CHAPTER 12

The Leverage Space Portfolio Model in the Real World

n.b. The balance of this text attempts to show a viable means for applying the theories in Part I of the text, "The Optimal f *Framework," and the resultant portfolio model of Chapter 10, "The Leverage Space Model." As such, terminologies used will reflect the new model. Rather than speak of* market systems, *we will refer to* scenario spectrums. *Rather than speak of* trades *or* plays, *or* results over a certain period, *we will speak of* scenarios. *However, the reader is alerted of the interchangeability of these terms.*

In applying the concepts of the Leverage Space Model in the real world, the problems can be twofold. First, there is the computational aspect. Fortunately, this can be overcome with computer power and good software. There is no reason, from a computational standpoint, to *not* employ the Leverage Space Portfolio Model. The discernment of the scenario spectrums, their constituent scenarios, and the joint probabilities between them are no more incalculable than, say, a stock's beta or a correlation coefficient.

The second impediment to employing these concepts in the real world has been that people's utility preference curves are *not* ln. People do not act to maximize their returns. Rather, they act to maximize returns within an acceptable level of risk.

This chapter shows how to maximize returns within a given level of risk. This is a far more real-world approach than the conventionally practiced mean-variance models. Further, risk, as used in this chapter, rather than

being the ersatz risk metric of "variance (or semivariance) in returns," as in classical portfolio construction, is addressed here as being risk of ruin or risk of drawdown to a certain degree. Thus, the Leverage Space Model has, as its risk metric, drawdown itself—seeking to provide the maximum gain for a given probability of a given level of drawdown.

Let us first consider the "Classical Gambler's Ruin Problem," according to Feller.[1] Assume a gambler wins or loses one unit with probability p and $(1 - p)$, respectively. His initial capital is z and he is playing against an opponent whose initial capital is $u - z$, so that the combined capital of the two is u.

The game continues until our gambler whose initial capital is z sees it grow to u, or diminish to 0, in which case we say he is *ruined*. It is the probability of this ruin that we are interested in, and this is given by Feller as follows:

$$RR = \frac{\left((1-p)/p\right)^u - \left((1-p)/p\right)^z}{\left((1-p)/p\right)^u - 1} \tag{12.01}$$

This equation holds if $(1 - p) \neq p$ (which would cause a division by 0). In those cases where $(1 - p)$ and p are equal:

$$RR = 1 - \frac{z}{u} \tag{12.01a}$$

The following table provides results of this formula according to Feller, where RR is the risk of ruin. Therefore, $1 - RR$ is the probability of success.[2]

Row	p	$(1 - p)$	z	u	RR	P (Success)
1	0.5	0.5	9	10	0.1	0.9
2	0.5	0.5	90	100	0.1	0.9
3	0.5	0.5	900	1000	0.1	0.9
4	0.5	0.5	950	1000	0.05	0.95
5	0.5	0.5	8000	10000	0.2	0.8

[1]William Feller, "*An Introduction to Probability Theory and Its Applications*," Volume 1 (New York: John Wiley & Sons, 1950), pp. 313–314.

[2]I have altered the variable names in some of Feller's formulas here to be consistent with the variable names I shall be using throughout this chapter, for the sake of consistency.

Row	*p*	(1 − *p*)	*z*	*u*	*RR*	*P* (Success)
6	0.45	0.55	9	10	0.210	0.790
7	0.45	0.55	90	100	0.866	0.134
8	0.45	0.55	99	100	0.182	0.818
9	0.4	0.6	90	100	0.983	0.017
10	0.4	0.6	99	100	0.333	0.667
11	0.55	0.45	9	10	0.035	0.965
12	0.55	0.45	90	100	0.000	1.000
13	0.55	0.45	99	100	0.000	1.000
14	0.6	0.4	90	100	0.000	1.000
15	0.6	0.4	99	100	0.000	1.000

Note in the table above the difference between row 2, in an even money game, and the corresponding row 7, where the probabilities turn slightly against the gambler. Note how the risk of ruin, *RR*, shoots upward.

Likewise, consider what happens in row 6, where, compared to row 7, the probabilities p and $(1 - p)$ have not changed, but the size of the stake and the target have changed (z and u—in effect, going from row 7 to row 6 is the same as if we were betting 10 units instead of 1 unit on each play!). Note that now the risk of ruin has been cut to less than a quarter of what it was on row 7. Clearly, in a seemingly negative expectation game, one wants to trade in higher amounts and quit sooner. According to Feller,

> *In a game with constant stakes, the gambler therefore minimizes the probability of ruin by selecting the stake as large as consistent with his goal of gaining an amount fixed in advance. The empirical validity of this conclusion has been challenged, usually by people who contend that every "unfair" bet is unreasonable. If this were to be taken seriously, it would mean the end of all insurance business, for the careful driver who insures against liability obviously plays a game that is technically unfair. Actually there exists no theorem in probability to discourage such a driver from taking insurance.*[3]

For our purposes, however, we are dealing with situations considerably more complicated than the simple dual-scenario case of a gambling illustration, and as such we will begin to derive formulas for the more complicated situation. As we leave the classical ruin problem according to Feller, keep in mind that these same principles are at work in investing as well, although the formulations do get considerably more involved.

[3]Feller, p. 316

Let's consider now what we are confronted with, mathematically, when there are various outcomes involved, and those outcomes are a function of a stake that is multiplicative across outcomes as the sequence of outcomes is progressed through.

Consider again our two-to-one coin toss with $f = .25$:

+2, −1	(Stream)
1.5, .75	(HPRs)

There are four possible chronological permutations of these two scenarios, as follows, and the terminal wealth relatives (TWRs) that result:

$$1.5 \times 1.5 = 2.25$$
$$1.5 \times .75 = 1.125$$
$$.75 \times 1.5 = 1.125$$
$$.75 \times .75 = .5625$$

Note that the expansion of all possible scenarios into the future is like that put forth in Chapter 6.

Now let's assume we are going to consider that we are ruined if we have only 60% ($b = .6$) of our initial stake. I will attempt to present this so that you can recognize how intuitively obvious this is. Take your time here. (Originally, I had considered this chapter as the entire text—there is a lot to cover here. The concepts of *ruin* and *drawdown* will be covered in detail.) Looking at the four outcomes, only one of them ever has your TWR dip to or below the absorbing barrier of .6, that being the fourth sequence of $.75 \times .75$. So we can state that in this instance, the risk of ruin of .6 equity left at any time is $^1/_4$:

$$RR(.6) = {}^1/_4 = .25$$

Thus, there is a 25% chance of drawing down to 60% or less on our initial equity in this simple case.

Any time the interim product <= $RR(b)$, we consider that ruin has occurred.

So in the above example:

$$RR(.8) = {}^2/_4 = 50\%$$

In other words, at an f value of .25 in our two-to-one coin-toss scenario spectrum, half of the possible arrangements of HPRs leave you with 80% or less on your initial stake (i.e., the last two sequences shown see 80% or less at one point or another in the sequential run of scenario outcomes).

Expressed mathematically, we can say that at any i in (12.02) if the interim value for (12.02) <= 0, then ruin has occurred:

$$\sum_{i=1}^{q}\left(\left(\prod_{t=0}^{i-1} HPR_t\right) * HPR_i - b\right) \tag{12.02}$$

where: $HPR_0 = 1.0$

q = The number of scenarios in multiplicative sequence (in this case 2, the same as n).[4]

b = That multiple on our stake, as a lower barrier, where we determine ruin to occur (0 <= b <= 1).

Again, if at any arbitrary q, we have a value <= 0, we can conclude that ruin has occurred.

One way of expressing this mathematically would be:

$$\text{int}\left(\frac{\sum_{i=1}^{q}\left(\left(\prod_{t=0}^{i-1} HPR_t\right) * HPR_i - b\right)}{\sum_{i=1}^{q}\left|\left(\left(\prod_{t=0}^{i-1} HPR_t\right) * HPR_i - b\right)\right|}\right) = \beta \tag{12.03}$$

where: $HPR_0 = 1.0$

q = The number of scenarios in multiplicative sequence.

$$\sum_{i=1}^{q}\left|\left(\left(\prod_{t=0}^{i-1} HPR_t\right) * HPR_i - b\right)\right| \neq 0$$

In (12.03) note that β can take only one of two values, either 1 (ruin has not occurred) or 0 (ruin has occurred).

There is the possibility that the denominator in (12.03) equals 0, in which case β should be set to 0.

We digress for purpose of clarity now. Suppose we have a stream of HPRs. Let us suppose we have the five separate HPRs of:

.9
1.05
.7
.85
1.4

[4]For the moment, consider q the same as n. Later in this chapter, they become two distinct variables.

