Cauê Mello da Silva

Framework colaborativo de simulação computacional para o estudo de regulação bancária

Brasília

2018

Cauê Mello da Silva

Framework colaborativo de simulação computacional para o estudo de regulação bancária

Dissertação apresentada ao Programa de Mestrado em Economia da Universidade de Brasília como requisito à obtenção do título de Mestre em Ciências Econômicas.

Universidade de Brasília

Faculdade de Economia, Administração e Contabilidade

Departamento de Economia

Orientador: Prof. Daniel Oliveira Cajueiro, PhD

Brasília

2018

Cauê Mello da Silva

Framework colaborativo de simulação computacional para o estudo de regulação bancária

Dissertação apresentada ao Programa de Mestrado em Economia da Universidade de Brasília como requisito à obtenção do título de Mestre em Ciências Econômicas.

Trabalho aprovado. Brasília, 27 de fevereiro de 2018:

Prof. Daniel Oliveira Cajueiro, PhD
Orientador

Thiago Christiano Silva, PhD
Banco Central do Brasil

Prof. Bernardo Alves Furtado, PhD
Instituto de Pesquisa Econômica Aplicada

Brasília
2018

Resumo

Este trabalho apresenta um *framework* computacional em linguagem Python, desenvolvido para estudar os efeitos de diferentes políticas regulatórias no mercado bancário através de simulações baseadas em agentes. Tem por objetivo principal disponibilizar um *software* de desenvolvimento colaborativo com boa manutenibilidade[1], que possa ser facilmente alterado para comportar novos modelos econômicos. O modelo inicial baseia-se em uma versão iterada do modelo Diamond-Dybvig, com diferentes tipos de agentes, tais como bancos, firmas, depositantes e o Banco Central, capazes de fazer escolhas e interagirem entre si. O modelo inicial também utiliza a estrutura de aprendizado EWA, permitindo que os agentes definam suas estratégias de acordo com experiências anteriores. Ao final, disponibiliza-se exemplos de utilização do *framework* através de simulações, permitindo a análise dos dados e o estudo das políticas regulatórias simuladas e das estruturas das redes formadas pelos bancos no mercado interbancário.

Palavras-chave: Modelo baseado em agentes. Políticas regulatórias. Redes bancárias. Python.

[1] Manutenibilidade é a capacidade do produto de *software* de ser modificado. As modificações podem incluir correções, melhorias ou adaptações do *software* devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais, conforme ABNT (2003).

Abstract

This work presents a computational framework in Python language, developed to study the effects of different regulatory policies in the banking market through agent-based simulations. Its main goal is to provide a collaborative software with good maintainability that can be easily modified to include new economic models. The initial model is based on an iterated version of Diamond-Dybvig model, with different agent types, such as banks, firms, depositors and the Central Bank, capable of making choices and interacting with each other. The initial model also uses the EWA learning structure, allowing agents to define their own strategies according to previous experiences. At the end, there are examples of using the framework through simulations, allowing the analysis of the data and the study of simulated regulatory policies and network structures formed by banks in the interbank market.

Keywords: Agent-based model. Regulatory policies. Banking networks. Python.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
ABM	Agent-based model - Modelo baseado em agentes
CAR	Capital Adequacy Ratio - Taxa de Adequação de Capital
EWA	Experience-Weighted Attraction
LGD	Loss Given Default - Perda dada a Inadimplência
ROE	Return on Equity - Retorno sobre o Patrimônio
UML	Unified Modeling Language - Linguagem de Modelagem Unificada

Sumário

Introdução

Nas últimas décadas, crises financeiras ocorreram ao redor do mundo e, apesar de diversos fatores serem colocados como causa, acredita-se que regulação adequada poderia ter sido usada para evitá-las (LEVINE, 2012).

Antes da crise de 2008, o foco da supervisão bancária era microprudencial, concentrada excessivamente sobre a regulação das instituições de maneira individual e omitindo a intermediação financeira como característica macroeconômica essencial. A regulação financeira, portanto, focou na solidez das instituições individuais e buscou corrigir as falhas de mercado derivadas da assimetria de informação, da responsabilidade limitada e de outras imperfeições, tais como as garantias implícitas ou explícitas do governo (BLANCHARD; DELLARICCIA; MAURO, 2010).

Pouca importância foi dada ao uso de índices regulatórios, tais como taxa de capital. Pelo contrário, dado o entusiasmo pela desregulação financeira, o uso de regulação prudencial com fins cíclicos foi considerado inadequado com respeito ao funcionamento dos mercados de crédito (BLANCHARD; DELLARICCIA; MAURO, 2010).

A crise, então, destacou a necessidade de ir além da abordagem microprudencial e gerou inúmeros discursos e trabalhos centrados no uso, na implementação e na eficácia de ferramentas macroprudenciais, bem como o impacto dessas ferramentas nos resultados macroeconômicos e na relação com a política monetária (GALATI; MOESSNER, 2013).

O estudo da interconectividade das instituições financeiras passou a ganhar importância cada vez maior, tratando o sistema financeiro como um sistema complexo, com modelos baseados em agentes heterogêneos de racionalidade limitada, cujo processo de aprendizagem influencia a dinâmica agregada do sistema. Uma linha de pesquisa relacionada analisa o sistema financeiro como uma rede dinâmica de agentes, conectados diretamente através de exposições mútuas no mercado interbancário e indiretamente através de carteiras similares ou compartilhando o mesmo grupo de depositantes (GALATI; MOESSNER, 2013).

Em Goodhart (2008), são enumerados sete itens regulatórios que devem ser considerados: garantia dos depósitos, regimes de insolvência bancária, operações no mercado monetário pelos bancos centrais, limites de regulação e risco reputacional, gerenciamento de risco de liquidez, efeitos pró-cíclicos nas taxas de adequação de capital (Basileia II e a falta de instrumentos anticíclicos) e gerenciamento de crises, interno e entre países.

Para avaliar corretamente esses itens e determinar ações adequadas, é importante que os tomadores de decisão entendam os arranjos das redes financeiras, que têm se tornado

cada vez mais complexas[2] (HALDANE; MAY, 2011). O trabalho de Georg (2013) apresenta um estudo no qual compara o impacto de diferentes estruturas de redes interbancárias na estabilidade financeira, a partir do efeito de contágio e choques comuns, utilizando um modelo dinâmico de multiagentes. Em Cajueiro e Tabak (2008), Silva et al. (2016) e Silva, Silva e Tabak (2017), são estudadas as características das redes interbancárias aplicadas ao caso brasileiro.

Um dos grandes problemas observados em ambientes com assimetria de informação é o da existência de corridas bancárias, que podem ter seus efeitos amplificados por conta da interconectividade entre agentes econômicos. Como apresentado em Diamond e Dybvig (1983), durante essa corrida, os depositantes impacientes se apressam a sacar seus depósitos com o receio de que seu banco se torne ilíquido e não possa honrar seu passivo de curto prazo. Como consequência dos saques repentinos de diversos clientes impacientes, o banco, mesmo que esteja saudável financeiramente, pode precisar liquidar seus ativos, causando grandes prejuízos. Dependendo da intensidade da crise, diversos bancos quebrarão com os saques realizados, agravando a situação e propagando o problema até mesmo a outros países (CETORELLI; GOLDBERG, 2011).

Outro problema que deve ser considerado no estudo das redes bancárias é a falência do banco por ausência de casamento entre os passivos de curto prazo (depósitos) e os ativos de longo prazo (empréstimos para o setor real). A quebra de um banco pode levar à quebra de outros por contágio direto, através do mercado interbancário, ou por contágio indireto, quando as carteiras de diferentes bancos possuem exposição correlacionada à mesma fonte de risco (NIER et al., 2007; SILVA; SILVA; TABAK, 2017).

De acordo com Markowitz (1952), existe uma carteira (portfólio ótimo) que maximiza o retorno esperado para um determinado nível de risco. Assim, bancos com a mesma aversão a risco tendem a investir em carteiras otimizadas muito parecidas, comprados em ativos similares. No caso de esses ativos perderem valor por qualquer motivo, o sistema torna-se exposto à falência de vários bancos simultaneamente (BEALE et al., 2011).

Para analisar o impacto da regulação nas redes financeiras, alguns modelos baseados em agentes vêm sendo desenvolvidos. Chakrabarti (2000) apresenta um modelo baseado em agentes do mercado de câmbio, no qual as partes aprendem de maneira bayesiana a partir das ordens anteriores dos outros participantes. Utilizando simulação, o autor obteve valores em conformidade com o observado empiricamente.

O trabalho de Montagna e Kok (2016) desenvolve um modelo baseado em agentes para explorar redes interbancárias com múltiplas camadas, utilizando uma amostra com os principais bancos da União Europeia. O trabalho conclui que os efeitos de contágio

[2] Redes complexas são redes normalmente utilizadas para modelar sistemas com topologias não triviais, sendo compostas por uma grande quantidade de vértices. Surgem como representação unificada de sistemas complexos em vários ramos científicos (SILVA; ZHAO, 2016).

quando os choques são propagados ao longo das diferentes camadas da rede podem ser substancialmente maiores do que a soma dos efeitos resultantes de contágio quando cada camada é considerada individualmente.

Por fim, Lucchetti (2016) e Barroso et al. (2016) fazem uma ampla análise de modelos baseado nos trabalhos de Barroso (2014) e Lima (2014), que avaliam os resultados de políticas de regulação, considerando o comportamento estratégico dos agentes frente às mudanças nas restrições e no ambiente econômico.

O trabalho de Lima (2014) apresenta um arcabouço computacional baseado em agentes para estudar o setor bancário. Apesar de o arcabouço ser utilizado como base dos trabalhos de Barroso (2014), Lucchetti (2016) e Barroso et al. (2016), ele requer uma modificação significativa do código para cada alteração e evolução do modelo. Assim, cada versão se torna incompatível com as demais, pois não há uma diretriz de versionamento e desenvolvimento colaborativo, restringindo o arcabouço a um pequeno grupo de acadêmicos.

Este trabalho propõe, portanto, o desenvolvimento de um *framework*, respeitando os princípios de engenharia de software e as características de qualidade descritas na ABNT (2003), em destaque à usabilidade e à manutenibilidade do código. Assim, o produto do trabalho possibilitará que pessoas de outras universidades, instituições financeiras ou bancos centrais possam avaliar, criticar e agregar novas funcionalidades ao modelo ou ao comportamento dos agentes.

O *framework* foi desenvolvido em Python e publicado como código aberto no GitHub[3]. Assim, os próximos pesquisadores que trabalharem com modelos de simulação de redes bancárias baseados em agentes poderão contribuir de uma forma mais fácil, partindo do modelo mais atual ou de outras versões desenvolvidas. Pesquisadores de instituições estrangeiras também poderão contribuir ao projeto, visto que ficará disponível em repositório público, com exemplos de utilização.

A arquitetura computacional utilizada na modelagem é baseada no paradigma de orientação a objetos. Nesse paradigma, os agentes e outros elementos são representados por classes, que contêm dados no formato de atributos (variáveis) e comportamentos no formato de métodos (procedimentos). Cada agente individualmente é um objeto diferente de sua classe, conhecido em engenharia de *software* pelo nome de instância da classe (SOMMERVILLE, 2004).

[3] <https://github.com/> - Plataforma de hospedagem de código-fonte com controle de versão usando o Git.

1 Revisão de Literatura

Ao pensar em um problema de economia, supõe-se inicialmente que os agentes são racionais. Assim, espera-se que eles assumam comportamentos bem definidos, facilitando a análise do problema estudado. Porém, quando os critérios de escolha dos agentes deixam de ser racionais, ou ainda, dependem do comportamento de outros agentes, fica inviável encontrar uma solução fechada através de modelos formados apenas por equações matemáticas.

1.1 Modelagem Baseada em Agentes

A modelagem nas ciências sociais, assim como na Física, geralmente utiliza especificação e análise de sistemas parametrizados e equações diferenciais. No entanto, é extremamente difícil capturar aspectos físicos, institucionais e comportamentais dos sistemas sociais com fidelidade empírica e ainda ser possível encontrar solução analítica. As entidades nos sistemas sociais não são infinitesimalmente pequenas, ou em quantidades infinitas, e suas identidades ou comportamentos são geralmente distintos uns dos outros. As simplificações comuns, como os comportamentos homogêneos assumidos ou a existência de agentes representativos únicos, são, portanto, problemáticas (BORRILL; TESFATSION, 2011).

Segundo Gaffeo et al. (2008), sempre que os agentes são heterogêneos em relação a suas estratégias e conjuntos de informações, a adesão total à modelagem do comportamento estratégico resulta em problemas computacionalmente complexos, ou seja, problemas cujo tempo de solução aumenta exponencialmente em relação ao tamanho do problema.

Um modelo baseado em agentes é formado por diversos agentes individualizados, implementados como objetos de uma ou mais classes de *software*, e modelados através de estados e regras de comportamento. A execução do modelo consiste, então, em criar a população de agentes, deixá-los interagirem entre si de acordo com as regras de comportamento estabelecidas, e por fim monitorar a execução. Assim, a própria execução do modelo é suficiente para fornecer a solução do problema original (AXTELL, 2000).

Como vantagens dessa abordagem, pode-se modelar os agentes de forma mais flexível, possivelmente racionais[1], com características e comportamentos diferentes dos demais (heterogeneidade), a depender das funções objetivo, e com autonomia, de forma que não haja nenhum controlador central definindo as regras de comportamento (EPSTEIN,

[1] Ou limitadamente racionais (*bounded rationality*), quando os agentes buscam uma solução satisfatória em vez da solução ótima (RUBINSTEIN, 1998).

1999).

Adicionalmente, à medida que o modelo vai sendo executado, é possível o acesso a todo o histórico da simulação, além da solução final, permitindo a análise posterior do estado transiente, e não apenas do estado estacionário, visto, inclusive, que um equilíbrio pode nem mesmo existir.

No estudo das ciências sociais, é muitas vezes imprescindível posicionar os agentes em espaços físicos ou estruturas de rede, restringindo a interação entre eles (*e.g.*, quando agentes só podem trocar informações se estiverem em nós conectados na rede). Em uma modelagem por equações matemáticas, é extremamente complexo colocar restrições aos estados dos agentes no modelo, porém, na baseada em agentes, é apenas mais uma regra a ser definida no conjunto total de comportamento de cada agente (AXTELL, 2000).

Apesar das vantagens destacadas, um modelo baseado em agentes deve ser simulado um número suficiente de vezes, com pequenas variações nas condições iniciais ou nos parâmetros, para confirmar o resultado e verificar o impacto dessas mudanças. Por outro lado, em modelos puramente matemáticos, é possível verificar com facilidade o impacto das alterações através de ferramentas como diferenciação de funções ou teorema da função implícita (AXTELL, 2000).

Essas questões são muito importantes para a análise econômica, muitas vezes até mais importante que a solução em si. Entretanto, como as ferramentas de computação (*hardware* e *software*) tornaram-se mais poderosas e fáceis de usar, é cada vez maior o número de pesquisas utilizando técnicas de simulação (MARKS, 2012).

1.2 Aprendizagem

Dotados de interesses, habilidades e informações que interagem em tempo real, os agentes se comportam em resposta às suas estruturas de incentivo, sujeitas a certas restrições. A importância desses incentivos e a capacidade de adaptar o comportamento, mudando decisões ou regras de tomada de decisão, distingue os sistemas sociais dos sistemas físicos (PAGE, 1997).

Assim, uma forma de modelagem é representar os agentes como organismos adaptativos, que aprendem e fazem escolhas de acordo com suas experiências passadas. Camerer (2011) apresenta evidências experimentais sugerindo que os modelos de aprendizagem geram previsões mais precisas do que os modelos baseados na racionalidade perfeita.

O modelo de aprendizado descrito por Camerer e Ho (1999), *Experience-Weighted Attraction* (EWA), captura a dupla natureza da aprendizagem adaptativa: a lei do efeito real e a lei do efeito simulado.

A lei do efeito real afirma que as ações escolhidas que são bem sucedidas serão

escolhidas mais frequentemente do que aquelas que não são bem sucedidas (aprendizado por reforço - *reinforcement learning*). A lei do efeito simulado, por sua vez, afirma que ações que não foram escolhidas, mas que teriam sido bem sucedidas, serão escolhidas no futuro com mais frequência (aprendizado baseado em crença - *belief-based learning*).

Na estrutura de aprendizado EWA, os agentes fazem suas escolhas de acordo com as funções de atração de cada ação, representando a propensão de escolhê-las entre as disponíveis, e depende da experiência desses agentes nas iterações anteriores. Os agentes participam, portanto, de um jogo iterativo. Usar o modelo EWA é útil para entender como a natureza do aprendizado afeta a convergência para o equilíbrio (POUGET, 2007).

2 Modelo

O modelo escolhido como base inicial para o *framework* desenvolvido foi concebido inicialmente por Barroso (2011), melhorado por Barroso (2014) e Lima (2014) e sumarizado por Barroso et al. (2016). Esse modelo consiste em iterações de ciclos de simulação discreta no tempo, baseado nos trabalhos de Diamond e Dybvig (1983), Allen e Gale (1998) e Allen e Gale (2000). São usados agentes com capacidade de aprender e adaptar suas estratégias de acordo com a estrutura de aprendizado proposta por Camerer e Ho (1999).

Os trabalhos de Lucchetti (2016) e Barroso et al. (2016) apresentam em detalhes o modelo utilizado, que será descrito a seguir neste capítulo. As características computacionais de implementação serão apresentadas a partir do Capítulo 3.

