

2 General Optimization with SIMPLE

2.1 Indexing Parameters and Variables

So far, we have learned how to use NUOPT functions to solve a variety of application problems. NUOPT for S-PLUS also offers a powerful and intuitive modeling language called SIMPLE that allows us to formulate complicated models to be passed on to the numerical optimization package of NUOPT. In the next few sections, we will introduce the reader to SIMPLE with the use of a fully worked-out application example. Note that this is meant to be an introduction; many applications covering different aspects of SIMPLE are treated throughout the rest of the book.

Suppose we invest our starting wealth of one monetary unit in a portfolio with n assets. After one period, we want to minimize the average shortfall over m scenarios of our final wealth, W , below some minimum wealth requirement, W_{\min} :

$$\frac{1}{m} \sum_{s=1}^m \max(W_{\min} - W_s, 0).$$

Minimization of this equation takes place subject to the set of constraints

$$\frac{1}{m} \sum_{s=1}^m W_s \geq W_{\text{target}}, \quad (2.1)$$

$$\sum_{i=1}^n w_i = 1, \quad (2.2)$$

$$w_i \geq 0, \quad (2.3)$$

where $W_s = \sum_{i=1}^n w_i (1 + R_{is})$ for all scenarios $s = 1, \dots, m$. Equation (2.1) states that average wealth should be above a specified target wealth, W_{target} , while (2.2) and (2.3) represent the usual full investment and non-negativity constraints. In order to define an optimization problem, we usually start with parameters and variables. Parameters are those objects that are treated as constants by NUOPT. Parameters could be the coefficients in an equation, the elements of a variance-covariance matrix or the returns in a scenario matrix. In the example above, the parameters that require definition are

- the elements of the $m \times n$ scenario matrix of asset returns \mathbf{S} ,
- the target wealth W_{target} , and
- the minimum wealth W_{min} .

Apart from parameters, the problem in (2.1)–(2.3) also includes yet unknown quantities called variables. These are the n asset weights. Note that we have not yet addressed W_s . It will be treated in the next section, as we could set it up as a variable or as an expression.

Notice that all the parameters and variables in our optimization problem are indexed (i.e., they have subscripts). This is necessary to know exactly how variables and parameters interact. In SIMPLE, we need to do the same. We can define variables and parameters using the direct or indirect methods. Suppose we are given a scenario matrix of asset returns \mathbf{S} for six scenarios and four assets:

```
m <- 6
S <- 1 + matrix(rmvnorm(m,
  mean = c(0.02, 0.06, 0.08, 0.12),
  cov = diag(c(0.02, 0.05, 0.1, 0.2))), ncol = 4)
S
numeric matrix: 6 rows, 4 columns.
```

	[,1]	[,2]	[,3]	[,4]
[1,]	1.1517432	1.1100860	0.8626316	1.071162
[2,]	0.8787766	0.9618646	0.6678107	1.009150
[3,]	0.9797007	1.0657226	1.1382760	1.066186
[4,]	1.1029785	1.0891477	0.7423621	1.047908
[5,]	1.1350944	1.2769152	1.3875332	1.278555
[6,]	1.1688834	1.3251761	1.0179170	1.175775

The number of columns equals the number of assets. Let us define the set of subscripts needed for our **variables** using the **direct method**. It is called direct because it forces us to explicitly define the index set.

```

n <- Set(1:ncol(S))          # set n contains integers
                              # from 1 to 4
i <- Element(set = n)        # i defined as elements of n
w <- Variable(index = i)      # w carries subscripts i
w
  1 2 3 4
  0 0 0 0
attr(, "indexes"):
[1] "i"

```

However, we can also define variables using the **indirect method**. The advantage of the indirect method is that variables are automatically assigned starting values.

```

w.start <- as.array(c(rep(1/ncol(S), ncol(S)))
# note: NuOPT expects arrays
n <- Set()
i <- Element(set = n)
w <- Variable(w.start, index = i)
w
      1      2      3      4
0.25 0.25 0.25 0.25
attr(, "indexes"):
[1] "i"

```

It is easy to see that the number of elements specified for `w.start` defines the number of variables, their subscripts, and their starting values.

After defining our set of variables, we need to assign the correct subscripts to our **parameters**, too. Let's forget for a moment that we have already defined all variables. We need to define two sets, the first for subscripts on assets and the second for subscripts on scenarios.

```

n <- Set(1:ncol(S))
m <- Set(1:nrow(S))
# elements in set of asset subscripts
i <- Element(set=n)
# elements in set of scenario subscripts
s <- Element(set=m)
> n
{ 1 2 3 4 }
> m
{ 1 2 3 4 5 6 }

```

The `SIMPLE` command `dprod()` offers an intuitive way to define parameters that carry two indices (such as our scenario matrix).¹

```
S1 <- Parameter(index = dprod(s, i))
S1
      1 2 3 4
1 0 0 0 0
2 0 0 0 0
3 0 0 0 0
4 0 0 0 0
5 0 0 0 0
6 0 0 0 0
attr(, "indexes"):
[1] "s" "i"
```

Note that we have not yet defined the parameters. If instead we type

```
S2 <- Parameter(S, index = dprod(s, i))
S2
      1      2      3      4
1 1.1517432 1.1100860 0.8626316 1.071162
2 0.8787766 0.9618646 0.6678107 1.009150
3 0.9797007 1.0657226 1.1382760 1.066186
4 1.1029785 1.0891477 0.7423621 1.047908
5 1.1350944 1.2769152 1.3875332 1.278555
6 1.1688834 1.3251761 1.0179170 1.175775

attr(, "indexes"):
[1] "s" "i"
```

we automatically assign the correct index to each row and column of our scenario matrix. After this has been done (after the index has been set), we could also define variables (asset weights):

```
w <- Variable(index=i)
```

Parameters that also require definition are minimum wealth and target wealth. Note that neither parameter needs to be indexed. Therefore we can use the simplest definition of parameters `SIMPLE` can offer.

```
Wealth.target <- Parameter(1.07)
Wealth.min <- Parameter(1.00)
```

For later convenience, we will also define the average return an asset offers across all scenarios as a parameter:

```
# calculate mean for dimension 2 (columns) of S
mu.bar <- apply(S, 2, mean)
# define parameter
mu.bar <- Parameter(as.array(mu.bar), index=i)
```

Now that we have set up variables and parameters, we need to build the model in (2.1)–(2.3) using arithmetic operations and expressions.

2.1.1 Arithmetic Operations and Expressions

Models typically involve the calculation of expressions. In our example, an expression that needs to be calculated is $W_s = \sum_{i=1}^n w_i (1 + R_{si})$. The syntax is virtually the same as for parameters and variables:

```
# wealth is defined(indexed) in each scenario
W <- Expression(index=s)
```

However, the calculation of an expression is yet new. Again SIMPLE is very intuitive, as it allows us to directly translate summations or products into code. Table 2.1 exhibits some examples from a portfolio context. In our example above, we still need to specify the defined expression for final portfolio wealth.

```
W[s] ~ Sum(w[i] * R[s,i], i)
```

Table 2.1 SIMPLE Definitions

Name	Mathematical Formula	SIMPLE formula
Weight on asset subgroup	$\sum_{i=3}^n w_i$	Sum(w[i], i, i>2)
Wealth for each scenario	$\sum_{i=1}^n w_i (1 + R_{si})$	Sum(w[i] * R[s,i], i)
Grand mean of all scenarios	$\frac{1}{nm} \sum_{s=1}^m \sum_{i=1}^n (1 + R_{si})$	Sum(Sum(R[s,i], i), s) / (n*m)

In terms of our objective (2.1), it is necessary to distinguish between cases where final wealth is higher or lower than minimum wealth. Combining Expression and yet another function `ife()`, we can achieve this. The use of `ife(condition, Expression1, Expression2)` allows calculating an expression contingent on whether a condition is met:

```
# define positive and negative wealth
W.pos <- Expression(index=s)
W.neg <- Expression(index=s)
# calculate expression
W.pos[s] ~ ife(W[s]>=Wealth.min, W[s], 0)
W.neg[s] ~ ife(W[s]<=Wealth.min, W[s], 0)
```

We can later use `W.neg[s]`, which effectively is $\max(W_s - W_{\min}, 0)$, within our objective function to solve our simple scenario-optimization model.

2.1.2 Constraints and Objectives

Constraints can be used in many forms.² The obvious use is to constrain asset weights in very much the same way the reader might write them down on paper.

```
W[i]>=0      # non-negativity constraint on all
              # assets
W[2]>=0.2    # minimum weight constraint on asset 2
Sum(w[i], i>2, i<6)>=0.5 # assets 3 to 5 must sum
                          # up to at least 50%
Sum(w[i], i)==1      # full investment constraint
```

Alternatively, we can also use constraints to implicitly define variables. Coming back to our original example, it might be useful to define a set of variables that equal $\max(W_s - W_{\min}, 0)$ in all scenarios.

```
up <- Variable(index=s)  # index new variables
dn <- Variable(index=s)
# implicitly define deviations with the use of
# constraints
up[s]-dn[s] == Sum(w[i]*S2[s,i], s)-Wealth.min
up[s] >= 0
dn[s] >= 0
```

We stop our discussion of constraints here, as later chapters will discuss more complicated modeling situations, such as the use of constraints in mixed integer problems, in more detail.

Nothing can be optimal without an **objective**. SIMPLE allows objectives to be specified in a very natural way. It only requires declaring the name of the objective and whether it is a maximization or a minimization problem. In this model, the objective is a risk measure (average shortfall) we want to minimize.

```
risk <- Objective(type="minimize")
```

Specification of this risk measure evolves in very much the same way as before.

```
risk ~ Sum(dn[s], s) / nrow(S)
```

We have now learned all the details necessary to solve our simple scenario-optimization model outlined at the beginning of this section.

2.1.3 Build, Control, and Solve Models

Remember that SIMPLE allows us to set up an optimization model that is then passed to the numerical algorithms available in NUOPT. How is this done? The first step is to build a complete model from the ingredients above by writing a function that contains the relevant commands, as in Code 2.1.

```
scenario.model <- function(S, Wealth.min,
  Wealth.target)
{
  # define subscripts
  n <- Set(1:ncol(S))
  m <- Set(1:nrow(S))
  i <- Element(set=n)
  s <- Element(set=m)

  # parameters
  S2 <- Parameter(S, index=dprod(s,i))
  mu.bar <- apply(S, 2, mean)
  mu.bar <- Parameter(as.array(mu.bar), index=i)
  Wealth.target <- Parameter(Wealth.target)
  Wealth.min <- Parameter(Wealth.min)

  # variables
  w <- Variable(index=i)
  up <- Variable(index=s)
  dn <- Variable(index=s)
  up[s]-dn[s] == Sum(w[i]*R[s,i], i)-Wealth.min
  up[s] >= 0
  dn[s] >= 0
}
```

```

# objective
risk <- Objective(type="minimize")
risk ~ Sum(dn[s],s)/nrow(S)

# constraints
Sum(mu.bar[i]*w[i],i) == Wealth.target
Sum(w[i],i) == 1
w[i] >= 0
}

```

Code 2.1 Scenario Modeling in SIMPLE

Next we need to expand the model into a system of equations that can be solved by NUOPT using the `System` command:

```

> scenario.system <- System(scenario.model, S,
  Wealth.min, Wealth.target)
Evaluating
  scenario.model(S,Wealth.min,Wealth.target)
... ok!
Expanding (1/7) (2/7) (3/7) (4/7) (5/7) (6/7) (7/7) ok!

