Improved techniques for using Monte Carlo in VaR estimation

4 July 2001

Abstract

This paper is about the class of problems where a large number of VaR estimates, using Monte Carlo simulation, are required for different portfolios, drawn from a fixed universe of products with a fixed data generating process. We focus on the matrix–vector product which dominates the computational cost of this problem. We develop a series of results based on deriving bounds for different matrix norms, which allow us to eliminate rows from the matrix–vector product. Using these results, we propose fourteen alternative new algorithms. We compare them in a numerical experiment, and find that the best of these algorithms offers between 80% to 90% gains in speed. These algorithms make it quite feasible to do intra–day, real–time VaR calculations, using full Monte Carlo, on the derivatives exchanges with the highest trading intensities presently known, while requiring modest computer hardware.

Contents

Algorithm 1 Simplest algorithm for VaR estimation using Monte Carlo

```
Input:  Portfolio $w \in \mathbb{R}^M$.
Output: VaR of portfolio $w$.
Procedure:
    1. Let $V_t$ be the value of the portfolio at the close of the previous day.
    2. Simulate $N$ draws $r_{n,t+1}$, from the joint distribution of returns on the underlyings.
    3. At each draw $r_{n,t+1}$, apply theoretical valuation formulas to obtain $V_{n,t+1}$, the value
       of the portfolio $w$ if prices changed by $r_{n,t+1}$. This gives draws from the distribution
       of the change in value of the portfolio $L_{n,t+1} = V_{n,t+1} - V_t$.
    4. Sort the $N$ values for $L_{n,t+1}$, and read off the percentile value, which is the desired
       VaR.
```

1 The problem

Value at Risk has become an important strategy for estimating capital requirements and performing risk management functions (Jorion 2000). We focus on the general case, where nonlinear products such as options may be present in the portfolio. There is a wide range of techniques which can be used in estimating VaR for nonlinear portfolios, ranging from crude "strategy–based rules", to linear or quadratic approximations, to Monte Carlo (Estrella et al. 1994). When precise estimates of VaR are desired for nonlinear portfolios, Monte Carlo procedures are favoured.

Consider the problem of estimating VaR on date t, on a one–day horizon at a 99% level of significance. The simplest Monte Carlo procedure (Algorithm 1) consists of simulating N draws from the joint distribution of underlying asset prices on date $t + 1$. At each of these draws, theoretical models, such as the Black–Scholes formula, are applied to reprice the products held in the portfolio. This gives N draws from the distribution of the one–day change in the dollar value of the portfolio. The VaR at a 99% level is estimated by reading off element $N/100$ after sorting the N different draws from the one–day change.

The appeal of the Monte Carlo procedure for VaR estimation lies in its generality and accuracy. It requires no simplifying assumptions about the joint distribution of the underlyings. It requires no approximations in calculating the prices of derivatives. It makes no assumptions about the probability distribution of the one–day change in price of derivatives, or of the portfolio. However, it involves considerable computational expenses: (a) in obtaining N draws from the joint distribution of the underlyings, (b) in repricing all products at each of these draws, and in then computing the VaR.[1] This computational cost has been a barrier limiting its application into risk containment problems in the real world.

In this paper, we focus on one special case: where *a large number of different VaR problems need be solved for different portfolios, for a fixed universe of assets with a fixed data generating process.* We offer ideas which yield significant improvements upon the computational cost faced in this special case.

[1]In this paper, we work with the simple case where the theoretical price of options, under each simulated scenario, is calculated using the closed–form Black/Scholes formula. However, our results are completely general, and apply regardless of the method use for pricing derivatives at each of the N draws.

Table 1 Statistical precision and computational cost associated with alternative choices of N in Algorithm 1

This table shows results obtained with Algorithm 1 for one simple portfolio: a long position on one futures ($T = 1$), one call option ($X = 1100, T = 1$) and one put option ($X = 1100, T = 1$) when $S = 1000$. The correct answer is 37.60427. We show the standard deviation of the Monte Carlo estimator of VaR obtained for different values of N, and the computational cost (in milliseconds) faced in doing this on a Pentium II processor at 350 Mhz. The fastest CPUs in the world today are no more than ten times faster than this.

N	σ	Cost (ms)
1000	1.79579	108.6
10000	0.5585	1128.4
100000	0.1766	11296.3

This is an important special case in the real–world. One example of its application lies in the risk–containment problem of the futures clearing corporation. Consider trades that take place on date $t + 1$. For all trades that take place within the trading day, futures clearing corporations typically require an initial margin reflecting the losses which can take place from the closing price of date t to the closing price of date $t + 1$.

The initial margin of the futures clearing corporation is best estimated using VaR. Indeed, Culp et al. (1998) observe that the initial margin of the futures clearing corporation was the first use of a notion like VaR. However futures exchanges in the real world today use a variety of inaccurate approximations to VaR in computing their initial margin requirements. Margin calculations at the Chicago Mercantile Exchange (CME) are done using the 'Standard Portfolio Analysis' (SPAN) system, and margin calculations at the Chicago Board Options Exchange (CBOE) use the 'Theoretical Intermarket Margining System' (TIMS). These systems can be visualised as producing approximations of VaR using ideas dating to the late 1970s and early 1980s.

Computational constraints are one important factor in explaining the simplifications which have gone into systems such as SPAN or TIMS. Every time a trade takes place, the positions of two economic agents are updated, and two VaR computations are required. The most active futures exchanges in the world today experience roughly 1,000,000 trades in around 20,000 seconds. This requires 100 VaR computations per second, on average. Given the unevenness of trading intensity in the day, this easily maps to a *peak* requirement of 500 VaR computations per second, or a VaR computation in two milliseconds. Simple summary statistics about the statistical precision and computation cost associated with alternative choices of N in Algorithm 1, for one plausible portfolio, are shown in Table 1. Most clearing corporations would be unwilling to accept a standard deviation in this VaR problem of above 0.19, since this turns out to yield an error of below 1% of the VaR with a 95% probability. This would require $N = 10^5$ or so, which costs 11.3 seconds per VaR calculation using Algorithm 1. This is around 5,000 times larger than the goal of making one VaR calculation in two milliseconds.

Apart from the risk containment problem of the futures clearing corporation, other situations where this problem arises include a financial institution with a large number of dealers or an Internet brokerage establishment with a large number of day traders. In each of these cases, the universe of traded products stays fixed in the day, and a large number of VaR computations are required intra–day in doing risk containment.

2 Notation and definitions

A summary of important symbols used is also given in Appendix A. Suppose there are T different underlyings. We are concerned with VaR calculations for a portfolio consisting of M products. These products could be linear or nonlinear products defined off these T underlyings without any restrictions. The portfolio is a vector $w \in \mathbb{R}^M$, where a component w_m denotes the number of securities held of product m. Let V_0 be the value of the portfolio on the previous day; our goal is to measure the VaR at a P level of significance, on a one–day horizon.

The Monte Carlo estimation is based on simulating random vectors from the joint distribution $f()$ of the underlyings $(r_1, r_2, \ldots, r_T)$. This simulation yields random vectors $r \in \mathbb{R}^T$. Our approach is completely general insofar as models of the returns process are concerned. The returns can be multivariate normal with constant or time-varying covariance matrices; the returns can be drawn from a distribution with fat tails; returns can be non-parametrically simulated from historical experience; etc.

For each product m, a valuation function $v(m, r)$ maps the returns outcome r into the value of a product, while $v_o(m)$ is the value of product m the previous day. Computing the valuation function would involve theoretical valuation using the cost of carry model for futures prices, Black–Scholes or other option pricing techniques for options, or more complex strategies for derivatives which are based on multiple underlyings (e.g. index futures expressed in a foreign currency).

Suppose a Monte Carlo simulation is based on N draws $r_n \sim f()$. Let the loss matrix L be an $N \times M$ matrix where $L_{nm} = v_o(m) - v(m, r_n)$. The $N \times 1$ vector $b = Lw$ is a set of T draws from the overnight loss on a portfolio w, with a positive component of b indicating a loss, and a negative component a profit. Let V_R be the order statistic estimated from the values of b: when we desire a 1% VaR off $N = 15000$ elements in b, we seek the $p = PN = 150^{\text{th}}$ largest component of b. The desired VaR estimate is V_R.

Shah et al. (2000) observe that within one day, the universe of products stays fixed and the data generating process stays fixed, hence the L matrix does not need to be re–evaluated. It can be computed once and held fixed. The VaR computation, then, only requires the matrix multiply $b = Lw$, and evaluation of the order statistic off b. They also observe that the naive approach (Algorithm 1) would involving sorting b and then finding the desired value for a time complexity of $O(N \log(N))$. However, this value can actually be found in $O(N)$ time using an order statistic finding (selection) algorithm. They obtain major gains by applying these two ideas. Once these ideas are in hand, their computational cost is dominated by the matrix–vector product Lw.

