QUICK GRID TO COPE WITH COUNTERPARTY RISKS

Radek Skoda, Gergely Szalka
European Bank for Reconstruction and Development
One Exchange Square
London EC2A 2JN, United Kingdom

Abstract

This paper describes, analyzes and evaluates algorithms and implementations for calculating potential future credit exposures borne by financial institutions. As potential future exposure belongs among the most important measures of counterparty risk, it needs to be calculated precisely and on a daily basis. This creates a very high demand on computer power where single computers can not cope with the high number of calculations within the given time. This paper focuses on a grid-type computer solution, the optimal number of computers connected to the grid, the speed gain of connecting an individual computer to the grid and the order in which the subtasks are assigned to the grid. The results indicate that using a grid for simulating potential future exposure profiles is a feasible solution for credit risk management practitioners.

Key words: potential future exposure, parallelisation, grid, Monte - Carlo simulation

1. Introduction

Let us imagine a model situation: We lend money to a friend for 5 days and then we start to ponder how much we might lose in the unlikely event of the friend dying 3 days from now. If we are engaged too deeply in such pessimistic thoughts, we would rather need a good psychologist, not a complex grid of computers to solve the problem. For a bank where the number of institutional debtors tends to be large, portfolios are more complex and usual investment horizons may be measured in decades it might be not too crazy to try to determine a potential future loss in the case where one of the debtors "dies" or, rather, defaults. Then the complex approach is worthwhile and may end up with the need of heavy calculations. It is especially likely when the bank wants to know every day the potential future loss for every debtor and for the entire maturity term of the portfolio. The remainder of this paper will show one of the ways how this common problem may be effectively tackled in reality.

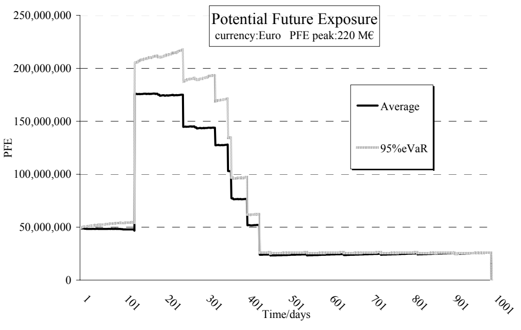
After a brief introduction of credit markets, a short discussion on advanced measures of risk, namely potential future exposure, is provided in section 2. The main part of the paper describes the calculation the exposure profiles, and how to come to terms with the huge amount of computation power that is needed to estimate a percentile of portfolio prices at a number of points in the future. In section 3 the simulation process is briefly described, followed by section 4 where the management of maintaining an acceptable precision within run-time constraints. In section 5 the findings and results are summarised.

2. Credit market developments and the need to know future exposures

In recent years credit and derivatives markets (Hull [1993], Hull [2002]) have increased in volume, importance and complexity and hence credit risk measurement has grown proportionally. Structured credit instruments have increasingly liquid markets; products where the credit risk can be transferred from one dealer to the other and price risk can be separated from the holder of the underlying instrument. Using credit default swaps, credit risk can be separated from the bond or other obligation one keeps in the balance sheet. Looking at developments in the credit and derivatives markets from another angle shows that handling counterparty transactions at portfolio level has important benefits: e.g. the effect of legal rights like close out and collateralisation can be measured. These considerations can dramatically change the amount of credit risk exposed to a given counterparts, especially when the amount one bank is owing to the trading partner is collateralised. The portfolio context is preferable also as it takes into account correlation between transactions. For effective measurement of all associated risks the portfolio approach requires advanced techniques to be properly deployed.

Potential future exposure

The starting point of any risk calculation is to calculate the *exposure*, the maximum amount the bank loses in case the debtor or counterparty defaults. *Exposure* today is the current positive value of the portfolio. Potential exposure, however, in a given future point is an estimated percentile of the distribution of the future value(s) of the portfolio – the value of the portfolio depends on the market prices and rates, so it can be thought of as a random variable. Say, that we estimate the potential exposure in such a way that it is not exceeded by the loss with 95% probability. This means that an estimate of the 95% percentile of the distribution of the future value(s) of the portfolio is needed. For a number of reasons the *expected shortfall*[1] is used instead, this being the expected value of the range that exceeds the percentile. In the following example *potential future exposure* will be the expected shortfall of the future value of the portfolio with 95% confidence. Exposure profile will be the potential future exposure numbers plotted against time.



