screenpredictors( ___,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input arguments in the previous syntax.

### Examples

#### Screen Predictors for a `creditscorecard` Object

Reduce the number of predictor variables by screening predictors before you create a credit scorecard.

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

```
load CreditCardData
```

Define 'IDVar' and 'ResponseVar'.

```
idvar = 'CustID';
responsevar = 'status';
```

Use `screenpredictors` to calculate the predictor screening metrics. The function returns a table containing the metrics values. Each table row corresponds to a predictor from the input table data.

```
metric_table = screenpredictors(data,'IDVar',idvar,'ResponseVar',responsevar)
```

```
metric_table=9×7 table
                InfoValue  AccuracyRatio  AUROC  Entropy  Gini  Chi2PValue
    _____  _____  _____  _____  _____  _____
    CustAge      0.18863      0.17095      0.58547  0.88729  0.42626  0.00074524
    TmWBank      0.15719      0.13612      0.56806  0.89167  0.42864  0.0054591
    CustIncome   0.15572      0.17758      0.58879  0.891    0.42731  0.0018428
    TmAtAddress  0.094574     0.010421     0.50521  0.90089  0.43377  0.182
    UtilRate     0.075086     0.035914     0.51796  0.90405  0.43575  0.45546
```



AMBalance	0.07159	0.087142	0.54357	0.90446	0.43592	0.48528
EmpStatus	0.048038	0.10886	0.55443	0.90814	0.4381	0.00037823
OtherCC	0.014301	0.044459	0.52223	0.91347	0.44132	0.047616
ResStatus	0.0097738	0.05039	0.5252	0.91422	0.44182	0.27875

```
metric_table = sortrows(metric_table, 'AccuracyRatio', 'descend')
```

```
metric_table=9x7 table
```

	InfoValue	AccuracyRatio	AUROC	Entropy	Gini	Chi2PValue
CustIncome	0.15572	0.17758	0.58879	0.891	0.42731	0.0018428
CustAge	0.18863	0.17095	0.58547	0.88729	0.42626	0.00074524
TmWBank	0.15719	0.13612	0.56806	0.89167	0.42864	0.0054591
EmpStatus	0.048038	0.10886	0.55443	0.90814	0.4381	0.00037823
AMBalance	0.07159	0.087142	0.54357	0.90446	0.43592	0.48528
ResStatus	0.0097738	0.05039	0.5252	0.91422	0.44182	0.27875
OtherCC	0.014301	0.044459	0.52223	0.91347	0.44132	0.047616
UtilRate	0.075086	0.035914	0.51796	0.90405	0.43575	0.45546
TmAtAddress	0.094574	0.010421	0.50521	0.90089	0.43377	0.182

Based on the AccuracyRatio metric, select the top predictors to use when you create the `creditscorecard` object.

```
varlist = metric_table.Row(metric_table.AccuracyRatio > 0.09)
```

```
varlist = 4x1 cell
```

```
{'CustIncome'}
{'CustAge' }
{'TmWBank' }
{'EmpStatus' }
```

Use `creditscorecard` to create a `createscorecard` object based on only the "screened" predictors.

```
sc = creditscorecard(data, 'IDVar', idvar, 'ResponseVar', responsevar, 'PredictorVars', varlist)
```

```
sc =
```

```
creditscorecard with properties:
```

```

    GoodLabel: 0
    ResponseVar: 'status'
    WeightsVar: ''
    VarNames: {1x11 cell}
    NumericPredictors: {'CustAge' 'CustIncome' 'TmWBank'}
    CategoricalPredictors: {'EmpStatus'}
    BinMissingData: 0
    IDVar: 'CustID'
    PredictorVars: {'CustAge' 'EmpStatus' 'CustIncome' 'TmWBank'}
    Data: [1200x11 table]
```

## Input Arguments

### **data** — Data for `creditscorecard` object

table | tall table | tall timetable

Data for the `creditscorecard` object, specified as a MATLAB table, tall table, or tall timetable, where each column of data can be any one of the following data types:

- Numeric
- Logical
- Cell array of character vectors
- Character array
- Categorical
- String

Data Types: 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 quotes. You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

Example: `metric_table = screenpredictors(data, 'IDVar', 'CustAge', 'ResponseVar', 'status', 'PredictorVars', {'CustID', 'CustIncome'})`

### **IDVar** — Name of identifier variable

' ' (default) | character vector

Name of identifier variable, specified as the comma-separated pair consisting of `'IDVar'` and a case-sensitive character vector. The `'IDVar'` data can be ordinal numbers or Social Security numbers. By specifying `'IDVar'`, you can omit the identifier variable from the predictor variables easily.

Data Types: char

### **ResponseVar** — Response variable name for “Good” or “Bad” indicator

last column of the data input (default) | character vector

Response variable name for the “Good” or “Bad” indicator, specified as the comma-separated pair consisting of `'ResponseVar'` and a case-sensitive character vector. The response variable data must be binary.

If not specified, `'ResponseVar'` is set to the last column of the input data by default.

Data Types: char

### **PredictorVars** — Names of predictor variables

set difference between `VarNames` and `{IDVar, ResponseVar}` (default) | cell array of character vectors | string array

Names of predictor variables, specified as the comma-separated pair consisting of `'PredictorVars'` and a case-sensitive cell array of character vectors or string array. By default, when you create a `creditscorecard` object, all variables are predictors except for `IDVar` and

**ResponseVar.** Any name you specify using 'PredictorVars' must differ from the IDVar and ResponseVar names.

Data Types: cell | string

### **WeightsVar — Name of weights variable**

' ' (default) | character vector

Name of weights variable, specified as the comma-separated pair consisting of 'WeightsVar' and a case-sensitive character vector to indicate which column name in the data table contains the row weights.

If you do not specify 'WeightsVar' when you create a creditscorecard object, then the function uses the unit weights as the observation weights.

Data Types: char

### **NumBins — Number of (equal frequency) bins for numeric predictors**

20 (default) | scalar numeric

Number of (equal frequency) bins for numeric predictors, specified as the comma-separated pair consisting of 'NumBins' and a scalar numeric.

Data Types: double

### **FrequencyShift — Indicates small shift in frequency tables that contain zero entries**

0.5 (default) | scalar numeric between 0 and 1

Small shift in frequency tables that contain zero entries, specified as the comma-separated pair consisting of 'FrequencyShift' and a scalar numeric with a value between 0 and 1.

If the frequency table of a predictor contains any "pure" bins (containing all goods or all bads) after you bin the data using `autobinning`, then the function adds the 'FrequencyShift' value to all bins in the table. To avoid any perturbation, set 'FrequencyShift' to 0.

Data Types: double

## **Output Arguments**

### **metric\_table — Calculated values for predictor screening metrics**

table

Calculated values for the predictor screening metrics, returned as table. Each table row corresponds to a predictor from the input table data. The table columns contain calculated values for the following metrics:

- 'InfoValue' — Information value. This metric measures the strength of a predictor in the fitting model by determining the deviation between the distributions of "Goods" and "Bads".
- 'AccuracyRatio' — Accuracy ratio.
- 'AUROC' — Area under the ROC curve.
- 'Entropy' — Entropy. This metric measures the level of unpredictability in the bins. You can use the entropy metric to validate a risk model.
- 'Gini' — Gini. This metric measures the statistical dispersion or inequality within a sample of data.