The VaR Decision Problem. There are situations where knowing VaR is in itself important. However, for the risk containment of the futures clearing corporation, it is more

Algorithm 2 Conventional technique for determining if the VaR exceeds collateral

```
Input: $L \in \mathbb{R}^{N \times M}$, $w \in \mathbb{R}^M$ and $V_c > 0$.
Output: Answer to the decision problem: Yes or No.
Procedure:
    1. Compute $b = Lw$ in $O(NM)$ time.
    2. Determine $V_R$, the $p^{\text{th}}$ largest component of $b$ in $O(N)$ time using the linear time
       algorithm for determining the order statistic.
    3. If $V_R > V_c$, then the answer is No. Otherwise, the answer is Yes.
```

important to *check whether the magnitude of VaR exceeds collateral.* We call this the "VaR decision problem":

> Given the loss matrix L, the collateral value V_c, and the weight vector w, is $V_R > V_c$, where V_R is the $p = PN^{\text{th}}$ largest component of Lw?

This paper focuses on solving the VaR decision problem, while potentially avoiding the calculation of VaR itself.

3 Solving the VaR decision problem

The conventional method for solving the VaR decision problem is to actually evaluate VaR(w) and then check if it exceeds V_c as shown in Algorithm 2. In this, the major cost of computation is the evaluation of the matrix–vector product Lw. This takes $O(NM)$ time. For a typical case, with $N = 15,000$ and $M = 50$, this computation requires roughly 1,500,000 floating point operations. As with Shah et al. (2000), using an $O(N)$ algorithm to lookup the order statistic from the b vector, we get a total complexity of $O(NM)$. The matrix-vector multiplication is the dominant factor in this.

This paper is based on two observations:

- The matrix L is constant throughout the day. We can therefore afford to spend a large amount of computational effort in analysing it initially, if this will result in answering the decision problem quickly when presented with a weight vector.
- If it becomes possible to answer the VaR decision problem without going through the evaluation of Lw, it would offer major gains.

Our strategy is based on reducing the amount of computation using the following basic paradigm.

Upper bound on V_R: Norms, or bounds on some norms, of b can often be computed much faster than the vector b itself. For example, the well-known inequality $\|b\| = \|Lw\| \leq \|L\|\|w\|$ can be used to bound $\|b\|$ under the L_1, L_2, and L_∞ norms easily. These computations take only $O(M)$ time, since $\|L\|$ can be pre-computed, and only $\|w\|$ needs to be determined each time. In fact, we can even find the exact value of the L_2 norm, if the bounds do not prove useful. Let the singular value decomposition of L be given by $L = UDV$. Then $\|b\|_2 = \|DVw\|_2$, since U is orthonormal. Since DV has only at most M non-zero rows,

Algorithm 3 Computation of $\sum b_i$ in $O(M)$ time.

```
Input: $L \in \mathbb{R}^{N \times M}$ and $w \in \mathbb{R}^M$.
Output: $S = \sum_{i=1}^{N} b_i$, where $b = Lw$.
Procedure:
    1. Note: $\sum b_i = \sum_i \sum_j L_{ij} w_j = \sum_j w_j \sum_i L_{ij}$
    2. Initialization: We precompute $S_j = \sum_i L_{ij}$ for each $1 \le j \le M$ once and store these values.
    3. Computation: Given a vector $w$, we compute $S = \sum_{i=1}^{M} w_i S_i$ in $O(M)$ time.
```

$\|b\|_2$ can be computed in $O(M^2)$ time, which is much less than $O(NM)$ required for the computation of b, since $M \ll N$.

The norms, in turn, place upper bounds on the magnitude of V_R. For example, if $P < 0.5$, then we obtain trivial bounds on V_R as $\|b\|_1/p$, $\|b\|_2/\sqrt{p}$, and $\|b\|_\infty$ using each norm independently. However, we can use combined information on these norms, and other information, to obtain better bounds. For example, we can determine $\sum b_i$ in $O(M)$ time as shown in Algorithm 3. This can be combined with the value of the L_2 norm to give a tighter bound, as shown later. All the bounds that we use are better than the trivial ones mentioned above.

At this point, we note that the algorithms we discuss generally have two phases. In the first phase, *initialization*, we precompute certain quantities that depend on the loss matrix L alone. We do not consider the cost of such computation, as long as they are feasible, since they are done once, at the beginning of the day. In the second phase, we perform calculations when presented with a portfolio and the collateral. We are primarily concerned with ensuring the efficiency of these computations.

We answer the decision problem using a three-step procedure. (i) We compute an upper bound $\hat{V}_R$ on the value of V_R. If $\hat{V}_R \le V_c$, then the answer to the decision problem is No. If $\hat{V}_R > V_c$, then the answer could be either a Yes or a No. Thus the upper bounds eliminate calculations only for situations where the VaR is not exceeded. (ii) In section 8, we give a simple heuristic which enables us to often reduce the computational effort when the VaR is exceeded. We try this heuristic if the first step fails to give an answer. This second step can give an answer Yes, or no answer at all. (iii) If this step too fails to give an answer, then we do the entire VaR computation the conventional way (though Algorithm 12 presents a slight improvement over this).

This solution strategy always gives the same answer as the conventional technique. If either of the first two steps succeeds in giving an answer, then we have a large saving in time. If not, we solve the problem the conventional way, but have "wasted" time in the first two steps. This strategy will be effective if the number of times we save on the computational effort by answering the question in the first two steps makes up for the loss in time when those steps are unsuccessful. In practice, the first two steps are sufficiently fast that the overhead associated with using them is small.

In the next few section, we derive bounds using different norms, and then present a few other heuristics too. We then test each bound or heuristic independently to determine their effectiveness, and finally combine the best ones to come up with a composite algorithm. In

Algorithm 4 Computation of the L_2 norm of b, or its square, in $O(M^2)$ time.

```
Input:  $L \in \mathbb{R}^{N\times M}$ and $w \in \mathbb{R}^M$.
Output:  $W = b^T b = \sum_{i=1}^N b_i^2$, where $b = Lw$.
Procedure:
    • Initialization:
        1. Compute the singular value decomposition of $L$: $L = UDV$.
        2. Precompute $DV$, and store the (at most) $M$ non-zero rows in the $M \times M$ matrix
           $D_v$.
    • Computation:
        1. Compute the matrix-vector product $\acute{w} = D_v w$ in $O(M^2)$ time.
        2. Compute $W = \sum_{i=1}^N \acute{w}_i^2$, the square of the $L_2$ norm, in $O(M)$ time.
    • The $L_2$ norm is easily computed as $\sqrt{W}$, but we really need only $W$ in our bounds.
    • $W$ is also the square of the the $L_2$ norm of $b = Lw$, since $U$ is orthogonal (Heath
      1996).
```

the rest of the paper, we assume that $P \leq 0.5$, which is a reasonable assumption in any VaR computation.

4 L_2 norm bound

We divide this procedure into two steps. First we determine the L_2 norm of $b = Lw$. We then determine an upper bound on V_R, given this norm and the sum of the components of b.

Determining the L_2 norm of b is easy, once we precompute the singular value decomposition of L, and is described in Algorithm 4.

We can also compute the sum S of the components of b as shown in Algorithm 3. We next use S, and the square of b's L_2 norm, W, to derive an upper bound on V_R.

Theorem 1 *Given that a vector b is in $\mathbb{R}^N$, p an integer in $[1, N/2]$, the p^{th} largest component of b, say V_R, is bounded by the following quantity:*

$$V_R \leq \frac{1}{N}\left(S + \sqrt{\frac{(N-p)}{p}(NW - S^2)}\right) \tag{1}$$

where $S = \sum_{i=1}^N b_i$ and $W = b^T b = \sum_{i=1}^N b_i^2$.

Algorithm 5 solves the decision problem through the computation of an upper bound on V_R using this L_2 norm.

Algorithm 5 Computation of an upper bound on the p^{th} largest component of Lw using the L_2 norm.

Input: $L \in \mathbb{R}^{N \times M}$ and $w \in \mathbb{R}^M$.
Output: An upper bound $\hat{V}_R$ on the p^{th} largest component of Lw.
Procedure:

1. Compute $S = \sum_i b_i$ in $O(M)$ using Algorithm 3.
2. Compute $W = b^T b$ in $O(M^2)$ time using Algorithm 4.
3. Compute $\hat{V}_R = \frac{1}{N}\left(S + \sqrt{\frac{(N-p)}{p}(NW - S^2)}\right)$ in constant time.