2.1 Organização Geral do Modelo

Os bancos formam uma rede endógena, conectando seus balaços patrimoniais por meio de empréstimos interbancários. As redes são formadas em decorrência das estratégias adaptativas dos bancos e da presença de choques estocásticos de liquidez na economia, de acordo com Camerer e Ho (1999) e Diamond e Dybvig (1983).

O objetivo primário do modelo é permitir a análise dos impactos de diversas políticas econômicas, representadas de forma exógena pelos parâmetros e configurações da simulação, possibilitando estudar inclusive o papel do Banco Central quanto ao seu objetivo de manter a estabilidade financeira.

Por ser um modelo estocástico, a simulação será executada um número significativo de vezes, na qual cada iteração é chamada ciclo. Cada ciclo, por sua vez, é dividido em três horizontes distintos de tempo, $t = 0, 1, 2$, de forma a facilitar a organização do modelo.

O primeiro período do ciclo, $t = 0$, é interpretado como o dia corrente, no qual os bancos determinam as estratégias que serão utilizadas no ciclo atual. As estratégias incluem, por exemplo, a escolha das quantias iniciais de empréstimos ao setor real, ativos líquidos, depósitos e capital.

O período seguinte, $t = 1$, representa o horizonte de curto prazo, no qual os bancos sofrem o choque de liquidez devido a depositantes impacientes, que sacam seus recursos de maneira prematura, visando antecipar o consumo. As instituições financeiras que ficarem, ao fim do choque, com problemas de liquidez tomarão empréstimos de outros bancos, utilizando o mercado interbancário, ou do Banco Central, no papel de emprestador de última instância, submetendo-se a uma taxa punitiva.

O último período do ciclo, $t = 2$, corresponde ao horizonte de longo prazo, no qual os empréstimos ao setor real da economia atingem a maturidade e os depositantes pacientes consomem seus depósitos. Os bancos, em seguida, calculam o retorno sobre o patrimônio (*Return on Equity* - ROE) para atualizar as estratégias utilizadas pela estrutura de aprendizado. Ao final desse período, pode haver banco insolvente em decorrência de perdas nos empréstimos ao setor real ou de contaminação financeira no mercado interbancário.

2.2 Aprendizado

O modelo adota a estrutura de aprendizado proposta por Camerer e Ho (1999), *Experience-Weighted Attraction* (EWA), para descrever as estratégias de escolha dos agentes. A EWA considera duas distintas abordagens de modelagem do comportamento dos agentes, o aprendizado por reforço (*Reinforcement Learning*) e o aprendizado baseado em crença (*Belief-based Learning*).

O aprendizado por reforço considera que as estratégias que tiveram melhores resultados no passado devem ser adotadas com maior probabilidade (ROTH; EREV, 1995). O aprendizado baseado em crença, por sua vez, examina, dentre as estratégias que não foram escolhidas, as que teriam dado resultado satisfatório, e as coloca como prováveis de serem escolhidas nos próximos ciclos (FUDENBERG; LEVINE, 1998).

2.2.1 Notação Utilizada

Neste trabalho, será utilizada a mesma notação definida em Camerer e Ho (1999). Os agentes são indexados por $i (i = 1, ..., n)$ e o espaço de estratégias do agente i, $S_i = \{s_i^1, s_i^2, ..., s_i^{m_i-1}, s_i^{m_i}\}$, consiste de m_i escolhas discretas. $S = S_1 \times ... \times S_n$ é o produto cartesiano dos espaços de estratégias individuais dos agentes, ou seja, é o espaço de estratégias do jogo.

A estratégia do jogador i é denotada por $s_i \in S_i$, $s = (s_1, ..., s_n) \in S$ é a combinação de estratégias de todos os agentes, e $s_{-i} = (s_1, ..., s_{i-1}, s_{i+1}, ..., s_n)$ é a combinação de estratégias de todos os agentes, exceto o i.

A função de ganho (*payoff*) do jogador i é representada por $\pi_i(s_i, s_{-i})$. Como o jogo evolui no tempo, utiliza-se a notação $s_i(t)$ para denotar a estratégia do jogador i no período t. De uma forma mais completa, $s_i^j \in S_i$ é a j-ésima estratégia do jogador i, de seu espaço de estratégias, tal que $j \in (1, ..., m_i)$.

A estrutura de aprendizado EWA assume que cada estratégia tem um valor numérico de atração, determinando a probabilidade da estratégia ser escolhida. Assim, cada modelo deve especificar os valores iniciais de atração, a forma de atualização da atração de acordo com a experiência, e como as probabilidades de escolha dependem do valor da atração.

2.2.2 Regras de Atualização

Há duas variáveis no modelo EWA que são atualizadas após cada período: $N(t)$, interpretada como o número de equivalências observadas de experiência passada, e $A_i^j(t)$, atração da estratégia s_i^j do agente i, após o período t.

As variáveis $N(t)$ e $A_i^j(t)$ começam com valores iniciais, que podem ser interpretados como experiência prévia. As equações a seguir apresentam as regras de atualização do EWA.

$$N(t) = \rho \cdot N(t-1) + 1, \quad t \geq 1 \tag{2.1}$$

O parâmetro ρ, na Equação 2.1, é a taxa de depreciação ou fator de desconto retrospectivo, que mede o impacto de experiências passadas no período atual.

A próxima regra descreve a atualização do nível de atração. Um componente chave da atualização é o ganho que cada estratégia forneceu, ou poderia ter fornecido, em um período anterior. O modelo pondera os ganhos hipotéticos de estratégias não escolhidas pelo parâmetro δ, e os ganhos de fato recebidos pela estratégia escolhida, $s_i(t)$, por $1 - \delta$.

A Equação 2.2 descreve a regra de atualização dos conjuntos de atração $A_i^j(t)$, que é a soma da atração anterior $A_i^j(t-1)$ depreciada (ponderada pela experiência anterior) com o ganho ponderado do período atual, normalizado pela experiência acumulada $N(t)$. O fator ϕ é a taxa de decaimento utilizada para depreciar atrações anteriores.

$$A_i^j(t) = \frac{\phi \cdot N(t-1) \cdot A_i^j(t-1) + [\delta + (1-\delta) \cdot I(s_i^j, s_i(t))] \cdot \pi(s_i^j, s_{-i}(t))}{N(t)} \tag{2.2}$$

A Equação 2.3 define a função indicador $I(x, y)$, utilizada como recurso matemático na ponderação do ganho das estratégias.

$$I(x,y) = \begin{cases} 1, & \text{se } x = y \\ 0, & \text{se } x \neq y \end{cases} \tag{2.3}$$

2.2.3 Probabilidades de Escolha

As atrações determinam as probabilidades de escolha de cada estratégia. $P_i^j(t)$ é a probabilidade do agente i escolher a estratégia s_i^j no período t. $P_i^j(t)$ deve ser monotonicamente crescente em $A_i^j(t)$ e decrescente em $A_i^k(t)$, com $k \neq j$.

Para o cálculo da probabilidade, utiliza-se a função exponencial (*logit*), de acordo com a Equação 2.4, por ser mais utilizada em estudos de escolhas com risco e incerteza,

conforme descrito em Camerer e Ho (1999).

$$P_i^j(t+1) = \frac{e^{\lambda . A_i^j(t)}}{\sum_{k=1}^{m_i} e^{\lambda . A_i^k(t)}} \tag{2.4}$$

O parâmetro λ mede a sensibilidade dos jogadores às atrações. Nessa função de probabilidade, o expoente no numerador é o efeito ponderado da atração da estratégia s_i^j, em relação a probabilidade de escolher essa estratégia.

Os valores dos parâmetros utilizados pelos agentes no modelo serão discutidos na realização dos experimentos.

2.3 Bancos

Os bancos, agentes principais do modelo, atuam como intermediários financeiros, levando os recursos dos depositantes às firmas. Também se conectam a outros bancos, no mercado interbancário, e ao Banco Central, quando necessário, dando origem à rede bancária. Os bancos são representados no modelo por seus balanços patrimoniais.

Os balanços patrimoniais são modelados para permitir aos bancos o controle entre o dinheiro depositado e os empréstimos de longo prazo, deixando que eles decidam entre assumir mais riscos através de mais empréstimos, obtendo mais lucro, ou manter a reserva mais alta, evitando um futuro problema de liquidez, que pode levar à sua liquidação.

No passivo do balanço patrimonial estão o capital, atualizado durante o ciclo e usado para o cálculo do ROE, os depósitos, que são os passivos líquidos, e os empréstimos, tomados no mercado interbancário ou do Banco Central, em caso de falta da liquidez exigida. Um resumo é apresentado na Tabela 1.

Tabela 1 – Passivo dos Bancos

Símbolo	Passivo	Maturidade	Custo
K_b	Capital	-	-
D_b	Depósitos	$t+2$	i_d
IL_b	Empréstimos Interbancários	$t+1$	i_i
C_b	Empréstimos do Banco Central	$t+1$	i_{cb}

Fonte: (LUCCHETTI, 2016)

No lado dos ativos, há os ativos líquidos, como dinheiro, que representam as reservas do banco, os empréstimos interbancários fornecidos e os empréstimos ao setor real da economia, que estão sujeitos a risco de crédito (*i.e.*, o banco possui um retorno esperado em relação aos empréstimos às firmas). Um resumo é apresentado na Tabela 2.

Tabela 2 – Ativo dos Bancos

Símbolo	Ativo	Maturidade	Retorno
L_b	Ativos Líquidos	t	-
IL_b	Empréstimos Interbancários	$t+1$	$r_i = i_i$
R_b	Empréstimos ao Setor Real	$t+2$	r_b

Fonte: (LUCCHETTI, 2016)

Para haver o incentivo de empréstimo ao setor real custeado pelos depósitos, deve existir uma diferença positiva entre as taxas, ou seja, $E(r_b) > i_d$ [1]. E para que os bancos não utilizem os recursos do Banco Central nos empréstimos, deve valer $E(r_b) < i_{cb}$, ou seja, i_{cb} é uma taxa punitiva. Por último, é necessário manter a condição $r_i = i_i > i_d$, pois o financiamento através dos depósitos é mais barato para os bancos.

Essas taxas são determinadas de maneira exógena, na configuração inicial do modelo, devendo satisfazer às regras definidas.

Durante a execução do modelo, os bancos participam de um jogo iterativo simultâneo, no qual cada um tenta maximizar o seu ROE. A estratégia do banco b, s_b^j, é representada pelo par (α^j, β^j), com α^j representando a razão entre o capital e o total de passivos, e β^j o percentual de ativos líquidos.

O tamanho do banco b é dado de forma exógena e representado por T_b, correspondendo à soma total de ativos (ou passivos). No início de cada ciclo, em $t = 0$, não há atividade interbancária, e o balanço do banco é determinado pela escolha da estratégia s_b^j e pelo tamanho do banco, T_b, de acordo com as equações a seguir:

1. Capital

$$K_b = \alpha^j \cdot T_b \tag{2.5}$$

2. Depósitos

$$D_b = (1 - \alpha^j) \cdot T_b \tag{2.6}$$

3. Ativos Líquidos

$$L_b = \beta^j \cdot T_b \tag{2.7}$$

[1] $E(r_b)$ representa o valor esperado da taxa a ser recebida pelos bancos nos empréstimos ao setor real da economia, considerando a probabilidade de *default* e a perda esperada.

4. Empréstimos às firmas

$$R_b = (1 - \beta^j) \cdot T_b \tag{2.8}$$

No final de cada ciclo, o banco calcula o lucro e o ROE através das equações 2.9 e 2.10.

$$\Pi_b(t) = K_b(t) - K_b(t-1) \tag{2.9}$$

$$ROE_b(t) = \frac{\Pi_b(t)}{K_b(t-1)} \tag{2.10}$$

Por último, por questões regulatórias, o banco também deve calcular o CAR, definido como a razão entre o capital e os ativos ajustados ao risco, conforme a Equação 2.11.

$$CAR_b = \frac{K_b}{IL_b + \sum_{f \in F_b} R_{b,f} \cdot w_f} \tag{2.11}$$

F_b é o conjunto de todas as firmas que possuem empréstimo no banco b. Assim, $R_{b,f}$ é o empréstimo do banco b à firma f, com nível de risco w_f, específico para cada firma (heterogeneidade entre as firmas).

Caso haja necessidade de adequação do CAR, o banco deve vender uma parte de sua carteira de empréstimo antes da maturação, com uma taxa de desconto δ_L para refletir a perda pela falta de liquidez.

2.4 Banco Central

O Banco Central atua no modelo como o agente definidor das principais regras que influenciam o comportamento dos bancos. Essas regras visam à manutenção da estabilidade do sistema financeiro e, para isso, o Banco Central pode agir como supervisor do sistema financeiro e como emprestador de última instância. As atribuições de política monetária não serão abordadas no modelo.

Quando os bancos sofrem problemas de liquidez, ou seja, não possuem dinheiro suficiente para suportar os saques dos depositantes, nem através da venda de seus ativos líquidos e nem de empréstimos no mercado interbancário, o Banco Central age como emprestador de última instância. Como a falta de liquidez gera problemas na rede bancária, por conta de possíveis contágios e contração do crédito, o Banco Central assume essa responsabilidade. Além disso, o choque pode atingir outros países, como abordado por Cetorelli e Goldberg (2011).

Para minimizar a necessidade de empréstimos de última instância, o Banco Central, atuando como supervisor, deve estabelecer regras a serem cumpridas pelos bancos. Neste modelo, o Banco Central determinará a mínima taxa de adequação de capital (*Capital Adequacy Ratio* - CAR) a ser seguida pelos bancos para continuar em operação. Se o CAR de um banco for menor que o CAR mínimo determinado, o Banco Central pode forçar essa adequação ou liquidar o banco no caso da impossibilidade de cumprir o requisito.

2.5 Depositantes

Depositantes possuem o comportamento descrito no modelo de Diamond e Dybvig (1983), no qual eles podem ser: pacientes, esperando o momento $t = 2$ para a retirada dos depósitos, recebendo os juros devidos; ou impacientes, retirando os depósitos ainda em $t = 1$ e abrindo mão da rentabilidade. Esse comportamento dos depositantes é o que determina a forma do choque de liquidez que será aplicado aos bancos.

Os depositantes podem escolher o período do saque aleatória ou estrategicamente, de acordo com a estrutura de aprendizado EWA. Se os depositantes não utilizam o aprendizado, deve-se definir de forma exógena a probabilidade com que eles agem com impaciência, sendo essa configuração importante para comparar o comportamento da rede frente a diferentes tipos de choque.

2.6 Firmas

No modelo básico, as firmas não agem de forma estratégica, porém são representadas individualmente para que seja possível modelar a heterogeneidade entre elas. As firmas podem, portanto, ser modeladas com diferentes valores de perda dada a inadimplência (*Loss Given Default* - LGD) e taxas de juros dos empréstimos tomados.

É importante ressaltar que, neste modelo, a demanda por crédito das firmas é inelástica, isto é, elas irão consumir todos os recursos que os bancos dispõem a elas (oferta de crédito), independente da taxa de juros ou outras condições contratuais. Assim, quando os bancos diminuem os recursos para o setor real, eles estão deixando de atender a uma parte da demanda, permitindo assim que seja possível estudar os impactos dessa estratégia na economia.

2.7 Ciclos

Os ciclos são divididos em três períodos, $t = 0$ representando o momento inicial, $t = 1$ o curto prazo dos depósitos e empréstimos, e $t = 2$ o momento de maturação. Essa

divisão facilita o processo de modelagem, como será apresentado no Capítulo 5, pois em cada período os agentes se comportam de maneira distinta.

No momento inicial, $t = 0$, os agentes tomam suas decisões, escolhendo as estratégias de maneira aleatória ou de acordo com a estrutura de aprendizagem, e o modelo é preparado para o novo ciclo.

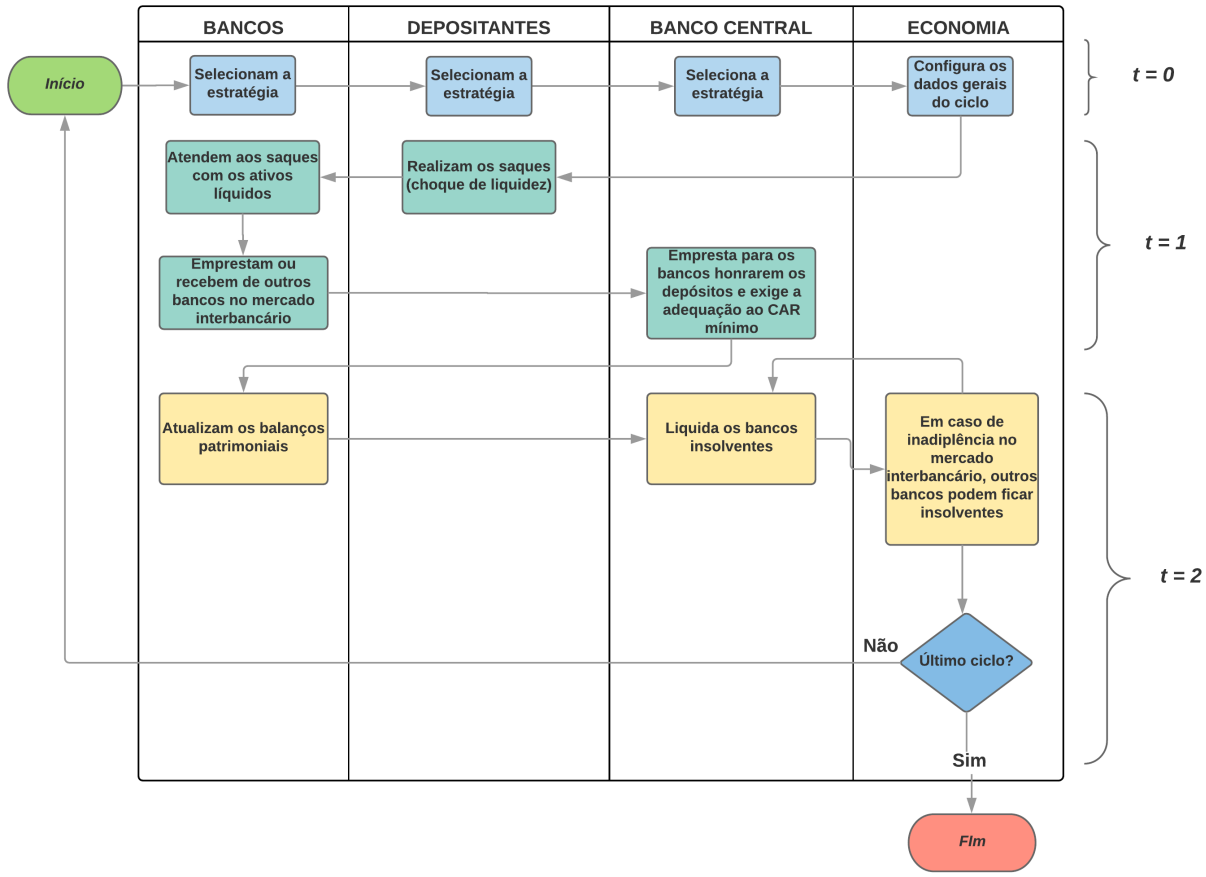
No período $t = 1$, ocorre o choque de liquidez por parte dos depositantes impacientes ou que percebem que seu banco corre risco de quebrar. Os bancos então liquidam seus ativos e, se ainda necessário para atender aos saques, pegam empréstimos de outros bancos no mercado interbancário. O Banco Central pode, por fim, emprestar para os bancos que não conseguirem liquidez suficiente e em seguida exigir de todos a adequação ao CAR mínimo.

