「取り付け騒ぎ」実験の技術的側面

齋藤哲哉[1)]
小林創[2)]
有馬守康[3)]
稲葉大[4)]

1．はじめに

本篇では本研究の中心をなす「取り付け騒ぎ」実験に関する先行研究となる，Garratt and Keister（2009）が米国で行った実験（Garatte-Keister 実験）と，その実験のベースモデルとなった Diamond-Dybvig モデル（Diamond and Dybvig, 1983）の理論を紹介し，我々が設計した実験の概要を示すとともに，その実験を実際に実施するにあたり，課題となった技術的な問題点と，その解決方法をまとめる．ただし，ここでは技術面を重視し，我々の実験の詳細については，重複を避けるため，他篇に譲ることとする．

この議論を進めるため，先ず 2 章で Diamond-Dybvig モデルの紹介を行い，それを念頭に 3 章で Garatte-Keister 実験と我々の実験の流れをまとめる．そして，4 章ではネットワーク関連の問題とその解決方法をまとめ，5 章で実際の実験プログラムのアーキテクチャを紹介し，6 章で技術面から見た実行結果を報告する．これらの議論は最後に 7 章でまとめる．

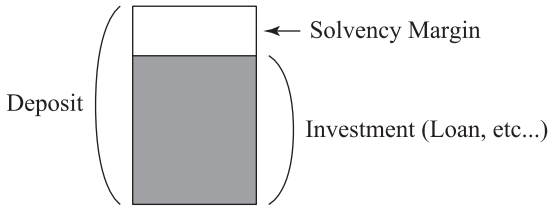
2．Diamond-Dybvig モデル

銀行は一般消費者と企業・投資家を含む，預金者から預金を集めて，それを貸付等で投資することで利息収入を得，その一部を預金者に戻すことで営業を行っている．そして，それら預金者は個々で投資することで得られるリターンや流動性などの要件を比較しながら，銀行預金を選ぶ．Diamond and Dybvig（1983）では，以下に概要を記す通り，そのような預金者の行動を考え，取り付け騒ぎが起きる可能性を理論的に明らかにしようとしている．

Diamond-Dybvig モデルでは，預金者はある一定期間の間，預金を銀行に預ける．また，概念的に預金者を一般消費者と投資家に分け，一般消費者は預金の満期まで緊急の流動性需要を生じさせないが，投資家は緊急の流動性需要を発生させる．そして，モデルの中では預金者は一つの集団としてしか表現されないが，緊急の流動性需要がその預金者の集団の中である一定の確率で発生するという形でモデル化されている．またさらに，預金者は短期的なスパンで預金している預金者と，より長期的なスパンで預金している預金者に分けられる．これら 2 つのタイプの預金者は，預金期間が始まる前に，確率的に振り分けられる．そして，実験には直接関係しないが，銀行は預金者を獲得するため，預金者の集団の期待利得を最大にするように，預金金利を決定する．

通常，図 1 に模式的に示したように，銀行は預金者から集めた預金を，融資などの方法を使って投資

図 1　銀行の支払い能力



するため，その預金の全額を現金として保有しているわけではない．銀行は，日常の流動性需要を見越し，それに応えられる現金を用意しているだけである．Diamond-Dybvig モデルでも同様の想定が置かれており，支払い余力（Solvency Margin）を超える流動性需要が生じる可能性がモデル化されている[5)]．そして，実際に支払い能力を超える流動性需要が発生した状態を「取り付け騒ぎ（Bank-Run）」と呼んでいる．

ではもう少し詳しく Diamond-Dybvig モデルにおける取り付け騒ぎ発生の原因を考えていく．このモデルでは，預金者が満期（maturity）まで預金を引き出さなければ，元本に利息が足された額が償還される．しかし，途中で引き出しを行った場合には，利息の支払いは行われず，銀行の倒産が起きないという前提では，元本のみの償還となる．この間，満期までには確率的に発生する流動性需要により，預金の引き出しが行われる．ただし，この確率的に発生する流動性需要が直接的に銀行の倒産を起こすことはない．また，満期は預金者のタイプによって異なっている．このモデルの時間軸は，単純化のために預金者を 2 タイプ，元本を d，利息を r_t（満期に関する預金者のタイプにより $t = 1, 2, \cdots$とする）として図 2 に模式的に表した通りである．この時，それぞれの満期まで預金を引き出さずに待っていた場合，満期には $d + r_t$を得ることができる．因みに，各預金者は図中の黒丸で示した時点でのみ銀行の状態を確認することができるが，その他の時点ではその状態を確認することができないため，個々の予測に頼ることとなる．