Note: If $\hat{V}_R \leq V_c$, then the answer to the decision problem is No. Otherwise, the answer is not determined by this bound. This procedure takes $O(M^2)$ time. If it fails to give an answer, then we fall back upon the conventional procedure of Algorithm 2.

5 L_1 norm bound

We next determine a bound for the VaR using the L_1 norm. It is easy to obtain the following upper bound on the p^{th} largest component: $V_R \leq \frac{1}{p}(\|b\|_1)$. However, unlike with the L_2 norm, there is no linear transform from the N dimensional domain space to the smaller M dimensional range space that always preserves the L_1 norm. Thus determining the L_1 norm accurately and fast enough for our purpose is generally not feasible. The crude bounds such as $\|Lw\| \leq \|L\|\|w\|$ are not very effective. Therefore we derive a more effective bound below.

The intuition behind the proof is as follows. If we are given just the norm of b, then, for the p th largest component V_R to be as large as possible, all the smaller components must be zero and the other components equal to V_R. Such an approach yields the bound: $V_R \leq \frac{1}{p}(\|b\|_1)$. However, if we are given addition information, such as the sum of the components of b, then it may turn out that such as extreme case (all the smaller components being equal to zero) is not feasible, and lemma 2 gives the bound when the norm and the sum are known. Unlike in the case of the L_2 norm, the exact L_1 norm cannot be found fast. Therefore we use bounds on the norm. If we add identical constants to each component of b, then it is possible to get a better bound on V_R. Corollary 1 gives the bound for this case, while lemma 1 gives the optimal value of the constant to make the bound tight. This result cannot be used directly, since it requires knowledge of V_R, which is the quantity that we desire to bound! Instead, in theorem 2, we develop a method of adding constants to b indirectly by modifying the loss matrix. We then derive a bound for V_R involving the unknown constant, and then we choose the constant using the results of the two lemmas mentioned above. This gives a bound on V_R, which may not be optimal. But it proves to be good in practice, and is much better than simple bounds, such as those from lemma 2.

Lemma 1 *Let $b \in \mathbb{R}^N$ and $\vec{k}$ be the vector in $\mathbb{R}^N$ with $\vec{k}_i = k$. Indices i are chosen such that $b_{i_1} \leq b_{i_2} \leq \ldots \leq b_{i_N}$. Then $X(k) = \|b + \vec{k}\|_1 + (N - 2p)k$ attains its global minimum when $-b_{i_{N-p+1}} \leq k \leq -b_{i_{N-p}}$ and the minimum value is*

$$\sum_{j=N-p+1}^{N} b_{i_j} - \sum_{j=1}^{N-p} b_{i_j} \tag{2}$$

This lemma implies that the minimum value of $\|b+\vec{k}\|_1 + (N-2p)k$ is the difference between the sum of the p largest components of b and its $(N-p)$ smallest components.

Lemma 2 *Given that $b \in \mathrm{I\!R}^N$ and the values of $S = \sum_{i=1}^N b_i$ and $\dot{W} = \|b\|_1$, an upper bound on the p^{th} largest component of b is given by*

$$V_R \leq \frac{1}{2p}(\dot{W} + S)$$

Using the results above, it is also easy to show that the bound is tight, if we are given just the values S, $\dot{W}$, N, and p. We should also note that we cannot compute the L_1 norm quickly. But an upper bound on the L_1 norm can be used instead of the exact value of that norm. Of course, the bound will be less tight, though still valid.

Corollary 1 *Given that $b \in \mathrm{I\!R}^N$, $\vec{k}$ the vector in $\mathrm{I\!R}^N$ with $\vec{k}_i = k$ for any k, and $S = \sum_{i=1}^N b_i$. Then an upper bound on the p^{th} largest component of b is given by*

$$V_R \leq \frac{1}{2p}(\|b+\vec{k}\|_1 + S + (N-2p)k)$$

Theorem 2 *Let $L \in \mathrm{I\!R}^{N\times M}$, L^j_{lp} the sum of the p largest components of column j of that matrix, L^j_{sp} the sum of the $N-p$ smallest components of column j, L^j_{ln} the sum of the p smallest components of column j, and L^j_{sn} the sum of the $N-p$ largest components of column j. Let $w \in \mathrm{I\!R}^N$, $S = \sum_{i=1}^N (Lw)_i$. Then an upper bound on the p^{th} largest component of Lw is given by*

$$V_R \leq \frac{1}{2p}\left[S + \sum_{i=1}^{N} \tilde{w}_i(L^i_{lp} - L^i_{sp}) - \sum_{i=1}^{N} \hat{w}_i(L^i_{ln} - L^i_{sn})\right]$$

where $\tilde{w}$ is given by

$$\tilde{w}_i = w_i, w_i \geq 0;\ \ \tilde{w}_i = 0,\ \text{otherwise} \tag{3}$$

and $\hat{w}$ by $\hat{w} = \tilde{w} - w$

Remark
Note that $\tilde{w}$ and $\hat{w}$ always have non-negative components. Furthermore, it is easy to verify that

$$Lw = L\tilde{w} - L\hat{w}$$

Algorithm 6 Computation of an upper bound on the p^{th} largest component of Lw using the L_1 norm.

```
Input: $L \in \mathbb{R}^{N \times M}$ and $w \in \mathbb{R}^M$.
Output: An upper bound $\hat{V}_R$ on the $p^{\text{th}}$largest component of $Lw$.
Procedure:
    • Initialization: These steps are only carried out once.
        1. Compute $L^j_{lp}$ = sum of the $p$ largest components of column $j$ of $L$, for each $j$, $1 \leq j \leq M$.
        2. Compute $L^j_{sp}$ = sum of the $N - p$ smallest components of column $j$ of $L$, for each $j$, $1 \leq j \leq M$.
        3. Compute $L^j_{lm}$ = sum of the $p$ smallest components of column $j$ of $L$, for each $j$, $1 \leq j \leq M$.
        4. Compute $L^j_{sm}$ = sum of the $N-p$ largest components of column $j$ of $L$, for each $j$, $1 \leq j \leq M$.
    • The following steps are carried out for the $w$ that is given.
        1. Compute $S = \sum_i b_i$ in $O(M)$ using Algorithm 3.
        2. Compute $\tilde{w}$ as defined by equation 3 in $O(M)$ time.
        3. Compute $\hat{w} = \tilde{w} - w$ in $O(M)$ time.
        4. Compute
           $$\hat{V}_R = \frac{1}{2p}\left[S + \sum_{i=1}^{N} \tilde{w}_i (L^i_{lp} - L^i_{sp}) - \sum_{i=1}^{N} \hat{w}_i (L^i_{ln} - L^i_{sn})\right]$$
           in $O(M)$ time.
    Note: If $\hat{V}_R \leq V_c$, then the answer to the decision problem is No. Otherwise, the answer is not determined by this bound. This procedure takes $O(M)$ time. If it fails to give an answer, then we try the conventional technique.
```

In order to obtain the bound above, we conceptually store the matrices L and $-L$ distinctly, and add a certain value $\tilde{k}_j$ and $\hat{k}_j$ to each element of column j of each of the respective matrices. We use corollary 1 to get a bound after using lemma 1 to choose an appropriate value of $\tilde{k}$ and $\hat{k}$.

Theorem 2 can be used to determine a bound on V_R as shown in Algorithm 6. Here, there is an alternative computation we can make. The singular value decomposition of $L = UDV$ can be precomputed, and we can store the (at most) M non-zero rows of (DV). Then $\acute{w} = (DV)w$ can be computed in $O(M^2)$ time. Since $Lw = U\acute{w}$, we can apply the algorithm given in Algorithm 6 to $U, \acute{w}$ instead of to L, w. The complexity of this alternative will be $O(M^2)$ due to the time taken to compute $\acute{w}$, but this can be worthwhile if the bound obtained is significantly better than with L.

6 L_∞ norm bound

The L_∞ norm of a vector gives the magnitude of that component of the vector with the largest absolute value. Clearly, no component, including the p^{th} largest one, can exceed this. As with the L_1 norm, there is no linear transform from the N dimensional domain space to the smaller M dimensional range space that always preserves the L_∞ norm, and therefore, we derive a simple upper bound on this norm instead of using the exact value of the norm.

Theorem 3 *Let L be an $N \times M$ matrix, L_p^j the largest component of column j of L, L_m^j the smallest component of column j, for each j, $1 \le j \le M$, and $w \in \mathbb{R}^N$. Then an upper bound on the p^{th} largest component of Lw is given by*

$$V_R \le \sum_{i=1}^{M} L_p^i \tilde{w}_i - \sum_{i=1}^{M} L_m^i \hat{w}_i$$

where $\tilde{w}$ is given by

$$\tilde{w}_i = w_i, w_i \ge 0;\ \ \dot{w}_i = 0,\ \text{otherwise} \tag{4}$$

and $\hat{w}$ by $\hat{w} = \tilde{w} - w$

Remark
This is an upper bound on any of the components of Lw, and not just on the p^{th} largest one.

