Team Members

1. Thongchai Wirojsakseree - Group Leader
2. Z L (Julian) Lim

FIS Project 2016

Failure Propagation Web App

---

1 Introduction

In this project, we extend the work on simulating failure propagation as presented in the lectures. We focus on the problem that extrapolations of bank stability from the study of a single entity is usually inaccurate with respects to the entire complex system. We aim to visualise the propagation of bank failure over variable inputs and to make this process easily available to clients. To that end, we created a web application in HTML5 and Javascript, with the main functions of allowing users to manipulate parameters such as bank data and triggering bank failure, then see the propagation of failure via simple and easy to understand charts and figures. This allows users to customise estimates of losses and propagation for specific situations which may be more pertinent to them, and also to set up customizable experiments.

2 Application Requirements

In the planning phase of the project, the following functional requirements were set forth:

1. The application should be able to simulate failure propagation for a network of up to 10 banks.
2. Users should be able to specify the banks in (1).
3. Users should be able to specify the banks that initially fail.
4. Users should be able to set the individual exposures of banks to each other.
5. Users should be able to set the threshold of individual banks.
6. Users should be able to do (4) and (5) either by uploading csv files, linking to urls, generating random numbers, or manually typing in the figures.

7. Following the setup of parameters, users should be be able to view interactive charts of each bank and the total network for 2(?) scenarios.
8. The charts should collectively present the following information:
    a. Losses (individual and total)
    b. Difference between loss and threshold
    c. Failed status (bank failed or not)
    d. Propagation status (total number of banks failed)
9. Users should be able to navigate freely and alter the simulation by changing input parameters.
10. Users should be able to customise the appearance of the charts.
11. Users should be able print and download the charts in various formats.

3 Failure Propagation

In a world of increasing interconnectivity, studying and predicting possible failure propagation which will result in market loss of billions is of utmost importance. Failure propagation is where the stability of multiple banks are compromised in a cascading manner, usually catalysed by a single major incident, such as that of the 2008 crisis. The entire project is founded on the premise that we cannot determine the behavior of a complex system from the behavior of a single entity within that system. In other words, we have to simulate the entire system to fully understand the effects of incidents. Identically, we cannot properly predict failure propagation by only examining one bank, hence we have to simulate a network of banks.

However, simulating systems is very technical, and not everyone can easily create a simulation of what their decisions as financial analysts to huge firms will have as impact to a larger network, for instance. Yet, it is imperative that financial experts understand fully the consequences of their actions. It is of no coincidence that the Chartered Financial Analyst program kicks off their syllabus with a hefty chapter on ethics. Hence, to allow better understanding and prediction of future crises brought about by systemic risk and failure propagation, our solution is a client friendly, browser based simulation engine.

Simulation of failure propagation was constructed pursuant to material in lecture 7. We are interested mainly in predicting if the trigger failure of a bank or banks will cause other banks to fail and thus culminate in the propagation of failure. Banks fail when capital losses exceed their individual threshold. Threshold is an asset figure which one can think of as the amount of shock the bank can absorb in terms of losses before it collapses. In our simulation, banks are related to other banks by the net amount of exposure or loan to the other banks, and banks make losses only by the loss of receivable credit when a bank that owes debt to it collapses. For example, if Barclay's has a net debt of £100 Million to Santander, and Santander files for bankruptcy, Barclay's loses the receivable £100 Million as Santander defaults on its debt. If Barclay's has a

threshold of £50 Million, this loss is 100% over the threshold, and as a result Barclay's collapses as well. We say that failure has propagated from Santander to Barclay's.

Prior to simulation, the necessary input parameters are the actual banks themselves, the threshold of each bank, and the amount and direction of exposure between each bank. By direction of exposure, we mean who has net debt and who is overall creditor. The last necessary input is the trigger failures, ie the banks which initially fail, setting off the propagation.

4 Application Design

We elected for our application to be web-app for the simple reason of being easily accessible to all operating systems, browsers, and devices. In addition, for non-technical users, this is not required any programming skills to receive the results. As is standard with all web applications, the programming languages are HTML5, the newest standard for marking up and laying out a webpage, javascript, one of the most popular scripting languages which allows interactions with users, animations, behind the scenes calculations, CSS3, de facto style language, and other frameworks such as Jquery and AJAX. With these languages, it is possible to include thousands of free world-class standard frameworks for fantastic visualizations such as Highcharts which is mainly used in our web-app.

