Financial Risk Forecasting
Seminar Week 7: Multivariate Volatility Models

Jon Danielsson
London School of Economics

Version 4.0

©Jon Danielsson. All rights reserved

7 Multivariate volatility

7.1 Why multivariate volatility models matter for financial risk

Understanding how asset volatilities move together is essential for portfolio risk management and derivatives pricing. Correlations change over time and spike during crises when diversification benefits disappear precisely when needed most. Multivariate models capture these dynamics for accurate portfolio VaR and optimal hedging strategies.

In this seminar, we will use the `Returns.RData` file that we created in Seminar 2. We focus on coding the multivariate GARCH with the `rmgarch` package. You will need to install `rmgarch` if you have not already done so.

For more detail, see Multivariate volatility models in the R notebook.

7.2 The plan for this week

1. Introduce multivariate volatility models
2. Build a bivariate EWMA model
3. Run DCC models with different specifications
4. Compare models

7.3 Loading data and libraries

Load the necessary libraries.

```
library(rmgarch)
```

```
load('Returns.RData')
```

7.4 Understanding multivariate volatility with EWMA

7.4.1 What is EWMA?

Exponentially Weighted Moving Average (EWMA) is one of the simplest approaches to multivariate volatility modelling. Unlike sample covariance which treats all observations equally, EWMA gives more weight to recent observations, making it responsive to changing market conditions.

EWMA serves as a foundation for understanding more complex models like DCC-GARCH. It estimates the covariance matrix at time t using:

$$\Sigma_t = \lambda \times \Sigma_{t-1} + (1-\lambda) \times y_{t-1} \times y'_{t-1}$$

where λ is the decay parameter (typically 0.94) and y_t is the vector of returns.

7.4.2 Four-asset portfolio example

We use four assets representing different sectors to demonstrate multivariate volatility modelling:

- JPM (JPMorgan Chase): Financial sector — sensitive to interest rates and credit conditions
- C (Citigroup): Financial sector — provides within-sector correlation example
- XOM (Exxon Mobil): Energy sector — commodity exposure with different risk drivers
- AAPL (Apple): Technology sector — growth stock with distinct volatility patterns

This selection provides examples of both high correlation (JPM-C) and lower correlation (tech-financial, energy-tech) pairs, making it ideal for demonstrating time-varying correlation dynamics.

```
# Convert to matrix for matrix algebra operations
y_4asset = as.matrix(Returns[,c("JPM","C","XOM","AAPL")])
```

We then do the setup. For four assets, we track 10 unique elements: 4 variances and 6 covariances (since the covariance matrix is symmetric).

```
dim(y_4asset)
```

```
[1] 6439    4
```

```
EWMA = matrix(nrow=dim(y_4asset)[1],ncol=3)
lambda = 0.94
S = cov(y_4asset)
S
```

```
              JPM            C           XOM          AAPL
JPM  0.0005431774 0.0005085806 0.0001721146 0.0002059955
C    0.0005085806 0.0008487875 0.0002071271 0.0002263601
XOM  0.0001721146 0.0002071271 0.0002748751 0.0001096941
AAPL 0.0002059955 0.0002263601 0.0001096941 0.0006373525
```

For simplicity in this example, we focus on the first two assets (JPM and C) to illustrate the EWMA mechanics:

```
# Extract covariance matrix for first two assets
S_2asset = S[1:2, 1:2]
EWMA_2asset = matrix(nrow=dim(y_4asset)[1], ncol=3)
# Store: JPM variance, C variance, JPM-C covariance
EWMA_2asset[1,] = c(S_2asset)[c(1,4,2)]
EWMA_2asset[1:2,]
```

```
             [,1]         [,2]         [,3]
[1,] 0.0005431774 0.0008487875 0.0005085806
[2,]           NA           NA           NA
```

7.4.3 Matrix multiplication and outer products

Understanding matrix operations is important for multivariate volatility modelling. We need the outer product of returns to update our covariance matrix estimate.