上記のような時間の流れを想定して，なぜ銀行の倒産が起きるのかを考える．銀行の倒産（取り付け騒ぎ）が発生した場合，その銀行の預金総額を D，支払い余力を $M \leq D$，預金引き出し要求数を n として，取り付け騒ぎに参加した預金者は $M/n \leq d$ を得るが，参加しなかった預金者の預金はゼロとなる．

実際の取り付け騒ぎは銀行や金融システムの不健全な運営も絡んで発生するが，いまは銀行が健全に運営されていたと仮定する．この時，常態として流動性需要がある確率分布に従って発生し，同時に，短期スパンの預金者による満期引き出しも行われる．この状態での取り付け騒ぎは，確率的に発生している流動性需要と短期スパンの預金者の存在が銀行を倒産させるかもしれないと，多くの預金者が予測した場合に発生する．したがって，このモデルにおいて，取り付け騒ぎが発生する原因の一つ考えられるのは，流動性需要の発生と短期スパンの預金者の存在である．

このモデルでは，引き出しや満期によって失われる，銀行の支払い余力の回復は行われないため，時間軸が図 2 に従って進んでいる場合，一回の流動性需要と短期スパンの預金者の満期の重なりだけではなく，これが何回か生じた場合の銀行の支払い余力の弱体化も影響する．同様の議論はサブ・プライム危機のメカニズムの解明を試みた Gertler and Kiyotaki（2015）でも述べられており，短期スパンの債権者（預金者）が増加するに従って，モーゲージ専門会社の資金調達が困難（取り付け騒ぎ）になり，それがその融資元の金融機関にも波及したと考えられている．

では，図 3 のように，ゲームの構造をステージゲームに分けることでもう少し分析を単純化して，さ

図２　モデルの時間軸

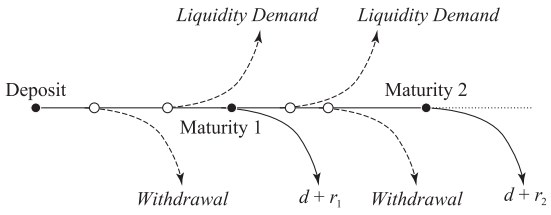
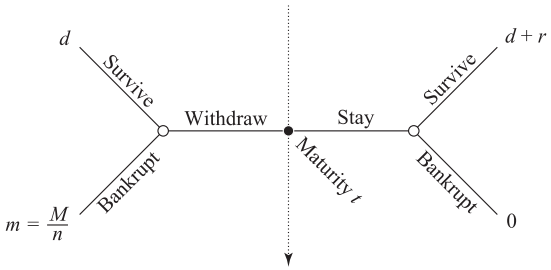


図３　ステージ・ゲームの木



らに重要な点を明らかにする．この図では，ステージゲームに注目しており，混同が生じないため，満期を表す添字 t を省いている．ここで p を預金者が主観的に予測する取り付け騒ぎが起こる確率とすると，この預金者が自分の預金を引き出さずに満期まで持っているための条件（無差別な状態を除く）は，次式のようになる．

$$(1-p)(d+r) > pm + (1-p)d \tag{1}$$

この条件を整理すると，以下のように単純化される．

$$p < \frac{r}{m+r} \tag{2}$$

そして，条件（2）を r と m に関してそれぞれ図示すると，図 4 および図 5 のようになる．ここでは一先ず，議論の単純化のため，預金者数は十分に大きいと仮定し，個々の行動は p に影響を与えないものとする．

まず，所与の m について図 4 に注目すると，与えられた p の下では r が十分に大きければ引き出さないことがわかる．また，そのような r の水準は，p に対して比例して大きくなることがわかる．そして，所与の r について図 5 を見ると，m が十分に大きければ引き出さないことがわかる．そのような m の水準は，p に対して反比例する．もし，預金者が同質的であったとすると，引き出しを行わせないような r と m の水準を選ぶことにより，確率的に発生する流動性需要がそれだけでは銀行を倒産に追い込まないという前提を考えると，$p = 0$，すなわち，預金者は満期まで預金を解約せず，その結果，銀行は倒産しないようになるという均衡を導くことができる．ここで m が銀行の支払い余力に比例するパラメーターであることに注意すると，これらの観測から以下の二つのことが言える．

