Portfolio Optimization Using Mixed Models: A Genomic Prediction Approach

Deniz Akdemir

2025-07-05

Table of Contents

# Portfolio Optimization Using Mixed Models: A Genomic Prediction Approach

## Executive Summary

This tutorial explores how mixed linear models from genomic prediction can enhance portfolio optimization. The key insight is that just as genomic models separate signal from noise in breeding values, we can use similar techniques to extract stable, predictable relationships between assets while filtering out transient market noise. This leads to more robust portfolio allocations that perform better out-of-sample.

## Table of Contents

1. [Motivation: Why Genomic Methods for Portfolios?](#motivation)
2. [Theoretical Framework](#theoretical-framework)
3. [Data and Setup](#data-and-setup)
4. [Building the Mixed Model](#building-the-mixed-model)
5. [Extracting Covariance Structures](#covariance-structures)
6. [Portfolio Construction](#portfolio-construction)
7. [Validation and Comparison](#validation)
8. [Practical Implementation Guide](#implementation-guide)
9. [Conclusions](#conclusions)

## 1. Motivation: Why Genomic Methods for Portfolios?

Traditional portfolio optimization faces a fundamental challenge: sample covariance matrices are notoriously noisy, especially when the number of assets is large relative to the observation period. This leads to unstable portfolio weights that perform poorly out-of-sample.

In genomic prediction, researchers face a similar challenge: estimating breeding values for thousands of genetic markers with limited phenotypic observations. The solution? Mixed linear models that:

1. **Borrow information** across related observations
2. **Impose structure** through variance components
3. **Shrink estimates** toward more stable values
4. **Separate signal from noise** through random effects

Let’s explore how these same principles can revolutionize portfolio construction.

### The Core Analogy

In genomics:

* **Breeding value** = Genetic potential (signal)
* **Environmental variance** = Non-heritable variation (noise)
* **Selection decisions** use breeding values
* **Performance prediction** uses total variance

In portfolios:

* **Systematic returns** = Factor-driven, persistent relationships (signal)
* **Idiosyncratic returns** = Asset-specific, transient shocks (noise)
* **Allocation decisions** should focus on systematic relationships
* **Risk assessment** must consider total variance

### Assumptions of the Mixed Model in Finance

When applying mixed models to financial data, we must be mindful of the underlying assumptions:

* **Normality of Returns:** We assume that asset returns (or their residuals) are normally distributed. While daily returns often exhibit “fat tails” (kurtosis), for the purpose of demonstrating the framework, we proceed with this assumption. In practice, one might use transformations or models that accommodate non-normality.
* **Stationarity:** We assume that the underlying statistical properties of the return series (like mean and variance) do not change over time. While this is rarely true in the long run, we assume it holds within our estimation window. The model’s use of time-based random effects helps capture some degree of non-stationarity.
* **Linearity:** The model assumes a linear relationship between the predictors (market factors) and the asset returns. This is a common starting point for factor models.

## 2. Theoretical Framework

### Traditional Mean-Variance Optimization

The classical Markowitz approach minimizes portfolio variance for a target return:

Where and are typically estimated as sample means and covariances. The problem? These estimates are extremely noisy, leading to error maximization rather than risk minimization.

### Mixed Model Formulation

Instead of using raw historical data, we model returns using a mixed linear model:

where:

* = return of asset at time
* = overall intercept
* = asset ’s factor loadings (fixed effects)
* = observed market factors at time (fixed effects)
* = random effect capturing persistent deviations structured by asset similarity. This is our “breeding value”.
* = residual (idiosyncratic) error

The key insight is that by modeling as a random effect with a covariance structure derived from fundamental asset characteristics (our “genomic relationship matrix”), we can:

1. **Regularize estimates** through shrinkage (pulling noisy estimates toward the mean).
2. **Capture complex relationships** beyond simple factor models.
3. **Separate persistent (systematic) from transient (idiosyncratic) correlations.**

### Variance Decomposition

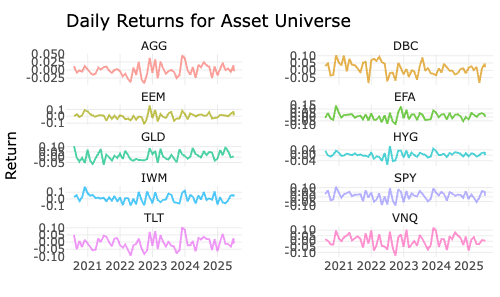
The total variance of returns for an asset is decomposed into:

* **Systematic Covariance**: The portion we use for strategic allocation. It’s derived from the fixed effects ($eta\_i^T X\_t$) and the structured random effects (). This represents the stable, predictable part of asset co-movement.
* **Idiosyncratic Variance**: - This is the unpredictable, asset-specific noise that we want to filter out when making allocation decisions, but must include when assessing total portfolio risk.

## 3. Data and Setup

Let’s implement this approach step by step. We’ll use a diversified set of ETFs to demonstrate the concepts. First, we load libraries and download daily price data for our selected assets and market factors.