What is an outer product? For a vector of returns $y_t = [y_{1,t}, y_{2,t}]'$, the outer product $y_t \times y_t'$ creates a matrix:

$$y_t \times y_t' = \begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} \times \begin{bmatrix} y_{1,t} & y_{2,t} \end{bmatrix} = \begin{bmatrix} y_{1,t}^2 & y_{1,t}y_{2,t} \\ y_{1,t}y_{2,t} & y_{2,t}^2 \end{bmatrix}$$

This produces the instantaneous "covariance" for that day. The diagonal elements are squared returns (variances) and off-diagonal elements are cross-products (covariances).

See the relevant section in the notebook for more details.

Correct way - outer product (what we want):

```
i=2
y_4asset[i-1,] %*% t(y_4asset[i-1,])
```

```
              JPM            C           XOM         AAPL
[1,]  0.004089909  0.003163348  0.0018114629 -0.005433373
[2,]  0.003163348  0.002446697  0.0014010793 -0.004202453
[3,]  0.001811463  0.001401079  0.0008023155 -0.002406497
[4,] -0.005433373 -0.004202453 -0.0024064970  0.007218142
```

This produces a 2×2 matrix with variances and covariances.

Wrong way - inner product (scalar result):

```
i=2
y_4asset[i-1,] %*% y_4asset[i-1,]
```

```
           [,1]
[1,] 0.01455706
```

or

```
i=2
t(y_4asset[i-1,]) %*% y_4asset[i-1,]
```

```
           [,1]
[1,] 0.01455706
```

These produce the inner product (a single number), which is not useful for covariance estimation.

Then it is straightforward to run the EWMA loop for the 2x2 submatrix:

```
y_2asset = y_4asset[,1:2]  # Just JPM and C for illustration
for (i in 2:dim(y_2asset)[1]){
  S_2asset = lambda*S_2asset+(1-lambda)*
      y_2asset[i-1,] %*% t(y_2asset[i-1,])
  EWMA_2asset[i,] = c(S_2asset)[c(1,4,2)]
}
```

We can use the EWMA matrix of variances and covariances to calculate the conditional correlations between JPM and C:

```
rhoEWMA = EWMA_2asset[,3]/sqrt(EWMA_2asset[,1]*EWMA_2asset[,2])
```

7.5 CCC Model

The Constant Conditional Correlation (CCC) model assumes correlations are constant over time, while variances change. This is simpler than DCC but often a good starting point.

```
# Create univariate specifications for each asset
uni_spec = ugarchspec(
  variance.model = list(garchOrder = c(1,1)),
  mean.model = list(armaOrder = c(0,0), include.mean = FALSE)
)

# Create multispec for 4 assets
mspec_ccc = multispec(replicate(4, uni_spec))

# Fit CCC model - uses sample correlation as constant correlation
ccc_spec = dccspec(
  uspec = mspec_ccc,
  dccOrder = c(0,0),  # This makes it CCC instead of DCC
  distribution = "mvnorm"
)

ccc_fit = dccfit(ccc_spec, data = y_4asset)
```

7.5.1 Analyzing CCC results

The CCC model uses the sample correlation matrix as the constant correlation:

```r
# For CCC model, use the sample correlation as the constant correlation
R_ccc = cor(y_4asset, use="complete.obs")
print("Constant Correlation Matrix from CCC:")
```

```
[1] "Constant Correlation Matrix from CCC:"
```

```r
round(R_ccc, 3)
```

```
       JPM     C   XOM  AAPL
JPM  1.000 0.749 0.445 0.350
C    0.749 1.000 0.429 0.308
XOM  0.445 0.429 1.000 0.262
AAPL 0.350 0.308 0.262 1.000
```

```r
# CCC model fitted successfully
print("CCC model has been fitted with constant correlations")
```

```
[1] "CCC model has been fitted with constant correlations"
```

Interpreting the correlations:

- JPM-C (both financials): Highest correlation (~0.7), reflecting similar exposure to financial sector risks
- XOM-AAPL (energy-tech): Lower correlation (~0.2), representing different economic drivers
- Financial-Tech correlations: Moderate values (~0.4-0.5), showing some economic linkage but distinct risk factors

