

# Matlab for Finance Course: Session 3

Dr. Peter A. Bebbington

Brainpool AI



[peter@brainpool.ai](mailto:peter@brainpool.ai)



[bebbington](#)



[@peterbebbington](#)

November 30, 2024

# REVIEW OF SESSION 2

- MATLAB Basics
  - Scripts and Editor
  - Executing Scripts
  - Debugging Tools
- Data Manipulation
  - Array Operations
  - Matrix Operations
  - Array Indexing
- Programming Concepts
  - Boolean Logic
  - Control Flow (if-else, while, for, switch)
  - Error Handling
- Best Practices
  - Common Mistakes
  - Performance Tips

# SESSION OBJECTIVES

- MATLAB Fundamentals
  - Function creation and usage
  - File I/O operations
  - Data formatting and display
- Data Handling
  - Stock data import (`getMarketDataViaTiingo`)
  - Time series manipulation
  - Matrix operations
- Financial Analysis
  - Returns calculation
  - Statistical analysis
  - Volatility estimation
- Risk Measures
  - VaR and CVaR implementation
  - Backtesting framework

The presentation includes MATLAB implementation for all topics.

# FUNCTIONS - Basic Structure

- Functions in MATLAB are defined in separate files with the .m extension
- Basic function structure:

```
1 function [output1, output2] = myFunction(input1, input2)
2     % Function description/help comment
3     % input1: description of first input
4     % input2: description of second input
5     % output1: description of first output
6     % output2: description of second output
7
8     % Function body
9     output1 = someCalculation(input1);
10    output2 = anotherCalculation(input2);
11 end
```

- Function name must match the filename (e.g., myFunction.m)
- Help comments are displayed when using `help myFunction`

# FUNCTIONS - Example

- Example of a plotting function:

```
1 function [y, h] = mysin_func(x)
2     % mysin takes the input argument and returns the
3     % sin of the argument and plots the result
4     y = sin(x);
5     h = figure;
6     plot(x, y);
7     xlabel("x");
8     ylabel("sin(x)");
9 end
```

- Call the function:

```
1 [y, h] = mysin_func(0:pi/50:2*pi);
```

# FILE TYPES

Most financial data that will be imported into MATLAB will come in three main forms:

- .csv: Comma Separated Values

```
1 Date,Open,High,Low,Close
2 2024-01-01,100.5,101.2,99.8,100.9
```

- .tsv: Tab Separated Values

```
1 Date    Open    High    Low    Close
2 2024-01-01 100.5  101.2  99.8   100.9
```

- .txt: Text data in some format

```
1 # Financial Data
2 100.5 101.2 99.8 100.9
```

Other types of data that will be imported include:

- .xls: Excel files
- .xml: Extensible Markup Language
- .mat: MATLAB binary files (loaded using `load data.mat`)

It is important to understand the organisation of different data types in order to understand the memory requirements for data.

# FILE FUNCTIONS

| Command                               | Meaning                         |
|---------------------------------------|---------------------------------|
| <code>fopen(filename)</code>          | Open a file                     |
| <code>fclose(fid)</code>              | Close a file                    |
| <code>fread(fid)</code>               | Read binary data                |
| <code>fwrite(fid,A,precision)</code>  | Write binary data               |
| <code>fprintf(fid,A,precision)</code> | Write formatted data            |
| <code>fscanf(fid,format)</code>       | Read formatted data             |
| <code>sprintf(format,A)</code>        | Write to a string               |
| <code>sscanf(s,format)</code>         | Read string                     |
| <code>ferror(fid)</code>              | Query about errors              |
| <code>feof(fid)</code>                | Test for end of file            |
| <code>fseek(fid,offset,origin)</code> | Set the file position indicator |

- Writing and reading numeric data:

```
1 A = [1 2 3 4 5];  
2 fid = fopen('some_data.txt', 'w');  
3 fwrite(fid, A);  
4 fclose(fid);  
5  
6 fid = fopen('some_data.txt', 'r');  
7 fread(fid)  
8 fclose(fid);
```