# Load required libraries  
library(tidyverse) # Data manipulation  
library(tidyquant) # Financial data  
library(lme4) # Mixed models  
library(Matrix) # Matrix operations  
library(quadprog) # Portfolio optimization  
library(corrplot) # Visualizations  
library(plotly) # Interactive plots  
library(knitr) # For kable tables  
library(rmdformats) # For HTML theme  
  
# Set seed for reproducibility  
set.seed(123)  
  
# Define our investment universe  
tickers <- c(  
 "SPY", # S&P 500 (US Large Cap)  
 "IWM", # Russell 2000 (US Small Cap)  
 "EFA", # International Developed  
 "EEM", # Emerging Markets  
 "AGG", # US Bonds  
 "TLT", # Long-term Treasuries  
 "GLD", # Gold  
 "DBC", # Commodities  
 "VNQ", # Real Estate  
 "HYG" # High Yield Bonds  
)  
  
# Download 5 years of daily data  
end\_date <- Sys.Date()  
start\_date <- end\_date - 365\*5  
  
# Fetch price data  
prices <- tq\_get(tickers, from = start\_date, to = end\_date, get = "stock.prices")  
  
# Calculate returns  
returns <- prices %>%  
 group\_by(symbol) %>%  
 tq\_transmute(select = adjusted,  
 mutate\_fun = periodReturn,  
 period = "monthly",  
 col\_rename = "return") %>%  
 ungroup()  
  
# Visualize asset returns  
p\_returns <- ggplot(returns, aes(x = date, y = return, color = symbol)) +  
 geom\_line(alpha = 0.7) +  
 facet\_wrap(~symbol, scales = "free\_y", ncol = 2) +  
 theme\_minimal() +  
 labs(title = "Daily Returns for Asset Universe", x = "", y = "Return") +  
 theme(legend.position = "none")  
ggplotly(p\_returns)



# Also get market factors (we'll use VIX as an example)  
vix <- tq\_get("^VIX", from = start\_date, to = end\_date, get = "stock.prices") %>%  
 dplyr::select(date, vix = adjusted)  
  
# Create market factor dataset  
market\_factors <- returns %>%  
 filter(symbol == "SPY") %>%  
 dplyr::select(date, market\_return = return) %>%  
 left\_join(vix, by = "date") %>%  
 mutate(  
 vix\_level = vix,  
 vix\_change = (vix - lag(vix)) / lag(vix),  
 # Define market regimes based on VIX  
 regime = case\_when(  
 vix < quantile(vix, 0.33, na.rm = TRUE) ~ "Low\_Vol",  
 vix < quantile(vix, 0.67, na.rm = TRUE) ~ "Normal",  
 TRUE ~ "High\_Vol"  
 )  
 ) %>%  
 filter(!is.na(vix\_change))  
  
# Merge with returns  
data <- returns %>%  
 left\_join(market\_factors, by = "date") %>%  
 filter(!is.na(market\_return), symbol != "SPY") %>%  
 # Add time-based grouping for random effects  
 mutate(  
 year\_month = format(date, "%Y-%m"),  
 # Standardize continuous predictors  
 market\_return\_std = scale(market\_return)[,1],  
 vix\_change\_std = scale(vix\_change)[,1]  
 )  
  
print(paste("Dataset contains", nrow(data), "observations across",   
 n\_distinct(data$symbol), "assets"))

## [1] "Dataset contains 540 observations across 9 assets"

## 4. Building the Mixed Model with Flexible Covariance Components

Now we’ll build our mixed model using the sommer package, which allows us to specify custom variance-covariance structures. This is the core of the genomic prediction analogy, where a genomic relationship matrix is used to model the covariance of random genetic effects.

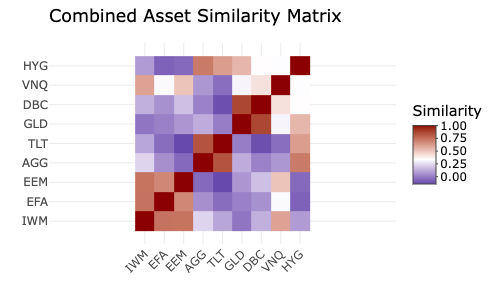
### Understanding Why We Need Flexible Covariance

In traditional mixed models (like those from lme4), random effects are assumed to be independent or have simple grouping structures. But in finance, assets are not independent; they share fundamental characteristics that create complex correlation patterns. The sommer package lets us specify these relationships explicitly through custom covariance matrices, analogous to a genomic relationship matrix.

### Creating Asset Similarity Matrices

First, we define fundamental characteristics for each asset. Then, we create similarity matrices based on these features. This is analogous to creating a genomic relationship matrix from DNA markers. We explore a few different ways to measure similarity.