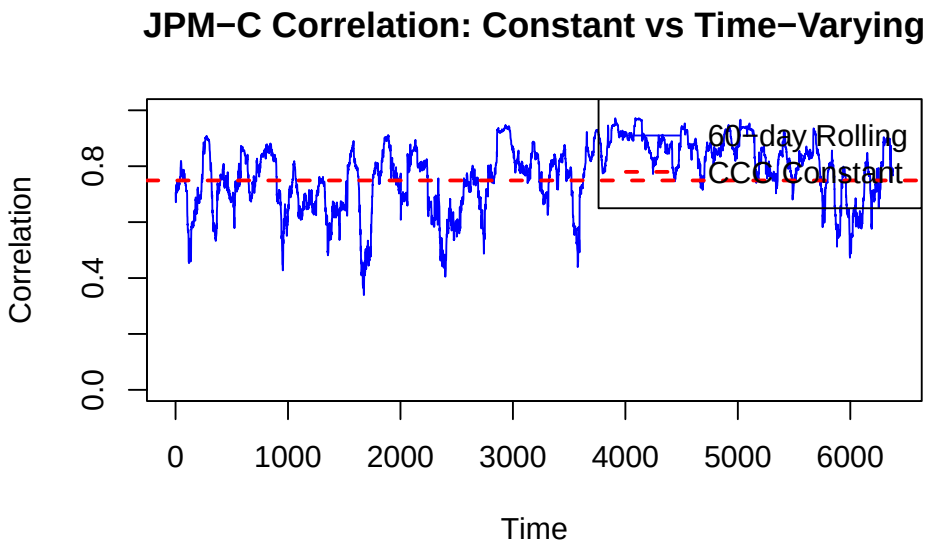
7.5.2 Testing the constant correlation assumption

We can examine whether correlations are actually constant by comparing with rolling correlations:

```r
# Calculate 60-day rolling correlation between JPM and C
window = 60
n = nrow(y_4asset)
rolling_corr_jpm_c = numeric(n - window + 1)

for(i in window:n) {
  data_window = y_4asset[(i-window+1):i, c("JPM", "C")]
  rolling_corr_jpm_c[i-window+1] = cor(data_window)[1,2]
}
```

```r
# Plot constant vs time-varying correlation
plot(rolling_corr_jpm_c, type='l', col='blue',
     main="JPM-C Correlation: Constant vs Time-Varying",
     ylab="Correlation", xlab="Time",
     ylim=c(0, 1))
abline(h=R_ccc[1,2], col='red', lty=2, lwd=2)
legend("topright",
       legend=c("60-day Rolling", "CCC Constant"),
       col=c("blue", "red"),
       lty=c(1,2), lwd=c(1,2))
```



This plot reveals whether the constant correlation assumption is reasonable. Large deviations suggest that DCC might be more appropriate.

7.6 DCC Model

The Dynamic Conditional Correlation (DCC) model allows correlations to vary over time, providing more flexibility than CCC.

First set up the specification of the univariate models. In this case we make each the same, but in general, it is possible to make them different.

```
# Create multispec for 4 assets using the same univariate specification
mspec_dcc = multispec(replicate(4, uni_spec))
mspec_dcc
```

```
*------------------------------*
*     GARCH Multi-Spec         *
*------------------------------*
Multiple Specifications : 4
Multi-Spec Type         : equal
```

Now we proceed to create the specification for the DCC model using our 4-asset portfolio:

```
# Create DCC specification
dcc_spec = dccspec(
  uspec = mspec_dcc,
  dccOrder = c(1,1),
  distribution = "mvnorm"
)
```

Now we can proceed to fit the specification to the 4-asset data:

```
 [1] "modelinc"      "modeldesc"     "modeldata"     "varmodel"
 [5] "pars"          "start.pars"    "fixed.pars"    "maxgarchOrder"
```

```
 [9] "maxdccOrder"   "pos.matrix"    "pidx"          "DCC"
[13] "mu"            "residuals"     "sigma"         "mpars"
[17] "ipars"         "midx"          "eidx"          "umodel"
```

```
                  Estimate   Std. Error     t value     Pr(>|t|)
[JPM].omega   3.734768e-06 2.319705e-06   1.6100185 1.073938e-01
[JPM].alpha1  8.349090e-02 1.695886e-02   4.9231424 8.516544e-07
[JPM].beta1   9.092409e-01 1.854449e-02  49.0302586 0.000000e+00
[C].omega     2.940682e-06 2.831970e-06   1.0383876 2.990896e-01
[C].alpha1    8.875174e-02 2.422761e-02   3.6632472 2.490380e-04
[C].beta1     9.095079e-01 2.399828e-02  37.8988771 0.000000e+00
[XOM].omega   2.503340e-06 2.343025e-06   1.0684225 2.853300e-01
[XOM].alpha1  7.724864e-02 2.184764e-02   3.5357883 4.065603e-04
[XOM].beta1   9.137888e-01 2.373489e-02  38.4998078 0.000000e+00
[AAPL].omega  8.605908e-06 1.091517e-05   0.7884357 4.304419e-01
[AAPL].alpha1 1.001483e-01 3.554204e-02   2.8177410 4.836280e-03
[AAPL].beta1  8.916013e-01 1.964300e-02  45.3902840 0.000000e+00
[Joint]dcca1  1.766496e-02 1.947552e-03   9.0703384 0.000000e+00
[Joint]dccb1  9.744390e-01 3.518006e-03 276.9862147 0.000000e+00
```