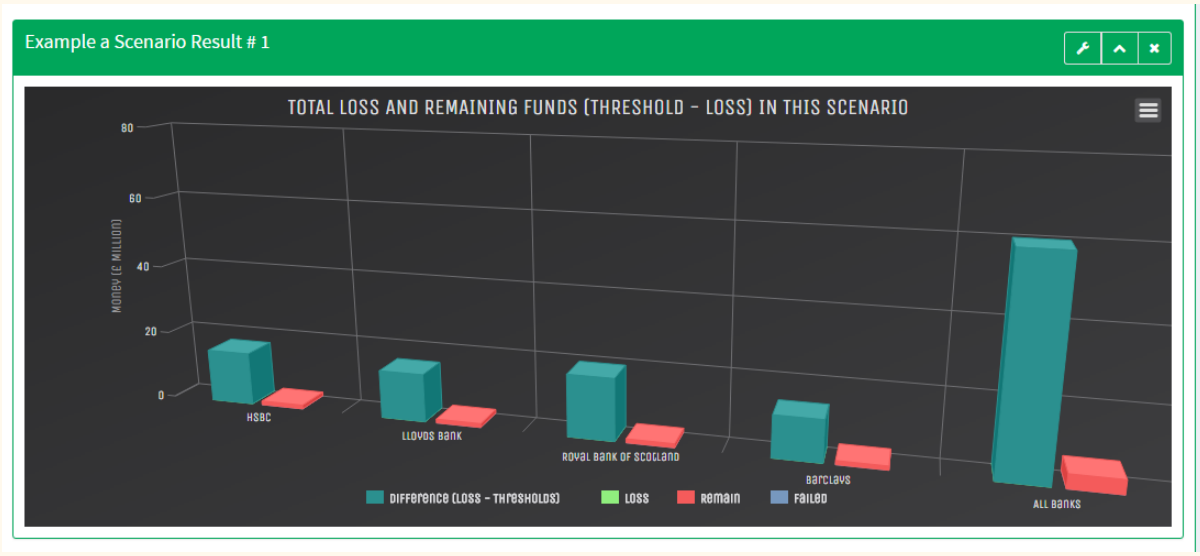

Figure 1 : The example result chert of the website.

We bootstrapped our application for the main purposes of aesthetics and mobile-friendliness. Bootstrapped applications scale properly to screens of any size, and thus do not require extra work and maintenance when posting from a smartphone to an HDTV, for instance.

No servers were required and hence no server side scripting was required as all calculations can be done by clients' web browsers such as Google Chrome and all inputs came from users so a database was not required. However, the temporary host is GitHub, http://chai41104.github.io.

In this project, there are two modes, Specific scenario, and All scenarios. In Specific scenario mode, the failure bank's parameter is given as one of the inputs to calculated the damage of that scenario on the financial system in step by step. While, All scenarios mode, the probability of the failure banks are given to calculate the probability of losing money in the financial system. In this mode, all scenarios are calculated and sum up at the end. It is good to inform that with more than two thousand lines of code, it is impossible to explain all of the code, showing with the attach files, in details. Hence, only the only main part of calculation will be explained here.

In the Specific scenario, four inputs are given, the number of banks for testing, FailedNow array to indicate which banks are failed, Exposures matrix to show how much money of each company owns by others and Threshold to show maximum limited loss of each company. With these inputs are passed to the main function, named mainCalculation. In the function, there is only one main loop inside that will stop when there are no new company fails. In each round of the loop, three main things to do. Firstly, failurePropagation is called by passing two parameters, failed parameter and the exposures matrix. In this function, the multiplication of matrix is called between the two inputs. As a result of that a loss matrix or array, which one dimension is only 1. It shows how much money each bank loses if some banks are failed to indicate in the failed parameter. Secondly, a function named PWComparison is called to check that which banks will fail due to two inputs, lose from the previous process and thresholds. Each bank will fail if their losses exceed their thresholds. This function is returned new failed parameter. Finally, PWOr function is called to merge the previous and the new failed parameter together. If there are some new companies failed, the main loop needs to continue. Otherwise, the function is done with the list of all failure banks. At the end, the results are illustrated to the users via charts. Some important results are such as which banks are failed, how much money loses for the whole system. If all banks are merged together, how much money requires to maintain the system.

```
function mainCalculation(failedNow) {

    var loss;

    do {
        // At the beginning, setting that failedIndicator is failedNow.
        var failedIndicator = failedNow;
        // Calculates what banks will fail.
        loss = failurePropagation(failedIndicator, exposures);
        // Calculattes what companies will fail.
        var failed = PWComparison(thresholds, loss);
        // merging the new result and old result together.
        failedNow = PWOr(failed, failedIndicator);
    // looping until no new failed bank.
    } while(failedMore(failedIndicator, failedNow));

}
```

Figure 2 : The simple version of mainCalculation function.

```
// point-wise comparison function for comparison that the banks are bankrupted or not .
function failurePropagation(failedIndicator, exposures){
    var loss = [];
    // Calculating in each bank.
    for(var i = 0; i < exposures.length; ++i) {
        var count = 0.0;
        // Calculating how much money that each bank losses.
        for(var j = 0; j < exposures[i].length; ++j) {
            // If other banks fail, this bank losses some money.
            if(failedIndicator[j] == 1) {
                // sum the loss.
                count += exposures[i][j];
            }
        }
        loss.push(count);
    }
    return loss;
}
```

Figure 3 : The simple version of failurePropagation function.