ⅰ．取り付け騒ぎの確率が高い場合は，十分に利息を大きくすることで，それを防ぐことができる．

ⅱ．取り付け騒ぎの確率が高い場合でも，支払い余力が充分に大きければ，それを防ぐことができる．

図４　引き出しをしない条件１

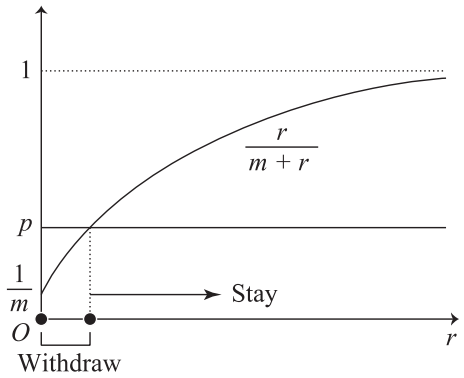
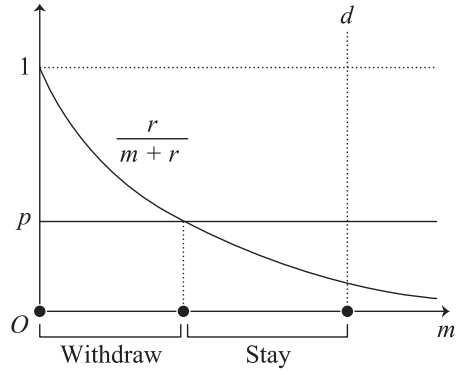


図５　引き出しをしない条件２



上記のような分析から，取り付け騒ぎを防ぐ条件を提示することができるが，ここである重要なことに気づく必要がある．もし $p = 0$（銀行は倒産しない）であるならば，預金者はそれぞれの満期まで引き出しを行わないことが合理的な行動となる．つまり，預金者が本当に合理的ならば，取り付け騒ぎは起こらないはずである．ではなぜ，現実には歴史上いくつもの取り付け騒ぎが起こるのだろう？ Diamond-Dybvig モデルにおいて，これは預金者間のコーディネーション・エラーとして解釈されている．理論では完全に説明できない部分を Garratt-Keister 実験（Garratt and Keister, 2009）と，我々の実験では検証しようとしている．次章でそれらの実験の概要を紹介する．

3．Garratt-Keister 実験と本実験の概要

Garratt and Keister（2009）では二種類の実験が行われた．一つは預金引出のチャンスは 1 回しかないもの，もう一つは預金引出のチャンスが複数回あるものである．我々の実験では，預金引出のチャンスが 1 回しかない実験のみを検証するが，これは預金引出チャンスが複数回あっても，実験自体が同質的と考えられるためである．以下では先ず Garratt-Keister 実験の流れをまとめ，我々の実験と合わせて，プログラム設計への導入とする．ただし，重複を避けるため，実際の実験の詳細とその結果については他篇に譲ることとする．

実験の時間軸はステージ・ゲームで構成される．ステージ・ゲームの利得表やステージ数はそれぞれの実験によって決定されるが，流れとしては以下のようなものとなる．

最初のステップとして，各ステージの開始時に被験者は小さなグループにランダムに分けられる．この時，どの被験者がどのグループに所属しているのかは，各個人にしか分からないようにする．従って，各被験者は自分がどのグループに属しているのかは知っているが，そのグループに誰が属しているのかは知らないという状態になる．ステージがスタートすると，被験者は利得表を見せられて各自の行動を決めるという段階に入る．行動を決めるために与えられている時間は，全ての被験者に一様に決められている．（例えば 30 秒）因みに，「何もしない」というのも「行動」に含むが，これは与えられた時間内に何もしない場合が「それ」に当てはまる．そして，ステージ・ゲームの最後に，そのステージの結果が表示され，被験者の習熟とそうかの確認を行う練習ステージを除き，同時に得点が積算されることになる．