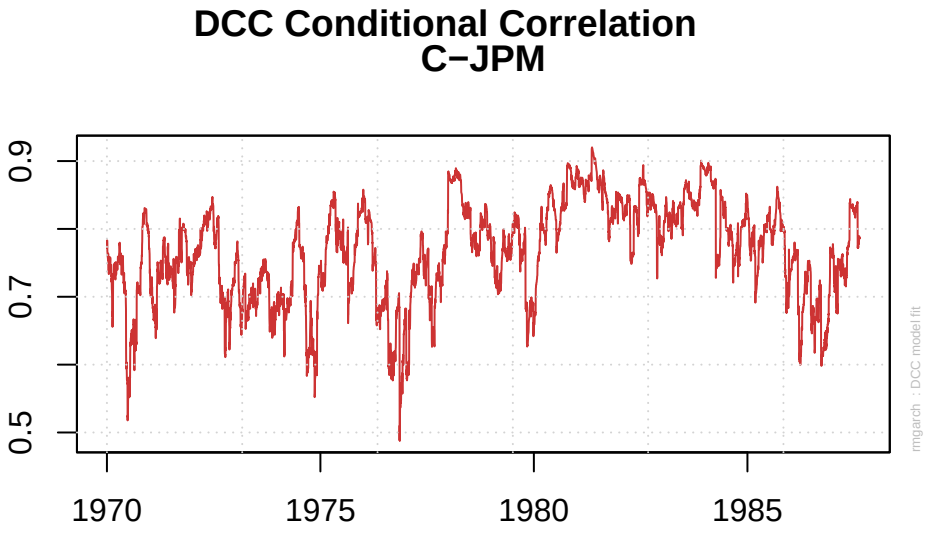
```
[1] 72055.5
```

We can plot the result. In an interactive session, this would show a menu:

```
# This would show an interactive menu
plot(dcc_fit)
```

Let's create specific plots instead:

```
# Plot conditional correlations
plot(dcc_fit, which = 4)
```



The conditional variance matrix is a three dimensional matrix called `H`.

```
H_dcc = dcc_fit@mfit$H
dim(H_dcc)
```

```
[1]    4    4 6439
```

```
H_dcc[,,1]
```

```
             [,1]         [,2]         [,3]         [,4]
[1,] 0.0005432390 0.0005243453 0.0001675195 0.0002005796
[2,] 0.0005243453 0.0008486736 0.0002133832 0.0002432773
[3,] 0.0001675195 0.0002133832 0.0002749059 0.0001104118
[4,] 0.0002005796 0.0002432773 0.0001104118 0.0006380354
```