No período $t = 2$, os bancos realizam seus lucros, os empréstimos e depósitos maturam, e cada agente atualiza seus dados de aprendizado, que serão necessários para as decisões dos próximos ciclos. Os bancos cujas perdas ultrapassem o capital serão liquidados pelo Banco Central e substituídos por outros iguais. A motivação para esse processo de substituição é que o espaço no mercado, antes ocupado pelo banco liquidado, será preenchido por um banco do mesmo tipo.

A Figura 1 ilustra um fluxograma simplificado de um ciclo de execução do modelo, separando as atividades pelo agente responsável e por cada período.

É importante destacar que os períodos dividem um ciclo em três momentos distintos, relacionados à maturação dos empréstimos e depósitos. Na situação real, porém, os período se sobrepõem, visto que cada empréstimo terá data inicial e tempo de maturação distintos. O modelo, para facilitar a definição das estratégias de cada agente, considera os períodos em sequência, sem sobreposição.

Figura 1 – Fluxograma simplificado dos períodos de execução do modelo



Fonte: Elaborada pelo autor

3 Escolha computacional

Com o objetivo de desenvolver um *framework* de grande alcance a diversos usuários e colaboradores, a escolha computacional tornou-se extremamente crítica, sendo o ponto chave do desenvolvimento da ferramenta.

Foi importante buscar uma linguagem computacional mais acessível para pesquisadores de diversas áreas, em destaque à área econômica, e com ferramentas de análise de dados integradas. Também foi imprescindível escolher a forma de permitir o acesso completo ao *framework*, para qualquer parte do mundo, com disponibilização de arquivos colaborativos e sem perder o controle de versão.

Este capítulo dedica-se a apresentar as justificativas técnicas para as escolhas computacionais realizadas no trabalho.

3.1 Python

O Python[1] é uma linguagem de programação de alto nível amplamente utilizada, criada por Guido van Rossum e lançada originalmente em 1991. É uma linguagem interpretada com uma filosofia de projeto que prioriza um código de fácil entendimento, com uma sintaxe que permite um menor número de linhas de código em comparação a linguagens como Java e C++. O Python também suporta os paradigmas de orientação a objetos e programação funcional.

Com as vantagens inerentes à linguagem, a comunidade científica elegeu o Python para desenvolvimento e hoje temos inúmeros pacotes com aplicações numéricas e científicas, como o SciPy[2], que é uma biblioteca de código aberto focada em *softwares* para matemática, ciência e engenharia, incluindo, dentre outros, o NumPy[3] e o Pandas[4], especializados em trabalhar com dados matriciais e em painéis. Assim, o Python tem se tornado cada vez mais popular na comunidade científica e nos projetos de código aberto.

Há ainda uma biblioteca para análise de redes complexas desenvolvida em Python chamada NetworkX[5], que permite a criação, manipulação e estudo da estrutura, dinâmica e funções de redes complexas.

O Python também possui uma biblioteca chamada Mesa, que surgiu como uma

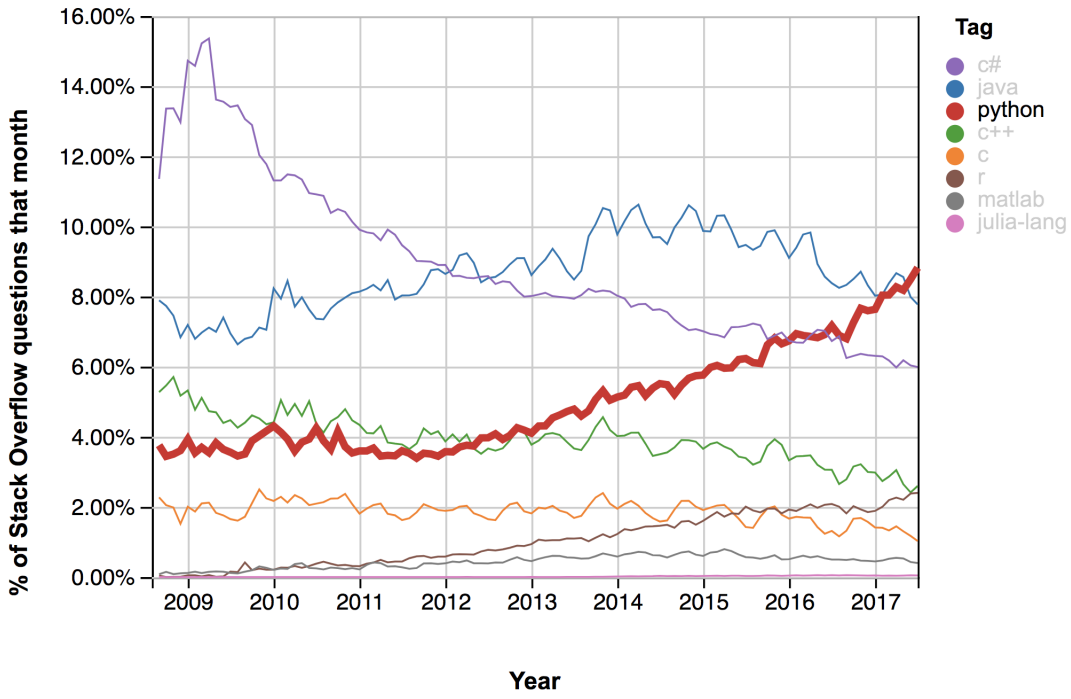
[1] Disponível em <https://www.python.org>.
[2] Disponível em <https://www.scipy.org>.
[3] Disponível em <http://www.numpy.org>.
[4] Disponível em <http://pandas.pydata.org>.
[5] Disponível em <https://networkx.github.io>.

alternativa para o uso do NetLogo[6], permitindo uma maior liberdade na modelagem, execução e análise dos dados da simulação. Mesa será descrita em detalhes no Capítulo 4.

Na Figura 2, vemos que o Python vem ganhando popularidade no Stack Overflow[7] em percentual de perguntas. Atualmente, já está a frente das principais linguagens, incluindo Java e C++ (Stack Overflow, 2017).

Em David Robinson (2017), é apresentada uma análise do crescimento do Python. Nos países de maior renda (*high-income economy*, como definido pelo *World Bank*), ele tem crescido a uma taxa anual de 27% e, em junho de 2017, foi o termo mais buscado no Stack Overflow. O trabalho também analisa os dados dos países restantes (incluindo o Brasil) e conclui que nesses países o crescimento do Python tem sido ainda maior.

Figura 2 – Crescimento das principais linguagens de acordo com o percentual de perguntas no Stack Overflow



Fonte: Stack Overflow (2017)

Na Figura 3, vemos o resultado de uma análise feita nos projetos de código aberto hospedados no GitHub, realizada anualmente pelos administradores do serviço de hospedagem. Em 2017, o Python ultrapassou o Java e tornou-se a segunda linguagem

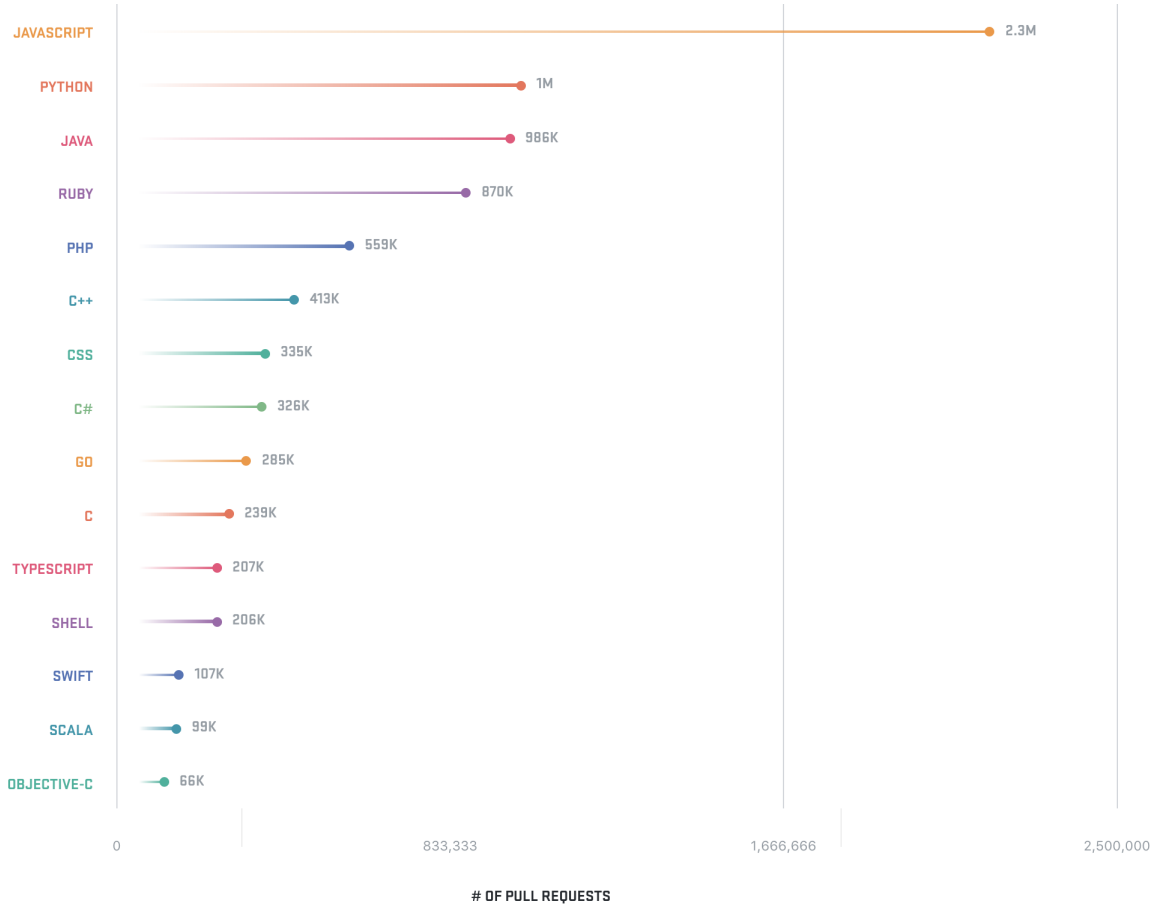
[6] NetLogo é um ambiente programável de modelagem multiagente. Disponível em <https://ccl.northwestern.edu/netlogo>.

[7] Fundado em 2008, é a maior comunidade online para compartilhamento de conhecimento entre programadores. Disponível em <https://stackoverflow.com>.

entre 337 com maior número total de contribuições (*pull requests*) em projetos hospedados na plataforma, ficando atrás apenas do Javascript, linguagem utilizada primariamente em componentes visuais de páginas *web* (GitHub Inc, 2017b).

É importante destacar que as linguagens R[8] e MATLAB[9], apesar de serem populares e utilizadas para análise de dados, não costumam ser escolhidas para soluções mais complexas ou projetos colaborativos, o que explica sua ausência na lista apresentada na Figura 3.

Figura 3 – As 15 linguagens mais populares no GitHub em 2017



Fonte: GitHub Inc (2017b)

De acordo com os dados apresentados, conclui-se que o Python é a linguagem mais adequada atualmente ao desenvolvimento de aplicações científicas para análise de dados, pela quantidade e qualidade de suas bibliotecas de uso gratuito. É também a linguagem mais propícia a um projeto que visa à ampla colaboração da comunidade, como pode-se notar pela quantidade de projetos de código aberto em Python que estão ativamente hospedados no GitHub.

[8] Disponível em <https://www.r-project.org/>.
[9] Disponível em <https://www.mathworks.com/products/matlab.html>.

3.2 GitHub

O GitHub é um serviço *online* que hospeda projetos oferecendo controle de versão com Git. É amplamente utilizado para código fonte, mas pode hospedar qualquer tipo de documento para produção colaborativa (arquivos LaTeX, por exemplo), pois o controle de versão distribuído oferece diversas funcionalidades, como requisição de mudanças, controles de acesso e até mesmo *wiki* personalizada.

Contando com mais de 74 milhões de projetos hospedados (GitHub Inc, 2017a) e sendo o 70º *website* mais popular do planeta (Alexa Internet Inc, 2017), o GitHub é a plataforma ideal para hospedagem de projetos que desejam alcançar um grande número de colaboradores e usuários pelo mundo.

4 Desenvolvimento

Com o objetivo de disponibilizar um código robusto e organizado conforme padrões de desenvolvimento de modelagem baseada em agentes, foi escolhida uma biblioteca específica para servir de base inicial do *framework*. Esta biblioteca, chamada Mesa[1], é escrita em Python e possui código aberto com a licença Apache 2.0[2], que permite seu uso e distribuição em outros *softwares*, inclusive proprietários.

Mesa possibilita a criação rápida de modelos baseados em agentes a partir de um conjunto de componentes-chave, como agendadores de tarefa e redes espaciais. Também facilita a visualização da simulação, através de uma interface com gráficos no navegador. Pelo fato de ser em Python, é possível ainda analisar os dados utilizando as ferramentas de análise já disponíveis, como o Pandas (MASAD; KAZIL, 2015).

Há outras ferramentas de modelagem baseada em agentes, mas Mesa é a única desenvolvida em Python, que tem ganhado bastante popularidade como linguagem de computação científica, possuindo um conjunto maduro de ferramentas de análise de dados, além de ser uma linguagem de programação mais geral. Isso explica o rápido crescimento e adoção da biblioteca Mesa, que possui atualmente 41 contribuidores (The Project Mesa Team, 2017), incluindo o autor desta dissertação, em razão de necessidades específicas deste projeto de pesquisa.

Mesa possui algumas vantagens em relação a outras bibliotecas, pois permite um maior controle no agendamento dos agentes, disponibiliza um coletor de dados exógeno cujo código não se confunde com o modelo, e uma opção de execução em lotes, exportando os dados diretamente para a ferramenta de análise. Fornece ainda uma visualização passo a passo no navegador, como forma de avaliar o comportamento do modelo desenvolvido (MASAD; KAZIL, 2015).

4.1 Arquitetura

O princípio da arquitetura do Mesa é a modularidade. Ele faz o mínimo de suposições acerca do modelo que será implementado, fornecendo um conjunto de componentes que podem ser facilmente combinados e estendidos para criar diferentes tipos de modelo.

Os módulos são divididos em três categorias distintas: modelagem, análise e visualização. Os componentes de modelagem são as peças-chave necessárias para a definição das propriedades do modelo, sendo compostos pelas seguintes classes:

[1] Disponível em <https://github.com/projectmesa/mesa>.
[2] A licença Apache 2.0 pode ser lida em <https://www.apache.org/licenses/LICENSE-2.0>.

1. A classe `Model` armazena os parâmetros gerais do modelo e age como um agregador dos outros componentes.

2. A classe `Agent` descreve os agentes do modelo através de subclasses, uma para cada tipo distinto de agente.

3. A classe `Scheduler` controla o regime de ativação dos agentes e o tempo de execução geral da simulação.

4. As classes `SpaceGrid` e `NetworkGrid` representam os espaços onde os agentes ficam situados, podendo ser um plano cartesiano ou uma rede.

Os componentes de análise são formados pela classe `DataCollector`, usada para guardar os dados da simulação em todos os ciclos, e pela classe `BatchRunner`, para automatizar diversas execuções do modelo. Por último, há os componentes de visualização, usados para mapear os objetos do modelo em representações visuais no navegador *web*, permitindo uma análise mais minuciosa da execução da simulação.

A Figura 4 apresenta um diagrama em Linguagem de Modelagem Unificada (*Unified Modeling Language* - UML) da arquitetura descrita, conforme apresentado em Masad e Kazil (2015).