Theorem 3 can be used to determine a bound on V_R as shown in Algorithm 7. Here too, as with the L_1 norm, there is an alternative computation we can make. The singular value decomposition of $L = UDV$ can be precomputed, and we can store the (at most) M non-zero rows of (DV). Then $\acute{w} = (DV)w$ can be computed in $O(M^2)$ time. Since $Lw = U\acute{w}$, we can apply the algorithm given in Algorithm 7 to $U, \acute{w}$ instead of to L, w, but with a time complexity of $O(M^2)$.

7 Miscellaneous techniques

We describe here certain miscellaneous techniques based on the Cauchy-Schwartz inequality (Friedberg et al. 1996). All the norms mentioned in this section will be L_2 norms.

Bound on the p^{th} "largest" row: If $b = Lw$ and L_i denotes the i th row of L, then $b_i = L_i w$. Using the Cauchy-Schwartz inequality, $|b_i| \le \|L_i\|\|w\|$ and therefore $b_i \le \|L_i\|\|w\|$. If W_L denotes the L_2 norm of the row with the p^{th} largest such norm, then $W_L\|w\|$ is clearly an upper bound on the p^{th} largest component of b. We precompute W_L. Given a vector w, we determine its L_2 norm easily in $O(M)$ time by summing the square of each component, and taking the square-root. The algorithm is defined in Algorithm 8.

Algorithm 7 Computation of an upper bound on the p^{th} largest component of Lw using the L_∞ norm.

```
Input: $L \in \mathbb{R}^{N \times M}$ and $w \in \mathbb{R}^M$.
Output: An upper bound $\hat{V}_R$ on the $p^{\text{th}}$ largest component of $Lw$.
Procedure:
    • Initialize: The following steps are carried out only once.
        1. Compute $L_p^j = \max_{i=1}^N L_{ij}$, for each $j$, $1 \le j \le M$.
        2. Compute $L_m^j = \min_{i=1}^N L_{ij}$, for each $j$, $1 \le j \le M$.
    • Computation: The following steps are carried out for the $w$ that is given.
        1. Compute $\tilde{w}$ as defined by equation 4 in $O(M)$ time.
        2. Compute $\hat{w} = \tilde{w} - w$ in $O(M)$ time.
        3. Compute
               $$\hat{V}_R = \sum_{i=1}^{M} L_p^i \tilde{w}_i - \sum_{i=1}^{M} L_m^i \hat{w}_i$$
           in $O(M)$ time.
Note: If $\hat{V}_R \le V_c$, then the answer to the decision problem is No. Otherwise, the answer
is not determined by this bound. This procedure takes $O(M)$ time. If it fails to give an
answer, then we fall back upon the conventional technique.
```

If $L = UDV$ is the singular value decomposition of L, then we can similarly use the p^{th} "largest" row of U and DVw. Here, we will need to compute $\|DVw\|$, for which we first need to compute DVw. This latter computation will require $O(M^2)$ time

Eliminate rows: We wish to check if $V_R > V_c$. Equivalently, we wish to check if there are p components of Lw with magnitude greater than $|V_c|$ and a positive sign. We, therefore, do not need to compute those components that we know will have a magnitude smaller than V_c. Using Cauchy-Schwartz inequality, we can ignore rows of L with L_2 norm less than $|V_c|/\|w\|$. We compute the inner product with w for only those rows of L having a greater magnitude, and then count the number of components greater than V_c. If the number exceeds p, then the answer to the decision problem is *YES*. Otherwise, it is *NO*. The algorithm is illustrated in Algorithm 9. If $L = UDV$ is the singular value decomposition of L, then a similar technique can be applied, using $U, (DVw)$, instead of L, w.

In fact, this technique is best applied after using the bounds derived above. If the bounds are unsuccessful in determining an answer, then we use the technique of row elimination. For this purpose, it will be convenient to pre-sort the rows in the order of increasing L_2 norm. Then we can determine the set of rows with norm greater than $|V_c|/\|w\|$ in $\log(N)$ time, and compute inner products with w for only this set of rows.

Algorithm 8 Computation of an upper bound on the p^{th} largest component of Lw using the p^{th} largest row norm.

Input: $L \in \mathbb{R}^{N \times M}$ and $w \in \mathbb{R}^M$.
Output: An upper bound $\hat{V}_R$ on the p^{th} largest component of Lw.
Procedure:

- Initialize: Compute the L_2 norm of each row of L, and let W_L be the p^{th} largest one.
- Computation:
  1. Compute $\|w\|_2$ in $O(M)$ time.
  2. Compute $\hat{V}_R = W_L \|w\|_2$ in constant time.

Note: If $\hat{V}_R \leq V_c$, then the answer to the decision problem is No. Otherwise, the answer is not determined by this bound. This procedure takes $O(M)$ time. If it fails to give an answer, then we fall back upon the conventional technique.

Algorithm 9 Eliminating rows with "small" norms.

Input: $L \in \mathbb{R}^{N \times M}$, $w \in \mathbb{R}^M$, and collateral V_c.
Output: An answer to the decision problem: Yes or No.
Procedure:

- Initialize:
  1. Compute the L_2 norm of each row of L.
  2. Sort the rows of L in descending order of their L_2 norms and store them in a matrix $\tilde{L}$. If $\tilde{L}_i$ denotes the ith row of $\tilde{L}$, then $i < j$ implies $\|\tilde{L}_i\| \geq \|\tilde{L}_j\|$ since the rows are sorted.
  3. Store the corresponding norms of the rows in a sorted array $\tilde{l}$.
- Computation:
  1. Compute $\|w\|_2$ in $O(M)$ time.
  2. Determine the largest index s such that $\tilde{l}_s > |V_c|/\|w\|$. If there is no such component, then set $s = 0$. This index can be determined in $O(\log N)$ time using binary search.
  3. If $s < p$, then there fewer than p rows with a potential for their inner product exceeding V_c, and therefore the answer to the decision problem is No. Otherwise compute inner product $\tilde{L}_i w$ for each $i \leq s$ and call the set of inner products computed $\hat{b}$. This takes $O(sM)$ time.
  4. If the number of elements in $\hat{b}$ greater than V_c is fewer than p, then the answer to the decision problem is No. Otherwise, it is Yes. This can be done in $O(s)$ time.

Therefore the total time taken is $O(sM + \log N)$. In the worst case, it can be $O(NM)$. However, if s is small, then there is a potential for a significant reduction in time.

Algorithm 10 Heuristic for potential reduction in time when VaR is exceeded, by first computing the inner product for rows with large norms.

Input: $L \in \mathbb{R}^{N\times M}$, $w \in \mathbb{R}^M$, and collateral V_c.
Output: An answer: Yes. to the decision problem, or Undecided.
Procedure:

- Initialize: Compute the L_2 norm of each row of L, and let $\hat{L}$ be the matrix consisting of the h rows with the largest L_2 norms, where h is a fixed constant greater than p.
- Computation:
  1. Compute $\hat{L}w$ in $O(hM)$ time.
  2. Count the number q of components of $\hat{L}w$ that are greater than V_c in $O(h)$ time.
  3. If $q \geq p$, then the answer is Yes. Otherwise, the answer could not be determined by this procedure, and so return Undecided.

Note: This procedure takes $O(sM)$ time, where we can choose the constant h such that $p < h \ll N$. If it fails to give an answer, then we compute the inner product with the rest of the rows, and proceed as usual. This procedure can use U, DVw instead of L, w too, but there will be an additional $O(M^2)$ time, due to the cost of computing DVw.

8 Reducing computation when VaR is exceeded

All the bounds derived above give an answer No, or no answer at all. They have the potential to reduce the amount of computation only if the VaR is not exceeded. The technique of row elimination is the only one described so far that is capable of reducing the amount of computation when the VaR is exceeded. We give here a heuristic which is capable of eliminating much of the computation when the VaR is exceeded.

Heuristic using rows with large L_2 norms: All other factors being equal, we may expect that rows of L with large norms will result in corresponding components of Lw with larger magnitudes than for those rows with smaller norms. In this heuristic, we compute the inner product with b for h rows of L with the largest L_2 norms, where we choose some $h > p$. We can then try either of the following two procedures: (i) Let q be the number of components with value greater than V_c. If $q \geq p$, then the VaR is exceeded, and the answer to the decision problem is Yes. Otherwise, we compute the inner product with the remaining rows and find the $p - q$ th smallest component of that, and check if it exceeds V_c. (ii) We determine the p^{th} smallest component of the inner product of w with the h selected rows. This can be accomplished in $O(h)$ time using the linear time order statistic selection algorithm. This gives a lower bound on the p^{th} largest component. We have outlined the algorithm using the variant (i) in Algorithm 10.