- Writing and reading text:

```
1 str = 'this is a test';  
2 fid = fopen('test.txt', 'w');  
3 fwrite(fid, str, 'char');  
4 fclose(fid);
```



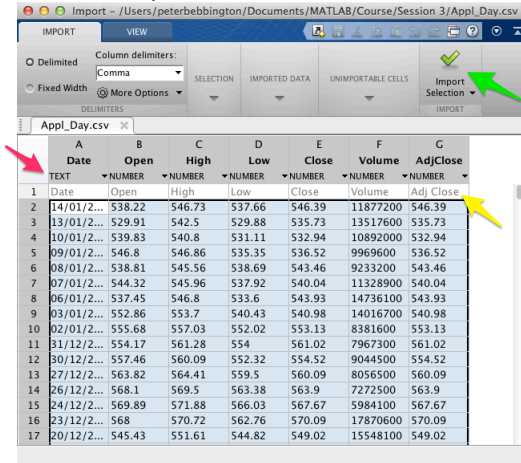
# TIMES TABLE EXAMPLE

```
1 display('Times Table:')
2 fprintf(1,' X '); % Write to command window
3 for i = 0:9
4     fprintf(1,'%2d ',i);
5 end
6 fprintf(1,'\n');
7 for i = 0:9
8     fprintf(1,'%2d ',i);
9     for j = 0:9
10        fprintf(1,'%2d ',i*j);
11    end
12    fprintf(1,'\n');
13 end
```

Output shows formatted multiplication table 0-9

# IMPORT TOOL

- Simply drag and drop a “.csv” file to the command window of Matlab to import data



Import - /Users/peterbebbington/Documents/MATLAB/Course/Session 3/Apl\_Day.csv

IMPORT VIEW

Delimited Column delimiters: Comma  
Fixed Width More Options

SELECTION IMPORTED DATA UNIMPORTABLE CELLS Import Selection IMPORT

Appl\_Day.csv

|    | A<br>Date  | B<br>Open | C<br>High | D<br>Low | E<br>Close | F<br>Volume | G<br>AdjClose |
|----|------------|-----------|-----------|----------|------------|-------------|---------------|
|    | TEXT       | NUMBER    | NUMBER    | NUMBER   | NUMBER     | NUMBER      | NUMBER        |
| 1  | Date       | Open      | High      | Low      | Close      | Volume      | Adj Close     |
| 2  | 14/01/2... | 538.22    | 546.73    | 537.66   | 546.39     | 11877200    | 546.39        |
| 3  | 13/01/2... | 529.91    | 542.5     | 529.88   | 535.73     | 13517600    | 535.73        |
| 4  | 10/01/2... | 539.83    | 540.8     | 531.11   | 532.94     | 10892000    | 532.94        |
| 5  | 09/01/2... | 546.8     | 546.86    | 535.35   | 536.52     | 9969600     | 536.52        |
| 6  | 08/01/2... | 538.81    | 545.56    | 538.69   | 543.46     | 9233200     | 543.46        |
| 7  | 07/01/2... | 544.32    | 545.96    | 537.92   | 540.04     | 11328900    | 540.04        |
| 8  | 06/01/2... | 537.45    | 546.8     | 533.6    | 543.93     | 14736100    | 543.93        |
| 9  | 03/01/2... | 552.86    | 553.7     | 540.43   | 540.98     | 14016700    | 540.98        |
| 10 | 02/01/2... | 555.68    | 557.03    | 552.02   | 553.13     | 8381600     | 553.13        |
| 11 | 31/12/2... | 554.17    | 561.28    | 554      | 561.02     | 7967300     | 561.02        |
| 12 | 30/12/2... | 557.46    | 560.09    | 552.32   | 554.52     | 9044500     | 554.52        |
| 13 | 27/12/2... | 563.82    | 564.41    | 559.5    | 560.09     | 8056500     | 560.09        |
| 14 | 26/12/2... | 568.1     | 569.5     | 563.38   | 563.9      | 7272500     | 563.9         |
| 15 | 24/12/2... | 569.89    | 571.88    | 566.03   | 567.67     | 5984100     | 567.67        |
| 16 | 23/12/2... | 568       | 570.72    | 562.76   | 570.09     | 17870600    | 570.09        |
| 17 | 20/12/2... | 545.43    | 551.61    | 544.82   | 549.02     | 15548100    | 549.02        |