4.2 Modelagem

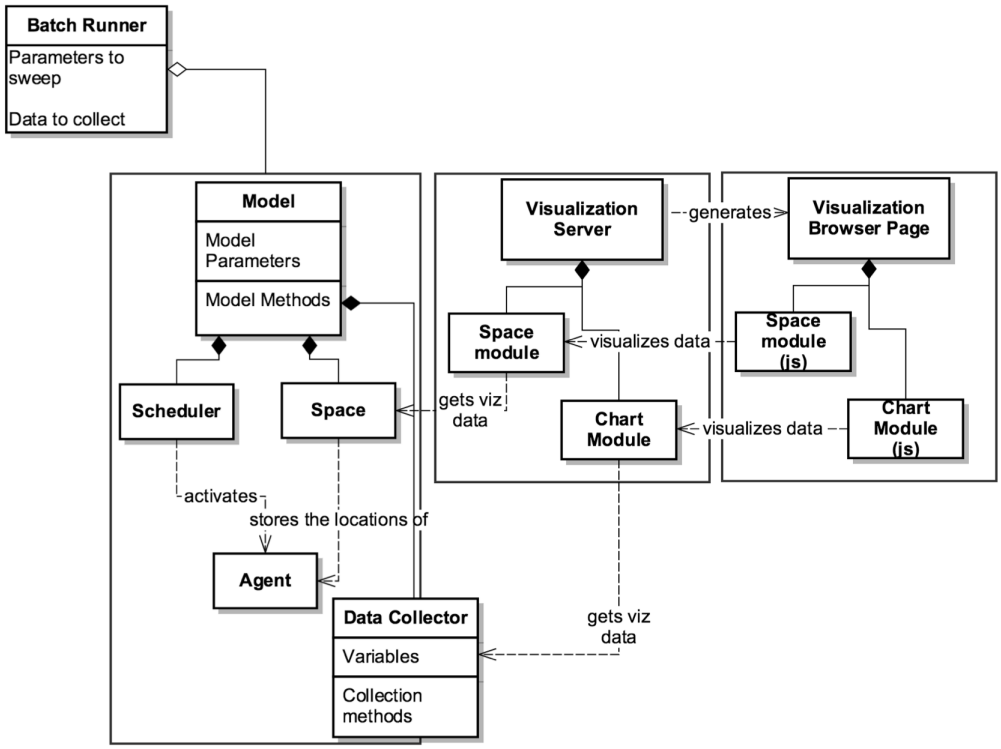
Para exemplificar o processo de modelagem, será implementada uma versão de modelo de distribuição de riqueza[3]. Nesse modelo, todos os agentes começam com a mesma quantia de dinheiro e, a cada passo, cada agente com ao menos uma unidade de dinheiro transfere uma unidade para outro agente de maneira aleatória. À medida que a simulação avança, a distribuição de riqueza entre os agentes muda de perfeitamente uniforme para uma viesada, pois muitos agentes ficam totalmente sem dinheiro.

Esse modelo de distribuição de riqueza foi inicialmente estudado por Cordier, Pareschi e Toscani (2005) e ganhou o nome de modelo de distribuição de riqueza de Boltzmann, pois a curva de distribuição final é análoga ao resultado da equação de Boltzmann, que mostra como as velocidades das moléculas são distribuídas em um gás ideal.

Para iniciar a modelagem, é necessário primeiro criar a classe do modelo e uma classe para cada tipo distinto de agente. O parâmetro do modelo é o número de agentes, e cada agente possui como parâmetro a quantidade de dinheiro. Para os agentes, também é necessário definir o comportamento, ou ações, através de métodos na classe. No presente

[3] O código completo está disponível em <https://github.com/projectmesa/mesa/tree/master/examples>.

Figura 4 – Diagrama UML simplificado da arquitetura da biblioteca Mesa



Fonte: Masad e Kazil (2015)

caso, a única ação de cada agente é dar uma unidade financeira para outro agente, implementada no método `step`. A implementação inicial está ilustrada na Figura 5.

O próximo componente que devemos adicionar ao modelo é o agendador (na classe `Scheduler`). Como nas simulações o tempo é discreto, precisamos de uma maneira prática para definir a ordem de ativação dos agentes. A biblioteca Mesa, portanto, possui o agendador em um componente separado, não havendo dificuldades em alterá-lo de forma a verificar os impactos desse tipo de mudança no modelo. O agendador também funciona como um pacote que agrega todos os agentes do modelo.

Os tipos principais de agendamento são a ativação síncrona, na qual todos os agentes tomam a decisão ao mesmo tempo e somente depois é atualizado o estado do modelo; a ativação uniforme, na qual todos os agentes são ativados na mesma ordem em todos os passos; e a ativação aleatória, na qual a ordem de ativação dos agentes pode alterar entre os diversos passos do modelo. Porém, estendendo a classe `BaseScheduler`, não há dificuldade em personalizar as regras de ativação.

O agendador é inicializado dentro do objeto do modelo e os agentes são nele inseridos logo ao serem criados (também podem ser adicionados ou removidos a qualquer momento

Figura 5 – *Boltzmann Wealth Model*, exemplo 1

```
from mesa import Model, Agent

class MoneyAgent(Agent):

    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.wealth = 1

    def step(self):
        if self.wealth > 0:
            self.give_money()

class MoneyModel(Model):

    def __init__(self, N):
        self.num_agents = N

    def create_agents(self):
        for i in range(self.num_agents):
            a = MoneyAgent(i)
```

Fonte: Elaborada pelo autor

da simulação). A classe do modelo também tem o método `step`, que ativa o agendador para chamar o método `step` dos agentes na ordem definida. No modelo apresentado neste capítulo, por exemplo, o agendador será de ativação aleatória, conforme Figura 6.

Figura 6 – *Boltzmann Wealth Model*, exemplo 2

```
from mesa.time import RandomActivation

class MoneyModel(Model):

    def __init__(self, N):
        self.num_agents = N
        self.schedule = RandomActivation(self)

    def create_agents(self):
        for i in range(self.num_agents):
            a = MoneyAgent(i, self)
            self.schedule.add(a)

    def step(self):
        self.schedule.step()
```

Fonte: Elaborada pelo autor

O componente seguinte a ser adicionado ao modelo é o componente espacial. Dessa

forma, os agentes possuem posições em uma grade ou rede, podendo se deslocar entre as posições e, inclusive, condicionar à posição atual a forma com que interagem entre si. Será utilizada neste exemplo a classe `NetworkGrid`, permitindo a criação de uma rede na qual os agentes ocuparão os vértices. Como regra do modelo, determinaremos que os agentes apenas troquem riqueza entre os vizinhos, ou seja, os que estão em nós adjacentes.

Antes de criar o componente espacial de rede, deve-se criar a estrutura de grafo que será utilizada. Para isso, utiliza-se a biblioteca NetworkX, que possui ferramentas para criar diversas estruturas distintas de rede. No exemplo, foi criado um grafo binomial (Erdős-Rényi) através do método `nx.erdos_renyi_graph(n=number_nodes, p=0.5)`.

A classe que representa a estrutura espacial já possui os métodos de posicionar o agente (usado quando os agentes são criados ou se movem) e de verificar quais agentes estão na adjacência. É importante ressaltar que o modelo sempre tem acesso à rede, podendo ler ou alterar qualquer propriedade durante a execução. Os códigos da Figura 7 e da Figura 8 ilustram os conceitos apresentados até o momento.

Figura 7 – *Boltzmann Wealth Model*, exemplo 3 (Modelo)

```
class MoneyModel(Model):

    def __init__(self, num_agents, num_nodes):
        super().__init__()
        self.num_agents = num_agents
        self.num_nodes = num_nodes if num_nodes >= self.num_agents else
                                                self.num_agents
        self.G = nx.erdos_renyi_graph(n=self.num_nodes, p=0.5)
        self.grid = NetworkGrid(self.G)
        self.schedule = RandomActivation(self)

        list_of_random_nodes = random.sample(self.G.nodes(), self.
                                                num_agents)

        for i in range(self.num_agents):
            a = MoneyAgent(i, self)
            self.schedule.add(a)
            self.grid.place_agent(a, list_of_random_nodes[i])

    def step(self):
        self.schedule.step()
```

Fonte: Elaborada pelo autor

Um modelo de simulação não é útil se não for possível verificar o comportamento dos agentes e as saídas produzidas pela execução. Há basicamente duas maneiras de extrair esses dados, uma qualitativa e outra quantitativa. A análise qualitativa é feita através da visualização direta, cujos componentes facilitam a geração e a apresentação de gráficos e tabelas em um navegador *web*. Para a análise quantitativa, a biblioteca possui um

Figura 8 – *Boltzmann Wealth Model*, exemplo 3 (Agente)

```
class MoneyAgent(Agent):

    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.wealth = 1

    def move(self):
        possible_steps = [node for node in self.model.grid.get_neighbors
                                            (self.pos, include_center=
                                            False) if
                          self.model.grid.is_cell_empty(node)]
        if len(possible_steps) > 0:
            new_position = random.choice(possible_steps)
            self.model.grid.move_agent(self, new_position)

    def give_money(self):
        neighbors_nodes = self.model.grid.get_neighbors(self.pos,
                                            include_center=False)
        neighbors = self.model.grid.get_cell_list_contents(
                                            neighbors_nodes)
        if len(neighbors) > 0:
            other = random.choice(neighbors)
            other.wealth += 1
            self.wealth -= 1

    def step(self):
        self.move()
        if self.wealth > 0:
            self.give_money()
```

Fonte: Elaborada pelo autor

componente de coleta de dados, representado pela classe `DataCollector`, que permite o armazenamento e a exportação dos dados desejados do modelo.

O coletor de dados armazena três categorias de informação: variáveis do modelo, variáveis dos agentes e tabelas gerais de dados. As variáveis do modelo e dos agentes são adicionadas ao coletor através de funções que possuem como parâmetros o modelo ou o agente, respectivamente. Quando o método `collect` é chamado, ele executa todas as funções do coletor retornando uma estrutura de dicionário contendo os dados retornados das funções, indexados pelo passo atual da simulação.

O coletor, portanto, deve ser adicionado à classe do modelo e sua chamada de coleta deve ser feita após cada passo em que há interesse nos dados.

A categoria de coletor de tabela é uma forma mais geral de registrar as informações do modelo, pois pode informar toda vez que um determinado evento ocorre, não sendo algo referente a algum passo específico (com isso podemos, por exemplo, registrar o momento

em que um determinado agente muda de estado e os dados referentes a essa mudança).

Internamente, o coletor guarda todas as variáveis e tabelas em dicionários e listas da biblioteca nativa do Python, reduzindo a dependência de bibliotecas extras e permitindo com facilidade a exportação para objetos dos tipos JSON[4] e CSV[5]. Entretanto, uma das grandes vantagens do Mesa é a fácil integração com o Pandas, a maior biblioteca de análise de dados do Python. Portanto, o coletor possui métodos para exportar os dados coletados diretamente para os painéis do Pandas, permitindo de forma rápida e interativa a análise dos dados, produção de gráficos e acesso a diversas ferramentas estatísticas e de aprendizado de máquinas (MASAD; KAZIL, 2015).

Para concluir o exemplo, o código na Figura 9 demonstra a utilização do coletor para verificar a riqueza de cada agente (coletor de variáveis dos agentes) e para calcular o coeficiente de Gini[6] (coletor de variáveis do modelo).

Figura 9 – *Boltzmann Wealth Model*, exemplo 4

```
from mesa.datacollection import DataCollector

class MoneyModel(Model):

    def __init__(self, num_agents, num_nodes):
        # ...
        self.datacollector = DataCollector(
            model_reporters={"Gini": compute_gini},
            agent_reporters={"Wealth": lambda _: _.wealth}
        )

    def step(self):
        self.schedule.step()
        self.datacollector.collect(self)

def compute_gini(model):
    agent_wealths = [agent.wealth for agent in model.schedule.agents]
    x = sorted(agent_wealths)
    N = model.num_agents
    B = sum(xi * (N - i) for i, xi in enumerate(x)) / (N * sum(x))
    return 1 + (1 / N) - 2 * B
```

Fonte: Elaborada pelo autor

Com o modelo montado, é possível executá-lo e analisar os dados. Na Figura 10 é apresentado o código que executa a simulação e gera os gráficos a partir dos dados capturados pelo coletor, conforme as funções utilizadas. O modelo foi então iniciado com 100 agentes distribuídos em uma rede de 120 nós, e executado 100 vezes.

[4] JavaScript Object Notation
[5] Comma-separated values
[6] O Coeficiente de Gini é uma medida de desigualdade, medindo no exemplo a desigualdade de renda.

As chamadas `datacollector.get_model_vars_dataframe()` e `datacollector.get_agent_vars_dataframe()` retornam objetos de painel do Pandas populados com os dados coletados na simulação. Em seguida, utiliza-se as próprias ferramentas visuais integradas ao Pandas para mostrar os gráficos desejados.

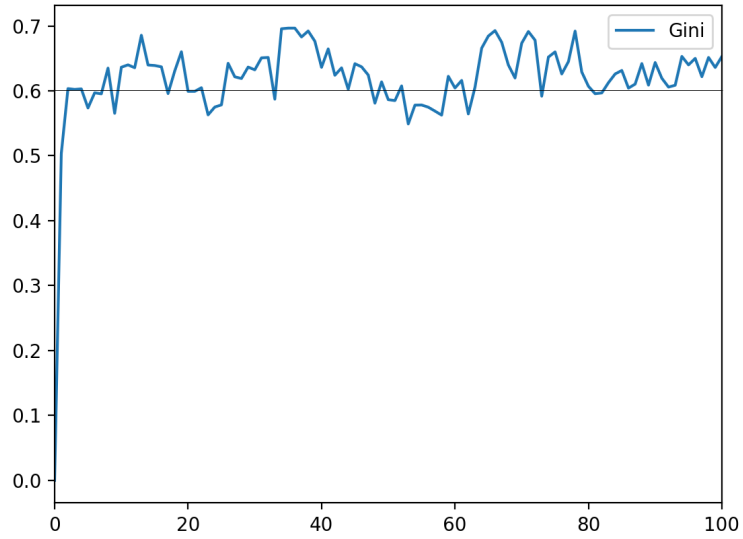
Figura 10 – Execução do modelo *Boltzmann Wealth Model*

```
import matplotlib.pyplot as plt

model = MoneyModel(100, 120) # Inicia o modelo
model.run_model(100) # Executa a simulacao durante 100 ciclos

gini = model.datacollector.get_model_vars_dataframe()
gini.plot() # Apresenta o grafico com os dados do coletor do modelo

agent_wealth = model.datacollector.get_agent_vars_dataframe()
end_wealth = agent_wealth.xs(100, level="Step")["Wealth"]
# Apresenta o grafico com os dados do coletor dos agentes
end_wealth.hist(bins=range(agent_wealth.Wealth.max()))

plt.show()
```

Fonte: Elaborada pelo autor

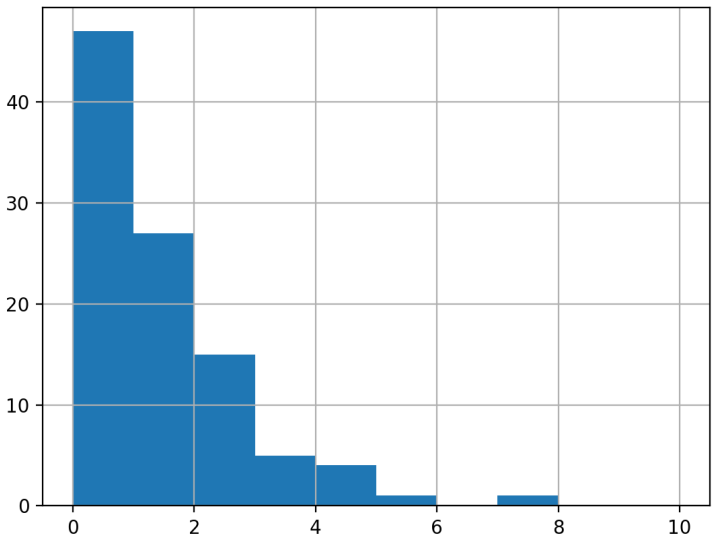
A Figura 11 apresenta o gráfico com a evolução do coeficiente de Gini ao longo de todos os ciclos (O coletor de dados do modelo foi configurado para calcular o coeficiente após cada final de ciclo). A Figura 12, por sua vez, apresenta um histograma com a distribuição da riqueza dos agentes ao final do último ciclo. No modelo, todos os 100 agentes iniciaram com uma unidade de riqueza, porém, ao final, 48% dos agentes ficaram sem riqueza alguma.

Figura 11 – Evolução do coeficiente de Gini ao longo da execução de 100 ciclos



Fonte: Elaborada pelo autor

Figura 12 – Histograma com a distribuição de frequência da riqueza dos agentes ao final do último ciclo



Fonte: Elaborada pelo autor

5 Resultados

5.1 Utilização do *Framework*

O *framework*, nomeado BankSim, foi desenvolvido em Python e encontra-se disponível em repositório público *online* do GitHub para a utilização por outros projetos, permitindo a inclusão de novos parâmetros e modelos de simulação. O endereço *web* do projeto é <https://github.com/banking-project/banksim>.

Para a utilização, o usuário deverá fazer inicialmente um *fork* do projeto em sua conta do GitHub e, então, começar a implementação das modificações do modelo atual, ou a criação dos novos modelos. Todas as alterações feitas pelos novos pesquisadores permanecem disponíveis em suas contas, podendo ainda ser submetidas para análise e possível aceitação no projeto original.

5.2 Simulação

Esta seção apresentará algumas simulações realizadas com o BankSim para demonstrar o uso da ferramenta, bem como sua aplicação nos estudos dos efeitos de políticas regulatórias. É importante destacar que estão disponíveis no repositório todos os códigos utilizados neste capítulo, tanto da execução da simulação, quanto do uso das ferramentas de análise dos dados.

