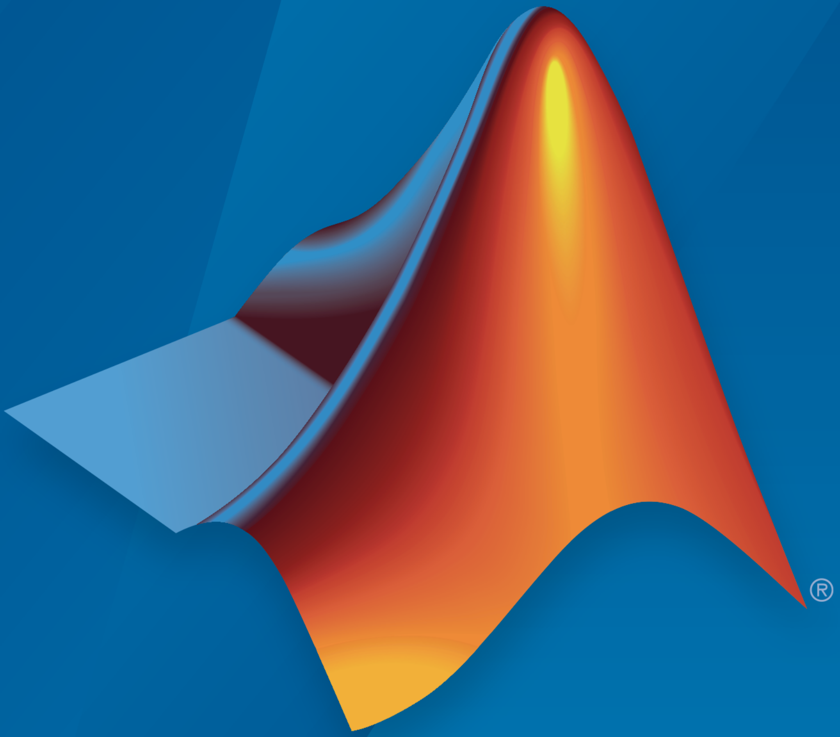
Risk Management Toolbox™

User's Guide



MATLAB®

R2016b

MathWorks®

Risk Management Toolbox™ User's Guide

Revision History

September 2016	Online only	New for Version 1.0 (Release 2016b)

Contents

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

4 Corporate Credit Risk Simulations for Portfolios

5 Functions — Alphabetical List

Getting Started

Risk Management Toolbox Product Description

Develop risk models and perform risk simulation

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

Key Features

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

Risk Modeling with Risk Management Toolbox

In this section...

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. 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 somewhat similar economic conditions.

Risk Management Toolbox provides a simulation framework for credit portfolios, 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*. For more information, see "Credit Simulation Using Copulas" on page 4-2.

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 1-day 95% VaR of a portfolio is 10MM, then there is a 95% chance that the portfolio loses less than 10MM the following day. In other words, only 5% of the time (or about once in 20 days) the portfolio losses exceed 10MM.

Backtesting, on the other hand, measures how accurate the VaR calculations are. For many portfolios, especially trading portfolios, VaR is computed daily. At the closing of the following day, the actual profits and losses for the portfolio are known, and can be compared to the VaR estimated the day before. You can use this daily data to assess the performance of VaR models, which is the goal of VaR backtesting. As such, backtesting is a method that looks retrospectively at data and refines the VaR models. Many VaR backtesting methodologies have been proposed. As a best practice, use more than one criterion to backtest the performance of VaR models, because all tests have strengths and weaknesses.

Risk Management Toolbox provides the following VaR backtesting individual tests:

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

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

See Also

`bin` | `cc` | `cci` | `confidenceBands` | `creditCopula` | `pof` | `portfolioRisk` | `riskContribution` | `runtests` | `simulate` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` | `varbacktest`

Related Examples

- “Common Binning Explorer Tasks” on page 3-4
- “Binning Explorer Case Study Example” on page 3-26
- “Stress Testing of Consumer Credit Default Probabilities Using Panel Data” on page 3-44
- “Credit Simulation Using Copulas” on page 4-2
- “`creditCopula` Simulation Workflow” on page 4-5
- “Modeling Correlated Defaults with Copulas” on page 4-11
- “VaR Backtesting Workflow” on page 2-8
- “Value-at-Risk Estimation and Backtesting” on page 2-13

More About

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

External Websites

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

Market Risk Measurements Using VaR BackTesting Tools

Overview of VaR Backtesting

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

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

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

Suppose that you have VaR limits and corresponding returns or profits and losses for days $t = 1,\ldots,N$. Use VaRt to denote the VaR estimate for day t (determined on day t − 1). Use Rt 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 – VaR level. The binomial test is approximately distributed as a standard normal distribution.

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

Traffic Light Test

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

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

For more information, see Bibliography for Basel Committee on Banking Supervision and `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 $p = 1 -$ VaR level.

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

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

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

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

Christoffersen’s Interval Forecast Tests

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

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

where

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

and

- π_0 — Probability of having a failure on period t, given that no failure occurred on period $t - 1$ = n01 / (n00 + n01)
- π_1 — Probability of having a failure on period t, given that a failure occurred on period $t - 1$ = n11 / (n10 + n11)
- π — Probability of having a failure on period t = (n01 + n11 / (n00 + n01 + n10 + n11)

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

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

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

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

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

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

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

In this statistic, $p = 1 -$ 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:

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

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

References

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

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

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

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

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

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

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

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

See Also

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

Related Examples

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

More About

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

VaR Backtesting Workflow

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

Step 1. Load the VaR backtesting data.

Use the `VaRBacktestData.mat` file to load the VaR data into the workspace. This example works with the `EquityIndex`, `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” for an example on how to generate this VaR data.

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

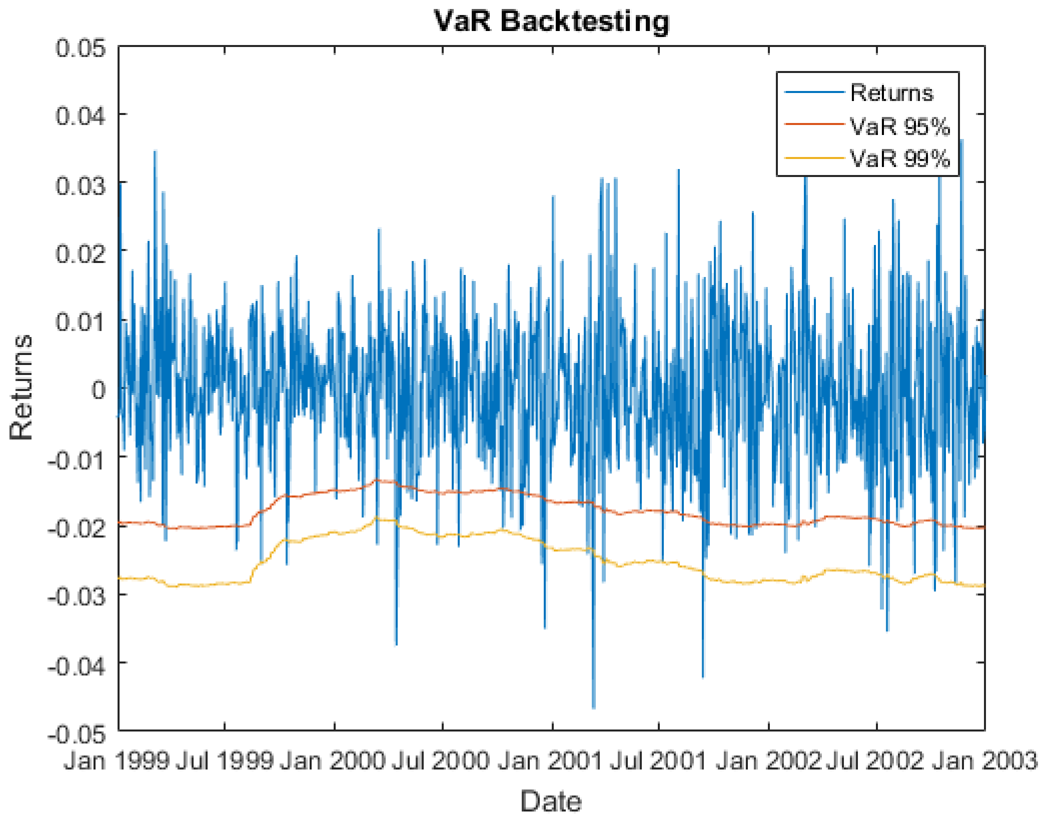
```
   -0.0043    0.0196    0.0277
   -0.0036    0.0195    0.0276
   -0.0000    0.0195    0.0275
    0.0298    0.0194    0.0275
    0.0023    0.0197    0.0278
```

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: [1043×1 double]
          VaRData: [1043×2 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 =

    PortfolioID      VaRID       VaRLevel    ObservedLevel    Observations    Failures
    ___________    __________    ________    _____________    ____________    ________

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

Step 5. Run all tests.

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

```
runtests(vbt)
```

```
ans =

    PortfolioID      VaRID       VaRLevel      TL        Bin       POF       TUFF
    ___________    __________    ________    ______    ______    ______    ______    _

    "S&P"          "Normal95"    0.95        green     accept    accept    accept    ac
    "S&P"          "Normal99"    0.99        yellow    reject    accept    accept    ac
```

Step 6. Run individual tests.

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

```
tl(vbt)
```

```
ans =

    PortfolioID      VaRID        VaRLevel      TL       Probability      TypeI      Increa
    ___________    ___________    ________    ______    ___________    ________    ______

    "S&P"          "Normal95"     0.95        green     0.77913        0.26396
    "S&P"          "Normal99"     0.99        yellow    0.97991        0.03686     0.2658
```

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    Fai
    ________________    ___________    ________    _____________    ____________    ___

    "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)
     PortfolioIDVaRID        VaRLevel      TL       Bin       POF       TUFF
    _________________    __________    ________    ______    ______    ______    ______

    "S&P, 1999-2000"    "Normal95"    0.95        green     accept    accept    accept
    "S&P, 1999-2000"    "Normal99"    0.99        green     accept    accept    accept
    "S&P, 2001-2002"    "Normal95"    0.95        green     accept    accept    accept
    "S&P, 2001-2002"    "Normal99"    0.99        yellow    accept    accept    accept
```

See Also
`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` |
`varbacktest`

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

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

Value-at-Risk Estimation and Backtesting

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

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

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

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

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

Load the Data and Define the Test Window

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

```
load VaRExampleData.mat
Returns = diff(sp)./sp(1:end-1);
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.

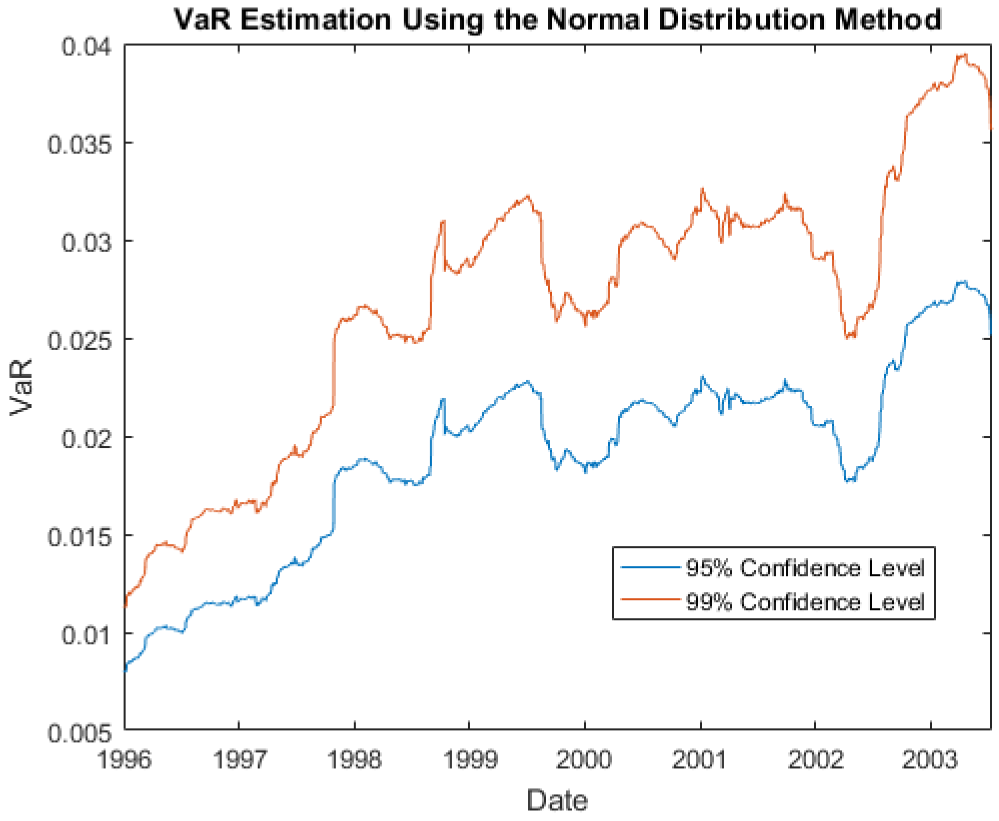
```
pVaR = [0.05 0.01];
```

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')
```

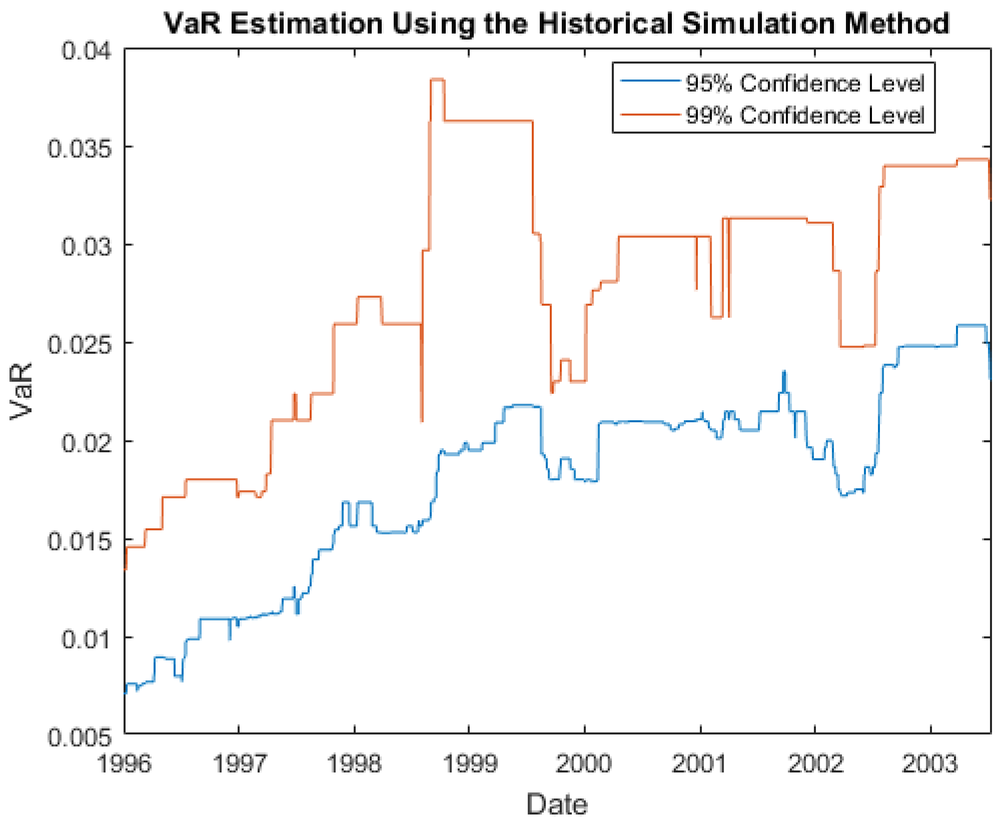


The normal distribution method is also known as parametric VaR because its estimation involves computing a parameter for the standard deviation of the returns. The advantage of the normal distribution method is its simplicity. However, the weakness of the normal distribution method is the assumption 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:

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

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

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

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

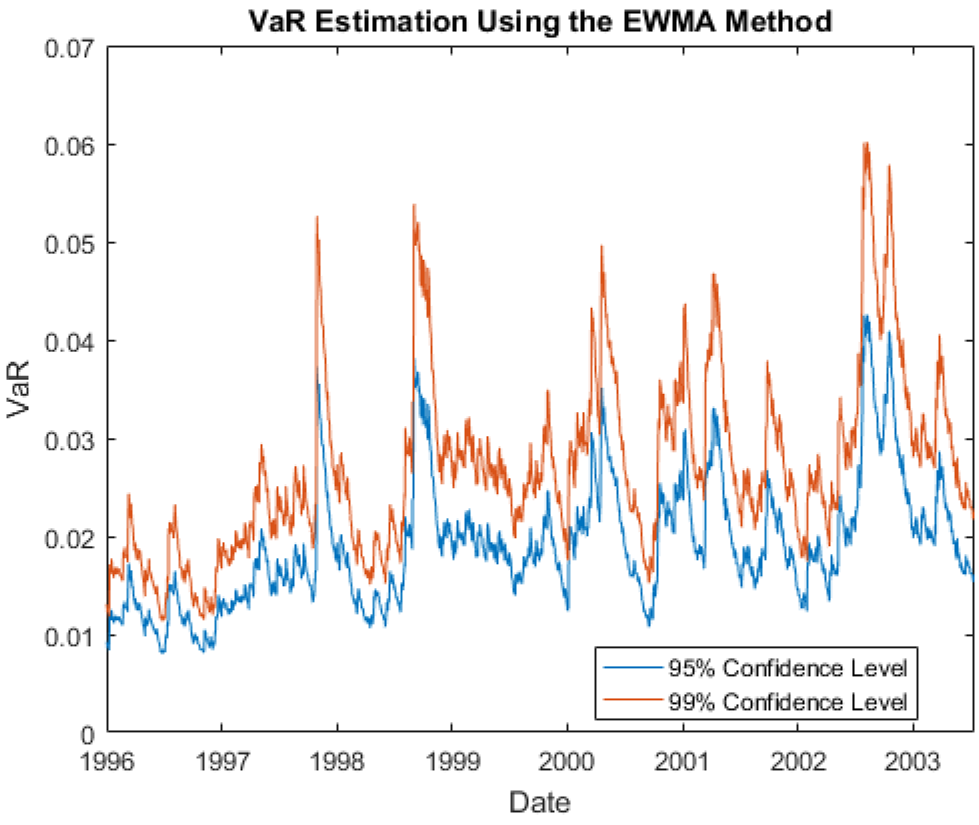
Initiate the EWMA using a warm-up phase to set up the standard deviation.

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

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

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

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

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

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



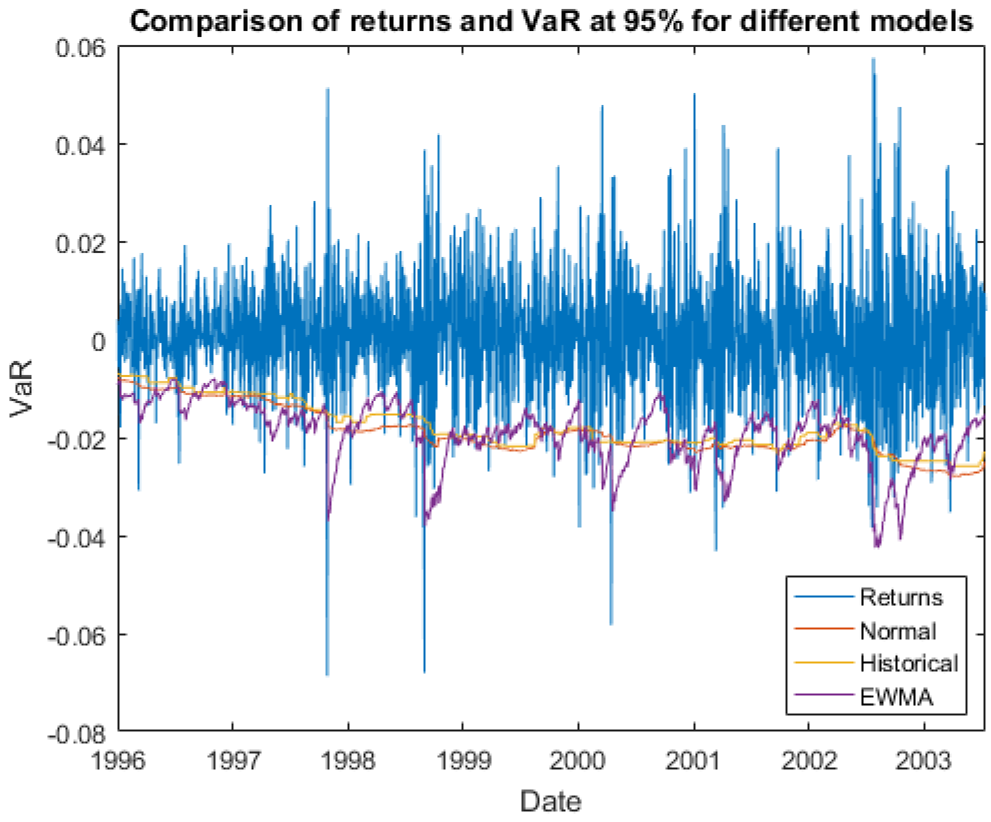
In the preceding figure, the EWMA reacts very quickly to periods of large (or small) returns.

VaR Backtesting

In the first part of this example, VaR was estimated over the test window with three different methods and at two different VaR confidence levels. The goal of VaR backtesting is to evaluate the performance of VaR models. A VaR estimate at 95% confidence is violated only about 5% of the time, and VaR failures do not cluster. Clustering of VaR failures indicates the lack of independence across time because the VaR models are slow to react to changing market conditions.

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

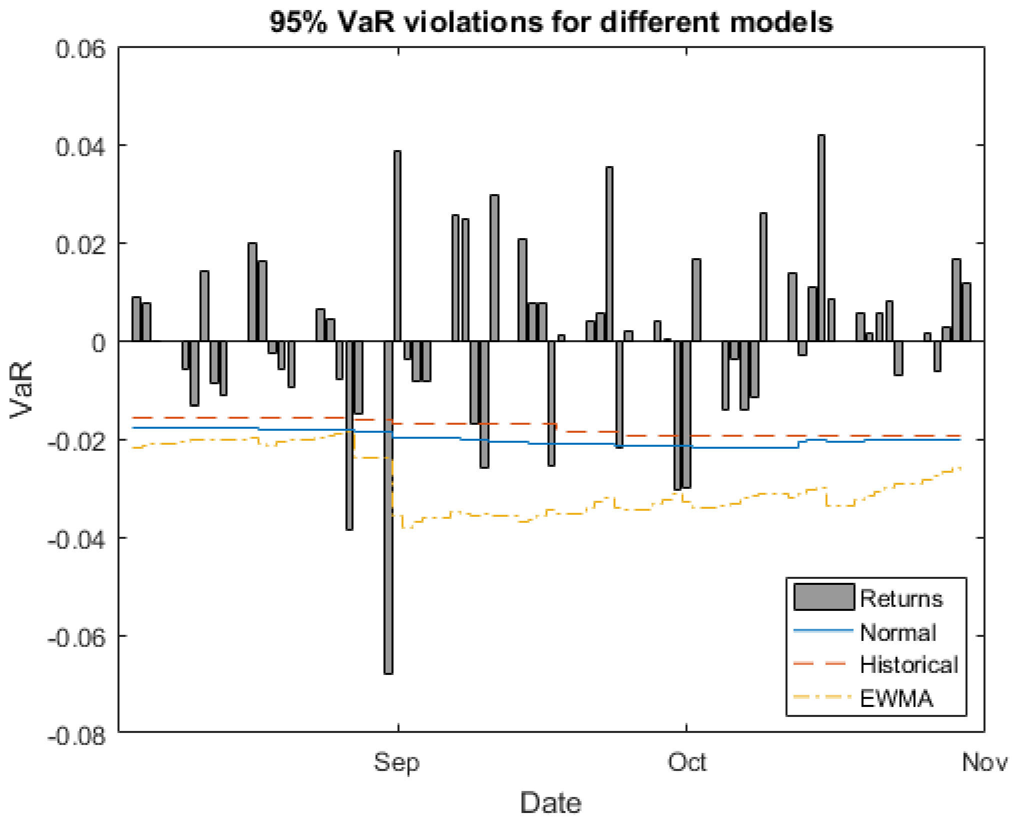
```
ReturnsTest = Returns(TestWindow);
DatesTest   = DateReturns(TestWindow);
figure;
plot(DatesTest,[ReturnsTest -Normal95 -Historical95 -EWMA95])
ylabel('VaR')
xlabel('Date')
legend({'Returns','Normal','Historical','EWMA'},'Location','Best')
title('Comparison of returns and VaR at 95% for different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



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

```
ZoomInd   = (DatesTest >= '5-Aug-1998') & (DatesTest <= '31-Oct-1998');
VaRData   = [-Normal95(ZoomInd) -Historical95(ZoomInd) -EWMA95(ZoomInd)];
VaRFormat = {'-','--','-.'};
figure;
bar(datenum(DatesTest(ZoomInd)),ReturnsTest(ZoomInd),'FaceColor',[0.6 0.6 0.6]);
hold on
for i = 1 : size(VaRData,2)
    stairs(datenum(DatesTest(ZoomInd))-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')
datetick('x','keeplimits');
```



A VaR failure or violation happens when the returns have a negative VaR. A closer look around August 27th to August 31st shows a significant dip in the returns. On the dates starting from August 27th onward, the EWMA follows the trend of the returns closely and more accurately. Consequently, EWMA has fewer VaR violations (two) compared to the normal distribution approach (seven violations) or the historical simulation method (eight violations).

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 = 

    PortfolioID       VaRID        VaRLevel    ObservedLevel    Observations    Failures
    ___________    ___________    ________    _____________    ____________    ________

    "S&P"          "Normal95"     0.95        0.94863          1966            101
    "S&P"          "Normal99"     0.99        0.98372          1966             32
```

The summary report shows that the observed level is close enough to the defined VaR level. The 95% and 99% VaR levels have at most `(1-VaR_level) x` *`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 = 

    PortfolioID       VaRID        VaRLevel      TL        Bin       POF       TUFF      
    ___________    ___________    ________    ______    ______    ______    ______    _
    
    "S&P"          "Normal95"     0.95        green     accept    accept    accept    ac
    "S&P"          "Normal99"     0.99        yellow    reject    reject    accept    re
```

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 =

    PortfolioID        VaRID         VaRLevel     TL       Bin       POF       TUFF
    ___________    ______________    ________    ______    ______    ______    ______

    "S&P"          "Normal95"        0.95        green     accept    accept    accept
    "S&P"          "Historical95"    0.95        yellow    accept    accept    accept
    "S&P"          "EWMA95"          0.95        green     accept    accept    accept
    "S&P"          "Normal99"        0.99        yellow    reject    reject    accept
    "S&P"          "Historical99"    0.99        yellow    reject    reject    accept
    "S&P"          "EWMA99"          0.99        red       reject    reject    accept
```

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

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

```
Ind2000 = (year(DatesTest) == 2000);
vbt2000 = varbacktest(ReturnsTest(Ind2000),[Normal95(Ind2000) Historical95(Ind2000) EWM
   'PortfolioID','S&P, 2000','VaRID',{'Normal','Historical','EWMA'});
runtests(vbt2000)
```

```
ans =

    PortfolioID       VaRID        VaRLevel     TL       Bin       POF       TUFF
    ___________    ____________    ________    _____    ______    ______    ______
    "S&P, 2000"    "Normal"        0.95        green    accept    accept    accept    a
    "S&P, 2000"    "Historical"    0.95        green    accept    accept    accept    a
    "S&P, 2000"    "EWMA"          0.95        green    accept    accept    accept    a
```

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

```
ans =

    PortfolioID       VaRID         VaRLevel      TL        Bin       POF       TUFF
    ___________    ____________    ________    ______    ______    ______    ______

    "S&P, 2002"    "Normal"        0.95        yellow    reject    reject    accept
    "S&P, 2002"    "Historical"    0.95        yellow    reject    accept    accept
    "S&P, 2002"    "EWMA"          0.95        green     accept    accept    accept
```

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

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

```
cci(vbt2002)
```

```
ans =

    PortfolioID       VaRID         VaRLevel     CCI      LRatioCCI    PValueCCI    Obse
    ___________    ____________    ________    ______    _________    _________    ____

    "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 `N00`, `N10`, `N01`, `N11` 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, andquartiles of the distribution of times between failures, in the columns `TBFMin`, `TBFQ1`, `TBFQ2`, `TBFQ3`, `TBFMax`.

```
tbfi(vbt2002)
```

```
ans =

    PortfolioID       VaRID         VaRLevel     TBFI     LRatioTBFI    PValueTBFI     Ob
    ___________    ____________     ________    ______    __________    __________    __

    "S&P, 2002"    "Normal"         0.95        reject    53.936        0.00010087    26
    "S&P, 2002"    "Historical"     0.95        reject    45.274         0.0010127    26
    "S&P, 2002"    "EWMA"           0.95        reject    25.756          0.027796    26
```

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 Practive 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

- "Overview of VaR Backtesting" on page 2-2
- "VaR Backtesting Workflow" on page 2-8

More About

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

Managing Consumer Credit Risk Using the Binning Explorer for Credit Scorecards

Overview of Binning Explorer

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

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

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

Using Binning Explorer:	
1.	Open the **Binning Explorer** app. • MATLAB® Toolstrip: On the **Apps** tab, under **Computational Finance**, click the app icon. • MATLAB Command prompt: Enter `binningExplorer`.
2.	Import the data into the app. You can import data into **Binning Explorer** by either starting directly from a data set or by loading an existing `creditscorecard` object from the MATLAB workspace.
3.	Use **Binning Explorer** to work interactively with the binning assignments for a scorecard.
4.	Export the scorecard to a new `creditscorecard` object. Continue the workflow from the MATLAB command line using `creditscorecard` object functions from Financial Toolbox. For more information, see creditscorecard.
Using `creditscorecard` Object Functions in Financial Toolbox:	
5.	Fit a logistic regression model.
6.	Review and format the credit scorecard points.
7.	Score the data.

