Network Contagion: Documentation

Francisco Ruela

July 26, 2019

Contents

1 Background

1.1 Relevant Definitions and Formulas

This section is meant to be a quick reference for the terms used in examples. All terms are defined in Glasserman and Young (2014). This is not an exhaustive list. If you are going into this for the first time I would strongly recommend reading Eisenberg and Noe (2002) and Glasserman and Young (2014) to get the background. Of course I am also assuming that you read the current draft or final version of Duarte and Jones that this work is for.

Nodes: $\mathbf{N} = 1, 2, \ldots, n$
Liabilities Matrix: $\overline{P} = (\overline{p}_{ij}), \overline{p}_{ij} \geq 0, \overline{p}_{ii} = 0$
Outside Assets Vector: $(c = c_1, c_2, \ldots, c_n) \in \mathbb{R}^n_+$, $c_i \geq 0$
Outside Liabilities Vector: $(b = b_1, b_2, \ldots, b_n) \in \mathbb{R}^n_+$, $b_i \geq 0$
Total Assets$_i$=$c_i + \sum_{j \neq i} \overline{p}_{ji}$
Total Liabilities$_i$: $\overline{p}_i = b_i + \sum_{j \neq i} \overline{p}_{ij}$
New Worth$_i$: $w_i = c_i + \overline{p}_{ji} - \overline{p}_i$
Financial Connectivity$_i$: $\beta_i = \frac{\overline{p}_i - b_i}{\overline{p}_i}$
Relative liabilities Matrix: $A = (a_{ij})$ where

$$a_{ij} = \begin{cases} \overline{p}_{ij}/\overline{p}_i, & \text{if } \overline{p}_i \\ 0, & \text{if } \overline{p}_i = 0 \end{cases}$$

Shock Realization: $x = (x_1, x_2, \ldots, x_n) \geq 0$
Clearing Vector: $p_i(x) = \overline{p}_i \wedge \sum_j (p_j(x) \cdot a_{ji}) + c_i - x_i$
Default Set: $D(p(x)) = \{i : p_i(x) < \overline{p}_i\}$
Systematic Impact of a Shock $= |x_i| + S(x)$ where

$$|x_i| = \sum_i x_i \text{ and } S(x) = \sum_i (\overline{p}_i - p_i(x))$$

Shortfall: $s_i = \overline{p}_i - p_i$ where

$$\begin{cases} s_i > 0, & \forall i \in D \\ s_i = 0, & \forall i \notin D \end{cases}$$

Shortfall Equation: Let subscript "D" designate vectors and matrices restricted to the default set D.

$$s_D = s_D A_D - (w_D - x_D) \Rightarrow s_D = (x_D - w_D)[I_D + A_D + A_D^2 + \cdots]$$

Depth of Node i: $u_D(x) = [I_D + A_D + A_D^2 + \cdots] \cdot 1_D$, $u_i(x) = 0 \forall i \notin D$
Network Volatility Index: $NVI = 1 - B = \frac{1}{1-\beta^+} \cdot \frac{\sum_i \delta_i c_i}{\sum_i c_i}$
Loss Equation: $L(x) = \sum_i (x_i \wedge w_i) + \sum_i ((x_i - w_i) u_i(x))$

1.2 Code Summary

At an mile-high level, the code takes in limited information about nodes and uses that to find the best and worst possible networks by total loss or (connected loss)/(unconnected loss). The main scripts are run_simulations_TESTS.m, knitro_helper.m, and simulation_ratio.m

1.2.1 run_simulations_TESTS.m

NOTE: The script run_simulations_TESTS.m is used for manual specifications. You want to use the non-test version in all other scenarios.

This is the executable script that calls the other two function. This document reads in the data, prepares it for the optimization (so some data manipulations, quick error/logic checks, setting up parallel processing etc), runs the specified optimization (there are a handful of setting that impact how the optimization itself is run), then takes the optimized data and creates the network diagram.

When specifying the data, a number of fields are required, while some others can be optimized over. The necessary parameters are $\overline{p}_i$ (total liabilities) and total assets. The code labels total assets as a, take care not to confuse this with the elements of the relative liabilities matrix. Since we have both total liabilities and assets, w_i must also be known. w_i could be derived, but it is a necessary field within the code - so take care. Note that these variables do not imply the values of other parameters necessary for the calculation of loss; however, they may provide bounds. So for example inside assets + outside assets = total assets.