```

Note that the fact that our expansion went well is not a guarantee that the NUOPT solver will be able to come up with a solution or that the solution found is indeed a global maximum or minimum. In order to see what has been passed on to NUOPT, we can view the complete system using the `show` command:

```

> show(scenario.system)
1-1 : -up[1]+dn[1]+1.15174*w[1]+1.11009*w[2]
      +0.862632*w[3]+1.07116*w[4]-1 == 0
1-2 : 0.878777*w[1]+0.961865*w[2]+0.667811*w[3]
      +1.00915*w[4]-up[2]+dn[2]-1 == 0
1-3 : 0.979701*w[1]+1.06572*w[2]+1.13828*w[3]
      +1.06619*w[4]-up[3]+dn[3]-1 == 0
1-4 : 1.10298*w[1]+1.08915*w[2]+0.742362*w[3]
      +1.04791*w[4]-up[4]+dn[4]-1 == 0
1-5 : 1.13509*w[1]+1.27692*w[2]+1.38753*w[3]
      +1.27856*w[4]-up[5]+dn[5]-1 == 0
1-6 : 1.16888*w[1]+1.32518*w[2]+1.01792*w[3]
      +1.17577*w[4]-up[6]+dn[6]-1 == 0
2-1 : up[1] >= 0
2-2 : up[2] >= 0
2-3 : up[3] >= 0
2-4 : up[4] >= 0

```

```

2-5 : up[5] >= 0
2-6 : up[6] >= 0
3-1 : dn[1] >= 0
3-2 : dn[2] >= 0
3-3 : dn[3] >= 0
3-4 : dn[4] >= 0
3-5 : dn[5] >= 0
3-6 : dn[6] >= 0
4-1 : 1.06953*w[1]+1.13815*w[2]+0.969422*w[3]
      +1.10812*w[4] == 1.07
5-1 : w[1]+w[2]+w[3]+w[4] == 1
6-1 : w[1] >= 0
6-2 : w[2] >= 0
6-3 : w[3] >= 0
6-4 : w[4] >= 0
risk<objective>:0.166667*dn[1]+0.166667*dn[2]
      +0.166667*dn[3]+0.1666670*dn[4]+ 0.166667*dn[5]
      +0.166667*dn[6] (minimize)

```

While this is only informative for a small set of scenarios, it can be a useful way of checking the model for different optimization problems. Finally we solve the optimization problem using the `solve` command:

```

solution <- solve(scenario.system, trace = T)
weight <- matrix(round(solution$variable$w$current,
      digit = 5) * 100, ncol = 1)
weight
      [,1]
[1,] 25.750
[2,]  0.000
[3,] 20.321
[4,] 53.929

```

As we already know from our previous discussion, use of a small number of scenarios is completely inappropriate, as it might offer arbitrage opportunities and misrepresent the underlying distributions (sampling error). For what follows, we draw 1000 scenarios for four assets. In order to get a complete picture of investment opportunities, we want to trace out an efficient frontier (i.e., the geometric location of the minimal average shortfall, relative to a specified minimum wealth, for each given wealth target). This requires a function (given in Code 2.2) that returns portfolio weights and associated risks for a single optimization run (i.e., for a given wealth target).

```

portfolio <- function(S, Wealth.min, Wealth.target)
{
  scenario.system <- System(scenario.model, S,
    Wealth.min, Wealth.target)
  solution <- solve(scenario.system, trace=T)
  weight <-
    matrix(round(solution$variable$w$current,
      digit=5)*100, ncol=1)
  risk <- solution$objective
  return(weight,risk)
}

```

Code 2.2 Portfolio Weights Function

Finally, we need a function (given in Code 2.3) that returns an efficient frontier, ranging from a minimum to maximum wealth target.

```

scenario.frontier <- function(S, Wealth.min,
  Wealth.target, n.pf)
{
  # contains risk return results
  Risk <- matrix(0, ncol=1, nrow=n.pf)
  # n.pf denotes number of frontier portfolios
  Return <- matrix(0, ncol=1, nrow=n.pf)

  # define wealth targets
  mu.max <- max(apply(S, 2, mean))
  mu.min <- min(apply(S, 2, mean))
  mu.range<-seq(mu.min, mu.max,
    (mu.max-mu.min)/(n.pf-1))

  # contains asset weights
  weight <- matrix(0, ncol=1, nrow=ncol(S))
  for(i in 1:n.pf){
    x <- portfolio(S, Wealth.min,
      Wealth.target=mu.range[i])
    Risk[i,1] <- x$risk
    Return[i,1] <- mu.range[i]
    weight <- cbind(weight,x$weight)
  }

  # plots frontier and frontier portfolios
  graphsheet()
  par(mfrow=c(1,2))
  plot(Risk, Return, type="b")
  title("Scenario Frontier")
}

```

```

barplot(weight[, -1])
title("Frontier Portfolios")
list("optimal.weights" = weight, "Risk"=Risk,
      "Return"=Return)
}

```

Code 2.3 Scenario Model

We can now run a frontier analysis plotting target wealth versus average wealth shortfall:

```

scenario.frontier(S, Wealth.min, Wealth.target,
  n.pf=5)

```

As inputs, we require a scenario matrix, minimum and target wealth, and the number of frontier portfolios. The results are shown in Figure 2.1. Each dot represents a different portfolio with portfolio weights shown in the bar chart to the right. Optimal solutions appear to be diversified (intermediate return portfolios contain all four assets). Minimum and maximum wealth target portfolios are fully invested in the minimum and maximum return assets, respectively. This ends our brief discussion of SIMPLE; many more examples will be given in the following chapters.

2.2 Function Optimization

So far, we have always used SIMPLE within a portfolio context. However, we can also apply it to a straightforward optimization problem. This will show the reader the flexibility and generality of SIMPLE. We start with maximizing a nonlinear function of two variables,

$$-10x_1^2 + 4x_1 + \frac{1}{3}x_2^2 + 20x_2, \quad (2.4)$$

under nonlinear constraints $x_1^2 + x_2^2 \leq 16$, $x_1x_2 \geq 3$, $x_1 \geq 0$, $x_2 \geq 0$.³ The necessary code is given in Code 2.4. Note that as the problem is small we neither defined the parameters nor indexed the variables.

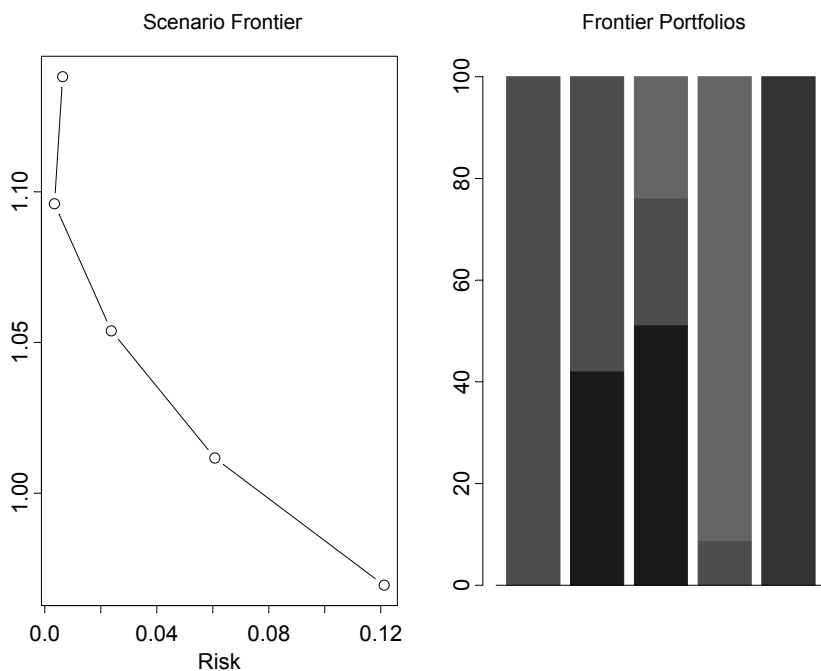


Figure 2.1 Scenario Frontier and Underlying Frontier Portfolio

```
> model.1 <- function(){
  x1 <- Variable()                                # define variables
                                              # with no index

  x2 <- Variable()
  f <- Objective(type="maximize") # define
    objective
  f ~ -10*x1^2 + 4*x1 + (1/3)*x2^2 + 20*x2
  x1^2+x2^2 <= 16                                # set constraint
  x1*x2 >= 3
  x1 >= 0
  x2 >= 0
}

> # transform model into system of equations
> system.model.1 <- System(model.1)
> # check model
> show(system.model.1)
1-1 : x1*x1+x2*x2 <= 16
2-1 : x1*x2 >= 3
3-1 : x1 >= 0
4-1 : x2 >= 0
```

```
f<objective>: -10*x1*x1+0.333333*x2*x2+4*x1+
  20*x2 (maximize)

> # solve model for x1 and x2
> x <- solve(system.model.1)

> x$variables$x1$current
[1] 0.7640694
> x$variables$x2$current
[1] 3.926346
```

Code 2.4 Maximizing a Nonlinear Function of Two Variables

However, it should be noted that the existence of a solution does not guarantee that we have found a global maximum (minimum). Suppose we want to maximize instead

$$f(x, y) = \cos(x + y)\cos(3x - y) + \cos(x - y)\sin(x + 3y) + 5 \exp\left(-\frac{x^2 + y^2}{8}\right) \quad (2.5)$$

with no constraints imposed. You should be able to write the necessary short piece of code yourself by now. In contrast with the previous example, we have used starting values to initialize the optimization:

```
> model.2 <- function(x.start, y.start){
  x <- Variable(x.start)
  y <- Variable(y.start)
  f <- Objective(type="maximize")
  f ~ cos(x+y)*cos(3*x-y) + cos(x-y)*sin(x+3*y) +
    5*exp(-(x^2+y^2)/8)
}

> system.model.2 <- System(model.2, x.start=20,
  y.start=20)
> # trace=F limits output to what is required
> solve(system.model.2, trace=F)$objective
  current
1.799038
```

In order to see whether this is a global maximum, we can plot (2.5) within the range of -5 to +5.

```
x <- seq(-5, 5, length=50) #define range for x and y
y <- seq(-5, 5, length=50)
```

```
# define function
f <- function(x,y){
  cos(x+y)*cos(3*x-y)+cos(x-y)*sin(x+3*y)+
  5*exp(-(x^2+y^2)/8)
}

# plot function
z <- outer(x,y,f)
persp(x,y,z)
contour(x,y,z, nlevels=10, xlab="x", ylab="y")
```

The objective value of 1.79 is not a global maximum, as we can see from Figure 2.2 and Figure 2.3. For a different set of starting values, we get the optimal solution shown in Code 2.5.