Yet another variant that combines the algorithms of Algorithm 9 and 10 is given in Algorithm 11.

Algorithm 11 Combining Algorithm 9 and 10.

Input: $L \in \mathbb{R}^{N \times M}$, $w \in \mathbb{R}^M$, and collateral V_c.
Output: An answer to the decision problem: Yes or No.
Procedure:

- Initialize:
  1. Compute the L_2 norm of each row of L.
  2. Sort the rows of L in descending order of their L_2 norms and store them in a matrix $\tilde{L}$. If $\tilde{L}_i$ denotes the i th row of $\tilde{L}$, then $i < j$ implies $\|\tilde{L}_i\| \geq \|\tilde{L}_j\|$.
  3. Store the corresponding norms of the rows in a sorted array $\tilde{l}$.
- Computation:
  1. Compute $\|w\|_2$ in $O(M)$ time.
  2. Determine the largest index s such that $\tilde{l}_s > |V_c|/\|w\|$. If there is no such component, then set $s = 0$. This index can be determined in $O(\log N)$ time using binary search.
  3. If $s < p$, then there fewer than p rows with a potential for their inner product exceeding V_c, and therefore the algorithm can terminate with the answer to the decision problem as: No.
  4. Initialize variable *count* to 0.
  5. Loop for $i = 1 \ldots s$
     (a) Compute $\dot{b}_i = \tilde{L}_i w$
     (b) If $\dot{b}_i > V_c$, then increment *count* by 1. If *count* $\geq p$, then terminate the computation with the answer Yes.
  6. If the computation has not terminated with an answer so far, then the output is No.

The computation complexity will be not more than the lower of the ones in Algorithms 9 and 10. This procedure can use U, DVw instead of L, w too, but there will be an additional $O(M^2)$ time required for the computation of DVw.

Algorithm 12 A composite algorithm for the VaR decision problem.

Input: $L \in \mathbb{R}^{N \times M}$, $w \in \mathbb{R}^{M}$, and collateral V_c.
Output: An answer to the decision problem: Yes or No.
Procedure:

- Step 1: Determine an upper bound $\hat{V}_R$ on the p^{th} largest component of Lw, using bounds given in Algorithm 5, 6, 7, or 8.
- Step 2: If $\hat{V}_R \leq V_c$, then terminate this algorithm with answer No.
- Step 3: Otherwise, apply Algorithm 10.
- Step 4: If the answer from the above step is Yes. then terminate this algorithm with answer Yes.
- Step 5: Apply Algorithm 9 and return the answer from this algorithm.

In any of the steps 1, 3, or 5, we can use U, DVw instead of L, w. Also, we can replace steps 3 to 5 with the algorithm in Algorithm 11.

9 A composite algorithm

The algorithms based on the bounds are effective in reducing computational cost only when the VaR is not exceeded, whereas the algorithm outlines in section 8 is effective only when the VaR is exceeded. Since we do not know ahead of time whether the VaR will be exceeded or not, we give below in Algorithm 12 a composite algorithm that combines both strategies, so that it will be effective under both cases.

10 Empirical tests

Now that we have several bounds and heuristics, we need to verify, empirically, how they perform in a realistic situation. We first describe the testing procedure and explain why we consider it realistic. We then then present the results on the effectiveness of the different procedures that we have tried.

The tests are performed on twelve sample L matrices on 1000 random portfolios each. Appendix C documents how the twelve L matrices were created. All the code is in Matlab, and run on an Intel Celeron 300MHz processor with 64MB RAM. The Matlab function *cputime* is used to give the CPU time consumed in seconds. We have determined the times consumed over (i) all the portfolios, (ii) the subset for which the VaR was exceeded, and (iii) the subset for which the VaR is not exceeded. The collateral value was selected by using the heuristic that V_c was 10% of the absolute value of the portfolio at the beginning of the day.

Algorithms tested: The specific algorithms tested are as given below:

1. The conventional technique; i.e. Algorithm 2.
2. The algorithm using the L_2 bound in Algorithm 5.
3. The algorithm using the L_1 bound in Algorithm 6.

4. The algorithm using the L_1 bound in Algorithm 6, with U, DVw replacing L, w.
5. The algorithm using the L_∞ bound in Algorithm 7.
6. The algorithm using the L_∞ bound in Algorithm 7, with U, DVw replacing L, w.
7. Algorithm 8.
8. Algorithm 8, with U, DVw replacing L, w.
9. Algorithm 9.
10. Algorithm 9, with U, DVw replacing L, w.
11. Algorithm 10, with $h = 0.05N$.
12. Algorithm 10, with U, DVw replacing L, w and $h = 0.05N$.
13. Algorithm 12, where the L_1 of Algorithm 6 is used in step 1 (with U, DVw replacing L, w), and the algorithm in Algorithm 2 is used instead in step 5.
14. Algorithm 12, where the L_1 of Algorithm 6 is used in step 1 (with U, DVw replacing L, w), and the algorithm in Algorithm 2 is used instead in step 5. Step 3 uses U, DVw instead of L, w and $h = 0.05N$.

We note that we have replaced row elimination with the conventional technique of Algorithm 2 in the composite algorithms. The reason for this is that our Matlab implementation of the former technique gave worse results than the latter. We believe the cause to be the creation of unnecessary temporary arrays in Matlab. This issue, however, requires deeper study. Even without taking advantage of the row elimination technique, there is a substantial improvement in performance. We further note that the results clearly show that *Alg 4* gives the most effective bounds, and therefore we use this one alone in step 1 of the composite algorithm.

In order to also verify correctness of our implementation, we first compute the VaR estimate with a crude technique, where we replace step 2 of Algorithm 2 with first sorting Lw and then selecting the p^{th} largest component. The results of the decision problem solved using this crude technique is then compared with the results obtained from each algorithm, to verify correctness of the implementation.

The test results are presented in Fig 1. Let T_i be the time taken for the i th test matrix $1 \leq i \leq 12$ with the conventional algorithm *Alg 1*, and t_i the time taken for the algorithm we are comparing with. Then t_i/T_i is the time taken as a fraction of the time taken for the conventional technique. (By definition, this ratio is 1 for the conventional technique). $(1/12)\sum_{i=1}^{12} t_i/T_i$ is the average fraction of time taken by the algorithm relative to the conventional technique. Consequently, subtracting this quantity from 1 gives the relative saving, when this ratio is less than 1. Similarly, $\min_{i=1}^{12} t_i/T_i$ gives the relative time for the matrix for which this algorithm was most effective, and $\max_{i=1}^{12} t_i/T_i$ gives the relative timing for the most ineffective case. We plot each of these along the same abscissa for each algorithm. We make similar plots for the subset of portfolios where the VaR was not exceeded are presented in Fig 2, and for the subset where the VaR was exceeded in Fig 3.

The composite algorithms give a saving of, on the average, around 87%, with around 85% when the VaR is not exceeded, and around 90% when the VaR is exceeded. Thus our technique

