Instructor Notes for Chapter 5

Chapter 5 introduces two concepts that might be new to many of you: matrices and quadratic programming. These math tools are very helpful in financial engineering. I preview these tools below.

You might find that this document is a useful guide as you study Chapter 5 and do your Chapter 5 homework.

Matrices

A matrix is a type of array comprising a set of vectors, such as a covariance matrix.

If you want to learn more about matrices, see the tutorials in the Matrix Algebra Tutorials content area in our Blackboard site.

Here’re some basics to get you started.

A vector is a one-dimensional array of values. You may think of a vector as as single column of data. For instance, the weights of assets in a two-asset portfolio my be displayed as follows:

ωp=[ω1ω2]ωp=[ω1ω2]

Or, as values:

ω=[0.60.4]ω=[0.60.4]

You’ve been using vectors in this course whenever you *concatenate* a set of data, as in R code:

y <- c(0.4, 0,6)

The transpose of a vector, denoted by superscript T, such as ωTωT *transforms* the coordinates of the vector, essentially transforming columns to row.

ωT=[ω1ω2]ωT=[ω1ω2]

Or, with values,

ωT=[0.40.6]ωT=[0.40.6]

A covariance matrix of two assets might look like this:

Σ=[σ2iσi,jσi,jσ2j]Σ=[σi2σi,jσi,jσj2]

or, with values

Σ=[0.040.060.060.09]Σ=[0.040.060.060.09]

Matrix multiplication is a bit more difficult. For example, if we wanted to compute the portfolio variance, we would use this expression:

σP=ωTΣωσP=ωTΣω

or…

ωp=[ω1ω2][σ2iσi,jσi,jσ2j][ω1ω2]ωp=[ω1ω2][σi2σi,jσi,jσj2][ω1ω2]

Let’s look at Example 1 in Chapter 5. Consider a set of four assets and suppose that the return vector *R* has mean vector and covariance matrix depicted below.

Mean vector

mu <- c(0.1, 0.2, 0.05, 0.1)

tibble(mu)

|  |
| --- |
|  |

| **mu**  <dbl> |
| --- |
| 0.10 |
| 0.20 |
| 0.05 |
| 0.10 |

4 rows

Covariance matrix

Sigma <- matrix(

c(0.05, 0.01, 0.02, 0, 0.01, 0.10, 0.05, 0.02, 0.02, 0.05, 0.20, 0.10, 0, 0.02, 0.10, 0.20), nrow=4, ncol=4)

Sigma

## [,1] [,2] [,3] [,4]

## [1,] 0.05 0.01 0.02 0.00

## [2,] 0.01 0.10 0.05 0.02

## [3,] 0.02 0.05 0.20 0.10

## [4,] 0.00 0.02 0.10 0.20

Weights vector

w <- c(0.2,0.3,0.1,0.4)

tibble(w)

|  |
| --- |
|  |

| **w**  <dbl> |
| --- |
| 0.2 |
| 0.3 |
| 0.1 |
| 0.4 |

4 rows

Weighted mean

wt\_mean <- w\*mu

tibble(wt\_mean)

|  |
| --- |
|  |

| **wt\_mean**  <dbl> |
| --- |
| 0.020 |
| 0.060 |
| 0.005 |
| 0.040 |

4 rows

Variance, and standard deviation

Note: the matrix multiplication operator, %\*%, is used to multiply matrices.

variance <- t(w) %\*% Sigma %\*% w *# matrix multiplication*

tibble(variance = variance,

std\_dev = sqrt(variance)) %>%

round(4)

|  |
| --- |
|  |

| **variance**  <dbl[,1]> | **std\_dev**  <dbl[,1]> |
| --- | --- |
| 0.0628 | 0.2506 |

1 row

Quadratic Programming

For an overview of quadratic programming, visit this Wikipedia [page](https://en.wikipedia.org/wiki/Quadratic_programming).

Quadratic programming is particularly helpful when matrix algebra is intractable or infeasible. Note, however, we’ll still use matrices to help set up the problem.

Solving linear equations

For relatively simple problems like solving the problem of two equations and two unknowns, we can use the solve() function in R. Let *A* denote an mm x mm invertible matrix and bb denote an mm x 11 vector.

Let A and b denote the corresponding R variables.

Ab=xAb=x

Then solve(A,b) return A−1bA−1b; the function with the second argument omitted, i.e., solve(A), returns A−1A−1.

Here’s an example. Assume the following:

3x+2y=83x+2y=8

x+y=2x+y=2

If we configure these equations in matrix form, we have

[3121][xy]=[82][3211][xy]=[82]

What are the values for x and y?

A <- matrix(c(3,1,2,1),nrow=2,ncol=2)

b <- matrix(c(8,2),nrow= 2, ncol = 1)

solve(A,b)

## [,1]

## [1,] 4

## [2,] -2

For more complex, quadratic equations, we’ll use the solve.QP() function.

