Matlab for Finance Course: Session 3

Dr. Peter A. Bebbington

Brainpool AI

peter@brainpool.ai

bebbington @peterbebbington

November 29, 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:

```matlab
function [output1, output2] = myFunction(input1, input2)
   % Function description/help comment
   % input1: description of first input
   % input2: description of second input
   % output1: description of first output
   % output2: description of second output

   % Function body
   output1 = someCalculation(input1);
   output2 = anotherCalculation(input2);
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:

```
function [y, h] = mysin_func(x)
    % mysin takes the input argument and returns the
    % sin of the argument and plots the result
    y = sin(x);
    h = figure;
    plot(x, y);
    xlabel("x");
    ylabel("sin(x)");
end
```

- Call the function:

```
[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

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

- `.tsv`: Tab Separated Values

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

- `.txt`: Text data in some format

```
    # Financial Data
    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**
`fopen(filename)`	Open a file
`fclose(fid)`	Close a file
`fread(fid)`	Read binary data
`fwrite(fid,A,precision)`	Write binary data
`fprintf(fid,A,precision)`	Write formatted data
`fscanf(fid,format)`	Read formatted data
`sprintf(format,A)`	Write to a string
`sscanf(s,format)`	Read string
`ferror(fid)`	Query about errors
`feof(fid)`	Test for end of file
`fseek(fid,offset,origin)`	Set the file position indicator

I/O EXAMPLES

- Writing and reading numeric data:

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

fid = fopen('some_data.txt', 'r');
fread(fid)
fclose(fid);
```

- Writing and reading text:

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

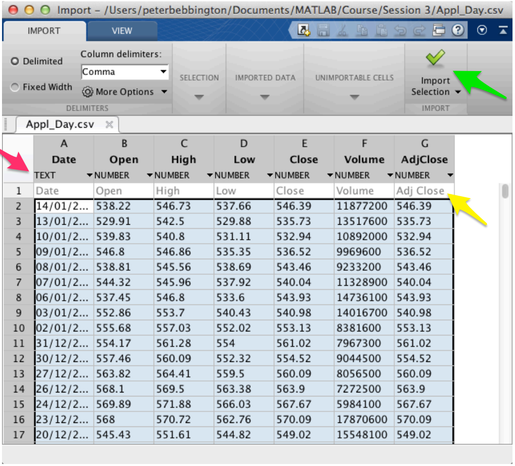
TIMES TABLE EXAMPLE

```
display('Times Table:')
fprintf(1,' X '); % Write to command window
for i = 0:9
    fprintf(1,'%2d ',i);
end
fprintf(1,'\n');
for i = 0:9
    fprintf(1,'%2d ',i);
    for j = 0:9
        fprintf(1,'%2d ',i*j);
    end
    fprintf(1,'\n');
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



	A	B	C	D	E	F	G
	Date	Open	High	Low	Close	Volume	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)

WORKSPACE

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

```
>> whos
  Name        Size     Bytes Class    Attributes
  Date      252x1      2016   double
  High      252x1      2016   double
  Low       252x1      2016   double
  Open      252x1      2016   double
  Volume    252x1      2016   double
  Close     252x1      2016   double
```

- `>> clear`

TIINGO DATA API

- 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

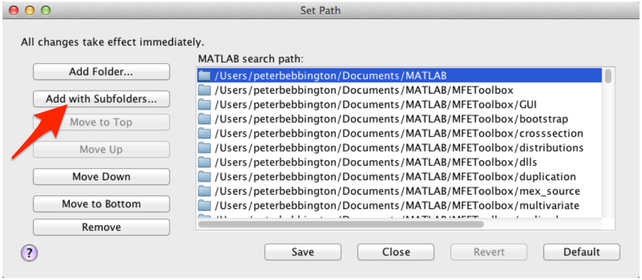
TIINGO API EXAMPLE

```
% 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]
```

ECONOMETRICS TOOLBOXES

- 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
    - JPLV7: www.spatial-econometrics.com

INSTALLING TOOLBOXES



All changes take effect immediately.

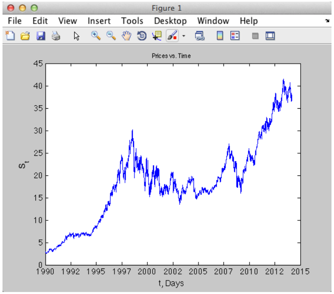
MATLAB search path:

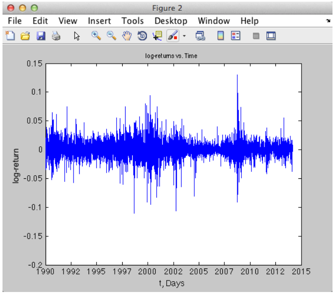
```
/Users/peterbebbington/Documents/MATLAB
/Users/peterbebbington/Documents/MATLAB/MFEToolbox
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/GUI
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/bootstrap
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/crosssection
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/distributions
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/dlls
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/duplication
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/mex_source
/Users/peterbebbington/Documents/MATLAB/MFEToolbox/multivariate
```

- 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