Further, let us suppose we determine b, that multiple on our stake, as a lower barrier, where we determine ruin to occur, as .6. The table below then demonstrates (12.03) and we can thus see that ruin has occurred at $q = 4$. Therefore, we conclude that this stream of HPRs resulted in ruin (even though ruin did not occur at the final point, the fact that it occurs at all, at any arbitrary point, is enough to determine that the sequence ruins).

q	**1**	**2**	**3**	**4**	**5**
HPR	0.9	1.05	0.7	0.85	1.4
TWR 1	0.9	0.945	0.6615	0.562275	0.787185
TWR − .6	0.3	0.345	0.0615	−0.03773	0.187185
TWR − .6/[TWR − .6]	1	1	1	−1	1

Using the mathematical sleight-of-hand, taking the integer of the quantity a sum divided by the sum of its absolute values (12.03), we derive a value of $\beta = \text{int}\left(\frac{3}{5}\right) = \text{int}(.6) = 0$. If the value in column 4 in the last row is 1, then $\beta = 1$.

Note that in (12.03) the HPRs appear to be taken in order; that is, they appear in a single, ordered sequence. Yet, we have four sequences in our example, so we are calculating β for each sequence. Recall that in determining optimal f, sequence does not matter, so we can use any arbitrary sequence of HPRs.

However, in risk-of-ruin calculations, order *does* matter(!) and we must therefore consider all permutations in the sequence of HPRs. Some permutations at a given set (b, $\text{HPR}_1 \ldots \text{HPR}_n$) will see $\beta = 0$, while others will see $\beta = 1$. Further, note that for n HPRs, that is, for $\text{HPR}_1 \ldots \text{HPR}_n$, there are n^n permutations.

Therefore, β must be calculated for all permutations of n things taken n at a time. The symbology for this is expressed as:

$$\forall nPn \tag{12.04}$$

More frequently, this is referred to as "for all permutations of n things taken q at a time," and appears as:

$$\forall nPq \tag{12.04a}$$

This is the case even though, for the moment in our discussion, $n = q$.

Notice that for n things taken q at a time, the total number of permutations is therefore n^q.

We can take the sum of these β values for all permutations (of n things taken q at a time, and again here, $n = q$ for the moment), and divide by the number of permutations to obtain a real probability of ruin, with $ruin$ defined as dropping to b of our starting stake, as $RR(b)$:

$$RR(b, q) = \frac{\forall nPq \sum_{k=1}^{n^q} \beta_k}{n^q} \tag{12.05}$$

This is what we are doing in discerning the probability of ruin to a given b, $RR(b)$. If there are two HPRs. There are $2 \times 2 = 4$ permutations, from which we are going to determine a β value for each [using $RR(.6)$]. Summing these β values and dividing by the number of permutations, 4, gives us our probability of ruin.

Note the input parameters. We have a value for b in $RR(b)$—that is, the percentage of our starting stake left. Various values for b, of course, will yield various results. Additionally, we are using HPRs, implying we have an f value here. Different f values will give different HPRs will give different values for β. Thus, what we are ultimately concerned with here—and the reader is advised at this point not to lose sight of this—is that we are essentially looking to hold b constant in our analysis and are concerned with those f values that yield an acceptable $RR(b)$. That is, we want to find those f values that give us an acceptable probability for a given risk of ruin.

Again we digress now for purposes of clarifying. For the moment, let us suspend the notion of each play's being a multiple on our stake. That is, let us suspend thinking of these streams in terms of HPRs and TWRs. Rather, let us simply contemplate the case of being presented with the prospect of three consecutive coin tosses. We can therefore say that there are eight separate streams, eight permutations, that the sequence which H and T may comprise ($\forall_2 P_3$).

H H H
H H T
H T H
H T T (ruin)*
T H H
T H T
T T H (ruin)
T T T (ruin)

Now let us say that if tails occurs in two consecutive tosses, we are ruined. Thus, we are trying to determine how many of those eight streams see two consecutive tails. That number, divided by eight (the number of permutuations) is therefore our "Probability of Ruin."

The situation becomes more complex when we add in the concept of multiples now. For example, in the previous example it may be that if the first toss is heads, then two subsequent tosses of tails would not result in ruin as the first play resulted in enough gain to avert ruin in the two subsequent tosses of tails*.

We return now to assigning HPRs to our coin tosses at an optimal f value of .25 and b of .6.

Note what happens as we increase the number of plays—in this case, from two plays (i.e., $q = 2$) to three plays ($q = 3$):

$$
\begin{aligned}
&\forall_2 P_3 = \\
&1.5 \times 1.5 \times 1.5 = 3.375 \\
&1.5 \times 1.5 \times .75 = 1.6875 \\
&1.5 \times .75 \times 1.5 = 1.6875 \\
&1.5 \times .75 \times .75 = .84375 \\
&.75 \times 1.5 \times 1.5 = 1.6875 \\
&.75 \times 1.5 \times .75 = .84375 \\
&.75 \times .75 \times 1.5 = .84375 \qquad \text{(ruin)} \\
&.75 \times .75 \times .75 = .421875 \qquad \text{(ruin)}
\end{aligned}
$$

Only the last two sequences saw our stake drop to .6 or less at any time.

$$RR(.6) = 2/8 = .25$$

Now for four plays:

$$
\begin{aligned}
&\forall_2 P_4 = \\
&1.5 \times 1.5 \times 1.5 \times 1.5 = 5.0625 \\
&1.5 \times 1.5 \times 1.5 \times .75 = 2.53125 \\
&1.5 \times 1.5 \times .75 \times 1.5 = 2.53125 \\
&1.5 \times 1.5 \times .75 \times .75 = 1.265625 \\
&1.5 \times .75 \times 1.5 \times 1.5 = 2.53125 \\
&1.5 \times .75 \times 1.5 \times .75 = 2.53125 \\
&1.5 \times .75 \times .75 \times 1.5 = 1.265625 \\
&1.5 \times .75 \times .75 \times .75 = .6328125 \\
&.75 \times 1.5 \times 1.5 \times 1.5 = 2.53125 \\
&.75 \times 1.5 \times 1.5 \times .75 = 1.265625
\end{aligned}
$$

$$.75 \times 1.5 \times .75 \times 1.5 = 1.265625$$
$$.75 \times 1.5 \times .75 \times .75 = .6328125$$
$$.75 \times .75 \times 1.5 \times 1.5 = 1.265625 \quad \text{(ruin)}$$
$$.75 \times .75 \times 1.5 \times .75 = .6328125 \quad \text{(ruin)}$$
$$.75 \times .75 \times .75 \times 1.5 = .6328125 \quad \text{(ruin)}$$
$$.75 \times .75 \times .75 \times .75 = .31640625 \quad \text{(ruin)}$$

Here, only the last four saw our stake drop to .6 or lower of initial equity at any time.

$$RR(.6) = 4/16 = .25$$

And now for five plays:

$$\forall_2 P_5 =$$
$$1.5 \times 1.5 \times 1.5 \times 1.5 \times 1.5 = 7.59375$$
$$1.5 \times 1.5 \times 1.5 \times 1.5 \times 0.75 = 3.796875$$
$$1.5 \times 1.5 \times 1.5 \times 0.75 \times 1.5 = 3.796875$$
$$1.5 \times 1.5 \times 1.5 \times 0.75 \times 0.75 = 1.8984375$$
$$1.5 \times 1.5 \times 0.75 \times 1.5 \times 1.5 = 3.796875$$
$$1.5 \times 1.5 \times 0.75 \times 1.5 \times 0.75 = 1.8984375$$
$$1.5 \times 1.5 \times 0.75 \times 0.75 \times 1.5 = 1.8984375$$
$$1.5 \times 1.5 \times 0.75 \times 0.75 \times 0.75 = 0.94921875$$
$$1.5 \times 0.75 \times 1.5 \times 1.5 \times 1.5 = 3.796875$$
$$1.5 \times 0.75 \times 1.5 \times 1.5 \times 0.75 = 1.8984375$$
$$1.5 \times 0.75 \times 1.5 \times 0.75 \times 1.5 = 1.8984375$$
$$1.5 \times 0.75 \times 1.5 \times 0.75 \times 0.75 = 0.94921875$$
$$1.5 \times 0.75 \times 0.75 \times 1.5 \times 1.5 = 1.8984375$$
$$1.5 \times 0.75 \times 0.75 \times 1.5 \times 0.75 = 0.94921875$$
$$1.5 \times 0.75 \times 0.75 \times 0.75 \times 1.5 = 0.94921875$$
$$1.5 \times 0.75 \times 0.75 \times 0.75 \times 0.75 = 0.474609375 \quad \text{(ruin)}$$
$$0.75 \times 1.5 \times 1.5 \times 1.5 \times 1.5 = 3.796875$$
$$0.75 \times 1.5 \times 1.5 \times 1.5 \times 0.75 = 1.8984375$$
$$0.75 \times 1.5 \times 1.5 \times 0.75 \times 1.5 = 1.8984375$$
$$0.75 \times 1.5 \times 1.5 \times 0.75 \times 0.75 = 0.94921875$$
$$0.75 \times 1.5 \times 0.75 \times 1.5 \times 1.5 = 1.8984375$$
$$0.75 \times 1.5 \times 0.75 \times 1.5 \times 0.75 = 0.94921875$$
$$0.75 \times 1.5 \times 0.75 \times 0.75 \times 1.5 = 0.94921875$$
$$0.75 \times 1.5 \times 0.75 \times 0.75 \times 0.75 = 0.474609375 \quad \text{(ruin)}$$