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

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

See Also

Apps
Binning Explorer

Functions
`creditscorecard`

Related Examples

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

More About

- “Credit Scorecard Modeling Workflow”

External Websites

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

Common Binning Explorer Tasks

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

In this section...

Import Data

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

Clean Start from Data

To start directly from data for a credit scorecard:

1. Place the credit scorecard data in your MATLAB workspace. The data must be in a MATLAB table, where each column of data can be any one of the following data types:

   - Numeric
   - Logical
   - Cell array of character vectors
   - Character array

- Categorical

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

Start from an Existing `creditscorecard` Object

To start using an existing `creditscorecard` object:

1. Place the `creditscorecard` object in your MATLAB workspace. Create the `creditscorecard` object either by using the `creditscorecard` function or by clicking **Export** in the **Binning Explorer** to export and save a `creditscorecard` object to the MATLAB workspace.
2. Open Binning Explorer from the MATLAB toolstrip: On the **Apps** tab, under **Computational Finance**, click the app icon.
3. From **Step 1** pane of the Import Data window, select the `creditscorecard` object.
4. 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.

Change Predictor Type

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

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

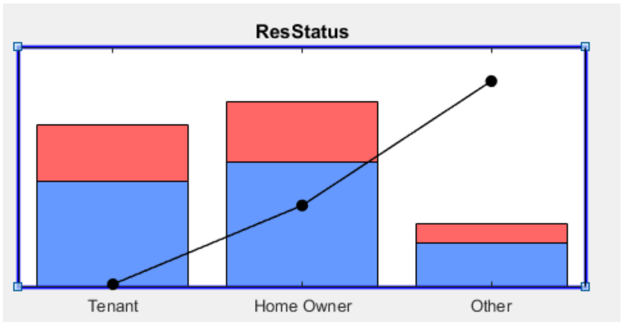
On the **Binning Explorer** toolbar, the predictor type for the selected predictor displays under **Predictor Type**.

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

Change Binning Algorithm for One or More Predictors

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

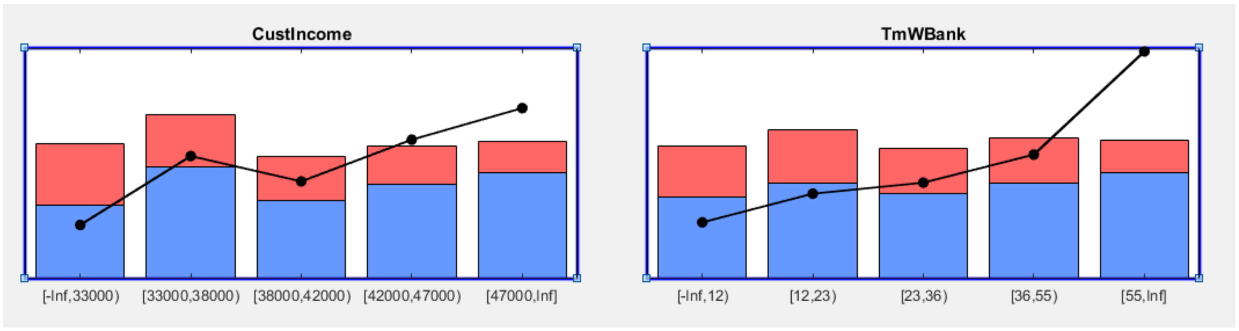
1 Click any predictor plot. The selected predictor plot displays with a blue outline.



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

2 On the **Binning Explorer** toolbar, under **Apply Monotone**, select `Monotone`, `Equal Frequency`, or `Equal Width`. The predictor plot is updated with a change of algorithm. The details in the **Bin Information** and **Predictor Information** panes are also updated.

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

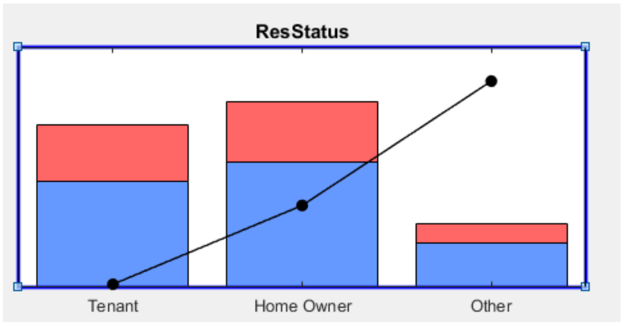


4 Under **Apply Monotone**, select `Monotone`, `Equal Frequency`, or `Equal Width`. All the selected predictor plots are updated for a change of algorithm.

Change Algorithm Options for Binning Algorithms

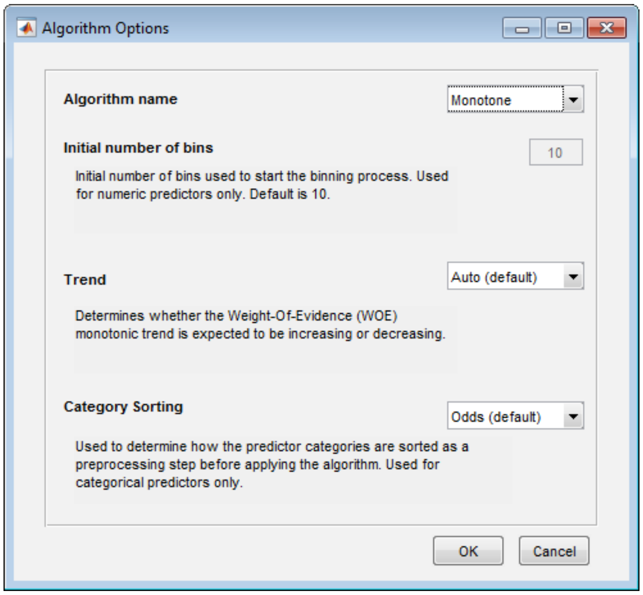
After you import data or a `creditscorecard` object into **Binning Explorer**, you can change can change the binning algorithm for an individual predictor or for multiple predictors.

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



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

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


Algorithm Options

Algorithm name Monotone

Initial number of bins 10

Initial number of bins used to start the binning process. Used for numeric predictors only. Default is 10.

Trend Auto (default)

Determines whether the Weight-Of-Evidence (WOE) monotonic trend is expected to be increasing or decreasing.

Category Sorting Odds (default)

Used to determine how the predictor categories are sorted as a preprocessing step before applying the algorithm. Used for categorical predictors only.

3 From the Algorithm Options dialog box, select an **Algorithm name**:

- `Monotone`

  - For **Initial number of bins**, enter an initial number of bins (default is 10). The initial number of bins must be an integer > 2. Used for numeric predictors only.
  - For **Trend**, select one of the following:

    - **Auto (default)** — Automatically determines if the WOE trend is increasing or decreasing.
    - **Increasing** — Looks for an increasing WOE trend.
    - **Decreasing** — Looks for a decreasing WOE trend.

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

  - For **Category Sorting**, select one of the following:

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

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

- `Equal Frequency`

  - For **Number of bins**, enter the number of bins. The default is `5`, and the number of bins must be a positive number.
  - For **Category Sorting**, select one of the following:

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

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

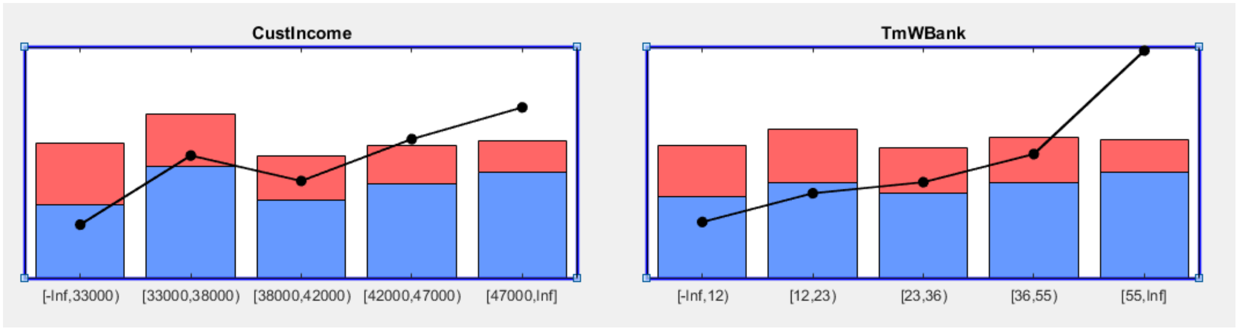
- `Equal Width`

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

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

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

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

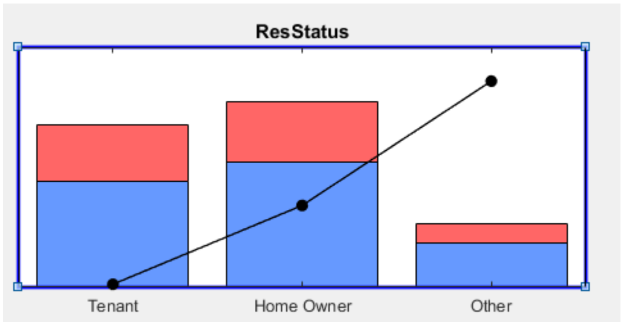


5 On the **Binning Explorer** toolbar, click **Algorithm Options** to open the Algorithm Options dialog box. Make your selection from the Algorithm Options dialog box and click **OK**. The selected predictor plots are updated for the change of algorithm.

Split Bins for a Numeric Predictor

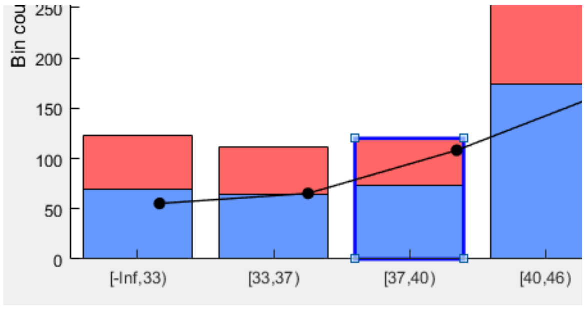
After you import data or a `creditscorecard` object into **Binning Explorer**, you can split bins for a numeric predictor.

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



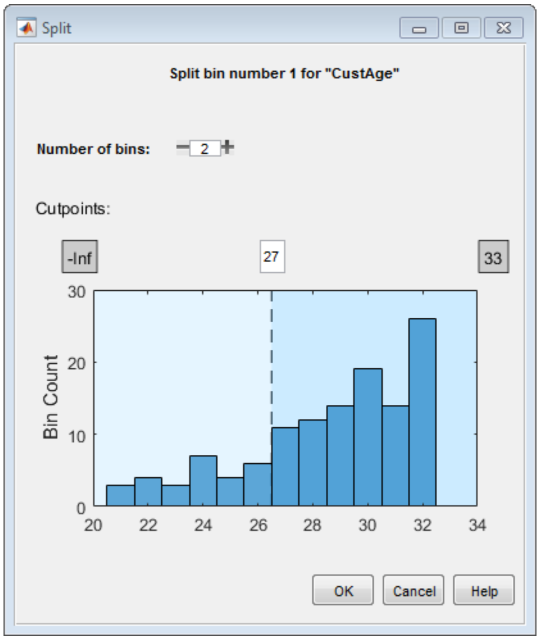
2 On the **Binning Explorer** toolbar, click **Manual Binning** to open the selected numeric predictor in a new tabbed window.

3 Click a bin to enable the **Split** button for that bin.



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

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



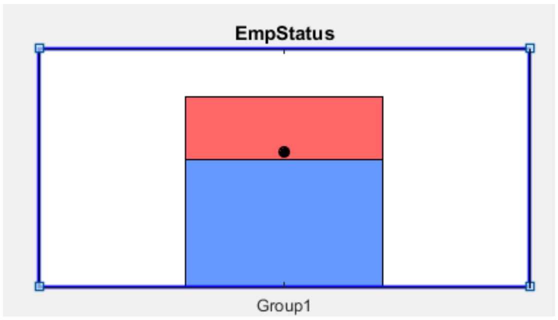
5 Use the **Number of bins** control to split the selected bin into multiple bins. Click **OK** to complete the split operation.

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

Split Bins for a Categorical Predictor

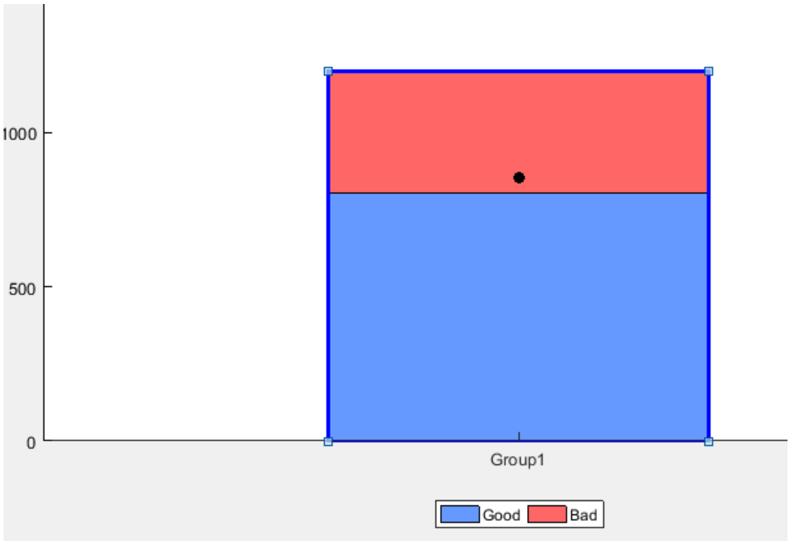
After you import data or a `creditscorecard` object into **Binning Explorer**, you can split bins for a categorical predictor.

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

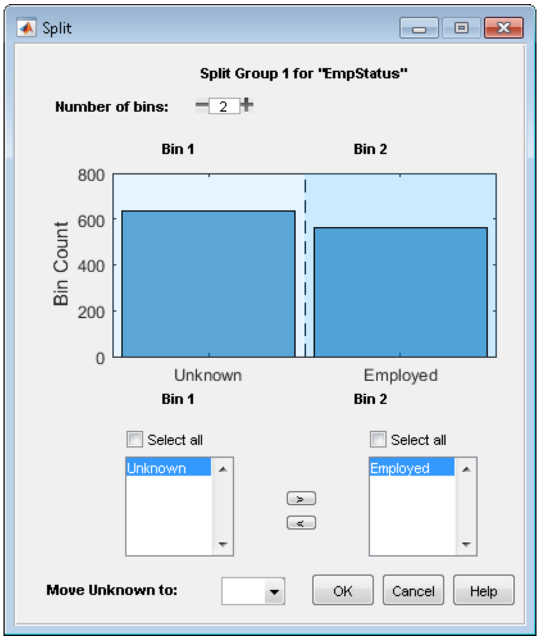


2 On the **Binning Explorer** toolbar, click **Manual Binning** to open the selected categorical predictor in a new tabbed window.

3 Click a bin to enable the **Split** button for that bin.



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



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

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

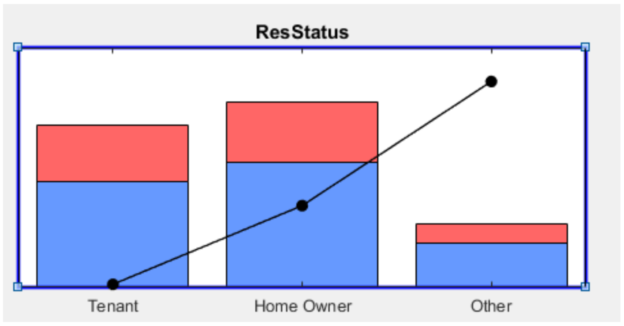
4 Click **OK** to complete the split operation.

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

Manual Binning to Merge Bins for a Numeric or Categorical Predictor

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

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

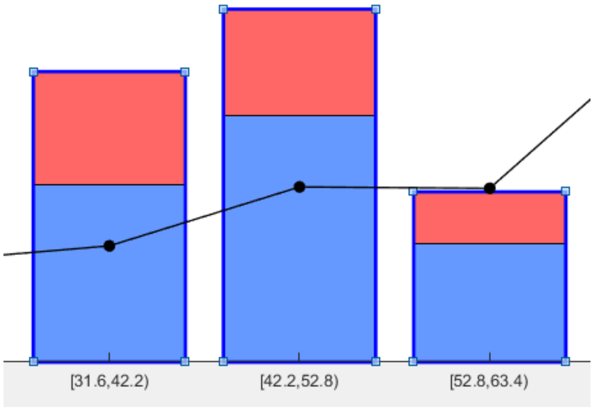


2 On the **Binning Explorer** toolbar, click **Manual Binning** to open the selected predictor in a new tabbed window.

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

---

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



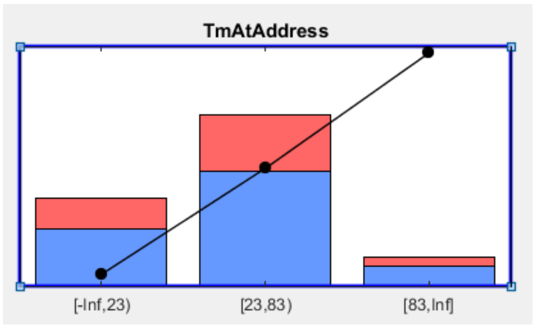
When performing a merge with a numeric predictor, the **Edges** text boxes on the **Binning Explorer** toolbar display the values for the edges of the selected bins to merge.

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

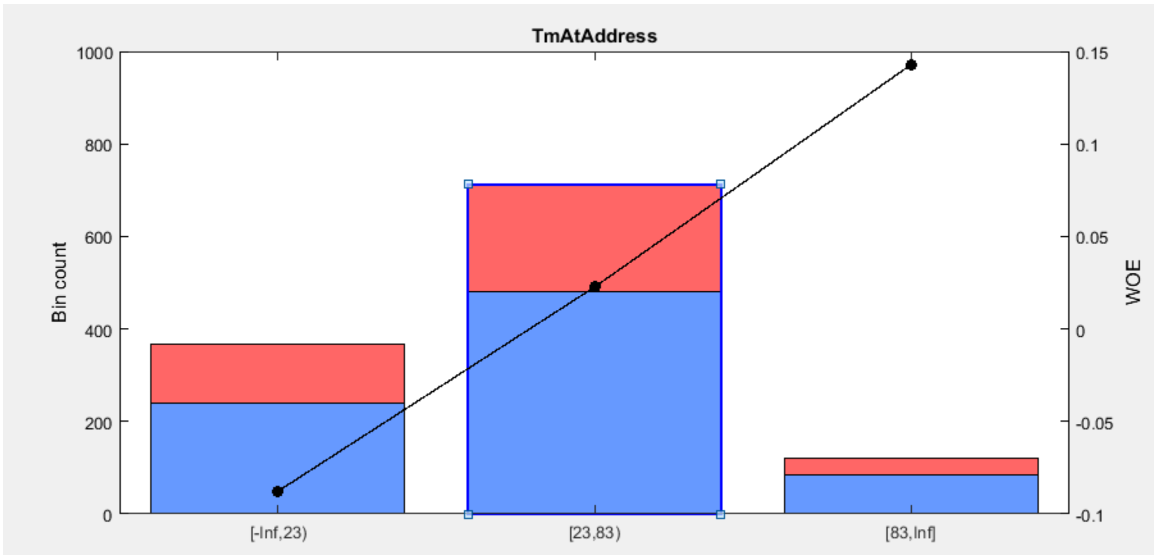
Change Bin Boundaries for a Single Predictor

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

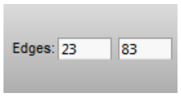
1 Click any predictor plot. The predictor plot displays with a blue outline.



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



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


Edges: 23 83

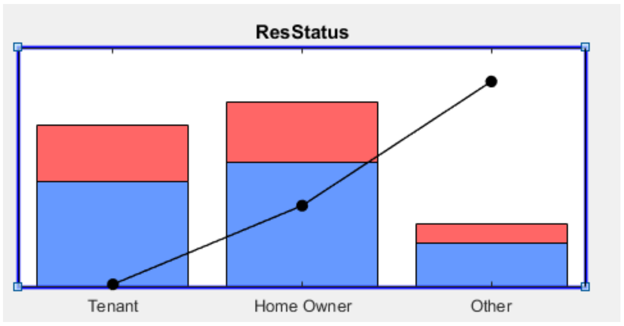
Edit the values in the **Edges** text boxes to change the selected bin's dimensions.

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

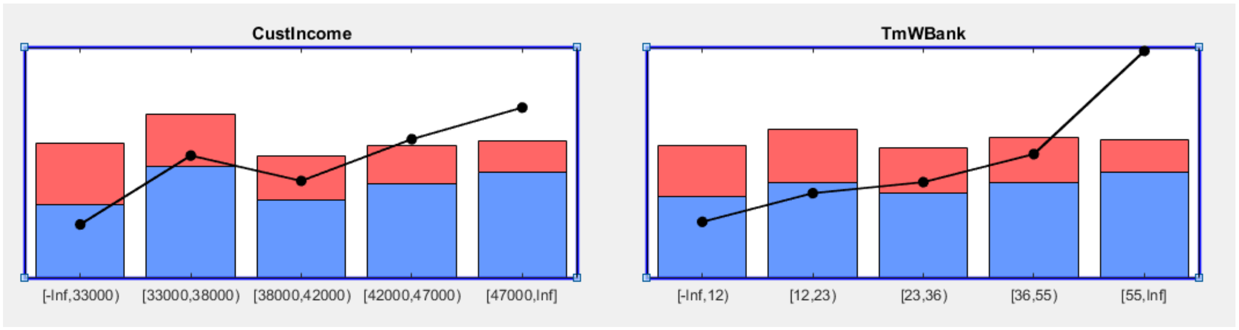
Change Bin Boundaries for Multiple Predictors

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

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



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



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

3 From the Algorithm Options dialog box, under **Algorithm name**, select a binning algorithm.

- For the `EqualWidth` and `EqualFrequency` options, enter a number in the **Number of bins** box. Optionally, for `EqualWidth` and `EqualFrequency` options, under **Category Sorting**, specify the type of sorting.
- For the `Monotone` option, the default of `10` is used for the **Initial number of bins**. Optionally, you can set values for **Trend** and **Category Sorting**.

4 Click **OK** to complete the operation. The plots for the selected predictors are updated with the new bin information. The details in the **Bin Information** and **Predictor Information** panes are also updated.

Set Options for Display

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

Plot Options

1 From the **Binning Explorer** toolbar item for **Plot Options**, select any of the following predictor plot options:

- **No labels** (default)
- **Bin count**
- **% Bin level**
- **% Data level**
- **% Total count**
- **WOE curve**

2 The selected label is applied to all predictor plots.

Table Options

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

1 From the **Binning Explorer** toolbar item for **Table Columns**, select any of the following options:

- **Odds**
- **WOE**
- **InfoValue**
- **Entropy**
- **Members** (option is enabled for categorical predictors)

2 When selected, these options are applied to all predictors for the information displayed in **Bin Information**.

Export and Save the Binning

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

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

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

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

Troubleshoot the Binning

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

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

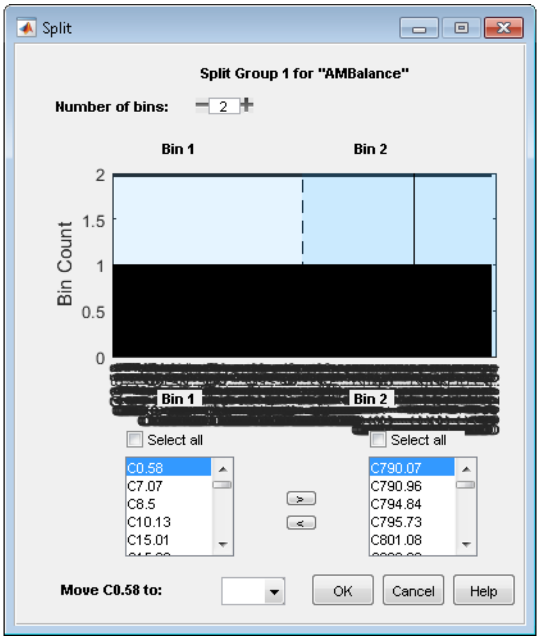
Numeric Predictor Converted to Categorical Predictor Does Not Display Split Data Properly

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

```
load CreditCardData
```

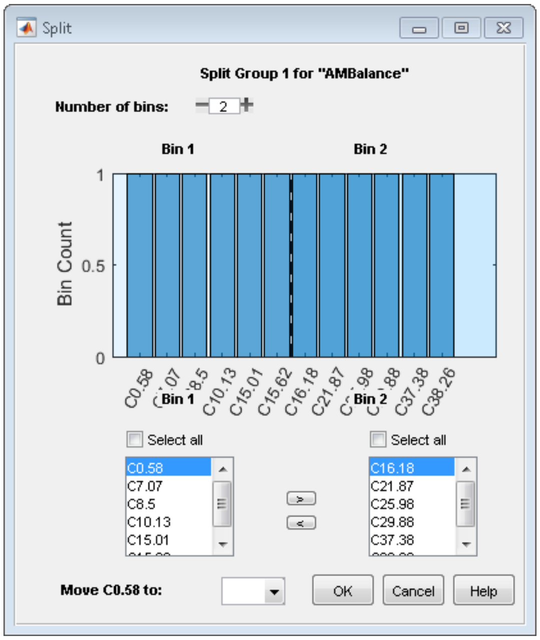
Open the **Binning Explorer** and select the numeric predictor **AMBalance**. From the Binning Explorer toolbar, change the predictor type to **Categorical**.

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



The predictor has too many categories to display properly.

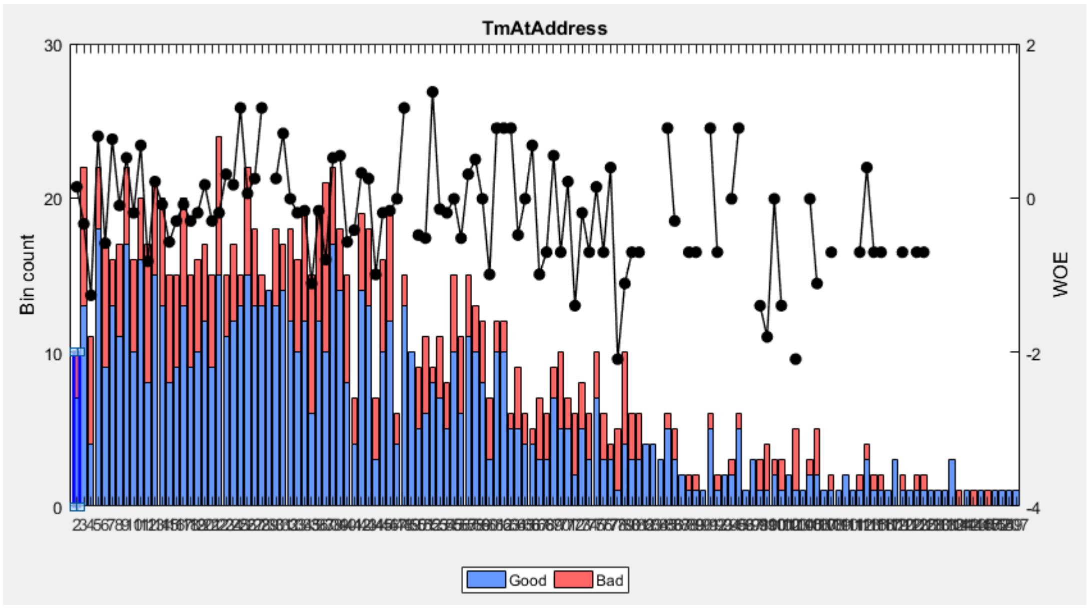
Solution: If you have a categorical predictor with a large number of categories, use the **Algorithm Options** to change the binning algorithm for that predictor to **Equal Frequency**, with the **Number of bins** set to 100 (or another smaller value). The Split dialog box then displays properly.



