Abstract

In this scientific research we present our algorithm for finding a low RWA value. Firstly, we explain the meaning of risk-weighted assets and why this term is important. After that, we describe the method we used to solve the problem. Text is accompanied by charts, which we extracted from the data set. Next, we explain the implementation and complexity of the algorithm. Finally, we analyze the results and discuss the benefits and faults of our approach.

Key words: asset, liability, risk-weighted assets, netting set, optimization, algorithm, banking

1 Risk-weighted assets (RWA)

Risk-weighted assets determine the minimum amount of capital that must be held by banks and other financial institutions in order to reduce the risk of failure. The goal is to prevent banks from losing large amounts of capital when a particular asset class decreases sharply in value. When calculating the risk-weighted assets of a bank, the assets are first categorized into different classes based on the level of risk and the potential of incurring a loss. The banks' loan portfolio, along with other assets such as cash and investments, is measured to determine their riskiness.

2 Method

The first thing we did when we read the problem was plotting the data set on charts, which revealed an interesting property: over 75% of contracts in the data set had interest rates in the interval $[1.65\%, 1.80\%]$, as you can see from Figure 1. The length of this interval is 0.15%, which is exactly our margin for interest rate difference. From that we concluded that it is possible to put any subset of these 75% of contracts into a single netting set.
That fact motivated us to split the contracts into 0.15% long intervals, containing as many contracts as possible. This idea is applicable to data sets in which contracts are nicely separated into such intervals, that is, the values of interest rates are dense, converging around a few values, which is a justified assumption for the set of bank contracts (most of bank contracts have close interest rates).
Let us call a set of contracts, whose interest rates are in one of such intervals, a *cluster*. Any two contracts in the same cluster can be placed in the same netting set. Following this idea of finding clusters that are as large as possible, we made 32 clusters. The largest among them contained 772 contracts.

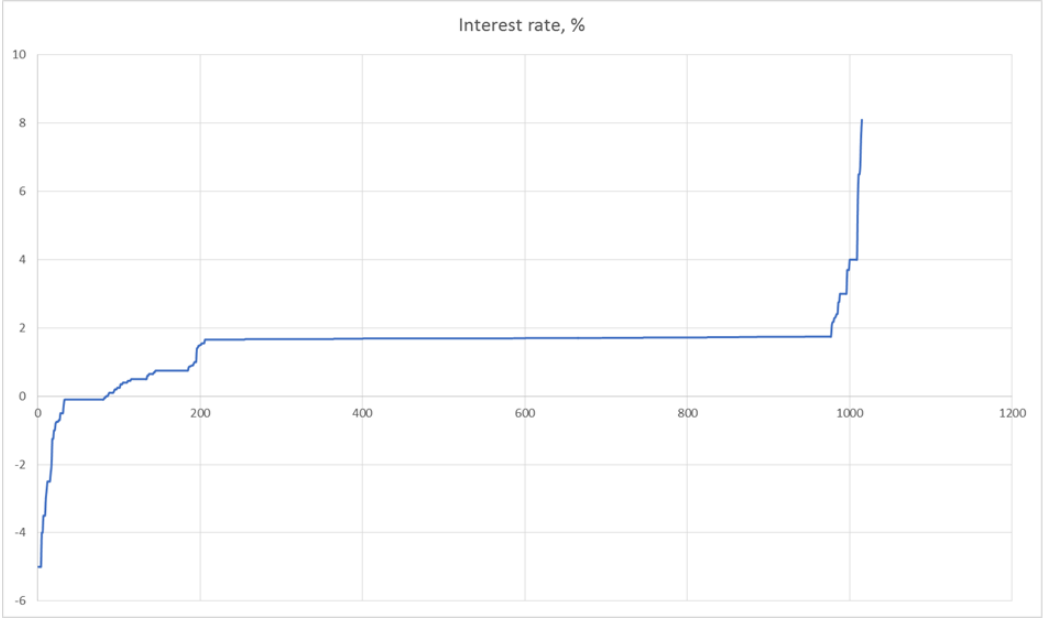

Figure 1

In a similar way, we got clusters for the Tier 2 problem. A similar fact about converging interest rates (that is, dense interest rates), as for the Tier 1 data set, holds true also for the Tier 2 data set. We can see this from Figure 2, where we can clearly see a line forming. In this case a cluster is defined both by an interval of interest rates of length 0.15% and by an interval of days of length 30. In a cluster defined this way, once again, we can choose any two contracts to be in the same netting set. This way, we reduced both Tier 1 and Tier 2 problems onto the same problem of solving clusters.