Let’s say that we want to compute the Weight Vector of a portfolio on the minimum-risk frontier. Thus, we have a constrained minimization problem.

A constrained minimization problem of this form is an example of a *quadratic programming problem*. The type of quadratic programming problems we will consider have three components:

*Component 1*: a quadratic objective function in the form

12xTDx−dTx12xTDx−dTx

*Component 2*: Subject to a set of equality constraints of the form

ATx=bATx=b

and *Component 3* a set of inequality constraints on the elements of x.

Let’s define terms:

* x is the vector of decision variables (for our problems, this is the weight vector – over vector of values we’re solving for.)
* D is a known N x N matrix (for our problems, this, typically, will be the covariance matrix.)
* d is a known N x 1 vector (for our problems, this, typically, will be the vector of known means)
* A is a known N x k matrix (it’s the matrix of coefficients of constraints)
* b is a known k x 1 vector (the vector of constraints)

In R, we can use the function solve.QP in the package quadprog to solve a general quadratic programming. Here’s the documentation: [quadprog reference](https://cran.r-project.org/web/packages/quadprog/quadprog.pdf)

The function solve.QP has several arguments:

* Dmat corresponds to the matrix D.
* dvec corresponds to the vector d.
* Amat corresponds to the matrix A.
* bvec corresponds to the vector b.
* meq specifies the number of columns of Amat that correspond to equality constraints, with the remaining columns corresponding to inequality constraints.

We’ll use Example 5.5 in the text to describe how to use R to find the weight vectors of portfolios on the minimum-risk frontier.

Ex. 5.5 Optimization

Consider the following four assets described in Ex. 5.1, with a mean return vector:

mu <- c(0.1, 0.2, 0.05, 0.1) *# mean vector*

tibble(mu)

|  |
| --- |
|  |

| **mu**  <dbl> |
| --- |
| 0.10 |
| 0.20 |
| 0.05 |
| 0.10 |

4 rows

and return covariance matrix

Sigma <- matrix(

c(0.05, 0.01, 0.02, 0,

0.01, 0.10, 0.05, 0.02,

0.02, 0.05, 0.20, 0.10,

0, 0.02, 0.10, 0.20),

nrow = 4,

ncol = 4

)

Sigma

## [,1] [,2] [,3] [,4]

## [1,] 0.05 0.01 0.02 0.00

## [2,] 0.01 0.10 0.05 0.02

## [3,] 0.02 0.05 0.20 0.10

## [4,] 0.00 0.02 0.10 0.20

Suppose we wish to find the portfolio on the minimum-risk frontier that has mean = 0.2.

Then the objective function is to minimize the variance, which we describe in this matrix multiplication form:

σP=ωTΣωσP=ωTΣω

with constraints such that (1) the weights must sum to 1

ωT1=1ωT1=1

and (2) the minimum-risk frontier’s mean = 0.2

ωTμ=0.2ωTμ=0.2

Therefore, in the notation of solve-QP,

* dvec = mu (i.e., the vector of the average return for each asset)
* Dmat is 2Σ2Σ (i.e., 2 x the covariance matrix. Why $ 2$? Because that’s the way the author of the solve.QP function designed the Dmat argument.)
* Amat is a 4 x 2 matrix with the column given by a vector of all ones and the second column given by the mean vector μμ. (The fist column has all ones because one is the coefficient of all the weight constraints.)
* bvec is the vector (1, 0.2) – the constraints for the sum of the weight (first value) and the mean constrain of 0.2

Thus, the constraint ATω=bATω=b specifies that the constraints for the sum of the weights (i.e., ∑Ni(ωi)=1∑iN(ωi)=1) and ωTμ=0.2ωTμ=0.2

A <- cbind(c(1,1,1,1),mu)

t(A)

## [,1] [,2] [,3] [,4]

## 1.0 1.0 1.00 1.0

## mu 0.1 0.2 0.05 0.1

**library**(quadprog)

mrf1 <- solve.QP(Dmat = 2\*Sigma, *# 2 x covariance matrix*

dvec = mu, *# avg return*

Amat = A, *# constraint coefficients*

bvec = c(1, 0.2), *# values of constraints*

meq = 2) *# treat 2 constraints as qualities*

wts <- mrf1$solution

wts

## [1] 0.3617325 0.8129462 -0.3741076 0.1994288

Thus, the mean and standard deviation of the return of the portfolio

mean <- sum(mrf1$solution \* mu)

std\_dev <- (mrf1$solution %\*% Sigma %\*% mrf1$solution) ^ 0.5

tibble(mean, std\_dev) %>%

round(4)

|  |
| --- |
|  |

| **mean**  <dbl> | **std\_dev**  <dbl[,1]> |
| --- | --- |
| 0.2 | 0.2649 |

1 row

Do the weights sum to one?

sum(wts)

## [1] 1

Yes, they do!