As simulações consistem em um número de mil ciclos[1], repetidos diversas vezes de forma a suavizar a aleatoriedade, para que possa ser observada a evolução das variáveis de interesse ao longo dos ciclos.

As variáveis estudadas são as seguintes:

1. Alocação de Capital: mede a quantidade total de capital alocado pelos bancos, para enfrentar riscos e manter o CAR adequado.

2. Dependência Interbancária: captura a quantidade total de empréstimos no mercado interbancário. É uma medida de risco de contágio.

3. Insolvências: representa o percentual de bancos que ficaram insolventes, independentemente do motivo.

[1] Mil ciclos mostrou-se suficiente para que as variáveis de estudo se estabilizassem.

Cada ciclo consiste de três períodos, $t = 0, 1, 2$, conforme apresentado no Capítulo 2. As variáveis foram medidas ao final de cada ciclo, após o período $t = 2$. Os parâmetros EWA utilizados nas simulações são apresentados na Tabela 3.

Tabela 3 – Parâmetros EWA

Símbolo	Parâmetro	Valor Utilizado
ρ	Fator de desconto retrospectivo	0
ϕ	Taxa de decaimento de atrações anteriores	1
$N(0)$	Experiência inicial	1
δ	Peso dos ganhos passados	1
$A(0)$	Atração inicial	0
λ	Sensibilidade às atrações	1

Fonte: Elaborada pelo autor

O modelo será simulado com um conjunto de 120 bancos, cada um com 50 firmas e 100 depositantes. Também estará presente o Banco Central, atuando como emprestador de última instância, provendo liquidez aos bancos que não a conseguirem no mercado interbancário.

Os bancos possuem um tamanho inicial determinado por uma distribuição de Pareto tipo II (distribuição de Lomax)[2] com parâmetro α, para haver assimetria nos tamanhos. Os depositantes não consideram o tamanho do banco para a decisão do saque, que é determinada por uma probabilidade exógena, e as firmas possuem os mesmos valores de risco para seus projetos.

Os parâmetros gerais de simulação são apresentados na Tabela 4.

Como indicador de eficiência bancária, pode-se usar a diferença entre os juros que os bancos cobram das firmas e os que remuneram os depósitos (*interest spread*). Como no modelo as taxas de juros são dadas de forma exógena, para medir o impacto econômico do *spread* bancário, realizou-se as simulações em dois cenários, com diferença apenas na taxa cobrada nos empréstimos às firmas, mantendo a taxa de remuneração dos depósitos constante em 0,5%:

1. *HighSpread*, no qual a taxa de juros para as firmas é de 8%.

2. *LowSpread*, no qual a taxa de juros para as firmas é de 6%.

A Figura 13 compara a evolução do capital agregado dos bancos. Com o objetivo de maximizar o ROE, eles escolhem inicialmente as estratégias com menor alocação de

[2] A função densidade de probabilidade da distribuição de Pareto utilizada é dada por $p(x) = \frac{\alpha}{(x+1)^{(\alpha+1)}}$.

Tabela 4 – Parâmetros Gerais

Parâmetro	Valor Utilizado
Número de repetições	50
Número de ciclos	1000
Número de bancos	120
Número de depositantes por banco	100
Número de firmas por banco	50
Parâmetro de tamanho dos bancos (α de Pareto)	5
Taxa de retorno dos depósitos	0,5%
Probabilidade de saque antecipado	15%
Taxa de juros no mercado interbancário	1%
Taxa de juros punitiva (Banco Central)	2,5%
Depreciação de ativos ilíquidos	25%
Depreciação dos empréstimos de bancos insolventes	40%

Fonte: Elaborada pelo autor

capital. Porém, como pode ser visto na Figura 14, a proporção de bancos insolventes nos primeiros ciclos é grande. As figuras apresentam os valores médios das variáveis a cada ciclo, com o intervalo de confiança de 95%.

Assim, com o passar dos ciclos, os bancos aprendem que devem aumentar a alocação de capital (aceitando a diminuição do ROE) de forma a evitar a insolvência. Nota-se nas figuras citadas que, por volta do ciclo de número 200, os bancos mantêm um valor suficiente de capital para evitar as insolvências, que se tornam bem mais raras.

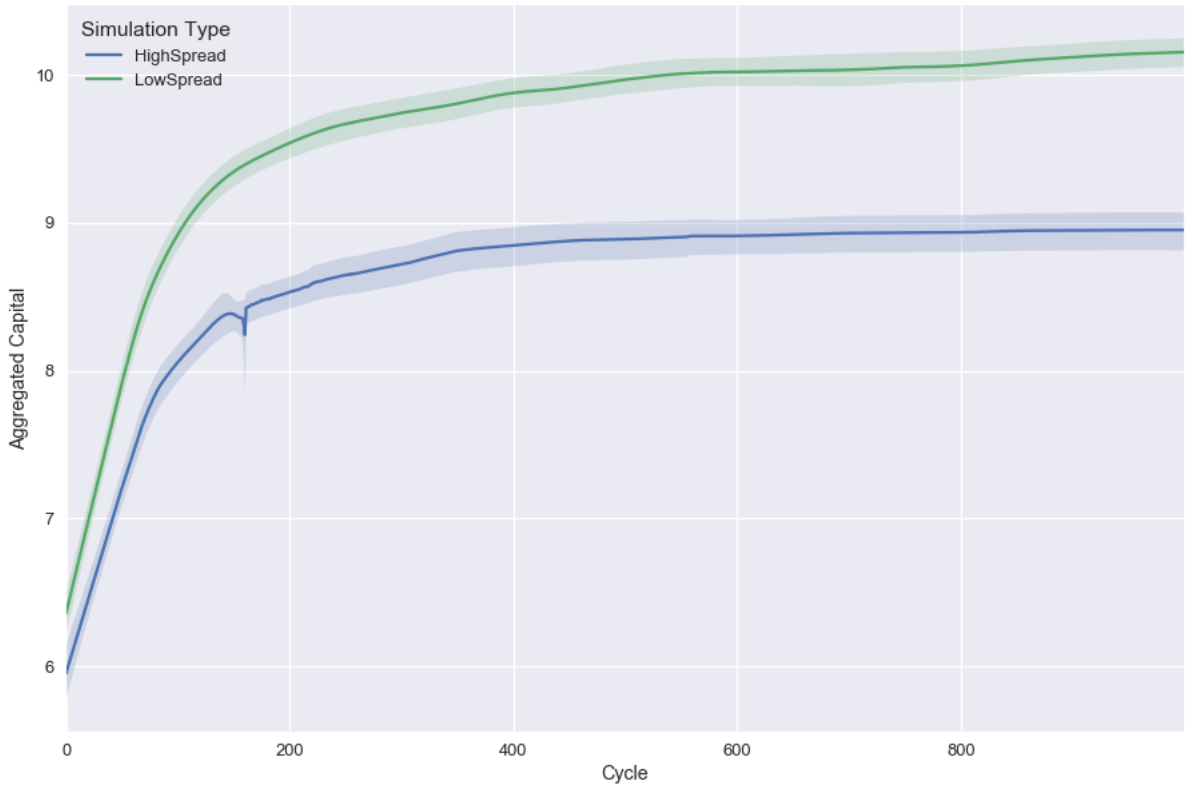
Na Figura 15, é possível verificar que os bancos, nos ciclos iniciais, tomam mais empréstimos no mercado interbancário no cenário de baixo *spread*, em comparação ao cenário de alto *spread*. Com um *spread* maior, eles escolhem estratégias com uma maior interconexão bancária ao longo dos ciclos, aumentando o risco de contágio na rede.

A Figura 16 mostra a estrutura da rede bancária em uma das simulações, ao final de mil ciclos. Dos 120 bancos da rede, optou-se por mostrar apenas os que estavam conectados através da rede interbancária. O sentido das setas indica o fluxo de empréstimo financeiro, enquanto a largura indica o volume.

É de interesse também o estudo da estrutura da rede interbancária formada. O trabalho de Silva et al. (2016), por exemplo, mostra um estudo de eficiência bancária de estruturas de redes complexas do tipo centro-periferia (*core-periphery*).

A Figura 17 apresenta a evolução do coeficiente de centralidade de núcleo (*coreness centrality measure*) da rede ao longo dos ciclos. Esse coeficiente calculado é definido como o coeficiente de correlação de *Pearson* não normalizado entre a estrutura da rede analisada

Figura 13 – Capital Agregado dos Bancos



Fonte: Elaborada pelo autor

e a estrutura de uma rede centro-periferia ideal, conforme Borgatti e Everett (2000).

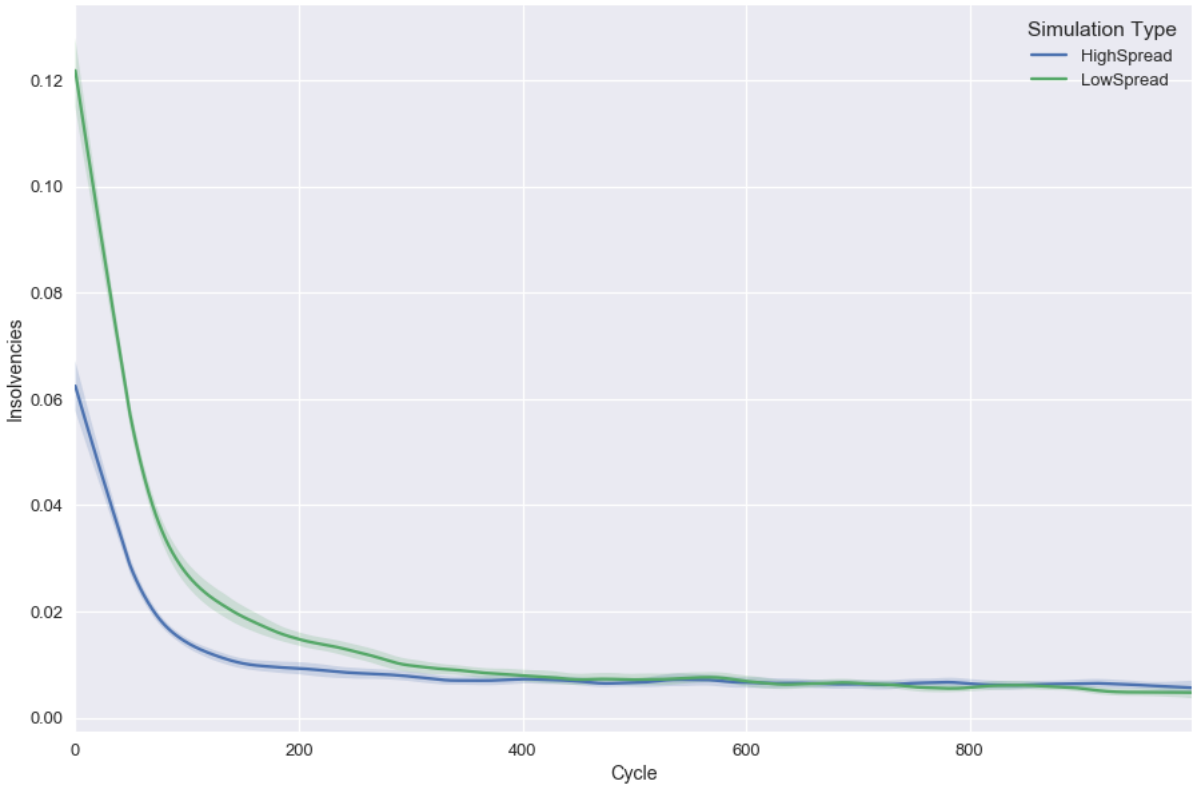
Na simulação apresentada, a rede formada possui um coeficiente de centralidade de núcleo alternando entre 39% e 42%. Assim, a rede assume a forma aproximada de uma rede centro-periferia ideal, com poucos bancos formando o centro da rede, fornecendo liquidez a um número maior de bancos, que formam a periferia da rede. A simulação confirma, portanto, que os bancos no modelo implementado formam redes no mercado interbancário de acordo com o observado empiricamente (LELYVELD et al., 2014).

A Figura 18 apresenta o histograma do coeficiente de centralidade de núcleo do banco 103, escolhido por ser o principal núcleo da rede no último ciclo. É possível verificar que, apesar de esse banco ser o principal núcleo da rede em 380 ciclos, durante 570 ciclos ele esteve com centralidade próxima de zero.

5.2.1 Validação Visual do Modelo

Nas figuras 19 e 20, são apresentados alguns dados de interesse em uma simulação do tipo *HighSpread*. Com os componentes visuais do *framework*, é possível montar gráficos com os dados desejados e desenhar a rede bancária, utilizando um componente específico

Figura 14 – Proporção de Bancos Insolventes



Fonte: Elaborada pelo autor

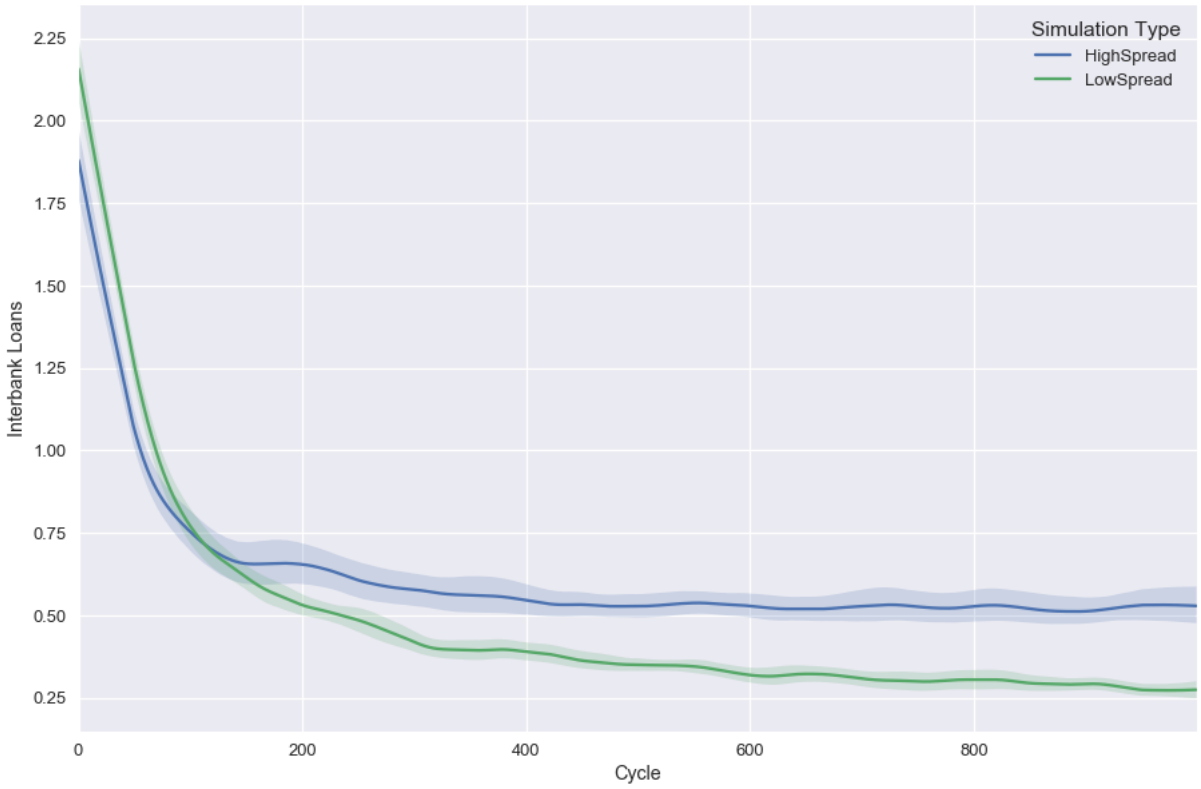
de rede baseado no D3.js[3]. Esses componentes ajudam a validar as regras do modelo, permitindo enxergar as variáveis de interesse a cada ciclo.

5.2.2 Complexidade Computacional

No desenvolvimento de *software* é importante avaliar a complexidade computacional do algoritmo utilizado. Como a simulação depende da interação entre os agentes, principalmente entre os bancos, foi realizado o estudo do tempo de execução de dez ciclos da simulação, em segundos, para diferentes quantidades de bancos. O resultado, apresentado na Figura 21, mostra que o tempo de execução cresce de forma exponencial em relação ao número de bancos presentes na rede interbancária.

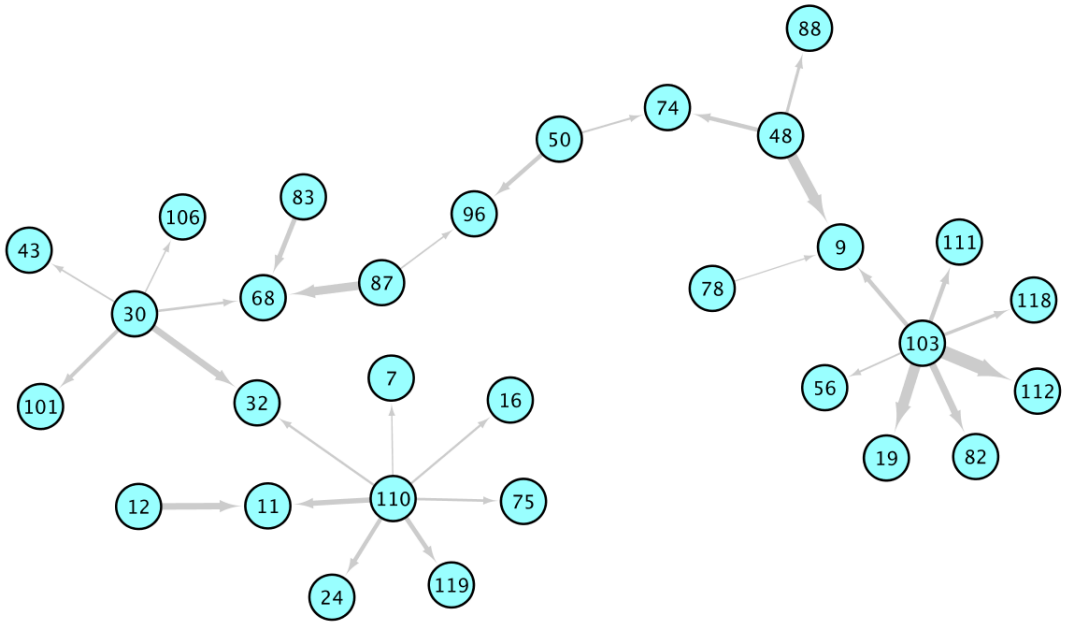
[3] D3.js é uma biblioteca em JavaScript para visualização de dados. Disponível em <https://d3js.org/>.