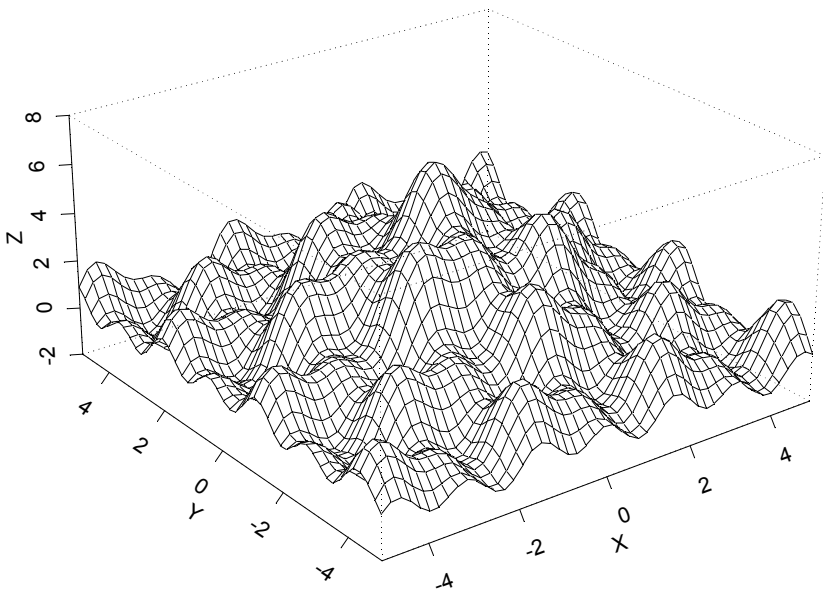


Figure 2.2 2D Plot for $\cos(x+y)\cos(3x-y) + \cos(x-y)\sin(x+3y) + 5\exp\left(-\frac{x^2+y^2}{8}\right)$

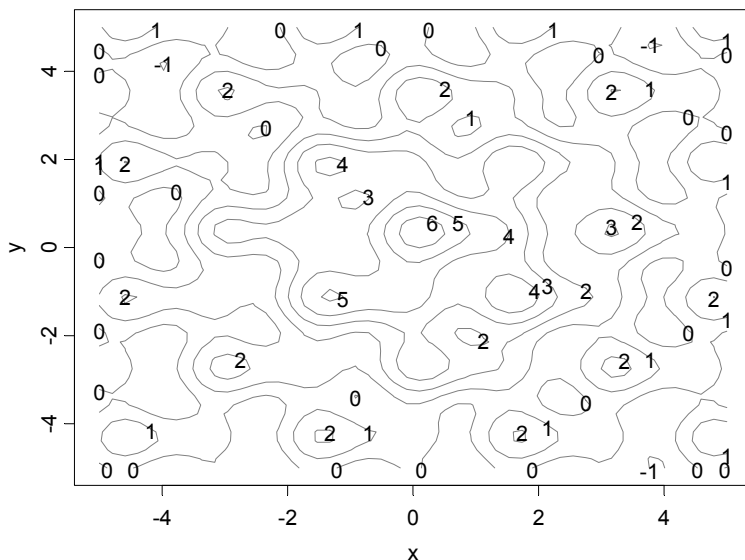


Figure 2.3 Contour Plot for $\cos(x + y)\cos(3x - y) + \cos(x - y)\sin(x + 3y)$

$$+ 5\exp\left(-\frac{x^2 + y^2}{8}\right)$$

```
> system.model.2 <- System(model.2, x.start=3.5,
  y.start=4)
> solve(system.model.2, trace=F)$objective
current
6.703374
```

Code 2.5 Function Optimization

Objective functions like that of (2.5) are called nonconvex. Their optimization will always require some heuristics. Without prior knowledge about “good” starting values, it is recommended to randomly search the space of admissible variable combinations and record the corresponding objective values in an attempt to find the global maximum. However, be aware that functions such as

$$1 - \exp\left(-\frac{1}{x^2 + y^2}\right)$$

contain many solutions with the same maximum; see Figure 2.4 and Figure 2.5.

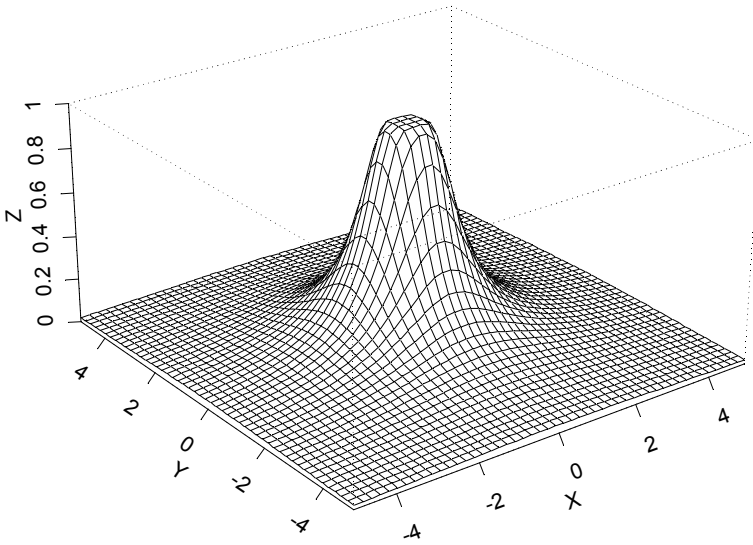


Figure 2.4 2D Plot for $1 - \exp\left(-\frac{1}{x^2 + y^2}\right)$

2.3 Maximum Likelihood Optimization

Models that can be understood intuitively and fit the data well have the greatest chance of receiving attention from practitioners. As market participants tend to think in regimes (e.g., periods of different volatility) it is natural to model a distribution as a combination of two (or more) normal distributions, with each distribution representing a different regime. Shifts from one regime to another take place randomly with a given probability. These mixtures of normal distributions have found great interest in finance, as they can model skewness as well as kurtosis and fit the data much better than a non-normal alternative. In this section, we will show how to simulate, estimate, test, and apply a mixture-of-normals model of asset returns (see Figure 2.6 using S-PLUS and NUOPT). In order to allow the reader to reproduce the calculations of this section, we will generate a data set by drawing a total of 15,000 samples from two normal distributions, each with a mean return of 10%: one with a volatility of 40%, representing a high-volatility regime, and one with a volatility of 15%, representing a low-volatility regime. The probability of a draw from the high-volatility regime is $\frac{1}{3}$. This results in the typical fat-tailed distribution that is characteristic of so many financial time series.

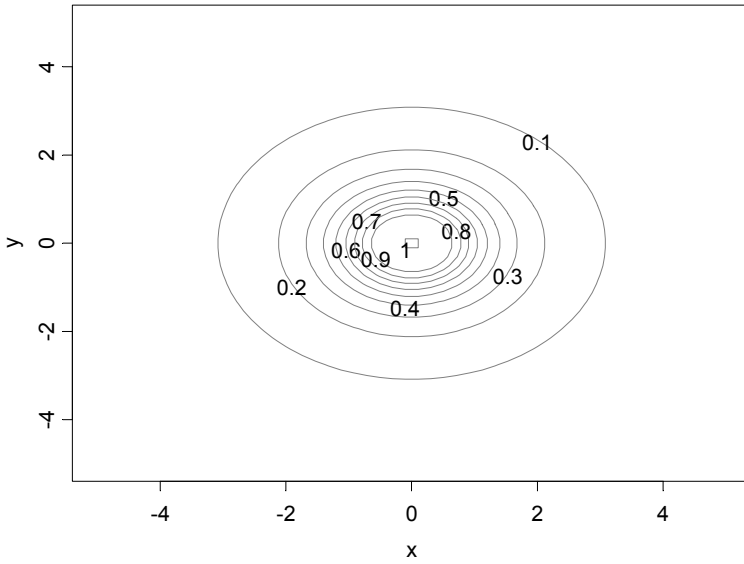


Figure 2.5 Contour Plot for $1 - \exp\left(-\frac{1}{x^2 + y^2}\right)$

```
data <- c(rnorm(5000, 0.1, 0.4),
         rnorm(10000, 0.1, 0.15))
hist(data)
```

The likelihood function for a mixture of the two normals with densities

$$f_i = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{-\frac{1}{2}\left(\frac{R_s - \mu_i}{\sigma_i}\right)^2\right\}, \quad i = 1, 2, \quad (2.6)$$

is usually written in its log form:

$$\log(L) = \sum_{s=1}^m \log(pf_1 + (1-p)f_2). \quad (2.7)$$

Maximum likelihood estimation then becomes the maximization of the objective (2.7) with respect to the variables p (probability of drawing from distribution #1), μ_1, σ_1 (mean and standard deviation of returns for distribution #1), and μ_2, σ_2 . The return data R_s for all $s = 1, \dots, m$ scenarios represent parameters (as they are fixed once the sample is drawn). We need to impose non-negativity constraints on standard deviations and on p :

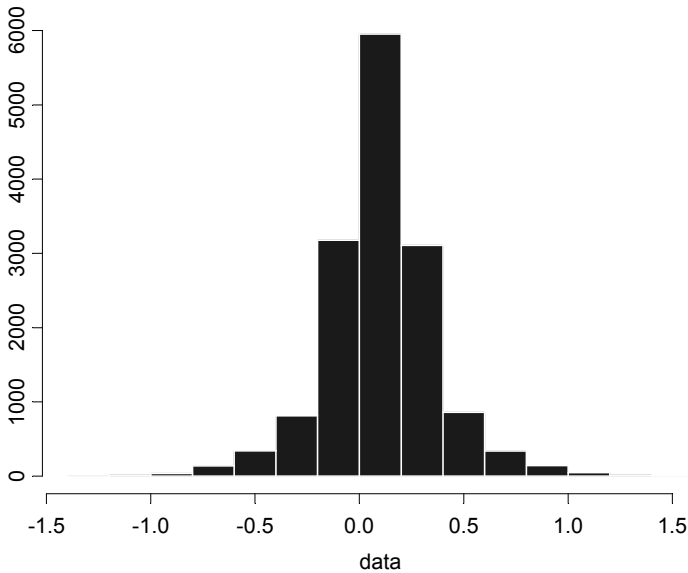


Figure 2.6 Mixture of Two Normals

```
MoN.model <- function(data){
  # parameter
  m <- Set(1:length(data))
  s <- Element(set=m)
  R <- Parameter(as.array(data), index=s)
  # variable
  I <- Set(1:5)
  i <- Element(set=I)
  p <- Variable(index=i)
  # objective
  logL <- Objective(type="maximize")
  logL ~ Sum(log(p[1]/(sqrt(2*pi)*p[3]))*
    exp(-0.5*((R[s]-p[2])/p[3])^2)
    +(1-p[1])/(sqrt(2*pi)*p[5])*
    exp(-0.5*((R[s]-p[4])/p[5])^2)),s)

  # constraints
  p[1]>=0
  p[3]>=0
  p[5]>=0
}
# solve system
MoN.system <- System(MoN.model, data=data)
```

```

solution <- solve(MoN.system, trace=F)
# get parameters and likelihood function
p <- solution$variables$p$current
> p

           1           2           3           4           5
0.3461775 0.1066749 0.3978774 0.09752759 0.1471847
attr(, "indexes"):
[1] "i"