- 'Chi2PValue' — Chi-square  $p$ -value. This metric is computed from the chi-square metric and is a measure of the statistical difference and independence between groups.
- 'PercentMissing' — Percentage of missing values in the predictor. This metric is expressed in decimal form.

## Extended Capabilities

### Tall Arrays

Calculate with arrays that have more rows than fit in memory.

This function supports input `data` that is specified as a tall column vector, a tall table, or a tall timetable. Note that the output for numeric predictors might be slightly different when using a tall array. Categorical predictors return the same results for tables and tall arrays. For more information, see `tall` and “Tall Arrays” (MATLAB).

### See Also

`bininfo` | `creditscorecard` | `modifybins` | `modifypredictor`

### Topics

“Feature Screening with `screenpredictors`”

**Introduced in R2019a**

# esbacktestbyte

Create `esbacktestbyte` object to run suite of Du and Escanciano expected shortfall (ES) backtests

## Description

The general workflow is:

- 1 Load or generate the data for the ES backtesting analysis.
- 2 Create an `esbacktestbyte` object. For more information, see [Create esbacktestbyte](#) on page 5-235 and [Properties](#) on page 5-238.
- 3 Use the `summary` function to generate a summary report on the failures and severities.
- 4 Use the `runtests` function to run all tests at once.
- 5 For additional test details, run the following individual tests:
  - `unconditionalDE` — Unconditional ES backtest by Du-Escanciano
  - `conditionalDE` — Conditional ES backtest by Du-Escanciano
- 6 `simulate` — Simulate critical values for test statistics

For more information, see “Overview of Expected Shortfall Backtesting” on page 2-21 and “Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano”.

## Creation

### Syntax

```
ebtde = esbacktestbyte(PortfolioData, DistributionName)
ebtde = esbacktestbyte( ____, Name, Value)
```

### Description

`ebtde = esbacktestbyte(PortfolioData, DistributionName)` creates an `esbacktestbyte` (`ebtde`) object using portfolio outcomes data and model distribution information. The `esbacktestbyte` object has the following properties:

- `PortfolioData` on page 5-0 — `NumRows`-by-1 numeric array or `NumRows`-by-1 table or timetable with a numeric column containing portfolio outcomes data.
- `VaRData` on page 5-0 — Computed VaR data using distribution information from `PortfolioData`, returned as a `NumRows`-by-`NumVaRs` numeric array.
- `ESData` on page 5-0 — Computed ES data using distribution information from `PortfolioData`, returned as a `NumRows`-by-`NumVaRs` numeric array.
- `Distribution` on page 5-0 — Model distribution information, returned as a structure.
- `PortfolioID` on page 5-0 — User-defined portfolio ID.
- `VaRID` on page 5-0 — VaRIDs for the corresponding column in `PortfolioData`.

- VaRLevel on page 5-0 — VaRLevel for the corresponding columns in PortfolioData.

ebtde = esbacktestbyde( \_\_\_\_, Name, Value) sets Properties on page 5-79 using name-value pairs and any of the arguments in the previous syntax. For example, ebtde = esbacktestbyde(PortfolioData, DistributionName, 'VaRID', 'TotalVaR', 'VaRLevel', . . . 99). You can specify multiple name-value pairs as optional name-value pair arguments.

### Input Arguments

#### PortfolioData — Portfolio outcome data

NumRows-by-1 numeric array | NumRows-by-1 table of numeric columns | NumRows-by-1 timetable with one numeric column

Portfolio outcome data, specified as a NumRows-by-1 numeric array, NumRows-by-1 table of numeric columns, or a NumRows-by-1 timetable with a numeric column containing portfolio outcomes data. The PortfolioData input argument sets the PortfolioData on page 5-0 property.

Unlike other ES backtesting classes, the esbacktestbyde does not require VaR data or ES data inputs. The distribution information from PortfolioData is sufficient to run the tests. esbacktestbyde uses the distribution information to apply the cumulative distribution function to the portfolio data and map it into the (0,1) interval. The ES backtests are applied to the mapped data.

---

**Note** Before applying the tests, the function discards rows with missing values (NaN) in the PortfolioData or Distribution parameters. Therefore, the reported number of observations equals the original number of rows minus the number of missing values.

---

Data Types: double | table | timetable

#### DistributionName — Model distribution name

character vector with a value of 'normal' or 't' | string with a value of "normal" or "t"

Model distribution name for ES backtesting analysis, specified as a character vector with a value of 'normal' or 't' or string with a value of "normal" or "t".

Data Types: char | string

#### 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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, . . . , NameN, ValueN.

Example: ebtde = esbacktestbyde(PortfolioData, "t", 'DegreesOfFreedom', 10, 'Location', Mu, 'Scale', Sigma, 'PortfolioID', "S&P", 'VaRID', ["t(10) 95%", "t(10) 97.5%", "t(10) 99%"], 'VaRLevel', VaRLevel)

#### Name-Value Pairs for 'normal' or 't' Distributions

#### PortfolioID — User-defined ID

character vector | string

User-defined ID for PortfolioData input, specified as the comma-separated pair consisting of 'PortfolioID' and a character vector or string. The 'PortfolioID' name-value pair argument sets the PortfolioID on page 5-0 property.

If `PortfolioData` is a numeric array, the default value for `PortfolioID` is `'Portfolio'`. If `PortfolioData` is a table or timetable, `PortfolioID` is set to the corresponding variable name in the table, by default.

Data Types: `char` | `string`

### **VaRID — VaR identifier**

character vector | cell array of character vectors | string | string array

VaR identifier for the VaR model, specified as the comma-separated pair consisting of `'VaRID'` and a character vector, cell array of character vectors, string, or string array.

You can specify multiple `VaRID` values by using a 1-by-`NumVaRs` (or `NumVaRs`-by-1) cell array of character vectors or a string vector with user-defined IDs for the different VaR levels. The `'VaRID'` name-value pair argument sets the `VaRID` on page 5-0 property.

If `NumVaRs` = 1, the default value for `VaRID` is `'VaR'`. If `NumVaRs` > 1, the default value is `'VaR1'`, `'VaR2'`, and so on.

Data Types: `char` | `cell` | `string`

### **VaRLevel — VaR confidence level**

0.95 (default) | numeric between 0 and 1

VaR confidence level, specified as the comma-separated pair consisting of `'VaRLevel'` and a scalar numeric value between 0 and 1 or a 1-by-`NumVaRs` (or `NumVaRs`-by-1) numeric array. The `'VaRLevel'` name-value pair argument sets the `VaRLevel` on page 5-0 property.

Data Types: `double`

### **Simulate — Indicates if simulation for statistical significance of tests runs**

true (default) | scalar logical with a value of true or false

Indicates if simulation for statistical significance of tests runs when an `esbacktestbyde` object is created, specified as the comma-separated pair consisting of `'Simulate'` and a scalar logical value.