- You can edit; data type (pink arrow), data field name (yellow arrow) and import data (green arrow)

- Now that we have the data in Matlab we can create a workspace

```
1 >> whos
2   Name      Size      Bytes Class      Attributes
3   Date      252x1      2016  double
4   High      252x1      2016  double
5   Low       252x1      2016  double
6   Open      252x1      2016  double
7   Volume    252x1      2016  double
8   Close     252x1      2016  double
```

- >> clear

- We'll use Tiingo API to download historical stock data
- Steps to get started:
  - Sign up for free account at [Tiingo](#)
  - Generate API key from your account dashboard
  - Set up API key in MATLAB:

```
setenv('TIINGO_API_KEY', 'your_api_key_here');
```

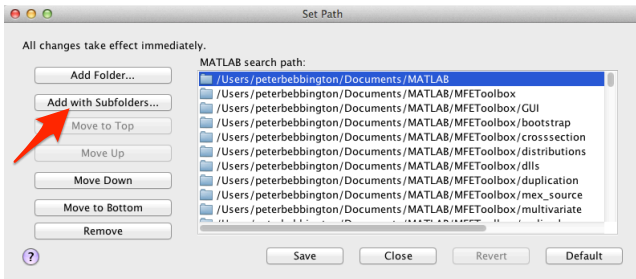
- Advantages of Tiingo:
  - Reliable and maintained API
  - High-quality financial data
  - Free tier available for academic use

# TIINGO API EXAMPLE

```
% Download one year of daily data for AAPL and MSFT
>> startdt = '2023-01-01';
>> enddt = '2023-12-31';
% Get AAPL data
>> aapl = tiingo_prices('AAPL', startdt, enddt, 'daily');
% Get MSFT data
>> msft = tiingo_prices('MSFT', startdt, enddt, 'daily');
% Data structure example:
>> aapl
    date: [252x1 datetime]
    open: [252x1 double]
    high: [252x1 double]
    low:  [252x1 double]
    close: [252x1 double]
    volume: [252x1 double]
```

- Matlab has its own Econometrics toolbox with rich functionality
- There are also third-party toolboxes that can be installed which can help with time series analysis for summer projects
- Two recommended toolboxes:
  - MFEToolbox: [www.kevinsheppard.com](http://www.kevinsheppard.com)
  - JPLV7: [www.spatial-econometrics.com](http://www.spatial-econometrics.com)

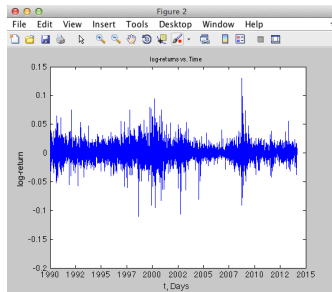
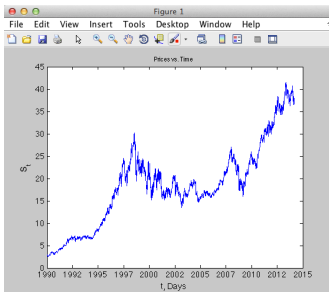
# INSTALLING TOOLBOXES



- Click “Add with Subfolders...” (red arrow) and Locate the two toolboxes and save.

# FINANCIAL SERIES

- A good starting point when analyzing financial time series is to plot basic quantities against time, such as price, log-returns, volume, etc...