Figura 15 – Empréstimos Interbancários



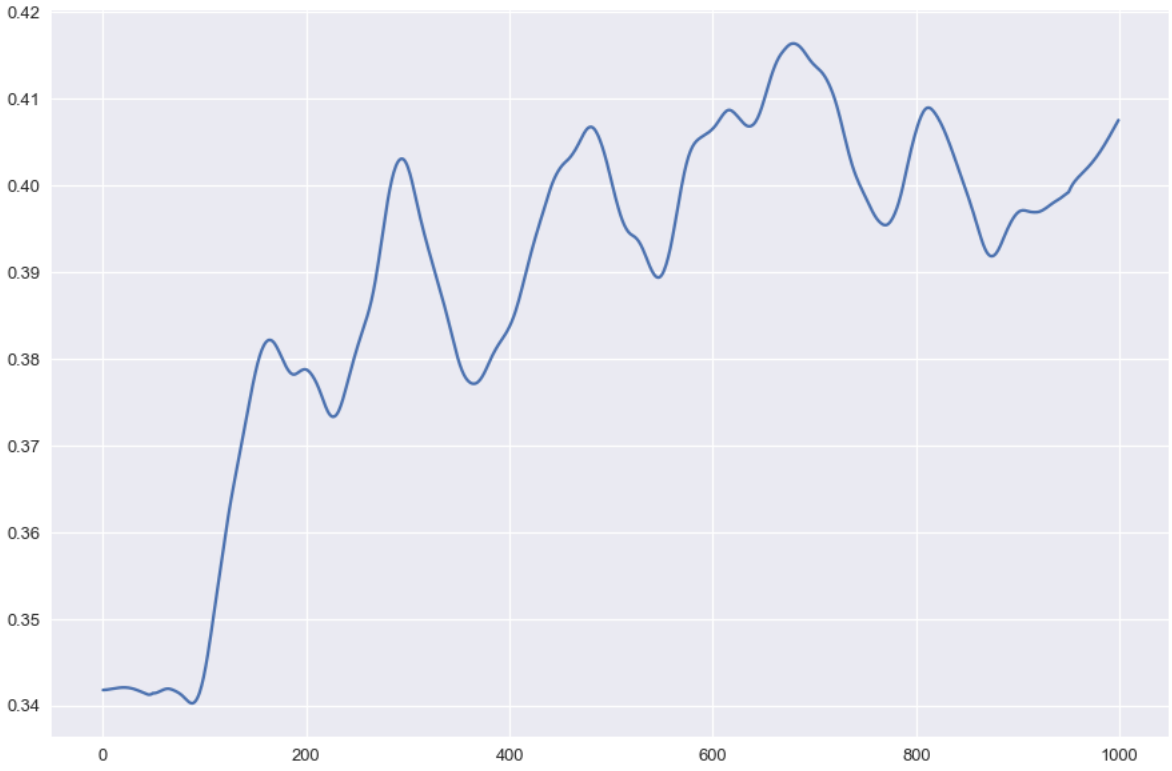
Fonte: Elaborada pelo autor

Figura 16 – Rede Interbancária ao Final do Último Ciclo



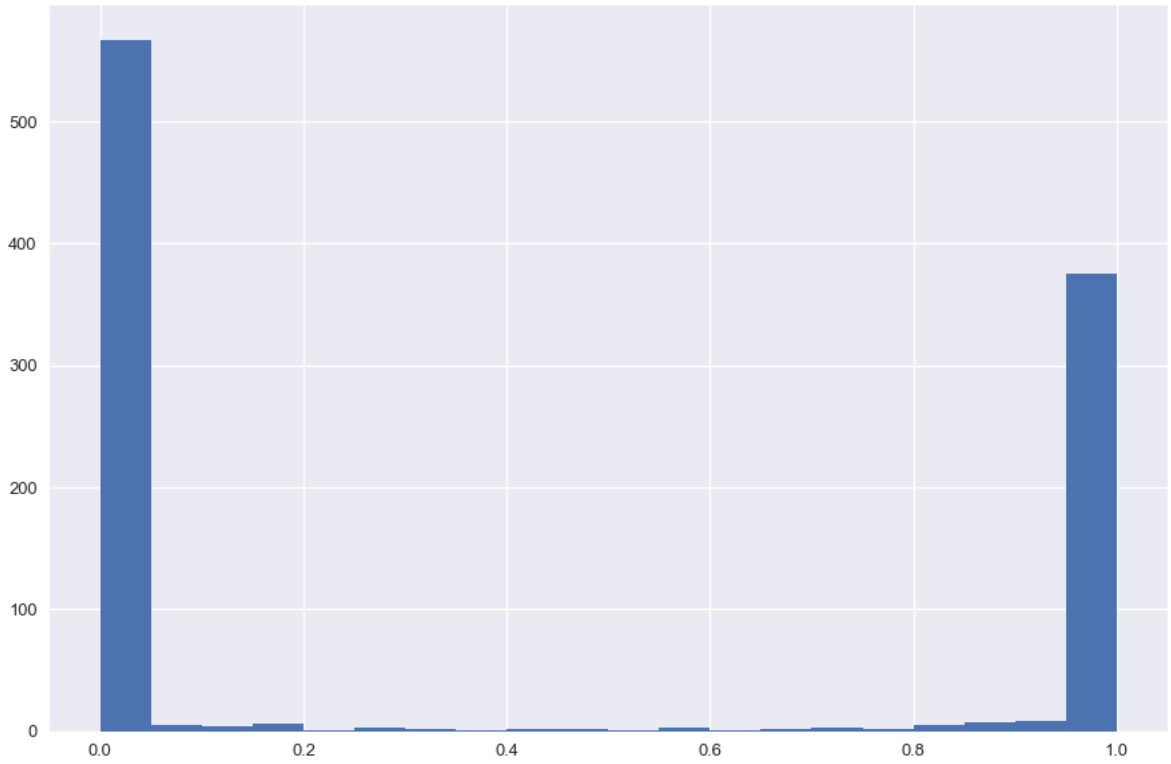
Fonte: Elaborada pelo autor

Figura 17 – Centralidade da Rede Interbancária



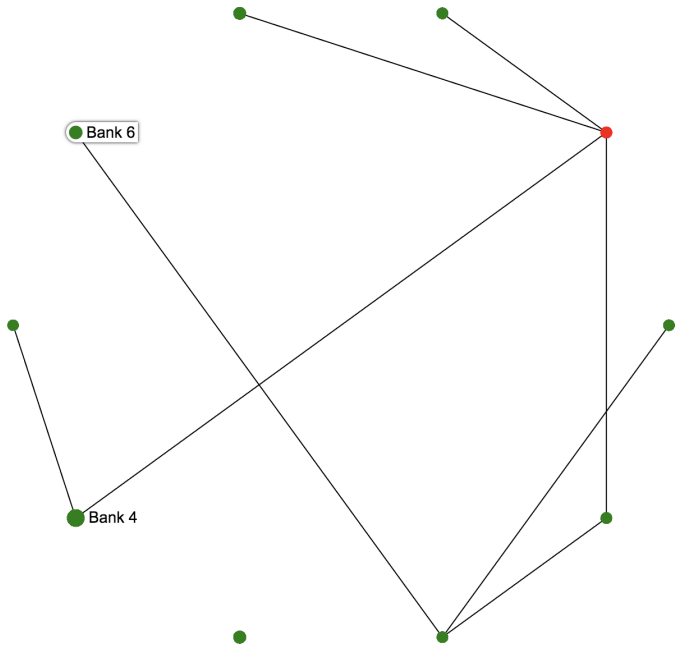
Fonte: Elaborada pelo autor

Figura 18 – Histograma do Coeficiente de Centralidade de Núcleo do Banco 103, ao longo dos ciclos



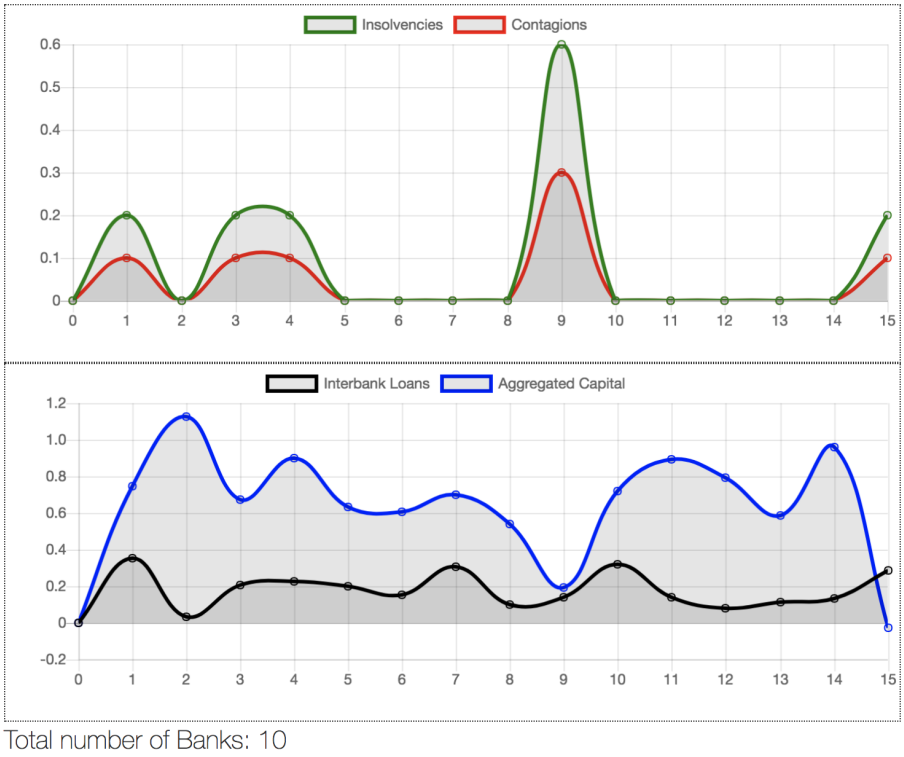
Fonte: Elaborada pelo autor

Figura 19 – Estrutura da Rede Bancária Durante a Simulação



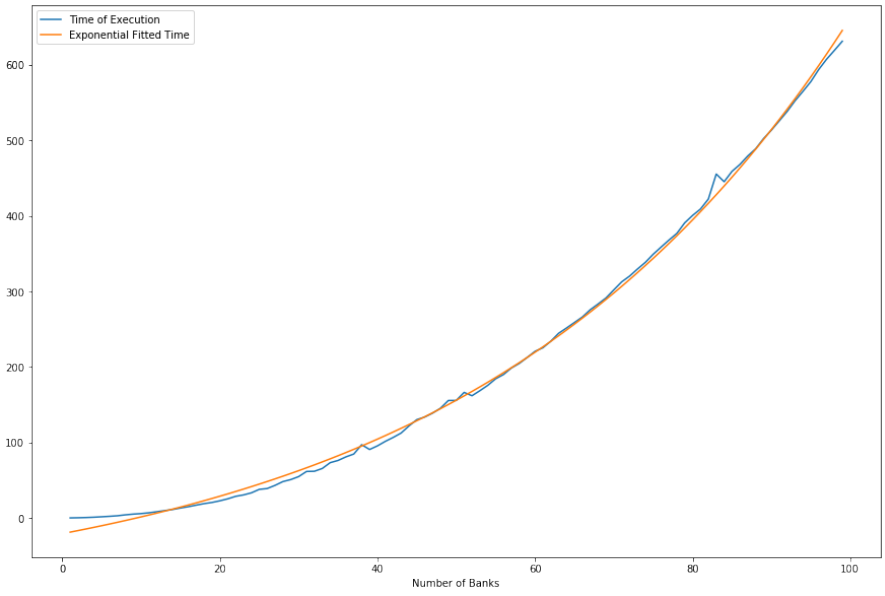
Fonte: Elaborada pelo autor

Figura 20 – Variáveis de Interesse ao Longo dos 15 Primeiros Ciclos



Fonte: Elaborada pelo autor

Figura 21 – Tempo de execução por quantidade de bancos



Fonte: Elaborada pelo autor

6 Conclusão

Este trabalho traz uma contribuição metodológica ao descrever um modelo de simulação de redes bancárias baseado em agentes com a finalidade de estudar o impacto da regulação no setor bancário, considerando o comportamento estratégico dos agentes frente às mudanças nas restrições e no ambiente econômico. O modelo permite ainda que os agentes sejam heterogêneos, capazes de fazer escolhas, e possam definir suas estratégias de acordo com experiências anteriores através do aprendizado EWA.

O trabalho traz ainda uma contribuição tecnológica ao entregar um *framework* computacional, desenvolvido em linguagem Python e com ferramentas de código livre, disponibilizado em repositório público *online*[1]. Dessa forma, pesquisadores interessados de qualquer instituição do mundo poderão avaliar, criticar e agregar novas funcionalidades ao modelo ou ao comportamento dos agentes. Também apresenta, por meio de exemplos computacionais, a utilização do *framework* para a simulação e avaliação de diferentes políticas regulatórias.

Dada a complexidade dos problemas estudados pelo modelo, pode-se considerar que a versão de simulação implementada neste trabalho contém muitas simplificações da economia real. Assim, o modo como o *framework* foi desenvolvido facilita trabalhos futuros que visem validar os resultados com dados reais. Por exemplo, o trabalho de Silva et al. (2016) apresenta um estudo de eficiência bancária de estruturas de redes complexas do tipo centro-periferia (*core-periphery*) com dados reais da economia brasileira. Uma proposta de trabalho futuro seria validar o presente modelo com os achados publicados no trabalho citado.

Como possibilidade de extensão do modelo e sugestão adicional de trabalho futuro, destaca-se o estudo da formação da rede interbancária conforme apresentado em Cajueiro (2005), que enumera fundamentos para a formação de diferentes tipos de redes complexas, considerando que cada nó na rede seja um agente desejando satisfazer sua preferência econômica, e que as conexões entre os nós dependa dos benefícios e custos associados a esse vértice.

[1] <https://github.com/banking-project/banksim>

Referências

Alexa Internet Inc. *GitHub Traffic Statistics*. 2017. Disponível em: <https://www.alexa.com/siteinfo/github.com>. Acesso em: 13 dez 2017. Citado na página 36.

ALLEN, F.; GALE, D. Optimal financial crises. *The Journal of Finance*, Wiley Online Library, v. 53, n. 4, p. 1245–1284, 1998. Citado na página 23.

ALLEN, F.; GALE, D. Financial contagion. *Journal of political economy*, JSTOR, v. 108, n. 1, p. 1–33, 2000. Citado na página 23.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR ISO/IEC 9126-1: Engenharia de software - Qualidade de produto*: Parte 1: Modelo de qualidade. Rio de Janeiro, 2003. 21 p. Citado 2 vezes nas páginas 3 e 17.

AXTELL, R. Why agents?: on the varied motivations for agent computing in the social sciences. Center on Social and Economic Dynamics Brookings Institution, 2000. Citado 2 vezes nas páginas 19 e 20.

BARROSO, R. V. *Modelo dinâmico computacional de rede de bancos*. Dissertação (Mestrado) — Universidade de Brasília, 2011. Citado na página 23.

BARROSO, R. V. *Avaliação de políticas regulatórias e de estrutura do mercado financeiro em um modelo dinâmico de sistema bancário com aprendizado*. Tese (Doutorado) — Universidade de Brasília, 2014. Citado 3 vezes nas páginas 16, 17 e 23.

BARROSO, R. V. et al. Interbank network and regulation policies: an analysis through agent-based simulations with adaptive learning. *Journal Of Network Theory In Finance*, v. 2, n. 4, p. 53–86, 2016. Citado 3 vezes nas páginas 16, 17 e 23.

BEALE, N. et al. Individual versus systemic risk and the regulator's dilemma. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 108, n. 31, p. 12647–12652, 2011. Citado na página 16.

BLANCHARD, O.; DELLARICCIA, G.; MAURO, P. Rethinking macroeconomic policy. *Journal of Money, Credit and Banking*, Wiley Online Library, v. 42, n. s1, p. 199–215, 2010. Citado na página 15.

BORGATTI, S. P.; EVERETT, M. G. Models of core/periphery structures. *Social networks*, Elsevier, v. 21, n. 4, p. 375–395, 2000. Citado na página 50.

BORRILL, P. L.; TESFATSION, L. 11 agent-based modeling: the right mathematics for the social sciences? *The Elgar companion to recent economic methodology*, Edward Elgar Publishing, v. 228, 2011. Citado na página 19.

CAJUEIRO, D. O. Agent preferences and the topology of networks. *Physical Review E*, APS, v. 72, n. 4, p. 047104, 2005. Citado na página 57.

CAJUEIRO, D. O.; TABAK, B. M. The role of banks in the brazilian interbank market: Does bank type matter? *Physica A: Statistical Mechanics and its Applications*, Elsevier, v. 387, n. 27, p. 6825–6836, 2008. Citado na página 16.