Figure 1 Overall performance of algorithms, relative to conventional

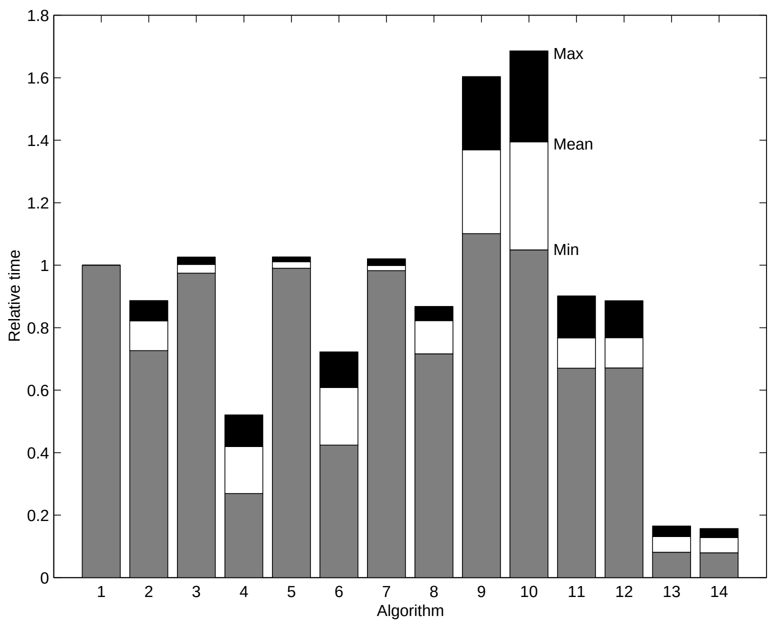


Figure 2 Performance on portfolios for which VaR was not exceeded

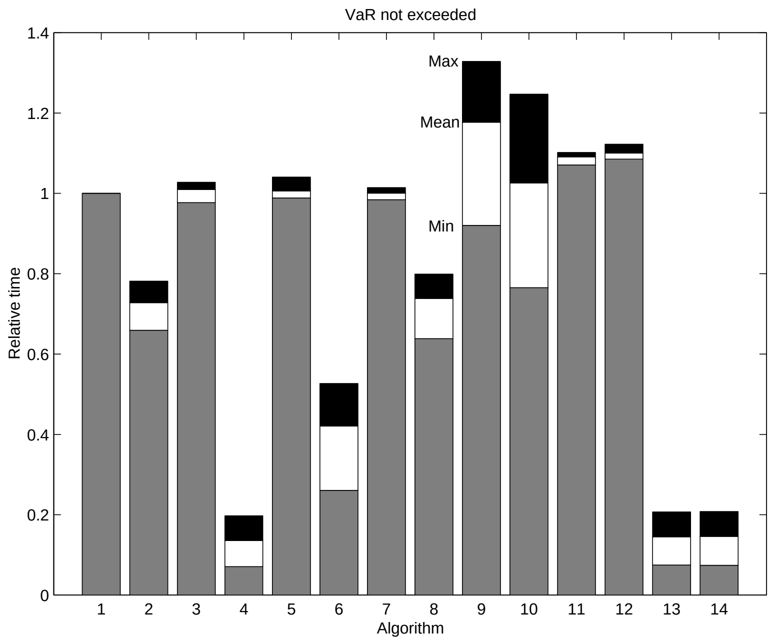
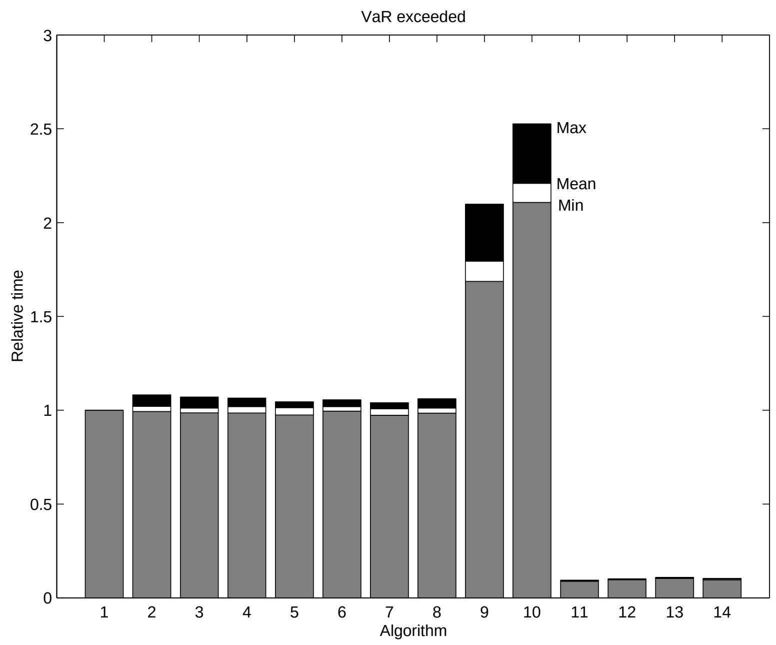


Figure 3 Performance on portfolios for which VaR was exceeded



is about seven times faster than the conventional technique. We can further observe that the L_1 bound appears to be the most effective. It is also apparent that using U, DVw is generally more effective than using L, w, despite the additional cost of computing DVw. We make a couple of further observations: (i) The composite algorithm performs well even in the worst case, that is, with the loss matrix where its performance is the worst (rather than for its worst portfolio). (ii) We can see from Fig 3, where the upper bounds are of no use, that computation of the bounds does not cost much compared with the total time taken. Thus the time taken for the algorithms that use only the upper bounds is not much more than that for the conventional technique.

11 Conclusions

In this paper, we set out to obtain faster methods for solving VaR problems in one restricted case: where VaR is done by Monte Carlo, where many VaR calculations are required for different portfolios in an environment with a fixed universe of portfolios and a fixed data generating process.

We shifted our goal away from the evaluation of VaR itself to solving the VaR decision problem. We find that our new proposals for solving the VaR decision problem yield important gains.

These techniques do not violate the innate parallelism in the VaR problem, which has been exploited by Shah et al. (2000). These innovations would easily blend with the parallel architecture proposed by them.

The ideas of Shah et al. (2000) make intra–day real–time VaR calculation possible in the risk–containment of the futures clearing corporation; the authors show real–world evidence of the feasibility of these methods for a simple case (with 33 derivative products being traded off one underlying). However, the large derivatives exchanges of the world would face an L matrix with around 10^5 rows and 10^2 columns. In this case, the matrix–vector product would be a formidable problem. The results of this paper are important insofar as they make it possible to attempt intra–day real–time VaR calculations on the largest futures exchanges of the world.

A Notation

D: The singular value decomposition of L is $L = UDV$.
L: Loss matrix, $\in \mathbb{R}^{N \times M}$.
L^i: Column i of L.
L^i: Row i of L.
L^j_m: Smallest component of column L^j.
L^j_p: Largest component of column L^j.
L^j_{lm}: Sum of the p smallest components of column L^j.
L^j_{lp}: Sum of the p largest components of column L^j.
L^j_{sm}: Sum of the $N - p$ largest components of column L^j.
L^j_{sp}: Sum of the $N - p$ smallest components of column L^j.
M: Number of products.
N: Number of draws in the Monte Carlo simulation.
P: Level of significance for VaR.
S: $\sum_{i=1}^{N} b_i$.
S_1, S_2: A decomposition of S as $S = S_1 + S_2$.
S_j: $\sum_{i=1}^{M} L_{ij}$.
T: Number of underlyings.
U: The singular value decomposition of L is $L = UDV$.
V: The singular value decomposition of L is $L = UDV$.
V_0: Value of a portfolio based on the prices at the beginning of the day.
V_R: VaR estimate.
V_c: Value of the collateral.
W: $\sum_{i=1}^{N} b_i^2$.
W_1, W_2: A decomposition of W as $W = W_1 + W_2$.
$X(k)$: A function defined as $\|b + \vec{k}\|_1 + (N - 2p)k$.
$\dot{W}$: $\|b\|_1$.
$\hat{V}_R$: A bound on V_R.
$\hat{X}$: Optimal value of $X(k)$ over all possible k.
$\tilde{L}, \hat{L}$: Matrices obtained from L through certain simple transformations.
$\tilde{w}, \hat{w}$: Decomposition of w as $w = \tilde{w} - \hat{w}$.
$\vec{k}$: A vector in $\mathbb{R}^N$ which each component equal to k.
b: Lw.
f: Joint distribution of the underlyings.
h: Number of rows $p < h \ll N$ used in the heuristic defined in Algorithm 10.
l: A sorted array of the L_2 norms of the rows of L.
p: $P \times N$.
r: Vector of values of the underlying.
v: Valuation function for products as a function of the underlyings.
v_o: Value of a product at the beginning of the day.
w: A portfolio, a vector in $\mathbb{R}^N$.

B Proofs

Theorem 1

Proof In order to simplify the notation, we can assume without loss of generality[2] that $b_1 \geq b_2 ... \geq b_N$ and therefore the p largest value is the component b_p. We wish to determine an upper bound on b_p, given S and W. (i) We first show that when b_p has its largest possible value, it must necessarily be the case that $b_1 = b_2 = ... = b_p$ and $b_{p+1} = ... = b_N$. (ii) We then use this to determine the largest possible value, which gives the bound in the theorem statement.

(i) Show that $b_1 = b_2 = ... = b_p$ and $b_{p+1} = ... = b_N$ when b_p attains it maximum possible value: We prove this by contradiction. Assume that this is not true.

The sum of the components of b is given by

$$S = S_1 + S_2 \tag{5}$$

where $S_1 = \sum_{i=1}^{p} b_i$ and $S_2 = \sum_{i=p+1}^{N} b_i$.

Similarly, $\|b\|_2$ is given by

$$W = W_1 + W_2 \tag{6}$$

where $W_1 = \sum_{i=1}^{p} b_i^2$ and $W_2 = \sum_{i=p+1}^{N} b_i^2$.