```
% Basic Statistics
mean_lr    = mean(lreturns);   % First moment
std_lr     = std(lreturns);    % Second moment (volatility)
ske_lr     = skewness(lreturns); % Third moment (asymmetry)
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)

SAMPLE STATISTICS: Tests

- Statistical tests help verify stylized facts of returns

```
% Serial Correlation Tests
sacf_lr   = sacf(lreturns, 1, 1, 0); % Return predictability
sacf_lr2  = sacf(lreturns.^2, 1, 1, 0); % Volatility
     clustering

% Normality Tests
[jb_lr, pval] = jarquebera(lreturns); % Jarque-Bera test
kst_lr      = kstest(lreturns);      % Kolmogorov-Smirnov
```

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

NORMALIZING

- Any Gaussian distributed random variable can be normalized:

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

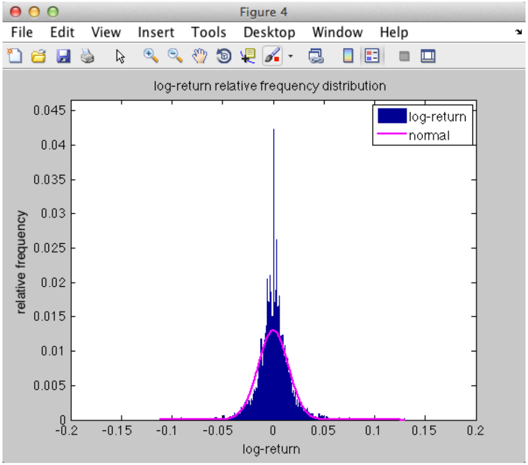
$$Z = \frac{X - \mu}{\sigma} \quad \text{(standardization)}$$