# SAMPLE STATISTICS: Basic Moments

- Basic statistics measure the shape and central tendencies of returns

```
1 % Basic Statistics
2 mean_lr   = mean(lreturns); % First moment
3 std_lr    = std(lreturns);  % Second moment (volatility)
4 ske_lr    = skewness(lreturns); % Third moment (asymmetry)
5 kurt_lr   = kurtosis(lreturns); % Fourth moment (tail
    thickness)
```

- For financial returns, we typically expect:
  - Mean close to zero
  - Significant volatility
  - Negative skewness (more extreme losses than gains)
  - High kurtosis (fat tails)

- Statistical tests help verify stylized facts of returns

```
1 % Serial Correlation Tests
2 sacf_lr   = sacf(lreturns, 1, 1, 0); % Return predictability
3 sacf_lr2  = sacf(lreturns.^2, 1, 1, 0); % Volatility
           clustering
4
5 % Normality Tests
6 [jb_lr, pval] = jarquebera(lreturns); % Jarque-Bera test
7 kst_lr       = kstest(lreturns);      % Kolmogorov-Smirnov
```

- Test Interpretations:
  - Serial correlation tests check for time dependencies
  - Normality tests verify distribution assumptions

- Any Gaussian distributed random variable can be normalized:

$$X \sim \mathcal{N}(\mu, \sigma^2)$$

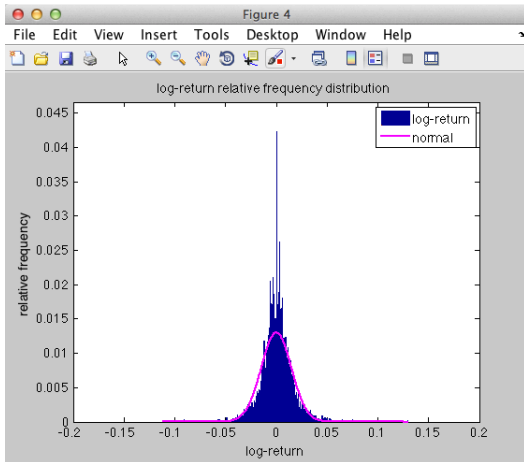
$$Z = \frac{X - \mu}{\sigma} \quad (\text{standardization})$$

$$X = \sigma Z + \mu \quad (\text{reconstruction})$$

- Analysis of return time series is better in this form for comparison between different time series such as a portfolio

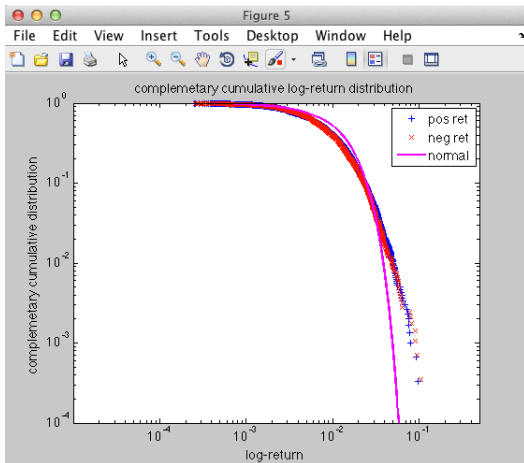
# COMPARISON WITH A GAUSSIAN

- Here we make a comparison of the empirical histogram against a parametrized normal distribution



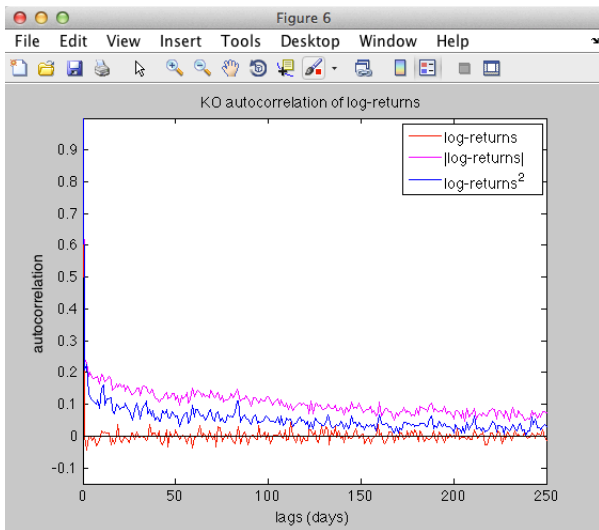
# COMPLEMENTARY CUMULATIVE DISTRIBUTION

- We see in this log-log plot the empirical time series differs from the tails of a normal distribution, indicating heavier tails in the data