$0.75 \times 0.75 \times 1.5 \times 1.5 \times 1.5 = 1.8984375$ (ruin)
$0.75 \times 0.75 \times 1.5 \times 1.5 \times 0.75 = 0.94921875$ (ruin)
$0.75 \times 0.75 \times 1.5 \times 0.75 \times 1.5 = 0.94921875$ (ruin)
$0.75 \times 0.75 \times 1.5 \times 0.75 \times 0.75 = 0.474609375$ (ruin)
$0.75 \times 0.75 \times 0.75 \times 1.5 \times 1.5 = 0.94921875$ (ruin)
$0.75 \times 0.75 \times 0.75 \times 1.5 \times 0.75 = 0.474609375$ (ruin)
$0.75 \times 0.75 \times 0.75 \times 0.75 \times 1.5 = 0.474609375$ (ruin)
$0.75 \times 0.75 \times 0.75 \times 0.75 \times 0.75 = 0.237304688$ (ruin)

Now my probability of ruin has *risen* to 10/32, or .3125. This is very disconcerting in that the probability of ruin increases the longer you continue to play.

Fortunately, this probability has an asymptote. In this two-to-one coin-toss game, at the optimal f value of .25 per play, it is shown the table below:

Play #	***RR*(.6)**
2	0.25
3	0.25
4	0.25
5	0.3125
6	0.3125
7	0.367188
8	0.367188
9	0.367188
10	0.389648
11	0.389648
12	0.413818
13	0.413818
14	0.436829
15	0.436829
16	0.436829
17	0.447441
18	0.447441
19	0.459791
20	0.459791
21	0.459791
22	0.466089
23	0.466089
24	0.47383
25	0.47383
26	0.482092

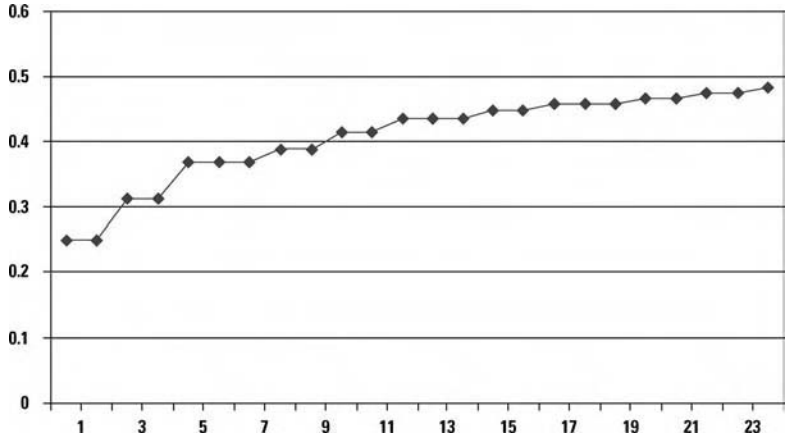

FIGURE 12.1 *RR*(.6) for two-to-one coin toss at $f = .25$

From this data, in methods to be detailed later in the text, we can determine that the asymptote, that is, the risk of ruin (defined as 60% of our initial equity left in this instance) is .48406 *in the long-run sense—that is, if we continue to play indefinitely.*

As shown in Figure 12.1, as q approaches infinity, $RR(b)$ approaches a horizontal asymptote. That is, $RR(b)$ *can* be determined in the long-run sense.

Additionally, it is perfectly acceptable to begin the analysis at $q = 1$, rather than $q = n$. Doing so aids in resolving the line and hence its asymptote.

Note that near the end of the previous chapter, a method employed in one form or another by a good deal of the larger, more successful trend-following funds, which "can be said to combine mean-variance with value at risk" was presented. Note that in the method presented—that is, in the way it is currently employed—it is akin to doing simply one run through the data, horizontally, with $n = q$ and solely for one value of k. Note that it would be if we only looked at the history of two tosses of our coin; there is no way we can approach or discern the asymptote through such a crude analysis.

Remember a very important caveat in this analysis: As demonstrated thus far it is assumed that there is no statistical dependency in the sequence of scenario outcomes across time. That is, we are looking at the stream of scenario outcomes across time in a pure sample with replacement manner; the past scenario outcome(s) do not influence the current one.

And what about more than a single scenario spectrum? This is easily handled by considering that the HPRs of the different scenario spectrums

cover the same time period. That is, we may have our scenarios derived such that they are the scenarios of outcomes for the next month—or the next day, and so on.

We therefore consider each combination of scenarios for each scenario spectrum. Thus, if we are looking at two scenario spectrums of our two-to-one coin toss, we would then have the following four outcomes:

Game 1	+2	+2	−1	−1
Game 2	+2	−1	+2	−1

The reason we have four outcomes is that we have two scenario spectrums with two scenarios in each. Thus, n in this case will equal 4.

When you have more than one scenario spectrum:

$$n = \prod_{i=1}^{m} \#\, scenarios_i \tag{12.06}$$

where: $m =$ The number of scenario spectrums you are including.

In other words, n is the product of all the scenario spectrums we are considering.

And in our example here, since there are two scenario spectrums ($m = 2$), each with two scenarios, we have $n = 4$.

The HPRs then for these four outcomes are the arithmetic average HPRs across scenario spectrums. The arithmetic average is used simply because an HPR represents the effect of trading one unit at a given value of f on the entire account.

So, if we assume we are going to trade at f values of .25,.25 in our example, we then have the following:

Game 1	+2	+2	−1	−1
Game 2	+2	−1	+2	−1

Converted to HPRs at .25,.25:

Game 1	1.5	1.5	.75	.75
Game 2	1.5	.75	1.5	.75

Arithmetic mean:

1.5	1.125	1.125	.75

Thus, we have $n = 4$, and the four values are (1.5, 1.125, 1.125, .75), which we would then use in our analysis.

We digress now. To this point, we have been discussing the probability of ruin, for an aggregate of one or more market systems or scenario spectrums. Risk of ruin, $RR(b)$ represents the probability of hitting or penetrating the lower absorbing barrier of $b\times$ initial stake. Thus, this lower absorbing barrier does *not* migrate upward, as equity may increase. If an account therefore increases twofold, this barrier does not move. For example, if $b = .6$, on a \$1 million account, then the lower absorbing barrier is at \$600,000. If the account doubles now, to \$2 million, then the lower absorbing barrier is *still* at \$600,000.

This might be what many want to use in determining their relative f values across components—that is, their allocations.

However, far more frequently we want to know the probabilities of touching a lower absorbing barrier from today—actually, from our highest equity point. In other words, we are concerned with risk of drawdown, far more so in most cases than risk of ruin. If our account doubles to \$2 million now, rather than being concerned with its going back and touching or penetrating \$600,000, we are concerned with its coming down or penetrating double that, or of its coming down to \$1.2 million.

This is so much the case that in most instances, for most traders, fund managers, or anyone responsible in a field exposed to risk, it is the de-facto and organically derived[5] definition of risk itself: "The probability of drawdown," or more precisely, the probability of a $1 - b$ percentage regression from equity highs [referred to herein now as $RD(b)$].

Again, fortunately, risk of drawdown [$RD(b)$] is very closely linked to risk of ruin [$RR(b)$], so much so that we can slide the two in and out of our discussion merely by changing Equation (12.03) to reflect risk of drawdown instead of risk of ruin:

$$\text{int}\left(\frac{\sum_{i=1}^{q}\left(\min\left(1.0,\left(\prod_{t=0}^{i-1}HPR_t\right)\right)*HPR_i-b\right)}{\sum_{i=1}^{q}\left|\left(\min\left(1.0,\left(\prod_{t=0}^{i-1}HPR_t\right)\right)*HPR_i-b\right)\right|}\right)=\beta \quad (12.03a)$$

where: $HPR_0 = 1.0.$

$$\sum_{i=1}^{q}\left|\left(\min\left(1.0,\left(\prod_{t=0}^{i-1}HPR_t\right)\right)*HPR_i-b\right)\right|\neq 0$$

[5]All too often, the definition of risk in literature pertaining to it has *ignored* the fact that this is exactly what practitioners in the field define risk to be! Rather than the tail wagging the dog here, we opt to accept this real-world definition for risk.

Calculating in β in subsequent equations by (12.03a) will give you risk of drawdown, as opposed to mere risk of ruin.