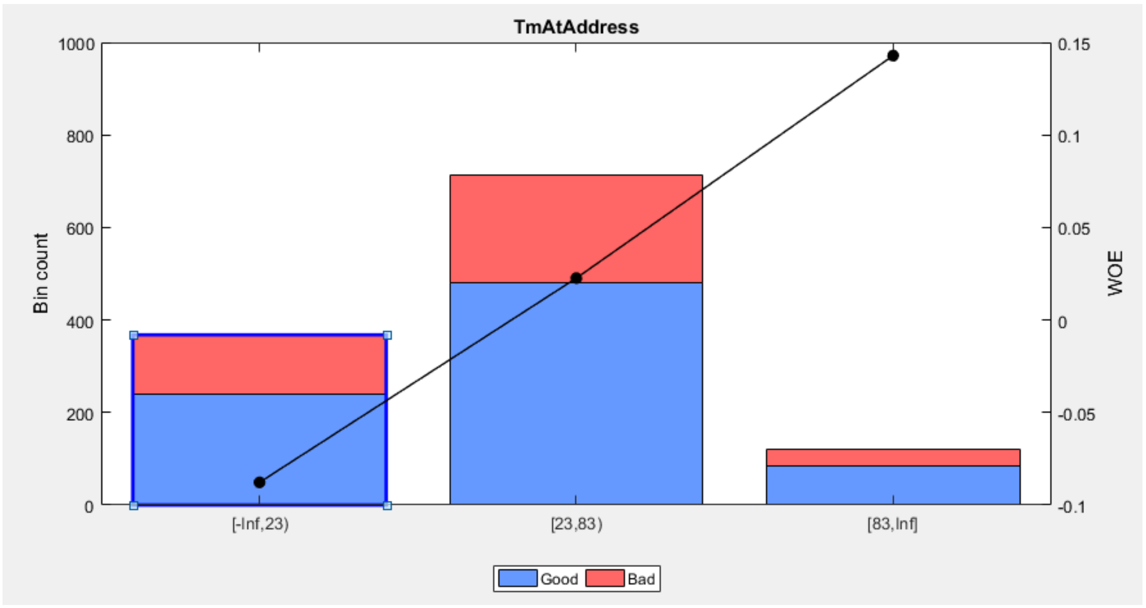
Predictor Plot Appears Distorted

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

```
load CreditCardData
```



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



See Also

Apps

Binning Explorer

Functions

`creditscorecard`

Related Examples

- "Binning Explorer Case Study Example" on page 3-26
- "Case Study for a Credit Scorecard Analysis"

More About

- "Overview of Binning Explorer" on page 3-2
- "Credit Scorecard Modeling Workflow"

External Websites

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

Binning Explorer Case Study Example

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

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

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

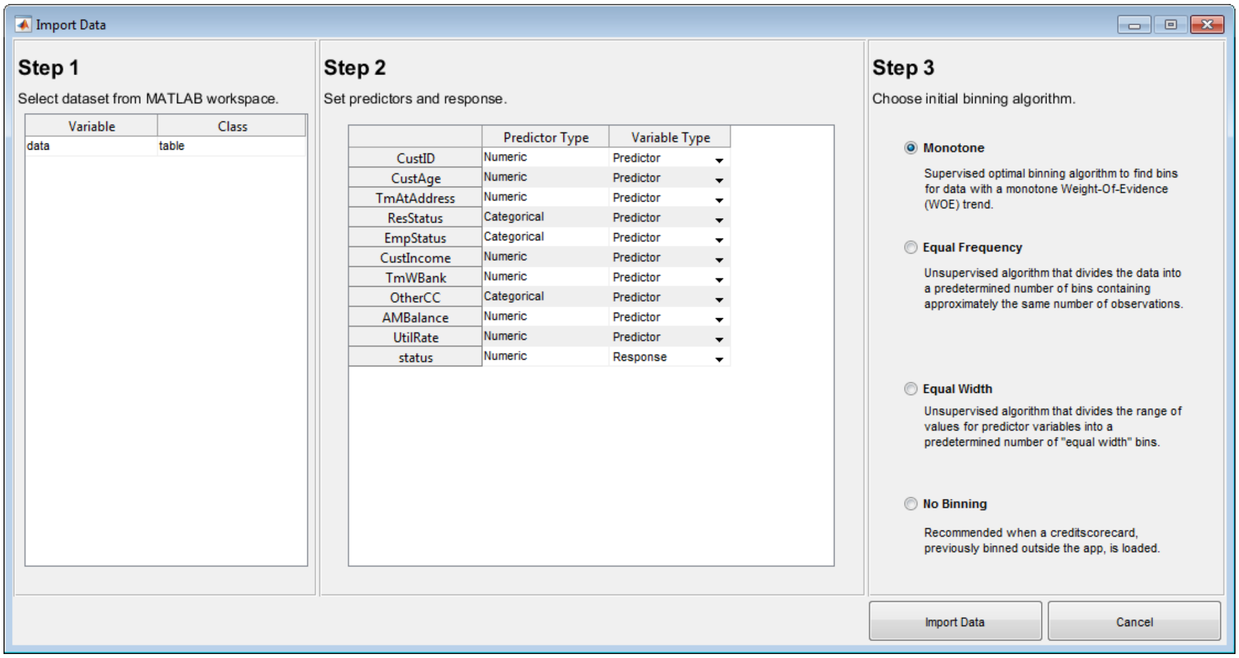
```
load CreditCardData
disp(data(1:10,:))
```

```
    CustID    CustAge    TmAtAddress     ResStatus     EmpStatus    CustIncome    TmWBank    OtherCC    AMBalance    UtilRate
    ______    _______    ___________    __________    _________    __________    _______    _______    _________    ______

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

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

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

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



Step 1

Select dataset from MATLAB workspace.

Variable	Class
data	table

Step 2

Set predictors and response.

	Predictor Type	Variable Type
CustID	Numeric	Predictor
CustAge	Numeric	Predictor
TmAtAddress	Numeric	Predictor
ResStatus	Categorical	Predictor
EmpStatus	Categorical	Predictor
CustIncome	Numeric	Predictor
TmWBank	Numeric	Predictor
OtherCC	Categorical	Predictor
AMBalance	Numeric	Predictor
UtilRate	Numeric	Predictor
status	Numeric	Response

Step 3

Choose initial binning algorithm.

[✓ **Monotone**]
Supervised optimal binning algorithm to find bins for data with a monotone Weight-Of-Evidence (WOE) trend.

Equal Frequency
Unsupervised algorithm that divides the data into a predetermined number of bins containing approximately the same number of observations.

Equal Width
Unsupervised algorithm that divides the range of values for predictor variables into a predetermined number of "equal width" bins.

No Binning
Recommended when a creditscorecard, previously binned outside the app, is loaded.

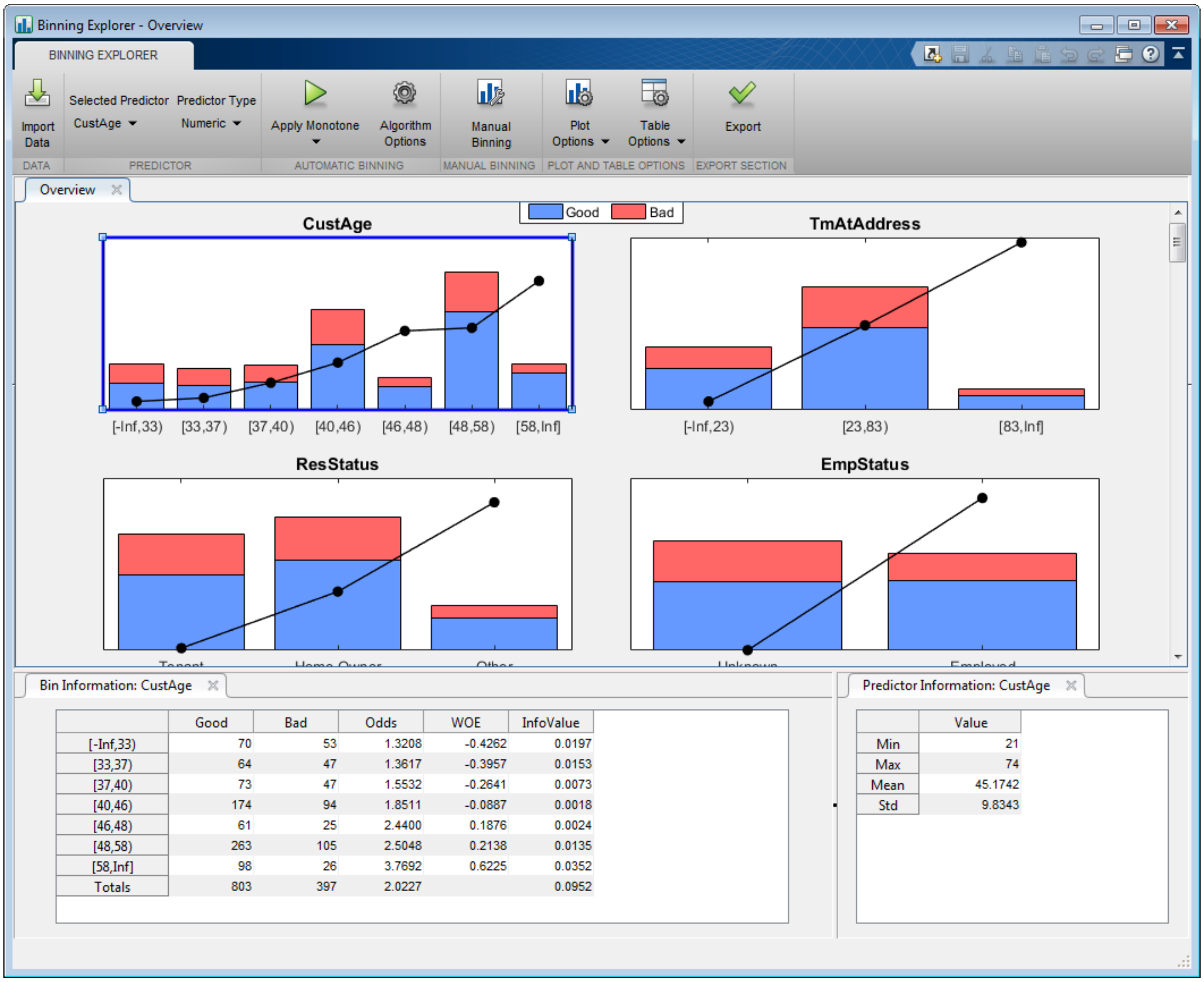
Under **Step 1**, select `data`.

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

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

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

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



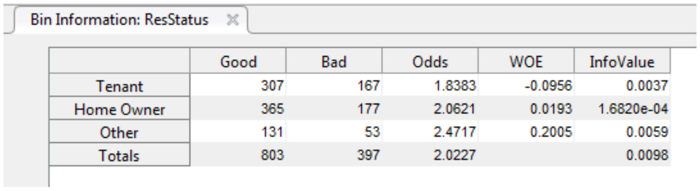
	Good	Bad	Odds	WOE	InfoValue
[-Inf,33)	70	53	1.3208	-0.4262	0.0197
[33,37)	64	47	1.3617	-0.3957	0.0153
[37,40)	73	47	1.5532	-0.2641	0.0073
[40,46)	174	94	1.8511	-0.0887	0.0018
[46,48)	61	25	2.4400	0.1876	0.0024
[48,58)	263	105	2.5048	0.2138	0.0135
[58,Inf]	98	26	3.7692	0.6225	0.0352
Totals	803	397	2.0227		0.0952

	Value
Min	21
Max	74
Mean	45.1742
Std	9.8343

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.

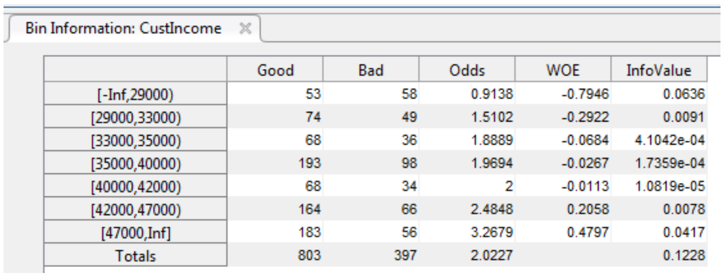
Click the **ResStatus** plot. The **Bin Information** pane contains the “Good” and “Bad” frequencies and other bin statistics such as weight of evidence (WOE).



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**.

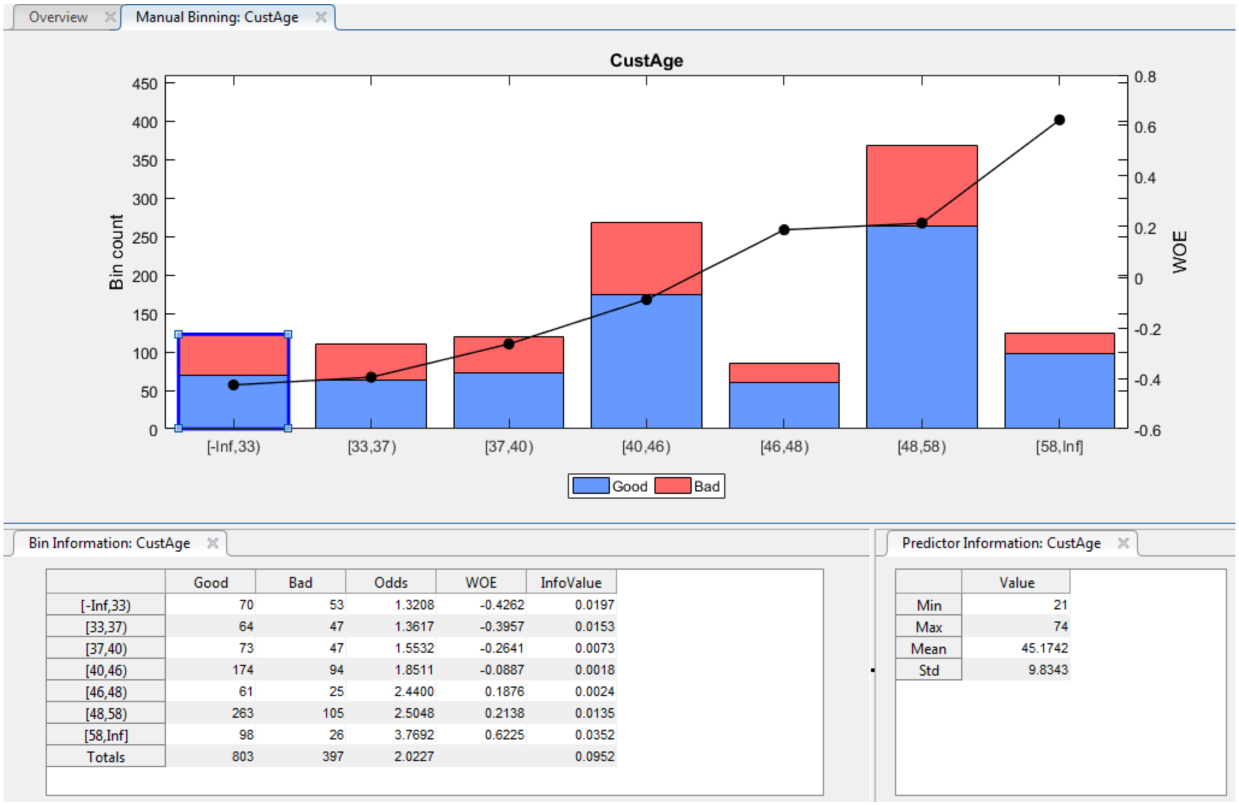


Bin Information: CustIncome

	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.

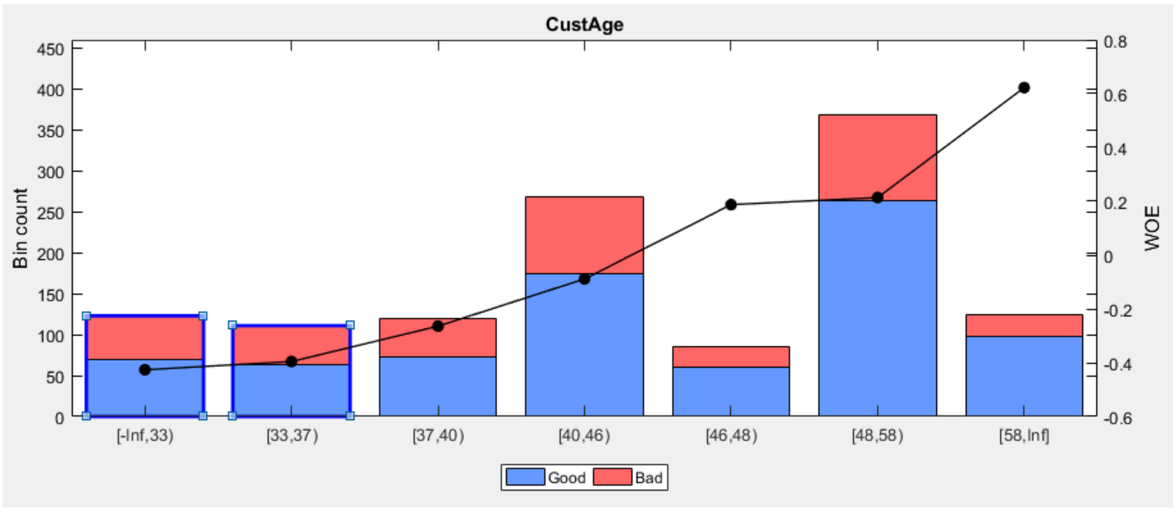
Click the **CustAge** predictor plot. Notice that bins 1 and 2 have similar WOEs, as do bins 5 and 6.



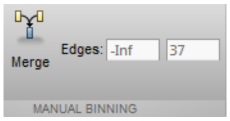
	Good	Bad	Odds	WOE	InfoValue
[-Inf,33)	70	53	1.3208	-0.4262	0.0197
[33,37)	64	47	1.3617	-0.3957	0.0153
[37,40)	73	47	1.5532	-0.2641	0.0073
[40,46)	174	94	1.8511	-0.0887	0.0018
[46,48)	61	25	2.4400	0.1876	0.0024
[48,58)	263	105	2.5048	0.2138	0.0135
[58,Inf]	98	26	3.7692	0.6225	0.0352
Totals	803	397	2.0227		0.0952

	Value
Min	21
Max	74
Mean	45.1742
Std	9.8343

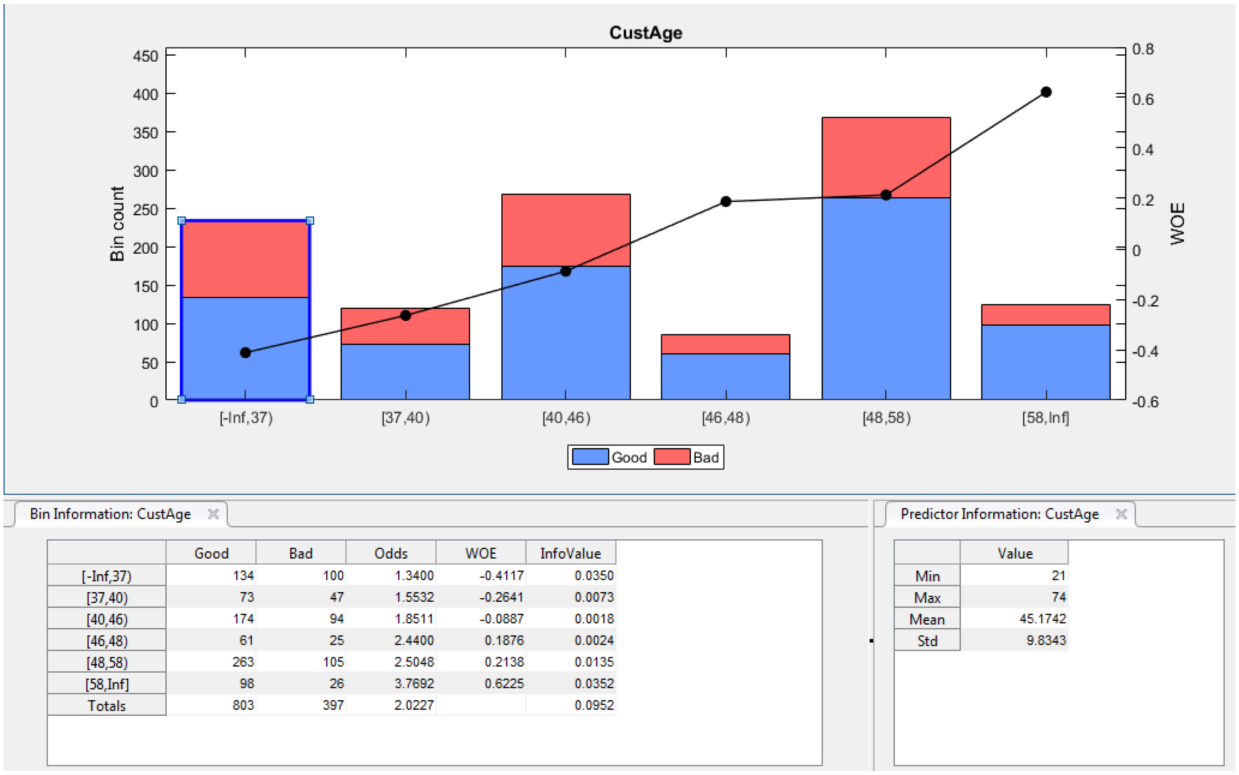
To merge bins 1 and 2, from the **Binning Explorer** toolbar, click **Manual Binning** to open the selected predictor in a new tabbed window. Alternatively, double-click the predictor plot to open the **Manual Binning** tab. Select bin 1 and 2 for merging by using **Ctrl** + click to multiselect these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



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



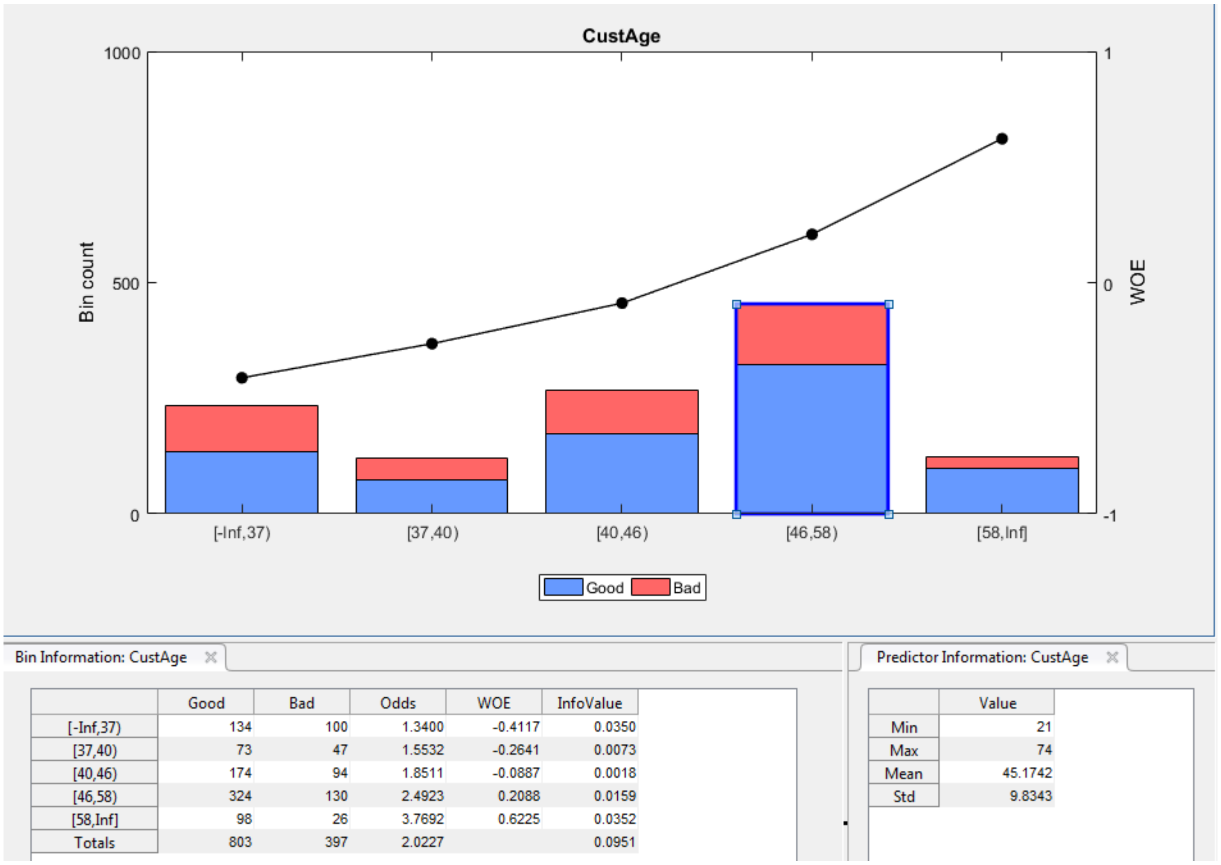
Click **Merge** to finish merging bins 1 and 2. The **CustAge** predictor plot is updated for the new bin information and the details in the **Bin Information** and **Predictor Information** panes are also updated.



	Good	Bad	Odds	WOE	InfoValue
[-Inf,37)	134	100	1.3400	-0.4117	0.0350
[37,40)	73	47	1.5532	-0.2641	0.0073
[40,46)	174	94	1.8511	-0.0887	0.0018
[46,48)	61	25	2.4400	0.1876	0.0024
[48,58)	263	105	2.5048	0.2138	0.0135
[58,Inf]	98	26	3.7692	0.6225	0.0352
Totals	803	397	2.0227		0.0952

	Value
Min	21
Max	74
Mean	45.1742
Std	9.8343

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



	Good	Bad	Odds	WOE	InfoValue
[-Inf,37)	134	100	1.3400	-0.4117	0.0350
[37,40)	73	47	1.5532	-0.2641	0.0073
[40,46)	174	94	1.8511	-0.0887	0.0018
[46,58)	324	130	2.4923	0.2088	0.0159
[58,Inf]	98	26	3.7692	0.6225	0.0352
Totals	803	397	2.0227		0.0951

	Value
Min	21
Max	74
Mean	45.1742
Std	9.8343

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 WOEs:

- 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 `creditscorecard` object from Binning Explorer.

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

Step 5. Fit a logistic regression model.

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

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

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

Generalized Linear regression model:
    logit(status) ~ 1 + CustAge + ResStatus + EmpStatus + CustIncome + TmWBank + OtherCC + AMBalance
    Distribution = Binomial

Estimated Coefficients:
                   Estimate      SE       tStat       pValue  
                   ________    _______    ______    __________

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


1200 observations, 1192 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 89.7, p-value = 1.42e-16
```

Step 6. Review and format scorecard points.