ステージ・ゲームは確率的な流動性需要の発生（強制引出と呼ぶ）がある場合とない場合，さらに，銀行の支払い余力が大きい場合と小さい場合に分けられており，合計で 4 通りの組み合わせがある．また，Garratt-Keister 実験のユーザー・インターフェイス（UI）では預金を引き出すボタンしかなかったが，我々の実験では，預金を引き出すボタンのみの UI と，預金を「引き出す」・「引き出さない」という二つの行動を示すボタンを配置した UI の二通りの操作画面を使った実験を実施する．従って，合計で 8 通りのステージ・ゲームが出来上がる[6]．これに合わせて，実際に得点が積算されない練習ステージ（被験者のプログラムお操作に対する確認を行う）も必要になるため，合計で 16 通りに対応出来るプログラムを準備する必要がある．

実験中の切り替えは，支払い余力が小さい場合の利得表（Type A）と大きい場合の利得表（Type B）を途中で切り替える．そして，それぞれの利得表を用いたステージの流れの中で，練習ステージを最初に行い，次に強制的な引き出しを行わない場合のステージ，それに続いて，強制的な引き出しを行うステージという流れの実験を繰り返すことになる．ただし，UI の変更は実験中に行う実験は行っていない．

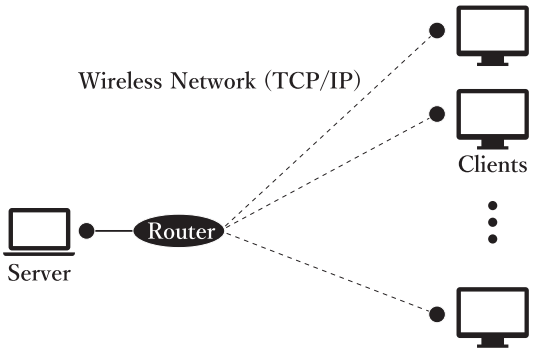
実際の実験プログラムはこの他に，ステージ・ゲームに先立って被験者の理解度を試すテストを実施するプログラムが必要となる．これは，被験者に設問に答えてもらい，その答えをサーバーに送信して採点し，満点であれば待機画面に遷移，そうでなければ再度，理解度を試すテストの画面に戻るという処理をしなければならない．さらに，実験終了後にはそれぞれの被験者が獲得したポイントを集計し，それに応じた報酬の支払いに対応しなければならない．次章からは経済学からは少し離れて，これら一連の実験の流れとステージ・ゲームに対応したプログラムを作成するための技術的な議論を行う．

4．技術的問題と解決方法

本実験では，最大 24 名の被験者（クライアント）が，ローカル・ネットワークを通じてサーバーと通信を行う．この通信は，各セッションの開始と終了のシグナルの受け取りや各被験者の行動の送信，各被験者の各セッションの結果の表示を行う．このため，サーバー側で出した命令をクライアントが受け取るまでの遅延（ディレイ）が大きくなると，実験に大きな支障を生じさせることになる．実際には，各セッションを 30 秒で設定するため，可能であればこのディレイを実効で 1 秒以下にすることが目標になる．

24 台のクライアント・マシンが，1 秒以内にサーバー側のシグナルを受け取ろうとした場合，クライアント側は，例えば 0.2 秒に一回の問い合わせを行う必要があったとする．この時，ネットワークを通過するピンガーの数は往復で 48 個となる．この通信一つ一つは大半が 10 バイト以下であり，これは昨今のウェブ・サービスの通信料としては大きくないが，大学のローカル・ネットワークで接続された 1

図 6　実験に用いたネットワーク



台のサーバー・マシンに，実験実施時間中（約 2 時間）継続的に送られた場合，高確率で DoS 攻撃（Denial of Service Attack）の疑い，もしくは，ハッキング目的の侵入と認識され，セキュリティの確保のため，その通信が自動的に遮断されてしまう．実際に，2016 年度に行った実験準備時に 10 台のみを接続する試運転の段階でこの状態が生じた．また，解決を試みる段階で，サーバーを外部のレンタル・サーバー（AWS 上に設置した仮想マシン）に置いて，10 台のクライアント・マシンにアクセスさせるという方法も試みたが，数分後には必ずそのアクセスが遮断されるという結果になった．これは，学内のコンピューターを外部サーバー侵入時の踏み台にさせない措置と考えられる[7]．

このようなネットワーク接続の問題を解決するために，本実験に関する全ての通信が閉鎖されたネットワーク内で行われるように[8]，ネットワークを構築した．具体的には，UNIX 規格である OSX をインストールした MacBook Pro をサーバーとして，ワイアレス対応のルーターにサーバーをストレート・ケーブルで接続し，ルーターはワイアレス対応のアンテナ（USB 接続）を持った各クライアント（Windows マシン）とワイアレスで接続して TCP/IP 通信を行わせるようにして，ネットワーク接続の問題を解消した[9]．（図 6 参照）