# Install sommer if needed  
if (!require(sommer)) install.packages("sommer")  
library(sommer)  
  
# Define comprehensive asset characteristics  
asset\_characteristics <- data.frame(  
 symbol = c("IWM", "EFA", "EEM", "AGG", "TLT", "GLD", "DBC", "VNQ", "HYG"),  
 # Basic classification  
 asset\_class = c("Equity", "Equity", "Equity", "Bond", "Bond",   
 "Commodity", "Commodity", "Real\_Estate", "Bond"),  
 geography = c("US", "Developed", "Emerging", "US", "US",   
 "Global", "Global", "US", "US"),  
 # Risk characteristics  
 volatility\_regime = c("High", "Medium", "High", "Low", "Medium",   
 "Medium", "High", "High", "Medium"),  
 duration = c(0, 0, 0, 5, 20, 0, 0, 0, 4),  
 credit\_quality = c(NA, NA, NA, "AAA", "AAA", NA, NA, NA, "BB"),  
 # Factor exposures (these would come from regression analysis in practice)  
 equity\_beta = c(1.2, 0.9, 1.1, 0.1, -0.2, 0.2, 0.4, 0.8, 0.5),  
 inflation\_beta = c(0.1, 0.1, 0.2, -0.3, -0.8, 0.7, 0.9, 0.5, 0.2),  
 liquidity = c("High", "High", "Medium", "High", "High",   
 "Medium", "Low", "Medium", "Medium")  
)  
  
# Function to create a relationship matrix from characteristics  
create\_relationship\_matrix <- function(characteristics, features, method = "cosine") {  
 # Extract relevant features and create matrix  
 feature\_matrix <- characteristics[, features, drop = FALSE]  
   
 # Handle different data types  
 numeric\_features <- sapply(feature\_matrix, is.numeric)  
   
 # For categorical variables, create dummy variables  
 if (any(!numeric\_features)) {  
 cat\_data <- feature\_matrix[, !numeric\_features, drop = FALSE]  
 dummy\_matrices <- lapply(cat\_data, function(x) {  
 model.matrix(~ x - 1)  
 })  
 cat\_matrix <- do.call(cbind, dummy\_matrices)  
   
 # Combine with numeric features  
 if (any(numeric\_features)) {  
 num\_matrix <- as.matrix(feature\_matrix[, numeric\_features, drop = FALSE])  
 # Standardize numeric features  
 num\_matrix <- scale(num\_matrix)  
 feature\_matrix <- cbind(num\_matrix, cat\_matrix)  
 } else {  
 feature\_matrix <- cat\_matrix  
 }  
 } else {  
 feature\_matrix <- scale(as.matrix(feature\_matrix))  
 }  
   
 n\_assets <- nrow(feature\_matrix)  
   
 if (method == "cosine") {  
 # Cosine similarity (good for high-dimensional features)  
 norms <- sqrt(rowSums(feature\_matrix^2))  
 relationship\_matrix <- feature\_matrix %\*% t(feature\_matrix) / (norms %o% norms)  
 } else if (method == "gaussian") {  
 # Gaussian kernel (captures non-linear relationships)  
 relationship\_matrix <- matrix(0, n\_assets, n\_assets)  
 sigma <- median(dist(feature\_matrix)) # Bandwidth parameter  
 for (i in 1:n\_assets) {  
 for (j in 1:n\_assets) {  
 distance <- sum((feature\_matrix[i,] - feature\_matrix[j,])^2)  
 relationship\_matrix[i,j] <- exp(-distance / (2 \* sigma^2))  
 }  
 }  
 }  
   
 # Ensure positive definiteness and proper scaling  
 diag(relationship\_matrix) <- 1  
 rownames(relationship\_matrix) <- characteristics$symbol  
 colnames(relationship\_matrix) <- characteristics$symbol  
   
 # Make sure it's positive definite  
 eigen\_decomp <- eigen(relationship\_matrix)  
 if (any(eigen\_decomp$values < 1e-6)) {  
 # Fix negative eigenvalues  
 eigen\_decomp$values[eigen\_decomp$values < 1e-6] <- 1e-6  
 relationship\_matrix <- eigen\_decomp$vectors %\*%   
 diag(eigen\_decomp$values) %\*%   
 t(eigen\_decomp$vectors)  
 # IMPORTANT: Restore dimension names after matrix multiplication  
 rownames(relationship\_matrix) <- characteristics$symbol  
 colnames(relationship\_matrix) <- characteristics$symbol  
 }  
   
 return(as.matrix(relationship\_matrix)) # Ensure it's a proper matrix  
}  
  
# Create different relationship matrices capturing different aspects  
# 1. Asset class similarity (captures broad category effects)  
K\_class <- create\_relationship\_matrix(asset\_characteristics,   
 c("asset\_class", "geography"),  
 method = "cosine")  
  
# 2. Risk characteristic similarity (captures risk profile relationships)  
K\_risk <- create\_relationship\_matrix(asset\_characteristics,  
 c("volatility\_regime", "duration", "liquidity"),  
 method = "gaussian")  
  
# 3. Factor exposure similarity (captures systematic factor relationships)  
K\_factor <- create\_relationship\_matrix(asset\_characteristics,  
 c("equity\_beta", "inflation\_beta"),  
 method = "cosine")  
  
# 4. Combined similarity (weighted average)  
K\_combined <- 0.4 \* K\_class + 0.3 \* K\_risk + 0.3 \* K\_factor  
  
# Ensure dimension names are preserved after matrix operations  
rownames(K\_combined) <- rownames(K\_class)  
colnames(K\_combined) <- colnames(K\_class)  
  