These profiles are good measures of potential exposure by themselves, but are crucial from control aspects allowing risk takers to ensure that they stay within an institution appetite for credit risk on a particular name (that is, comply credit limits).

3. Calculating exposure profile

Estimates of the percentile of a potential portfolio value distribution at a given date are given here. First one has to simulate the market drivers that – through the pricing function – will determine the value of the portfolio (Duffie [2001]). A one factor model for yield curves and foreign exchange rates is used here:

$$dr = f(t,r)\,dt + g(t,r)\,dz, \qquad (1)$$

where *f(t,r)* and *g(t,r)* functions are described in the model (Rebonato [1996], Jarrow [1998]), *dz* denotes the standard Wiener process, and *t* stands for time. In fact not one driver but a handful of factors are modelled since each currency has its own process (exchange rate, interest rate, etc.). So, rather, one should write:

$$\Delta R = F(t,R) + G(t,R)\Delta Z, \qquad (2)$$

[1] The same quantity is also called eVaR (excess value-at-risk) and CVaR (conditional value-at-risk) in the literature.

where R is a vector of market factors, F is a vector function, G is a matrix containing all the correlations between the drivers, and let Z denotes a vector of independent Wiener processes. As soon as the market drivers are simulated, the portfolio can be priced. One goes along all transactions and calculates the price of each of them; having the prices so determined one adds the prices together according to the applicable legal rights (if netting agreements are in place, prices are added together otherwise only the positive values are aggregated). The above can be formalized as:

$$p(t) = P(R_t, t, P_{t-1}, P_{t-2,} \ldots). \qquad (3)$$

Here, p is the price of the portfolio at any time t. The function $P(.\ ,t)$ defines the price of the portfolio using the $R(t)$ set of market drivers. The P function has explicit t dependence because of the portfolio, and is not continuous[2] in t. If collateral agreements are present, P is also conditional on its own previous values.

Considering the above one can easily see that even if a Wiener process is applied for market drivers (i.e. no jumps), the portfolio structure itself introduces non-continuity to the problem. This means that there is no easy way to forecast exposure profile points using previously calculated points[3].

Having completed the pricing of the portfolio once, one simulated value for the portfolio price distribution is obtained. For estimating the percentile of this distribution one has to repeat simulating market drivers and pricing the portfolio again and again.

Having the percentile estimated for one date point, one has to re-calculate the estimate of the percentile for a number of other date points in order to get the potential future exposure profile.

[2] Consider the P function just before and after a payment has been made.

[3] Except, when having additional information on the portfolio, such as there are no payments, etc.

It is obvious that this process can use immense computing power. Monte Carlo methods are not famous for their speed, and on top of that, the measure to be estimated is the expected shortfall and not the expected value; for estimating 95% percentile one uses only 5% of the simulated data.

The details of why as many simulations as possible are needed will not be addressed here, except to recall the adage: to halve the length of confidence interval four times the work is needed.

Why time steps are needed?

As already discussed the portfolio price function, which takes the input of market drivers and returns the price of the portfolio, is non-continuous. This means that there can be – and in fact there are – discontinuities when moving along the time axis. If the granularity of the time axis is generous and there is a value step-up and a step-down shortly after each other, there is a high probability that a peak is “missed”. Missing a peak is a danger as it may underestimate potential credit exposure. Also the fact that peaks appear on some days and not on other ones undermine the statistical robustness and the reliability of estimating the percentiles.

4. To *cut* a long story short

There are, generally speaking, two possible ways how to decrease run-time. Either one has to decrease the amount of work to be done or increase the capacity of the processing devices. Optimising the code, trying to make the Monte Carlo method more effective will fulfil the former; enabling the task to use more computing power is the solution to the latter.

Variance reduction techniques

There are existing techniques to increase the efficiency of the Monte Carlo (Glasserman et al. [2002]) however, the outlined problem is quite specific and does not behave like a ”typical Monte Carlo”:

- Random number generation is not an issue.
- The goal is to avoid – as far as possible – any calculation, not just speeding up the random number generator.
- Typically the re-simulation of market drivers and random number generation is less than 1% of the total workload for a portfolio where there are more than a handful of trades.