CAMERER, C. F. *Behavioral game theory: Experiments in strategic interaction*. [S.l.]: Princeton University Press, 2011. Citado na página 20.

CAMERER, C. F.; HO, T. H. Experience weighted attraction learning in normal form games. *Econometrica*, v. 67, p. 827–874, 1999. Citado 4 vezes nas páginas 20, 23, 24 e 26.

CETORELLI, N.; GOLDBERG, L. S. Global banks and international shock transmission: Evidence from the crisis. *IMF Economic Review*, 2011. ISSN 20414161. Citado 2 vezes nas páginas 16 e 28.

CHAKRABARTI, R. Just another day in the inter-bank foreign exchange market. *Journal of Financial Economics*, Elsevier, v. 56, n. 1, p. 29–64, 2000. Citado na página 16.

CORDIER, S.; PARESCHI, L.; TOSCANI, G. On a kinetic model for a simple market economy. *Journal of Statistical Physics*, Springer, v. 120, n. 1, p. 253–277, 2005. Citado na página 38.

David Robinson. *The Incredible Growth of Python*. 2017. Disponível em: <https://stackoverflow.blog/2017/09/06/incredible-growth-python>. Acesso em: 13 dez 2017. Citado na página 34.

DIAMOND, D. W.; DYBVIG, P. H. Bank runs, deposit insurance and liquidity. *The Journal of Political Economy*, v. 91, n. 3, p. 401–419, 1983. Citado 3 vezes nas páginas 16, 23 e 29.

EPSTEIN, J. M. Agent-based computational models and generative social science. *Complexity*, Wiley Online Library, v. 4, n. 5, p. 41–60, 1999. Citado na página 19.

FUDENBERG, D.; LEVINE, D. K. *The theory of learning in games*. [S.l.]: MIT press, 1998. v. 2. Citado na página 24.

GAFFEO, E. et al. Adaptive microfoundations for emergent macroeconomics. *Eastern Economic Journal*, Springer, v. 34, n. 4, p. 441–463, 2008. Citado na página 19.

GALATI, G.; MOESSNER, R. Macroprudential policy–a literature review. *Journal of Economic Surveys*, Wiley Online Library, v. 27, n. 5, p. 846–878, 2013. Citado na página 15.

GEORG, C.-P. The effect of the interbank network structure on contagion and common shocks. *Journal of Banking & Finance*, Elsevier, v. 37, n. 7, p. 2216–2228, 2013. Citado na página 15.

GitHub Inc. *GitHub About Page*. 2017. Disponível em: <https://github.com/about>. Acesso em: 13 dez 2017. Citado na página 36.

GitHub Inc. *The State of the Octoverse 2017*. 2017. Disponível em: <https://octoverse.github.com>. Acesso em: 13 dez 2017. Citado na página 35.

GOODHART, C. A. E. The regulatory response to the financial crisis. *CESifo working paper*, n. 2257, 2008. Citado na página 15.

HALDANE, A. G.; MAY, R. M. Systemic risk in banking ecosystems. *Nature*, Nature Publishing Group, v. 469, n. 7330, p. 351, 2011. Citado na página 15.

LELYVELD, I. van et al. Finding the core: Network structure in interbank markets. *Journal of Banking & Finance*, Elsevier, v. 49, p. 27–40, 2014. Citado na página 50.

LEVINE, R. The governance of financial regulation: reform lessons from the recent crisis. *International Review of Finance*, Wiley Online Library, v. 12, n. 1, p. 39–56, 2012. Citado na página 15.

LIMA, J. I. A. de Vasconcellos e. *Um arcabouço computacional para estudo do setor bancário através de modelos baseados em agentes*. Dissertação (Mestrado) — Universidade de Brasília, 2014. Citado 2 vezes nas páginas 17 e 23.

LUCCHETTI, A. H. *Interbank network and regulation policies : an analysis through agent-based simulations with adaptive learning*. Dissertação (Mestrado) — Universidade de Brasília, 2016. Citado 5 vezes nas páginas 16, 17, 23, 26 e 27.

MARKOWITZ, H. Portfolio selection. *The journal of finance*, Wiley Online Library, v. 7, n. 1, p. 77–91, 1952. Citado na página 16.

MARKS, R. E. Analysis and synthesis: multi-agent systems in the social sciences. *The Knowledge Engineering Review*, Cambridge University Press, v. 27, n. 2, p. 123–136, 2012. Citado na página 20.

MASAD, D.; KAZIL, J. Mesa: an agent-based modeling framework. In: *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*. [S.l.: s.n.], 2015. p. 53–60. Citado 4 vezes nas páginas 37, 38, 39 e 43.

MONTAGNA, M.; KOK, C. Multi-layered interbank model for assessing systemic risk. 2016. Citado na página 16.

NIER, E. et al. Network models and financial stability. *Journal of Economic Dynamics and Control*, Elsevier, v. 31, n. 6, p. 2033–2060, 2007. Citado na página 16.

PAGE, S. E. On incentives and updating in agent based models. *Computational Economics*, Springer, v. 10, n. 1, p. 67–87, 1997. Citado na página 20.

POUGET, S. Adaptive traders and the design of financial markets. *The Journal of Finance*, Wiley Online Library, v. 62, n. 6, p. 2835–2863, 2007. Citado na página 21.

ROTH, A. E.; EREV, I. Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, v. 8, p. 164–212, 1995. Citado na página 24.

RUBINSTEIN, A. *Modeling bounded rationality*. [S.l.]: MIT press, 1998. Citado na página 19.

SILVA, T. C. et al. Financial networks, bank efficiency and risk-taking. *Journal of Financial Stability*, Elsevier, v. 25, p. 247–257, 2016. Citado 3 vezes nas páginas 16, 49 e 57.

SILVA, T. C.; SILVA, M. A. da; TABAK, B. M. Systemic risk in financial systems: A feedback approach. *Journal of Economic Behavior & Organization*, Elsevier, v. 144, p. 97–120, 2017. Citado na página 16.

SILVA, T. C.; ZHAO, L. *Machine learning in complex networks*. [S.l.]: Springer, 2016. v. 2016. Citado na página 16.

SOMMERVILLE, I. Software engineering. international computer science series. *ed: Addison Wesley*, 2004. Citado na página 17.

Stack Overflow. *Stack Overflow Trends*. 2017. Disponível em: <https://insights.stackoverflow.com/trends>. Acesso em: 13 dez 2017. Citado na página 34.

The Project Mesa Team. *Project Mesa - GitHub Page*. 2017. Disponível em: <https://github.com/projectmesa/mesa>. Acesso em: 15 dez 2017. Citado na página 37.

Apêndices

APÊNDICE A – Arquitetura e Diagramas Complementares

A arquitetura do *framework* será apresentada a seguir, através da descrição das principais classes e métodos utilizados na execução da simulação. É importante destacar que qualquer alteração do modelo pode ser realizada através de subclasses das classes apresentadas.

A.1 Modelo

Conforme explicado no Capítulo 4, a classe `Model` da biblioteca Mesa contém a definição dos métodos necessários para o controle básico da execução da simulação, sendo o ponto central da modelagem.

A classe `BankingModel`, por sua vez, é uma subclasse de `Model` e contém a implementação e definição específica do modelo a ser utilizado, explicado no Capítulo 2. A Figura 22 apresenta um diagrama de classes utilizando a UML para sintetizar a estrutura dessa classe.

Na inicialização[1] da classe `BankingModel`, são passados os parâmetros utilizados na definição do modelo, através do método `__init__`. Se for necessário fazer uma nova simulação com parâmetros diferentes, uma nova instância dessa classe deve ser inicializada.

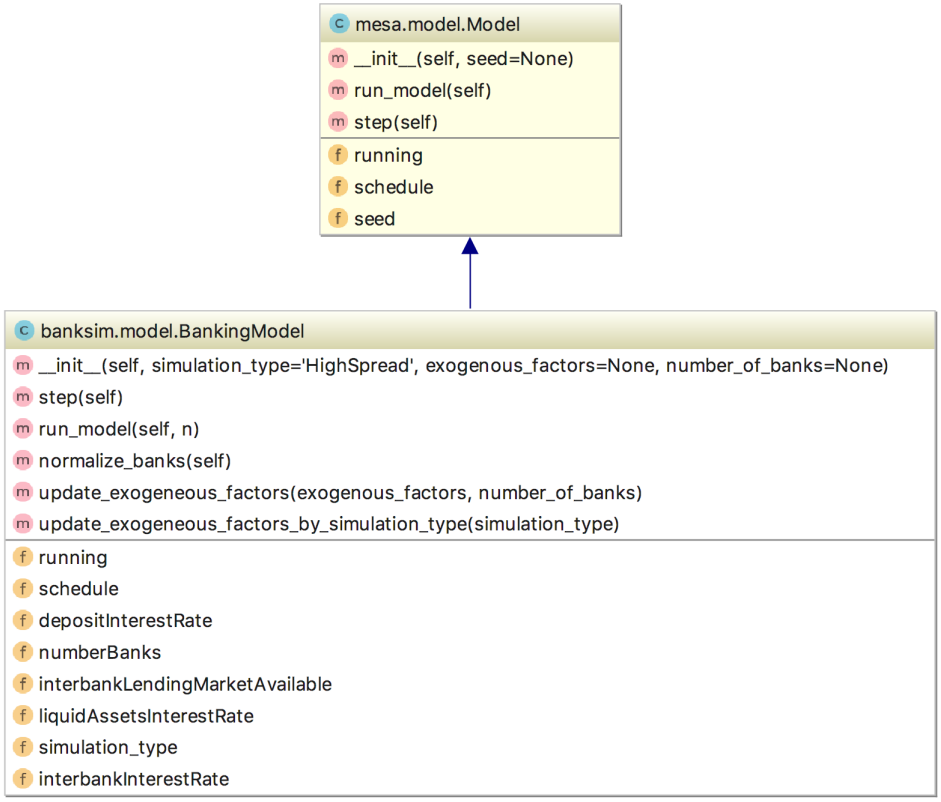
Essa classe também é responsável pela inicialização dos agentes do modelo, descritos na seção A.2. Todos os agentes são criados nesta etapa de inicialização, e a quantidade de cada agente é definida pelos parâmetros do modelo.

Além dos agentes, a classe `BankingModel` também inicializa o agendador, que é utilizado para definir a ordem de ativação dos agentes. O agendador básico do modelo está definido na classe `MultiStepActivation`, representada no diagrama da Figura 23.

Esse agendador garante que todos os agentes serão ativados sempre na mesma ordem, conforme definido na classe `MultiStepActivation`. Neste modelo, cada ciclo é formado por três períodos, representados no agendador pelos métodos `period_0`, `period_1` e `period_2`.

[1] No Python, a instanciação de uma classe já executa a inicialização do objeto instanciado através da chamada do método `__init__`.

Figura 22 – Diagrama UML da classe de controle do modelo do BankSim



Fonte: Elaborada pelo autor

A.2 Agentes

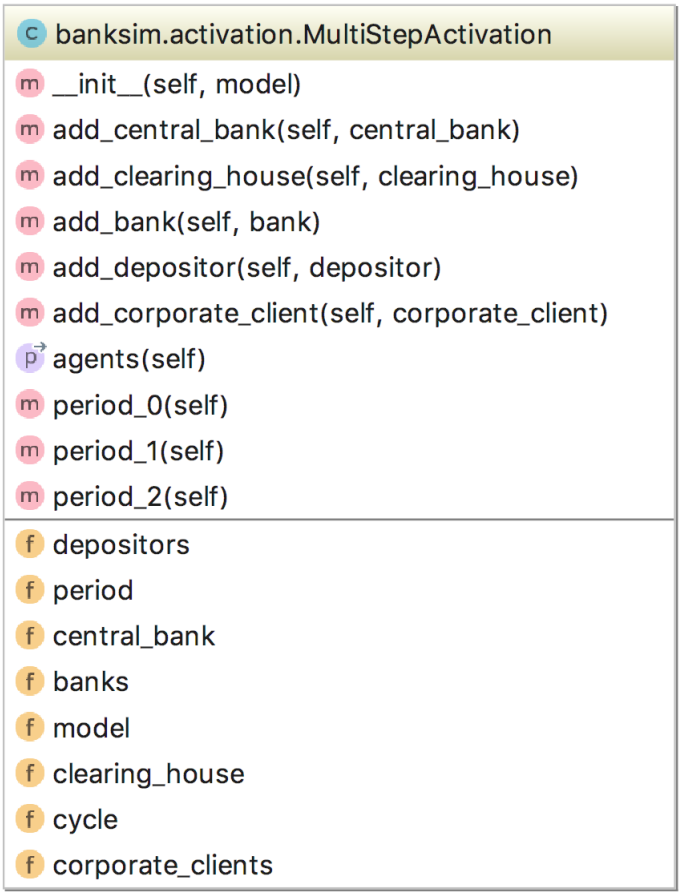
Os agentes do modelo são representados pelas seguintes classes:

1. A classe `CorporateClient` representa as firmas, com suas diferentes características de empréstimos, como a perda dada a inadimplência e taxas de juros utilizadas, por exemplo.

2. A classe `ClearingHouse` corresponde a câmara de compensação (*Clearing House*), responsável pela compensação de empréstimos e pela cobrança de garantias dos participantes. Sua presença no modelo é importante para estudar sua influência na redução do risco sistêmico.

3. A classe `Depositor` descreve os depositantes, que mantêm seus recursos nos bancos e decidem em qual momento realizarão os saques.

Figura 23 – Diagrama UML da classe de agendamento dos agentes



Fonte: Elaborada pelo autor

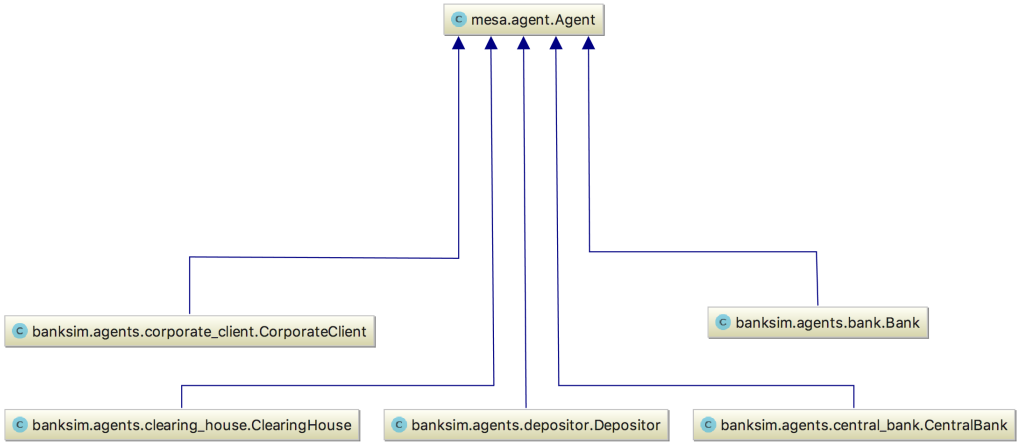
4. A classe `CentralBank` representa o Banco Central, objeto de estudo por sua atuação para manter a estabilidade do sistema financeiro. No modelo atual, não são tratadas as questões de política monetária.

5. A classe `Bank` corresponde aos bancos, agentes de interesse central no modelo, pois atuam como intermediários financeiros, emprestando os fundos dos depositantes para as firmas. Os bancos se conectam entre si no mercado interbancário e com o Banco Central, formando a rede endógena.

As classes descritas acima são sintetizados no diagrama ilustrado na Figura 24.

Na seção A.4, de forma complementar, são apresentados os diagramas de classe específicos de cada uma das classes enumeradas acima.