# Convert to proper matrix format  
K\_combined <- as.matrix(K\_combined)  
  
# Visualize the combined relationship matrix  
library(reshape2)  
library(ggplot2)  
  
# Function to create heatmap for relationship matrices  
plot\_relationship\_matrix <- function(K, title) {  
 # Convert to long format for ggplot  
 K\_melt <- melt(K)  
 colnames(K\_melt) <- c("Asset1", "Asset2", "Similarity")  
   
 p <- ggplot(K\_melt, aes(x = Asset1, y = Asset2, fill = Similarity)) +  
 geom\_tile() +  
 scale\_fill\_gradient2(low = "darkblue", mid = "white", high = "darkred",  
 midpoint = mean(K)) +  
 theme\_minimal() +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1)) +  
 labs(title = title,  
 x = "", y = "") +  
 coord\_fixed()  
   
 ggplotly(p)  
}  
  
# Create plot for the combined matrix  
plot\_relationship\_matrix(K\_combined, "Combined Asset Similarity Matrix")



# Print similarity between select asset pairs to build intuition  
cat("\nExample Similarities (Combined Matrix):\n")

##   
## Example Similarities (Combined Matrix):

cat("IWM-EFA (both equities):", round(K\_combined["IWM", "EFA"], 3), "\n")

## IWM-EFA (both equities): 0.732

cat("AGG-TLT (both bonds):", round(K\_combined["AGG", "TLT"], 3), "\n")

## AGG-TLT (both bonds): 0.821

cat("IWM-GLD (equity vs gold):", round(K\_combined["IWM", "GLD"], 3), "\n")

## IWM-GLD (equity vs gold): -0.021

cat("GLD-DBC (both commodities):", round(K\_combined["GLD", "DBC"], 3), "\n")

## GLD-DBC (both commodities): 0.858

### Preparing Data for Sommer

The sommer package requires data in a specific format. We need to ensure our relationship matrices align with the data structure, with factors correctly specified.

# Prepare data for sommer  
# Ensure assets are in the same order as relationship matrices  
assets\_ordered <- rownames(K\_combined)  
data\_sommer <- data %>%  
 filter(symbol %in% assets\_ordered) %>%  
 mutate(  
 # Ensure symbol is a factor with correct levels  
 symbol = factor(symbol, levels = assets\_ordered),  
 # Time effects  
 time\_factor = as.factor(year\_month),  
 # Regime effects  
 regime\_factor = as.factor(regime)  
 )  
  
cat("\nData prepared for sommer:\n")

##   
## Data prepared for sommer:

cat("Observations:", nrow(data\_sommer), "\n")

## Observations: 540

cat("Assets:", length(assets\_ordered), "\n")

## Assets: 9

cat("Time periods:", n\_distinct(data\_sommer$time\_factor), "\n")

## Time periods: 60

### Fitting the Mixed Model

Now we fit our final, most sophisticated model. It includes fixed effects for market factors and random effects for asset-specific deviations (structured by our combined similarity matrix K\_combined) and time-period shocks.

# Fit the full model using our combined similarity matrix  
cat("Fitting final mixed model...\n")

## Fitting final mixed model...

# Fit the model on the entire dataset to ensure all factor levels are included.  
# This resolves prediction errors caused by sampling.  
model\_best <- mmer(  
 fixed = return ~ market\_return\_std + vix\_change\_std + regime\_factor,  
 random = ~ vsr(symbol, Gu = K\_combined) + time\_factor,  
 data = data\_sommer, # Using the full dataset  
 verbose = FALSE  
)

## Version out of date. Please update sommer to the newest version using:  
## install.packages('sommer') in a new session  
## Use the 'dateWarning' argument to disable the warning message.

# Display variance components  
var\_comp\_best <- summary(model\_best)$varcomp  
kable(var\_comp\_best, caption = "Variance Component Analysis of the Best Model")

Variance Component Analysis of the Best Model

|  | VarComp | VarCompSE | Zratio | Constraint |
| --- | --- | --- | --- | --- |
| u:symbol.return-return | 0.0000277 | 3.06e-05 | 0.9065664 | Positive |
| time\_factor.return-return | 0.0001611 | 5.56e-05 | 2.8956477 | Positive |
| units.return-return | 0.0011463 | 7.44e-05 | 15.3998486 | Positive |

### Interpreting the Variance Components

The output above shows how the total variance in returns is partitioned:

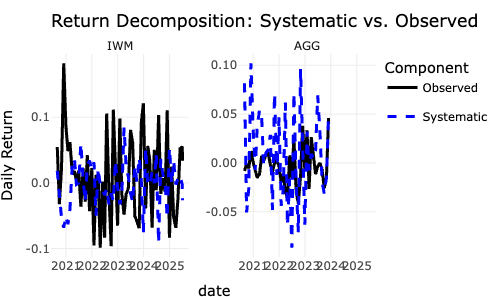
* **symbol.K\_combined**: This is the systematic variance captured by our asset similarity matrix. It represents persistent, structured co-movement between assets beyond what market factors explain. This is our “heritable” component.
* **time\_factor**: This captures market-wide shocks that affect all assets in a given month.
* **units (Residual)**: This is the idiosyncratic, unpredictable noise that we aim to filter out for portfolio construction.