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

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

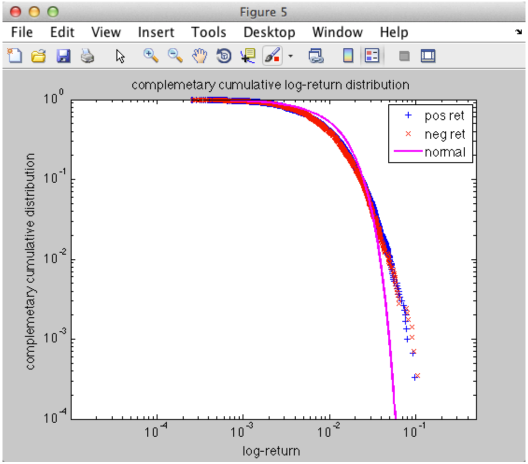
COMPARISON WITH A GAUSSIAN

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



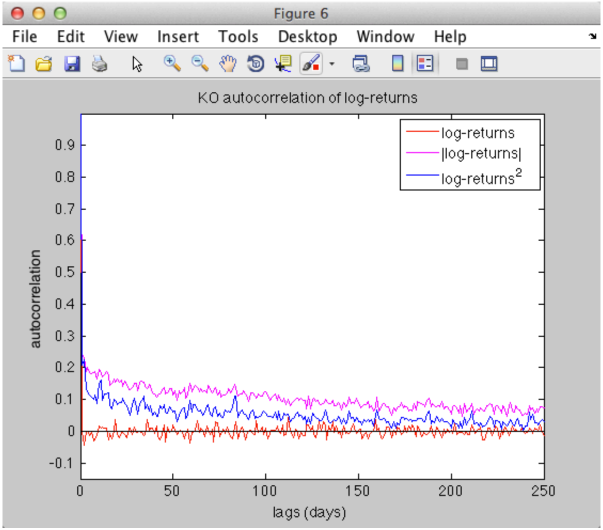
COMPLEMENTARY CUMULATIVE DISTRIBUTION

- 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

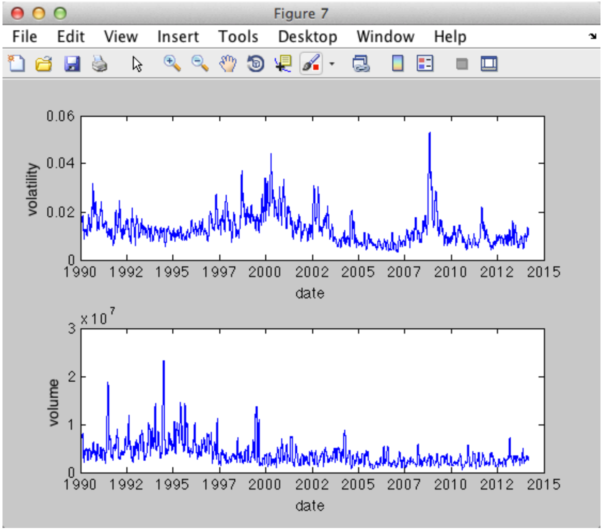
- Efficient method to calculate rolling window volatility using cumulative sums

```
% Step 1: Calculate cumulative sums
window = 252; % Trading days in a year
Y1 = cumsum(lreturns(:,1), 1); % Cumsum of returns
Y2 = cumsum(lreturns(:,1).^2, 1); % Cumsum of squared returns
% Step 2: Calculate rolling window volatility
volatility = sqrt(
    (Y2((window+1):end) - Y2(1:(end-window))) / window %
         Variance
    - ((Y1((window+1):end) - Y1(1:(end-window))) / window).^2
         % Mean²);
```

- 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

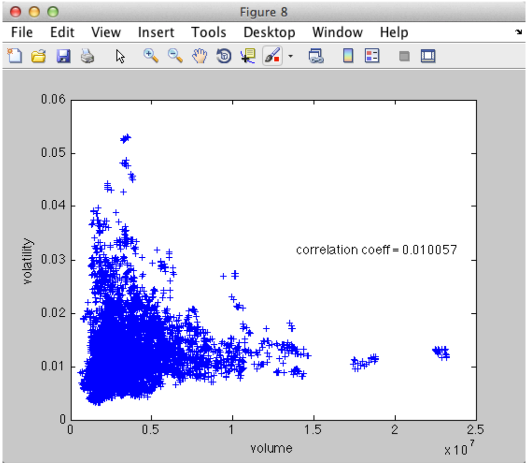
VOLATILITY

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



VOLATILITY Vs. VOLUME

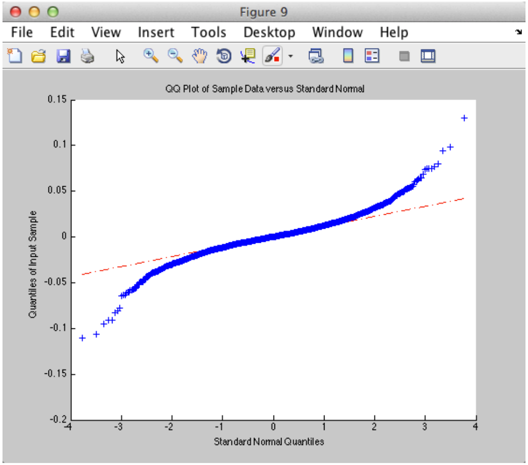
- Higher trading volume often associated with higher volatility
- Correlation coefficient indicates strength of relationship



- 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

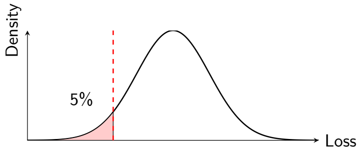
- VaR is formally defined as:

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

- Breaking down the equation:
  - VaR_α: Value at Risk at confidence level α
  - $\inf$: 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

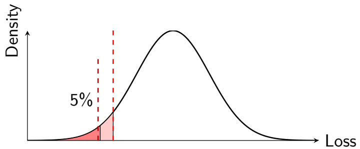
- CVaR is formally defined as:

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

- Breaking down the equation:
  - CVaR_α: 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 γ
- 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

PARAMETRIC CVaR

- Conditional Value at Risk (CVaR) calculations:

```
% Parametric CVaR
m = mean(lreturns(:,1));
s = std(lreturns(:,1));
CVaR_95 = -m + s*(normpdf(norminv(0.05,0,1),0,1))/(1-0.95);
CVaR_99 = -m + s*(normpdf(norminv(0.01,0,1),0,1))/(1-0.99);

% Empirical CVaR
CVaR_95_emp = -mean(slr(1:ceil(N*0.05))); % 5% loss
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 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:

```matlab
% Count VaR violations
violations = sum(returns < -VaR_95);
violation_rate = violations/length(returns);
% Kupiec test
[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

BACKTESTING - IMPLEMENTATION

- Basic VaR Violation Test:

```
% Count VaR violations
violations = sum(returns < -VaR_95);
violation_rate = violations/length(returns);

% Expected rate for 95% VaR is 0.05
excess = (violation_rate - 0.05)/0.05;
fprintf('Violation excess: %.2f%%\n', excess*100);
```

- Rolling Window Analysis:

```
window = 252; % One trading year
for t = window:length(returns)
    % Estimate VaR using rolling window
    rolling_var = std(returns(t-window+1:t));
    rolling_VaR = norminv(0.05)*rolling_var;

    % Check for violation
    violations(t) = returns(t) < -rolling_VaR;
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: Capital $=$ VaR $\times$ 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:

```
% Time between violations
durations = diff(find(violations));
 [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