The variables that explicitly may be left as "NaN" are c_i outside assets, b_i outside liabilities, β_i (financial connectivity), d inside assets, and f inside liabilities.

The core item we are trying to estimate by optimizing is $\overline{P}$, the liabilities matrix since this represents the unknown network topology. An equivalent optimization is A, the relative liabilities matrix. These are equivalent since $\overline{p}_i$ is a forced input. Note that c_i and b_i provide information about the network but not the firm to firm level connections.

1.2.2 knitro_helper.m

This function manages the actual optimization. It includes a catch that causes the function to loop if there is an error. If the tester counter grows to large (which you will see in the command window). It takes in the cleaned inputs and has some unclear outputs.

The natural outputs are: [xstar,fval,exitflag,output,lambda,grad,t1]

- xstar contains all of the optimized values, the first N^2 are the elements of the optimized A matrix, (values go Astar - b (outside liabilities) - c (outside assets))
- fval is the "optimal" value of the parameter being optimized over (so either L or L/L^0)
- t1 is ?
- the other outputs are knitro outputs normal knitro outputs that are relavent to the optimization but not necessarily us.

These outputs are renamed - making it a bit confusing if you don't realize this and are trying to interpret results.

- Astar is the optimized A matrix formatted from xstar.
- output is the full xstar matrix (values go Astar - b (outside liabilities) - c (outside assets))
- obj_value is fval, the "optimal" value of the parameter being optimized over (so either $\overline{L}$ or $\overline{L}/\overline{L^0}$)

1.2.3 simulation_ratio.m

This file does the bulk of the work underlying the optimization. It takes our inputs, calculates the Loss and the derivatives/gradients necessary for the nitro optimizer to be able to pick the next attempt for optimization calculation. It also conducts the actual simulation testing. So using the inputted values it runs simulations of different shocks based on a specified distribution, which allows for a proper estimation of $\overline{L}$ and $\overline{L^0}$

2 Detailed Description of run_simulations.m

2.1 Constraints

The way knitro interprets and we construct constraints are not immediately intuitive. There are 3 types of constraints. Equality constraints, inequality constraints, and non-linear constraints.

2.1.1 Optimizable Parameters

The vector of optimizable parameters, x, is a column vector constructed as follows

$$x = \begin{bmatrix} A \\ - \\ b \\ - \\ c \end{bmatrix}$$

Since x must be a column vector, A is reconstructed as the "stacking" of its own columns. b is a column vector of missing 'b's. and c is a column vector of missing 'c's.

2.2 Constraints with Known Inside Assets and Liabilities

Knowing inside assets and liabilities greatly simplifies the optimization problem and makes it easier to understand how the constraints are modified when inside assets or liabilities are unknown.

First note that our optimizable parameters solely consist of the elements of A:

$$x = \begin{bmatrix} a_{1,1} \\ a_{2,1} \\ \vdots \\ a_{n,1} \\ a_{2,1} \\ \vdots \\ a_{n,n} \end{bmatrix}$$

Our first set of constraints makes sure that payments from other nodes adds up to inside assets:

$$d_i = \sum_j p_{j,i} = \sum_j \overline{p}_i \cdot \frac{p_{j,i}}{\overline{p}_i}$$

$$d_i = \sum_j \overline{p}_i a_{j,i}$$

Our second set of constraints makes sure that payments to other nodes adds up to inside liabilities:

$$f_i = \sum_j \overline{p}_i a_{i,j}$$

Our third constraint makes sure that each node does not pay itself:

$$a_{ii} = 0$$

The inequality constraint makes sure that the sum of payments to other nodes is less than inside liabilities. Note that this is a looser version of the second constraint so it is unnecessary in this scenario.

$$f_i \geq \sum_j \overline{p}_i a_{i,j}$$

$$\frac{f_i}{\overline{p}_i} \geq \sum_j a_{i,j}$$

Complementary constraint to guarantee we are only using net liabilities:

$$a_{i,j} \cdot a_{j,i} = 0$$

Given the existence of this constraint it is unclear to me why we need the 3rd equality constraint in the code.