Since b_p is the lowest among $b_1, ..., b_p$, it cannot be greater than the mean of these quantities and therefore we have the following inequality

$$b_p \leq S_1/p$$

Since we seek to prove the result by contradiction, we assume that at least one of the following relations is false:

$$b_1 = b_2 = ... = b_p \text{ or } b_{p+1} = ... = b_N \tag{7}$$

We use the property that for any $b \in \mathbb{R}^N$

$$\sum_{i=1}^{N} b_i \leq \sqrt{N}\|b\|_2 \tag{8}$$

with equality if and only if all the components of b are equal. (We use the properties that $\sum_{i=1}^{N} b_i \leq \|b\|_1 \leq \sqrt{N}\|b\|_2$ (Heath 1996).) Applying this to $(b_1, ..., b_p)$ and $(b_{p+1}, ..., b_N)$, and with our assumption about equation 7, at least one of the following inequalities must be true

$$pW_1 > S_1^2 \text{ or } (N-p)W_2 > S_2^2 \tag{9}$$

In order to show the contradiction, we find a vector $\hat{b}$ such that $\sum_{i=1}^{N} \hat{b}_i = S$, $\sum_{i=1}^{N} \hat{b}_i^2 = W$, and $\hat{b}_p > b_p$. (Again, we assume without loss of generality that the components of $\hat{b}$ are in descending order.) This will

[2] To be precise, we need to state that there are indices $i_1, i_2, \cdots, i_N$ such that $b_{i_1} \geq b_{i_2} ... \geq b_{i_N}$. In order to simplify the notation, we remove the subscript i and simply refer to them as $b_1, b_2, ..., b_N$.

contradict the assumption that the the p^{th} largest component attains its maximum possible value as b_p, given S and W.

Consider the vector $\hat{b}$ given by

$$\hat{b}_1 = \hat{b}_2 = ... = \hat{b}_p = \frac{S_1 + \delta}{p}$$

and

$$\hat{b}_{p+1} = ... = \hat{b}_N = \frac{S_2 - \delta}{N - p}$$

This clearly satisfies Equation 5. We next choose $\delta > 0$ such that it will satisfy Equation 6.

We wish to have

$$\frac{1}{p}\left(S_1 + \delta\right)^2 + \frac{1}{N-p}\left(S_2 - \delta\right)^2 = W$$

Straightforward algebraic manipulations give us the following quadratic equation:

$$N\delta^2 + 2\delta[S_1(N-p) - S_2 p] + (N-p)S_1^2 + pS_2^2 - Wp(N-p) = 0$$

Solving this equation, we obtain the larger of the two solutions as:

$$\delta = (1/N)\left(-\beta + \sqrt{\beta^2 + N[Wp(N-p) - pS_2^2 - (N-p)S_1^2]}\right)$$

where $\beta = (S_1(N-p) - S_2 p)$. In order to show that δ is positive, we need only show that $[Wp(N-p) - pS_2^2 - (N-p)S_1^2] > 0$. That is, we wish to show that

$$(W_1 + W_2)p(N-p) - pS_2^2 - (N-p)S_1^2 > 0$$

That is,

$$(N-p)[W_1 p - S_1^2] + p[W_2(N-p) - S_2^2] > 0$$

But at least one of the terms within the square brackets is greater than zero, by equation 9, and neither of the terms is less than zero by equation 8. Of course, p and $N-p$ too are positive since $p \in [1, N/2]$. Therefore the desired inequality is satisfied and δ is indeed greater than zero. Thus $\hat{b}$ satisfies both the equations 5 and 6, and $\hat{b}_p = S_1/p + \delta/p > b_p$, contradicting our assumption that b_p is the largest possible value. Thus the assumption is false and we must have $b_1 = ... = b_p$ and $b_{p+1} = ... = b_N$ when b_p attains its maximum.

(ii) Determine maximum value of b_p: We need to solve the following equations, in light of the above result:

$$pb_p + (N-p)b_N = S$$

$$pb_p^2 + (N-p)b_N^2 = W$$

Solving these through straightforward algebraic manipulations, and solving the resulting quadratic equation and choosing the larger of the two possible values for b_p, we get

$$b_p = (1/N)\left(S + \sqrt{\frac{(N-p)}{p}(N*W - S^2)}\right)$$

The term within the radical sign is non-negative from Equation 8, and thus this value of b_p can indeed be attained and is the largest possible value.

Remark
The above proof also provides a vector that has the same sum and L_2 norm as the vector b, with the p largest component being equal to the upper bound. Thus this bound is tight if the only information available is S, W, N, and p.

QED.

Lemma 1

Proof Without loss of generality, in order to simplify the notation, let us take $b_1 \leq b_2 \leq \ldots \leq b_N$. This is equivalent to just renaming the components of b. We thus wish to show that $X(k)$ attains its global minimum value for all k such that $-b_{N-p+1} \leq k \leq -b_{N-p}$.

$$X(k) = \sum_{i=1}^{N} \|b_i + k\|_1 + (N-2p)k$$

Let us first consider the case when $-b_N \leq k \leq -b_1$. We will later show that minimum value(s) will, indeed, occur in this interval.

Let q be such that such that $-b_{q+1} \leq k \leq -b_q$. Then

$$\begin{aligned} X(k) &= \sum_{i=1}^{q} |b_i + k| + \sum_{i=q+1}^{N} |b_i + k| + (N-2p)k \\ &= \sum_{i=q+1}^{N} (b_i + k) - \sum_{i=1}^{q} (b_i + k) + (N-2p)k \\ &= \sum_{i=q+1}^{N} b_i - \sum_{i=1}^{q} b_i + 2(N-q-p)k \end{aligned}$$

If $q = N-p$, then $X(k)$ is constant in the corresponding interval $k \in [-b_{N-p+1}, -b_{N-p}]$ and its value is given by $\hat{X} = \sum_{i=N-p+1}^{N} b_i - \sum_{i=1}^{N-p} b_i$.

We show that $\hat{X}$ is the minimum value of $X(k)$ for $k \in [-b_N, -b_1]$. Consider k in any other interval $[-b_{s+1}, -b_s]$. So $k = -b_s - \delta$, for some δ such that $b_{s+1} - b_s \geq \delta \geq 0$. Then the value of $X(k)$ is given by:

$$X = \sum_{i=s+1}^{N} b_i - \sum_{i=1}^{s} b_i + 2(N-s-p)(-b_s - \delta)$$

$$X - \hat{X} = \sum_{i=s+1}^{N} b_i - \sum_{i=1}^{s} b_i - \sum_{i=N-p+1}^{N} b_i + \sum_{i=1}^{N-p} b_i + 2(N-s-p)(-b_s - \delta)$$

Case 1: $s < N - p$

$$X - \hat{X} = \sum_{i=s+1}^{N-p} b_i + \sum_{i=s+1}^{N-p} b_i - 2(N - s - p)(b_s + \delta)$$

$$= 2\left[\sum_{i=s+1}^{N-p} b_i - (N - s - p)(b_s + \delta)\right] = 2\left[\sum_{i=s+1}^{N-p} (b_i - [b_s + \delta])\right]$$

But $b_s + \delta \leq b_i$ for $i \in [s+1, N-p]$ from the condition: $b_{s+1} - b_s \geq \delta \geq 0$. Therefore $X - \hat{X} \geq 0$.

Case 2: $s > N - p$

$$X - \hat{X} = -2\left[\sum_{i=N-p+1}^{s} b_i + (N - s - p)(b_s + \delta)\right] = 2\left[\sum_{i=N-p+1}^{s} (b_s + \delta - b_i)\right]$$

since $s > N - p$. Since $b_s \geq b_i$ for $i \leq s$, we have $X - \hat{X} \geq 0$ for this case too.

Therefore $\hat{X}$ is the minimum value of $X(k)$ when $-b_N \leq k \leq -b_1$.

If $k < -b_N$, then

$$X(k) = -\sum_{i=1}^{N}(b_i + k) + (N - 2p)k = -\sum_{i=1}^{N} b_i - 2pk$$

But,

$$X(-b_N) = -\sum_{i=1}^{N} b_i + 2pb_N$$

Since $k < -b_N$ here, $-k > -b_N$ and thus $X(k) > X(-b_n)$ Similarly it is easy to show that $X(k) > X(-b_1)$ when $k > -b_1$. Therefore the minimum value lies in $-b_N \leq k \leq -b_1$ and is attained throughout the interval $-b_{N-p+1} \leq k \leq -b_{N-p}$, with the minimum value being $\sum_{i=N-p+1}^{N} b_i - \sum_{i=1}^{N-p} b_i$. QED.

Lemma 2

Proof Once again, without loss of generality, we assume that $b_1 \geq b_2 \geq \ldots \geq b_N$. We wish to show that $b_p \leq \frac{1}{2p}(\dot{W} + S)$.