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

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

```
disp(p1)

    Predictors            Bin              Points  
    ____________    __________________    _________

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

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

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

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

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

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

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

```
     Predictors             Bin               Points  
    ____________    ___________________    _________

    'CustAge'       'Up to 36'              -0.15314
    'CustAge'       '37 to 39'             -0.062247
    'CustAge'       '40 to 45'              0.045763
    'CustAge'       '46 to 57'               0.22888
    'CustAge'       '58 and up'              0.48354
    'ResStatus'     'Tenant'               -0.031302
    'ResStatus'     'Home Owner'             0.12697
    'ResStatus'     'Other'                   0.37652
    'EmpStatus'     'Unknown'                -0.076369
    'EmpStatus'     'Employed'                0.31456
    'CustIncome'    'Up to 28999'            -0.45455
    'CustIncome'    '29000 to 32999'          -0.1037
    'CustIncome'    '33000 to 41999'         0.077768
    'CustIncome'    '42000 to 46999'          0.24406
    'CustIncome'    '47000 and up'            0.43536
    'TmWBank'       'Up to 11'               -0.18221
    'TmWBank'       '12 to 44'              -0.038279
    'TmWBank'       '45 to 70'                0.39569
    'TmWBank'       '71 and up'               0.95074
    'OtherCC'       'No'                       -0.193
    'OtherCC'       'Yes'                     0.15868
    'AMBalance'     'Up to 558.87'             0.3552
    'AMBalance'     '558.88 to 1597.43'     -0.026797
    'AMBalance'     '1597.44 and up'         -0.21168
```

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

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

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

```
    Predictors             Bin             Points
    ____________    ___________________    ______

    'CustAge'       'Up to 36'             53.239
    'CustAge'       '37 to 39'             59.796
    'CustAge'       '40 to 45'             67.587
    'CustAge'       '46 to 57'             80.796
    'CustAge'       '58 and up'            99.166
    'ResStatus'     'Tenant'               62.028
    'ResStatus'     'Home Owner'           73.445
    'ResStatus'     'Other'                91.446
    'EmpStatus'     'Unknown'              58.777
    'EmpStatus'     'Employed'             86.976
    'CustIncome'    'Up to 28999'          31.497
    'CustIncome'    '29000 to 32999'       56.805
    'CustIncome'    '33000 to 41999'       69.896
    'CustIncome'    '42000 to 46999'       81.891
    'CustIncome'    '47000 and up'          95.69
    'TmWBank'       'Up to 11'             51.142
    'TmWBank'       '12 to 44'             61.524
    'TmWBank'       '45 to 70'             92.829
    'TmWBank'       '71 and up'            132.87
    'OtherCC'       'No'                   50.364
    'OtherCC'       'Yes'                  75.732
    'AMBalance'     'Up to 558.87'         89.908
    'AMBalance'     '558.88 to 1597.43'    62.353
    'AMBalance'     '1597.44 and up'       49.016
```

Step 7. Score the data.

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

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

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

    CustAge    ResStatus    EmpStatus    CustIncome    TmWBank    OtherCC    AMBalance
    _______    _________    _________    __________    _______    _______    _________

    80.796     62.028       58.777        95.69        92.829     75.732     62.353   
    99.166     73.445       86.976        95.69        61.524     75.732     62.353   
    80.796     62.028       86.976       69.896        92.829     50.364     62.353   
    80.796     73.445       86.976        95.69        61.524     75.732     89.908   
    99.166     73.445       86.976        95.69        61.524     75.732     62.353   
    99.166     73.445       86.976        95.69        92.829     75.732     62.353   
    53.239     73.445       58.777       56.805        61.524     75.732     62.353   
    80.796     91.446       86.976        95.69        61.524     50.364     49.016   
    80.796     62.028       58.777        95.69        61.524     75.732     89.908   
    80.796     73.445       58.777        95.69        61.524     75.732     62.353   
```

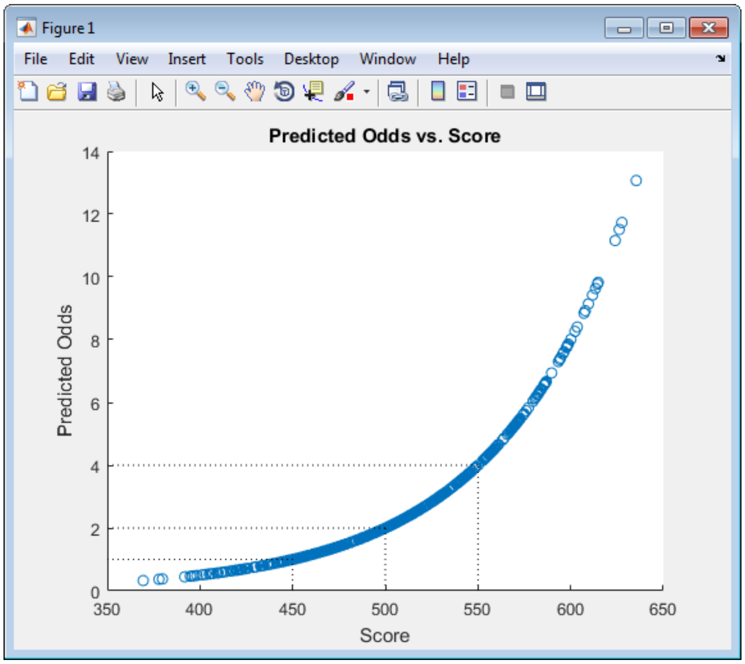
Step 8. Calculate the probability of default.

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

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

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

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

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

hold on

xLimits = xlim;
yLimits = ylim;

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

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

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

hold off
```

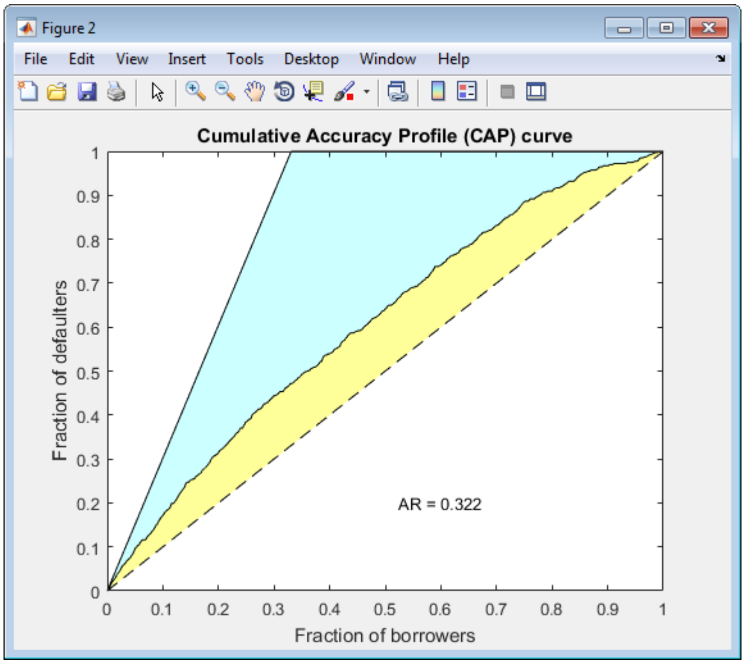


Step 9. Validate the credit scorecard model using the CAP, ROC, and Kolmogorov-Smirnov statistic

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

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

```
          Measure                Value  
    ______________________    _______

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

    Scores    ProbDefault    TrueBads    FalseBads    TrueGoods    FalseGoods    Sensitivity    FalseAlarm      PctObs  
    ______    ___________    ________    _________    _________    __________    ___________    __________    __________

     369.4      0.7535          0            1           802          397                  0    0.0012453     0.00083333
    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
```



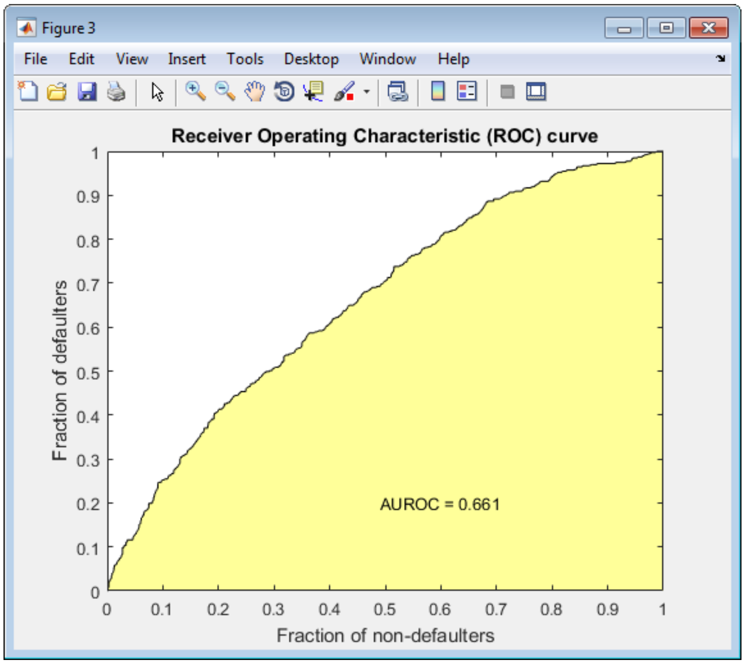
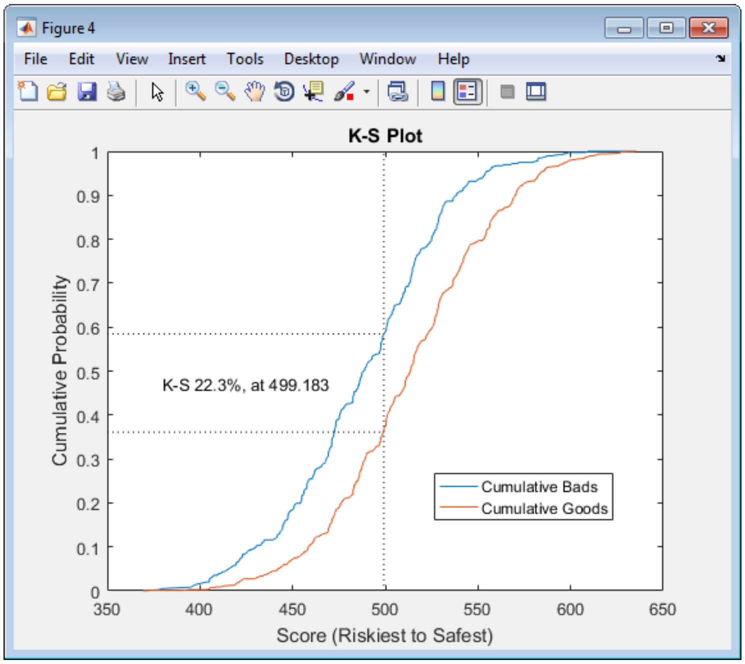


Figure 3
File Edit View Insert Tools Desktop Window Help
Receiver Operating Characteristic (ROC) curve
AUROC = 0.661
Fraction of defaulters
Fraction of non-defaulters
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1



See Also

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

Related Examples

- "Troubleshooting Credit Scorecard Results"
- "Credit Rating by Bagging Decision Trees"
- "Stress Testing of Consumer Credit Default Probabilities Using Panel Data" on page 3-44

More About

- "About Credit Scorecards"
- "Credit Scorecard Modeling Workflow"
- Monotone Adjacent Pooling Algorithm (MAPA)

- creditscorecard

External Websites

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

Stress Testing of Consumer Credit Default Probabilities Using Panel Data

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

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

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

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

Panel Data Description

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

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

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

- `Year`: Calendar year.

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

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

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

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

Load the Panel Data

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

```
load RetailCreditPanelData.mat

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

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

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

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

```
First ten rows:
    ID    ScoreGroup     YOB    Default    Year
    __    ___________    ___    _______    ____

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

Last ten rows:
     ID       ScoreGroup     YOB    Default    Year
    _____     ___________    ___    _______    ____

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

Total number of IDs: 96820
Total number of rows: 646724
```

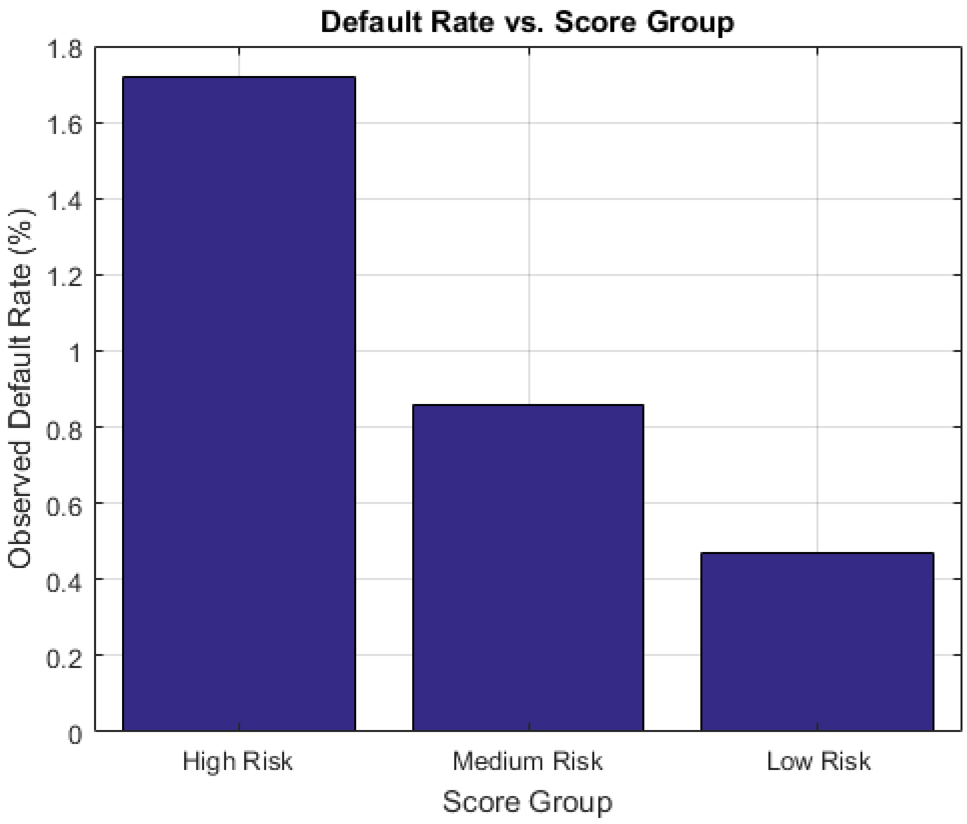
Default Rates by Score Groups and Years on Books

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

```
DefRateByScore = varfun(@mean,data,'InputVariables','Default',...
   'GroupingVariables','ScoreGroup');
NumScoreGroups = height(DefRateByScore);

disp(DefRateByScore)

figure;
bar(double(DefRateByScore.ScoreGroup),DefRateByScore.mean_Default*100)
set(gca,'XTickLabel',categories(data.ScoreGroup))
```

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

```
    ScoreGroup      GroupCount     mean_Default
    ___________     __________     ____________

    High Risk       2.0999e+05       0.017167
    Medium Risk     2.1743e+05      0.0086006
    Low Risk         2.193e+05      0.0046784
```

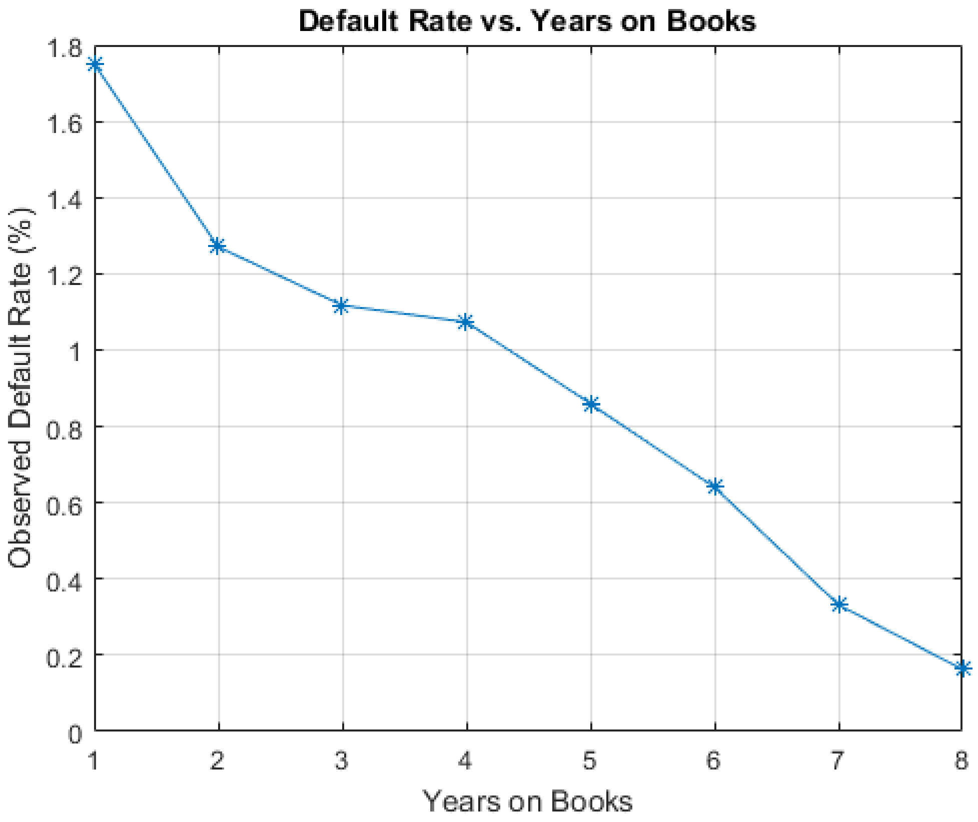


Next, compute default rates grouping by years on books (represented by the `YOB` variable). The resulting rates are conditional one-year default rates. For example, the default rate for the third year on books is the proportion of loans defaulting in the third year, relative to the number of loans that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

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

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

disp(DefRateByYOB)

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

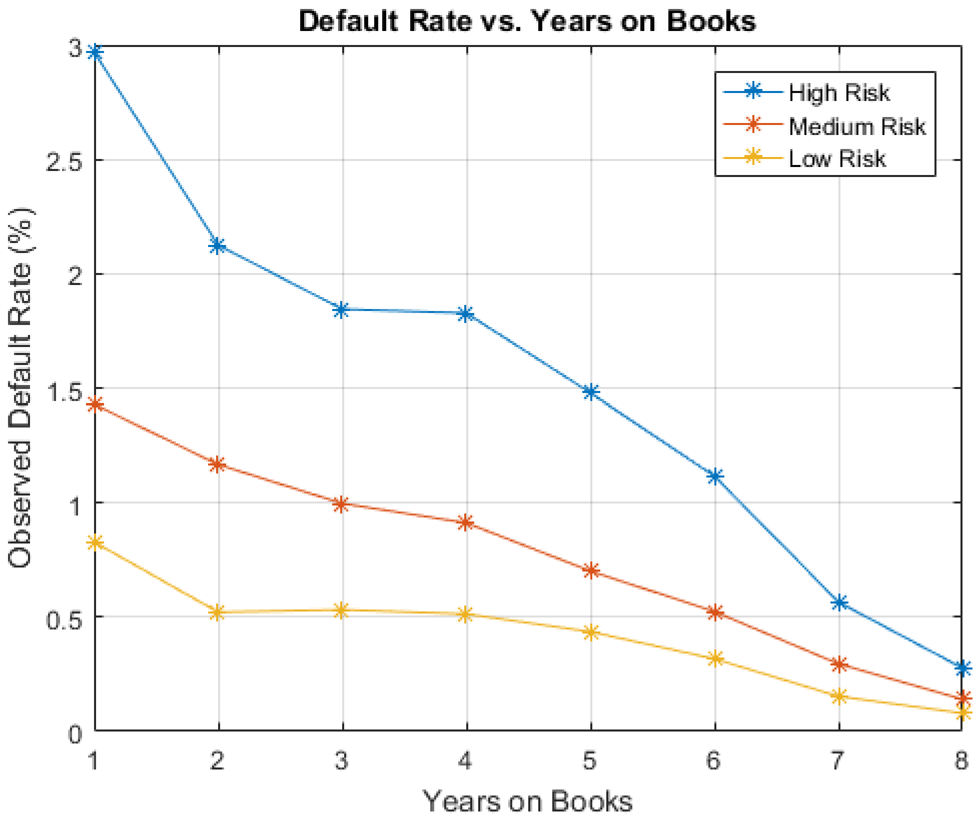
```
    YOB     GroupCount     mean_Default
    ___     __________     ____________

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



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

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

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

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

```
    ScoreGroup      YOB    GroupCount    mean_Default
    ___________     ___    __________    ____________

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

      ...
```



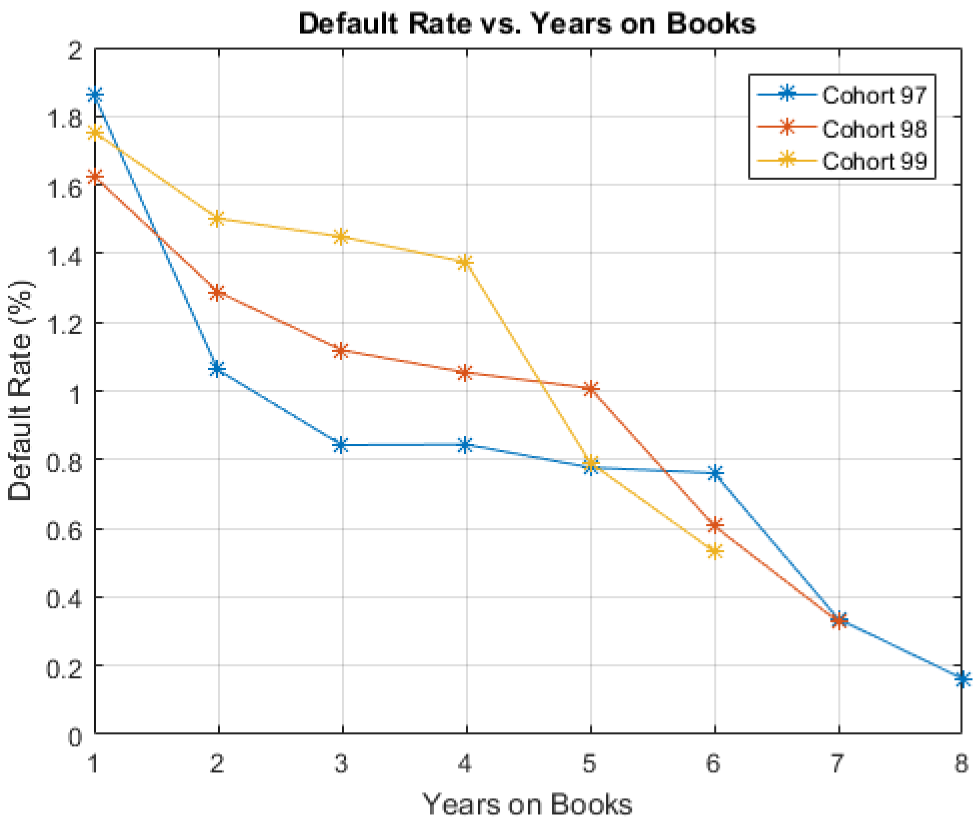
Years on Books Versus Calendar Years

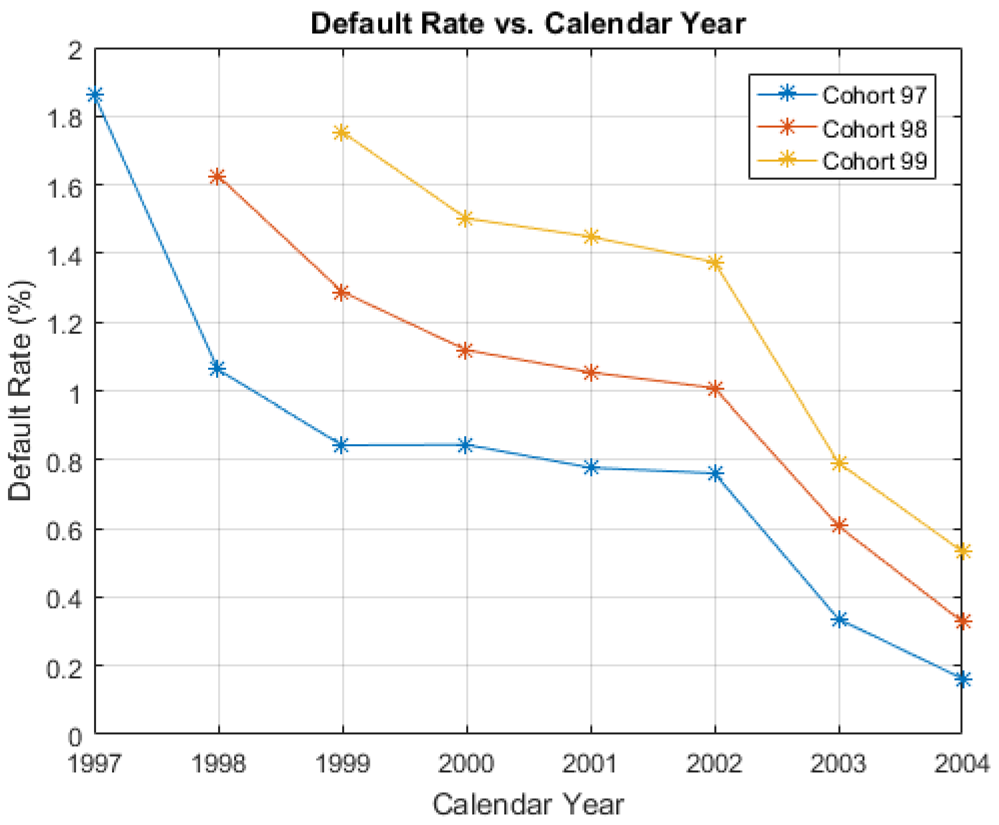
The data contains three cohorts, or vintages: loans started in 1997, 1998, and 1999. No loan in the panel data started after 1999.

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

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

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

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

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

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

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

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


Default Rate vs. Years on Books
Cohort 97
Cohort 98
Cohort 99
Default Rate (%)
Years on Books
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
1
2
3
4
5
6
7
8



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:

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

1M is an indicator with a value `1` for `Medium Risk` loans and `0` otherwise, and similarly for *1L* 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: *aH* for `High Risk`, *aH+aM* for `Medium Risk`, and *aH+aL* for `Low Risk`.

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

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

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

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

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

Estimated Coefficients:
                              Estimate       SE         tStat        pValue
                              ________    ________    _______    ___________

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


388018 observations, 388014 error degrees of freedom
Dispersion: 1
Chi^2-statistic vs. constant model: 1.83e+03, p-value = 0
```

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

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

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

To account for panel data effects, a more advanced model using mixed effects can be fitted using the `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');
```

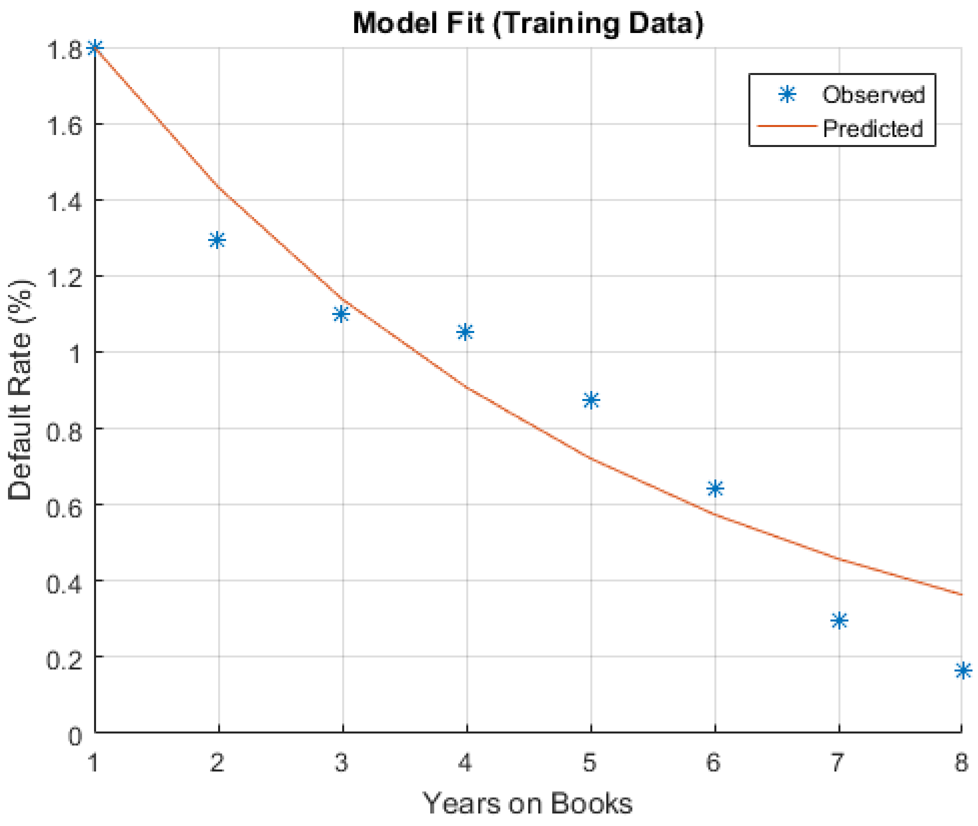
The `(1|ID)` term in the formula adds a *random effect* to the model. This effect is a predictor whose values are not given in the data, but calibrated together with the model coefficients. A random value is calibrated for each ID. This additional calibration requirement substantially increases the computational 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,:));
```

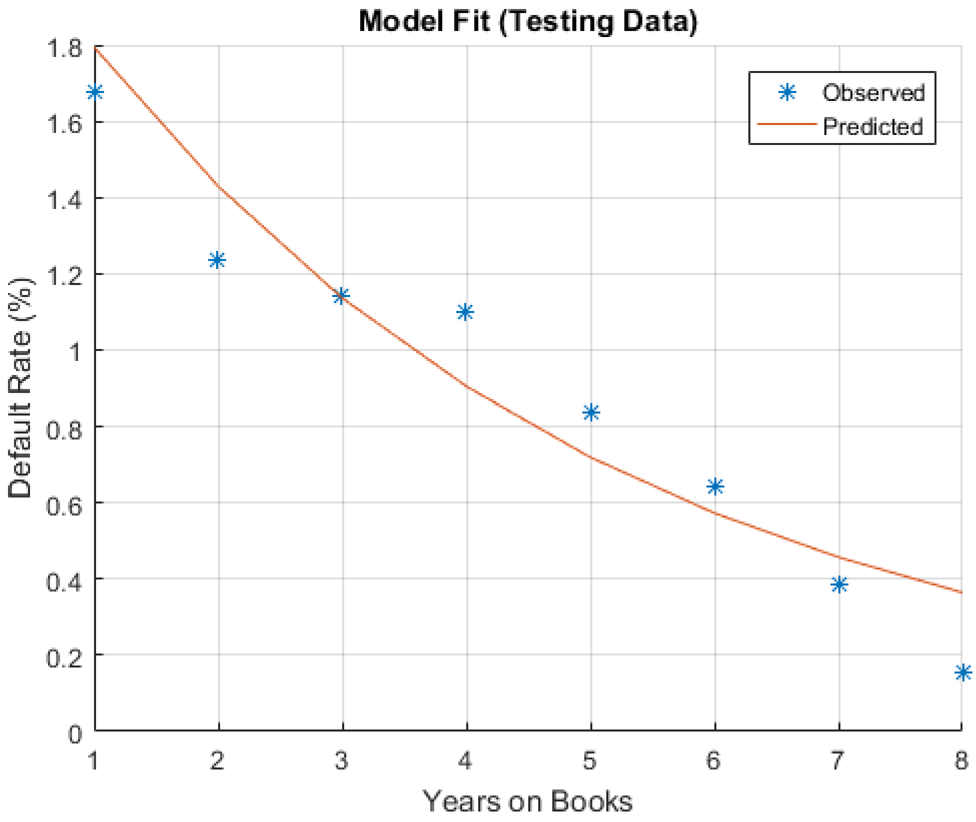
Visualize the in-sample fit.