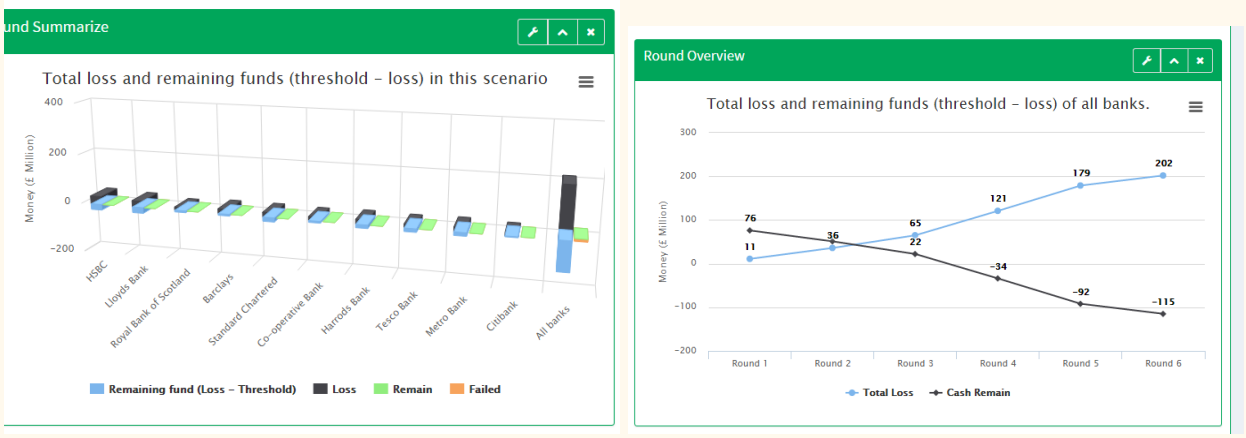

Figure 4 : The example result of Specific scenario mode.

All scenarios mode provides the probability of each situation in the system. In this mode, it is similar to the Specific scenario mode. It requires the Failure Probability of Banks instances of the failed parameter. In the main process, the new permutation function is added. This function is simply to generate all possible failed scenarios using the recursive technique. In each scenario, the main calculation is called as same as the previous mode to generate some statistics such as the money loss. In addition, probScenario is called in each scenario to calculate how likely in each scenario will happen based on the Failure Probability of Banks. Simply, this function uses a calculation of probability, the result is based on the multiplication of the probability of each bank, failureProb when that bank is failed and (1-failureProb) when that bank doesn't fail. At the end, the results are illustrated to the users via charts. Some important results are such as how likely the system will lose money exceed 20 million pounds, how much money the system needs for the probability more than 10 percent.

```
// This do permutation.
function permutation(failed, index, workingFunction) {
    // Need to be more than 0, index of array can't be negative.
    if(index >= 0) {
        failed[index] = 0.0;
        permutation(failed, index - 1, workingFunction);
        failed[index] = 1.0;
        permutation(failed, index - 1, workingFunction);
    }
    else {
        // call mainCalculation.
        var totalloss = mainCalculation(failed);
    }
}
```

Figure 5 : The simple version of permutation function.

```
// probScenario function is calculated probability of each scenario.
function probScenario(failureProb, failed) {
    var prob = 1.0;
    // Calculating in each bank.
    for(var i = 0; i < failureProb.length; ++i) {
        // if banks not failed.
        if(failed[i] < 1.0) {
            prob *= (1-failureProb[i]);
        }
        // if banks failed.
        else {
            prob *= (failureProb[i]);
        }
    }
    // How likely of this scenario.
    return prob;
}
```

Figure 6 : The simple version of probScenario function.