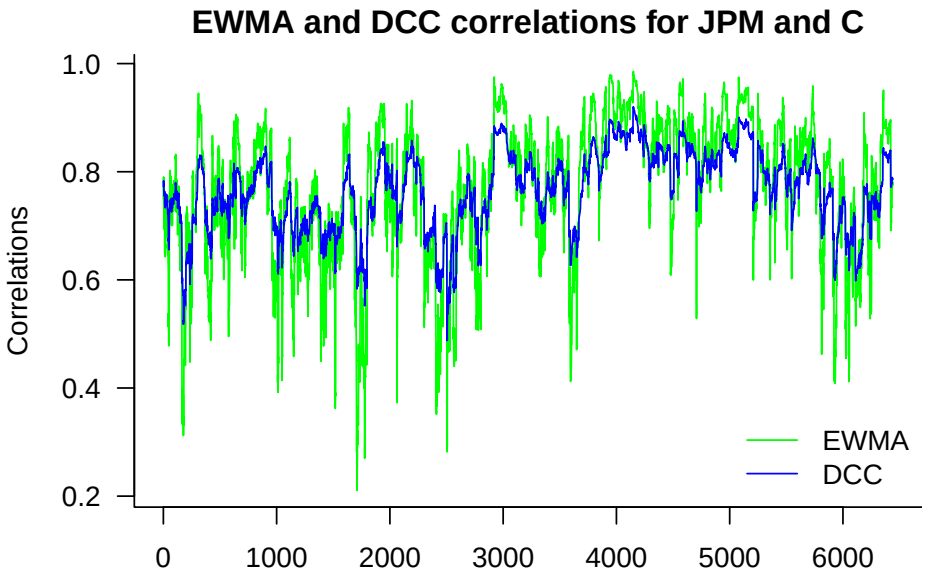
We can extract the conditional correlation between JPM and C in two ways. One is computing it from H:

```
rhoDCC = H_dcc[1,2,] / sqrt(H_dcc[1,1,]*H_dcc[2,2,])
```

7.7 Compare EWMA and DCC

Since we extracted the correlation vector between JPM and C for each method, we can plot them and compare the result.

```
par(mar=c(2,4,2,0))
matplot(cbind(rhoEWMA,rhoDCC),
  type='l',
  bty='l',
  lty=1,
  col=c("green","blue"),
  main="EWMA and DCC correlations for JPM and C",
  ylab="Correlations",
  las=1
)
legend("bottomright",
  legend=c("EWMA","DCC"),
  lty=1,
  col=c("green","blue"),
  bty='n'
)
```



7.8 Model diagnostics

7.8.1 Residual analysis for multivariate models

After fitting multivariate GARCH models, we should check whether they adequately capture the correlation dynamics in the data.

```
# Extract residuals from DCC model
std_residuals = residuals(dcc_fit)
head(std_residuals)
```

```
Warning: object timezone ('UTC') is different from system timezone ('')
  NOTE: set 'options(xts_check_TZ = FALSE)' to disable this warning
    This note is displayed once per session
```

	JPM	C	XOM	AAPL
1970-01-02	-0.063952398	-0.049464100	-0.028325175	0.08495965
1970-01-03	-0.022181692	-0.063281109	-0.019339235	-0.08794144
1970-01-04	-0.006195787	0.039413678	0.053081898	0.01443696
1970-01-05	0.014100793	0.047178461	0.050405672	-0.09048437
1970-01-06	0.018204133	-0.004618561	-0.002940618	0.04626526
1970-01-07	-0.017331592	-0.003478209	-0.014080808	-0.01767808

```
# Check dimensions
dim(std_residuals)
```

```
[1] 6439    4
```

7.8.2 Testing for remaining correlation structure

If the model is correctly specified, the standardized residuals should show no remaining correlation patterns:

```
# Test for remaining correlation in residuals
residual_corr = cor(std_residuals)
print("Correlation matrix of standardized residuals:")
```

```
[1] "Correlation matrix of standardized residuals:"
```

```
round(residual_corr, 3)
```