```

We have been able to almost exactly recover our assumed parameters when simulating the data set.⁴ This is due to the high number of observations and the consequent low sampling error. However if you need assurance that our estimated distribution is significantly different from its (single) normal alternative, we can employ a likelihood ratio test (with obvious significance), as illustrated in Code 2.6.

```

LogL.uc <- sum(log(p[1]*dnorm(data, p[2], p[3]) +
  (1-p[1])*dnorm(data, p[4], p[5])))
LogL.c <- sum(log(dnorm(data, mean(data),
  stdev(data))))
LR.test <- -2*(LogL.c-LogL.uc)
LR.test
[1] 2242.879

```

Code 2.6 Maximum Likelihood Optimization and Regime Probabilities

So far, we have estimated the unconditional probability p (33%) that a given return is drawn from a hectic (i.e., high-volatility period). However, using Bayes' rule, we can also calculate the conditional probability that a given observation has been drawn from the hectic distribution

$$prob(\text{hectic} | R_s, p, \mu_1, \sigma_1, \mu_2, \sigma_2) = \frac{pf(R_s | \mu_1, \sigma_1)}{pf(R_s | \mu_1, \sigma_1) + (1-p)f(R_s | \mu_2, \sigma_2)} \quad (2.8)$$

where $f(R_s | \mu_i, \sigma_i)$ denotes the usual marginal density. We can calculate (2.8) for every data point, as shown in Figure 2.7. The probability that a given data point has been drawn from the hectic regime is highest when the observation is extreme (positive as well as negative). For small return realizations, the reverse is true. This calculation is useful, as we can use it to estimate the correlation between two assets conditional on the first asset experiencing a hectic regime.⁵

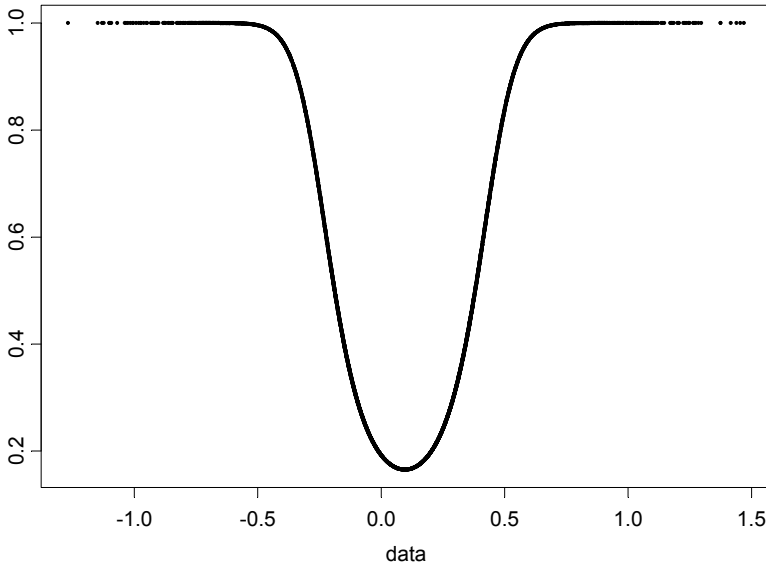


Figure 2.7 Conditional Probability

2.4 Utility Optimization

2.4.1 *Semi-quadratic Utility Maximization*

A great deal of research has been put into determining whether some types of objectives in portfolio optimization are compatible with the behavior of utility-maximizing agents. The utility function (2.9) is particularly interesting, as it has been shown that the decision-making under the mean-semi-variance objective (see Chapter 3) is fully compatible with the maximization of expected utility from

$$U_s = \begin{cases} R_{ps} & \text{for } R_{ps} \geq \tau \\ R_{ps} - \kappa(\tau - R_{ps})^2 & \text{for } R_{ps} < \tau, \end{cases} \quad (2.9)$$

where $R_{ps} = \sum_{i=1}^n w_i (1 + R_{is})$ equals the portfolio return and κ can be interpreted as a risk aversion parameter. It is also called **semi-quadratic**, as the quadratic part is only defined for $R_{ps} \leq \tau$. Figure 2.8 shows a semi-quadratic utility function for $\kappa = 4$. The expected utility maximization of (2.9) allows us

to introduce (in Code 2.7) the use of `ife()` and `Expression()` in defining an objective function that is piecewise-defined.

```
# generate scenarios for asset.1 to asset.4
scenarios <- matrix(rmvnorm(1000,
  mean=c(0.02, 0.04, 0.05, 0.08),
  cov=diag(c(0.02, 0.04, 0.1, 0.2))), ncol=4)

FUT.model <- function(scenarios, k, g, h)
{
  # number of observations and assets
  n.obs <- nrow(scenarios)
  n.assets <- ncol(scenarios)

  # NuOPT set up
  asset <- Set()
  period <- Set(1:n.obs)
  j <- Element(set=asset)
  t <- Element(set=period)

  # define parameters
  S <- Parameter(scenarios, index=dprod(t,j))
  K <- Parameter(k)
  G <- Parameter(g)
  H <- Parameter(h)

  # define "x" variable (weights)
  x <- Variable(index=j)
  R <- Expression(index=t)
  R[t] ~ Sum(x[j]*S[t,j],j)
  u <- Expression(index=t)
  u[t] ~ ife(R[t]>=G, R[t], R[t]-K*(R[t]-G)^H)

  # define utility measure
  utility <- Objective(type="maximize")
  utility ~ Sum(u[t],t)/n.obs

  # constraints (add up)
  Sum(x[j],j) == 1
  # constraints (non-negativity)
  x[j] >= 0
}

# run model
```

```
FUT.system <- System(FUT.model, scenarios, k=1,
  g=0, h=2)
solution <- solve(FUT.system, trace=T)
weight <- matrix(round(solution$variable$x$current,
  digit=4)*100, ncol=1)
```

Code 2.7 Semi-quadratic Utility Optimization

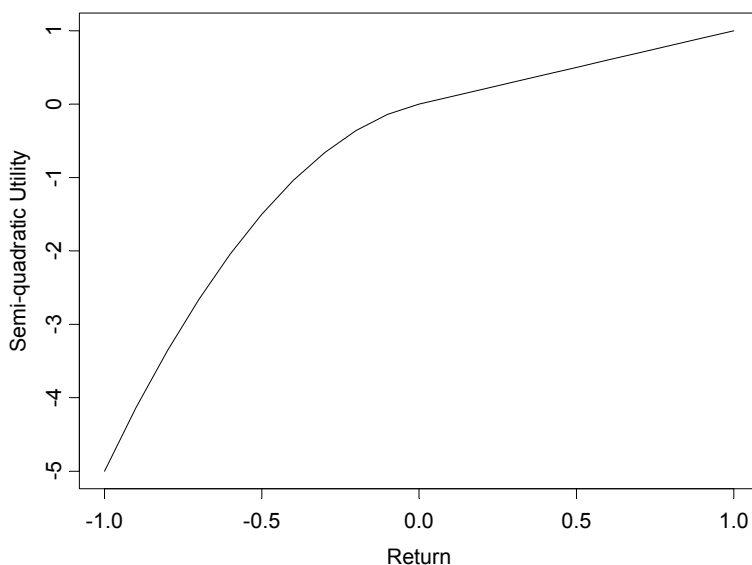


Figure 2.8 Semi-quadratic Utility

We can repeat this optimization for risk aversion parameters ranging from 1 to 10 (this is left to the reader as an exercise) to arrive at the matrix of portfolio weights for different risk aversions.

```
> weight
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]  0.01  0.09 14.89 23.18 28.07 31.38 33.74
[2,] 44.76 52.60 46.16 42.50 40.35 38.87 37.80
[3,] 18.30 21.25 19.00 17.51 16.63 16.05 15.63
[4,] 36.92 26.06 19.94 16.80 14.95 13.70 12.82

      [,8] [,9] [,10]
[1,] 35.48 36.83 37.95
[2,] 37.00 36.38 35.85
[3,] 15.34 15.12 14.93
[4,] 12.18 11.68 11.27
```

```
> barplot(weight,
  legend=paste("asset.", sep="", 1:4),
  names=paste("", sep="", 1:10),
  xlab="risk aversion", ylab="weight")
```

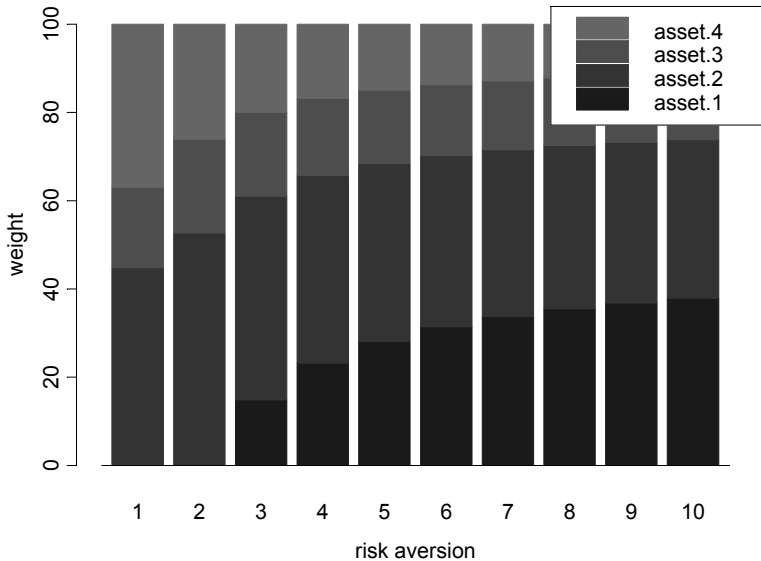


Figure 2.9 Optimal Portfolios for Semi-quadratic Utility

Figure 2.9 plots the resulting weights in a bar chart. As risk aversion rises, our portfolio optimization will reduce the weightings in the riskier assets. The reader might want to show that high risk aversions will recover the minimum variance portfolio. Why is this the case? Haven't we explicitly stated non-mean-variance objectives? The answer is that our return generation has been drawing returns from a normal distribution; hence the set of all mean-variance-efficient portfolios will also be the optimal set of portfolios for non-mean-variance preferences.

2.4.2 Utility Optimization using Piecewise Linear Approximation

In many optimization applications, it is useful to linearize a nonlinear objective function to speed up calculations or to involve linear solver technology, which is widely available and well-developed. In this section, we will show how to do this within NUOPT. Suppose we assume a standard CRRA (constant relative risk aversion) utility function that expresses utility in scenario s as

$$U_s = \begin{cases} \frac{1}{1-\gamma} \left(\sum_{i=1}^n w_i (1 + R_{is}) \right)^{1-\gamma} & \text{for } \gamma \geq 0 \\ \log \left(\sum_{i=1}^n w_i (1 + R_{is}) \right) & \text{for } \gamma = 1, \end{cases} \quad (2.10)$$

where γ reflects risk aversion. The higher γ , the higher the risk aversion. Values between 3 and 5 are assumed to be realistic for decision makers. Figure 2.10 shows (2.10) for $\gamma = 5$.