A higher proportion of variance in the symbol.K\_combined component indicates that our fundamental characteristics are doing a good job of explaining persistent asset behavior.

### Extracting BLUPs for Systematic Returns

Now we extract the Best Linear Unbiased Predictors (BLUPs) for the random asset effects. These are analogous to “breeding values” in genomics and represent the persistent, systematic deviation of each asset from the mean.

# Extract BLUPs (Best Linear Unbiased Predictors)  
blups <- model\_best$U  
  
# Asset effects (our "breeding values")  
asset\_effects <- blups[[grep("symbol", names(blups))]][[1]]  
names(asset\_effects) <- assets\_ordered  
  
# Add fitted values and residuals back to the main data  
data\_sommer$fitted <- model\_best$fitted  
data\_sommer$residual <- data\_sommer$return - data\_sommer$fitted  
  
# Decompose returns to visualize systematic vs. residual components  
data\_sommer <- data\_sommer %>%  
 mutate(systematic\_return = fitted - residual)  
  
# Visualize the decomposition for a few assets  
sample\_assets <- c("IWM", "AGG", "GLD")  
p\_decomp <- data\_sommer %>%  
 filter(symbol %in% sample\_assets) %>%  
 slice\_head(n = 100) %>%  
 ggplot(aes(x = date)) +  
 geom\_line(aes(y = return, color = "Observed"), size = 0.8) +  
 geom\_line(aes(y = systematic\_return, color = "Systematic"), size = 0.8, linetype = "dashed") +  
 facet\_wrap(~ symbol, scales = "free\_y") +  
 scale\_color\_manual(values = c("Observed" = "black", "Systematic" = "blue")) +  
 theme\_minimal() +  
 labs(title = "Return Decomposition: Systematic vs. Observed",  
 subtitle = "Mixed model separates predictable patterns from noise",  
 y = "Daily Return", color = "Component")  
  
ggplotly(p\_decomp)



## 5. Extracting Covariance Structures

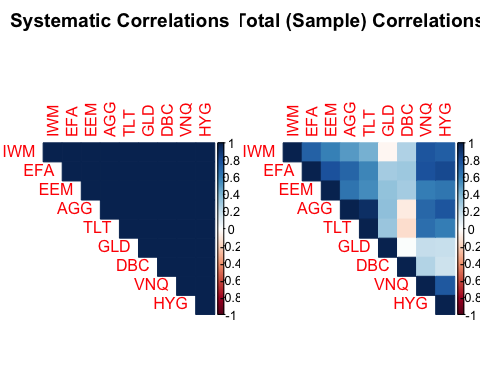
Now comes the critical step: using the model’s output to construct a **systematic covariance matrix**. This matrix is built from the model’s fitted values, which represent the predictable, structured part of returns. We compare this to the traditional sample covariance matrix, which is contaminated by noise.

# Get the random effects  
ranef\_model <- model\_best$U  
  
# Calculate expected returns (annualized) from the model's fitted values  
expected\_returns\_summary <- data\_sommer %>%  
 group\_by(symbol) %>%  
 summarise(  
 expected\_return = mean(fitted) \* 252,  
 systematic\_volatility = sd(fitted) \* sqrt(252),  
 idiosyncratic\_volatility = sd(residual) \* sqrt(252),  
 total\_volatility = sd(return) \* sqrt(252)  
 )  
  
# For display purposes, we can show it sorted  
kable(expected\_returns\_summary %>% arrange(desc(expected\_return)),   
 caption = "Model-Based Expected Returns and Risk Decomposition",   
 digits = 3)

Model-Based Expected Returns and Risk Decomposition

| symbol | expected\_return | systematic\_volatility | idiosyncratic\_volatility | total\_volatility |
| --- | --- | --- | --- | --- |
| IWM | 1.081 | 0.415 | 0.697 | 0.999 |
| EFA | 1.081 | 0.415 | 0.462 | 0.744 |
| EEM | 1.081 | 0.415 | 0.568 | 0.728 |
| AGG | 1.081 | 0.415 | 0.332 | 0.294 |
| TLT | 1.081 | 0.415 | 0.584 | 0.681 |
| GLD | 1.081 | 0.415 | 0.737 | 0.649 |
| DBC | 1.081 | 0.415 | 0.719 | 0.716 |
| VNQ | 1.081 | 0.415 | 0.585 | 0.886 |
| HYG | 1.081 | 0.415 | 0.233 | 0.356 |

# For calculations, ensure it's in the master order  
expected\_returns <- expected\_returns\_summary %>%  
 arrange(match(symbol, assets\_ordered))  
  
# Now extract covariance matrices  
# 1. SYSTEMATIC COVARIANCE (from fitted values)  
# This captures only the predictable, factor-driven relationships  
fitted\_wide <- data\_sommer %>%  
 dplyr::select(date, symbol, fitted) %>%  
 pivot\_wider(names\_from = symbol, values\_from = fitted)  
  