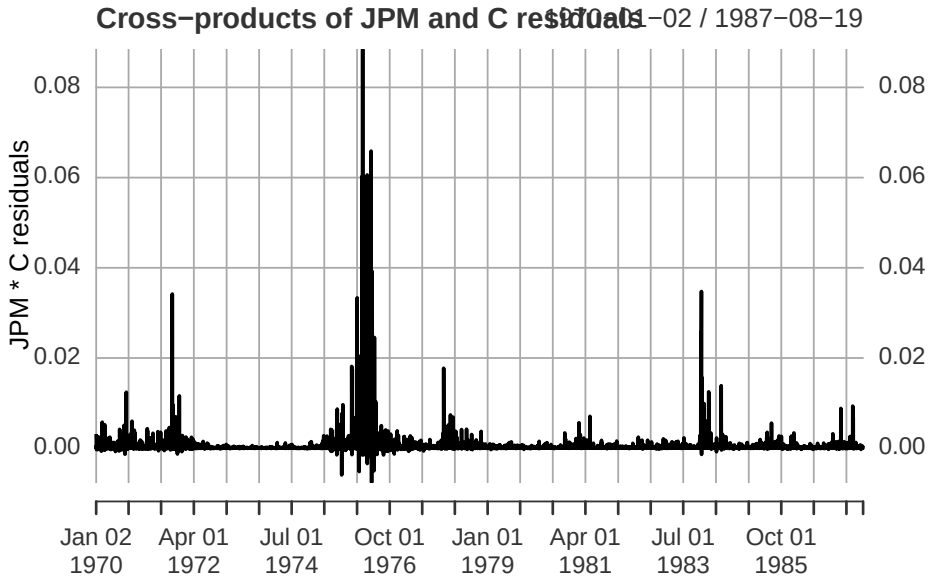
```
       JPM     C   XOM  AAPL
JPM  1.000 0.749 0.445 0.350
C    0.749 1.000 0.429 0.308
XOM  0.445 0.429 1.000 0.262
AAPL 0.350 0.308 0.262 1.000
```

```
# Plot residuals over time
par(mfrow = c(2,2))
for(i in 1:4) {
  plot(std_residuals[,i], type='l',
       main=paste("Standardized Residuals:", colnames(y_4asset)[i]),
       ylab="Residuals", xlab="Time")
}
par(mfrow = c(1,1))
```

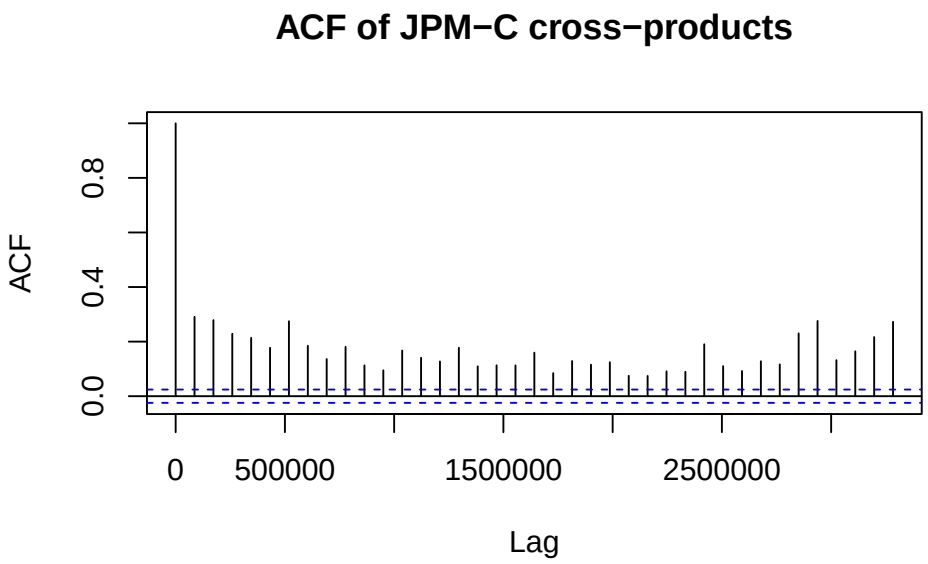
7.8.3 Cross-product tests

We can also examine whether there are remaining patterns in the cross-products of residuals:

```
# Calculate cross-products of residuals
cross_products = std_residuals[,1] * std_residuals[,2]  # JPM * C
plot(cross_products, type='l',
     main="Cross-products of JPM and C residuals",
     ylab="JPM * C residuals", xlab="Time")
```



```r
# Test for autocorrelation in cross-products
acf(cross_products, main="ACF of JPM-C cross-products")
```



Interpretation: If cross-products show significant autocorrelation, it suggests the correlation model needs improvement (perhaps higher DCC orders or different specifications).