Also, for specific portfolios, some of the standard speed-up techniques may be deployed. It includes antithetic paths, control varieties, stratified sampling or importance sampling to mention just the most obvious candidates. However, as the Monte Carlo calculation is to be used to calculate credit exposure profiles for a variety of different portfolios with very different features, there is no simple recipe which ones should be used.

Increasing computing power

It is always tempting to increase the ‘efficiency’ by purchasing new processors, however each computer has a limit whereas the computing power that Monte Carlo estimates can use up is unlimited.

A better solution seems to be parallel processing, which cannot only use multiprocessor machines but also can be run on grid type of hardware configuration. Monte Carlo simulation is the perfect candidate for parallel runs. One has to study the task and determine how the simulation can be cut into smaller blocks. A natural selection would be that one profile point is calculated in one block, thus the whole simulation would be divided into blocks so that each block calculates the distribution of the potential portfolio price, does the statistics, and returns one number: the excess percentile of credit exposure. It is easy to implement, and no large data movements are necessary. However, it already has been pointed out that different date points are not necessarily independent in the case where collateral management is included in the calculation. Thus, it is necessary to divide the simulation space along another dimension: the number of simulations. The problem here is that one can not do the statistics inside the blocks but will need to store the simulated data, and there should be a block that collects all the data and does the final statistics.

To examine what is the condition for being it worth sending the calculation to the grid, let us break our processes into the following sections:

$$T_{total} = T_{init} + T_{sim} + T_{finish}. \tag{4}$$

T_{total} is the run time of the whole calculation, T_{init} is the time spent on initialising the task, T_{sim} is the time spent on the actual simulation, and finally T_{finish} is the time spent finishing the task, preparing the statistics. Going one step further let us break the simulation into N blocks as one wants to describe the situation on the grid. Rewriting the formula above yields the following:

$$T_{total} = \hat{T}_{init} \\ + N \cdot (\tau_{init} + \tau_{sim}(N) + \tau_{finish}(N)) + \hat{T}_{finish} \tag{5}$$

Here, $\hat{T}_{init}$ is the time that the whole process spends on initialising the grid computing components, whereas τ_{init} is the time that is needed to initialise one single block (reading input data for example). τ_{sim} is the time spent on the simulation inside the block and τ_{finish} is the time spent on finishing the process (saving the results to the network/database etc.); $\hat{T}_{finish}$ is spent on retrieving the results from the network or from a database ($T_{i/o}$) plus the time to finish the post processing (T_{post}), like calculating the statistics, preparing the output. Note, that τ_{finish} and τ_{sim} might depend on *N*. The dependence is thought to be inverse: in case of increasing *N*, obviously τ_{sim} (as $\tau_{sim} = T_{sim}/N$) and hence τ_{finish} should decrease. The reason for the second statement being true is that as the work the block is supposed to do is decreased, it is logical to assume that the amount of data processed is smaller as well. Here, the relation may not be that straight forward as for the simulation. However, it can very well be the case that for one third of work, the time spent on finishing the work is one half. The dependence of τ_{init} on *N* is thought to be minimal; hence the function argument is omitted from now on.

If all the blocks are running on the same machine (memory is shared) the saving of data at the end of each block (τ_{finish}) and the collecting of the data at completion of the whole process ($T_{i/o}$) can be saved. Thus, $\hat{T}_{finish} = T_{post}$ and $\tau_{finish} = 0$. This is the case when one block after the other is to be run on the same machine[4]. On top of all these, for one machine calculation the initialisation of the grid can be also skipped: $\hat{T}_{init} = 0$.

Another special case will be running several blocks on a grid of computers/processors where the number of processors, *M* is greater than the number of blocks, *N*. Also, for the sake of simplicity let us assume that the speed of all processors is the same. Then,

$$T_{total} = \hat{T}_{init} + (\tau_{init} + \tau_{sim}(N) + \tau_{finish}(N)) + \hat{T}_{finish} + N \cdot T_{grid} \qquad (6)$$

First, the whole task has to be initialised then it has to be cut into *N* pieces. Each piece is running on a different machine, hence the waiting time is the completion time of one block, then the results are to be collected and the statistics done.

The last part of the formula T_{grid} contains all the time that is spent managing the grid[5], communicating between the processors, etc. This is increasing when the number of blocks is increasing, however, normally still stays relatively small.