The main difference in the mechanics of (12.03a) over (12.03) is that at any time in the running product of HPRs, if the running product is greater than 1.0, then the value 1.0 is replaced for the running product at that point.

Herein is some very bare-bones Java code for calculating equation (12.05) for one or more scenario spectrums, for determining either risk of ruin [$RR(b)$] or risk of drawdown [$RD(b)$]:

```
import java.awt.*;
import java.io.*;
import java.util.*;


public class MaxTWR4VAR{
    String lines [];
    String msnames [];
    double f [];
    double b;
    boolean usedrawdowninsteadofruin;
    double plays[][];
    double hprs [][];
    double hpr [];//the composite (arithmetic average per
   time period) of the hprs
    int N; //the number of plays.Capital used to correspond
   to variables in the book
    long NL;// N as a long to avoid many casts
    public MaxTWR4VAR(String[] args){
        try{
            b=Double.parseDouble(args[1]);
        }catch(NumberFormatException e){
            System.out.println("Command Line format:
   MaxTWR4VAR inputfile riskofdrawdown(0.0..1.0)
   calculateRD(true/false)");
            return;
        }
        if(args.length>2){
   usedrawdowninsteadofruin=Boolean.valueOf (args[2])
        .booleanValue();
        }
        getinputdata(args[0]);
        createHPRs();
        control();
    }
```

```
 public static void main(String[] args){
     MaxTWR4VAR maxTWR4VAR = new MaxTWR4VAR(args);
 }

 protected void getinputdata(String fileName){
     String filetext = readInputFile(fileName);
     lines = getArgs(filetext,"\r\n");
     N=lines.length-2;
     NL=(long)N;
     plays=new double[N][];
     for(int i=0;i<lines.length;i++){
         System.out.println("line "+i+" : "+lines[i]);
         if(i==0){
             msnames = getArgs(lines[i],",");
         }else if(i==1){
             f =
convertStringArrayToDouble(getArgs(lines[i],","));
         }else{
             plays[i-2]=
convertStringArrayToDouble(getArgs(lines[i],","));
         }
     }
     System.out.println("b     : "+b);
     if(usedrawdowninsteadofruin){
         System.out.println("pr of : drawdown");
     }else{
         System.out.println("pr of : ruin");
     }

 }

 protected void  createHPRs(){
     //first find the biggest loss
     double biggestLoss[] = new double [N];
     hprs = new double [plays[0].length][N];
     Arrays.fill(biggestLoss,Double.MAX_VALUE);
     for(int j=0;j<msnames.length;j++){
         for(int i=0;i<N;i++){
             if(plays[i][j]<biggestLoss[j]){
                 biggestLoss[j]=plays[i][j];
             }
         }
     }
     //fing the hpr for each msnames for each associated
f
     for(int j=0;j<msnames.length;j++){
```

```
            for(int i=0;i<N;i++){
                hprs[j][i]= 1.0 + f[j]  (-plays[i][j] /
biggestLoss[j]);
            }
        }
        //take the arithmetic average of the hprs
        hpr = new double[N];
        for(int i=0;i<N;i++){//go through each play
            for(int j=0;j<msnames.length;j++){ //go through
each msnames
                hpr[i] += hprs[j][i];
            }
        }
        for(int i=0;i<N;i++){
            hpr[i] /= (double)msnames.length;
        }
    }

    protected String readInputFile(String fileName){
        FileInputStream fis = null;
        String str = null;
        try {
            fis = new FileInputStream(fileName);
            int size = fis.available();
            byte[] bytes = new byte [size];
            fis.read(bytes);
            str = new String(bytes);
        } catch (IOException e) {
        } finally {
            try {
                fis.close();
            } catch (IOException e2) {
            }
        }
        return str;
    }

    protected String[] getArgs(String parameter, String
delimiter){
        String args[];
        int nextItem=0;
        StringTokenizer stoke=new
StringTokenizer(parameter,delimiter);
        args=new String[stoke.countTokens()];
        while(stoke.hasMoreTokens()){
            args[nextItem]=stoke.nextToken();
```

```
            nextItem=(nextItem+1)%args.length;
        }
        return args;
    }

    protected double [] convertStringArrayToDouble(String
[] s){
        double [] d = new double[s.length];
        for(int i = 0; i<s.length; i++){
            try{
                d[i]=Double.parseDouble(s[i]);
            }catch(NumberFormatException e){
                d[i]=0.0;
            }
        }
        return d;
    }

    protected int B(double [] hprset,boolean drawdown){
        double interimHPR=1.0;
        double previnterimHPR=1.0;
        double numerator=0.0;
        double denominator=0.0;
        for(int i=0;i<hprset.length;i++){
            double useinvalue = previnterimHPR;
            if(drawdown && previnterimHPR>1.0)
                useinvalue = 1.0;

            interimHPR = useinvalue x hprset[i];
            //interimHPR = previnterimHPR x hprset[i];
            double value = interimHPR - b;
            numerator += value;
            denominator += Math.abs(value);
            previnterimHPR = interimHPR;
        }
        if(denominator==0.0){
            return 0;
        }else{
            double x = (numerator/denominator);
            if(x>=0){
                return (int)x;
            }else{
                return 0;
            }
        }
    }
```

```
 //n things taken q at a time where q>=n
 //we really cannot use this as we get OutOfMemoryError
early on
 //because we try to save the whole array. Instead, use
nPq_i()
 protected double[][] nPq(int nopermutations, int q){
     double hprpermutation[][]=new
 double[nopermutations][q];
     for(int column=0;column<q;column++){ // go
 through column x column
         for(int pn=0;pn<nopermutations;pn++){ // go
 through permutation x permutation
             if(column==0){
                 hprpermutation[pn][column] =  hpr[pn  %
 N];
             }else{
                 hprpermutation[pn][column] =
 hpr[(pn/(int)(Math.pow((double)N,(double)column))) % N];
             }
         }
     }
     return hprpermutation;
 }

 //n things taken q at a time where q>=n to return the
i'th item
 protected double[] nPq_i(int q, long pn){
     double hprpermutation[]=new double[q];
     int x = 0;
     for(int column=0;column<q;column++){ // go through
column x column
         if(column==0){
             x = (int)(pn  % NL);
         }else{
             x =
(int)((pn/(long)(Math.pow((double)N,(double)column))) %
NL);
         }
         hprpermutation[q-1-column] =  hpr[x];
     }
     return hprpermutation;
 }

 protected void control(){
     int counter=1;
```

```
        while(1==1){
            long passed=0;
            long nopermutations = (long)
   Math.pow((double)hpr.length,(double)counter);
            for(long pn=0;pn<nopermutations;pn++){
                double hprpermutation[]=nPq_i(counter,pn);

   passed+=(long)B(hprpermutation,usedrawdowninsteadofruin);
            }
            double result=1.0-
   (double)passed/(double)nopermutations;
            System.out.println(counter+" = "+result);
            counter++;
        }
    }
}
```

The code is presented "as-is," with no warranties whatsoever. Use it as you see fit. It is merely a bare-bones implementation of Equation (12.05). I wrote it in as generic a flavor of Java as possible, intentionally avoided using an object-oriented approach, and intentionally kept it in the lowest-common-denominator syntax across languages, so that you can transport it to other languages more easily. The code can be made *far* more efficient than what is presented here. This is presented merely to give programmers of this concept a starting reference point.

Note that the input file format must be formatted as follows: a straight ASCII text file, wherein the first line is the scenario spectrum name, the second line is the f value to be used on that scenario spectrum, and all subsequent lines are the simple stream of individual scenario outcomes. For example:

```
Coin Toss 1
.25
−1
2
```

This shows the scenario spectrum "Coin Toss 1," at an f of .25 with two outcomes, one of –1 and the other of +2.

For situations of multiple scenario spectrums, again the first line is the scenario spectrum names, comma-delimited is the second line; the respective f values, comma delimited; and each line after that represents a

simultaneous outcome for both scenario spectrums, wherein each combination of scenarios from both scenario spectrums occur.

```
Coin Toss 1,Coin Toss 2
.25,.25
2,2
2,−1
−1,2
−1,−1
```

So in this file, the first outcome sees both scenario spectrums gaining two units. The next outcome sees Coin Toss 1 gaining two units, while Coin Toss 2 loses one unit (–1). Then Coin Toss 1 loses one unit (–1) and Coin Toss 2 gains two units. For the last outcome, they both lose one unit (–1). (Thus, $n = 4$ in this file. In all data files, therefore, since the first two lines are scenario spectrum name(s) and respective f value(s), n equals the number of lines in the file minus 2.)