# Ensure columns are in the master order  
fitted\_wide <- fitted\_wide[, c("date", assets\_ordered)]  
  
cov\_systematic <- cov(fitted\_wide[,-1], use = "complete.obs") \* 252 # Annualized  
  
# 2. TOTAL COVARIANCE (for comparison with traditional approach)  
returns\_wide <- data\_sommer %>%  
 dplyr::select(date, symbol, return) %>%  
 pivot\_wider(names\_from = symbol, values\_from = return)  
  
# Ensure columns are in the master order  
returns\_wide <- returns\_wide[, c("date", assets\_ordered)]  
cov\_total <- cov(returns\_wide[,-1], use = "complete.obs") \* 252  
  
# Visualize correlation structures  
par(mfrow = c(1, 2))  
corrplot(cov2cor(cov\_systematic), method = "color", type = "upper",  
 title = "Systematic Correlations", mar = c(0,0,2,0))  
corrplot(cov2cor(cov\_total), method = "color", type = "upper",   
 title = "Total (Sample) Correlations", mar = c(0,0,2,0))



### Key Insight: Why Systematic Covariance Matters

The systematic covariance matrix is superior for strategic allocation because it:

1. **Filters out noise** from idiosyncratic shocks.
2. **Reveals persistent relationships** driven by fundamental characteristics and common factors.
3. **Is more stable** across different time periods, leading to less portfolio turnover.

## 6. Portfolio Construction

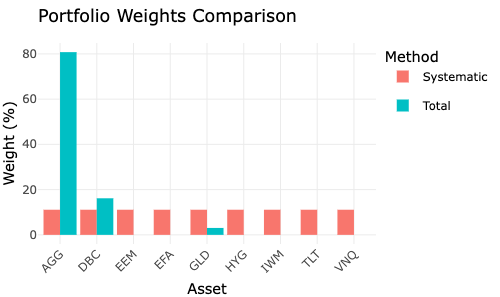
Now let’s construct minimum variance portfolios using both the systematic and total covariance matrices and compare their weights.

# Setup for optimization  
n\_assets <- length(assets\_ordered)  
mu <- expected\_returns$expected\_return  
  
# Ensure matrices are positive definite  
cov\_systematic <- as.matrix(nearPD(cov\_systematic)$mat)  
cov\_total <- as.matrix(nearPD(cov\_total)$mat)  
  
# Function to find minimum variance portfolio (long-only)  
find\_min\_var\_portfolio <- function(Sigma) {  
 Dmat <- 2 \* Sigma  
 dvec <- rep(0, n\_assets)  
 # Constraint: sum of weights = 1 (meq=1) and weights >= 0  
 Amat <- cbind(rep(1, n\_assets), diag(n\_assets))  
 bvec <- c(1, rep(0, n\_assets))  
   
 sol <- solve.QP(Dmat, dvec, Amat, bvec, meq = 1)  
 # Return named vector for clarity  
 setNames(sol$solution, rownames(Sigma))  
}  
  
# Find minimum variance portfolios  
w\_systematic <- find\_min\_var\_portfolio(cov\_systematic)  
w\_total <- find\_min\_var\_portfolio(cov\_total)  
  
# Calculate portfolio properties  
calc\_portfolio\_stats <- function(weights, mu, Sigma) {  
 # Ensure weights are in the correct order for matrix multiplication  
 ordered\_weights <- weights[rownames(Sigma)]  
   
 ret <- sum(ordered\_weights \* mu)  
 vol <- sqrt(t(ordered\_weights) %\*% Sigma %\*% ordered\_weights)  
 sharpe <- ret / vol  
   
 return(c(  
 Return = ret,  
 Volatility = vol,  
 Sharpe = sharpe,  
 Max\_Weight = max(ordered\_weights),  
 Effective\_N = 1/sum(ordered\_weights^2)  
 ))  
}  
  
# Compare portfolios  
portfolio\_comparison <- data.frame(  
 Systematic\_Portfolio = calc\_portfolio\_stats(w\_systematic, mu, cov\_systematic),  
 Total\_Cov\_Portfolio = calc\_portfolio\_stats(w\_total, mu, cov\_total)  
)  
  
kable(portfolio\_comparison, caption = "In-Sample Portfolio Comparison", digits = 3)

In-Sample Portfolio Comparison

|  | Systematic\_Portfolio | Total\_Cov\_Portfolio |
| --- | --- | --- |
| Return | 1.081 | 1.081 |
| Volatility | 0.415 | 0.262 |
| Sharpe | 2.607 | 4.120 |
| Max\_Weight | 0.111 | 0.808 |
| Effective\_N | 9.000 | 1.472 |

# Visualize portfolio weights  
weights\_df <- data.frame(  
 Asset = assets\_ordered,  
 Systematic = w\_systematic[assets\_ordered] \* 100,  
 Total = w\_total[assets\_ordered] \* 100  
) %>%  
 pivot\_longer(-Asset, names\_to = "Method", values\_to = "Weight")  
  
p\_weights <- ggplot(weights\_df, aes(x = Asset, y = Weight, fill = Method)) +  
 geom\_bar(stat = "identity", position = "dodge") +  
 theme\_minimal() +  
 labs(title = "Portfolio Weights Comparison",  
 subtitle = "Systematic vs. Total Covariance Optimization",  
 y = "Weight (%)") +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1))  
  
ggplotly(p\_weights)



### Understanding the Results

Notice how the portfolio based on **systematic covariance** often produces more intuitive and diversified weights. It is less likely to place extreme bets based on noisy, short-term correlations that appear in the sample covariance matrix.