Data Types: `logical`

### **Name-Value Pairs for 'normal' Distributions**

#### **Mean — Means for the normal distribution**

0 (default) | vector

Means for the normal distribution, specified as the comma-separated pair consisting of `'Mean'` and a `NumRows`-by-1 vector. This parameter is used only when `DistributionName` is `'normal'`.

Data Types: `double`

#### **StandardDeviation — Standard deviations**

1 (default) | positive vector

Standard deviations, specified as the comma-separated pair consisting of `'StandardDeviation'` and a `NumRows`-by-1 positive vector. This parameter is only used when `DistributionName` is `"normal"`.

Data Types: `double`

**Name-Value Pairs for 't' Distributions****DegreesOfFreedom — Degrees of freedom for 't' distribution**

scalar integer  $\geq 3$

Degrees of freedom for 't' distribution, specified as the comma-separated pair consisting of 'DegreesOfFreedom' and a scalar integer  $\geq 3$ .

---

**Note** You must set this name-value parameter when `DistributionName` is 't'.

---

Data Types: double

**Location — Location parameters for 't' distribution**

0 (default) | vector

Location parameters for 't' distribution, specified as the comma-separated pair consisting of 'Location' and a NumRows-by-1 vector. This parameter is used only when `DistributionName` is 't'.

Data Types: double

**Scale — Scale parameters for 't' distribution**

1 (default) | positive vector

Scale parameters for 't' distribution, specified as the comma-separated pair consisting of 'Scale' and a NumRows-by-1 positive vector. This parameter is used only when `DistributionName` is 't'.

Data Types: double

**Properties****PortfolioData — Portfolio data for ES backtesting analysis**

numeric array

Portfolio data for ES backtesting analysis, returned as a NumRows-by-1 numeric array containing a copy of the portfolio data.

Data Types: double

**VaRData — VaR data computed using distribution information**

numeric array

VaR data computed using distribution information, returned as a NumRows-by-NumVaRs numeric array.

Data Types: double

**ESData — ES data computed using distribution information**

numeric array

ES data computed using distribution information, returned as a NumRows-by-NumVaRs numeric array.

Data Types: double



**Distribution — Model distribution information**

struct

Model distribution information, returned as a struct.

For a normal distribution, the `Distribution` structure has the fields 'Name' (set to `normal`), 'Mean', and 'StandardDeviation', with values set to the corresponding inputs.

For a `t` distribution, the `Distribution` structure has the fields 'Name' (set to `t`), 'DegreesOfFreedom', 'Location', and 'Scale', with values set to the corresponding inputs.

Data Types: struct

**PortfolioID — Portfolio identifier**

string

Portfolio identifier, returned as a string.

Data Types: string

**VaRID — VaR identifier**

string | string array

VaR identifier, returned as a 1-by-NumVaRs string array containing the VaR ES model, where NumVaRs is the number of VaR levels.

Data Types: string

**VaRLevel — VaR level**

numeric array with values between 0.90 and 0.99

VaR level, returned as a 1-by-NumVaRs numeric array.

Data Types: double

esbacktestbyte Property	Set or Modify Property from Command Line Using esbacktestbyte	Modify Property Using Dot Notation
PortfolioData	Yes	No
VaRData	No	No
ESData	No	No
Distribution	Yes	No
PortfolioID	Yes	Yes
VaRID	Yes	Yes
VaRLevel	Yes	Yes

**Object Functions**

summary	Basic expected shortfall (ES) report on failures and severity
runtests	Run all expected shortfall (ES) backtests for esbacktestbyte object
unconditionalDE	Unconditional Du-Escanciano (DE) expected shortfall (ES) backtest
conditionalDE	Conditional Du-Escanciano (DE) expected shortfall (ES) backtest
simulate	Simulate Du-Escanciano (DE) expected shortfall (ES) test statistics

## Examples

### Create an esbacktestbyde Object and Run ES Backtests

Create an `esbacktestbyde` object for a  $t$  model with 10 degrees of freedom at three different VaR levels, and then run Du and Escanciano ES backtests.

```
load ESBacktestDistributionData.mat
rng('default'); % For reproducibility
ebtde = esbacktestbyde>Returns,"t",...
    'DegreesOfFreedom',T10DoF,...
    'Location',T10Location,...
    'Scale',T10Scale,...
    'PortfolioID',"S&P",...
    'VaRID',"t(10) 95%","t(10) 97.5%","t(10) 99%",...
    'VaRLevel',VaRLevel);
runtests(ebtde)
```

```
ans=3x5 table
PortfolioID      VaRID      VaRLevel      ConditionalDE      UnconditionalDE
-----
"S&P"            "t(10) 95%"      0.95          reject             accept
"S&P"            "t(10) 97.5%"    0.975         reject             accept
"S&P"            "t(10) 99%"      0.99          reject             reject
```

### Create Two esbacktestbyde Objects and Run ES Backtests

Create two `esbacktestbyde` objects, one with a normal distribution and another with a  $t$  distribution with 5 degrees of freedom, at three different VaR levels. Then run Du and Escanciano ES backtests using `runtests`.

```
load ESBacktestDistributionData.mat
rng('default'); % For reproducibility
ebtde1 = esbacktestbyde>Returns,"normal",...
    'Mean',NormalMean,...
    'StandardDeviation',NormalStd,...
    'PortfolioID',"S&P",...
    'VaRID',"Normal 95%","Normal 97.5%","Normal 99%",...
    'VaRLevel',VaRLevel);
ebtde2 = esbacktestbyde>Returns,"t",...
    'DegreesOfFreedom',T5DoF,...
    'Location',T5Location,...
    'Scale',T5Scale,...
    'PortfolioID',"S&P",...
    'VaRID',"t(5) 95%","t(5) 97.5%","t(5) 99%",...
    'VaRLevel',VaRLevel);
```

Concatenate results in a single table.

```
t = [runtests(ebtde1);runtests(ebtde2)];
disp(t)
```

PortfolioID	VaRID	VaRLevel	ConditionalDE	UnconditionalDE
"S&P"	"Normal 95%"	0.95	reject	accept
"S&P"	"Normal 97.5%"	0.975	reject	reject
"S&P"	"Normal 99%"	0.99	reject	reject
"S&P"	"t(5) 95%"	0.95	reject	accept
"S&P"	"t(5) 97.5%"	0.975	reject	accept
"S&P"	"t(5) 99%"	0.99	accept	accept

## References

- [1] Du, Z., and J. C. Escanciano. "Backtesting Expected Shortfall: Accounting for Tail Risk." *Management Science*. Vol. 63, Issue 4, April 2017.
- [2] Basel Committee on Banking Supervision. "*Minimum Capital Requirements for Market Risk*". January 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

`conditionalDE` | `esbacktestbysim` | `runtests` | `simulate` | `summary` | `unconditionalDE`

## Topics

- "Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Rolling Windows and Multiple Models for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Expected Shortfall Estimation and Backtesting"
- "Overview of Expected Shortfall Backtesting" on page 2-21
- "ES Backtest Using Du-Escanciano Method" on page 2-24
- "Comparison of ES Backtesting Methods" on page 2-26

## Introduced in R2019b

## summary

Basic expected shortfall (ES) report on failures and severity

### Syntax

```
S = summary(ebtde)
```

### Description

`S = summary(ebtde)` returns a basic report on the given `esbacktestbyte` data. The report includes the number of observations, number of failures, observed confidence level, and so on. See `S` for details.