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

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



Visualize the out-of-sample fit.

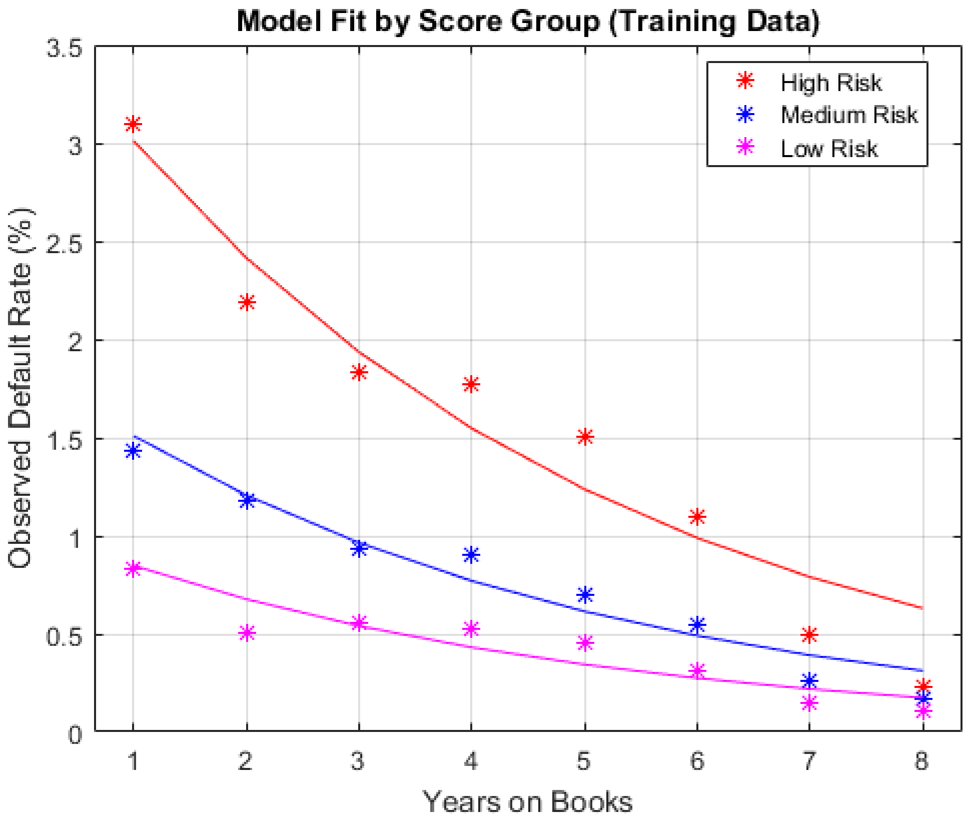
```
PredPDTestYOB = varfun(@mean,data(TestDataInd,:),...
    'InputVariables',{'Default','PDNoMacro'},'GroupingVariables','YOB');

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



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

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

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

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



Model of Default Rates Including Macroeconomic Variables

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

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

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

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

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

```
    ID     ScoreGroup      YOB     Default      Year     PDNoMacro       GDP      Market
    __    ___________     ___     _______      ____     _________      _____     ______

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

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

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

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

Estimated Coefficients:
```

```
                                   Estimate        SE         tStat       pValue   
                                  __________    _________    ________    __________

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


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

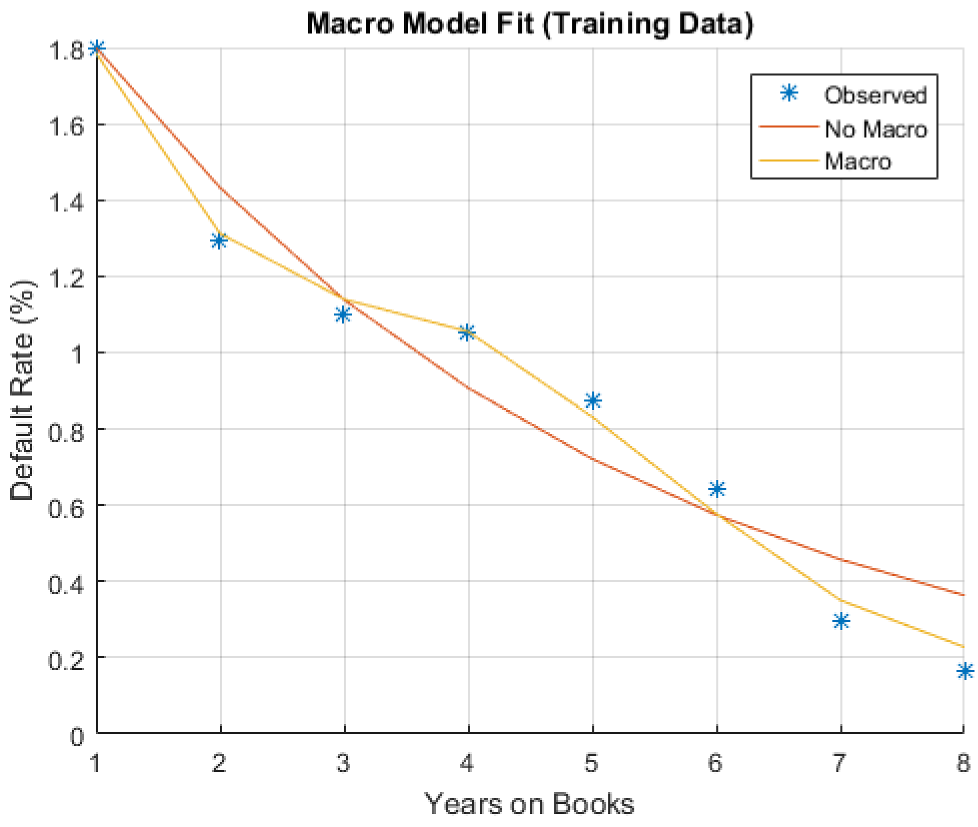
Both macroeconomic variables show a negative coefficient, consistent with the intuition that higher economic growth reduces default rates.

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

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

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

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

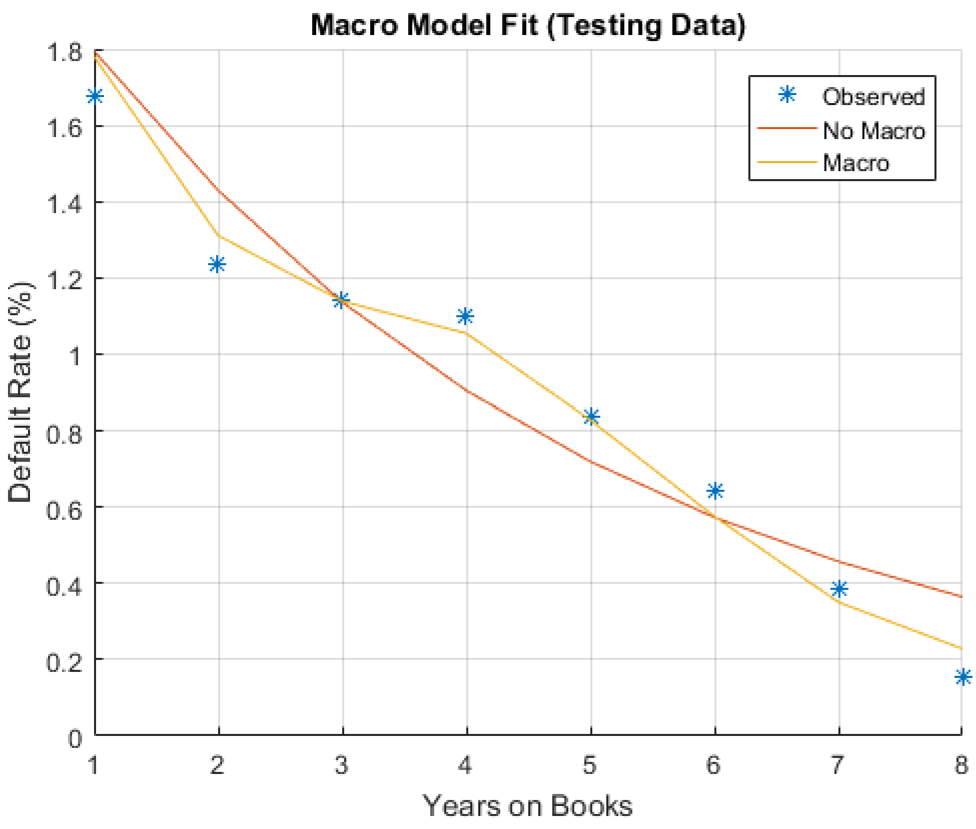
```
PredPDTrainYOBMacro = varfun(@mean,data(TrainDataInd,:),...
    'InputVariables',{'Default','PDMacro'},'GroupingVariables','YOB');

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

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

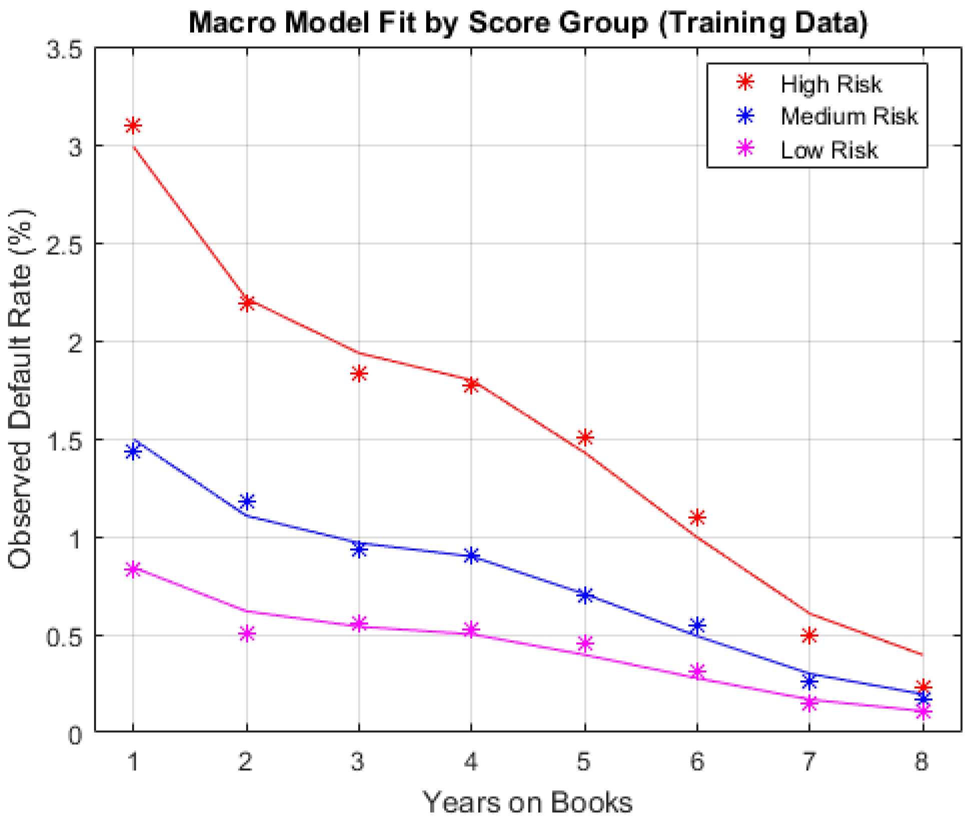
```
title('Macro Model Fit (Testing Data)')
grid on
```



Visualize the in-sample fit for all score groups.

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

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

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



Stress Testing of Probability of Default

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

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

```
disp(dataMacroStress)
```

```
                 GDP     Market
                _____    ______

    Baseline     2.27    15.02
    Adverse      1.31     4.56
    Severe      -0.22    -5.64
```

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

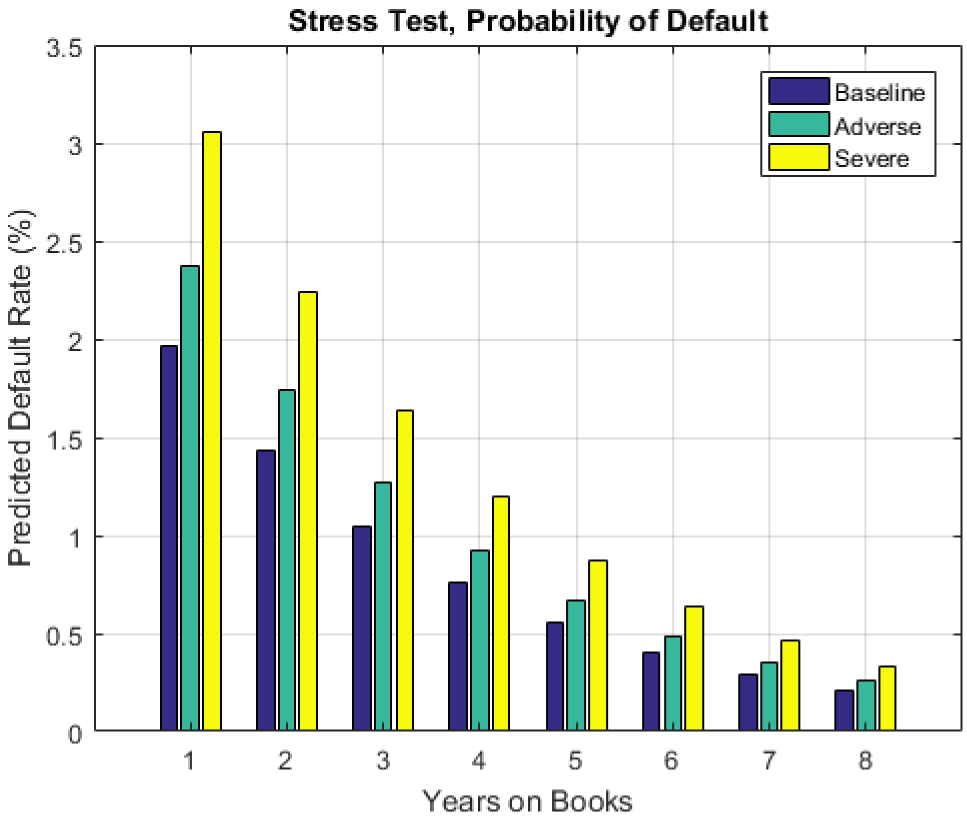
```
dataBaseline = table;
[ScoreGroup,YOB]=meshgrid(1:NumScoreGroups,1:NumYOB);
dataBaseline.ScoreGroup = categorical(ScoreGroup(:),1:NumScoreGroups,...
   categories(data.ScoreGroup),'Ordinal',true);
dataBaseline.YOB = YOB(:);
dataBaseline.ID = ones(height(dataBaseline),1);
dataBaseline.GDP = zeros(height(dataBaseline),1);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

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

```
% Predict baseline the probabilities of default
dataBaseline.GDP(:) = dataMacroStress.GDP('Baseline');
dataBaseline.Market(:) = dataMacroStress.Market('Baseline');
dataBaseline.PD = predict(ModelMacro,dataBaseline);

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

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

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

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

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



References

1 Generalized Linear Models documentation: http://www.mathworks.com/help/stats/generalized-linear-regression.html.

2 Generalized Linear Mixed Effects Models documentation: http://www.mathworks.com/help/stats/generalized-linear-mixed-effects-models.html.

3 Federal Reserve, Comprehensive Capital Analysis and Review (CCAR): http://www.federalreserve.gov/bankinforeg/ccar.htm.

4 Bank of England, Stress Testing: http://www.bankofengland.co.uk/financialstability/pages/fpc/stresstest.aspx.

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

See Also

`fitglm` | `fitglme`

Related Examples

- “Credit Rating by Bagging Decision Trees”

More About

- “About Credit Scorecards”
- “Credit Scorecard Modeling Workflow”
- creditscorecard

Corporate Credit Risk Simulations for Portfolios

Credit Simulation Using Copulas

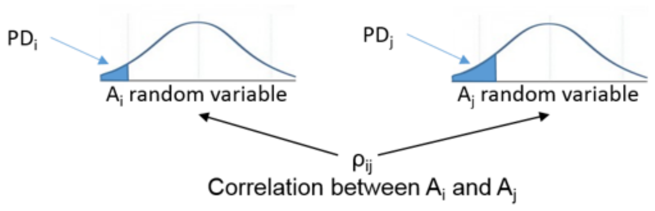
Predicting the credit losses for a counterparty depends on three main elements:

- 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 − *Recovery*

If these quantities are known at future time t, then the expected loss is `PD × EAD × 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 ith 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 jth 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 `w1`,...,`w`M. If Z and ε_i are standard normal variables, then Ai is also a standard normal.

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

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

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

When the latent variables Ai 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

The `creditCopula` object is used to simulate and analyze multifactor credit default simulations. These simulations assume that you calculated the main inputs to this model on the 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 `creditCopula` 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 `creditCopula` object to calculate several risk measures at the portfolio level and the risk contributions from individual obligors. The outputs of the `creditCopula` model and the associated functions are:

- The full simulated distribution of portfolio losses across scenarios and the losses on each counterparty across scenarios. See `creditCopula` properties and`simulate`.
- Risk measures (VaR, CVaR, EL, Std) with confidence intervals. See `portfolioRisk`.
- Risk contributions per counterparty (for EL & CVaR). See `riskContribution`.
- Risk measures and associated confidence bands. See `confidenceBands`.

See Also

`confidenceBands` | `creditCopula` | `portfolioRisk` | `riskContribution` | `simulate`

Related Examples

- "`creditCopula` Simulation Workflow" on page 4-5
- "Modeling Correlated Defaults with Copulas" on page 4-11

More About

- creditCopula

creditCopula Simulation Workflow

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

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

Step 1. Create a creditCopula object with a two-factor model.

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

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

```
  creditCopula with properties:

            Portfolio: [100×5 table]
    FactorCorrelation: [2×2 double]
             VaRLevel: 0.9500
      PortfolioLosses: []
    CounterpartyLosses: []

    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 creditCopula object to 99% (the default is 95%).

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

Step 3. Run a simulation.

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

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

```
cc = 

  creditCopula with properties:

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

Step 4. Generate a report for the portfolio risk.

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

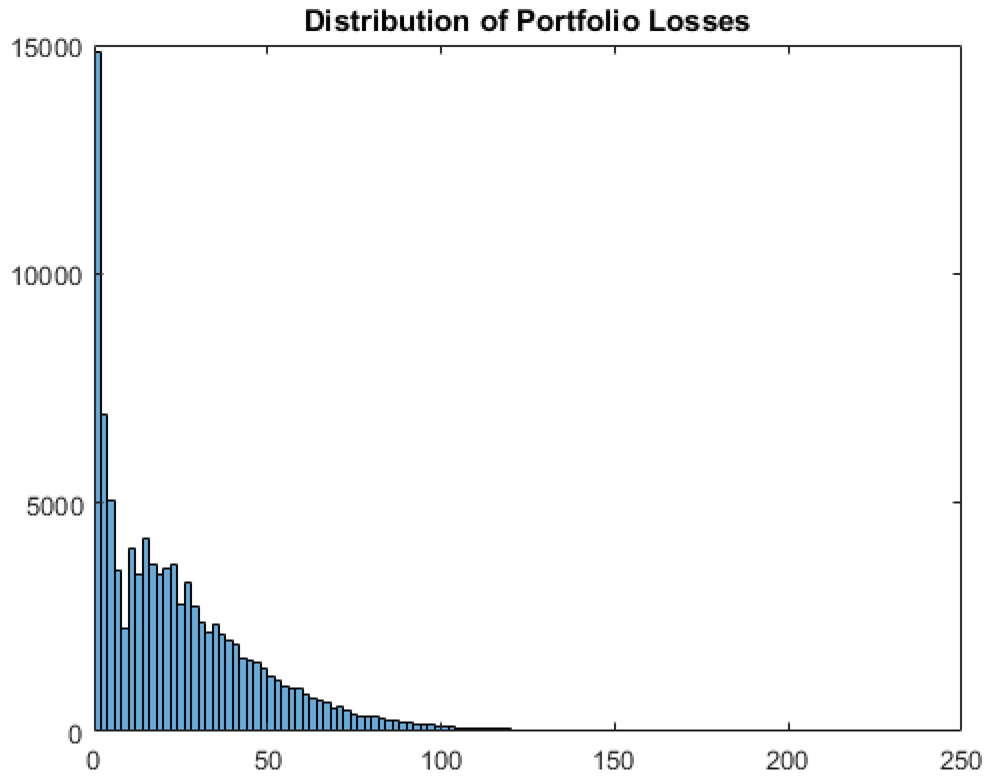
```
[portRisk,RiskConfidenceInterval] = portfolioRisk(cc)
```

```
portRisk = 

      EL        Std       VaR      CVaR 
    ______    ______    ______    _____

    24.768    23.667    102.21    120.9


RiskConfidenceInterval = 

          EL                 Std                 VaR                 CVaR      
    ________________    ________________    ________________    ________________

    24.621    24.914    23.564    23.771    101.12    103.32    119.78    122.01
```

Step 5. Visualize the distribution.

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

```
histogram(cc.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(cc);

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

```
    ID        EL           CVaR   
    __    __________    _________

     1      0.038907     0.090834
     2      0.068209      0.24527
     3        1.2473       2.6579
     4     0.0025223    0.0065685
     5       0.11905      0.28625
     6       0.12397        0.521
     7       0.83246       1.8713
     8    0.00093656            0
     9       0.91591       4.0861
    10       0.24168       1.8591
```

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 `creditCopula` object (`cct`).

```
cct = simulate(cc,1e5,'Copula','t','DegreesOfFreedom',10)
```

```
cct = 

  creditCopula with properties:

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

Step 8. Compare confidence bands for different copulas.

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

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

```
ans = 

    NumScenarios    Lower      Std      Upper 
    ____________    ______    ______    ______

    10000             23.5    23.773    24.053
    20000            23.64    23.834    24.032
    30000           23.682    23.841    24.003
    40000           23.514     23.65    23.789
    50000           23.569    23.692    23.816
    60000            23.59    23.702    23.816
    70000           23.616     23.72    23.825
    80000           23.623    23.721    23.819
    90000           23.614    23.705    23.797
    1e+05            23.58    23.667    23.755


ans = 

    NumScenarios    Lower      Std      Upper 
    ____________    ______    ______    ______

    10000           32.413     32.79    33.176
    20000            31.84    32.102    32.368
    30000           31.663    31.876    32.091
    40000           31.623    31.807    31.993
    50000           31.908    32.073    32.241
    60000           31.919     32.07    32.223
    70000           31.934    32.074    32.216
    80000           32.033    32.165    32.298
    90000           32.081    32.205     32.33
    1e+05           32.138    32.256    32.375
```

See Also

`confidenceBands` | `creditCopula` | `portfolioRisk` | `riskContribution` | `simulate`

Related Examples

- "Credit Simulation Using Copulas" on page 4-2
- "Modeling Correlated Defaults with Copulas" on page 4-11

More About

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

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 `creditCopula` 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 `creditCopula` 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 `creditCopula` 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 `creditCopula` object with the portfolio information and the factor correlation.

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

% Change the VaR level to 99%.
cc.VaRLevel = 0.99;

disp(cc)
```

```
  creditCopula with properties:

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

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

```
ans = 

    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 `creditCopula` object populates the `PortfolioLosses` and `CounterpartyLosses` properties with the simulation results.

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

```
  creditCopula with properties:

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

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

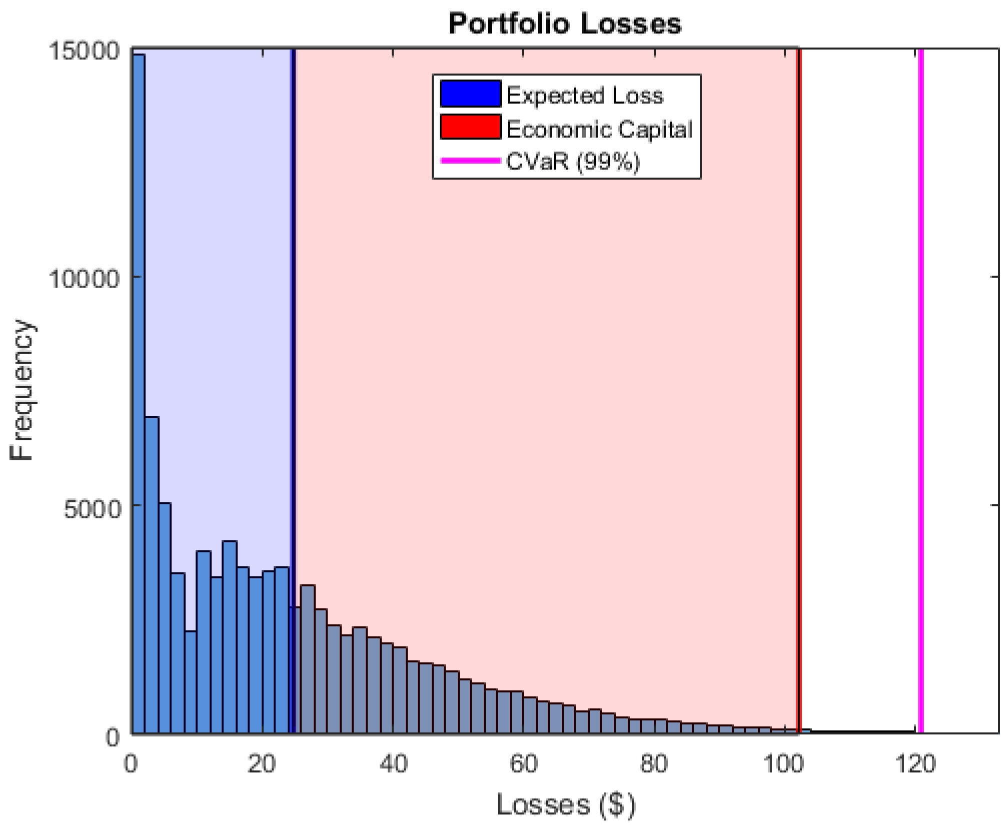
```
[pr,pr_ci] = portfolioRisk(cc);

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 
    ______    ______    ______    _____

    24.768    23.667    102.21    120.9



Confidence intervals for the risk measures:
           EL                  Std                  VaR                  CVaR      
    ________________    ________________    ________________    ________________

    24.621    24.914    23.564    23.771    101.12    103.32    119.78    122.01
```

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 = 

    ID        EL            CVaR    
    __    __________    __________

    1       0.038907      0.090834
    2       0.068209       0.24527
    3         1.2473        2.6579
    4      0.0025223     0.0065685
    5        0.11905       0.28625
```

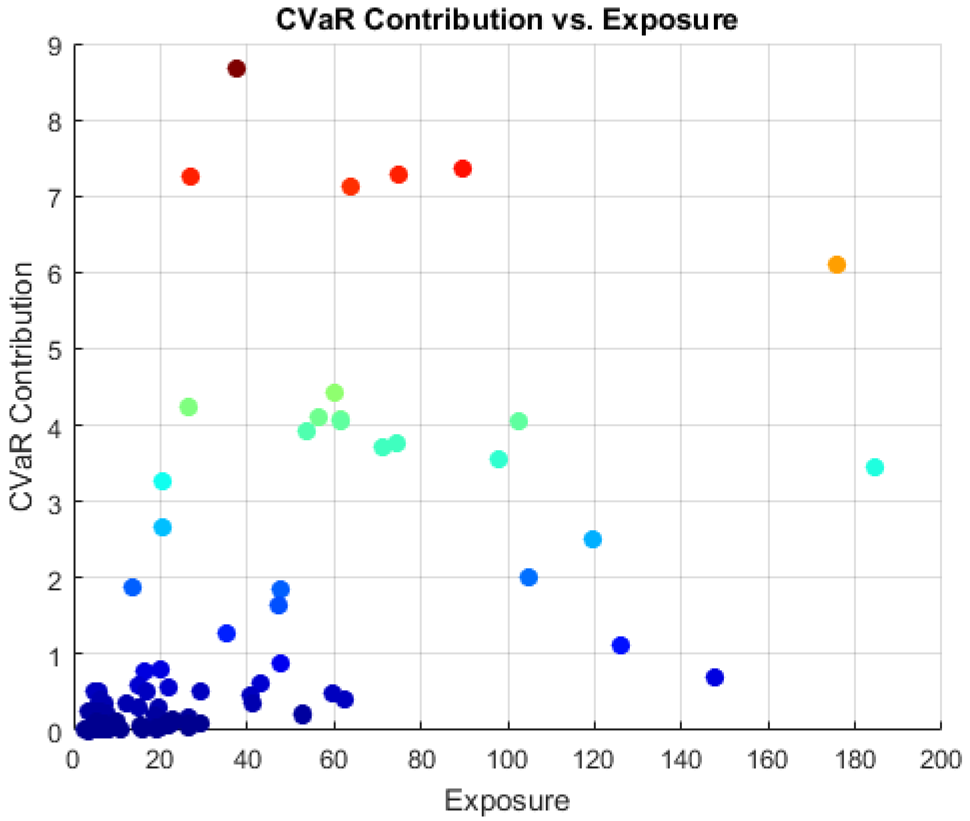
Find the riskiest counterparties by their CVaR contributions.