2.3 Constraints with Unknown Inside Assets and Liabilities

Saying we don't know inside assets and liabilities is also equivalent to saying we don't know outside assets and liabilities as well. For sake of simplicity of notation assume we are missing all inside asset and liability information. This means our optimization matrix will look like:

$$x = \begin{bmatrix} a_{1,1} \\ a_{2,1} \\ \vdots \\ a_{n,1} \\ a_{2,1} \\ \vdots \\ a_{n,n} \\ b_1 \\ \vdots \\ b_n \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

For sake of clarity let α_i be the assets of firm i. For all constraints we start with Our first set of constraints makes sure that payments from other nodes adds up to inside assets:

$$\sum_j \overline{p}_i a_{j,i} = d_i = \alpha_i - c_i$$

$$\alpha_i = \sum_j \overline{p}_i a_{j,i} + c_i$$

Our second set of constraints makes sure that payments to other nodes adds up to inside liabilities:

$$\sum_j \overline{p}_i a_{i,j} = f_i = \overline{p}_i - b_i$$

$$\overline{p}_i = \sum_j \overline{p}_i a_{i,j} + b_i$$

Our third constraint makes sure that each node does not pay itself:

$$a_{ii} = 0$$

The inequality constraint makes sure that the sum of payments to other nodes is less than inside liabilities. This is here to help the optimizer find solutions faster.

$$f_i \geq \sum_j \overline{p}_i a_{i,j}$$

$$\frac{f_i}{\overline{p}_i} \geq \sum_j a_{i,j}$$

Complementary constraint to guarantee we are only using net liabilities:

$$a_{i,j} \cdot a_{j,i} = 0$$

2.4 Lower and upper bounds

Similar to x, the vectors of upper and lower bounds are split into 3 sections:

1. Elements of A are bounded as follows:
$$0 \leq a_{ij} \leq \frac{f_i}{\overline{p}_i}$$
If f is missing, we bound by 1.

2. Our missing b are bounded as follows:
$$0 < b_i < \overline{p}_i$$
Since matlab wants non-strict inequalities we use
$$10^{-5} \leq b_i \leq \overline{p}_i - 10^{-5}$$

3. Our missing c are bounded as follows:
$$0 < c_i < \alpha_i$$
Where α_i is firm i's assets. Since matlab wants non-strict inequalities we use
$$10^{-5} \leq c_i \leq \alpha_i - 10^{-5}$$

3 Disaggregating Nodes

As one would expect, aggregated nodes (Insurance Companies, Other, REITs) consistently rank high in asset sorting of firms. It is possible that this aggregation is leading to infeasibility issues in the sample. To investigate this issue, we are adding an option to desegregate nodes to the RAN code, but the process isn't 100% straight forward.

3.1 Constraints

A big challenge of the disaggregation is that we have little information on the new disaggregated nodes. For example we don't know $\overline{p}_i$ for them which means if we wanted to follow the path of using the relative liabilities matrix we would be stuck with many non-linear constraints which is suboptimal. To resolve this we change the construction of the constraints fairly significantly.

We do follow the general format as above but rather than transforming a_{ij} to $\overline{p}_{ij}$, using $\overline{p}_{i,j}$ we directly solve for the liabilities matrix.

We change the matrix of optimizable parameters to the following. We say the first $n_0 - 1$ nodes are not aggregated, and fall in set N_0 and the last $n - n_0$ nodes are the disaggregated nodes and fall in set D. We know certain variables for the aggregate nodes: $a_{AGG}, \overline{p}_{AGG}, c_{AGG}$, and d_{AGG} are known. We do not know b_{AGG} or f_{AGG}

$$x = \begin{bmatrix} \overline{p}_{1,1} \\ \overline{p}_{2,1} \\ \vdots \\ \overline{p}_{n,1} \\ \overline{p}_{2,1} \\ \vdots \\ \overline{p}_{n,n} \\ b_1 \\ \vdots \\ b_n \\ c_1 \\ \vdots \\ c_n \end{bmatrix}$$

1. Our first set of constraints makes sure that payments from other nodes adds up to inside assets:

$$d_{i \in N_0} = \sum_j \overline{p}_{j,i}$$

$$d_{AGG} = \sum_j \sum_{i \in D} \overline{p}_{j,i}$$

   In this case our first $n_0 - 1$ rows of the constraints matrix are constructed rather analogously to the previously shown method. The final row ensures that the inside assets of the new nodes add up to the inside assets of the aggregate node we are trying to disaggregate.