To this point, we have not alluded to the probabilities of the scenario outcomes. Rather, as if the scenario outcomes were like a stream of trades, or a stream of coin toss results, we have quietly assumed for simplicity's sake that there has been an equal probability of occurrence with each scenario outcome. In other words, we have been inexplicitly saying to this point that for each scenario (or individual combinations of scenarios from multiple spectrums occurring simultaneously), the probability of the kth outcome among the n^q outcomes is:

$$p_k = {}^1\!/_{n^q} \tag{12.07}$$

Usually, however, we do not have the luxury of the convenience of all scenarios having the same probability of occurrence.[6]

To address this, we return now to Equation (12.05). We will discuss first the case of a single scenario spectrum. In this case, we not only have outcomes for each scenario [which comprise the HPRs used in Equation

[6]Note, however, that if we *were* talking about scenarios made up of individual coin tosses, or of results of trading a given market system over a given day, or if we did use purely empirical data in discerning our scenario spectrums and probabilities, we could use Equation (12.07) for the said probabilities. In other words, if, say, we used the last 24 trading months and examined the prices of stock ABC, we could conceivably create a scenario spectrum of 24 bins, each with an outcome of those months, each with a probability given in (12.07).

(12.03) or (12.03a) for β], but we also have a probability of its occurrence, p.

$$RX(b,q) = \frac{\forall nPq \sum_{k=1}^{n^q} (\beta_k * p_k)}{\forall nPq \sum_{k=1}^{n^q} p_k} \tag{12.05a}$$

where: β = The value given in (12.03) or (12.02).
p_k = The probability of the kth occurrence.

For each k, this is the product of the probabilities for that k. That is, you can think of it as the horizontal product of the probabilities from 1 to q for that k. For each k, you calculate a β . Each β_k, as you can see in (12.03) or (12.03a), cycles through from $i = 1$ to q HPRs. Each HPR$_i$ has a probability associated with it ($Prob_{k,i}$). Multiplying these probabilities along as you cycle through from $i = 1$ to q in (12.03) or (12.03a) as you discern β_k will give you p_k in the single scenario case. For example, in a coin toss, where the probabilities are always .5 for each scenario, then however the permutation of scenarios in (12.03) or (12.03a), p_k will be $.5 \times .5 = .25$ when $q = 2$ in discerning β_k, for each k, it will equal $.25 \times .25 \times .25 = .015625$ when $q = 3$, ad infinitum for the single scenario set case.

$$p_k = \prod_{i=1}^{q} Prob_{k,i} \tag{12.07a}$$

To help dispel confusion, let's return to our simple single coin toss and examine the nomenclature of our variables:

- There is one scenario spectrum: $m = 1$.
- This scenario spectrum has two scenarios: $n = 2$ [per (12.06)].
- We are expanding out in this example to three sequential outcomes, $q = 3$. We traverse this, "Horizontally," as $i = 1$ to q (as in [12.02])
- Therefore we have $n^q = 2^3 = 8$ sequential outcome possibilities. We traverse this, "vertically," as $k = 1$ to n^q (as in [12.04])

As we get into multiple scenarios, calculating the individual $Prob_{k,i}$'s gets a little trickier. The matter of joint probabilities pertaining to given outcomes at i, for m spectrums was covered in Chapter 9 and the reader is referred back to that discussion for discerning $Prob_{k,i}$'s when $m > 1$.

Thus, of note, there is a probability at a particular i of the manifestations of each individual scenario occurring in m spectrums together (this is a $Prob_{k,i}$). Thus, on a particular i in multiplicative run from 1 to q, in a

particular horizontal run of k from 1 to n^q, we have a probability $Prob_{k,i}$. Now multiplying these $Prob_{k,i}$'s together in the horizontal run for i from 1 to q will give the p_k for this k.

$$\text{Prob}_{1,1} * \text{Prob}_{1,2} * \cdots * \text{Prob}_{1,q} = p_1$$
$$\cdots$$
$$\text{Prob}_{n^q,1} * \text{Prob}_{n^q,2} * \cdots * \text{Prob}_{n^q,q} = p_{n^q}$$

n.b. Now, when dependency is present in the stream of outcomes, the p_k values are necessarily affected.

For example, in the simplistic binomial outcome case of a coin toss, where I have two possible outcomes ($n = 2$), heads and tails, with outcomes $+2$ and -1, respectively, and I look at flipping the coin two times ($q = 2$), I have the following four (n^q) possible outcomes:

				p_k
Outcome 1	($k = 1$)	H	H	.25
Outcome 2	($k = 2$)	H	T	.25
Outcome 3	($k = 3$)	T	H	.25
Outcome 4	($k = 4$)	T	T	.25

Now let us assume there is perfect negative correlation involved—that is, winners always beget losers, and vice versa. In this idealized case, we then have the following:

				p_k
Outcome 1	($k = 1$)	H	H	0
Outcome 2	($k = 2$)	H	T	.5
Outcome 3	($k = 3$)	T	H	.5
Outcome 4	($k = 4$)	T	T	0

Unfortunately, when serial dependency seems to exist, it is never at such an idealized value as 1.0, as shown here. Fortunately, however, serial dependency rarely exists, and its appearance of existence in small amounts is usually, and typically, incidental, and can thus be worked with as being zero. However, if the p_k values are deemed to be more than merely "incidental," then they can, and in fact, must, be accounted for as they are used in the equations given in this chapter.

Additionally, the incorporation of rules to address dependency when it seems present, of the type like, "Don't trade after two consecutive losers, etc," could in this analysis be turned into the familiar tails, or T in the following stream:

H H T H T T H H

The dependency rules would transform the stream to:

H H T H T T H

Such a stream could therefore be incorporated into these equations, amended as such, with the same probabilities.

Note the nomenclature in (12.05a) $RX(b, q)$, referring to the fact that this equation can be used for either risk of ruin, $RR(b, q)$ or risk of drawdown, $RD(b, q)$.

Additionally, note that the denominator in this case is simply the sum of the probabilities. Typically, this should equal 1, excepting for floating point round-off error. However, this is often not the case when we get into some of the shortcut methods listed later, so (12.05a) will not be rewritten here with a denominator of 1.

The full equation, then, for determining risk of drawdown at a given q is then given as:

$$RD(b, q) = \frac{\forall nPq \sum_{k=1}^{n^q} \left(\text{int} \left(\frac{\sum_{i=1}^{q} \left(\min\left(1.0, \left(\prod_{t=0}^{i-1} HPR_t\right)\right) * HPR_i - b\right)}{\sum_{i=1}^{q} \left| \left(\min\left(1.0, \left(\prod_{t=0}^{i-1} HPR_t\right)\right) * HPR_i - b\right)\right|} \right)_k * \prod_{i=1}^{q} Prob_{k,i} \right)}{\forall nPq \sum_{k=1}^{n^q} \left(\prod_{i=1}^{q} Prob_{k,i} \right)} \tag{12.05a}$$

where: $HPR_0 = 1.0$.

$$\sum_{i=1}^{q} \left| \left(\min\left(1.0, \left(\prod_{t=0}^{i-1} HPR_t\right)\right) * HPR_i - b\right)\right| \neq 0$$

That's it. There is your equation. Solving (12.05b) will give you the probability of drawdown. Though it looks daunting, the only inputs required to calculate it are a given level of drawdown (expressed as $1 - b$; thus, if I am considering a 20% drawdown, I will use $1 - .2 = .8$ as my b value), the f values of the scenario spectrums (from which the HPRs are then derived), and the joint probabilities of the scenarios across the spectrums.

Why is (12.05b) so important? Because everything in (12.05b) you will keep constant. The only thing that will change are the f values of the components in the portfolio, the scenario spectrums from which the HPRs are derived.

Therefore, given (12.05b), one can determine the portfolio that is growth optimal within a given acceptable $RD(b)$! In other words, starting from the standpoint of "I want to have no more than an x percent probability of a

drawdown greater than $1 - b$," you can discern the portfolio that is growth optimal.

Essentially then, the new model is:

Maximize TWR where RD(b) <= an acceptable probability of hitting b. (12.08)

Also expressed as:

Maximize (9.04) where (12.05b) <= an acceptable probability of hitting b.

That is, whenever an allocation is measured in, say, the genetic algorithm for discerning if it is a new, optimal allocation mix, then it can be measured against (12.05b) given the f values of the candidate mix, the drawdown being considered as $1 - b$, to see whether $RD(b)$, as given by (12.05b) is acceptable (i.e., if $RD(b) <= x$).

Additionally, the equation can be looked at in terms of a fund as a scenario spectrum. We can use (12.05), (12.05a), and (12.05b) to determine an allocation to that specific fund in terms of maximum drawdown and maximum risk of ruin probabilities, rather than looking to discern the relative weightings within a portfolio. That is, in the former we are seeking an individual f value that will give us probabilities of drawdowns and ruin which are palatable to us and/or will determine the notional funding amount that accomplishes these tolerable values. In the latter, we are looking for a set of f values to allocate among m components within the portfolio to accomplish the same.

How many q is enough q? How elusive is that asymptote, that risk of drawdown?