Unlike other ES backtesting classes, the `esbacktestbyte` object does not require VaR data or ES data inputs. `esbacktestbyte` internally computes VaR and ES data based on distribution information to determine the severity information reported by the `summary` function.

### Examples

#### Create an `esbacktestbyte` Object and Run ES Backtest Summary Report

Create an `esbacktestbyte` object for a  $t$  model with 10 degrees of freedom, and then run a basic ES backtest summary report.

```
load ESBacktestDistributionData.mat
rng('default'); % For reproducibility
ebtde = esbacktestbyte>Returns,"t",...
    'DegreesOfFreedom',T10DoF,...
    'Location',T10Location,...
    'Scale',T10Scale,...
    'PortfolioID',"S&P",...
    'VaRID',["t(10) 95%","t(10) 97.5%","t(10) 99%"],...
    'VaRLevel',VaRLevel);
summary(ebtde)
```

```
ans=3x11 table
PortfolioID      VaRID      VaRLevel      ObservedLevel      ExpectedSeverity      ObservedSev
```

PortfolioID	VaRID	VaRLevel	ObservedLevel	ExpectedSeverity	ObservedSev
"S&P"	"t(10) 95%"	0.95	0.94812	1.3288	1.4515
"S&P"	"t(10) 97.5%"	0.975	0.97202	1.2652	1.4134
"S&P"	"t(10) 99%"	0.99	0.98627	1.2169	1.3947

### Input Arguments

**ebtde** — `esbacktestbyte` object  
object

esbacktestbyde object contains a copy of the data (the PortfolioData, VaRData, and ESData properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested.

---

**Note** Unlike other ES backtesting classes, esbacktestbyde does not require VaR data or ES data inputs. esbacktestbyde internally computes VaR and ES data based on distribution information to determine the severity information reported by summary. For more information on creating an esbacktestbyde object, see esbacktestbyde.

---

## 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:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR levels
- 'VaRLevel' — VaR level
- 'ObservedLevel' — Observed confidence level, defined as the number of periods without failures divided by number of observations
- 'ExpectedSeverity' — Expected average severity ratio, that is, the average ratio of ES to VaR over the periods with VaR failures
- 'ObservedSeverity' — Observed average severity ratio, that is, the average ratio of loss to VaR over the periods with VaR failures
- '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 1 minus the VaR level
- 'Ratio' — Ratio of number of failures to expected number of failures
- 'Missing' — Number of periods with missing values removed from the sample

---

**Note** The 'ExpectedSeverity' and 'ObservedSeverity' ratios are undefined (NaN) when there are no VaR failures in the data.

---

## References

- [1] Du, Z., and J. C. Escanciano. "Backtesting Expected Shortfall: Accounting for Tail Risk." *Management Science*. Vol. 63, Issue 4, April 2017.
- [2] Basel Committee on Banking Supervision. "*Minimum Capital Requirements for Market Risk*". January 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

conditionalDE | esbacktestbyde | esbacktestbysim | runtests | simulate | unconditionalDE

**Topics**

“Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano”

“Rolling Windows and Multiple Models for Expected Shortfall (ES) Backtesting by Du and Escanciano”

“Expected Shortfall Estimation and Backtesting”

“Overview of Expected Shortfall Backtesting” on page 2-21

“ES Backtest Using Du-Escanciano Method” on page 2-24

“Comparison of ES Backtesting Methods” on page 2-26

**Introduced in R2019b**

## runtests

Run all expected shortfall (ES) backtests for `esbacktestbyde` object

### Syntax

```
TestResults = runtests(ebtde)
TestResults = runtests( ____, Name, Value)
```

### Description

`TestResults = runtests(ebtde)` runs all the tests for the `esbacktestbyde` object. `runtests` reports only the final test result. For test details such as  $p$ -values, run the individual tests:

- `unconditionalDE`
- `conditionalDE`

`TestResults = runtests( ____, Name, Value)` specifies options using one or more name-value pair arguments in addition to the input argument in the previous syntax.

### Examples

#### Create an `esbacktestbyde` Object and Run ES Backtests

Create an `esbacktestbyde` object for a  $t$  model with 10 degrees of freedom, and then run ES backtests.

```
load ESBacktestDistributionData.mat
rng('default'); % For reproducibility
ebtde = esbacktestbyde>Returns,"t",...
    'DegreesOfFreedom',T10DoF,...
    'Location',T10Location,...
    'Scale',T10Scale,...
    'PortfolioID',"S&P",...
    'VaRID',"t(10) 95%","t(10) 97.5%","t(10) 99%",...
    'VaRLevel',VaRLevel);
runtests(ebtde)
```

```
ans=3x5 table
    PortfolioID      VaRID      VaRLevel      ConditionalDE      UnconditionalDE
    _____      _____      _____      _____      _____
    "S&P"           "t(10) 95%"      0.95           reject            accept
    "S&P"           "t(10) 97.5%"    0.975          reject            accept
    "S&P"           "t(10) 99%"      0.99           reject            reject
```

To view complete details for the tests, use the name-value pair argument `'ShowDetails'`.

```
runtests(ebtde,'ShowDetails',true)
```

```
ans=3x8 table
    PortfolioID      VaRID      VaRLevel      ConditionalDE      UnconditionalDE      CriticalValue
```

"S&P"	"t(10) 95%"	0.95	reject	accept	"large-samp
"S&P"	"t(10) 97.5%"	0.975	reject	accept	"large-samp
"S&P"	"t(10) 99%"	0.99	reject	reject	"large-samp

## Input Arguments

### ebtde — esbacktestbyde object

object

esbacktestbyde object, which contains a copy of the data (the `PortfolioData`, `VarData`, and `ESData` properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktestbyde object, see esbacktestbyde.

### 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 quotes. You can specify several name and value pair arguments in any order as `Name1`, `Value1`, ..., `NameN`, `ValueN`.

Example: `TestResults = runtests(ebtde, 'CriticalValueMethod', 'simulation', 'TestLevel', 0.99, 'ShowDetails', true)`

### CriticalValueMethod — Method to compute critical values, confidence intervals, and *p*-values

'large-sample' (default) | character vector with values of 'large-sample' or 'simulation' | string with values of "large-sample" or "simulation"

Method to compute critical values, confidence intervals, and *p*-values, specified as the comma-separated pair consisting of 'CriticalValueMethod' and character vector or string with a value of 'large-sample' or 'simulation'.

Data Types: char | string

### NumLags — Number of lags in the conditionalDE test

1 (default) | positive integer

Number of lags in the conditionalDE test, specified as the comma-separated pair consisting of 'NumLags' and a positive integer.

Data Types: double

### TestLevel — Test confidence level

0.95 (default) | numeric value between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric value between 0 and 1.