The rest of our algorithm revolves around something we call *zebras*. Let us split the assets and liabilities of each cluster into two arrays, one for assets and one for liabilities, sorted by their values. We define positive and negative zebras separately. We get a positive zebra using the following algorithm:

1. Choose the asset with the largest value.
2. Repeat while possible:
   (a) If the previously chosen contract was an asset, choose a liability contract with the largest value which hasn't been chosen previously, such that its value is smaller than the value of the last asset
   (b) If the previously chosen contract was a liability, choose an asset contract with the largest value which hasn't been chosen previously, such

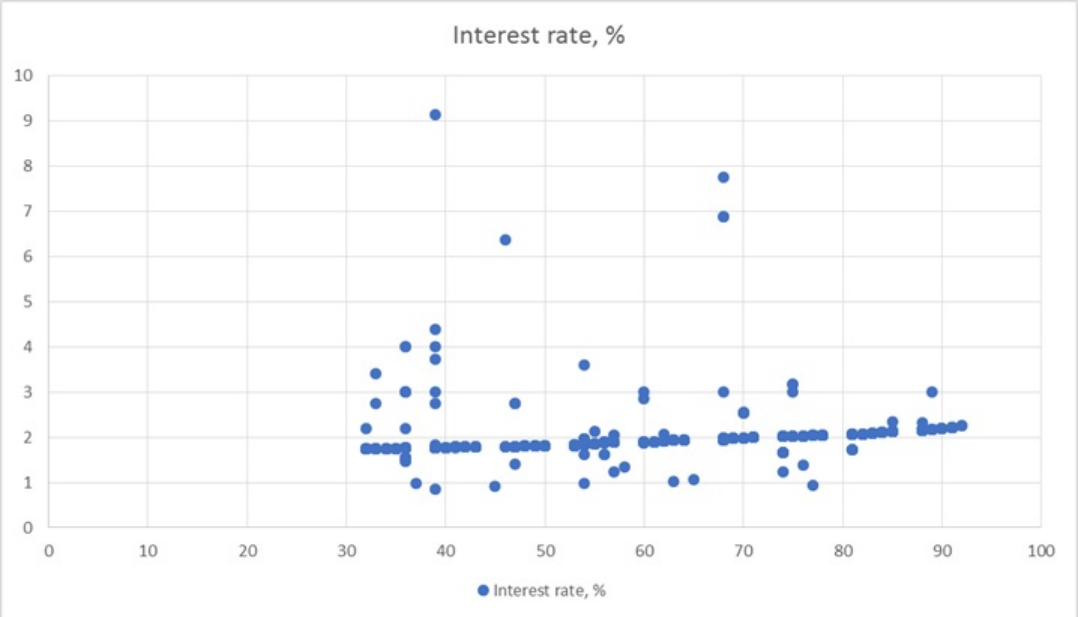

Figure 2

that its value is smaller than the value of the last liability

A negative zebra is defined in a similar manner, by choosing a liability as the starting contract, instead of an asset. In a positive zebra, it is easy to show, that the sum of assets is larger than the sum of liabilities. The opposite holds true for negative zebras. Each zebra will be a netting set on its own. The intuition behind the idea of making zebras, is that, given a uniformly distributed set of contracts (based on their values), the residual asset (or liability, if using a negative zebra) will be relatively small, because in each iteration of step 2.(a), the asset and the liability cancel (net) each other out very well. In that way we get a low first summand in the RWA formula. The way to get a low second summand, is to alternate between creating positive and negative zebras. Given that the contracts' values are somewhat uniformly distributed, we can realize that in step 2.(a), the difference between the asset and the liability will be similar in size, no matter if we're creating a negative or a positive zebra. It follows naturally, then, that the residual asset and the residual liability from consecutive zebras are similar in size, therefore partially canceling each other out in the second summand of the RWA formula.

3 Implementation and Complexity Analysis

Our algorithm was implemented in C++. For complexity analysis purposes, let's denote the number of contracts with N and the number of clusters with C. The memory complexity is $O(N)$ for the entire algorithm, so from now on we'll denote time complexity as only complexity. For the partition of contracts into clusters, we will use a very simple greedy algorithm. Let's pick an interest rate value that will be the beginning of the current cluster and see how many contracts would fall into that interval. We repeat this process for every possible starting value. Then we pick the interval that contains the most contracts, and remove those contracts. We do this until no contracts remain. The complexity of our implementation is $O(CN^2)$, which can easily be optimized to $O(CNlogN)$ for the Tier 1 problem, using the binary search algorithm. Given that the interest rates in our data set are very dense, C is much smaller than N, giving us a useful complexity, that can be applied to even larger (dense) data sets. The construction of zebras has been done using two Binary Search Trees (implemented in C++ Standard Template Library as sets), one for assets and one for liabilities. The complexity of constructing all zebras is $O(NlogN)$ amortized, because we choose each contract once, and to add or remove a contract from the Binary Search Trees, we need $O(logN)$ complexity. Since the slowest part of the algorithm is constructing clusters, the overall time complexity is $O(CN^2)$. The results were checked using a separate checking code, which used simple brute force to see whether all netting sets were valid and the result correct.

4 Results

By following the algorithm of alternating positive and negative zebras for each cluster, we got the following results:

$$RWA = 616484289.876375 \text{ for the Tier 1 problem and}$$
$$RWA = 1714288868.451210 \text{ for the Tier 2 problem}$$

constructing 32 clusters in both cases. Number of netting sets used in Tier 1 was 566, and 320 for Tier 2. This result in both cases is similar in order of magnitude as the value of the largest contract. For both Tiers, program finished execution in less than half a second.

5 Discussion

This algorithm works especially well when the interest rates of contracts are dense and the values of contracts uniformly distributed. Both of these are reasonable

assumptions for a set of banking contracts. In cases where the interest rates are uniformly distributed, our approach is not applicable, but such a case is improbable in banking.

References

[1] https://corporatefinanceinstitute.com/resources/knowledge/finance/risk-weighted-assets/

[2] https://www.investopedia.com/terms/r/riskweightedassets.asp