In seeking the asymptote to (12.05), (12.05a), (12.05b) we seek that point where each increase in q is met with $RX(b)$ increasing by so slight a margin as to be of no consequence to our analysis. So it would appear that when $RX(b)$ for a given value of q, $RX(b,q)$ is less than some small amount, a, where we say we are done discerning where the asymptote lies—we can assume it lies "just above" $RX(b,q)$.

Yet, again refer to Figure 12.1. Note that the real-life gradations of $RX(b)$ are not necessarily smooth, but do go upward with spurious stairsteps, as it were. So it is not enough to simply say that the asymptote lies "just above" $RX(b,q)$ unless we have gone for a number of iterations, z, before q where $RX(b,q) - RX(b,q-1) <= a$.

In other words, we can say we have arrived at the asymptote, and that the asymptote lies "just above" $RX(b,q)$ when, for a given $a$and z:

$$RX(b, q) - RX(b, q-1) <= a, \text{ and } \ldots \text{ and } RX(b, q) - RX(b, q-z) <= a \quad (12.09)$$

where: $q > z$.

The problem with Equation (12.05a) or (12.05b) now [and (12.05a) or (12.05b) will give you the same answer as (12.05) when the probabilities of each kth occurrence are identical] is that it increases as q increases, increasing to an asymptote.

It is relatively easy to create a chart of the sort shown in Figure 12.1 derived from the table on page 386 to attempt to discern an asymptote when $q = 2$ as in our simple two-to-one coin-toss situation. However, when we have 26 plays—that is, when we arrive at a value of $q = 26$, then $n^q = 2^{26} = 67{,}108{,}864$ permutations. That is over 67 million β values to compute.

And that's in merely calculating the $RR(b)$ for a single coin-toss scenario spectrum! When we start getting into multiple scenario spectrums with more than two scenarios each, where n equals the results of (12.06), then clearly, computer power—speed and raw memory requirements—are vital resources in this pursuit.

Suppose I am trying to consider one scenario spectrum with, say, 10 scenarios in it. To make the pass through merely when $q = n$, I have $10 \text{ ^ } 10 = 10{,}000{,}000{,}000$ (ten billion) permutations! As we get into multiple scenario spectrums now, the problem explodes on us exponentially.

Most won't have access to the computing resources that this exercise requires for some time. However, we can implement two mathematical shortcuts here to arrive at very accurate conclusions in a mere fraction of the time, with a mere fraction of the computational requirements.

Now, can't I take a random sample of these 10 billion permutations and use that as a proxy for the full 10 billion? The answer is yes, and can be found by statistical measures used for sample size determination for binomially distributed outcomes (note that β is actually a binomial value for whether we have hit a lower absorbing barrier or not; it is either true or false).

To determine our sample size, then, from binomially distributed data, we will use Equation (12.10):

$$\left(\frac{s}{x}\right)^2 * p * (1 - p) \tag{12.10}$$

where: s = The number of sigmas (standard deviations) confidence level for an error of x.
x = The error level.
p = The probability of the null hypotheses.

That last parameter, p, is circularly annoying. If I know p, the probability of the null hypotheses, then why am I sampling to discern, in essence, p?

Note, however, that in (12.10), any deviation in p away from $p = .5$ will give a smaller answer for (12.10). Thus, a smaller sample size would be

required for a given s and x. Therefore, if we simply set p to .5, we are being conservative, and requiring that (12.10) err on the side of conservatism (i.e., as a larger sample size).

Simply put then, we need only answer for s and x. So if I want to find the sample size that would give me an error of .001, with a confidence to s standard deviations, solving for (12.10) yields the following:

$$2 \text{ sigma} = \left(\frac{2}{.001}\right)^2 * .5 * (1 - .5) = 1{,}000{,}000$$

$$3 \text{ sigma} = \left(\frac{3}{.001}\right)^2 * .5 * (1 - .5) = 2{,}250{,}000$$

$$5 \text{ sigma} = \left(\frac{5}{.001}\right)^2 * .5 * (1 - .5) = 6{,}250{,}000$$

Now the reader is likely to inquire, "Are these sample sizes independent of the actual population size?" The sample sizes for the given parameters to (12.10) will be the same regardless of whether we are trying to estimate a population of 1,000 or 10,000,000.

"So I need only do this once; I don't need to keep increasing q?"

Not so. Rather, you use (12.10) to discern the minimum sample size required at each q. You still need to subsequently increase q, and the answer [as provided by (12.05), (12.05a), or (12.05b)] will keep increasing to the asymptote. The reason you must keep increasing q is that at each q, the binomial distribution is different, as was demonstrated earlier in this chapter.

One of the key caveats in implementing Equation (12.10) is that it is provided for a "random" sample size. However, these minimum, random sample sizes provided for in (12.10) tend to be rather large. Thus, it's important to make sure, since we are generating random numbers by computer, that we are not cycling in our random numbers so soon that it will cause distortion in randomness, and that the random numbers generated be isotropically distributed.

I strongly suggest to the ambitious readers who attempt to program these concepts that they incorporate the most powerful random number generators they can. Over the years this has been something of a moving target, and, likely and hopefully will continue to be. Currently, I am partial to the Mersenne Twister algorithm.[7] You can use other random number generators, but your results will be accurate only to the extent of the randomness provided by them.

[7]Makato Matsumoto and Takuji Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," *ACM Transactions on Modeling and Computer Simulation*, Vol. 8, No. 1 (January 1998), pp. 3–30.

There are additional real-world implementation issues in terms of adding floating point numbers millions of times considering the floating point roundoff errors, and so on. Ultimately, we are trying to get a "reasonable and real-world workable" resolution of the curves for *RR* and *RD* so that we can determine their asymptotes.

This particular shortcut is invoked only if the number of permutations at a given q exceeds n^q. If not, just run all the permutations. For example, where $q = 1$, where we start, there are $10^1 = 10$ permutations. Thus, we just run all 10. At $q = 2$, we have $10^2 = 100$ permutations, and again run all permutations. However, at $10^7 = 10{,}000{,}000$, which is greater than the 6,250,000 sample size required, we would begin using the sample size when $q = 7$ in this case.

Let's look at a real-world implementation of what has been discussed thus far. Consider a single scenario spectrum with the following scenarios:

Outcome	Probability
−1889	0.015625
−1430.42	0.046875
−1295	0.015625
−750	0.0625
−450	0.125
0	0.203125
390	0.078125
800	0.328125
1150	0.0625
1830	0.046875

This is a case of a single scenario spectrum of 10 scenarios. Therefore, on our $n = q$ pass through the data (i.e., $q = 10$), we are going to have $n \wedge q$, or $10 \wedge 10 = 10{,}000{,}000{,}000$ (ten billion) permutations, as alluded to earlier.

Now, we will attempt to calculate the risk of ruin, with ruin defined as having 60% of our initial equity left.

Running these 10 billion calculations outright gives:

$$RR(.6, 10) = .1906955154$$

at an f value of .45.

Using (12.10) with $s = 5, x = .001, p = .5$, we iterate through q obtaining quite nicely, and in a tiny fraction of the time it took to actually calculate the

actual value at $RR(.6,10)$ just presented (i.e., 10 billion iterations for $q = 10$ actually versus 6,250,000! This is .000625 of the time!):

q	*RR*(.6)
1	0.015873
2	0.047367
3	0.07433
4	0.097756
5	0.118505
6	0.136475
7	0.150909
8	0.16485
9	0.178581
10	0.191146
11	0.202753
12	0.209487
13	0.21666
14	0.220812
15	0.244053
16	0.241152
17	0.257894
18	0.269569
19	0.276066
20	1

Note that at $q = 20$ we have $RR(.6) = 1$. This is merely an indication that we have overflowed the value for a long data type in Java.[8] This is still far from the asymptote.

Also note the floating point roundoff error even at $q = 1$. This value should have been 0.015625, not 0.015873.