Data Types: double

### ShowDetails — Flag to display all details in output

false (default) | scalar logical with a value of true or false



Flag to display all details in output including the columns for critical-value method, number of lags tested, and test confidence level, specified as the comma-separated pair consisting of 'ShowDetails' and a scalar logical value.

Data Types: `logical`

## 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:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR levels
- 'VaRLevel' — VaR level
- 'ConditionalDE' — Categorical array with the categories 'accept' and 'reject', which indicate the result of the conditionalDE test
- 'UnconditionalDE' — Categorical array with the categories 'accept' and 'reject', which indicate the result of the unconditionalDE 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.

If you set the `ShowDetails` optional name-value argument to `true`, the `TestResults` table also includes 'CriticalValueMethod', 'NumLags', and 'TestLevel' columns.

---

## References

- [1] Du, Z., and J. C. Escanciano. "Backtesting Expected Shortfall: Accounting for Tail Risk." *Management Science*. Vol. 63, Issue 4, April 2017.
- [2] Basel Committee on Banking Supervision. "*Minimum Capital Requirements for Market Risk*". January 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

`conditionalDE` | `esbacktestbyde` | `esbacktestbysim` | `simulate` | `summary` | `unconditionalDE`

## Topics

- "Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Rolling Windows and Multiple Models for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Expected Shortfall (ES) Backtesting Workflow with No Model Distribution Information" on page 2-29
- "Expected Shortfall Estimation and Backtesting"
- "Overview of Expected Shortfall Backtesting" on page 2-21
- "ES Backtest Using Du-Escanciano Method" on page 2-24
- "Comparison of ES Backtesting Methods" on page 2-26

**Introduced in R2019b**

# unconditionalDE

Unconditional Du-Escanciano (DE) expected shortfall (ES) backtest

## Syntax

```
TestResults = unconditionalDE(ebtde)
[TestResults,SimTestStatistic] = unconditionalDE( ____,Name,Value)
```

## Description

`TestResults = unconditionalDE(ebtde)` runs the unconditional Du-Escanciano (DE) expected shortfall (ES) backtest [1]. The unconditional test supports critical values by large-scale approximation and by finite-sample simulation.

`[TestResults,SimTestStatistic] = unconditionalDE( ____,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input argument in the previous syntax.

## Examples

### Create an `esbacktestbyde` Object and Run an UnconditionalDE Test

Create an `esbacktestbyde` object for a  $t$  model with 10 degrees of freedom, and then run an `unconditionalDE` test.

```
load ESBacktestDistributionData.mat
rng('default'); % For reproducibility
ebtde = esbacktestbyde>Returns,"t",...
    'DegreesOfFreedom',T10DoF,...
    'Location',T10Location,...
    'Scale',T10Scale,...
    'PortfolioID',"S&P",...
    'VaRID',["t(10) 95%","t(10) 97.5%","t(10) 99%"],...
    'VaRLevel',VaRLevel);
unconditionalDE(ebtde)
```

```
ans=3x14 table
PortfolioID      VaRID      VaRLevel      UnconditionalDE      PValue      TestStatistic
-----
"S&P"           "t(10) 95%"      0.95          accept              0.181        0.028821
"S&P"           "t(10) 97.5%"    0.975         accept              0.086278     0.015998
"S&P"           "t(10) 99%"      0.99          reject              0.016871     0.0080997
```

## Input Arguments

**ebtde** — `esbacktestbyde` object  
object

esbacktestbyde (ebtde) object, which contains a copy of the data (the PortfolioData, VarData, and ESData properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktestbyde object, see esbacktestbyde.

### 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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: TestResults = unconditionalDE(ebtde, 'CriticalValueMethod', 'large-sample', 'TestLevel', 0.99)

### CriticalValueMethod — Method to compute critical values, confidence intervals, and *p*-values

'large-sample' (default) | character vector with values of 'large-sample' or 'simulation' | string with values of "large-sample" or "simulation"

Method to compute critical values, confidence intervals, and *p*-values, specified as the comma-separated pair consisting of 'CriticalValueMethod' and a character vector or string with a value of 'large-sample' or 'simulation'.

Data Types: char | string

### TestLevel — Test confidence level

0.95 (default) | numeric value between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric value 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:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR levels
- 'VaRLevel' — VaR level
- 'UnconditionalDE' — Categorical array with the categories 'accept' and 'reject', which indicate the result of the unconditional DE test
- 'PValue' — *P*-value of the unconditional DE test
- 'TestStatistic' — Unconditional DE test statistic
- 'LowerCI' — Confidence-interval lower limit for the unconditional DE test statistic
- 'UpperCI' — Confidence-interval upper limit for the unconditional DE test statistic
- 'Observations' — Number of observations
- 'CriticalValueMethod' — Method for computing confidence intervals and *p*-values

- 'MeanLS' — Mean of the large-sample normal distribution; if `CriticalValueMethod` is 'simulation', 'MeanLS' is reported as NaN
- 'StdLS' — Standard deviation of the large-sample normal distribution; if `CriticalValueMethod` is 'simulation', 'StdLS' is reported as NaN
- 'Scenarios' — Number of scenarios simulated to get the  $p$ -values; if `CriticalValueMethod` is 'large-sample', the number of scenarios is reported as NaN
- 'TestLevel' — Test confidence level

---

**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.

---

### SimTestStatistic — Simulated values of the test statistics

numeric array

Simulated values of the test statistics, returned as a NumVaRs-by-NumScenarios numeric array.

## More About

### Unconditional DE Test

The unconditional DE test is a two-sided test to check if the test statistic is close to an expected value of  $\alpha/2$ , where  $\alpha = 1 - VaRLevel$ .

The test statistic for the unconditional DE test is

$$U_{ES} = \frac{1}{N} \sum_{t=1}^N H_t$$

where

- $H_t$  is the cumulative failures or violations process;  $H_t = (\alpha - U_t)I(U_t < \alpha) / \alpha$ , where  $I(x)$  is the indicator function.
- $U_t$  are the ranks or mapped returns  $U_t = P_t(X_t)$ , where  $P_t(X_t) = P(X_t | \theta_t)$  is the cumulative distribution of the portfolio outcomes or returns  $X_t$  over a given test window  $t = 1, \dots, N$  and  $\theta_t$  are the parameters of the distribution. For simplicity, the subindex  $t$  is both the return and the parameters, understanding that the parameters are those used on date  $t$ , even though those parameters are estimated on the previous date  $t-1$ , or even prior to that.

### Significance of the Test

The test statistic  $U_{ES}$  is a random variable and a function of random return sequences:

$$U_{ES} = U_{ES}(X_1, \dots, X_N).$$

For returns observed in the test window  $1, \dots, N$ , the test statistic attains a fixed value:

$$U_{ES}^{obs} = U_{ES}(X_1^{obs}, \dots, X_N^{obs}).$$

In general, for unknown returns that follow a distribution of  $P_t$ , the value of  $U_{ES}$  is uncertain and follows a cumulative distribution function:

$$P_U(x) = P[U_{ES} \leq x].$$

This distribution function computes a confidence interval and a  $p$ -value. To determine the distribution  $P_U$ , the `esbacktestbyte` class supports the large-sample approximation and simulation methods. You can specify one of these methods by using the optional name-value pair argument `CriticalValueMethod`.