Assume this is not true. Then there exists a vector b with the desired $\sum_{i=1}^{N} b_i = S$ and $||b||_1 = \dot{W}$ such that $b_p = \frac{1}{2p}(\dot{W} + S + \delta)$, for some $\delta > 0$. Therefore

$$\sum_{i=1}^{p} b_i \geq \frac{1}{2}(\dot{W} + S + \delta) \tag{10}$$

Note that $\sum_{i=1}^{p} b_i > 0$ since $\dot{W} \geq |S|$. From equation 10, we get

$$\sum_{i=p+1}^{N} b_i \leq \frac{1}{2}(S - \dot{W} - \delta)$$

Note that $\sum_{i=p+1}^{N} b_i < 0$ since $\dot{W} \geq |S| \geq S$ and $\delta > 0$. Since both terms above are negative, the inequality implies that the magnitude of the left hand side is at least great as that of the right hand side. Therefore:

$$|\sum_{i=p+1}^{N} b_i| = -\sum_{i=p+1}^{N} b_i \geq |\frac{1}{2}(S - \dot{W} - \delta)| = \frac{1}{2}(\dot{W} + \delta - S)$$

$$\sum_{i=p+1}^{N} |b_i| \geq |\sum_{i=p+1}^{N} b_i| \geq= \frac{1}{2}(\dot{W} + \delta - S)$$

Similarly,

$$\sum_{i=1}^{p} |b_i| \geq \frac{1}{2}(\dot{W} + S + \delta)$$

Adding the above two equations, we get

$$\dot{W} \geq \frac{1}{2}(\dot{W} + S + \delta) + \frac{1}{2}(\dot{W} + \delta - S) = \dot{W} + \delta$$

This is impossible since $\delta > 0$.

Therefore $b_p \leq \frac{1}{2p}(\dot{W} + S)$ QED.

Corollary 1

Proof Again, we take $b_1 \geq b_2 \geq \ldots \geq b_N$, without loss of generality. Applying Lemma 2 to the vector $(b + \vec{k})$ we get

$$|b_p + k| \leq \frac{1}{2p}(\|b + \vec{k}\|_1 + S + Nk)$$

Since the absolute value is an upper bound on the number, $b_p + k$ is less than or equal to the right hand side, and the result follows immediately. QED.

Theorem 2

Proof Let $\vec{k}$ denote the vector in $\mathbb{R}^N$ with $\vec{k}_i = k$. Define the following two matrices

$$\tilde{L} = L + [\vec{\tilde{k}_1}\ \vec{\tilde{k}_2}\ \ldots\ \vec{\tilde{k}_M}]$$

and

$$\hat{L} = -L + [\vec{\hat{k}_1}\ \vec{\hat{k}_2}\ \ldots\ \vec{\hat{k}_M}]$$

where the $\tilde{k}_i$ and $\hat{k}_i$ will be suitably define later to optimize the bound.

Now,

$$\tilde{L}\tilde{w} + \hat{L}\hat{w} = Lw + \vec{k}$$

where $k = \sum_{i=1}^{M}(\tilde{w}_i\tilde{k}_i + \hat{w}_i\hat{k}_i)$.

From corollary 1, a bound on the p^{th} largest component of Lw is given by

$$V_R \leq \frac{1}{2p}\left(\|Lw + \vec{k}\|_1 + S + (N-2p)k\right)$$

If L^j denotes column j of matrix L, then we get

$$2pV_R \leq S + \|\sum_{i=1}^{M}(L^i + \vec{\tilde{k}_i})\tilde{w}_i + \sum_{i=1}^{M}(-L^i + \vec{\hat{k}_i})\hat{w}_i\|_1 + (N-2p)\sum_{i=1}^{M}\left(\vec{\tilde{k}_i}\tilde{w}_i + \vec{\hat{k}_i}\hat{w}_i\right)$$

Using the triangle inequality and $\tilde{w}_i, \hat{w}_i \geq 0$ we get

$$2pV_R \leq S + \sum_{i=1}^{M}\tilde{w}_i\left(\|L^i + \vec{\tilde{k}_i}\|_1 + (N-2p)\tilde{k}_i)\right) + \sum_{i=1}^{M}\hat{w}_i\left(\| - L^i + \vec{\hat{k}_i}\|_1 + (N-2p)\hat{k}_i)\right)$$

Using lemma 1, we can minimize each $\|L^i + \vec{\tilde{k}_i}\|_1 + (N-2p)\tilde{k}_i$ and similarly for the $\hat{w}$'s. With the k's as given in that lemma, the bound $\|L^i + \vec{\tilde{k}_i}\|_1 + (N-2p)\tilde{k}_i$ is given by the difference of the p largest components of the column and the $N-p$ smallest components of that column: $(L^i_{lp} - L^i_{sp})$. Similarly, for $\hat{w}$s, the bound is given by: $-(L^i_{ln} - L^i_{sn})$ (since the p largest values of $-L^i$ are $-1 \times$ (the p smallest values of L^i). QED.

Theorem 3

Proof The proof is rather simple. We observe that

$$Lw = L\tilde{w} - L\hat{w} = L\tilde{w} + (-L)\hat{w}$$

$$= \sum_{i=1}^{M} L^i\tilde{w}_i + \sum_{i=1}^{M}(-L)^i\hat{w}_i$$

where L^i denotes column i of L. Since $L_{ji} \leq L^i_p$, $-L_{ji} \leq -L^i_m$, and $\tilde{w}_i, \hat{w}_i \geq 0$, each component of Lw must be at most

$$\sum_{i=1}^{M} L^i_p\tilde{w}_i + \sum_{i=1}^{M}(-L)^i_m\hat{w}_i$$

and the result follows trivially. QED.

C The twelve L matrices used in empirical measurement

Our test environment is an index futures and index options market with one underlying index, India's NSE-50 index (Shah & Thomas 1998). We assume there are three futures products. The spot index is set to $S = 1000$, and a range of five option strikes are available, from $X = 800$ to $X = 1200$ in steps of 100. Three option expiration dates, five option strikes and call/put gives thirty options. Hence there are 33 products in all.

The underlying stock index process is assumed to follow a GARCH(1,1) process (Bollerslev et al. 1992):

$$
\begin{aligned}
r_t &= \mu + \epsilon_t \\
\epsilon_t &\sim N(0, h_t) \\
h_t &= \alpha_0 + \alpha_1 \epsilon_{t-1}^2 + \gamma_1 h_{t-1} \\
h_0 &= H_0
\end{aligned}
$$

The parameter values are estimated using India's NSE–50 index (Thomas 1998) and prove to be: $\mu = 0, H_0 = 1.96, \alpha_0 = 0.13, \alpha_1 = 0.1, \gamma_1 = 0.85$.

In this, we obtain twelve test cases by varying three tuning parameters:

1. The expiration dates can be set to 1,2,3 months, or to 1,2,3 quarters.
2. The previous day's return can be set to $r_{t-1} = 0$ or $r_{t-1} = -3$. This works through the GARCH model and yields lower or higher volatility today.
3. The GARCH H_0 parameter can be set to 1, 2 or 3.

This gives us $2 \times 2 \times 3$ or 12 cases.

References

Bollerslev, T., Chou, R. Y. & Kroner, K. F. (1992), 'ARCH modeling in finance: A review of the theory and empirical evidence', *Journal of Econometrics* **52**(1–2), 5–60.

Culp, C. L., Miller, M. H. & Neves, A. M. P. (1998), 'Value at risk: Uses and abuses', *Journal of Applied Corporate Finance* **10**(4), 26–38.

Estrella, A., Hendricks, D., JohnKambhu, Chin, S. & Walter, S. (1994), 'The price risk of options positions: Measurement and capital requirements', *FRBNY Quarterly Review* pp. 27–43.

Friedberg, S. H., Insel, A. J. & Spence, L. E. (1996), *Linear Algebra*, 3rd edn, Prentice Hall.

Heath, M. T. (1996), *Scientific Computing : An Introductory Survey*, McGraw-Hill Series in Computer Science, McGraw Hill.

Jorion, P. (2000), *Value at Risk : The Benchmark for Controlling Market Risk*, 2nd edn, McGraw Hill.

Shah, A. & Thomas, S. (1998), Market microstructure considerations in index construction, *in* 'CBOT Research Symposium Proceedings', Chicago Board of Trade, pp. 173–193.

Shah, A., Thomas, S., Darbha, G. & Misra, S. (2000), Risk measurement in parallel, Technical report, IGIDR.

Thomas, S. (1998), Volatility forecasting in Indian financial markets, *in* S. Thomas, ed., 'Derivatives markets in India', Invest India – Tata McGraw–Hill Series, Tata McGraw–Hill, chapter 24, pp. 225–233.