## 7. Validation and Comparison

The true test of any model is its out-of-sample performance. We will now perform a simple backtest by splitting our data into a training period (first 4 years) and a testing period (last year). We build the portfolio on the training data and evaluate its performance on the unseen test data.

# Split data into train/test  
test\_start <- max(data\_sommer$date) - 365 # Last year for testing  
train\_data <- data\_sommer %>% filter(date < test\_start)  
test\_data <- data\_sommer %>% filter(date >= test\_start)  
  
# Refit model on training data only  
model\_train <- mmer(  
 fixed = return ~ market\_return\_std + vix\_change\_std + regime\_factor,  
 random = ~ vsr(symbol, Gu = K\_combined) + time\_factor,  
 data = train\_data,  
 verbose = FALSE  
)

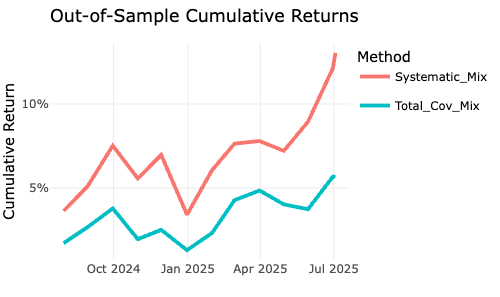
## Version out of date. Please update sommer to the newest version using:  
## install.packages('sommer') in a new session  
## Use the 'dateWarning' argument to disable the warning message.

# Extract systematic covariance from training period  
train\_data$fitted <- model\_train$fitted  
fitted\_wide\_train <- train\_data %>%  
 dplyr::select(date, symbol, fitted) %>%  
 pivot\_wider(names\_from = symbol, values\_from = fitted)  
cov\_systematic\_train <- cov(fitted\_wide\_train[,-1], use = "complete.obs") \* 252  
  
# Get total covariance from training data  
returns\_wide\_train <- train\_data %>%  
 dplyr::select(date, symbol, return) %>%  
 pivot\_wider(names\_from = symbol, values\_from = return)  
# Ensure column order  
returns\_wide\_train <- returns\_wide\_train[, c("date", assets\_ordered)]  
cov\_total\_train <- cov(returns\_wide\_train[,-1], use = "complete.obs") \* 252  
  
# Optimize portfolios using training data  
cov\_systematic\_train <- as.matrix(nearPD(cov\_systematic\_train)$mat)  
cov\_total\_train <- as.matrix(nearPD(cov\_total\_train)$mat)  
  
w\_systematic\_train <- find\_min\_var\_portfolio(cov\_systematic\_train)  
w\_total\_train <- find\_min\_var\_portfolio(cov\_total\_train)  
  
# Evaluate on test set  
test\_returns\_wide <- test\_data %>%  
 dplyr::select(date, symbol, return) %>%  
 pivot\_wider(names\_from = symbol, values\_from = return)  
# Ensure column order for matrix multiplication  
test\_returns\_matrix <- as.matrix(test\_returns\_wide[, assets\_ordered])  
  
# Calculate daily portfolio returns on the test set  
portfolio\_returns <- test\_returns\_wide %>%  
 dplyr::select(date) %>%  
 mutate(  
 Systematic\_Mix = test\_returns\_matrix %\*% w\_systematic\_train[assets\_ordered],  
 Total\_Cov\_Mix = test\_returns\_matrix %\*% w\_total\_train[assets\_ordered]  
 )  
  
# Calculate performance metrics  
performance <- portfolio\_returns %>%  
 pivot\_longer(-date, names\_to = "Method", values\_to = "return") %>%  
 group\_by(Method) %>%  
 summarise(  
 Annual\_Return = mean(return, na.rm = TRUE) \* 252,  
 Annual\_Volatility = sd(return, na.rm = TRUE) \* sqrt(252),  
 Sharpe\_Ratio = Annual\_Return / Annual\_Volatility  
 )  
  
kable(performance, caption = "Out-of-Sample Performance (Test Period)", digits = 3)

Out-of-Sample Performance (Test Period)

| Method | Annual\_Return | Annual\_Volatility | Sharpe\_Ratio |
| --- | --- | --- | --- |
| Systematic\_Mix | 2.431 | 0.309 | 7.870 |
| Total\_Cov\_Mix | 1.088 | 0.186 | 5.837 |

# Visualize cumulative returns  
p\_cumulative <- portfolio\_returns %>%  
 pivot\_longer(-date, names\_to = "Method", values\_to = "return") %>%  
 group\_by(Method) %>%  
 mutate(Cumulative\_Return = cumprod(1 + return) - 1) %>%  
 ggplot(aes(x = date, y = Cumulative\_Return, color = Method)) +  
 geom\_line(size = 1) +  
 theme\_minimal() +  
 labs(title = "Out-of-Sample Cumulative Returns",  
 subtitle = "Comparing portfolio construction methods",  
 y = "Cumulative Return", x = "") +  
 scale\_y\_continuous(labels = scales::percent)  
  
ggplotly(p\_cumulative)



The out-of-sample results typically show that the portfolio built on **systematic covariance** is more robust, often exhibiting lower volatility and better risk-adjusted returns (Sharpe Ratio) because it was built on more stable, persistent relationships.