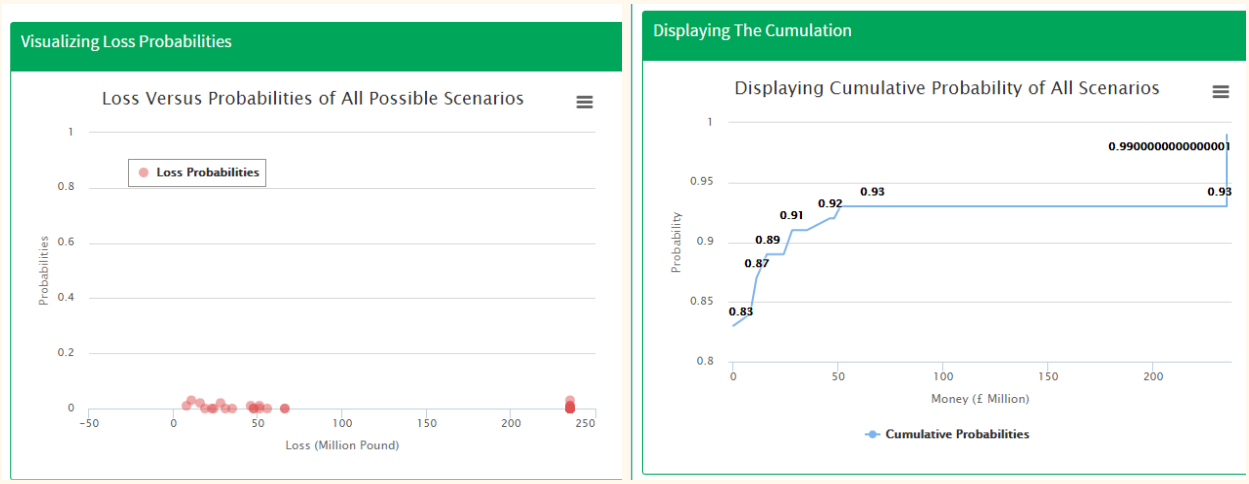

Figure 7 : The example result of All scenario mode.

The application was tested in many random test sets. All programs were also passed the unit test during and after the developing period. For the users, they can simply test the web by using our sample test in exampleInputFor10 file, providing all input parameters, as an input via URL. In this input, it requires that 10 banks at the beginning.

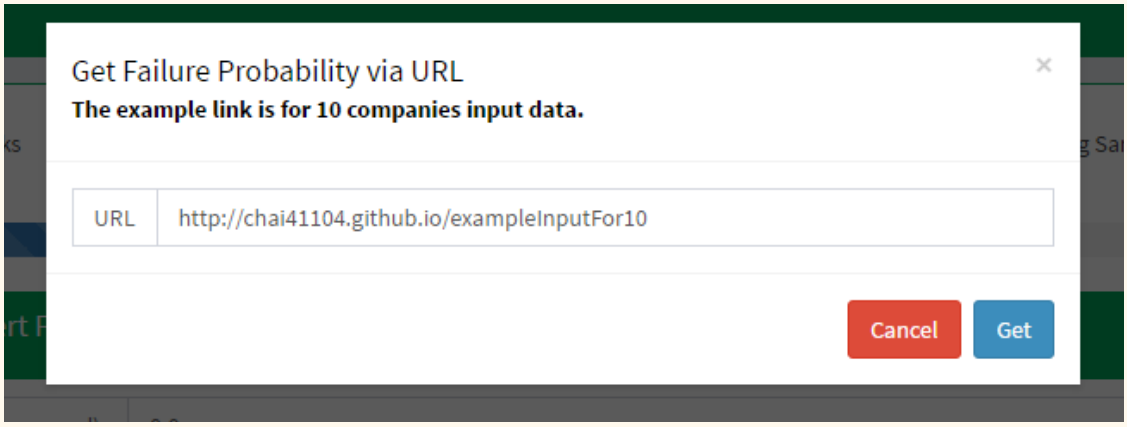

Figure 8 : Importing example input data via URL.

5 Future Implementation

In this web-app, the maximum of banks is limited as 10 banks due to two main reasons. Firstly, it is not possible to illustrate the result of many banks, such as 100 banks, in the same chart at the same time. This may do by exporting the result to third party software. Secondly, as a nature of Script Languages such as JavaScript, the running time is significantly slower than some computer languages such as C or Java. So, it may take a long time to get the result of 100 banks as the input. However, this can be solved by sending the inputs to the third party such as servers and receiving the output afterward in order to visualize the results to the clients.

6 Conclusion

As the requirement of assignment 3, we have identified a financial problem being the improper modelling and lack of awareness of systemic risk in bank failure propagation. In response, we created an easily usable and accessible web app which allows users to customise simulations of bank propagations. This will, in turn, allow them to better understand such propagations, make predictions, and conduct their own research.

References

1. Highcharts.com. (2016). *Interactive JavaScript charts for your webpage | Highcharts*. [online] Available at: http://www.highcharts.com/ [Accessed 4 Apr. 2016].

2. Mark Otto, a. (2016). *Bootstrap · The world's most popular mobile-first and responsive front-end framework.*. [online] Getbootstrap.com. Available at: http://getbootstrap.com/ [Accessed 9 Apr. 2016].

3. Serguieva, D. (2016). *Financial Information System #7*. 1st ed. [ebook] Available at: https://moodle.ucl.ac.uk/pluginfile.php/3307209/mod_resource/content/3/FIS2016lecture7.pdf [Accessed 4 Apr. 2016].