These calculations were performed by extending the class of the previous Java program earlier in this chapter, and is included herein:

```
import java.awt.*;
import java.io.*;
import java.util.*;

public class MaxTWR4VARWithProbs extends MaxTWR4VAR{
    double probs[][];
    double probsarray[];
    double probThisB;
```

[8]Again, all of the code presented here can, even under present-day Java, be made far more efficient and robust than what is shown here. This is merely presented as a starting point for those wishing to pursue these concepts in code.

```
 public MaxTWR4VARWithProbs(String[] args){
    super(args);
 }

 public static void main(String[] args){
     MaxTWR4VARWithProbs maxTWR4VARWithProbs = new
MaxTWR4VARWithProbs(args);
  }

  protected void getinputdata(String fileName){
      String filetext = readInputFile(fileName);
      lines = getArgs(filetext,"\r\n");
      N=lines.length-2;
      NL=(long)N;
      plays=new double[N][];
      probs=new double[N][lines.length-2];
      for(int i=0;i<lines.length;i++){
          System.out.println("line "+i+" : "+lines[i]);
          if(i==0){
              msnames = getArgs(lines[i],",");
          }else if(i==1){
              f =
convertStringArrayToDouble(getArgs(lines[i],","));
          }else{

              plays[i-2]=
convertStringArrayToDouble(getArgs(lines[i],","),i-2);
          }
      }
      System.out.println("b : "+b);
      if(usedrawdowninsteadofruin){
          System.out.println("pr of : drawdown");
      }else{
          System.out.println("pr of : ruin");
      }

  }

  protected double [] convertStringArrayToDouble(String
[] s,int lineno){
      double [] d = new double[s.length];
      probs[lineno]= new double[s.length];
      for(int i = 0; i<s.length; i++){
          String ss[] = getArgs(s[i],";");
          try{
              d[i]=Double.parseDouble(ss[0]);
              probs[lineno][i]=Double.parseDouble(ss[1]);
```

```
            }catch(NumberFormatException e){
                d[i]=0.0;
                probs[lineno][i]=0.0;
            }
        }
        return d;
    }

    protected int B(double [] hprset,boolean drawdown){
        double interimHPR=1.0;
        double previnterimHPR=1.0;
        double numerator=0.0;
        double denominator=0.0;
        probThisB=1.0;
        for(int i=0;i<hprset.length;i++){
            double useinvalue = previnterimHPR;
            if(drawdown && previnterimHPR>1.0)
                useinvalue = 1.0;

            interimHPR = useinvalue ×  hprset[i];
            //interimHPR = previnterimHPR ×  hprset[i];
            double value = interimHPR - b;
            numerator += value;
            denominator += Math.abs(value);
            previnterimHPR = interimHPR;
            probThisB *= probsarray[i];
        }
        if(denominator==0.0){
            return 0;
        }else{
            double x = (numerator/denominator);
            if(x>=0){
                return (int)x;
            }else{
                return 0;
            }
        }
    }

   //n things taken q at a time where q>=n to return the
  i'th item
    protected double[] nPq_i(int q, long pn){
        double hprpermutation[]=new double[q];
        probsarray=new double[q];
        int x = 0;
        for(int column=0;column<q;column++){ // go through
column x column
```

```
          if(column==0){
              x = (int)(pn % NL);
          }else{
                x =
(int)((pn/(long)(Math.pow((double)N,(double)column))) %
NL);
          }
          int a = q-1-column;
          hprpermutation[a] = hpr[x];
          probsarray[a] = probs[x][0];//it's zero here
because we are only figuring one MS
     }
     return hprpermutation;
 }


  protected void control(){
       double sigmas = 5.0;
       double errorsize = .001;
       double samplesize = Math.pow(sigmas/errorsize,2.0)
x .25;
       long samplesizeL = (long)(samplesize+.5);
       int counter=1;
       RalphVince.Math.MersenneTwisterFast generator = new
RalphVince.Math.MersenneTwisterFast(System.currentTime
Millis());
     java.util.Random random = new java.util.Random();
     while(1==1){
           long permutationcount = 0L;
           double passed=0.0;
           double sumOfProbs=0.0;
           long nopermutations = (long)
Math.pow((double)hpr.length,(double)counter);
         if(nopermutations<(long)samplesize){
              for(long pn=0;pn<nopermutations;pn++){
                  double
hprpermutation[]=nPq_i(counter,pn);
                double theB =
(double)B(hprpermutation,usedrawdowninsteadofruin);
                  if(theB>0.0){
                      theB *= probThisB;
                      passed += theB;
                  }
                  sumOfProbs += probThisB;
                  permutationcount++;
              }
           }else{
```

```
                do{
                    generator.setSeed(random.nextLong());
                    long
 pn=(long)(generator.nextDouble()*(double)nopermutations);
                    double
 hprpermutation[]=nPq_i(counter,pn);
                  double   theB =
 (double)B(hprpermutation,usedrawdowninsteadofruin);
                    if(theB>0.0){
                       theB *= probThisB;
                       passed += theB;
                    }
                    sumOfProbs += probThisB;
                    permutationcount++;

                }while(permutationcount<samplesizeL);
            }
            double result=1.0-passed/sumOfProbs;
            System.out.println(counter+" = "+result);
            counter++;
        }
    }
}
```

Unlike the previous code provided, this code class works only with one market system, and the format for the input file differs from the first in that in this class, each line from the third line on is a semicolon-delimited value pair of outcome;probability.

Thus, the input file in this real-world example appears as:

```
Real-world example file of a single scenario spectrum
.45
-1889;0.015625
-1430.42;0.046875
-1295;0.015625
-750;0.0625
-450;0.125
0;0.203125
390;0.078125
800;0.328125
1150;0.0625
1830;0.046875
```

The technique of using a random sample gets our first few values for the line of RX to q up and running with very good estimates in short order.

With the second technique, to be presented now, we can extrapolate out that line and hence seek its horizontal asymptote. Fortunately, lines derived from the Equations (12.05), (12.05a), and (12.05b) do possess an asymptote and are of the form:

$$RX'(b, q) = \text{asymptote-variableA} * \text{EXP}(-\text{variable}B * q) \qquad (12.11)$$

$RX'(b,q)$ will be the surrogate point, the value along the y axis for a given q along the x axis in the Cartesian plane.

We can use equation (12.11) as a surrogate for the actual calculations in (12.05), (12.05a), or (12.05b) when q gets too computationally expensive.

To do this, we need only know three values: the asymptote, variable$A$, and variable$B$.

We can find these values by any method of mathematical minimization whereby we minimize the squares of the differences between the observed values and the values given by (12.11). Those values with the minimum sum of the differences squared are those values that best fit this line, this proxy of actual $RX(b,q)$ values when q is too computationally expensive.

The process is relatively simple. We take those values we were able to calculate for $RX(b,q)$. For each of these values, we compare corresponding points derived from (12.11) and square the differences between the two. We then sum the squares.

Thus, we have a sum of the squared differences of our points to (12.11) for a given (asymptote, variable$A$, variable$B$). Proceeding with a mathematical minimization routine (Powell's, Downhill Simplex, even the genetic algorithm, though this will be far from the most efficient means— for a list and detailed explanation of these methods, see "Numerical Recipes,"[9] Press et al.) we arrive at that set of variable values that minimizes the sum of the differences squared between the observed points and their corresponding points as given by (12.11).

Returning, for example, to our two-to-one coin toss, we had calculated by equation (12.05) those $RR(.6)$ values, and these were given in Table 4.2. Here, using Microsoft Excel's Solver function, we can calculate the parameters in (12.11) that yield the best fit:

asymptote	0.48406
variableA	0.37418
variableB	0.137892

[9]Press, William H.; Flannery, Brian P.; Teukolsky, Saul A.; and Vetterling, William T., *Numerical Recipes: The Art of Scientific Computing*, (New York: Cambridge University Press, 1986).

These values given by (12.11) are shown in the table below.

Play#	Observed (12.05)	Calculated (12.10)
2	0.25	0.200066
3	0.25	0.236646
4	0.25	0.268515
5	0.3125	0.296278
6	0.3125	0.320466
7	0.367188	0.341538
8	0.367188	0.359896
9	0.367188	0.375889
10	0.389648	0.389822
11	0.389648	0.40196
12	0.413818	0.412535
13	0.413818	0.421748
14	0.436829	0.429774
15	0.436829	0.436767
16	0.436829	0.442858
17	0.447441	0.448165
18	0.447441	0.452789
19	0.459791	0.456817
20	0.459791	0.460326
21	0.459791	0.463383
22	0.466089	0.466046
23	0.466089	0.468367
24	0.47383	0.470388
25	0.47383	0.472149
26	0.482092	0.473683

This fitted line now, Equation (12.10), is shown superimposed as the solid line over Figure 12.1, now as Figure 12.2.

Now that we have our three parameters, I can determine for, say, a q of 300, by plugging in these values into (12.10), that my risk of ruin [$RR(.6)$] is .484059843.

At a q of 4,000 I arrive at nearly the same number. Obviously, the horizontal asymptote is very much in this vicinity.

The asymptote of such a line is determined, as pointed out earlier by (12.09), since the line given by (12.10) is a smooth one.