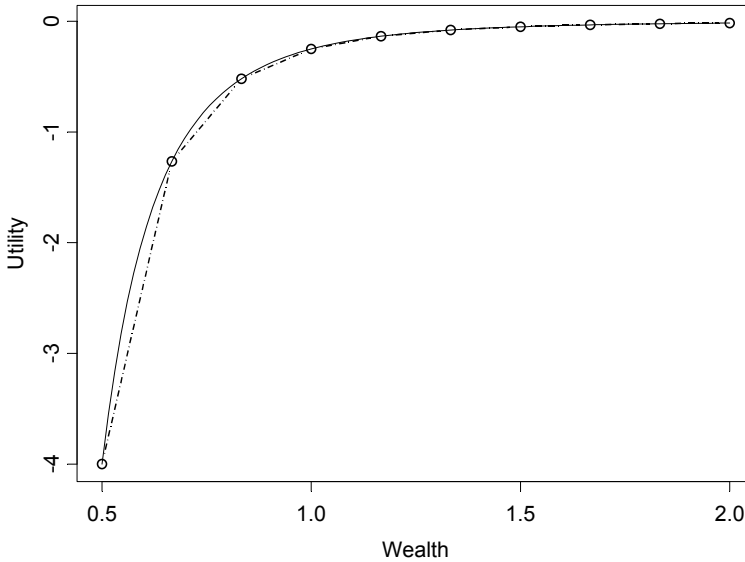


Figure 2.10 Piecewise Linear Approximation of CRRA Utility Function

```
risk.aversion <- 5
wealth <- seq(0.5, 2, length=100)
Utility <- 1/(1-risk.aversion)*
  wealth^(1-risk.aversion)
wealth.grid <- seq(0.5, 2, length=10)
utility.grid <- 1/(1-risk.aversion)*
  wealth.grid^(1-risk.aversion)
plot(wealth, Utility, type="l")
lines(wealth.grid, utility.grid, type="b", col=5)
```

Notice that higher risk aversion is marked by greater curvature in the utility function. Also, for large levels of wealth, the utility function becomes extremely flat. Hence, for large wealth levels, a linear approximation works relatively well,

while for small levels of wealth it becomes more critical (i.e., more steps are needed).

The utility function above is concave. We can therefore interpret the decreasing slopes of the approximating line segments as decreasing marginal utility. The notion of concavity is important, as it will allow us to use separate variables for each line segment. Maximizing expected utility amounts to finding

$$\max_{\mathbf{w}} \frac{1}{m} \sum_{s=1}^m U_s \quad (2.11)$$

subject to the usual non-negativity and group constraints. Before we present the final program, we will describe the basic idea behind our approach. Equation (2.11) is approximated with $j = 1, \dots, k$ line segments. The slope of each line segment (marginal utility) is denoted by a_j . We also need to define wealth for each line segment j and each scenario $s = 1, \dots, m$, leaving us with $m \cdot k$ wealth variables. Utility in state s is hence given by

$$U_s = \sum_{j=1}^k a_j W_{s,j}. \quad (2.12)$$

Note that marginal utilities and total utility (given as the product of marginal utility and wealth) are defined for their respective line segments

$$\begin{aligned} a_1 W_{1,s} & \text{ for } 0 \leq W_{1,s} \leq \bar{W}_1 \\ a_2 W_{2,s} & \text{ for } \bar{W}_1 \leq W_{2,s} \leq \bar{W}_2 \\ a_3 W_{3,s} & \text{ for } \bar{W}_2 \leq W_{3,s} \leq \bar{W}_3 \\ & \dots \end{aligned}$$

where \bar{W}_j reflects the boundary of the respective line segment. Next we need to link wealth with portfolio allocation using

$$\sum_{i=1}^n w_i (1 + R_{i,s}) = \sum_{j=1}^k W_{s,j}. \quad (2.13)$$

It might immediately come to mind that we need to impose constraints on $W_{s,j}$ that guarantee, for example, that

$$W_{2,s} = 0 \text{ for } W_{1,s} < \bar{W}_1. \quad (2.14)$$

Constraints of this format, however, would not qualify for a linear program. Fortunately we do not need to impose these constraints, as our utility function is

concave. This assures that smaller wealth variables are used first, as they have the largest impact on the objective function due to having the largest slopes. Expected utility can now be expressed as

$$\frac{1}{m} \sum_{s=1}^m \left(\sum_{j=1}^k a_j W_{s,j} \right). \quad (2.15)$$

The corresponding code is given in Code 2.8.

```
utility.model <- function(S, risk.aversion,
  wealth.grid)
{
  # check data for missing values
  if(any(is.na(S))==T)
    stop("no missing data are allowed")

  # check for number of observations and assets
  n.obs <- nrow(S)
  n.assets <- ncol(S)
  n.grids <- length(wealth.grid)

  # slope
  utility.grid <- 1/(1-risk.aversion)*
    wealth.grid^(1-risk.aversion)
  slope <- rep(0,(n.grids-1))
  for(i in 1:(n.grids-1)){
    slope[i] <- (utility.grid[i+1]-
      utility.grid[i])/(wealth.grid[i+1]-
      wealth.grid[i])
  }

  # NuOPT set up
  asset <- Set()
  period <- Set()
  grid <- Set()
  j <- Element(set=asset)
  t <- Element(set=period)
  k <- Element(set=grid)

  # define parameters
  R <- Parameter(S, index=dprod(t,j))
  slope <- Parameter(as.array(slope), index=k)
  bounds <- Parameter(as.array(wealth.grid[-1]),
    index=k)
```

```

# define variable weights
asset.weight <- Variable(index=j)

# define wealth dummies
wealth.dummy <- Variable(index=dprod(t,k))

# equalize utility and end-of-period wealth
1 + Sum(R[t,j]*asset.weight[j],j) ==
    Sum(wealth.dummy[t,k],k)

# define risk measure
utility <- Objective(type="maximize")
utility ~ Sum(Sum(wealth.dummy[t,k]*
    slope[k],k),t)/(n.obs-1)

# constraints (add up)
Sum(asset.weight[j],j) == 1
wealth.dummy[t,k] <= bounds[k]
wealth.dummy[t,k] >= 0
asset.weight[j] >= 0
}

utility.system <- System(utility.model, S,
    risk.aversion, wealth.grid)
show(utility.system)
solution <- solve(utility.system, trace=T)

```

Code 2.8 Piecewise Linearization of Utility Function

Effectively we changed a nonlinear program with a small number of variables (portfolio weights) into a linear program with a large number of variables (portfolio weights plus wealth variables).

2.5 Multistage Stochastic Programming

2.5.1 Sample Problem: Financial Planning

Dynamic stochastic programming is a highly specialized and technical field in the application of optimization techniques to financial problems.⁶ In order to introduce readers to the core concepts of stochastic programming for asset allocation problems, we will use the financial planning example introduced in Birge and Louveaux (1997) and Brandimarte (2002).

Preferences. Suppose we are equipped with initial wealth W_0 and need to meet a future liability L at the end of time $T = t_3$. In order to achieve this, we can invest in stocks and bonds. Preferences are modeled with a piecewise linear utility function (to avoid nonlinearity in the resulting mathematical program). If our final period wealth exactly meets the liabilities, we enjoy zero utility. Downside deviations are penalized with downside costs c_d , while upside deviations provide us with rewards c_u . Needless to say, the disutility from downside deviations is larger than the utility from upside deviations ($c_d > c_u$).

Scenario Tree. In contrast with all other applications in this book, we allow for intermediate decisions. (We will later see single-period models that only allow one decision at the start of an investment period.) Apart from today, we can reallocate assets at times t_1 and t_2 . At t_3 , our time horizon ends and all we can do is watch the outcome of our decisions made in t_2 . Typically, we describe uncertainty in multistage stochastic programming in a (nonrecombining) scenario tree as described in Figure 2.11. The root of a scenario tree reflects the current date where we look for an optimal decision. Note that although we allow for future decisions (in fact we choose today knowing that we can decide again later, contingent on what has happened), this does not mean that we will implement decisions at later stages as we travel through the scenario tree. In fact, a scenario tree is solved on a rolling basis. Each complete path from the root of the tree to the leaf (for example, the sequence of nodes $0 \rightarrow 2 \rightarrow 6 \rightarrow 13$) is called a scenario. Each scenario is a realization of a random variable. In the tree depicted in Figure 2.11, all scenarios are equally probable ($p_s = \frac{1}{8}$ for all s). Uncertainty is revealed as we move along the path. While at the start (t_0) we don't know which of the eight scenarios will be realized at t_3 , we know considerably more at time t_1 . If we arrive at node 1, we can say with certainty that we are in one of the scenarios w_1, w_2, w_3 , or w_4 , but we don't yet know which. For optimization purposes, this means that all decisions taken at node 1 must not be arrived at with the knowledge of which scenario will eventually come true. This information is simply not available at time t_1 . Otherwise, decisions would optimize for the known future scenario, discarding the effect of a decision on all other scenarios (which are not relevant, as they are known not to become true).

In order to keep things transparent, we have assumed that each node has two descendants with equal probability on each path. Hence each node has exactly one ancestor.