# AUTOCORRELOGRAM

- Shows correlation between returns at different time lags
- Helps identify patterns and dependencies in the time series



# VOLATILITY CALCULATION: The Cumsum Trick

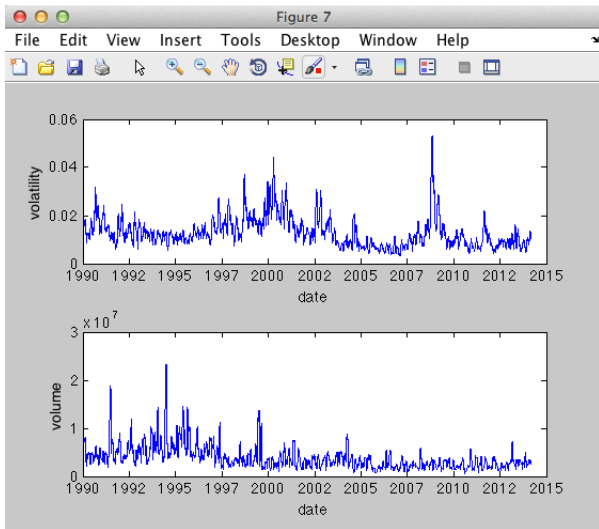
- Efficient method to calculate rolling window volatility using cumulative sums

```
1 % Step 1: Calculate cumulative sums
2 window = 252; % Trading days in a year
3 Y1 = cumsum(lreturns(:,1), 1); % Cumsum of returns
4 Y2 = cumsum(lreturns(:,1).^2, 1); % Cumsum of squared returns
5 % Step 2: Calculate rolling window volatility
6 volatility = sqrt(
7     (Y2((window+1):end) - Y2(1:(end-window))) / window %
8     Variance
9     - ((Y1((window+1):end) - Y1(1:(end-window))) / window).^2
10    % Mean^2);
```

- Key advantages:
  - Vectorized operations (faster than loops)
  - Memory efficient (only stores cumsum)
  - Based on variance formula:  $\sigma^2 = E[X^2] - (E[X])^2$
- Window calculations:
  - $Y2(t+w) - Y2(t)$ : Sum of squared returns in window
  - $Y1(t+w) - Y1(t)$ : Sum of returns in window
  - Division by window gives rolling means

# VOLATILITY

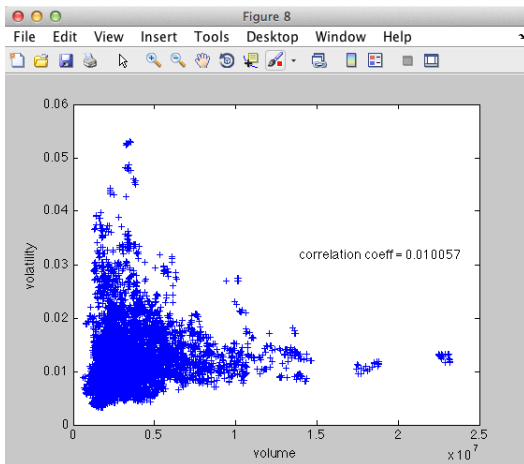
- Volatility measures the dispersion of returns over time
- Calculated using a rolling window of 252 trading days