Pros and cons

To be able to compare single block calculation with grid computing let us recall the general run-time equation (5) and the grid run-time formula of equation (6). For the single block calculation one can write:

$$\begin{aligned} T_{total} &= T_{init} + T_{sim} + T_{finish} = \\ &= \tau_{init} + N \cdot \tau(N)_{sim} + T_{post} \end{aligned} \qquad (7)$$

Running the whole calculation in one bunch has one advantage that is worth mentioning: skipping the *collecting* bit of $\hat{T}_{finish}$ that is $T_{i/o}$.

$$T_{finish} = T_{post} = \hat{T}_{finish} - T_{i/o} \qquad (8)$$

Let us have a closer look at equation (6) and equation (7). This is to determine whether it is worth breaking a task into blocks and running it on the grid, versus running it as

[4] Note, that here it does not make much sense breaking up the process into more than one block, as for each block τ_{init} time is needed to get the block initialised, and the simulation does not take any shorter breaking it into many blocks as one follows the other one.

[5] The more blocks one has the more workload on the grid is generated; that is why we have $N \cdot T_{grid}$ in the formula. That work includes the communication between the machines, auditing the processes, querying the progress made on the jobs, etc.

one task. Comparing the two equations the condition of using the grid boils down to the following:

$$\hat{T}_{init} + T_{i/o} + N \cdot T_{grid} + \tau_{fin}(N) < (N-1) \cdot \tau_{sim}(N) \tag{9}$$

Just a quick look on the formula yields the following remarks: the left hand side contains costs of the grid whereas the right hand side the benefits. Supposing that the number of machines on the grid M is large enough not to violate the assumption of $M > N$, and $\tau_{finish}(N) << \tau_{sim}(N)$; now it is beneficial to increase the number of blocks. However, there is a limit (apart from the number of processors on the grid) as while N is increasing, $\tau_{sim}(N)$ is decreasing faster than $\tau_{finish}(N)$.

Away from ideal

Let us summarise the assumptions made about the grid so that the above equation can be derived: *(i)* the number of computers available is large *(ii)* all computers are the same *(iii)* each block to be sent to the grid contains equal amount of work. In the following section let us try to come up with formulas for the less ideal case.

Relaxing: number of computers is large

If N gets larger than M, then it is not enough to wait for one block to finish, as there are not enough processors to distribute all the blocks. Hence, the equation (6) is to be modified as:

$$\hat{T}_{init} + T_{i/o} + N \cdot T_{grid} + \zeta_N \cdot \tau_{fin}(N) + (\zeta_N - 1) \cdot \tau_{init} < (N-1) \cdot \tau_{sim}(N) \tag{10}$$

and

$$\zeta_N = 1 + \mathrm{int}\left(\frac{N-1}{M}\right). \tag{11}$$

Conceptually this means that it is not enough to wait for only one block, but for ζ_N blocks. Also, note that not only the finishing bits are present on the cost side of the equation, but also the initialisation (τ_{init}) bits, that is due to the fact, that if two or more blocks are running after each other, they have to be initialised separately. It is easy to see that the optimum for run-time on the grid is when $N = M$, supposing that $\tau_{finish}(N) < \tau_{sim}(N)$ still holds. However, if N is just one bigger than M, then the completion time nearly doubles. One needs to wait

$$\tau_{fin}(N+1) + \tau_{init} + \tau_{sim}(N+1) \tag{12}$$

more, as the last block will run on a single machine while the other machines will be idle. This cost in run-time can be substantial unless one is careful with the issue.

If $N = 2M+1$, the situation improves, as the idle machines will be waiting for completion of the last task only for shorter time:

$$\tau_{fin}(2N+1) + \tau_{init} + \tau_{sim}(2N+1) \tag{13}$$

as the workload of one task decreases if we increase the number of tasks. The simulation part for example:

$$1/2\,\tau_{sim}(N) = \tau_{sim}(2N)\,. \tag{14}$$

Thus, breaking one task into many blocks seems to be a good idea, as the residual tasks that one might have to wait for are shorter, however, increasing the number of blocks costs run-time (managing the blocks on the grid, initialising each block, etc.).

If the number of machines on the grid fluctuates, or it is not known in advance, the tasks can be split into $N >> M$ blocks in order to minimise this residual cost (one or a few blocks remaining running at the end). Of course one also has to keep in mind the additional cost of doing so. This would determine the upper boundary of N.