Optimization Model. We formulate the optimization model in its most direct form, called the **split-variable formulation**. Assume a_{it}^s is the amount (not weight) invested in asset i at the beginning of period t in scenario s . From the assumptions above, it is clear we need to choose allocations for two assets

($I = 2$, stocks and bonds) at three points in time (t_0, t_1, t_2) for eight scenarios each. This amounts to $48 = 2 \cdot 3 \cdot 8$ variables. Equally, R_{it}^s denotes the return of asset i in scenario $s = 1, \dots, S$ (if the return of an asset is 5%, then R_{it}^s is 1.05), where $S = 8$ in the current example. Again this means 48 return realizations. We can now start to formulate our simple asset-liability model. The investor's objective is to maximize utility arising from period t_3 ,

$$\max \sum_s p_s (c_u \cdot \text{surplus}_s^+ - c_d \cdot \text{surplus}_s^-) \quad (2.16)$$

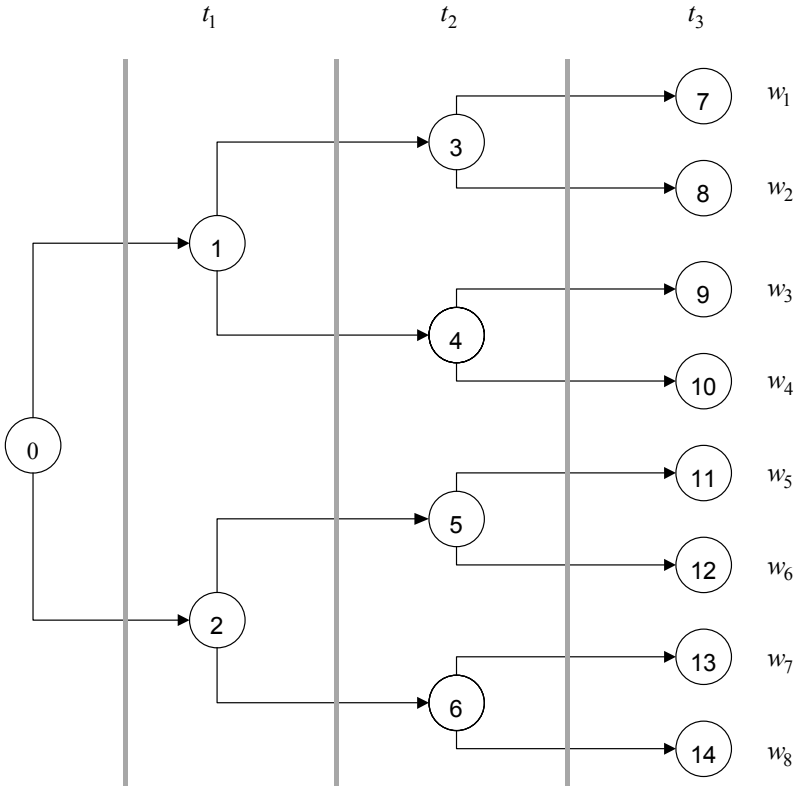


Figure 2.11 Event Tree for Multistage Stochastic Programming

where $\text{surplus}_s^+, \text{surplus}_s^-$ are variables restricted by non-negativity constraints, namely,

$$\text{surplus}_s^+, \text{surplus}_s^- \geq 0,$$

defining the surplus in state s . For any state, either $surplus_s^+$ (surplus) or $surplus_s^-$ (shortfall) is positive.

$$\sum_{i=1}^I R_{it_3}^s a_{it_2}^s = L + surplus_s^+ - surplus_s^- \quad \forall s \in S. \quad (2.17)$$

At times t_1 and t_2 , wealth accumulates over time according to

$$\sum_{i=1}^I R_{i,t}^s a_{i,t-1}^s = \sum_{i=1}^I a_{i,t}^s \quad \forall s \in S, t = t_1, t_2. \quad (2.18)$$

This ensures that we can only invest what we have earned over the last period. In addition to these intertemporal budget constraints, we need to add the well-known budget constraint (to every single scenario) at time t_0

$$\sum_{i=1}^I a_{it_0}^s = W_0 \quad \forall s \quad (2.19)$$

with the usual non-negativity constraint on invested capital

$$a_{it}^s \geq 0 \quad \forall s \forall t. \quad (2.20)$$

So far, we can perfectly adjust to the scenario tree by making optimal decisions for every scenario, which in fact allows us to look ahead to the most favorable state of the world. As this will overstate the value of the objective function, we need to enforce **nonanticipativity**: at each node, we require that allocations be the same for all scenarios that are still undistinguishable. At node 0, all allocations have to be the same across all scenarios:

$$a_{it_0}^{s=1} = a_{it_0}^{s=2} = \dots = a_{it_0}^{s=8}. \quad (2.21)$$

At nodes 1 and 2, we already know more about which path in the scenario tree we are on. However, we still need to restrict allocations to be the same for all paths crossing through nodes 1 and 2:

$$\begin{aligned} a_{it_1}^{s=1} &= a_{it_1}^{s=2} = \dots = a_{it_1}^{s=4}, \\ a_{it_1}^{s=5} &= a_{it_1}^{s=6} = \dots = a_{it_1}^{s=8}. \end{aligned} \quad (2.22)$$

Again the same logic applies to decisions taken in t_2 . Although we know considerably more in node 3, we cannot anticipate the node at which the stochastic process will arrive:

$$\begin{aligned}
 a_{it_2}^{s=1} &= a_{it_2}^{s=2}, \\
 a_{it_2}^{s=3} &= a_{it_2}^{s=4}, \\
 a_{it_2}^{s=5} &= a_{it_2}^{s=6}, \\
 a_{it_2}^{s=7} &= a_{it_2}^{s=8}.
 \end{aligned} \tag{2.23}$$

While this all looks a bit messy, we will show in the next section how we can easily implement this model within SIMPLE. In fact, all we have done is to set up a linear program with a large number of variables that are linked by nonanticipativity constraints.

2.5.2 Solving a Multistage Stochastic Program with NuOPT

In order to solve the problem laid out in the previous section, we need to specify the scenario tree first. The specification of the scenario tree is critical to the stochastic programming approach. In essence, a scenario tree tries to approximate a continuous (multivariate) distribution with a small amount of discrete scenarios. If the number of scenarios is too small or not representative of the continuous distribution, we will end up with a solution that is largely affected by estimation (specification) error. Making the number of scenarios large only partially helps: while the discrete approximation will become better, the number of variables will rise, making solutions computationally very expensive or infeasible.⁷ Suppose we specify two return realizations (binomial tree) for each asset. Either equities outperform bonds $R_{equity} = 1.25 > R_{bonds} = 1.14$ or vice versa $R_{equity} = 1.06 < R_{bonds} = 1.12$. The optimization model in the above section can be formulated shown in Code 2.9.

```

DSP.model <- function(s.eq, s.bd, u.c, l.c, W.init,
  Liab)
{
  scenarios <- Set()
  steps <- Set()
  h <- Element(set=steps)
  s <- Element(set=scenarios)
  s.eq <- Parameter(s.eq, index=dprod(s,h))
  s.bd <- Parameter(s.bd, index=dprod(s,h))
  u.c <- Parameter(u.c)
  l.c <- Parameter(l.c)
  Liab <- Parameter(Liab)
}

```

```

W.init <- Parameter(W.init)
w.eq <- Variable(index=dprod(s,h))
w.bd <- Variable(index=dprod(s,h))
upper <- Variable(index=s)
lower <- Variable(index=s)
w.eq[s,1]+w.bd[s,1] == W.init
w.eq[s,1]*s.eq[s,1] + w.bd[s,1]*s.bd[s,1] ==
  w.eq[s,2]+w.bd[s,2]
w.eq[s,2]*s.eq[s,2] + w.bd[s,2]*s.bd[s,2] ==
  w.eq[s,3]+w.bd[s,3]
w.eq[s,3]*s.eq[s,3] + w.bd[s,3]*s.bd[s,3] ==
  Liab+upper[s]-lower[s]
w.eq[s,h]>=0
w.bd[s,h]>=0
upper[s]>=0
lower[s]>=0
w.eq[1,1]==w.eq[2,1]
w.eq[2,1]==w.eq[3,1]
w.eq[3,1]==w.eq[4,1]
w.eq[4,1]==w.eq[5,1]
w.eq[5,1]==w.eq[6,1]
w.eq[6,1]==w.eq[7,1]
w.eq[7,1]==w.eq[8,1]
w.eq[1,2]==w.eq[2,2]
w.eq[2,2]==w.eq[3,2]
w.eq[3,2]==w.eq[4,2]
w.eq[5,2]==w.eq[6,2]
w.eq[6,2]==w.eq[7,2]
w.eq[7,2]==w.eq[8,2]
w.eq[1,3]==w.eq[2,3]
w.eq[3,3]==w.eq[4,3]
w.eq[5,3]==w.eq[6,3]
w.eq[7,3]==w.eq[8,3]
w.bd[1,1]==w.bd[2,1]
w.bd[2,1]==w.bd[3,1]
w.bd[3,1]==w.bd[4,1]
w.bd[4,1]==w.bd[5,1]
w.bd[5,1]==w.bd[6,1]
w.bd[6,1]==w.bd[7,1]
w.bd[7,1]==w.bd[8,1]
w.bd[1,2]==w.bd[2,2]
w.bd[2,2]==w.bd[3,2]
w.bd[3,2]==w.bd[4,2]
w.bd[5,2]==w.bd[6,2]
w.bd[6,2]==w.bd[7,2]

```



```

w.bd[7,2]==w.bd[8,2]
w.bd[1,3]==w.bd[2,3]
w.bd[3,3]==w.bd[4,3]
w.bd[5,3]==w.bd[6,3]
w.bd[7,3]==w.bd[8,3]
utility <- Objective(type="maximize")
utility ~ Sum(u.c*upper[s]-l.c*lower[s],s)
}

s.eq <- matrix(cbind(c(rep(1.25,4),rep(1.06,4)),
  rep(c(1.25, 1.25, 1.06,1.06),2),
  rep(c(1.25, 1.06),2)), ncol=3, nrow=8)
s.bd <- matrix(cbind(c(rep(1.14,4),rep(1.12,4)),
  rep(c(1.14, 1.14, 1.12,1.12),2),
  rep(c(1.14, 1.12),2)), ncol=3, nrow=8)
u.c <- 1
l.c <- 4

W.init <- 55
Liab <- 80

DSP.system <- System(DSP.model, s.eq, s.bd, u.c,
  l.c, W.init, Liab)
solution <- solve(DSP.system, trace=T)

```

Code 2.9 Stochastic Multiperiod Optimization

The solution is given below. We see that the nonanticipativity constraints force allocations to be the same for indistinguishable states of the world.

```

solution
$variables:
$w.eq:
      1      2      3
1 41.47927 65.09458 8.383990e+001
2 41.47927 65.09458 8.383990e+001
3 41.47927 65.09458 7.041860e-011
4 41.47927 65.09458 7.041860e-011
5 41.47927 36.74322 7.041443e-011
6 41.47927 36.74322 7.041443e-011
7 41.47927 36.74322 6.400000e+001
8 41.47927 36.74322 6.400000e+001
attr(,"indexes"):
[1] "s" "h"

$w.bd:

```

```

      1      2      3
1 13.52073 2.168138 7.857768e-011
2 13.52073 2.168138 7.857768e-011
3 13.52073 2.168138 7.142857e+001
4 13.52073 2.168138 7.142857e+001
5 13.52073 22.368029 7.142857e+001
6 13.52073 22.368029 7.142857e+001
7 13.52073 22.368029 5.604172e-011
8 13.52073 22.368029 5.604172e-011
attr(, "indexes"):
[1] "s" "h"

$upper:
      1      2      3      4      5
24.79988 8.870299 1.428571 1.114117e-012 1.428571
      6      7      8
1.114083e-012 1.079866e-012 6.548158e-013

attr(, "indexes"):
[1] "s"

$lower:
      1      2      3
6.548158e-013 6.548159e-013 6.548167e-013

      4      5
1.588384e-012 6.548167e-013
      6      7      8
1.588446e-012 1.663613e-012 12.16
attr(, "indexes"):
[1] "s"

$objective:
[1] -12.11268

```

2.5.3 An Alternative Formulation

The formulation above was intuitive but unfortunately requires us to use many variables (too many if the number of scenarios or variables becomes larger). Alternatively, we can write the financial planning problem using what is called the **compact formulation**. First let us distinguish between the root node $n = 0$, decision nodes $n \in D = \{1, \dots, 6\}$, and end nodes $n \in E = \{7, 8, \dots, 14\}$. As each node (apart from node 0) has exactly one ancestor (nonrecombining tree), we

can identify every ancestor with a deterministic function $f(n)$. For example, $f(n=4)=1$ (i.e., the unique ancestor of node 4 is given by node 1). For each end node, we define $surplus_s^+, surplus_s^- > 0$ and the objective also remains the same:

$$\max \sum_s p_s (c_u \cdot surplus_s^+ - c_d \cdot surplus_s^-). \quad (2.24)$$

The decisions at node 0 need to satisfy the budget constraint $\sum_{i=1}^I a_{it_0}^s = W_0$. For all other decision nodes, we require

$$\sum_{i=1}^I R_{i,n} a_{i,f(n)} = \sum_{i=1}^I a_{i,n} \quad \forall n \in D. \quad (2.25)$$

We cannot use more wealth than the wealth generated by moving from the ancestor of n to n itself. Also, for all end nodes we demand

$$\sum_{i=1}^I R_{i,n} a_{i,f(n)} = L + surplus_s^+ - surplus_s^- \quad \forall n \in E. \quad (2.26)$$

The model is completed with $a_{i,n} \geq 0$. Apart from the usual dummy variables (defining nonzero surplus levels), we only require $14 = 2 \cdot 7$ variables. The formulation in SIMPLE is left to readers as an exercise.⁸

2.6 Optimization within S-Plus

2.6.1 Optimization with One Variable

While NUOPT offers powerful optimization routines, the standard version of S-PLUS already comes with some built-in functions. We will review these functions using some simple finance-related examples.