# VOLATILITY Vs. VOLUME

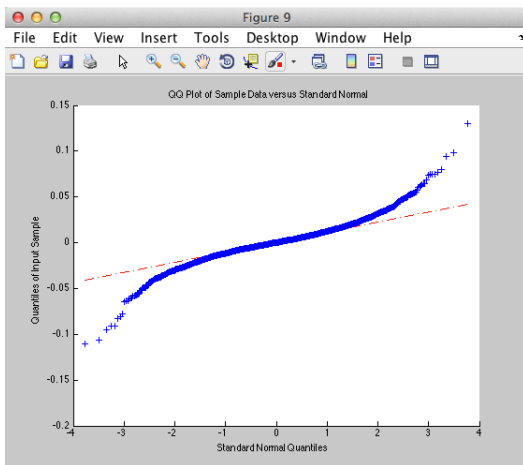
- Higher trading volume often associated with higher volatility
- Correlation coefficient indicates strength of relationship



- Important for trading strategy and risk management

# QQPLOT (QUANTILE-QUANTILE PLOT)

- Compares empirical distribution against theoretical normal
- Straight line indicates normality; deviations show fat tails



- Financial returns typically show deviations at the tails

# VALUE AT RISK (VaR) - Mathematical Definition

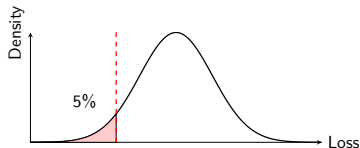
- VaR is formally defined as:

$$\text{VaR}_\alpha \triangleq \inf\{l \in \mathbb{R} : F_L(l) \geq \alpha\}$$

- Breaking down the equation:
  - $\text{VaR}_\alpha$ : Value at Risk at confidence level  $\alpha$
  - $\inf$ : Infimum (minimum value)
  - $l \in \mathbb{R}$ : Loss value in real numbers
  - $F_L(l)$ : Cumulative distribution function of losses
  - $\geq \alpha$ : Probability threshold (e.g., 0.95)
- In simpler terms:
  - VaR is the smallest loss value
  - Where the probability of exceeding this loss
  - Is less than or equal to  $1 - \alpha$  (e.g., 5%)

# VALUE AT RISK (VaR) - Visualization

- VaR represents a threshold where probability of larger losses is  $1 - \alpha$



- Example interpretation:
  - $\text{VaR}_{95\%} = \$100$  means there's a 5% chance of losing more than \$100
  - Red area shows probability of extreme losses

# VAR ESTIMATION IN MATLAB

- Parametric estimation (assuming normal distribution):  
`ValueAtRisk = portvrisk(PortReturn, PortRisk,  
RiskThreshold, PortValue)`
- Function parameters:
  - PortReturn: Expected portfolio return
  - PortRisk: Portfolio standard deviation
  - RiskThreshold: Confidence level (e.g., 0.95)
  - PortValue: Current portfolio value
- Limitations:
  - Assumes normal distribution
  - May underestimate tail risk
  - Compare with empirical estimation

# CONDITIONAL VALUE AT RISK (CVaR) - Mathematical Definition

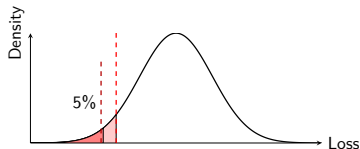
- CVaR is formally defined as:

$$\text{CVaR}_\alpha = \mathbb{E}[L|L \geq \text{VaR}_\alpha] = \frac{1}{1-\alpha} \int_\alpha^1 \text{VaR}_\gamma(L) d\gamma$$

- Breaking down the equation:
  - $\text{CVaR}_\alpha$ : Expected loss exceeding VaR
  - $\mathbb{E}[L|L \geq \text{VaR}_\alpha]$ : Conditional expectation
  - $\frac{1}{1-\alpha}$ : Normalization factor
  - $\text{VaR}_\gamma(L)$ : VaR at confidence level  $\gamma$
- In simpler terms:
  - CVaR is the average loss in the worst  $(1-\alpha)\%$  of cases
  - More conservative than VaR
  - Accounts for the shape of the tail distribution