次に，ネットワーク接続の問題が解決したとして，ディレイを生じさせる原因としては，（1）ネットワーク速度そのものとハードウェアの処理能力と（2）使用するプログラミング言語とソフトウェアの 2 点が挙げられる．これらを完全にコントロールするのは不可能であるが，一定の制約条件の下での善処は可能である．

（1）ネットワーク速度そのものとハードウェアの処理能力

ネットワーク速度のボトルネックとして，アンテナとルーターの性能が挙げられる．アンテナに関しては，本実験を開始した 2016 年当時で通信速度の上限がなるべく大きく（300Mbps），アンテナの方向が変えられるものの中から予算に見合うものを選択した．そして，ルーターは関西大学経済実験センターに備え付けのもの（最大接続台数 50 台）のものを用いたが，アンテナが方向指向性ではなかったため，2 台使用した．また，ルーターとサーバーを接続する LAN ケーブルにはギガビット・イーサーネット対応のものを使用した．

次にサーバーとクライアントであるが，後述の通りクライアント側は関西大学経済実験センターに備え付けのものを使用するため，その負担が多くない仕様[10]でプログラミングを行うが，その反動としてサーバー側の処理は大きくなる．このことを鑑み，2011 年式と型は古いが，サーバー・マシンにはコア毎のクロック数は若干低めではあるが，処理の一つ一つは重いものではなく，問題になるのは同時に到

着する要求処理の多さであるため，並列処理に強いクァッド・コア CPU を持つ MacBook Pro を用いた．

(2) 使用するプログラミング言語とソフトウェア

プログラミング言語は，簡便性と処理速度を勘案して選択する必要がある．C 言語の処理速度は最も早いが，現場でプログラムの修正が必要になった場合，コンパイル作業なども含めると，処理が複雑になってプログラムが巨大にならないならば，その効用は小さくなる．またデータベースと連携するということを考えると，プログラムが大きくないうちは PHP が適当であると考えた．

次に使用するソフトウェアであるが，後述するように，実験プログラムをウェブ・アプリケーションとして開発するためのウェブ・サーバーと SQL サーバーが必要になる．ウェブ・サーバーは PHP との連携を考え，Apache 2 を用いた．また，セキュリティーの問題や検索速度の問題はネットワークが完全に閉鎖された状態であることと，データ量は非常に多くなるが，大きなデータからの検索は実験中に行われないことを考え，SQLite 3 を用いた[11]．さらに，このように Apache + PHP + SQLite という組み合わせは無償で，且つ，非常に普及しており，実験プログラム開発時の問題解決の際，非常に多くのリソースが存在し，例えば，ウェブ・サーバーに設定次第では Apache よりも高速に動作すると言われる Nginx などを用いたりするのに比べ，効率的な開発ができる利点がある．

5. 設計方針とプログラミング

実験プログラムはディレイを防ぐため，最小限の通信情報量で動作するように設計した．また，サーバー側でクライアント側のステージの遷移や意思決定時間などを一元管理させ，クライアント間の時間的差異を生まないために，クライアント側には極力負担をかけないような設計を行った．実際のプログラムの仕様概要は以下の通りである．

(1) サーバー側プログラム

サーバーではウェブ・サーバーとデータベース・サーバーを稼働させて，クライアントを管理する処理を行う．後述するが，ウェブ・サーバーではクライアント側からの信号を受け取り，それに応答する形でサーバー側からの指示を返信する作業を担当させる．そして，クライアント側から送られてくる被験者の行動に関する全ての情報は，分析に実際に使用するか否かを別として，その全てをデータベースに記録するようにした．さらに，クライアントの信号を受け取ることを利用して，サーバー側でクライアントとの通信の状態を逐次モニタリングできるようにし，それを 1 秒間に数回の間隔でアップデートして UI に表示し，実験中にディレイが発生していないか，また，接続が切断されていないかを確認できるようにした．この UI 実装には，JavaScript（miniAjax）を用いた．

実験の開始から終了や各ステージ間の遷移，利得表の変更や確率的な流動性需要の有無など，サーバー側からの指示は，管理者 ID でログインすることにより表示される管理画面で，ほぼ全ての操作を行えるように設計した．ただし，被験者数や各グループの大きさなど，実験中に操作することのない一部の設定に関しては設定ファイル（init.inc）を用意し，実験開始前にそれを編集することで，実験環境を設定・変更できるようにした．