A Simple Root Finding Problem. For continuous functions of one variable, S-PLUS offers the functions `uniroot` (find a zero) and `optimize` (find a local minimum). Suppose we need to find the internal rate of return of a savings plan that invested 1000 dollars every year for 10 years. The final wealth was 18,000 dollars. Mathematically, we need to solve for

$$-\frac{1000}{(1+y)^1} - \frac{1000}{(1+y)^2} - \dots - \frac{1000}{(1+y)^{10}} + \frac{18000}{(1+y)^{11}} = 0.$$

We first write a function that calculates the internal rate of return for a series of equally spaced cash flows.

```
IRR <- function(cash.flow)
{
  pv <- function(x, cash.flow){
    sum(cash.flow*
        (1/(1+x))^(1:length(cash.flow)-1))
  }
  irr <- uniroot(pv, c(-0.99, 0.99),
    cash.flow=cash.flow)$root
}
```

In the example above, our cash flows need to be expressed by `cash.flow <- c(rep(-1000,10),18000)`. If we call `IRR(cash.flow)`, we get 0.1046 (about 10.5%). Note that we could have also used the function `polyroot(z)`, as the problem of finding the appropriate internal rate of return is indeed a polynomial:

$$1000a + 1000a^2 + \dots + 1000a^{10} - 18000a^{11} = 0, a = (1+y)^{-1}. \quad (2.27)$$

The reader is encouraged to try this function.

Implied Volatility. Another typical root-finding problem in finance is to find the implied volatility of an option from quoted prices. The **implied volatility** ($\sigma_{implied}$) is defined as the volatility that equalizes model price (under the assumption that we use the correct model) and quoted price. We hence need to find

$$C_{market} - C_{Black-Scholes}(\sigma_{implied}) = 0. \quad (2.28)$$

Suppose a one year, at the money European call option trades at 16%. The one year interest rate is 3%. We first code a function to generate Black-Scholes model prices,

$$C_{Black-Scholes} = S \cdot N(d) - \exp(-rT) \cdot X \cdot N(d - \sigma\sqrt{T})$$

$$d = \frac{1}{\sigma\sqrt{T}} \left(\log\left(\frac{S}{X}\right) + \left(r + \frac{1}{2}\sigma^2\right)T \right). \quad (2.29)$$

```
Black.Scholes.Call <- function(S,X,r,Time,sigma)
{
  d1 <- (log(S/X)+(r+0.5*sigma^2)*Time) /
    (sigma*sqrt(Time) )
```

```

d2 <- d1-sigma*sqrt(Time)
premium <- S*pnorm(d1)-exp(-r*Time)*X*pnorm(d2)
list("premium"=premium)
}

```

Now we can define and solve the root-finding problem using Code 2.10.

```

f <- function(sigma,premium2){
  Black.Scholes.Call(S,X,r,Time,sigma)$premium -
  premium2
}

iv <- uniroot(f,c(0,1),keep.xy=T,premium2=0.16)
iv$root

```

Code 2.10 Root-Finding Problems

The implied volatility for the example above is 37%. It needed six evaluations (find out with `iv$nf`) to arrive at this result. Alternatively, we could have directly applied Newton's method.⁹ The updating formula for volatility can be expressed as

$$\sigma_{s+1} = \sigma_s + \left(\frac{dC}{d\sigma}\right)^{-1} [C_{\text{market}} - C_{\text{Black-Scholes}}(\sigma_s)], \quad (2.30)$$

where

$$\frac{dC}{d\sigma} = S\sqrt{T} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}x^2\right) \quad (2.31)$$

is also called “vega” (the option sensitivity to changes in volatility). We suggest extending the option code by two additional lines to allow for clean programming.

```

vega <- 1/(sqrt(2*pi)*exp((-d1^2)/2))*sqrt(Time)
list("premium"=premium, "vega"=vega)

```

Additionally we need to specify an initial estimate for σ as well as the maximum number of iterations and a convergence threshold. Having done this, we can code the following function that allows us to find implied volatilities via Newton's method:

```

sigma <- 0.2
max.it <- 10
tol <- 0.000000001

```

```

for(i in 1:max.it)
{
  diff <- premium -
    Black.Scholes.Call(S,X,r,Time,sigma)$premium
  vega <- Black.Scholes.Call(S,X,r,Time,sigma)$vega
  sigma <- sigma+diff/vega
  if(abs(diff) < tol) break
}
sigma

```

The code above will loop through a maximum number of iterations (`max.it`) but will stop as soon as the difference between the actual premium and model premium is sufficiently small (`tol`).

Portfolio Optimization and the Log-normality Assumption. Suppose now an investor is planning for her retirement and needs to decide how to split up her wealth between equities (or any other risky portfolio) and cash. Rather than going through the mathematics of optimal asset allocation, we will apply scenario optimization in conjunction with the standardized binomial density. The standardized binomial density can be regarded as the discrete version of the continuous standard normal density (at equally spaced points). The random variable z_i is evaluated at points $i = 0, 1, \dots, n$, where $z_i = (2i - n)/\sqrt{n}$. The associated probability $b(z_i)$ amounts to $(n!/i!(n-i)!)(\frac{1}{2})^n$. In S-PLUS, we can easily generate z_i and $b(z_i)$:

```

n <- 100
z <- (2*(0:n)-n)/sqrt(n)
b.z <- dbinom(0:n, n, 0.5)

```

Note that z_i has mean zero and variance 1 (and skewness of zero and kurtosis of 3). In the first instance, we generate returns that are log-normally distributed $R_i = \exp(\mu + \sigma z_i) - 1$. Note that the expected return and variance of R_i can be calculated from $E(R) = \sum_{i=1}^n \exp(\mu + \sigma z_i) b(z_i)$ and $Var(R) = \sum_{i=0}^n (R_i - E(R))^2 b(z_i)$. We can also find a discrete version of the normal distribution with exactly the same expected return and variance:

```

R.lz <- exp(0.1+0.2*z)-1
mean <- sum(b.z*R.lz)
sig <- sqrt(sum(b.z*(R.lz-mean)^2))
R.z <- mean+sig*z

```

However, both distributions will exhibit different shapes. While the normal distribution is symmetric (offering the same odds for large negative and positive deviations from the mean return), the log-normal distribution shows fewer large negative returns and more large positive returns than the normal distribution. The reason for this is the positive skewness of the log-normal distribution. While the log-normal distribution is a correct representation of total returns (unleveraged losses should never exceed 100%), the normal distribution is not. We are accustomed to applying the normal distribution because of its additivity (weighted means yield portfolio mean), which unfortunately does not carry over to the lognormal distribution. Now we want to ask how allocations differ if we use the correct log-normal distribution rather than the normal distribution.¹⁰ Suppose our investor aims at maximizing expected (power) utility. She hence wants to maximize

$$E(U) = \frac{1}{1-\gamma} E(W_i^{1-\gamma}) = \frac{1}{1-\gamma} \sum_{i=0}^n [1 + wR_i + (1-w)r] b(z_i), \quad (2.32)$$

where $r = 0.03$ denotes the risk-free rate and γ expresses the degree of risk aversion. For normally distributed returns, we calculate utility from Code 2.11.

```
utility <- function(w, R=R.z)
{
  wealth <- 1+w*R+(1-w)*0.03
  sum(1/(1-g)*(wealth)^(1-g)*b.z)
}
```

Code 2.11 Edgeworth Expansion and Portfolio Optimization

The optimal portfolio allocation can now be found from `optimize(utility, c(0,1), maximum=T)$maximum` and amounts to 47.89% when $\gamma = 4$. For lognormally distributed returns, instead we find the optimal allocation to be 56.59%, an almost 9% difference. It becomes clear from this example that the choice of distribution is not trivial.

2.6.2 General Optimization

Credit Risk. The assessment of credit risk attracts increased interest by banks, regulators, and institutional investors. At its heart is the so-called **loss distribution** (i.e., the probability distribution of credit-related losses) illustrated in Figure 2.12. All risk quantities (expected loss, unexpected loss, economic capital, etc.) can be derived from it. We know from standard corporate finance that holding a corporate bond is equivalent to holding a long position in government bonds (assumed to be free of credit risk) and a short position in out-of-the-money put options on the underlying assets of the firm (written to

shareholders in exchange for a premium that allows higher coupons for corporate bond returns). The return due to changes in the fate of the firm (the market value of underlying assets) is highly asymmetric. There is a high likelihood of zero losses (the put expires worthless) and a small likelihood of very large losses (the put ends in the money and is exercised by shareholders; i.e., the corporation defaults). Candidates for these kinds of asymmetric distributions are the general Beta distribution, the Weibull distribution, and the Gamma distribution, among others. Suppose credit losses \tilde{L} follow a Gamma distribution and we observe losses L_i for $i=1, \dots, n$. How do we fit the distribution to the data? We suggest using maximum likelihood methods; that is, we work out the likelihood function for the Gamma $G(\alpha, \beta)$ distribution,

$$f(L) = \frac{\beta^\alpha}{\Gamma(\alpha)} L^{\alpha-1} \exp(-\beta L), \quad 0 < L < \infty. \quad (2.33)$$

Note that $E(L) = \alpha/\beta$ and $\text{Var}(L) = \alpha/\beta^2$.

$$l(\alpha, \beta | L) = \prod_{i=1}^n f(L_i | \alpha, \beta) = \frac{\beta^{n\alpha}}{\Gamma(\alpha)^n} \left(\prod_{i=1}^n L_i \right)^{\alpha-1} \exp(-\beta \sum_{i=1}^n L_i). \quad (2.34)$$