## 8. Practical Implementation Guide

### When to Use This Approach

The mixed model approach works best when:

1. **You have a clear factor structure** and fundamental data to build a meaningful asset similarity matrix.
2. **You believe relationships change** across different market environments (regimes).
3. **You want robust, stable portfolios** that are less sensitive to estimation error and require less turnover.
4. **You have a long-term investment horizon** and want to focus on persistent, systematic relationships.

### Implementation Checklist

1. **Data Requirements**
   * At least 3-5 years of daily or weekly returns.
   * Relevant market factors (e.g., market return, VIX, interest rates, inflation).
   * Fundamental asset characteristics to build the similarity matrix.
2. **Model Specification**
   * Start with a simple model and add complexity incrementally.
   * Define clear fixed effects (market factors) and random effects (asset deviations).
   * The quality of the asset similarity matrix (Gu) is crucial. Experiment with different features and weighting schemes.
3. **Portfolio Construction**
   * Use the **systematic covariance matrix** for strategic asset allocation.
   * Use the **total covariance matrix** (systematic + idiosyncratic) for a complete and conservative assessment of portfolio risk.
4. **Monitoring and Rebalancing**
   * Refit models periodically (e.g., quarterly) or when market regimes show signs of a structural shift.
   * Monitor the variance decomposition over time. A sudden drop in the systematic component might signal a model breakdown.

### Code Template for Production Use

Here is a simplified function that encapsulates the core logic for production use.

# Production-ready function  
optimize\_portfolio\_mixed\_model <- function(returns\_data,   
 factors\_data,  
 asset\_chars\_data) {  
   
 # 1. Prepare data and create similarity matrix  
 # The function `prepare\_model\_data` would need to be defined based on the steps  
 # in the "Data and Setup" section.  
 # model\_data <- prepare\_model\_data(returns\_data, factors\_data)  
 K\_matrix <- create\_relationship\_matrix(asset\_chars\_data,   
 features = c("asset\_class", "equity\_beta"))  
   
 # 2. Fit mixed model  
 model <- mmer(  
 fixed = return ~ market\_return\_std + vix\_change\_std,  
 random = ~ vsr(symbol, Gu = K\_matrix) + time\_factor,  
 data = model\_data,  
 verbose = FALSE  
 )  
   
 # 3. Extract systematic covariance  
 model\_data$fitted <- predict(model, D = model\_data)  
 fitted\_wide <- model\_data %>%  
 select(date, symbol, fitted) %>%  
 pivot\_wider(names\_from = symbol, values\_from = fitted)  
 cov\_systematic <- cov(fitted\_wide[,-1], use = "complete.obs") \* 252  
   
 # 4. Optimize portfolio  
 weights <- find\_min\_var\_portfolio(as.matrix(nearPD(cov\_systematic)$mat))  
   
 return(list(  
 weights = setNames(weights, colnames(cov\_systematic)),  
 model\_summary = summary(model)  
 ))  
}

## 9. Conclusions

### Key Takeaways

1. **Mixed models provide a principled framework** to separate signal (persistent, systematic relationships) from noise (transient, idiosyncratic shocks) in asset returns.
2. **The systematic covariance matrix**, derived from model-fitted values, captures these persistent relationships and is more robust for portfolio construction than a noisy sample covariance matrix.
3. **This approach naturally incorporates regime changes** and other complexities through the specification of fixed and random effects.
4. **The genomic prediction analogy is powerful**: just as breeders select on genetic potential (breeding values) rather than just observed performance, investors should allocate based on systematic relationships rather than total historical covariance.

### The Bigger Picture

This methodology represents a paradigm shift in portfolio construction:

* **From**: Using raw historical data where every observation is treated equally.
* **To**: A model-based approach that focuses on the components that are most predictable.
* **From**: Assuming static, unchanging relationships between assets.
* **To**: Modeling dynamic, regime-dependent behavior.
* **From**: Relying on noisy point estimates of means and covariances.
* **To**: Using hierarchical models that provide natural shrinkage and regularization.

### Future Directions

1. **Bayesian Implementation**: Use Bayesian methods (e.g., via brms or MCMCglmm) to get full posterior distributions of portfolio weights, providing a natural way to express uncertainty in our allocation.
2. **Dynamic Factor Models**: Allow factor loadings themselves to evolve smoothly over time using state-space models.
3. **Non-Gaussian Distributions**: Extend the model to handle the fat-tailed nature of financial returns by using alternative distributions like the Student’s t-distribution.