(2) クライアント側プログラム

クライアント側では，最初に個々の被験者を識別する必要がある．この識別作業は，各クライアント・

マシンに ID を降り，その ID をプルダウン・メニューから選び，あらかじめランダムに付与した 4 桁の暗証番号を入力してログインすることで，実装している．この認証はサーバー側で行う．また，受付時にどの被験者にどの ID を割り振ったかを記帳しておき，支払いなどその後の事務処理における照合を可能な様にした．

ログイン後，被験者の理解度を確認するテストの画面に移行させ，テストの採点を行い，実験に入る流れになるが，これらの画面遷移のほぼ全てをサーバー側からの指示を受け取ることで実現する．そのために，1 秒間に数回の間隔でサーバー側に自動的に問い合わせを行い，信号をその都度受け取ることになるが，これは JavaScript（miniAjax）を用いて，信号受信のための画面遷移はさせない形で実装した．

プログラミングの多くは信号の送受信と被験者の行動の記録であるので，ソースコードを参照していただければ特に問題ないと思われるが，被験者の各グループへのランダムな割り当て，流動性需要の確率的発生に関するプログラムの実装は，ここで紹介しておく必要があると思われる．以下，順を追ってその方法をまとめる．

（3）被験者の各グループへの割り当て

各ステージ開始時に，被験者は毎回各グループ（銀行と呼ぶ）に割り当てられる．これを実装するために，被験者の ID（UID）が入った配列を考え，これをシャッフルして，グループのサイズに切り分けていくイメージのプログラムにしている．このプログラムの模式図は図 7 に示したとおりである．

図 7 の例では，スペースの関係で各グループのサイズを 3 にしてあるが，実際の実験では，そのサイズは 5 となる．この図が示す通り，UID が格納された配列をシャッフルすることで，UID の順番はランダムに入れ替えられる．そして，ランダムな入れ替えを行った後の配列をグループ数で区切るようにして，for ループを使って配列の格納番号（Array ID）を利用して取り出し，グループを割り当てるという手順となる[12]．その結果，各グループに配置される被験者はステージ毎にランダムとなり，意図した実験環境を実装できるようになる．そして被験者はその結果を受け取り，ステージ・ゲームをプレーすることになる．

また後半の実験では，このプログラムのパフォーマンスを上げるため，もう少し改良を加えたものを用いた．改良プログラムでは，ASCII コードを利用して 5 つずつアルファベット（銀行の ID）を配列に格納し，その配列をシャッフルするという方法を用いた．

（4）流動性需要の確率的発生

流動性需要の発生は，Garratt-Keister 実験の確率過程に従って発生させるが，この確率過程を実装するためには擬似乱数を用いた処理を行った．この処理は，表 1 と図 8 に記した通りの過程と確率分布に

図 7　グループ割り当てのイメージ

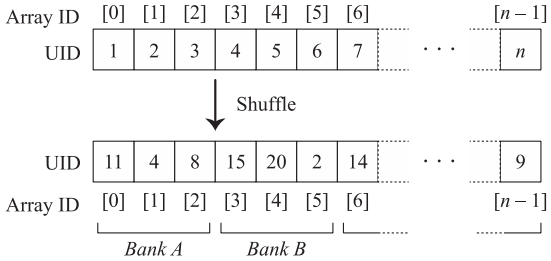


図８　流動性需要の発生（強制預金引出）

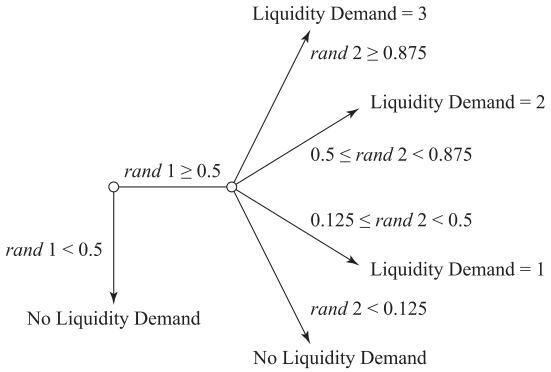


表１　流動性需要の発生確率

流動性需要	0	1	2	3	合計
確率	0.125	0.375	0.375	0.125	1.000

よって決める．図 8 の中で，rand 1 及び rand 2 と記されているのは，それぞれ，流動性需要の抽選を行うか否かの判定のための乱数と，流動性需要の数を決定するために発生させた乱数の値である．