Now solve for the distribution parameters (α, β) that give the drawn data sample the maximum likelihood. Calculating first and second derivatives of the Gamma likelihood is a tedious algebraic exercise. Fortunately, S-PLUS offers a fast and reliable alternative. The function `nlminb` (find local minimum for smooth functions subject to box constraints) offers the most general optimization routine in S-PLUS.¹¹

First we define the (log) likelihood function for the sampled data by summing over the log densities:

```
log.L <- function(x) {
  e <- log(dGamma(LOSSES, x[1], x[2]))
  -sum(e)
}
```

Note that we put a minus sign in front of the sum, as `nlminb` is designed to minimize functions. Next we sample 1000 draws from a hypothetical distribution and call the optimization routine using Code 2.12.

```
> LOSSES <- rGamma(1000, 1, 5)
> hist(LOSSES, probability=T, xlab="LOSSES",
      main="GAMMA DISTRIBUTION OF CREDIT LOSSES")
> result <- nlminb(start=c(1,1), objective=log.L)
```



```
> result$parameters  
[1] 1.044975 5.228908
```

Code 2.12 Fitting a Loss Distribution

GAMMA DISTRIBUTION OF CREDIT LOSSES

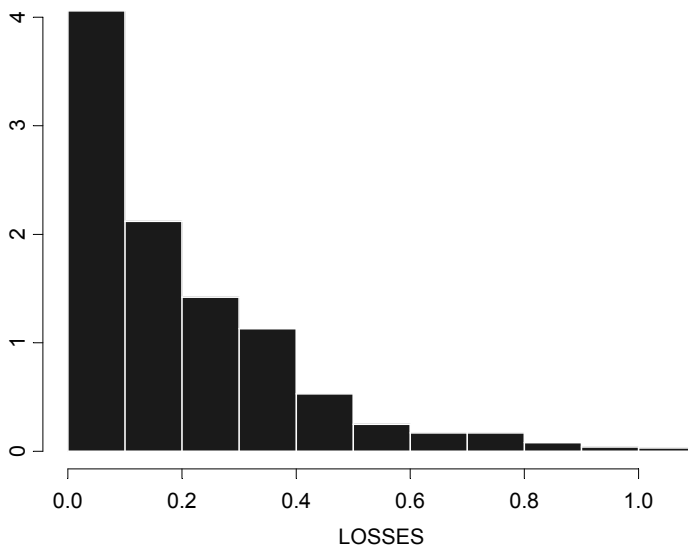


Figure 2.12 Loss Distribution

All we need to provide is a set of starting values (`start=c(1,1)`) and an objective (`objective=log.L`). The result differs from the simulated distribution due to sampling error. Note that we did not need a vector of first derivatives (`gradient`) or matrix of second derivatives (`Hessian`), as S-PLUS will approximate them with finite differences instead. Of course, if the derivatives are known (or have been calculated using `deriv`), they can be supplied. For more details, see the S-PLUS manual.

Term Structure Calibration. Modern academic finance generates a variety of term structure models; it can be difficult, even for the most ambitious researcher, to stay current with all of them. While some models require complex numerical techniques, parsimonious and intuitive models are more appealing to practitioners. One model of this sort (a similar version is used by the French Central Bank to calculate monthly available zero coupon rates) is described below. It postulates that the zero curve has the form¹²

$$R(T, \theta) = level - spread \left(\frac{1 - \exp(-aT)}{aT} \right) + curvature \left(\frac{(1 - \exp(-aT))^2}{4aT} \right) \quad (2.35)$$

$$\theta = (level, spread, curvature, a).$$

If maturity goes to infinity, all we are left with is the *level* term. We can hence interpret it as the long-term interest rate. If maturity goes to zero, the *spread* converges to $level - R(T, \theta)$ and can hence be regarded as a long-short spread. The two remaining parameters are *curvature*, describing how much the curve bends up or down, and the scale parameter, a , which can be interpreted as the strength of the mean reversion. Assuming our yield curve model is correct, we create a data set by introducing random error to our parameterized model (see Figure 2.13):

```
level <- 4
spread <- 1
curvature <- 10
a <- 1
Time <- seq(1, 30, 0.25)
zero <- (level-spread*((1-exp(-a*Time))/(a*Time)) +
  curvature*((1-exp(-a*Time))^2/(4*a*Time))) +
  rnorm(length(Time))/50
plot(Time, zero, ylab="ZERO RATE",
  xlab="TIME TO MATURITY")
```

In order to fit the model to the simulated data, we minimize the squared difference between model yield and simulated yield,

$$\min_{\theta} \sum_{i=1}^n [R(T) - R(T, \theta)]^2. \quad (2.36)$$

We summarize the minimization above in a special function called `term.fit`. This function is then minimized with respect to its parameter vector (see Code 2.13).

```
term.fit <- function(x) {
  zero.fit <- (x[1]-x[2]*
    ((1-exp(-x[4]*Time))/(x[4]*Time)) +
    x[3]*((1-exp(-x[4]*Time))^2/(4*x[4]*Time)))
  e <- sum((zero.fit-zero)^2)
  zero.fit
}
```

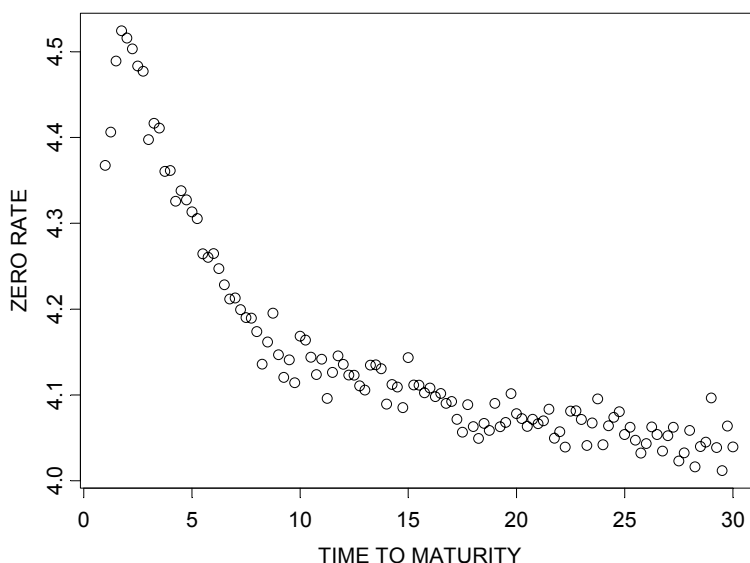


Figure 2.13 Simulated Yield Curve Data

```
result <- nlminb(start=c(1,1,1,0.4),
  objective=term.fit,
  lower=c(0.0001,0.0001,0.0001,0.0001),
  upper=c(10, 5, 40, 1))
x <- result$parameters
zero.fit <- (x[1]-x[2]*
  ((1-exp(-x[4]*Time))/(x[4]*Time)) +
  x[3]*((1-exp(-x[4]*Time))^2/(4*x[4]*Time)))
plot(Time, zero, pch=1, ylab="ZERO RATE",
  xlab="TIME TO MATURITY")
points(Time, zero.fit, type="l")
```

Code 2.13 Term Structure Fitting

The result is shown in Figure 2.14. With the principle above in mind, we could also fit this particular zero curve to any coupon bond curve, as the functional form for the zero curve is known and any coupon bond can be modeled as a portfolio of associated zero coupon bonds.

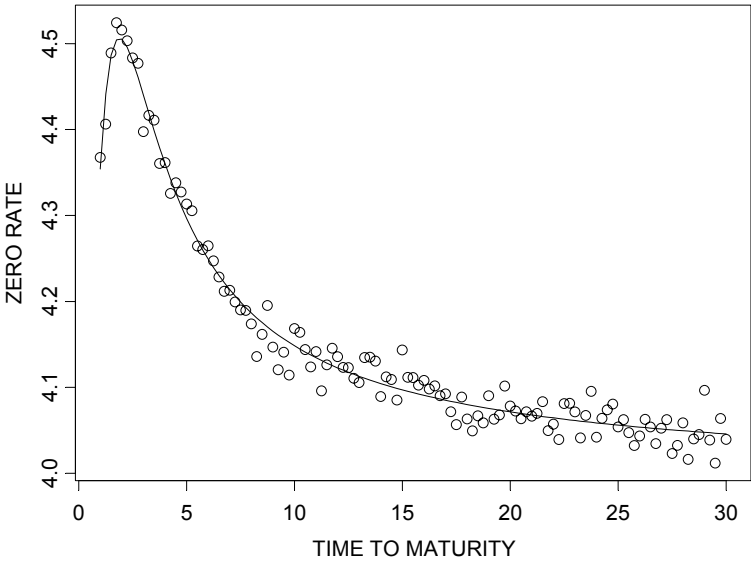


Figure 2.14 Simulated and Fitted Yield Curve Data

Exercises

1. Repeat the scenario optimization example, but solely use `Expressions` to define end-of-period wealth. Do not include variables other than asset weights.
2. Minimize the function: $f(x, y) = (x - y^2)(x - 4y^2)$. Check your solution. What is going on?
3. Use `SIMPLE` to program an active portfolio optimization using the covariance approach.
4. Use the program you wrote in Exercise 3 to trace out an active frontier without and with short-selling constraints. How do the frontiers differ? When does the benchmark matter in portfolio construction? Hint: See Grinold and Kahn (2000).

Endnotes

¹ We have used different names for the SIMPLE versions of *S* to avoid conflicts—SIMPLE objects do not work like regular *S-PLUS* objects.

² We always need the “ \leq ” in order to get closed constraint sets.

³ The reader is encouraged to check the convexity of the objective function by calculating the matrix of 2nd derivatives (the Hessian matrix) and checking that it is positive semi-definite.

⁴ Note that the solution to the SIMPLE system is not unique, as the roles of the two distributions can be interchanged (i.e., if $(p_1, p_2, p_3, p_4, p_5)$ is a solution, then so is $(1 - p_1, p_4, p_5, p_2, p_3)$).

⁵ See Kim and Finger (2000) for more details on how to use a mixture of normals for stress testing and risk management.

⁶ See Ziemba and Mulvey (1998) or Scherer (2004).

⁷ Scenario generation is a complex field in its own right. Interested readers are referred to Ziemba (2003) for a nice review.

⁸ Readers are encouraged to send in their solutions. The best program will be printed in the next edition, and the developer will get a single copy of all further editions of this book for free.

⁹ See Judd (1998) for an economics-related textbook on numerical techniques.

¹⁰ See the excellent exposition by Campbell and Viceira (2002).

¹¹ As `nlminb` encapsulates other functions such as `ms()` or `nlmin()` we will not discuss these.

¹² See El Karoui et al. (1998). It has been named the extended Vasicek model.



<http://www.springer.com/978-0-387-21016-2>

Modern Portfolio Optimization with NuOPT™, S-PLUS®,
and S+Bayes™

Scherer, B.; Martin, R.D.

2005, XXII, 406 p., Hardcover

ISBN: 978-0-387-21016-2