For the large-sample approximation method, the distribution  $P_U$  is derived from an asymptotic analysis. If the number of observations  $N$  is large, the test statistic  $U_{ES}$  is distributed as

$$U_{ES} \xrightarrow{\text{dist}} N\left(\frac{\alpha}{2}, \frac{\alpha(1/3 - \alpha/4)}{N}\right) = P_U$$

where  $N(\mu, \sigma^2)$  is the normal distribution with mean  $\mu$  and variance  $\sigma^2$ .

Because the test statistic cannot be smaller than 0 or greater than 1, the analytical confidence interval limits are clipped to the interval  $[0, 1]$ . Therefore, if the analytical value is negative, the test statistic is reset to 0, and if the analytical value is greater than 1, it is reset to 1.

The  $p$ -value is

$$p_{\text{value}} = 2 * \min\{P_U(U_{ES}^{\text{obs}}), 1 - P_U(U_{ES}^{\text{obs}})\}.$$

The test rejects if  $p_{\text{value}} < \alpha_{\text{test}}$ .

For the simulation method, the distribution  $P_U$  is estimated as follows

- 1 Simulate  $M$  scenarios of returns as

$$X^s = (X_1^s, \dots, X_N^s), \quad s = 1, \dots, M.$$

- 2 Compute the corresponding test statistic as

$$U_{ES}^s = U_{ES}(X_1^s, \dots, X_N^s), \quad s = 1, \dots, M.$$

- 3 Define  $P_U$  as the empirical distribution of the simulated test statistic values as

$$P_U = P[U_{ES} \leq x] = \frac{1}{M} I(U_{ES}^s \leq x),$$

where  $I(\cdot)$  is the indicator function.

In practice, simulating ranks is more efficient than simulating returns and then transforming the returns into ranks. For more information, see `simulate`.

For the empirical distribution, the value of  $1 - P_U(x)$  can differ from the value of  $P[U_{ES} \geq x]$  because the distribution may have nontrivial jumps (simulated tied values). Use the latter probability for the estimation of confidence levels and  $p$ -values.

If  $\alpha_{\text{test}} = 1 - \text{test confidence level}$ , then the confidence intervals levels  $CI_{\text{lower}}$  and  $CI_{\text{upper}}$  are the values that satisfy equations:

$$P_U(CI_{\text{lower}}) = P[CI_{\text{lower}} \leq U_{ES}] = \frac{\alpha_{\text{test}}}{2},$$

$$P[U_{ES} \geq CI_{\text{upper}}] = \frac{\alpha_{\text{test}}}{2}.$$

The reported confidence interval limits  $CI_{lower}$  and  $CI_{upper}$  are simulated test statistic values  $U_{ES}^s$  that approximately solve the preceding equations.

The  $p$ -value is determined as

$$p_{value} = 2 * \min\{P[U_{ES} \leq U_{ES}^{obs}], P[U_{ES} \geq U_{ES}^{obs}]\}.$$

The test rejects if  $p_{value} < \alpha_{test}$ .

## References

- [1] Du, Z., and J. C. Escanciano. "Backtesting Expected Shortfall: Accounting for Tail Risk." *Management Science*. Vol. 63, Issue 4, April 2017.
- [2] Basel Committee on Banking Supervision. "Minimum Capital Requirements for Market Risk". January 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

conditionalDE | esbacktestbyde | esbacktestbysim | runtests | simulate | summary

## Topics

- "Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Rolling Windows and Multiple Models for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Expected Shortfall Estimation and Backtesting"
- "Overview of Expected Shortfall Backtesting" on page 2-21
- "ES Backtest Using Du-Escanciano Method" on page 2-24
- "Comparison of ES Backtesting Methods" on page 2-26

## Introduced in R2019b

## conditionalDE

Conditional Du-Escanciano (DE) expected shortfall (ES) backtest

### Syntax

```
TestResults = conditionalDE(ebtde)
[TestResults,SimTestStatistic] = conditionalDE( ____,Name,Value)
```

### Description

`TestResults = conditionalDE(ebtde)` runs the conditional expected shortfall (ES) backtest by Du and Escanciano [1]. The conditional test supports critical values by large-scale approximation and by finite-sample simulation.

`[TestResults,SimTestStatistic] = conditionalDE( ____,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input argument in the previous syntax.

### Examples

#### Create an `esbacktestbyde` Object and Run a `conditionalDE` Test

Create an `esbacktestbyde` object for a  $t$  model with 10 degrees of freedom and 2 lags, and then run a `conditionalDE` test.

```
load ESBcktestDistributionData.mat
rng('default'); % For reproducibility
ebtde = esbacktestbyde>Returns,"t",...
    'DegreesOfFreedom',T10DoF,...
    'Location',T10Location,...
    'Scale',T10Scale,...
    'PortfolioID',"S&P",...
    'VaRID',["t(10) 95%","t(10) 97.5%","t(10) 99%"],...
    'VaRLevel',VaRLevel);
conditionalDE(ebtde,'NumLags',2)
```

```
ans=3x13 table
PortfolioID      VaRID      VaRLevel      ConditionalDE      PValue      TestStatistic
```

PortfolioID	VaRID	VaRLevel	ConditionalDE	PValue	TestStatistic
"S&P"	"t(10) 95%"	0.95	reject	3.2121e-09	39.113
"S&P"	"t(10) 97.5%"	0.975	reject	1.6979e-07	31.177
"S&P"	"t(10) 99%"	0.99	reject	9.1526e-05	18.598

### Input Arguments

**ebtde** — `esbacktestbyde` object  
object



esbacktestbyde object, which contains a copy of the data (the PortfolioData, VarData, and ESData properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktestbyde object, see esbacktestbyde.

### 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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

Example: TestResults = conditionalDE(ebtde, 'CriticalValueMethod', 'simulation', 'NumLags', 10, 'TestLevel', 0.99)

### CriticalValueMethod — Method to compute critical values, confidence intervals, and *p*-values

'large-sample' (default) | character vector with values of 'large-sample' or 'simulation' | string with values of "large-sample" or "simulation"

Method to compute critical values, confidence intervals, and *p*-values, specified as the comma-separated pair consisting of 'CriticalValueMethod' and a character vector or string with a value of 'large-sample' or 'simulation'.

Data Types: char | string

### NumLags — Number of lags in conditionalDE test

1 (default) | positive integer

Number of lags in the conditionalDE test, specified as the comma-separated pair consisting of 'NumLags' and a positive integer.

Data Types: double

### TestLevel — Test confidence level

0.95 (default) | numeric value between 0 and 1

Test confidence level, specified as the comma-separated pair consisting of 'TestLevel' and a numeric value 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:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR levels
- 'VaRLevel' — VaR level
- 'ConditionalDE' — Categorical array with the categories 'accept' and 'reject', which indicate the result of the conditional DE test

- 'PValue' —  $P$ -value of the conditional DE test
- 'TestStatistic' — Conditional DE test statistic
- 'CriticalValue' — Critical value for the conditional DE test
- 'AutoCorrelation' — Autocorrelation for the reported number of lags
- 'Observations' — Number of observations
- 'CriticalValueMethod' — Method used to compute confidence intervals and  $p$ -values
- 'NumLags' — Number of lags
- 'Scenarios' — Number of scenarios simulated to get the  $p$ -values
- 'TestLevel' — Test confidence level

---

**Note** If you specify `CriticalValueMethod` as 'large-sample', the function reports the number of 'Scenarios' as NaN.

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.