In the following section we focus our attention on a grid where the computing units are not the same anymore. Here, we have to address the issue as setting $M = N$ does not solve the problem any more.

Relaxing: all computers are the same

The result of equation (10) holds, because of the assumption that all computers are the same in terms of computing power. Relaxing this assumption brings two issues to the fore: *(i)* the need of measuring the performance of each computer on the grid *(ii)* the ability of the grid to distribute task blocks accordingly.

The question is not only whether it is worth running the task on a grid or not. It is also the question how to schedule the blocks in such a way that the tasks finish as soon as possible. The scheduler should be able to estimate workload on each processor on the grid, and based on this set of information the scheduler should decide how to distribute the blocks of tasks.

First a function that will measure how many blocks are completed in a given time is constructed. Ignoring the time cost of maintaining the grid, the following can be written:

$$\Xi(t) = \sum_{i=1}^{M} \mathrm{int}\left(v_i \cdot \frac{t}{w(N)} \right), \tag{15}$$

where w is the time consumption of one block

$$w(N) = \tau_{init} + \tau_{sim}(N) + \tau_{finish}(N) \tag{16}$$

and v_i is the speed (CPU power) of the processor for the i-th machine on the grid (for i=1…M). Let us illustrate the formula above on the figure below (see fig. 2.). It depicts four servers, each running on different speed, so for each of them calculating the unit block takes different times (time is denoted by t on the horizontal axis).

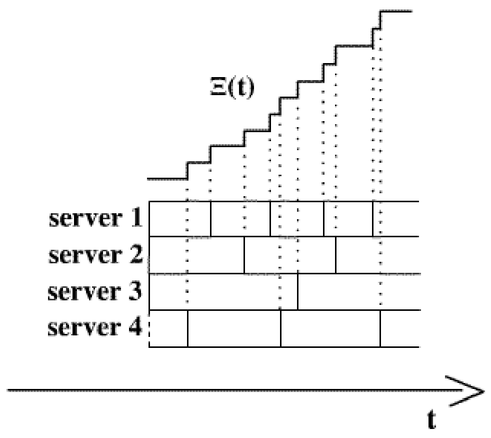


Figure 2: Step function describing how many blocks have been finished on the grid by time *t*. The blocks plotted next to each server, symbolise one unit of task.

Now, let us write $\Xi(t)$ in the following form:

$$\Xi(t) = \frac{\langle v \rangle_{i=1,M}}{w(N)} \cdot t + \vartheta(t, M, \ldots) \\ = \Xi'(t) + \vartheta(t, M, \ldots) \tag{17}$$

This would then translate into the following figure:

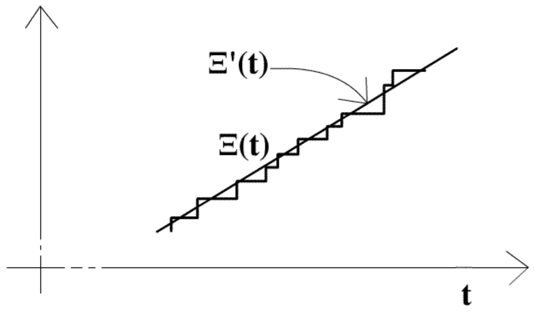


Figure 3: The exact and 'average' step function is plotted against time describing how many tasks have been finished by a given time

The symbol $\langle v \rangle$ means the average of the machine speed on the grid, and the term ϑ incorporates all fluctuation around the linear term. Thus, on average the time needed to get another block finished is exactly $w/\langle v\rangle$.

Going back to the criteria for breaking up the task one finds that as far as the formula below holds

$$T_{grid} + \frac{\tau_{fin}(N) + \tau_{init}}{M} < \frac{\langle v\rangle}{w(N+1)} - \frac{\langle v\rangle}{w(N)}, \tag{18}$$

it is worth increasing the number of blocks, as one gains more on average having shorter blocks than loosing on increasing the cost of doing so on the grid. The context of this formula is probabilistic, as it is not known in advance how the scheduler is going to schedule the blocks, because of the inhomogenity of the grid. Note, that ζ_N is replaced with a smooth term such a way that the expected value of the difference between the original and the new term is zero:

$$\left\langle \zeta_N - \frac{1}{2} - \frac{N}{M} \right\rangle = 0. \tag{19}$$

Blocks: not equal workload

The last case left to handle is the case when one cannot guarantee that the workload of the blocks of tasks remains the same. Most of the results from the last section remain however true. There is a trick with which one can decrease the run-time easily: if the workload of each block is known in advance, the large blocks should come first, and the small ones later. This way it can be ensured, that until the very last seconds the full computing power of the grid is utilised, and it is not likely that the completion of the task waits only for one of the largest blocks to finish[6]. Scheduling the jobs this way is similar to packing a suitcase: big shoeboxes come first, filling up the gaps with socks last.