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

```
ans = 

    ID      EL        CVaR 
    __    _______    ______

    89     2.2752    8.6709
    96     1.3172    7.3477
    22     1.5716    7.2763
    16     1.6156    7.2421
    66    0.85578    7.1286
```

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

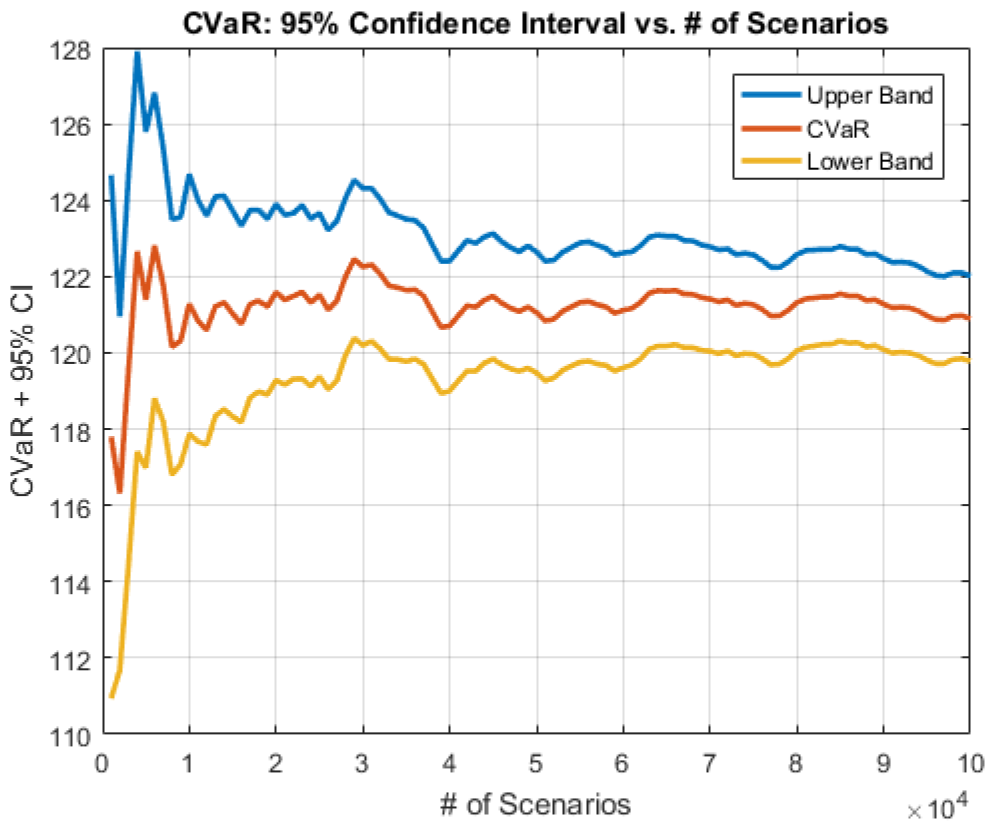
Use the `confidenceBands` function to investigate the convergence of the simulation. By default, the CVaR confidence bands are reported, but confidence bands for all risk measures are supported using the optional `RiskMeasure` argument.

```
cb = confidenceBands(cc);

% The confidence bands are stored in a table.
cb(1:5,:)
```

```
ans =
    NumScenarios    Lower      CVaR      Upper 
    ____________    ______    ______    ______

    1000            110.94     117.8    124.66
    2000            111.69    116.32    120.95
    3000            114.45    119.56    124.66
    4000            117.41    122.66     127.9
    5000            116.99    121.39    125.79
```

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

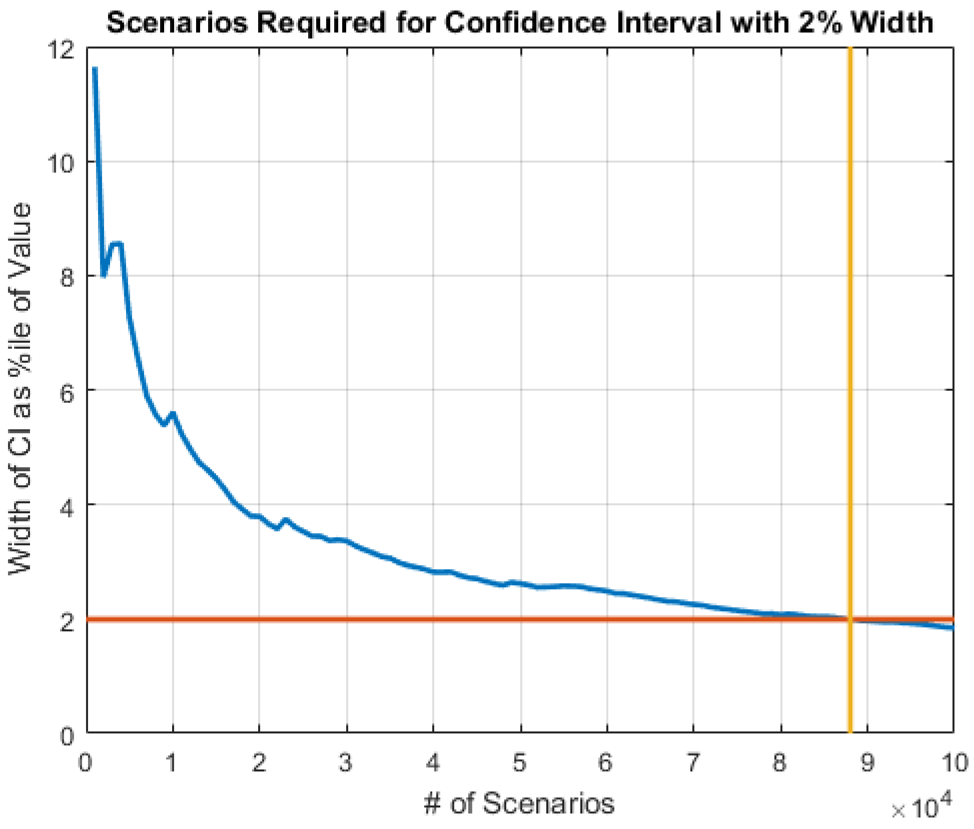
```
figure;
plot(...
    cb.NumScenarios,...
    cb{:,{'Upper' 'CVaR' 'Lower'}},...
    'LineWidth',2);

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.

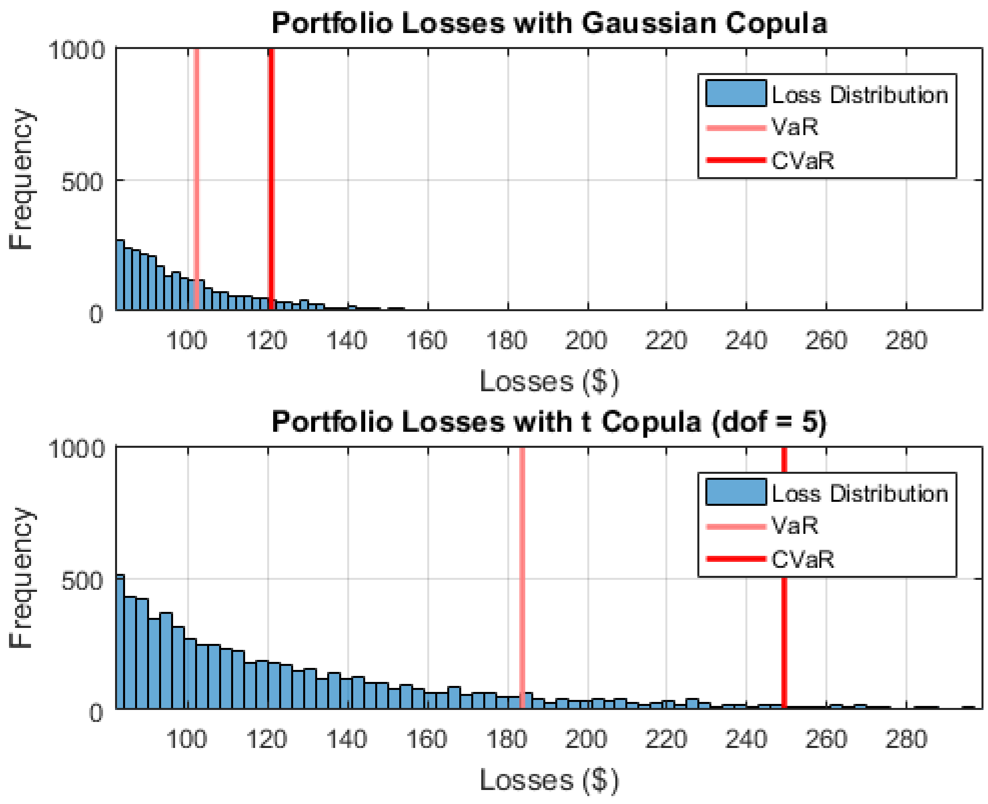
```
fprintf('Portfolio risk with Gaussian copula:\n');
disp(pr)

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

```
Portfolio risk with Gaussian copula:
      EL        Std        VaR       CVaR 

    ______    ______    ______    _____

    24.768    23.667    102.21    120.9



Portfolio risk with t copula (dof = 5):
      EL        Std        VaR        CVaR 

    ______    ______    ______    ______

    24.688    38.673    183.74    249.31
```

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` | `creditCopula` | `portfolioRisk` | `riskContribution` | `simulate`

Related Examples

- "Credit Simulation Using Copulas" on page 4-2
- "`creditCopula` Simulation Workflow" on page 4-5

More About

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

Functions — Alphabetical List

Binning Explorer

Bin data and export into a `creditscorecard` object

Description

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

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

Open the Binning Explorer App

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

Examples

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

More About

- “Overview of Binning Explorer” on page 3-2

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

See Also

Functions
`creditscorecard`

Introduced in R2016b

creditCopula

Simulate and analyze multifactor credit default model

Description

The `creditCopula` class simulates portfolio losses due to counterparty defaults using a multifactor model. The inputs to the model describe the credit-sensitive portfolio of exposures:

- `EAD` — Exposure at default
- `PD` — Probability of default
- `LGD` — Loss given default (1 − *Recovery*)
- `Weights` — Factor and idiosyncratic model weights

After the `creditCopula` object is created using the `creditCopula` function, use the `simulate` function to simulate credit defaults using the multifactor model. The results are stored in the form of a distribution of losses at the portfolio and counterparty level. Several risk measures at the portfolio level are calculated, as well as the risk contributions from individual obligors. The model calculates:

- The full simulated distribution of portfolio losses across scenarios
- The losses on each counterparty across scenarios
- Several risk measures (VaR, CVaR, EL, Std) with confidence intervals
- Risk contributions per counterparty (for EL and CVaR)

Create Object

To create a `creditCopula` object, use the `creditCopula` function.

Properties

`Portfolio` — Details of credit portfolio
table

Details of credit portfolio, specified as a MATLAB table that contains all the portfolio data that was passed as input into the `creditCopula` constructor.

The `Portfolio` table has a column for each of the constructor inputs (`EAD`, `PD`, `LGD`, `Weights`, and `ID`). Each row of the table represents one counterparty.

For example:

```
ID     EAD         PD           LGD        Weights
__    ______    _________     _______     _________
1     122.43     0.064853     0.68024     0.3  0.7
2     70.386     0.073957     0.59256     0.3  0.7
3     79.281     0.066235     0.52383     0.3  0.7
4     113.42      0.01466     0.43977     0.3  0.7
5     100.46    0.0042036     0.41838     0.3  0.7
```

Data Types: `table`

`FactorCorrelation` — Correlation matrix for credit factors
matrix

Correlation matrix for credit factors, specified as a `NumFactors`-by-`NumFactors` matrix. Specify the correlation matrix using the optional name-value pair argument `FactorCorrelation` with the `creditCopula` constructor.

Data Types: `double`

`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'` with the `creditCopula` constructor.

Data Types: `double`

`PortfolioLosses` — Total portfolio losses
vector

Total portfolio losses, specified as a `NumScenarios`-by-`1` vector. The `PortfolioLosses` property is empty after construction with the `creditCopula` constructor. After the `simulate` function is invoked, the `PortfolioLosses` property is populated with the vector of portfolio losses.

Data Types: `double`

`CounterpartyLosses` — Individual counterparty losses across all scenarios
sparse matrix

Individual counterparty losses across all scenarios, specified as a `NumCounterparties`-by-`NumScenarios` sparse matrix of counterparty losses. The `CounterpartyLosses` property is empty after construction with the `creditCopula` constructor. After the `simulate` function is invoked, it is populated with the losses for each counterparty over each scenario. This matrix can become large, and for this reason it is stored as a sparse matrix.

Data Types: `double`

Object Functions

creditCopula	Create creditCopula object
simulate	Simulate credit defaults using a creditCopula object
portfolioRisk	Generate portfolio-level risk measurements
riskContribution	Generate risk contributions for each counterparty in portfolio
confidenceBands	Confidence interval bands

Examples

Create a `creditCopula` Object and Simulate Credit Portfolio Losses

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cc = 

  creditCopula with properties:

             Portfolio: [100×5 table]
```

```
     FactorCorrelation: [2×2 double]
              VaRLevel: 0.9500
       PortfolioLosses: []
    CounterpartyLosses: []
```

Set the `VaRLevel` to 99%.

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

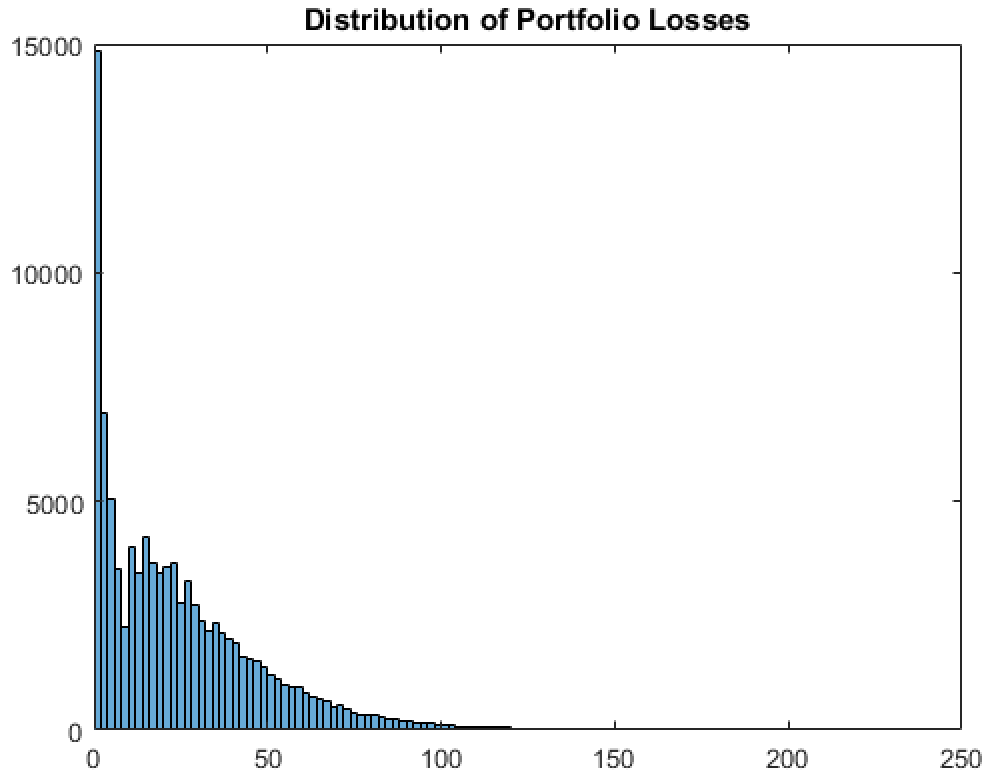
Simulate 100,000 scenarios and view the portfolio risk measures.

```
 cc = simulate(cc,1e5)
 portRisk = portfolioRisk(cc)
```

```
cc = 

  creditCopula with properties:

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


portRisk = 

      EL        Std       VaR       CVaR 
    ______    ______    ______    _____

    24.768    23.667    102.21    120.9
```

View a histogram of the portfolio losses.

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



For further analysis, use the `simulate`, `portfolioRisk`, and `riskContribution` functions with the `creditCopula` object.

- "Credit Simulation Using Copulas" on page 4-2
- "`creditCopula` Simulation Workflow" on page 4-5
- "Modeling Correlated Defaults with Copulas" on page 4-11

References

Crouhy, M., Galai, D., and Mark, R. "A Comparative Analysis of Current Credit Risk Models." *Journal of Banking and Finance.* Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance.* Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. *“CreditMetrics – Technical Document.”* J. P. Morgan, New York, 1997.

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

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA.* Wiley Finance, 2007.

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

See Also

`confidenceBands` | `creditCopula` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

More About

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

Introduced in R2016b

creditCopula

Create `creditCopula` object

Syntax

```
cc = creditCopula(EAD,PD,LGD,Weights)
cc = creditCopula( ___ ,Name,Value)
```

Description

`cc = creditCopula(EAD,PD,LGD,Weights)` creates a `creditCopula` object. For more information on using a `creditCopula` object, see creditCopula.

`cc = creditCopula( ___ ,Name,Value)` adds optional name-value pair arguments. For more information on using a `creditCopula` object, see creditCopula.

Examples

Create a `creditCopula` Object and Simulate Credit Portfolio Losses

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cc = 

  creditCopula with properties:

            Portfolio: [100×5 table]
    FactorCorrelation: [2×2 double]
```

```
             VaRLevel: 0.9500
       PortfolioLosses: []
    CounterpartyLosses: []
```

Set the `VaRLevel` to 99%.

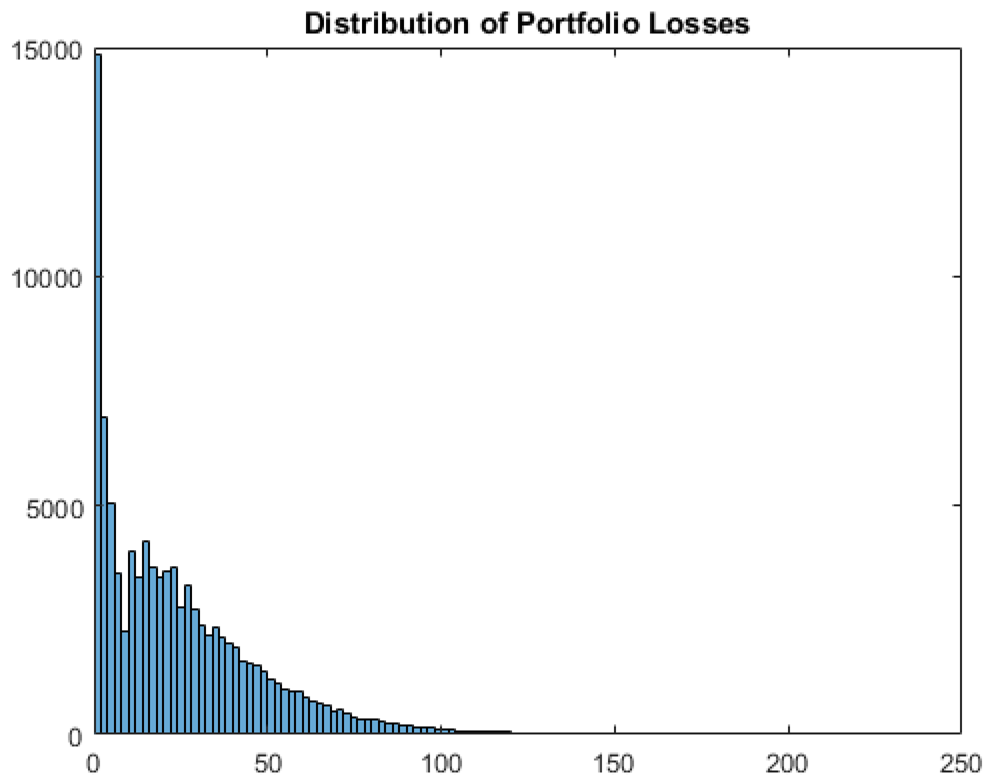
```
cc.VaRLevel = 0.99;
```

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

```
 cc = simulate(cc,1e5)
 portRisk = portfolioRisk(cc)


cc = 

  creditCopula with properties:

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


portRisk = 

      EL        Std        VaR       CVaR 
    ______    ______    ______    _____

    24.768    23.667    102.21    120.9
```

View a histogram of the portfolio losses.

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



For further analysis, use the `simulate`, `portfolioRisk`, and `riskContribution` functions with the `creditCopula` object.

- “Credit Simulation Using Copulas” on page 4-2
- “`creditCopula` Simulation Workflow” on page 4-5
- “Modeling Correlated Defaults with Copulas” on page 4-11

Input Arguments

EAD — Exposure at default
numeric vector

Exposure at default, specified as a `NumCounterparties`-by-`1` vector of credit exposures.

Note: The `creditCopula` model simulates defaults and losses over some fixed time period (for example, one year). The counterparty exposures (`EAD`) and default probabilities (`PD`) must both be specific to a particular time.

Data Types: `double`

`PD` — Probability of default
numeric vector with elements between `0` and `1`

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

Note: The `creditCopula` model simulates defaults and losses over some fixed time period (for example, one year). The counterparty exposures (`EAD`) and default probabilities (`PD`) must both be specific to a particular time.

Data Types: `double`

`LGD` — Loss given default
numeric vector with elements between `0` and `1`

Loss given default, specified as a `NumCounterparties`-by-`1` numeric vector with elements between `0` and `1`, representing the fraction of exposure that is lost when a counterparty defaults. `LGD` is defined as (1 − *Recovery*). For example, a `LGD` of 0.6 implies a 40% recovery rate in the event of a default.

Data Types: `double`

`Weights` — Factor and idiosyncratic weights
array of factor and idiosyncratic weights

Factor and idiosyncratic weights, specified as a `NumCounterparties`-by-(`NumFactors` + `1`) array. Each row contains the factor weights for a particular counterparty. Each column contains the weights for an underlying risk factor. The last column in `Weights` contains the idiosyncratic risk weight for each counterparty. The idiosyncratic weight represents the company-specific credit risk. The total of the weights for each counterparty (that is, each row) must sum to `1`.

For example, if a counterparty's creditworthiness was composed of 60% US, 20% European, and 20% idiosyncratic, then the `Weights` vector would be `[0.6 0.2 0.2]`.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `cc = creditCopula(EAD,PD,LGD,Weights,'VaRLevel',0.99)`

`'ID'` — User-defined IDs for counterparties

`ID` is numeric vector (default) | vector

User-defined IDs for counterparties, specified as a `NumCounterparties`-by-`1` vector of `ID`s for each counterparty. `ID` is used to identify exposures in the `Portfolio` table and the risk contribution table. `ID` must be a numeric, a string array, or a cell array of character vectors. If unspecified, `ID` defaults to a numeric vector.

Data Types: `double` | `string` | `cell`

`'VaRLevel'` — Value at risk level

`0.95` (default) | decimal

Value at risk level (used for reporting `VaR` and `CVaR`) specified as a decimal.

Data Types: `double`

`'FactorCorrelation'` — Factor correlation matrix

identity matrix (default) | correlation matrix

Factor correlation matrix, specified as a `NumFactors`-by-`NumFactors` matrix that defines the correlation between the risk factors. If not specified, the factor correlation matrix defaults to an identity matrix, meaning that factors are not correlated.

Data Types: `double`

Output Arguments

cc — creditCopula object

object

creditCopula object contains the provided portfolio data, that is stored in the Portfolio property.

The Portfolio property is a table that has a column for each of the constructor inputs (EAD, PD, LGD, Weights, and ID). Each row of the Portfolio table represents one counterparty.

The creditCopula object has the following properties:

- Portfolio:

  A table with the following variables:

  - ID — An ID to identify each counterparty
  - EAD — Exposure at default
  - PD — Probability of default
  - LGD — Loss given default
  - Weights — Factor and idiosyncratic weights for counterparties
- FactorCorrelation:

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

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

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

  Counterparty losses, a NumCounterparties-by-NumScenarios sparse matrix of individual counterparty losses across the scenarios. This property is empty until the simulate function is used.

For more information on `creditCopula` objects, see creditCopula.

More About

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

References

Crouhy, M., Galai, D., and Mark, R. "A Comparative Analysis of Current Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. *"CreditMetrics – Technical Document."* J. P. Morgan, New York, 1997.

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

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

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

See Also

`confidenceBands` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

Introduced in R2016b

confidenceBands

Confidence interval bands

Syntax

```
cbTable = confidenceBands(cc)
cbTable = confidenceBands(cc,Name,Value)
```

Description

`cbTable = confidenceBands(cc)` returns a table of the requested risk measure and its associated confidence bands. `confidenceBands` is used to investigate how the values of a risk measure and its associated confidence interval converge as the number of scenarios increases. The `simulate` function must be run before `confidenceBands` is used. For more information on using a `creditCopula` object, see creditCopula.

`cbTable = confidenceBands(cc,Name,Value)` adds optional name-value pair arguments. The `simulate` function must be run before `confidenceBands` is used. For more information on using a `creditCopula` object, see creditCopula.

Examples

Generate a Table of the Associated Confidence Bands for a Requested Risk Measure for a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
cc = 

  creditCopula with properties:

              Portfolio: [100×5 table]
      FactorCorrelation: [2×2 double]
               VaRLevel: 0.9500
        PortfolioLosses: []
     CounterpartyLosses: []
```

Set the `VaRLevel` to 99%.

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

Use the `simulate` function before running `confidenceBands`. Use `confidenceBands` with the `creditCopula` object to generate the `cbTable`.