Let's go back to our real-world example now, the single scenario set of 10 scenarios. Fitting to our earlier case of a single scenario set with 10 scenarios, whereby we were able to calculate the $RR(.6)$ values for $q = 1 \ldots 19$,

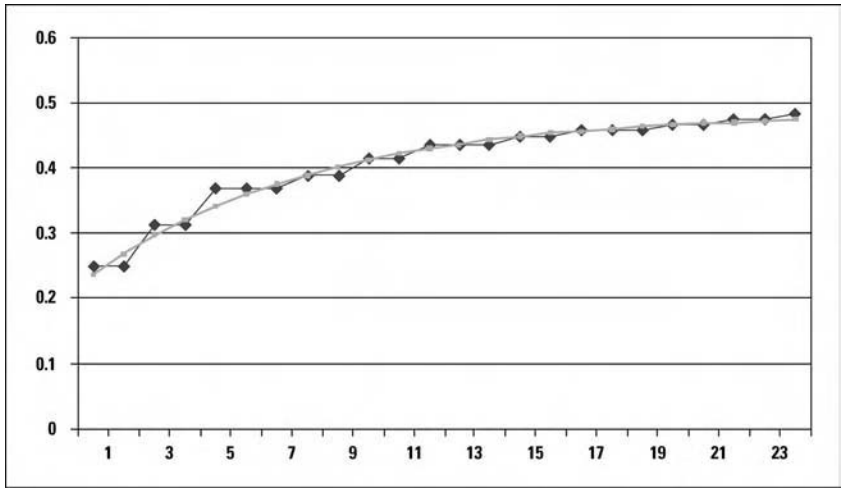

FIGURE 12.2 *RR*(.6) observed and calculated for two-to-one coin toss $f = .25$

by taking 6,250,000 samples for each q (beyond $q = 6$) , and using these 10 data points ($q = 1 \ldots 19$) as input to find those values of the parameters in (12.11) that minimize the sum of the squares of the differences between the answers given by those parameters in (12.11) and the actual values we got [by estimating the actual values using (12.10)], gives us the corresponding best fit parameters for (12.11) as follows:

asymptote = 0.397758
exponent = 0.057114
coefficient = 0.371217

The data points and corresponding function (12.11) then appear graphically as Figure 12.3.

And, if we extend this out to see the asymptote in the function, we can compress the graphic as shown in Figure 12.4.

Using these two shortcuts allow us to accurately estimate what the function for $RX(\,)$ is, and discern where the asymptote is, as well as how many q—which can be thought of as a surrogate for time—out it is.

Now, if you are trying to fit (12.10) to a risk of ruin, $RR(b)$, you will fit to find the three parameters that give the best line, as we have done here.

However, if you are trying to fit to risk of drawdown, $RD(b)$, you will only fit for variable A and variable B. You will *not* fit for the asymptote. Instead, you will assign a value of 1.0 to the asymptote, and fit the other two parameters from there.

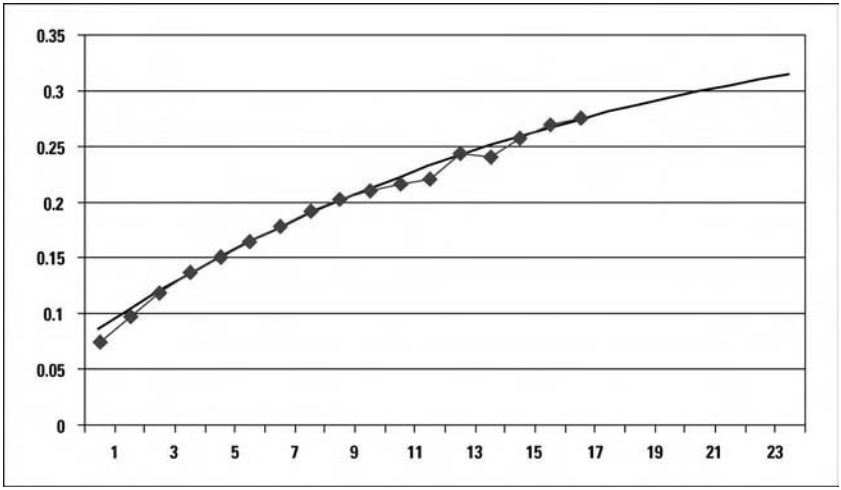

FIGURE 12.3 *RR*(.6) for real-world example at $f = .45$

To confirm the reader's burgeoning uneasiness at this point, consider the following:

In the long-run sense, the probability of hitting a drawdown (of *any* given magnitude, b) approaches 1, approaches *certainty* as you continue to trade (i.e., as q increases).

$$\lim_{q \to \infty} RD(b, q) = 1.0 \tag{12.12}$$

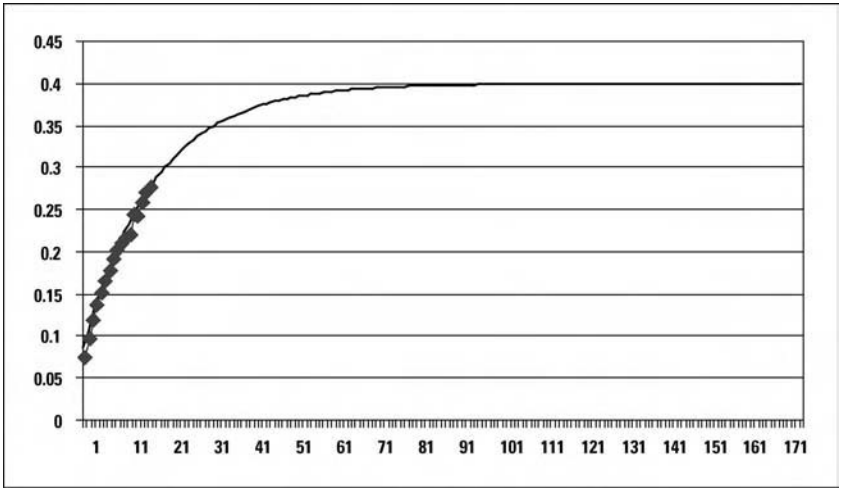

FIGURE 12.4 Figure 12.3 larger field-of-view

This is not as damning a statement as it appears on first reading. Consider the real-world example just alluded to wherein $RR(.6) = 0.397758$. Since the probability of hitting a drawdown of any given magnitude (let's say, a 99% drawdown, for argument sake) approaches 1 as q approaches infinity, yet there is only a roughly 40% chance of dropping back to roughly 60% of starting equity, we can only conclude that so many q have transpired so as to cause the account to have grown by such an amount that a 99% drawdown still leaves 60% of initial capital.

What we can know, and use, is that (12.05b) can give us a probability of drawdown for a given q. We can use it to know, for instance, what the probability of drawdown is over, say, the next quarter.

Further, since, we have a geometric mean HPR for each value of (12.05b), we can determine what T we are looking at to reach a specified growth.

$$T = \log_G \text{target} \tag{5.07b}$$

where: target = The target TWR.
G = The geometric mean HPR corresponding to the allocation set used in (12.05b).

Thus, for example, if my target is a 50% return (i.e., target TWR = 1.5) and my geometric mean HPR from the allocation set I will use in (12.05b) is 1.1, then I will expect it to take T periods, on average, to reach my target TWR:

$$T = \log_{1.1} 1.5 = 4.254164$$

So I would want to consider the $RD(b, 4.254164)$ in this case to be below my threshold probability of such a drawdown.

Notice that we are now considering a risk of drawdown (or ruin) versus that of hitting an upper barrier [i.e., target TWR, or u from (12.01)] . Deriving T from (5.07b) to use as input to (12.05) is akin to using Feller's classical ruin given in (12.01) only for the more complex case of:

1. A lower barrier, which is not simply just zero.
2. For multiple scenarios, not just the simple binomial gambling sense (of two scenarios).
3. These multiple scenarios are from multiple scenario spectrums, with outcomes occurring simultaneously, with potentially complicated joint probabilities.
4. More importantly, we are dealing here with geometric growth, not the simple case in Feller where a gambler wins or loses a constant unit with either outcome.

Such analysis—determining T as either the horizon over the next important period (be it a quarter, a year, etc.), or backing into it as the expected number of plays to reach a given target, is how we can determine the portfolio allocation that is growth optimal while remaining within the constraints of an acceptable level of a given drawdown over such a period.

In other words, if we incorporate the concepts detailed in this chapter, we can see that the terrain in leverage space is *pock-marked*, has holes in it, where we cannot reside. These holes are determined by the utility preference pertaining to an unacceptable probability of an unacceptable drawdown.[10] We seek the highest point where the surface has not been removed under our feet via the analysis of this chapter.

The process detailed in this chapter allows you to maximize returns for a given probability of seeing a given level of drawdown over a given period—which *is* risk. This is something that has either been practiced by intuition by others, with varying degrees of success, or practiced with a different metric for risk other than drawdown or risk of ruin—often alluded to as *value at risk*.

Essentially, by seeking that highest point (altitude determined as a portfolio's geometric mean HPR or TWR) in the $n + 1$ dimensional landscape of n components, one can mark off those areas within the landscape that cannot be considered for optimal candidates as those areas where the probability of risk of ruin or drawdown to a certain point is exceeded.

[10]Note that as one nears the $f_1 = 0 \ldots . f_n = 0$ point, as described at the end of Chapter 10, the likelihood of not being at a hole in the landscape becomes assured, without going through the analysis outlined in this chapter. However, that is a poor surrogate for *not* going through this analysis, as one would pay the consequences for deviating far left on all axes in leverage space.