---

### **SimTestStatistic — Simulated values of the test statistics**

numeric array

Simulated values of the test statistics, returned as an `NumVaRs-by-NumScenarios` numeric array.

## **More About**

### **Conditional DE Test**

The conditional DE test is a one-sided test to check if the test statistic is much larger than zero.

The test statistic for the conditional DE test is derived in several steps. First, define the autocovariance for lag  $j$ :

$$\nu_j = \frac{1}{N-j} \sum_{t=j+1}^N (H_t - \alpha/2)(H_{t-j} - \alpha/2)$$

where

- $\alpha = 1 - \text{VaRLevel}$ .
- $H_t$  is the cumulative failures or violations process:  $H_t = (\alpha - U_t)I(U_t < \alpha) / \alpha$ , where  $I(x)$  is the indicator function.
- $U_t$  are the ranks or mapped returns  $U_t = P_t(X_t)$ , where  $P_t(X_t) = P(X_t | \theta_t)$  is the cumulative distribution of the portfolio outcomes or returns  $X_t$  over a given test window  $t = 1, \dots, N$  and  $\theta_t$  are the parameters of the distribution. For simplicity, the subindex  $t$  is both the return and the parameters, understanding that the parameters are those used on date  $t$ , even though those parameters are estimated on the previous date  $t-1$ , or even prior to that.

The exact theoretical mean  $\alpha/2$ , as opposed to the sample mean, is used in the autocovariance formula, as suggested in the paper by Du and Escanciano [1].

The autocorrelation for lag  $j$  is then

$$\rho_j = \frac{Y_j}{Y_0}$$

The test statistic for  $m$  lags is

$$C_{ES}(m) = N \sum_{j=1}^m \rho_j^2$$

*Significance of the Test*

The test statistic  $C_{ES}$  is a random variable and a function of random return sequences or portfolio outcomes  $X_1, \dots, X_N$ :

$$C_{ES} = C_{ES}(X_1, \dots, X_N).$$

For returns observed in the test window  $1, \dots, N$ , the test statistic attains a fixed value:

$$C_{ES}^{obs} = C_{ES}(X_{obs1}, \dots, X_{obsN}).$$

In general, for unknown returns that follow a distribution of  $P_t$ , the value of  $C_{ES}$  is uncertain and it follows a cumulative distribution function:

$$P_C(x) = P[C_{ES} \leq x].$$

This distribution function computes a confidence interval and a  $p$ -value. To determine the distribution  $P_C$ , the `esbacktestbyde` class supports the large-sample approximation and simulation methods. You can specify one of these methods by using the optional name-value pair argument `CriticalValueMethod`.

For the large sample approximation method, the distribution  $P_C$  is derived from an asymptotic analysis. If the number of observations  $N$  is large, the test statistic is approximately distributed as a chi-square distribution with  $m$  degrees of freedom:

$$C_{ES}(m) \xrightarrow{dist} \chi_m^2 = P_C$$

Note that the limiting distribution is independent of  $\alpha$ .

If  $\alpha_{test} = 1 - \text{test confidence level}$ , then the critical value  $CV$  is the value that satisfies the equation

$$1 - P_C(CV) = \alpha_{test}.$$

The  $p$ -value is determined as

$$P_{value} = 1 - P_C(C_{ES}^{obs}).$$

The test rejects if  $p_{value} < \alpha_{test}$ .

For the simulation method, the distribution  $P_C$  is estimated as follows

**1** Simulate  $M$  scenarios of returns as

$$X^s = (X_1^s, \dots, X_N^s), \quad s = 1, \dots, M.$$

**2** Compute the corresponding test statistic as

$$C_{ES}^s = C_{ES}(X_1^s, \dots, X_N^s), \quad s = 1, \dots, M.$$

3 Define  $P_C$  as the empirical distribution of the simulated test statistic values as

$$P_C = P[C_{ES} \leq x] = \frac{1}{M} I(C_{ES}^S \leq x),$$

where  $I(\cdot)$  is the indicator function.

In practice, simulating ranks is more efficient than simulating returns and then transforming the returns into ranks. `simulate`.

For the empirical distribution, the value of  $1 - P_C(x)$  may be different than  $P[C_{ES} \geq x]$  because the distribution may have nontrivial jumps (simulated tied values). Use the latter probability for the estimation of confidence levels and  $p$ -values.

If  $\alpha_{test} = 1 - \text{test confidence level}$ , then the critical value of levels  $CV$  is the value that satisfies the equation

$$P[C_{ES} \geq CV] = \alpha_{test}.$$

The reported critical value  $CV$  is one of the simulated test statistic values  $C_{ES}^S$  that approximately solves the preceding equation.

The  $p$ -value is determined as

$$p_{value} = P[C_{ES} \geq C_{ES}^{obs}].$$

The test rejects if  $p_{value} < \alpha_{test}$ .

## References

- [1] Du, Z., and J. C. Escanciano. "Backtesting Expected Shortfall: Accounting for Tail Risk." *Management Science*. Vol. 63, Issue 4, April 2017.
- [2] Basel Committee on Banking Supervision. "*Minimum Capital Requirements for Market Risk*". January 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

`esbacktestbyde` | `esbacktestbysim` | `runtests` | `simulate` | `summary` | `unconditionalDE`

## Topics

- "Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Rolling Windows and Multiple Models for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Expected Shortfall Estimation and Backtesting"
- "Overview of Expected Shortfall Backtesting" on page 2-21
- "ES Backtest Using Du-Escanciano Method" on page 2-24
- "Comparison of ES Backtesting Methods" on page 2-26

## Introduced in R2019b

# simulate

Simulate Du-Escanciano (DE) expected shortfall (ES) test statistics

## Syntax

```
ebtde = simulate(ebtde)
ebtde = simulate( ___,Name,Value)
```

## Description

`ebtde = simulate(ebtde)` performs a simulation of the Du-Escanciano (DE) [1] expected shortfall (ES) test statistics. `simulate` simulates scenarios and calculates the supported test statistics for each scenario. The function uses the simulated test statistics to estimate the significance of the ES backtests when the `CriticalValueMethod` name-value pair argument for `unconditionalDE` or `conditionalDE` is set to `'simulation'`.

`ebtde = simulate( ___,Name,Value)` specifies options using one or more name-value pair arguments in addition to the input argument in the previous syntax.