```
cc = simulate(cc,1e5);
cbTable = confidenceBands(cc,'RiskMeasure','Std','ConfidenceIntervalLevel',0.9);
cbTable(1:10,:)
```

```
ans = 

    NumScenarios    Lower      Std     Upper 
    ____________    ______    ______    ______

     1000           23.008    23.852    24.766
     2000           22.922    23.517    24.146
     3000           23.159     23.65    24.164
     4000           23.762    24.198    24.652
     5000           23.646    24.035    24.437
     6000           23.788    24.145    24.513
     7000           23.709    24.038    24.377
     8000           23.496    23.801    24.115
     9000            23.43    23.717    24.012
    10000             23.5    23.773    24.053
```

- “Credit Simulation Using Copulas” on page 4-2
- “`creditCopula` Simulation Workflow” on page 4-5
- “Modeling Correlated Defaults with Copulas” on page 4-11

Input Arguments

`cc` — `creditCopula` object
object

`creditCopula` object, specified using the updated `creditCopula` object obtained from running the `simulate` function, which must be run before `confidenceBands` is used.

For more information on `creditCopula` objects, see creditCopula.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `cbTable = confidenceBands(cc,'RiskMeasure','Std','ConfidenceIntervalLevel',0.92,'NumPoin`

`'RiskMeasure'` — Risk measure to investigate
`CVaR` (default) | character vector with values `'EL'`, `'Std'`, `'VaR'`, or `'CVaR'`

Risk measure to investigate, specified as character vector with possible values:

- `'EL'` — Expected loss, the mean of portfolio losses
- `'Std'` — Standard deviation of the losses
- `'VaR'` — Value at risk at the threshold specified by the `VaRLevel` property of the `creditCopula` object
- `'CVaR'` — Conditional VaR at the threshold specified by the `VaRLevel` property of the `creditCopula` object

Data Types: `char`

`'ConfidenceIntervalLevel'` — Confidence interval level
`0.95` (default) | decimal

Confidence interval level, specified as a decimal. For example, if you specify `0.95`, a 95% confidence interval is reported in the output table (`cbTable`).

Data Types: `double`

`'NumPoints'` — Number of scenario samples to report
`100` (default) | nonnegative integer

Number of scenario samples to report, specified as a nonnegative integer. The default is `100`, meaning confidence bands are reported at 100 evenly spaced points of increasing sample size ranging from 0 to the total number of simulated scenarios.

Note: `NumPoints` must be a numeric scalar greater than `1`, and is typically much smaller than total number of scenarios simulated. `confidenceBands` should be used to obtain a qualitative idea of how fast a risk measure and its confidence interval are converging. Specifying a large value for `NumPoints` is not recommended and could cause performance issues with `confidenceBands`.

Data Types: `double`

Output Arguments

`cbTable` — Requested risk measure and associated confidence bands
table

Requested risk measure and associated confidence bands at each of the `NumPoints` scenario sample sizes, returned as a table containing the following columns:

- `NumScenarios` — Number of scenarios at the sample point
- `Lower` — Lower confidence band
- *`RiskMeasure`* — Requested risk measure where the column takes its name from whatever risk measure is requested with the optional input `RiskMeasure`
- `Upper` — Upper confidence band

More About

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

References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance.* Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance.* Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. *“CreditMetrics – Technical Document.”* J. P. Morgan, New York, 1997.

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

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA.* Wiley Finance, 2007.

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

See Also

`creditCopula` | `portfolioRisk` | `riskContribution` | `simulate` | `table`

Introduced in R2016b

portfolioRisk

Generate portfolio-level risk measurements

Syntax

```
[riskMeasures,confidenceIntervals] = portfolioRisk(cc)
[riskMeasures,confidenceIntervals] = portfolioRisk(cc,Name,Value)
```

Description

`[riskMeasures,confidenceIntervals] = portfolioRisk(cc)` returns tables of risk measurements for the portfolio losses. The `simulate` function must be run before `portfolioRisk` is used. For more information on using a `creditCopula` object, see creditCopula.

`[riskMeasures,confidenceIntervals] = portfolioRisk(cc,Name,Value)` adds an optional name-value pair argument for `ConfidenceIntervalLevel`. The `simulate` function must be run before `portfolioRisk` is used.

Examples

Generate Tables for Risk Measure and Confidence Intervals for a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cc = 

  creditCopula with properties:

            Portfolio: [100×5 table]
```

```
     FactorCorrelation: [2×2 double]
              VaRLevel: 0.9500
       PortfolioLosses: []
    CounterpartyLosses: []
```

Set the `VaRLevel` to 99%.

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

Use the `simulate` function before running `portfolioRisk`. Then use `portfolioRisk` with the `creditCopula` object to generate the `riskMeasure` and `ConfidenceIntervals` tables.

```
cc = simulate(cc,1e5);
[riskMeasure,confidenceIntervals] = portfolioRisk(cc,'ConfidenceIntervalLevel',0.9)
```

```
riskMeasure =

      EL        Std       VaR       CVaR 
    ______    ______    ______    _____

    24.768    23.667    102.21    120.9


confidenceIntervals =

           EL                   Std                   VaR                   CVaR       
    ________________    _________________    ________________    ________________

    24.645    24.891    23.58     23.755    101.29    103.14    119.96    121.83
```

- “Credit Simulation Using Copulas” on page 4-2
- “`creditCopula` Simulation Workflow” on page 4-5
- “Modeling Correlated Defaults with Copulas” on page 4-11

Input Arguments

`cc` — `creditCopula` object

object

`creditCopula` object, specified as an updated `creditCopula` object obtained by running the `simulate` function, which must be run before `portfolioRisk` is used.

For more information on `creditCopula` objects, see creditCopula.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `[riskMeasure,confidenceIntervals] = portfolioRisk(cc,'ConfidenceIntervalLevel',0.92)`

`'ConfidenceIntervalLevel'` — Confidence interval level
`0.95` (default) | decimal

Confidence interval level, specified as a decimal. For example, if you specify `0.95`, a 95% confidence interval is reported in the output table (`riskMeasures`).

Data Types: `double`

Output Arguments

`riskMeasures` — Risk measures
table

Risk measures, returned as a table containing the following columns:

- `EL` — Expected loss, the mean of portfolio losses
- `Std` — Standard deviation of the losses
- `VaR` — Value at risk at the threshold specified by the `VaRLevel` property of the `creditCopula` object
- `CVaR` — Conditional VaR at the threshold specified by the `VaRLevel` property of the `creditCopula` object

`confidenceIntervals` — Confidence intervals
table

Confidence intervals, returned as a table of confidence intervals corresponding to the portfolio risk measures reported in the `riskMeasures` table. Confidence intervals are reported at the level specified by the `ConfidenceIntervalLevel` parameter.

More About

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

References

Crouhy, M., Galai, D., and Mark, R. "A Comparative Analysis of Current Credit Risk Models." *Journal of Banking and Finance.* Vol. 24, 2000, pp. 59–117.

Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance.* Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. *"CreditMetrics – Technical Document."* J. P. Morgan, New York, 1997.

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

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA.* Wiley Finance, 2007.

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

See Also

`confidenceBands` | `creditCopula` | `riskContribution` | `simulate` | `table`

Introduced in R2016b

riskContribution

Generate risk contributions for each counterparty in portfolio

Syntax

```
riskContributions = riskContribution(cc)
```

Description

`riskContributions = riskContribution(cc)` returns a table of risk contributions for each counterparty in the portfolio. The `riskContributions` table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

For more information on using a `creditCopula` object, see creditCopula.

Examples

Determine the Risk Contribution for Each Counterparty for a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cc = 

  creditCopula with properties:

              Portfolio: [100×5 table]
      FactorCorrelation: [2×2 double]
               VaRLevel: 0.9500
```

```
          PortfolioLosses: []
      CounterpartyLosses: []
```

Set the `VaRLevel` to 99%.

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

Use the `simulate` function before running `riskContribution`. Then use `riskContribution` with the `creditCopula` object to generate the `riskContributions`.

```
cc = simulate(cc,1e5);
riskContributions = riskContribution(cc);
riskContributions(1:10,:)
```

```
ans =

    ID        EL           CVaR
    __    __________    _________

     1      0.038907     0.090834
     2      0.068209      0.24527
     3        1.2473       2.6579
     4     0.0025223    0.0065685
     5       0.11905      0.28625
     6       0.12397        0.521
     7       0.83246       1.8713
     8    0.00093656            0
     9       0.91591       4.0861
    10       0.24168       1.8591
```

- "Credit Simulation Using Copulas" on page 4-2
- "`creditCopula` Simulation Workflow" on page 4-5
- "Modeling Correlated Defaults with Copulas" on page 4-11

Input Arguments

`cc` — `creditCopula` object
object

`creditCopula` object, specified as an updated `creditCopula` object obtained by running the `simulate` function, which must be run before `riskContribution` is used.

For more information on `creditCopula` objects, see creditCopula.

Output Arguments

`riskContributions` — Risk contributions

table

Risk contributions, returned as a table containing the following risk contributions for each counterparty:

- `EL` — The expected loss for the particular counterparty over the scenarios
- `CVaR` — The conditional value at risk for the particular counterparty over the scenarios

The `riskContributions` table allocates the full portfolio risk measures to each counterparty, such that the counterparty risk contributions sum to the portfolio risks reported by `portfolioRisk`.

More About

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

References

Crouhy, M., Galai, D., and Mark, R. "A Comparative Analysis of Current Credit Risk Models." *Journal of Banking and Finance.* Vol. 24, 2000, pp. 59–117.

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

Gordy, M. "A Comparative Anatomy of Credit Risk Models." *Journal of Banking and Finance.* Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. *"CreditMetrics – Technical Document."* J. P. Morgan, New York, 1997.

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

Kalkbrener, M., Lotter, H., and Overbeck, L. “Sensible and Efficient Capital Allocation for Credit Portfolios.” *Risk.* 17, 2004, pp. S19–S24.

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA.* Wiley Finance, 2007.

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

See Also

`confidenceBands` | `creditCopula` | `portfolioRisk` | `simulate` | `table`

Introduced in R2016b

simulate

Simulate credit defaults using a `creditCopula` object

Syntax

```
cc = simulate(cc,NumScenarios)
cc = simulate(___,Name,Value)
```

Description

`cc = simulate(cc,NumScenarios)` performs the full simulation of credit scenarios and computes defaults and losses for the portfolio defined in the `creditCopula` object.

For more information on using a `creditCopula` object, see creditCopula.

`cc = simulate(___,Name,Value)` adds optional name-value pair arguments for (`Copula`, `DegreesOfFreedom`, and `BlockSize`).

Examples

Run a Simulation Using a `creditCopula` Object

Load saved portfolio data.

```
load CreditPortfolioData.mat;
```

Create a `creditCopula` object with a two-factor model.

```
cc = creditCopula(EAD,PD,LGD,Weights2F,'FactorCorrelation',FactorCorr2F)
```

```
cc = 

  creditCopula with properties:

            Portfolio: [100×5 table]
```

```
     FactorCorrelation: [2×2 double]
              VaRLevel: 0.9500
       PortfolioLosses: []
    CounterpartyLosses: []
```

Set the `VaRLevel` to 99%.

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

Use the `simulate` function with the `creditCopula` object. After using `simulate`, you can then use the `portfolioRisk`, `riskContribution`, and `confidenceBands` functions with the updated `creditCopula` object.

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

```
cc = 

  creditCopula with properties:

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

For instance, you can use `riskContribution` with the `creditCopula` object to generate the `riskContributions`.

```
riskContributions = riskContribution(cc);
riskContributions(1:10,:)
```

```
ans = 

    ID        EL          CVaR  
    __    __________    ________

     1      0.038756     0.62827
     2      0.068083     0.37989
     3          1.24      3.7233
     4     0.0027588    0.034156
     5       0.11941      0.5625
     6       0.12628      0.74279
     7       0.83595       2.6062
     8    0.00075817     0.018731
     9       0.93719       5.4739
    10       0.25739       4.1895
```

- "Credit Simulation Using Copulas" on page 4-2
- "`creditCopula` Simulation Workflow" on page 4-5
- "Modeling Correlated Defaults with Copulas" on page 4-11

Input Arguments

`cc` — `creditCopula` object
object

`creditCopula` object, obtained by running the `creditCopula` constructor.

For more information on a `creditCopula` object, see creditCopula.

`NumScenarios` — Number of scenarios to simulate
nonnegative integer

Number of scenarios to simulate, specified as a nonnegative integer. Scenarios are processed in blocks to conserve machine resources.

Data Types: `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `obj= simulate(obj,NumScenarios,'Copula','t','DegreesOfFreedom',5)`

`'Copula'` — Type of copula
`'Gaussian'` (default) | character vector with values `'Gaussian'` or `'t'`

Type of copula, specified as a character vector with a value of:

- `'Gaussian'` — A Gaussian copula
- `'t'` — A *t* copula with degrees of freedom specified using `DegreesOfFreedom`.

Data Types: `char`

`'DegreesOfFreedom'` — Degrees of freedom for t copula
`5` degrees of freedom (default) | nonnegative numeric value

Degrees of freedom for a *t* copula, specified as a nonnegative numeric value. If `Copula` is set to `'Gaussian'`, the `DegreesOfFreedom` parameter is ignored.

Data Types: `double`

`'BlockSize'` — Number of scenarios to process in each iteration
`1e5/Number of counterparties` (default) | nonnegative numeric value

Number of scenarios to process in each iteration, specified as a nonnegative numeric value.

Data Types: `double`

Output Arguments

`cc` — Updated `creditCopula` object
object

Updated `creditCopula` object. The object is populated with the simulated `PortfolioLosses` and `CounterpartyLosses`.

For more information on a `creditCopula` object, see creditCopula.

Note: In the `simulate` function, the `Weights` are transformed to ensure that the latent variables have a mean of `0` and a variance of `1`.

More About

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

References

Crouhy, M., Galai, D., and Mark, R. “A Comparative Analysis of Current Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 59–117.

Gordy, M. “A Comparative Anatomy of Credit Risk Models.” *Journal of Banking and Finance*. Vol. 24, 2000, pp. 119–149.

Gupton, G., Finger, C., and Bhatia, M. *“CreditMetrics – Technical Document.”* J. P. Morgan, New York, 1997.

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

Löffler, G., and Posch, P. *Credit Risk Modeling Using Excel and VBA*. Wiley Finance, 2007.

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

See Also

`confidenceBands` | `creditCopula` | `portfolioRisk` | `riskContribution` | `table`

Introduced in R2016b

varbacktest

Suite of value-at-risk (VaR) back tests

Description

The general workflow is:

1. Load or generate the data for the VaR backtesting analysis.
2. Create a `varbacktest` object using the `varbacktest` constructor.
3. Use the `summary` function to generate a summary report for the given data on the number of observations and the number of failures.
4. Use the `runtests` function to run all tests at once.
5. For additional test details, run the following individual tests:

   - `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 2-8.

Create Object

To create a `varbacktest` (`vbt`) object, use the `varbacktest` function.

Properties

`PortfolioData` — Portfolio data for VaR backtesting analysis

numeric array

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

Data Types: `double`

`VaRData` — VaR data for VaR backtesting analysis
numeric array

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

Data Types: `double`

`PortfolioID` — Portfolio identifier
string

Portfolio identifier, specified as a string.

Data Types: `string`

`VaRID` — VaR identifier
string array

VaR identifier, specified as a `1`-by-`NumVaRs` string array containing the VaR IDs for the corresponding columns in `VaRData`.

Data Types: `string`

`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` Function	Modify Property Using Dot Notation
`PortfolioData`	Yes	No
`VaRData`	Yes	No
`PortfolioID`	Yes	Yes
`VaRID`	Yes	Yes

`varbacktest` Property	Set or Modify Property from Command Line Using `varbacktest` Function	Modify Property Using Dot Notation
`VaRLevel`	Yes	Yes

Object Functions

varbacktest	Create varbacktest object using portfolio outcomes data and corresponding value-at-risk (VaR) data
tl	Traffic light test for value-at-risk (VaR) backtesting
bin	Binomial test for value-at-risk (VaR) backtesting
pof	Proportion of failures test for value-at-risk (VaR) backtesting
tuff	Time until first failure test for value-at-risk (VaR) backtesting
cc	Conditional coverage mixed test for value-at-risk (VaR) backtesting
cci	Conditional coverage independence test for value-at-risk (VaR) backtesting
tbf	Time between failures mixed test for value-at-risk (VaR) backtesting
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%

The `varbacktest` constructor takes in portfolio outcomes data and corresponding value-at-risk (VaR) data and returns a `varbacktest` object.

Create a `varbacktest` object.

```
load VaRBacktestData
```

```
 vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

`vbt`, the `varbacktest` object, contains a copy of the given portfolio data (`PortfolioData` property), the given VaR data (`VaRData` property) and all combinations of portfolio ID, VaR ID, and VaR level to be tested (`PortfolioID`, `VaRID`, and `VaRLevel` properties).

Run the tests using the `vbt` object.

```
 runtests(vbt)
```

```
ans =

    PortfolioID    VaRID    VaRLevel     TL      Bin       POF       TUFF       CC   
    ___________    _____    ________    _____    ______    ______    ______    ______

    "Portfolio"    "VaR"    0.95        green    accept    accept    accept    accept
```

Change the `PortfolfioID` and `VaRID` properties using dot notation. For more information on creating a `varbacktest` object, see `varbacktest`.

```
vbt.PortfolioID = 'S&P'
vbt.VaRID = 'Normal at 95%'
```

```
vbt = 

  varbacktest with properties:

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

```
      PortfolioID: "S&P"
            VaRID: "VaR"
         VaRLevel: 0.9500


vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "S&P"
            VaRID: "Normal at 95%"
         VaRLevel: 0.9500
```

Run all tests using the updated `varbacktest` object.

```
runtests(vbt)
```

```
ans =

    PortfolioID         VaRID          VaRLevel     TL       Bin       POF       TUFF
    ___________    _______________    ________    _____    ______    ______    ______

    "S&P"          "Normal at 95%"    0.95        green    accept    accept    accept
```

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

References

Basel Committee on Banking Supervision, *Supervisory Framework for the Use of 'Backtesting' in Conjunction with the Internal Models Approach to Market Risk Capital Requirements*. January, 1996, http://www.bis.org/publ/bcbs22.htm.

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

Cogneau, Ph. *“Backtesting Value-at-Risk: How Good is the Model?"* Intelligent Risk, PRMIA, July, 2015.

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

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

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

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

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

See Also

`varbacktest`

More About

- "Overview of VaR Backtesting" on page 2-2

Introduced in R2016b

varbacktest

Create `varbacktest` object using portfolio outcomes data and corresponding value-at-risk (VaR) data

Syntax

```
vbt = varbacktest(PortfolioData,VaRData)
vbt = varbacktest(___,Name,Value)
```

Description

`vbt = varbacktest(PortfolioData,VaRData)` creates a `varbacktest` object using portfolio outcomes data and corresponding value-at-risk (VaR) data. For more information on using a `vbt` object, see varbacktest.

`vbt = varbacktest(___,Name,Value)` adds optional name-value pair arguments. For more information on using a `vbt` object, see varbacktest.

Examples

Run VaR Backtests for a Single VaR at 95%

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
```

```
        VaRLevel: 0.9500
```

`vbt`, the `varbacktest` object, contains a copy of the given portfolio data (`PortfolioData` property), the given VaR data (`VaRData` property) and all combinations of portfolio ID, VaR ID, and VaR level to be tested (`PortfolioID`, `VaRID`, and `VaRLevel` properties).

Run the tests using the `varbacktest` object.

```
 runtests(vbt)
```

```
ans =

    PortfolioID    VaRID    VaRLevel     TL      Bin       POF       TUFF       CC
    ___________    _____    ________    _____    ______    ______    ______    ______

    "Portfolio"    "VaR"    0.95        green    accept    accept    accept    accept
```

Change the `PortfolfioID` and `VaRID` properties using dot notation. For more information on a `varbacktest` object, see `varbacktest`.

```
vbt.PortfolioID = 'S&P'
vbt.VaRID = 'Normal at 95%'
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "S&P"
            VaRID: "VaR"
         VaRLevel: 0.9500


vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "S&P"
            VaRID: "Normal at 95%"
         VaRLevel: 0.9500
```

Run all tests using the updated `varbacktest` object.

```
runtests(vbt)
```

```
ans =

    PortfolioID          VaRID          VaRLevel     TL         Bin        POF        TUFF
    ___________    _______________    ________    _____    ______    ______    ______

    "S&P"          "Normal at 95%"    0.95        green    accept    accept    accept
```

Run VaR Backtests for Multiple VaRs at Different Confidence Levels

Create a `varbacktest` object that has multiple VaR identifiers with different confidence levels.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99]);
```

Run the summary report for the `varbacktest` object.

```
summary(vbt)
```

```
ans =

    PortfolioID         VaRID          VaRLevel    ObservedLevel    Observations    Failu
    ___________    ______________    ________    _____________    ____________    _____

    "Equity"       "Normal95"        0.95        0.94535          1043            57
    "Equity"       "Normal99"        0.99         0.9837          1043            17
    "Equity"       "Historical95"    0.95        0.94343          1043            59
    "Equity"       "Historical99"    0.99        0.98849          1043            12
    "Equity"       "EWMA95"          0.95        0.94343          1043            59
    "Equity"       "EWMA99"           0.99        0.97891           1043           22
```

Run all tests using the `varbacktest` object.

```
 runtests(vbt)


ans = 

    PortfolioID        VaRID          VaRLevel      TL       Bin       POF       TUFF
    ___________    _______________    ________    ______    ______    ______    ______

    "Equity"       "Normal95"         0.95        green     accept    accept    accept
    "Equity"       "Normal99"         0.99        yellow    reject    accept    accept
    "Equity"       "Historical95"     0.95        green     accept    accept    accept
    "Equity"       "Historical99"     0.99        green     accept    accept    accept
    "Equity"       "EWMA95"           0.95        green     accept    accept    accept
    "Equity"       "EWMA99"           0.99        yellow    reject    reject    accept
```

Run the traffic light test (`tl`) using the `varbacktest` object.

```
 tl(vbt)


ans = 

    PortfolioID        VaRID          VaRLevel      TL      Probability      TypeI
    ___________    _______________    ________    ______    ___________    _________

    "Equity"       "Normal95"         0.95        green     0.77913          0.26396
    "Equity"       "Normal99"         0.99        yellow    0.97991          0.03686
    "Equity"       "Historical95"     0.95        green     0.85155          0.18232
    "Equity"       "Historical99"     0.99        green     0.74996          0.35269
    "Equity"       "EWMA95"           0.95        green     0.85155          0.18232
    "Equity"       "EWMA99"           0.99        yellow    0.99952        0.0011122
```

Run VaR Backtests for Multiple Portfolios and Concatenate Results

Use the `varbacktest` constructor with table inputs and name-value pair arguments to create two `varbacktest` objects and run the concatenated summary report. The `varbacktest` constructor uses the variable names in the table inputs as `PortfolioID` and `VaRID`.

```
load VaRBacktestData
vbtE = varbacktest(DataTable(:,2),DataTable(:,3:4),'VaRLevel',[0.95 0.99]);
vbtD = varbacktest(DataTable(:,5),DataTable(:,6:7),'VaRLevel',[0.95 0.99]);
[summary(vbtE); summary(vbtD)]
```

```
ans =

     PortfolioID            VaRID           VaRLevel    ObservedLevel    Observations
    _____________    ____________________    ________    _____________    ____________

    "Equity"         "VaREquity95"           0.95        0.94343          1043
    "Equity"         "VaREquity99"           0.99        0.97891          1043
    "Derivatives"    "VaRDerivatives95"      0.95        0.95014          1043
    "Derivatives"    "VaRDerivatives99"      0.99        0.97028          1043
```

Run all the tests and concatenate the results.

```
[runtests(vbtE); runtests(vbtD)]
```

```
ans =

     PortfolioID            VaRID           VaRLevel      TL        Bin        POF        T
    _____________    ____________________    ________    ______    ______    ______    _

    "Equity"         "VaREquity95"           0.95        green     accept    accept    ac
    "Equity"         "VaREquity99"           0.99        yellow    reject    reject    ac
    "Derivatives"    "VaRDerivatives95"      0.95        green     accept    accept    ac
    "Derivatives"    "VaRDerivatives99"      0.99        red       reject    reject    ac
```

Run the `pof` test and concatenate the results.

```
 [pof(vbtE); pof(vbtD)]
```

```
ans =

     PortfolioID            VaRID           VaRLevel     POF       LRatioPOF     PValueP
    _____________    ____________________    ________    ______    __________    _______

    "Equity"         "VaREquity95"           0.95        accept       0.91023       0.34
    "Equity"         "VaREquity99"           0.99        reject        9.8298     0.0017
    "Derivatives"    "VaRDerivatives95"    0.95        accept    0.00045457       0.982
    "Derivatives"    "VaRDerivatives99"    0.99        reject        26.809    2.2457e-
```

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

Input Arguments

`PortfolioData` — Portfolio outcomes data

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

Portfolio outcomes data, specified as a `NumRows`-by-`1` numeric array or `NumRows`-by-`1` table with a numeric column containing portfolio outcomes data. `PortfolioData` can be expressed as returns, or alternatively as profits and losses. There are no validations in the tool regarding the units of portfolio or VaR data. It is the user's responsibility to provide the portfolio and VaR data in the same units.

Missing values (`NaN`s in the `PortfolioData`) are discarded before applying the tests. Therefore, a different number of observations is reported (`NumObs`) for series with different number of missing values. The reported number of observations (`NumObs`) equals the original number of rows in the data (`NumRows`) minus the number of periods with missing values.

Data Types: `double` | `table`

`VaRData` — Value-at-risk (VaR) data

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

Value-at-risk (VaR) data, specified using a `NumRows`-by-`NumVaRs` numeric array, or `NumRows`-by-`NumVaRs` table with numeric columns.

`VaRData` data must be in the same units `PortfolioData`. There are no validations in the tool regarding the units of portfolio or VaR data. These may be expressed as returns, or alternatively as profits and losses. It is the user's responsibility to provide the portfolio and VaR data in the same units.

If `VaRData` has more than one column (`NumVaRs> 1`), the `PortfolioData` is tested against each column in `VaRData`. By default, a `0.95` VaR confidence level is used for all columns in `VaRData`. (Use `VaRLevel` to specify different VaR confidence levels.)

The convention is that VaR is a positive amount. Therefore, a failure is recorded when the loss (the negative of the portfolio data) exceeds the VaR, that is, when

```
-PortfolioData > VaRData
```

For example, a VaR of one million (positive) is violated whenever there is an outcome worse than a one-million loss (the negative of the portfolio outcome, or loss, is larger than the VaR).

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

Missing values (`NaN`s in `VaRData`) are discarded before applying the tests. Therefore, a different number of observations is reported (`NumObs`) for series with different number of missing values. The reported number of observations (`NumObs`) equals the original number of rows in the data (`NumRows`) minus the number of periods with missing values.

Example:

Data Types: `double` | `table`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `vbt = varbacktest(PortfolioData,VaRData,'PortfolioID','Equity100','VaRID','TotalVaR'`

`'PortfolioID'` — User-defined ID for `PortfolioData` input
character vector | string

User-defined ID for `PortfolioData` input, specified as a character vector or string.

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

Data Types: `char` | `string`

'VaRID' — VaR identifier for VaRData columns
character vector | cell array of character vectors | string | string array

VaR identifier for VaRData columns, specified using a character vector or string. Multiple VaRIDs are specified using a 1-by-NumVaRs (or NumVaRs-by-1) cell array of character vectors or string vector with user-defined IDs for the VaRData columns.

If VaRData is a numeric array, the default value for VaRID is 'VaR'. If NumVaRs = 1 or NumVaRs > 1, the default value is 'VaR1', 'VaR2', and so on. If VaRData is a table, 'VaRID' is set by default to the corresponding variable names in the table

Data Types: char | cell | string

'VaRLevel' — VaR confidence level
0.95 (default) | numeric with values between 0 and 1 | numeric array with values between 0 and 1

VaR confidence level, specified as a numeric between 0 and 1 or a 1-by-NumVaRs numeric array with values between 0 and 1 for the corresponding columns in VaRData.

Data Types: double

Output Arguments

vbt — varbacktest object
object

varbacktest (vbt) object, contains a copy of the given portfolio data and VaR data (the PortfolioData and VaRData properties) and all combinations of portfolio ID, VaR ID, and VaR level to be tested (the PortfolioID, VaRID, and VarLevel properties). For more information on a vbt object, see varbacktest.

The vbt object has the following properties:

- PortfolioData — NumRows-by-1 numeric array containing a copy of the PortfolioData
- VaRData — NumRows-by-NumVaRs numeric array containing a copy of the VaRData
- PortfolioID — String containing the PortfolioID
- VaRID — 1-by-NumVaRs string vector containing the VaRIDs for the corresponding columns in VaRData

- `VaRLevel` — `1`-by-`NumVaRs` numeric array containing the `VaRLevel`s for the corresponding columns in `VaRData`.

More About

- varbacktest

References

Basel Committee on Banking Supervision, *Supervisory Framework for the Use of 'Backtesting' in Conjunction with the Internal Models Approach to Market Risk Capital Requirements*. January, 1996, http://www.bis.org/publ/bcbs22.htm.

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

Cogneau, P. *"Backtesting Value-at-Risk: How Good is the Model?"* Intelligent Risk, PRMIA, July, 2015.