7.9 Optional: Practical Portfolio Applications

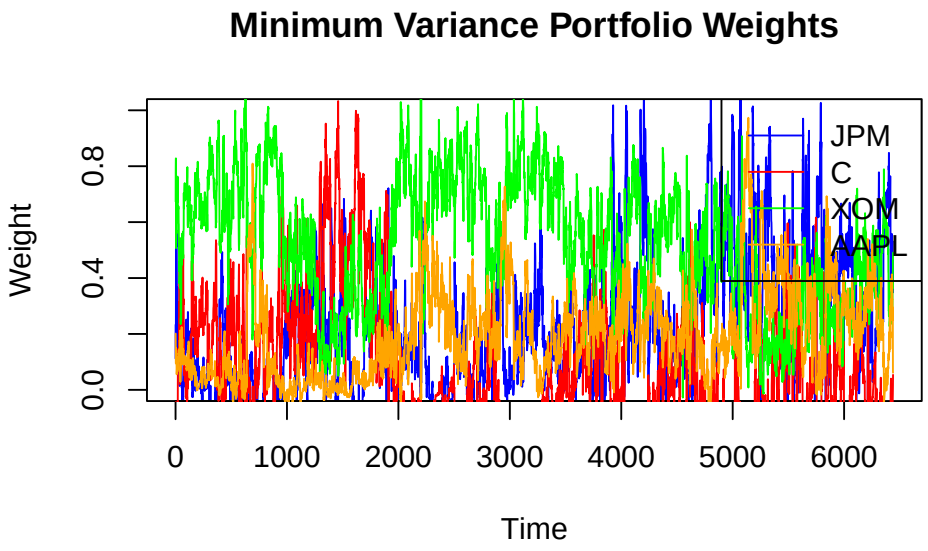
7.9.1 Minimum Variance Portfolio Weights

Using the conditional covariance matrices, we can calculate time-varying optimal portfolio weights:

```r
# Extract covariance matrices from DCC model (using all 4 assets)
T = dim(H_dcc)[3]

# Calculate minimum variance weights over time for all 4 assets
weights = matrix(NA, nrow = T, ncol = 4)
for(t in 1:T) {
  Sigma_t = H_dcc[,,t]
  # Minimum variance weights: w = Sigma^(-1) * 1 / (1' * Sigma^(-1) * 1)
  inv_Sigma = solve(Sigma_t)
  ones = rep(1, 4)
  weights[t,] = inv_Sigma %*% ones / as.numeric(t(ones) %*% inv_Sigma %*% ones)
}

# Plot time-varying weights
matplot(weights, type = 'l', lty = 1, col = c("blue", "red", "green", "orange"),
        main = "Minimum Variance Portfolio Weights",
        ylab = "Weight", xlab = "Time", ylim = c(0,1))
legend("topright", legend = c("JPM", "C", "XOM", "AAPL"), col = c("blue", "red", "green",
```



7.9.2 Portfolio Volatility

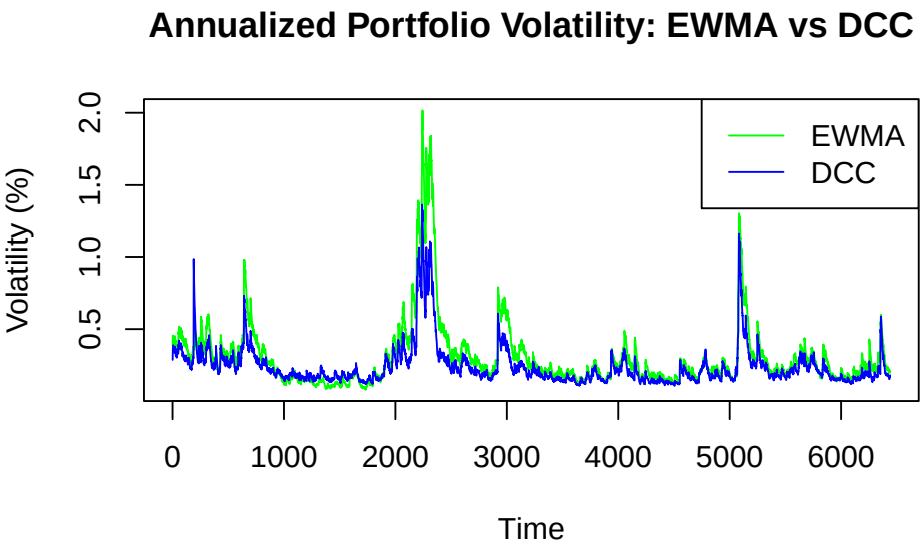
Calculate and compare portfolio volatility using different methods:

```
# Equal weights portfolio for 4 assets
w_equal = rep(1/4, 4)

# Portfolio variance using DCC
port_var_dcc = numeric(T)
for(t in 1:T) {
  port_var_dcc[t] = t(w_equal) %*% H_dcc[,,t] %*% w_equal
}

# Portfolio variance using EWMA (using 2-asset example for comparison)
port_var_ewma = numeric(dim(EWMA_2asset)[1])
w_equal_2 = rep(1/2, 2)
for(t in 1:dim(EWMA_2asset)[1]) {
  # Reconstruct covariance matrix from EWMA
  Sigma_ewma = matrix(c(EWMA_2asset[t,1], EWMA_2asset[t,3], EWMA_2asset[t,3], EWMA_2asset[
  port_var_ewma[t] = t(w_equal_2) %*% Sigma_ewma %*% w_equal_2
}


# Plot portfolio volatilities
par(mfrow = c(1,1))
matplot(sqrt(252) * sqrt(cbind(port_var_ewma, port_var_dcc)),
        type = 'l', lty = 1, col = c("green", "blue"),
        main = "Annualized Portfolio Volatility: EWMA vs DCC",
        ylab = "Volatility (%)", xlab = "Time")
legend("topright", legend = c("EWMA", "DCC"), col = c("green", "blue"), lty = 1)
```