# CONDITIONAL VALUE AT RISK (CVaR) - Visualization

- CVaR measures the average loss beyond VaR



- Example interpretation:
  - If  $\text{VaR}_{95\%} = \$100$ ,  $\text{CVaR}_{95\%} = \$130$  for normally distributed returns
  - CVaR represents average loss in worst 5% of cases
  - Darker red area shows the region CVaR measures

- Conditional Value at Risk (CVaR) calculations:

```
1 % Parametric CVaR
2 m = mean(lreturns(:,1));
3 s = std(lreturns(:,1));
4 CVaR_95 = -m + s*(normpdf(norminv(0.05,0,1),0,1))/(1-0.95);
5 CVaR_99 = -m + s*(normpdf(norminv(0.01,0,1),0,1))/(1-0.99);
6
7 % Empirical CVaR
8 CVaR_95_emp = -mean(slr(1:ceil(N*0.05))); % 5% loss
9 CVaR_99_emp = -mean(slr(1:ceil(N*0.01))); % 1% loss
```

- CVaR represents the expected loss exceeding VaR
- Also known as Expected Shortfall (ES)
- More coherent risk measure than VaR



# INTERPRETING VAR RESULTS

- Sample VaR Results:

| Method     | 95% VaR | 99% VaR |
|------------|---------|---------|
| Parametric | 1.75%   | 2.49%   |
| Empirical  | 1.56%   | 3.10%   |

- Key Insights:

- Parametric assumes normal distribution
- Empirical uses actual historical data
- Difference suggests non-normal distribution
- Higher empirical 99% VaR indicates fat tails

- Practical Example (for \$1,000,000 portfolio):

- 95% Parametric: Max loss of \$17,520
- 95% Empirical: Max loss of \$15,612
- 99% Parametric: Max loss of \$24,894
- 99% Empirical: Max loss of \$30,968

- Recommendation:

- Rely more on empirical values
- Better captures actual risk characteristics
- More conservative at higher confidence levels

# COMPARISON OF RISK MEASURES

- Key differences between VaR and CVaR:
  - VaR: Maximum loss at confidence level
  - CVaR: Average loss beyond VaR

| Property            | VaR      | CVaR      |
|---------------------|----------|-----------|
| Coherence           | No       | Yes       |
| Tail Sensitivity    | Limited  | High      |
| Ease of Calculation | Higher   | Lower     |
| Regulatory Use      | Basel II | Basel III |

# BACKTESTING - OVERVIEW

- Purpose of Backtesting:
  - Validate risk model accuracy
  - Meet regulatory requirements
  - Improve risk estimation
- Key Concepts:
  - VaR violation: When actual loss exceeds VaR estimate
  - Expected violation frequency:  $(1 - \alpha)$  for  $\text{VaR}_\alpha$
  - Example: For 95% VaR, expect violations in 5% of cases
- Backtesting Period:
  - Typically 250-500 trading days
  - Basel requirement: Minimum 250 days
  - Need sufficient data for statistical significance

# BACKTESTING RISK MEASURES

- Verify accuracy of risk measures:

```
1 % Count VaR violations
2 violations = sum(returns < -VaR_95);
3 violation_rate = violations/length(returns);
4 % Kupiec test
5 [h,p] = kupiectest(violations, length(returns), 0.05);
```

- Testing approaches:
  - Violation ratio analysis
    - Compares actual violations to expected frequency
    - For 95% VaR, expect violations 5% of the time
    - Ratio  $> 1$  indicates model underestimates risk
  - Independence tests
    - Examines clustering of violations
    - Uses Christoffersen's test for independence
    - Clustering suggests model weakness in stress periods
  - Dynamic backtesting
    - Uses rolling windows (typically 250 days)
    - Updates risk estimates with new data
    - Better captures changing market conditions

- Basic VaR Violation Test:

```
1 % Count VaR violations
2 violations = sum(returns < -VaR_95);
3 violation_rate = violations/length(returns);
4
5 % Expected rate for 95% VaR is 0.05
6 excess = (violation_rate - 0.05)/0.05;
7 fprintf('Violation excess: %.2f%%\n', excess*100);
```