2. Our second set of constraints makes sure that payments to other nodes adds up to inside liabilities:

$$f_{i \in N_0} = \sum_j \overline{p}_{i,j}$$

$$\overline{p}_{AGG} = \sum_{i \in D} \left[b_i + \sum_j \overline{p}_{i,j} \right]$$

3. Our third constraint makes sure that each node does not pay itself:

$$a_{ii} = 0$$

4. We add a fourth constraint so that we have values for outside assets for our new nodes. Simply:

$$\sum_{i \in D} c_i = c_{AGG}$$

There are two inequality constraints driven by this set of inequalities:

$$0 < w < c$$

While these rules apply to all nodes, we only need to set them for the disaggregated nodes (since we don't know w)

1. We start with $w < c$:

$$\begin{aligned} w &= a - \overline{p} \\ w &= (c + d) - (b + d) \\ w &= c + d - b - f \\ w &< c \\ c + d - b - f &< c \\ d - b - f &< 0 \end{aligned}$$

2. The second set of constraints is based on $0 < w$:

$$\begin{aligned} w &= c + d - b - f \\ 0 &< c + d - b - f \\ b + f - c - d &< 0 \end{aligned}$$

Complementary constraint to guarantee we are only using net liabilities:

$$a_{i,j} \cdot a_{j,i} = 0$$

Note that with these parameters we can immediate derive A. How?

$$\overline{p}_i = b_i + \sum_{j \neq i} \overline{p}_{ij}$$

$$a_{ij} = \begin{cases} \overline{p}_{ij} / \overline{p}_i, & \text{if } \overline{p}_i \\ 0, & \text{if } \overline{p}_i = 0 \end{cases}$$

3.2 Lower and upper bounds

Similar to x, the vectors of upper and lower bounds are split into 3 sections:

1. Elements of $\overline{P}$ are bounded as follows:
$$0 \leq \overline{p}_{ij} \leq \overline{p}_{AGG}$$
2. Our missing b are bounded as follows:
$$0 < b_i < \overline{p}_i$$
Since matlab wants non-strict inequalities we use
$$10^{-5} \leq b_i \leq \overline{p}_{AGG} - 10^{-5}$$
3. Our missing c are bounded as follows:
$$0 < c_i < \alpha_i$$
Where α_i is firm i's assets. Since matlab wants non-strict inequalities we use
$$10^{-5} \leq c_i \leq c_{AGG} - 10^{-5}$$

3.3 Transforming the gradient

Since the optimizable parameters for the disaggregated node case is

$$x = \begin{bmatrix} P \\ - \\ b \\ - \\ c \end{bmatrix}$$

the gradient vector used for the standard case must be transformed. Fortunately we need only transform $\frac{dL}{dA}$ and not the entire matrix.

To get from $\frac{dL}{dA}$ to $\frac{dL}{d\overline{P}}$, I calculate the $\frac{da_{i,j}}{d\overline{p}_{i,j}}$:

$$\begin{aligned} a_{i,j} = \frac{\overline{p}_{i,j}}{\overline{p}_i} \Rightarrow \overline{p}_{i,j} &= a_{i,j}\overline{p}_i = a_{i,j}(b_i + \sum \overline{p}_{i,j}) \\ &= a_{i,j}b_i + a_{i,j}\sum \overline{p}_{i,j} \end{aligned}$$

fix i,j of interest as x,y

$$\begin{aligned} \Rightarrow \overline{p}_{x,y} - a_{x,y}\overline{p}_{x,y} &= a_{x,y}b_x + a_{x,y}\sum_{j \neq x,y} \overline{p}_{x,j} \\ \Rightarrow \overline{p}_{x,y} &= (\frac{a_{x,y}}{1 - a_{x,y}})(b_x + \sum_{j \neq x,y} \overline{p}_{x,j}) \end{aligned}$$

now take derivative w.r.t. $a_{x,y}$

$$\Rightarrow \frac{da_{x,y}}{d\overline{p}_{x,y}} = \frac{b_x + \sum_{j \neq x,y} \overline{p}_{x,j}}{(a_{x,y} - 1)^2}$$