[6] Just by rearranging the order of subtasks, at EBRD a 20% decrease in run-time was realised.

How many jobs finishing on the same time?

It is interesting to note, that the fluctuating part of

$$\zeta_N - \frac{1}{2} - \frac{N}{M} \tag{20}$$

is known quite well[7]. If the system is random enough the distribution of the time needed to get another task finished is described by the so-called Poisson statistics[8] developed in the framework of *Random Matrix Theory* [Mehta 1967].

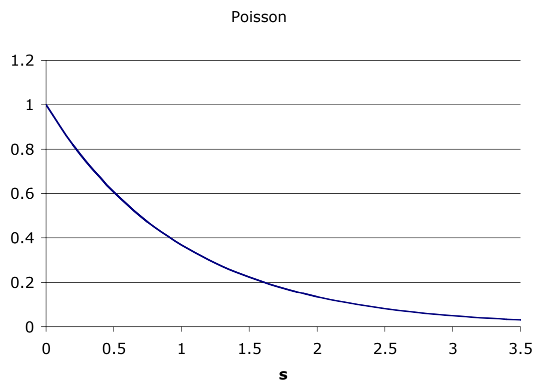


Figure 4: Decay of the Poisson spacing statistic

As seen on the density function of the Poisson statistic most of the time intervals are close to zero and for large *N* and *M* it will have a substantial effect on the network traffic. However, the tail of the distribution is decreasing more slowly than the normal distribution, thus relatively large time lags between two subtasks finishing after each other still occur. If the system is

[7] There are a few assumptions required for this result. The speed of the computers has to be 'heterogeneous enough' meaning that the speed of the computers should be different, also the size of the blocks should vary.

[8] The time between two blocks finishing on the grid has the same statistical properties as e.g. the energy level differences (spacing statistics) of a class of one dimensional quantum graph.

homogenous (lot of computers with the same computation capacity/speed on the grid), the distribution will be even more concentrated around zero. This shows that even in case of a heterogeneous system (different computers, different size of blocks) there will be times when many things will happening at once, like many jobs finishing at the same time (bad news for the grid/network) followed by calmer times when the capacity of the grid will not be fully utilised. The result is warning us that the whole construction of the grid (network, broker, etc.) should be designed such a way to be able to cope this clustered workload peaks. Another way would be to implement such mechanism that would repulse those finishing times that are too close to each other in order to smoothen out the workload on the grid over time.

5. Summary

One of the problems when generating potential future exposure profiles for complex portfolios lies in the huge amount of simulations to be performed. The issue can successfully be tackled by deploying grid-type computer architecture. It has been shown that the number of computers connected to the grid contributes to the performance in a step function manner. It was also demonstrated that the speed of each individual computer has a nonlinear impact on the speed of the grid. The last but not least point made confirms that the order in which the subtasks are assigned to the grid has a substantial effect on the total speed. All together the paper demonstrates that using a grid for simulating potential future exposure profiles is an efficient and feasible approach and the associated difficulties are not difficult to surmount.

6. References

Hull, J.C. [2002]: *Fundamentals of Futures and Options Markets*, Prentice-Hall International Inc, London.

Hull, J.C. [1993]: *Options, Future and other Derivative Securities*, Prentice-Hall International, Inc.

Duffie, D. [2001]: *Dynamic Asset Pricing Theory*, Princeton University Press, Princeton and Oxford.

Rebonato, R. [1998]: *Interest-rate Option Models*, Wiley, Chicester.

Jarrow, R.A. [1996]: *Modelling Fixed Income Securities and Interest Rate Options*, McGraw – Hill, New York.

Glasserman, P., Heidelberger, P. and Shahabuddin, P. [2000]: Variance Reduction Techniques for Estimating Value-at-Risk, *Management Science*.

Mehta, M.L. [1967]: *Random Matrices and the Statistical Theory of Energy Levels*, Academic Press, New York and London.