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

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

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

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

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

See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `table` | `tbf` | `tbfi` | `tl` | `tuff`

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: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `bin` test results.

```
TestResults = bin(vbt)
```

```
TestResults = 

    PortfolioID     VaRID     VaRLevel      Bin       ZScoreBin     PValueBin     Observatio
    ___________     _____     ________     ______     _________     _________     __________

    "Portfolio"     "VaR"     0.95         accept     0.68905       0.24539       1043
```

Run Bin Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `bin` test results using the `TestLevel` optional argument.

```
TestResults = bin(vbt,'TestLevel',0.90)
```

```
TestResults = 

    PortfolioID         VaRID          VaRLevel      Bin       ZScoreBin     PValueBin     
    ___________     ______________     ________     ______     _________     __________    _
    "Equity"       "Normal95"        0.95        accept    0.68905          0.24539
    "Equity"       "Normal99"        0.99        reject     2.0446         0.020448
    "Equity"       "Historical95"    0.95        accept     0.9732          0.16523
    "Equity"       "Historical99"    0.99        accept    0.48858          0.31257
    "Equity"       "EWMA95"          0.95        accept     0.9732          0.16523
    "Equity"       "EWMA99"          0.99        reject     3.6006       0.00015875
```

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

Input Arguments

`vbt` — varbacktest object
object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = bin(vbt,'TestLevel',0.99)`

`'TestLevel'` — Test confidence level
`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as 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 – 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(N*p*(1 - p))`.

The test statistic for the `bin` test is the z-score, defined as:

```
ZScoreBin = (x - N*p)/sqrt(N*p*(1 - p)).
```

The z-score is approximately follows a standard normal distribution. This approximation is not reliable for small values of `N` or small values of `p`, but for typical uses in VaR backtesting analyses (`N = 250` or much larger, `p` in the range 1 – 10%) the approximation gives results in line with other tests.

The `p`-value of the `bin` test is the probability that a standard normal distribution exceeds the absolute value of the z-score

```
PValueBin = 1 - F(|ZScoreBin|),
```

where `F` is the standard normal cumulative distribution. When too few failures are observed, relative to the expected failures, `PValueBin` is (approximately) the probability of observing that many failures or fewer. For too many failures, this is (approximately) the probability of observing that many failures or more.

- "Binomial Test" on page 2-3
- varbacktest

References

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

See Also

`cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` | `varbacktest`

Introduced in R2016b

cc

Conditional coverage mixed test for value-at-risk (VaR) backtesting

Syntax

```
TestResults = cc(vbt)
TestResults = cc(vbt,Name,Value)
```

Description

`TestResults = cc(vbt)` generates the conditional coverage (CC) mixed test for value-at-risk (VaR) backtesting.

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

Examples

Generate CC Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `cc` test results.

```
TestResults = cc(vbt)
```

```
TestResults = 

    PortfolioID    VaRID    VaRLevel      CC       LRatioCC    PValueCC     POF      LRa
    ___________    _____    ________    ______    ________    ________    ______    ___

    "Portfolio"    "VaR"    0.95        accept    0.72013     0.69763     accept    0.4
```

Run the CC Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'},
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `cc` test results using the `TestLevel` optional input.

```
TestResults = cc(vbt,'TestLevel',0.90)
TestResults =

    PortfolioID        VaRID          VaRLevel      CC       LRatioCC     PValueCC       PO
    ___________    _______________    ________    ______    ________    _________    ___

     "Equity"      "Normal95"           0.95      accept    0.72013       0.69763    acc
     "Equity"      "Normal99"           0.99      accept     4.0757       0.13031    rej
     "Equity"      "Historical95"       0.95      accept     1.0487       0.59194    acc
     "Equity"      "Historical99"       0.99      accept     0.5073       0.77597    acc
     "Equity"      "EWMA95"             0.95      accept    0.95051       0.62173    acc
     "Equity"      "EWMA99"             0.99      reject     10.779     0.0045645    rej
```

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

Input Arguments

`vbt` — varbacktest object

object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = cc(vbt,'TestLevel',0.99)`

`'TestLevel'` — Test confidence level

`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as 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 two degrees of freedom. See the Algorithms section in `pof` and `cci` for the definition of their likelihood ratios.

The `p`-value of the `cc` test is the probability that a chi-square distribution with two degrees of freedom exceeds the likelihood ratio `LRatioCC`,

```
PValueCC = 1 - F(LRatioCC)
```

where `F` is the cumulative distribution of a chi-square variable with two degrees of freedom.

The result of the `cc` test is to accept if

```
  F(LRatioCC) < F(TestLevel)
```

and reject otherwise, where `F` is the cumulative distribution of a chi-square variable with two degrees of freedom.

- “Christoffersen’s Interval Forecast Tests” on page 2-5
- varbacktest

References

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`

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: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `cci` test results.

```
TestResults = cci(vbt)
```

```
TestResults = 

    PortfolioID    VaRID    VaRLevel     CCI      LRatioCCI    PValueCCI    Observatio
    ___________    _____    ________    ______    _________    _________    __________

    "Portfolio"    "VaR"    0.95        accept    0.25866      0.61104      1043
```

Run the CCI Test for VaR Backtests for Multiple VaR's at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `cci` test results using the `TestLevel` optional input.

```
TestResults = cci(vbt,'TestLevel',0.90)
```

```
TestResults = 

    PortfolioID         VaRID          VaRLevel     CCI      LRatioCCI    PValueCCI    Ob
    ___________    ______________    ________    ______    _________    _________    __
    "Equity"      "Normal95"        0.95        accept     0.25866     0.61104       1
    "Equity"      "Normal99"        0.99        accept     0.56393     0.45268       1
    "Equity"      "Historical95"    0.95        accept     0.13847     0.70981       1
    "Equity"      "Historical99"    0.99        accept     0.27962     0.59695       1
    "Equity"      "EWMA95"          0.95        accept    0.040277     0.84094       1
    "Equity"      "EWMA99"          0.99        accept     0.94909     0.32995       1
```

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

Input Arguments

`vbt` — varbacktest object
object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = cci(vbt,'TestLevel',0.99)`

`'TestLevel'` — Test confidence level
`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as 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:

- `p01` = Probability of having a failure on period *t*, given that there was no failure on period$t-1$

  ```
  = N01 / (N00 + N01)
  ```
- `p11` = Probability of having a failure on period *t*, given that there was a failure on period$t-1$

  ```
  = N11 / (N10 + N11)
  ```

Define also the unconditional probability estimate of observing a failure:

`pUC` = Probability of having a failure on period *t*

```
   = (N01 + N11) / (N00 + N01 + N10 + N11)
```

The likelihood ratio of the CCI test is then given by

```
LRatioCCI = -2*((N00+N10)*log(1 - pUC) + (N01+N11)*log(pUC)...
            - N00*log(1-p01) - N01*log(p01)...
            - N10*log(1-p11) - N11*log(p11) )
```

which is asymptotically distributed as a chi-square distribution with one degree of freedom.

The `p`-value of the CCI test is the probability that a chi-square distribution with one degree of freedom exceeds the likelihood ratio `LRatioCCI`,

```
PValueCC = 1 - F(LRatioCCI)
```

where `F` is the cumulative distribution of a chi-square variable with one degree of freedom.

The result of the test is to accept if

```
 F(LRatioCCI) < F(TestLevel)
```

and reject otherwise, where `F` is the cumulative distribution of a chi-square variable with one degree of freedom.

If one or more of the quantities `N00`, `N10`, `N01`, or `N11` are zero, the likelihood ratio is handled differently. The likelihood ratio as defined above is composed of three likelihood functions of the form

```
L = (1 - p)^n1 * p^n2
```

For example, in the numerator of the likelihood ratio, there is a likelihood function of the form `L` with `p` = `pUC`, `n1` = `N00` + `N10`, and `n2` = `N01` + `N11`. There are two such likelihood functions in the denominator of the likelihood ratio.

It can be shown that whenever `n1` = `0` or `n2` = `0`, the likelihood function `L` can be replaced by the constant value `1`. Therefore, whenever `N00`, `N10`, `N01`, or `N11` is zero, replace the corresponding likelihood functions by `1` in the likelihood ratio, and the likelihood ratio is well defined.

- "Christoffersen's Interval Forecast Tests" on page 2-5
- varbacktest

References

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`

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: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `pof` test results.

```
TestResults = pof(vbt,'TestLevel',0.99)
```

```
TestResults = 

    PortfolioID    VaRID    VaRLevel     POF      LRatioPOF    PValuePOF    Observatio
    ___________    _____    ________    ______    _________    _________    __________

    "Portfolio"    "VaR"    0.95        accept    0.46147      0.49694      1043
```

Run the POF Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `pof` test results using the `TestLevel` optional input.

```
TestResults = pof(vbt,'TestLevel',0.90)
```

```
TestResults = 

    PortfolioID         VaRID          VaRLevel     POF      LRatioPOF    PValuePOF    O
    ___________    ______________    ________    ______    _________    _________    _
    "Equity"      "Normal95"       0.95        accept    0.46147          0.49694    1(
    "Equity"      "Normal99"       0.99        reject     3.5118         0.060933    1(
    "Equity"      "Historical95"   0.95        accept    0.91023          0.34005    1(
    "Equity"      "Historical99"   0.99        accept    0.22768          0.63325    1(
    "Equity"      "EWMA95"         0.95        accept    0.91023          0.34005    1(
    "Equity"      "EWMA99"         0.99        reject     9.8298        0.0017171    1(
```

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

Input Arguments

`vbt` — varbacktest object

object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = pof(vbt,'TestLevel',0.99)`

`'TestLevel'` — Test confidence level

`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as 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*((N - x)*log(N*(1 - pVaR)/(N - x)) + x*log(N*pVaR/x))
```

where `N` is the number of observations, x is the number of failures, and `pVaR` = 1 − `VaRLevel`. This test statistic is asymptotically distributed as a chi-square distribution with one degree of freedom. By the properties of the logarithm,

```
LRatioPOF = -2*N*log(1-pVaR)   if x = 0,
```

and

```
LRatioPOF = -2*N*log(pVaR)     if x = N.
```

The `p`-value of the POF test is the probability that a chi-square distribution with one degree of freedom exceeds the likelihood ratio `LRatioPOF`

```
PValuePOF = 1 - F(LRatioPOF),
```

where `F` is the cumulative distribution of a chi-square variable with one degree of freedom.

The result of the test is to accept if

```
 F(LRatioPOF) < F(TestLevel)
```

and reject otherwise, where `F` is the cumulative distribution of a chi-square variable with one degree of freedom.

- "Kupiec's POF and TUFF Tests" on page 2-4
- varbacktest

References

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`

Introduced in R2016b

runtests

Run all tests in `varbacktest`

Syntax

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

Description

`TestResults = runtests(vbt)` runs all the tests in the `varbacktest` object. `runtests` reports only the final test result. For test details such as likelihood ratios, run individual tests.

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

Examples

Run All VaR Backtests

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the TestResults report for all VaR backtests.

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

```
TestResults = 

    PortfolioID    VaRID    VaRLevel     TL      Bin       POF       TUFF       CC 
    ___________    _____    ________    _____    ______    ______    ______    ______

    "Portfolio"    "VaR"    0.95        green    accept    accept    accept    accept
```

Run All VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the varbacktest constructor with name-value pair arguments to create a varbacktest object and run all tests.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99]);
    runtests(vbt)
```

```
ans = 

    PortfolioID        VaRID         VaRLevel      TL       Bin       POF       TUFF
    ___________    ______________    ________    ______    ______    ______    ______

    "Equity"       "Normal95"        0.95        green     accept    accept    accept
    "Equity"       "Normal99"        0.99        yellow    reject    accept    accept
    "Equity"       "Historical95"    0.95        green     accept    accept    accept
    "Equity"       "Historical99"    0.99        green     accept    accept    accept
    "Equity"       "EWMA95"          0.95        green     accept    accept    accept
    "Equity"       "EWMA99"          0.99        yellow    reject    reject    accept
```

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

Input Arguments

`vbt` — varbacktest object
object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

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

`'TestLevel'` — Test confidence level
`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as a numeric between `0` and `1`.

Data Types: `double`

Output Arguments

`TestResults` — Results
table

Results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- `'PortfolioID'` — Portfolio ID for the given data
- `'VaRID'` — VaR ID for each of the VaR data columns provided
- `'VaRLevel'` — VaR level for the corresponding VaR data column
- `'TL'` — Categorical (ordinal) array with categories `green`, `yellow`, and `red` that indicate the result of the traffic light (`tl`) test

- `'Bin'` — Categorical array with categories `accept` and `reject` that indicate the result of the `bin` test
- `'POF'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `pof` test.
- `'TUFF'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `tuff` test
- `'CC'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `cc` test
- `'CCI'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `cci` test
- `'TBF'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `tbf` test
- `'TBFI'` — Categorical array with the categories `accept` and `reject` that indicate the result of the `tbfi` test

Note: For the test results, the terms `accept` and `reject` are used for convenience, technically a test does not accept a model. Rather, a test fails to reject it.

More About

- varbacktest

See Also

`cc` | `cci` | `pof` | `summary` | `tbf` | `tbfi` | `tl` | `tuff` | `varbacktest`

Introduced in R2016b

summary

Report on varbacktest data

Syntax

```
S = summary(vbt)
```

Description

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

Examples

Generate a Summary Report

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the summary report.

```
S = summary(vbt)
```

```
S =

    PortfolioID     VaRID     VaRLevel     ObservedLevel     Observations     Failures     Ex
    ___________     _____     ________     _____________     ____________     ________     _

    "Portfolio"     "VaR"     0.95         0.94535           1043             57           52
```

Run a Summary Report for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object and generate a summary report.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99]);
S = summary(vbt)
```

```
S =

    PortfolioID         VaRID          VaRLevel     ObservedLevel     Observations     Failu
    ___________     ______________     ________     _____________     ____________     _____

    "Equity"        "Normal95"         0.95         0.94535           1043             57
    "Equity"        "Normal99"         0.99          0.9837           1043             17
    "Equity"        "Historical95"     0.95         0.94343           1043             59
    "Equity"        "Historical99"     0.99         0.98849           1043             12
    "Equity"        "EWMA95"           0.95         0.94343           1043             59
    "Equity"        "EWMA99"           0.99         0.97891           1043             22
```

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

Input Arguments

`vbt` — varbacktest object

object

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

Output Arguments

s — Summary report
table

Summary report, returned as a table. The table rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- 'PortfolioID' — Portfolio ID for the given data
- 'VaRID' — VaR ID for each of the VaR data columns provided
- 'VaRLevel' — VaR level for the corresponding VaR data column
- 'ObservedLevel' — Observed confidence level, defined as number of periods without failures divided by number of observations
- 'Observations' — Number of observations, where missing values are removed from the data
- 'Failures' — Number of failures, where a failure occurs whenever the loss (negative of portfolio data) exceeds the VaR
- 'Expected' — Expected number of failures, defined as the number of observations multiplied by one minus the VaR level
- 'Ratio' — Ratio of the number of failures to expected number of failures
- 'FirstFailure' — Number of periods until first failure
- 'Missing' — Number of periods with missing values removed from the sample

More About

- varbacktest

See Also

cc | cci | pof | runtests | tbf | tbfi | tl | tuff | varbacktest

Introduced in R2016b

tbf

Time between failures mixed test for value-at-risk (VaR) backtesting

Syntax

```
TestResults = tbf(vbt)
TestResults = tbf(vbt,Name,Value)
```

Description

`TestResults = tbf(vbt)` generates the time between failures mixed test (TBF) for value-at-risk (VaR) backtesting.

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

Examples

Generate TBF Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `tbf` test results.

```
TestResults = tbf(vbt)
```

```
TestResults =

    PortfolioID     VaRID     VaRLevel      TBF       LRatioTBF     PValueTBF      POF      L
    ___________     _____     ________     ______     _________     _________     ______    _

    "Portfolio"     "VaR"     0.95         reject     88.952        0.0055565     accept    (
```

Run the TBF Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tbf` test results using the `TestLevel` optional input.

```
TestResults = tbf(vbt,'TestLevel',0.90)
TestResults =

    PortfolioID         VaRID          VaRLevel     TBF       LRatioTBF    PValueTBF     F
    ___________    _______________    ________    ______    _________    _________    _

    "Equity"       "Normal95"         0.95        reject    88.952       0.0055565    ac
    "Equity"       "Normal99"         0.99        reject    26.441        0.090095    re
    "Equity"       "Historical95"     0.95        reject     83.63        0.023609    ac
    "Equity"       "Historical99"     0.99        accept    16.456         0.22539    ac
    "Equity"       "EWMA95"           0.95        accept    72.545         0.12844    ac
    "Equity"       "EWMA99"           0.99        reject     41.66       0.0099428    re
```

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

Input Arguments

`vbt` — varbacktest object
object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = tbf(vbt,'TestLevel',0.99)`

`'TestLevel'` — Test confidence level
`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as 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
- `'TBFMin'` — Minimum value of observed times between failures
- `'TBFQ1'` — First quartile of observed times between failures
- `'TBFQ2'` — Second quartile of observed times between failures
- `'TBFQ3'` — Third quartile of observed times between failures
- `'TBFMax'` — Maximum value of observed times between failures
- `'TestLevel'` — Test confidence level

Note: For `tbf` test results, the terms `accept` and `reject` are used for convenience, technically a `tbf` test does not accept a model. Rather, the test fails to reject it.

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 TBFI tests

```
LRatioTBF = LRatioPOF + LRatioTBFI,
```

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 TBFI tests accept.

- "Haas's Time Between Failures or Mixed Kupiec's Test" on page 2-6
- varbacktest

References

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`

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: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `tbfi` test results.

```
TestResults = tbfi(vbt)
```

```
TestResults =

    PortfolioID    VaRID    VaRLevel     TBFI     LRatioTBFI    PValueTBFI    Observat
    ___________    _____    ________    ______    __________    __________    ________

    "Portfolio"    "VaR"    0.95        reject    88.491        0.0047475     1043
```

Run the TBFI Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'}
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tbfi` test results using the `TestLevel` optional input.

```
TestResults = tbfi(vbt,'TestLevel',0.90)
```

```
TestResults =

    PortfolioID        VaRID          VaRLevel     TBFI     LRatioTBFI    PValueTBFI
    ___________    ______________    ________    ______    __________    __________
    "Equity"       "Normal95"        0.95        reject    88.491        0.0047475
    "Equity"       "Normal99"        0.99        accept    22.929          0.15157
    "Equity"       "Historical95"    0.95        reject    82.719         0.022513
    "Equity"       "Historical99"    0.99        accept    16.228          0.18101
    "Equity"       "EWMA95"          0.95        accept    71.635          0.12517
    "Equity"       "EWMA99"          0.99        reject     31.83         0.080339
```

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

Input Arguments

`vbt` — varbacktest object
object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = tbfi(vbt,'TestLevel',0.99)`

`'TestLevel'` — Test confidence level
`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as 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

```
LRatioTBFI_i = LRatioTUFF(ni) = ...
             -2*(log(pVaR) + (ni - 1)*log(1 - pVaR) + ...
              ni*log(ni) - (ni - 1)*log(ni - 1)).
```

As with the `tuff` test, `LRatioTBFI_`i = `-2*log(pVaR)` if `n_`i = `1`.

The TBFI likelihood ratio `LRatioTBFI` is then the sum of the individual likelihood ratios for all times between failures

```
 LRatioTBFI = sum_{i=1:x} LRatioTBFI_i,
```

which is asymptotically distributed as a chi-square distribution with x degrees of freedom, where x is the number of failures.

The `p`-value of the `tbfi` test is the probability that a chi-square distribution with x degrees of freedom exceeds the likelihood ratio `LRatioTBFI`

```
 PValueTBFI = 1 - F(LRatioTBFI)
```

where `F` is the cumulative distribution of a chi-square variable with x degrees of freedom and x is the number of failures.

The result of the test is to accept if

```
F(LRatioTBFI) < F(TestLevel)
```

and reject otherwise, where `F` is the cumulative distribution of a chi-square variable with x degrees of freedom and x is the number of failures.

If there are no failures in the sample, the test statistic is not defined. This is handled the same as a TUFF test with no failures. For more information, see `tuff`.

- "Haas's Time Between Failures or Mixed Kupiec's Test" on page 2-6
- varbacktest

References

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

See Also

`bin` | `cc` | `cci` | `pof` | `runtests` | `summary` | `tbf` | `tl` | `tuff` | `varbacktest`

Introduced in R2016b

tl

Traffic light test for value-at-risk (VaR) backtesting

Syntax

```
TestResults = tl(vbt)
```

Description

`TestResults = tl(vbt)` generates the traffic light (TL) test for value-at-risk (VaR) backtesting.

Examples

Generate Traffic Light Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `tl` test results.

```
TestResults = tl(vbt)
TestResults =

    PortfolioID    VaRID    VaRLevel     TL      Probability     TypeI     Increase
    ___________    _____    ________    _____    ___________    _______    ________

    "Portfolio"    "VaR"    0.95        green    0.77913        0.26396    0
```

Run the TL Test for VaR Backtests for Multiple VaRs at Different Confidence Levels

Use the `varbacktest` constructor with name-value pair arguments to create a `varbacktest` object.

```
load VaRBacktestData
    vbt = varbacktest(EquityIndex,...
       [Normal95 Normal99 Historical95 Historical99 EWMA95 EWMA99],...
       'PortfolioID','Equity',...
       'VaRID',{'Normal95' 'Normal99' 'Historical95' 'Historical99' 'EWMA95' 'EWMA99'},
       'VaRLevel',[0.95 0.99 0.95 0.99 0.95 0.99])
```

```
vbt =

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tl` test results.

```
TestResults = tl(vbt)
```

```
TestResults =

    PortfolioID        VaRID         VaRLevel      TL       Probability      TypeI
    ___________    ______________    ________    ______    ___________    _________

    "Equity"       "Normal95"        0.95        green     0.77913          0.26396
    "Equity"       "Normal99"        0.99        yellow    0.97991          0.03686
    "Equity"       "Historical95"    0.95        green     0.85155          0.18232
    "Equity"       "Historical99"    0.99        green     0.74996           0.35269
    "Equity"       "EWMA95"          0.95        green     0.85155           0.18232
    "Equity"       "EWMA99"          0.99        yellow    0.99952         0.0011122
```

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

Input Arguments

`vbt` — varbacktest object
object

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

Output Arguments

`TestResults` — `tl` test results
table

`tl` test results, returned as a table where the rows correspond to all combinations of portfolio ID, VaR ID, and VaR levels to be tested. The columns correspond to the following information:

- `'PortfolioID'` — Portfolio ID for the given data
- `'VaRID'` — VaR ID for each of the VaR data columns provided
- `'VaRLevel'` — VaR level for the corresponding VaR data column
- `'TL'` — Categorical (ordinal) array with the categories `green`, `yellow`, and `red` that indicate the result of the traffic light `tl` test
- `'Probability'` — Cumulative probability of observing up to the corresponding number of failures
- `'TypeI'` — Probability of observing the corresponding number of failures or more if the model is correct
- `'Increase'` — Increase in the scaling factor

- `'Observations'` — Number of observations
- `'Failures'` — Number of failures

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 5-96.

The Basel Committee reports, as an example, a table of the three zones for 250 time periods and a VaR confidence level of 0.99. The increase in scaling factor in the table reported by Basel has some ad-hoc adjustments (rounding, etc.) not explicitly described in the Basel document. The following table compares the increase in scaling factor reported in the Basel document for the case of 250 periods and 0.99 % VaR confidence level, and the increase in the factors reported by the TL test.

Failures	Zone	Increase Basel	Increase TL
0	Green	0	0
1	Green	0	0
2	Green	0	0
3	Green	0	0
4	Green	0	0
5	Yellow	0.40	0.3982
6	Yellow	0.50	0.5295
7	Yellow	0.65	0.6520
8	Yellow	0.75	0.7680
9	Yellow	0.85	0.8791
10	Red	1	1

The `tl` function computes the scaling factor following the methodology described in the Basel document (see Bibliography) and is explained in the “Algorithms” on page 5-96 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 − `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,

```
Probability = Probability(X <= 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)` <= `0.95`
- Yellow: `0.95` < `F(x|N,p)` <= `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 >= 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*(zAssumed/zObserved - 1),
```

with the restriction that the increase cannot be negative or greater than `1`. The baseline scaling factor in the Basel rules is 3.

The `tl` function computes the scaling factor following this methodology, which is also described in the Basel document (see Bibliography). The `tl` function does not apply any ad-hoc adjustments.

- “Traffic Light Test” on page 2-3
- varbacktest

References

Basel Committee on Banking Supervision, *Supervisory Framework for the Use of 'Backtesting' in Conjunction with the Internal Models Approach to Market Risk Capital Requirements*. January, 1996, http://www.bis.org/publ/bcbs22.htm.

See Also

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

Introduced in R2016b

tuff

Time until first failure test for value-at-risk (VaR) backtesting

Syntax

```
TestResults = tuff(vbt)
TestResults = tuff(vbt,Name,Value)
```

Description

`TestResults = tuff(vbt)` generates the time until first failure (TUFF) test for value-at-risk (VaR) backtesting.

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

Examples

Generate TUFF Test Results

Create a `varbacktest` object.

```
load VaRBacktestData
vbt = varbacktest(EquityIndex,Normal95)
```

```
vbt = 

  varbacktest with properties:

    PortfolioData: [1043×1 double]
          VaRData: [1043×1 double]
      PortfolioID: "Portfolio"
            VaRID: "VaR"
         VaRLevel: 0.9500
```

Generate the `tuff` test results.

```
TestResults = tuff(vbt)


TestResults =

    PortfolioID    VaRID    VaRLevel     TUFF     LRatioTUFF    PValueTUFF    FirstFai
    ___________    _____    ________    ______    __________    __________    ________

    "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: [1043×1 double]
          VaRData: [1043×6 double]
      PortfolioID: "Equity"
            VaRID: [1×6 string]
         VaRLevel: [0.9500 0.9900 0.9500 0.9900 0.9500 0.9900]
```

Generate the `tuff` test results using the `TestLevel` optional input.

```
TestResults = tuff(vbt,'TestLevel',0.90)


TestResults =

    PortfolioID        VaRID          VaRLevel     TUFF     LRatioTUFF    PValueTUFF
    ___________    ______________    ________    ______    __________    __________
    "Equity"       "Normal95"        0.95        accept     1.7354        0.18773
    "Equity"       "Normal99"        0.99        accept    0.36686        0.54472
    "Equity"       "Historical95"    0.95        accept     1.5348         0.2154
    "Equity"       "Historical99"    0.99        accept    0.36686        0.54472
    "Equity"       "EWMA95"          0.95        accept    0.13304         0.7153
    "Equity"       "EWMA99"          0.99        accept    0.14596        0.70243
```

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

Input Arguments

`vbt` — varbacktest object
object

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

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `TestResults = tuff(vbt,'TestLevel',0.99)`

`'TestLevel'` — Test confidence level
`0.95` (default) | numeric between `0` and `1`

Test confidence level, specified as 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(pVaR) + (n - 1)*log(1 - pVaR) + ...
             n*log(n) - (n - 1)*log(n - 1)),
```

where `n` is the number of periods until the first failure and `pVaR` = 1 − `VaRLevel`. By the properties of the logarithm (if `n` = 1),

```
LRatioTUFF = -2*log(pVaR).
```

This is asymptotically distributed as a chi-square distribution with one degree of freedom.

The `p`-value of the `tuff` test is the probability that a chi-square distribution with one degree of freedom exceeds the likelihood ratio `LRatioTUFF`

```
 PValueTUFF = 1 - F(LRatioTUFF),
```

where `F` is the cumulative distribution of a chi-square variable with one degree of freedom.

The result of the test is to accept if

```
 F(LRatioTUFF) < F(TestLevel)
```

and reject otherwise, where `F` is the cumulative distribution of a chi-square variable with 1 degree of freedom.

If the sample has no failures, the test statistic is not defined. However, there are two cases distinguished here:

- If the number of observations is large enough that no matter when the first failure occurred it would be too late to pass the test, then the model is rejected. Technically, this happens if the number of observations `N` is larger than `1/pVaR` (large enough relative to the VaR confidence level) and if the test fails when `n = N+1` (the earliest observation for the first VaR failure). In this case, the likelihood ratio is reported for `n = N+1`, and the corresponding `p`-value.
- In all other cases, it is not possible to tell with certainty whether the result of the test would eventually be to accept or reject the model. There are ranges of possible first failure values that would result in accepting or rejecting the model. In these cases, the `tuff` function accepts the model and reports undefined (`NaN`) values for the likelihood ratio and `p`-value.

- “Kupiec’s POF and TUFF Tests” on page 2-4
- varbacktest

References

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`

Introduced in R2016b