The maximization problem

When we apply the risk-aversion criterion to our portfolio analysis problem, we engender a maximization problem. We want to maximize our mean return, given our specified level of risk, subject to our (risk-adjusted) restrictions on the portfolio weights.

Because we’re now considering a maximization problem, we’ll apply this function:

ωTμ−λ2ωTΣωωTμ−λ2ωTΣω

where the risk-aversion parameter λ>0λ>0 is given. (Note: we merely multiplied the minimization function we looked at in the minimum variance portfolio problem above by -1 to make the function a maximization function.)

Our goal is to choose the weight vector ωω that maximizes this function, which we denote by ωλωλ.

We’ll call the portfolio with weight vector ωλωλ the *risk-averse portfolio with parameter*λλ.

As noted in Proposition 5.5 in the text, for a given value of λ>0λ>0, the weight vector that maximizes

ωTμ−λ2ωTΣωωTμ−λ2ωTΣω

subject to the restriction ωT1=1ωT1=1, is given by

ωλ=ωmv+1λν⎯⎯⎯ωλ=ωmv+1λν¯

where ωmvωmv is the weight vector of the minimum variance portfolio.

ν⎯⎯⎯=Σ−1(μ−μmv1)ν¯=Σ−1(μ−μmv1)

and μmvμmv is the mean return on the minimum-variance portfolio. Let’s apply this theory in an Example 5.8 in the text.

Ex. 5.8

Consider the assets in Ex. 5.6 with mean and covariance and covariance and weights for minimum-variance portfolio:

*# mean vector*

mu <- c(0.25, 0.125, 0.3)

*# covariance matrix*

Sig <- matrix(

c(0.25, 0.1, 0.24,

0.1, 0.16, 0.096,

0.24, 0.096, 0.36),

byrow = T, nrow = 3)

*# find unnormalized weights*

w0 <- solve(Sig, c(1,1,1))

*# normalize the weights*

w\_mv <- w0/sum(w0)

w\_mv %>% round(5)

## [1] 0.24296 0.71303 0.04401

The vector νν may be calculated as follows:

m <- sum(w\_mv \* mu)

m

## [1] 0.1630722

vbar <- solve(Sig, mu - m\*c(1,1,1))

vbar %>% round(5)

## [1] 0.19440 -0.60703 0.41263

The weight vector ωλωλ corresponding to any value of λλ may now be calculated using w\_mv and vbar.

For λλ = 1

lambda <- 1

wt\_lam <- w\_mv + vbar/lambda

wt\_lam %>% round(5)

## [1] 0.43735 0.10600 0.45665

For λλ = 4

lambda <- 4

wt\_lam <- w\_mv + vbar/lambda

wt\_lam %>% round(5)

## [1] 0.29156 0.56127 0.14717

For λ=1λ=1, the mean return of the portfolio with weight vector ωλωλ is

lambda <- 1

mu <- sum((w\_mv + vbar/lambda)\*mu)

tibble(mean\_risk\_averse = mu) %>%

round(5) %>% tibble()

|  |
| --- |
|  |

| **mean\_risk\_averse**  <dbl> |
| --- |
| 0.25958 |

1 row

and the standard deviation of the return is

lambda <- 1

var <- (w\_mv+vbar/lambda) %\*% Sig %\*% (w\_mv+vbar/lambda)

sd <- var^0.5

tibble(std\_dev\_return = sd) %>%

round(5) %>% tibble()

|  |
| --- |
|  |

| **std\_dev\_return**  <dbl[,1]> |
| --- |
| 0.48899 |

1 row

For λ=4λ=4, the mean return of the portfolio with weight vector $\_{lambda} is

lambda <- 4

mu <- sum((w\_mv + vbar/lambda)\*mu)

tibble(mean\_risk\_averse = mu) %>%

round(5) %>% tibble()

|  |
| --- |
|  |

| **mean\_risk\_averse**  <dbl> |
| --- |
| 0.25958 |

1 row

and the standard deviation of the return is

lambda <- 4

var <- (w\_mv+vbar/lambda) %\*% Sig %\*% (w\_mv+vbar/lambda)

sd <- var^0.5

tibble(std\_dev\_return = sd) %>%

round(5) %>% tibble()

|  |
| --- |
|  |

| **std\_dev\_return**  <dbl[,1]> |
| --- |
| 0.38554 |

1 row

Closing comments.

If you’re finding the quadratic programming material hard to understand, don’t worry. That just means you’re normal. The quadratic programming techniques are relatively advanced material for finance (and other business) students. For starters, use the examples in the text or in my scripts as the pattern to follow as you do your homework. Refer to this document to gain insight on what the patterns mean. The more practice you get in applying these the quadratic programming techniques the quicker your understanding will develop. Before long, you’ll be able to apply these techniques without referring to the examples in the text or my scripts.