Figura 24 – Diagrama UML das classes dos agentes



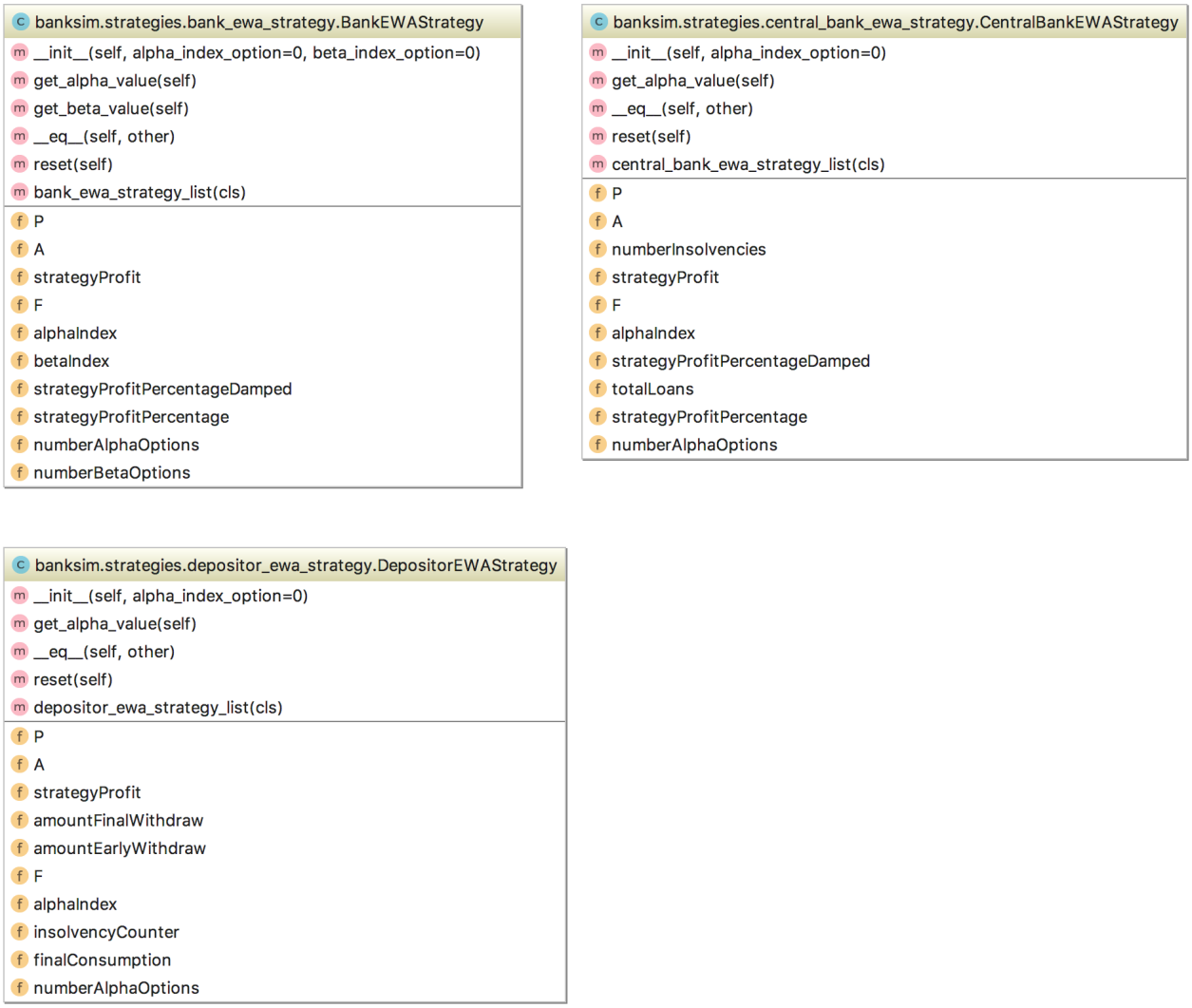
Fonte: Elaborada pelo autor

A.3 Aprendizado

As classes `BankEWAStrategy`, `CentralBankEWAStrategy` e `DepositorEWAStrategy` armazenam os dados utilizados pelos agentes para a escolha das estratégias, conforme o esquema de aprendizado do EWA. A Figura 25 ilustra, através de um diagrama de classes, os dados e métodos dessas classes.

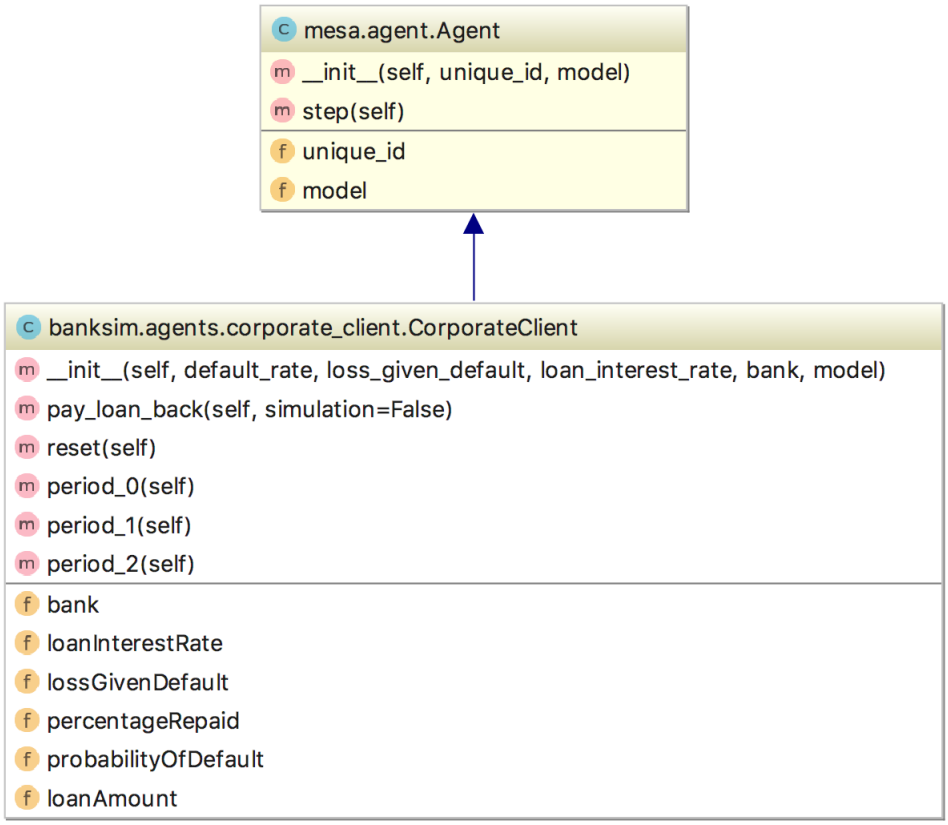
A.4 Diagramas Complementares

Figura 25 – Diagrama UML das classes de aprendizado dos agentes



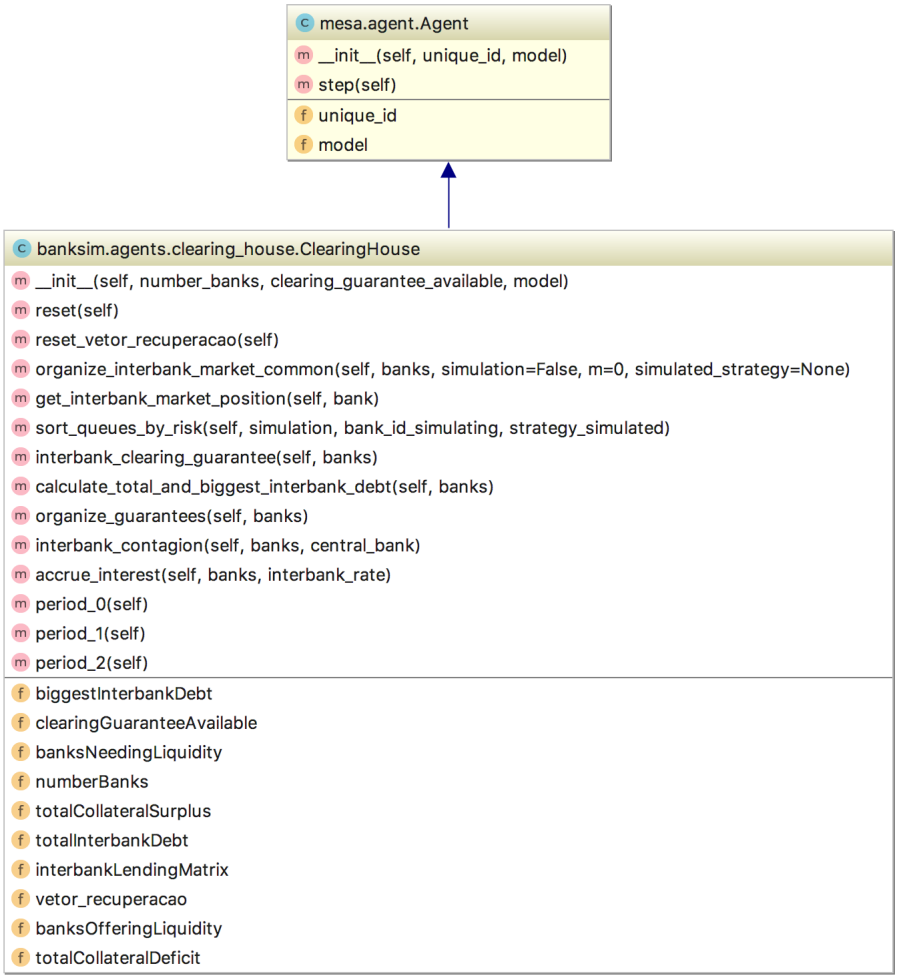
Fonte: Elaborada pelo autor

Figura 26 – Diagrama UML da classe `CorporateClient`



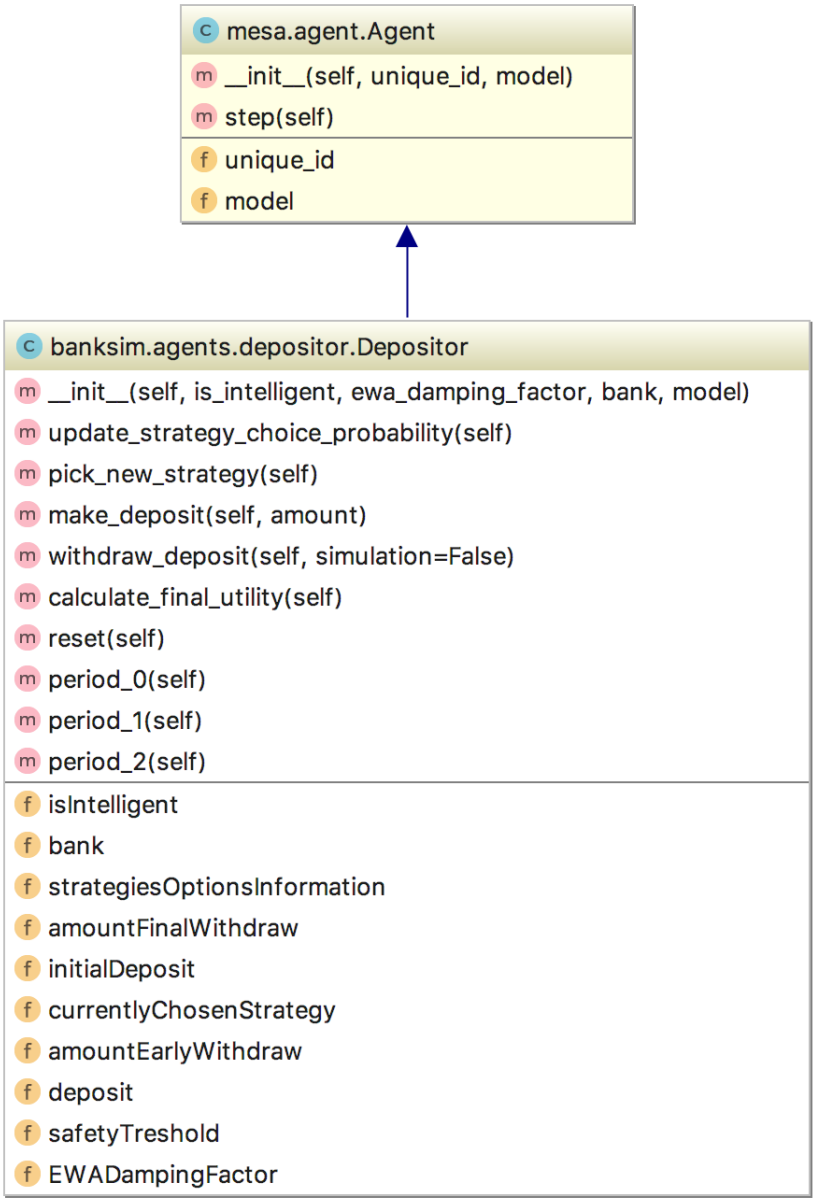
Fonte: Elaborada pelo autor

Figura 27 – Diagrama UML da classe `ClearingHouse`



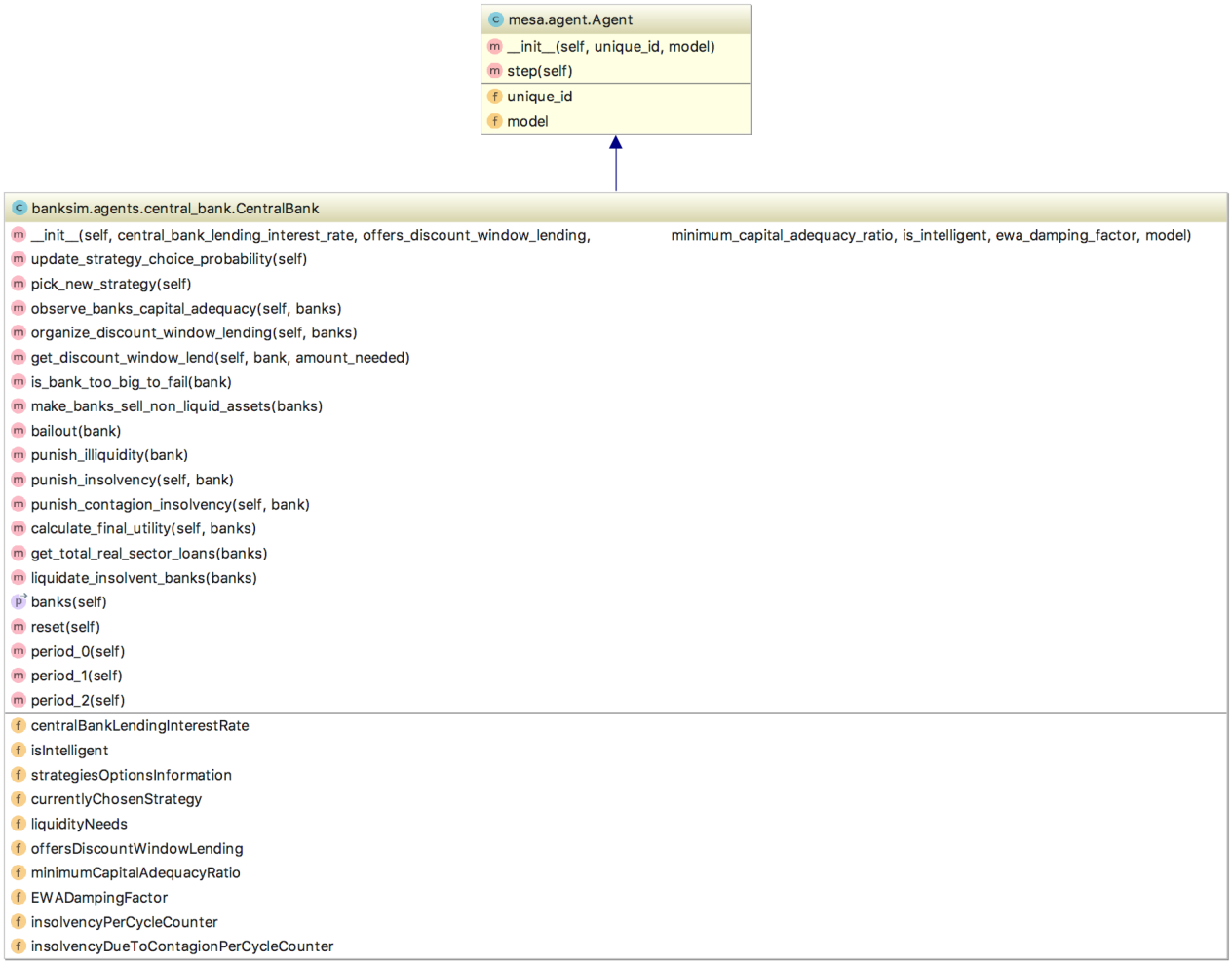
Fonte: Elaborada pelo autor

Figura 28 – Diagrama UML da classe `Depositor`



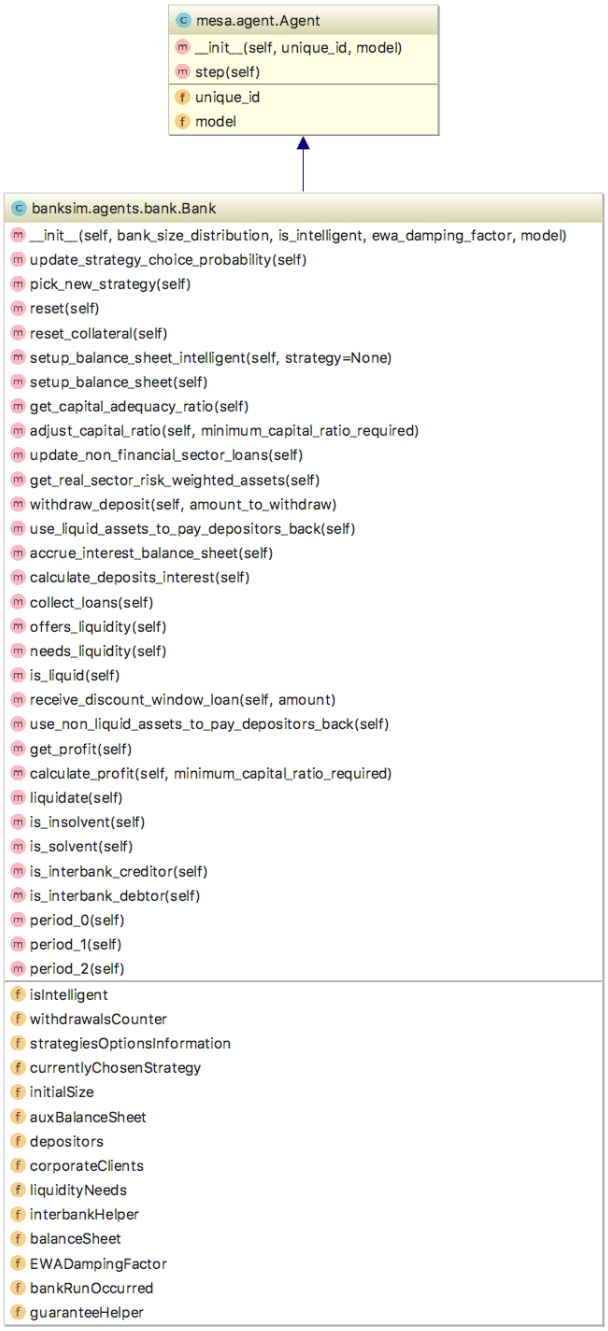
Fonte: Elaborada pelo autor

Figura 29 – Diagrama UML da classe `CentralBank`



Fonte: Elaborada pelo autor

Figura 30 – Diagrama UML da classe `Bank`



Fonte: Elaborada pelo autor