- Rolling Window Analysis:

```
1 window = 252; % One trading year
2 for t = window:length(returns)
3     % Estimate VaR using rolling window
4     rolling_var = std(returns(t-window+1:t));
5     rolling_VaR = norminv(0.05)*rolling_var;
6
7     % Check for violation
8     violations(t) = returns(t) < -rolling_VaR;
9 end
```

# STATISTICAL TESTS FOR BACKTESTING

- Kupiec Test (Unconditional Coverage):
  - Tests if violation frequency matches expected rate
  - Null hypothesis: Observed rate = Expected rate
  - Uses likelihood ratio test
- Christoffersen Test (Conditional Coverage):
  - Tests independence of violations
  - Checks for violation clustering
  - Combines tests for frequency and independence
- Dynamic Quantile Test:
  - Tests if violations are predictable
  - Uses regression-based approach
  - More powerful than basic tests

# ADVANCED BACKTESTING TECHNIQUES

- Purpose of Multipliers:
  - Determines capital requirements for market risk
  - Penalizes banks for inaccurate VaR models
  - Formula:  $\text{Capital} = \text{VaR} \times \text{Multiplier}$
- Traffic Light Approach (Basel):

| Zone   | Violations | Multiplier |
|--------|------------|------------|
| Green  | 0-4        | 3.00       |
| Yellow | 5-9        | 3.40-3.85  |
| Red    | 10+        | 4.00       |

- Duration-Based Tests:

```
1 % Time between violations
2 durations = diff(find(violations));
3 [h,p] = duration_test(durations, alpha);
```

- Multiple VaR Levels:
  - Test at different confidence levels
  - Compare 95%, 99%, 99.9% VaR
  - Check consistency across levels

# KUPIEC TEST (UNCONDITIONAL COVERAGE)

- Purpose:
  - Tests if the observed violation frequency equals expected rate
  - Known as Proportion of Failures (POF) test
  - Fundamental VaR validation tool
- Test Statistics:
  - Let  $N$  = number of violations
  - Let  $T$  = total number of observations
  - Let  $p$  = expected violation rate (e.g., 0.05 for 95% VaR)
  - Let  $\hat{p} = N/T$  = observed violation rate
- Likelihood Ratio Test:

$$LR_{POF} = -2 \ln \left[ \frac{(1-p)^{T-N} p^N}{(1-\hat{p})^{T-N} \hat{p}^N} \right] \\ \sim \chi^2(1)$$



# INTERPRETING KUPIEC TEST RESULTS

- Null Hypothesis:
  - $H_0$ : The model's violation rate equals the expected rate
  - $H_1$ : The model's violation rate differs from expected
- Decision Rules:
  - Reject  $H_0$  if  $LR_{POF} > \chi^2_{1,\alpha}$
  - Typical significance level  $\alpha = 0.05$
  - Critical value  $\chi^2_{1,0.05} = 3.841$
- Limitations:
  - Only tests violation frequency
  - Ignores clustering of violations
  - Low power for small samples
  - Should be combined with other tests

# BACKTESTING RESULTS

- Detailed analysis of VaR model performance:

| Metric              | 95% VaR | 99% VaR |
|---------------------|---------|---------|
| Actual Violations   | 142     | 69      |
| Expected Violations | 187     | 37      |
| Violation Rate      | 3.79%   | 1.84%   |
| Kupiec Test p-value | 0.0004  | 0.0000  |
| Model Rejected      | Yes     | Yes     |

- Distribution characteristics:
  - Mean Return: 0.033%
  - Standard Deviation: 1.078%
  - Skewness: -0.783
  - Kurtosis: 12.630
- Key findings:
  - 95% VaR overestimates risk (conservative)
  - 99% VaR underestimates risk (inadequate)
  - Heavy-tailed return distribution
  - Both models rejected by Kupiec test