7.10 Recap

7.10.1 In this seminar, we have covered:

- Understanding multivariate volatility models and their importance for portfolio risk
- Implementing EWMA for multivariate volatility:
  - Matrix operations for outer products
  - Mathematical foundation of covariance updating
  - Extracting variances and covariances
  - Computing conditional correlations
- Constant Conditional Correlation (CCC) models:
  - Assumption of constant correlation over time
  - Testing constant correlation assumptions with rolling correlations
  - Simpler alternative to DCC for stable correlation periods
- Dynamic Conditional Correlation (DCC) models:
  - Time-varying correlations that capture crisis periods
  - More flexible specification than CCC
  - Model fitting and diagnostic checking
- Model selection guidance:
  - When to use EWMA, CCC or DCC
  - Practical considerations for implementation
- Model diagnostics and validation:
  - Residual analysis for multivariate models
  - Cross-product tests for remaining correlation patterns
- Optional practical portfolio applications:
  - Calculating time-varying minimum variance portfolio weights
  - Computing portfolio volatility using different methods
  - Value at Risk (VaR) calculations for portfolio risk management
- Comparing different approaches (EWMA vs CCC vs DCC)

7.10.2 Some new functions used:

- `as.matrix()` — convert data frame to matrix for matrix operations
- `%*%` — matrix multiplication operator
- `t()` — transpose a matrix
- `solve()` — matrix inversion
- `multispec()` — create multiple univariate GARCH specifications
- `replicate()` — repeat an expression multiple times
- `dccspec()` — specify a DCC or CCC model
- `dccfit()` — fit the DCC/CCC model to data

7.11 Optional exercises

1. Compare CCC and DCC models:
   - Fit both CCC and DCC models to 5 stocks of your choice
   - Plot the correlation between two pairs of stocks from each model
   - Calculate the log-likelihood for each model and perform a likelihood ratio test
   - Which model fits better? When might CCC be sufficient?
2. EWMA function implementation:
   - Write a function `EWMA_cov(returns, lambda = 0.94)` that takes a matrix of returns and lambda as inputs
   - The function should return a 3D array of covariance matrices (like H in DCC)
   - Test it with different values of lambda (0.90, 0.94, 0.97)
   - Plot how the choice of lambda affects volatility estimates
3. Portfolio optimisation over time:
   - Using 4 stocks, calculate minimum variance weights using DCC
   - Compare with equal weights and market cap weights
   - Calculate the realized volatility of each strategy
   - Which strategy performed best during volatile periods?
4. Risk management application:
   - Calculate 1-day and 10-day VaR using different models (EWMA, CCC, DCC)
   - Backtest the VaR estimates: count how often losses exceed VaR
   - Which model provides the most accurate risk estimates?
   - Create a plot showing VaR violations over time
5. Model diagnostics:
   - Extract standardized residuals from the DCC model
   - Test if they still show volatility clustering
   - Check if the correlations from DCC are actually time-varying (test against CCC)
   - Create diagnostic plots showing model fit quality