## Examples

### Create an `esbacktestbyde` Object and Run a Simulation

Create an `esbacktestbyde` object for a  $t$  model with 10 degrees of freedom. First, run a `conditionalDE` test based on 1000 scenarios and then use the `simulate` function to run a second simulation with 5000 scenarios.

```
load ESBacktestDistributionData.mat
rng('default'); % For reproducibility
% Constructor runs simulation with 1000 scenarios
ebtde = esbacktestbyde>Returns,"t",...
    'DegreesOfFreedom',T10DoF,...
    'Location',T10Location,...
    'Scale',T10Scale,...
    'PortfolioID',"S&P",...
    'VaRID',"t(10) 95%","t(10) 97.5%","t(10) 99%",...
    'VaRLevel',VaRLevel);
% Run conditionalDE tests
conditionalDE(ebtde,'CriticalValueMethod','simulation')
```

```
ans=3x13 table
    PortfolioID      VaRID      VaRLevel      ConditionalDE      PValue      TestStatistic      Criti
    _____      _____      _____      _____      _____      _____      _____
    "S&P"            "t(10) 95%"      0.95          reject            0.003        15.285            3
    "S&P"            "t(10) 97.5%"    0.975         reject            0.006        16.177            3
    "S&P"            "t(10) 99%"      0.99          reject            0.037        6.9975            4
```

The tests report 1000 scenarios, see the `Scenarios` column.

Run a second simulation with 5000 scenarios

```
ebtde = simulate(ebtde, 'NumScenarios', 5000);
conditionalDE(ebtde, 'CriticalValueMethod', 'simulation')
```

```
ans=3x13 table
    PortfolioID      VaRID      VaRLevel      ConditionalDE      PValue      TestStatistic      Crit
-----
    "S&P"           "t(10) 95%"      0.95          reject            0.0016      15.285            3
    "S&P"           "t(10) 97.5%"   0.975         reject            0.0046      16.177            3
    "S&P"           "t(10) 99%"     0.99          reject            0.0362      6.9975            3
```

The tests show 5000 scenarios and updated  $p$ -values and critical values.

## Input Arguments

### ebtde — esbacktestbyde object

object

esbacktestbyde object, which contains a copy of the data (the PortfolioData, VarData, ESData, and Distribution properties) and all combinations of portfolio ID, VaR ID, and VaR levels to be tested. For more information on creating an esbacktestbyde object, see esbacktestbyde.

### 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 quotes. You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.

```
Example: ebtde =
simulate(ebtde, 'NumLags', 10, 'NumScenarios', 1000000, 'BlockSize', 10000, 'TestList', 'conditionalDE')
```

### NumLags — Number of lags in the conditionalDE test statistic

5 (default) | positive integer

Number of lags in the conditionalDE test statistic, specified as the comma-separated pair consisting of 'NumLags' and a positive integer. The simulated test statistics are stored for all lags from 1 to NumLags, so that the conditionalDE test results are available for any number of lags between 1 and NumLags after running the simulate function.

Data Types: double

### NumScenarios — Number of scenarios to simulate

1000 (default) | scalar positive integer

Number of scenarios to simulate, specified using the comma-separated pair consisting of 'NumScenarios' and a scalar positive integer.

Data Types: double

### BlockSize — Number of scenarios to simulate in single simulation block

1000 (default) | scalar positive integer

Number of scenarios to simulate in a single simulation block, specified using the comma-separated pair consisting of 'BlockSize' and a scalar positive integer.

Data Types: double

### TestList – Indicator for which test statistics to simulate

["conditionalDE", "unconditionalDE"] (default) | character vector with a value of 'conditionalDE' or 'unconditionalDE' | string with a value of "conditionalDE" or "unconditionalDE"

Indicator for which test statistics to simulate, specified as the comma-separated pair consisting of 'TestList' and a cell array of character vectors or a string array with the value 'conditionalDE', 'unconditionalDE'.

Data Types: cell | string

## Output Arguments

### ebtde – Updated esbacktestbyte object

object

ebtde is returned as an updated esbacktestbyte object. After you run `simulate`, the updated esbacktestbyte object stores the simulated test statistics, which unconditionalDE uses to calculate *p*-values and generate test results.

For more information on the esbacktestbyte object, see esbacktestbyte.

## More About

### Simulation of Test Statistics

The simulation of test statistics requires simulating scenarios of returns, assuming the distribution of returns  $X_t \sim P_t$  is correct (null hypothesis), and computing the corresponding tests statistics for each scenario.

More specifically, the following steps describe the simulation process. The description uses the conditional test statistic  $C_{ES}$  for concreteness, but the same steps apply to the unconditional test statistic  $U_{ES}$ .

- 1 Simulate  $M$  scenarios of returns as

$$X^s = (X_1^s, \dots, X_N^s), \quad s = 1, \dots, M.$$

- 2 Compute the corresponding test statistic as

$$C_{ES}^s = C_{ES}(X_1^s, \dots, X_N^s), \quad s = 1, \dots, M.$$

- 3 Define  $P_C$  as the empirical distribution of the simulated test statistic values as

$$P_C = P[C_{ES} \leq x] = \frac{1}{M} I(C_{ES}^s \leq x),$$

where  $I(\cdot)$  is the indicator function.

To compute the test statistic in step 2, the ranks or mapped returns  $U_t = P_t(X_t)$  need to be computed (see the definition of the test statistics for unconditionalDE and conditionalDE). Assuming that

the model distribution is correct, the ranks  $U_t$  are always uniformly distributed in the unit interval. Therefore, in practice, directly simulating ranks is more efficient than simulating returns and then transforming the returns into ranks.

The `simulate` function implements the simulation process more efficiently as follows:

- 1 Simulated  $M$  scenarios of returns as

$$U^s = (U_1^s, \dots, U_N^s), \quad s = 1, \dots, M,$$

with  $U_t^s \sim \text{Uniform}(0, 1)$ .

- 2 Compute the corresponding test statistic  $C_{ES}$  using the simulated ranks  $U^s$  as

$$C_{ES}^s = C_{ES}(U_1^s, \dots, U_N^s), \quad s = 1, \dots, M.$$

- 3 Define  $P_C$  as the empirical distribution of the simulated test statistic values as

$$P_C = P[C_{ES} \leq x] = \frac{1}{M} I(C_{ES}^s \leq x).$$

After you determine the empirical distribution of the test statistic  $P_C$  in step 3, the significance of the test follows the descriptions provided for `unconditionalDE` and `conditionalDE`. The same steps apply to the unconditional test statistic  $U_{ES}$  and its distribution function  $P_U$ .

## References

- [1] Du, Z., and J. C. Escanciano. "Backtesting Expected Shortfall: Accounting for Tail Risk." *Management Science*. Vol. 63, Issue 4, April 2017.
- [2] Basel Committee on Banking Supervision. "Minimum Capital Requirements for Market Risk". January 2016 (<https://www.bis.org/bcbs/publ/d352.pdf>).

## See Also

`conditionalDE` | `esbacktestbyde` | `esbacktestbysim` | `runtests` | `summary` | `unconditionalDE`

## Topics

- "Workflow for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Rolling Windows and Multiple Models for Expected Shortfall (ES) Backtesting by Du and Escanciano"
- "Expected Shortfall Estimation and Backtesting"
- "Overview of Expected Shortfall Backtesting" on page 2-21
- "ES Backtest Using Du-Escanciano Method" on page 2-24
- "Comparison of ES Backtesting Methods" on page 2-26

## Introduced in R2019b