上記の過程に従うと，ステージの最初に流動性需要の抽選を行うか否かを，グループ毎に 50％の確率で決め，もしこの抽選で流動性需要の発生に関する抽選を行うことになった場合，表 1 の確率分布に従って，強制的に預金を引き出すことになる人数（0〜3 名）を確率的に決める．そして，流動性需要の数が決められたら，次はそのグループに配置された被験者の中からランダムに強制的に引き出しを行わせる UID を配列のシャッフルを利用して選ぶ[13]．これに選ばれた場合，その被験者の画面には，行動を選ぶフェイズが終了した後にその旨が表示されることになっており，そのステージでの意思決定は無視されて，預金が引き出される．

ちなみに，第一段階目の抽選を省いて流動性需要の判定を行う方法も当然考えられるが，この場合，0.5625 の確率で強制的な引き出しは行われず，0.4375 の確率で 1〜3 名に対して強制的な引き出しを行わせるように乱数を一回だけ評価することになる．しかし，この部分は今後の拡張性と先行研究との整合性も考え，Garratt-Keister 実験の通りの方法を踏襲することにした．

6．実行結果（技術面）

この実験プログラムの実行に関わるネットワーク上の問題については先に述べたとおりである．ここでは，ネットワークの問題が解消されているものとして，実際にこのプログラムがラボの中で実験中にどのように動いたのかをまとめる．実験は表 2 の通りに本研究課題実施期間中の 2016 年 8 月と 2017 年 2 月に合計 4 セッションを，関西大学経済実験センターにて，Garratt-Keister 実験の再現とその UI を改善した実験を行った．他の実験の被験者数は 25 であるのに対し，2017 年 7 月 13 日午後の被験者数が他よりも少ない．これは，被験者の集まりの状態が良くなかったためである．また，全ての実験を通して，グループの大きさは 4〜5 となっている[14]．

まず，クライアント側の処理であるが，おおむね良好であったとはいえ，実験中に処理が集中するタ

表２　実験実施状況

日付	被験者数	グループ数	ユーザー・インターフェイス
2016 年 8 月 2 日	25	5	「引き出す」のみ
2017 年 2 月 21 日	25	5	「引き出す」のみ
2017 年 2 月 22 日（午前）	25	5	「引き出す」・「引き出さない」
2017 年 2 月 22 日（午後）	20	4	「引き出す」・「引き出さない」
2017 年 7 月 5 日	20	4	「引き出す」のみ
2017 年 7 月 6 日	20	4	「引き出す」のみ
2017 年 7 月 12 日	25	5	「引き出す」・「引き出さない」
2017 年 7 月 13 日	10	2	「引き出す」・「引き出さない」

イミングで，何度か遅延が発生した．しかし，ステージ中での問題は全期間を通して 1 件のみであり，その他は良好であった．その 1 件の問題は，ステージが開始されたのに画面が遷移せず，ステージが終わってしまったというものである．この遅延はネットワークやサーバー側が原因ではなく，クライアント側に問題があったと思われるため，再読み込みを行い，その後は問題が発生することはなかった．また，これに対して被験者がどう行動しようとしていたのかを聞き取り，引き出さないつもりだったことを確認した．何も行動をしなければ引き出しをしないという行動になるように設計されていたため，データは修正せず，そのままとした．

ステージ開始前後にクライアント側との接続が途絶えることが数回発生したが，これはワイヤレス・ルーターの電波探査時のアンテナの方向，もしくは，ルーターへのアクセスの集中が原因と思われる．しかし，実験プログラムは正常に作動していたため，データ取得に問題は生じなかった．

他の問題としては，ウェブ・ブラウザーの問題と思われるが，しばらく操作をしない時間が続いた場合に実験セッションが維持されず，再度ログインしなければならないことが数回あった．しかし，ステージ中にこの現象は生じなかったため，データ記録上の問題は発生しなかった．この現象は全てのクライアント・マシンで発生したわけではなく，ほぼ全てのクライアントは問題を起こさなかった．

7．まとめ

本編では，Diamond-Dybvig モデルを紹介し，それを基にした Garratt-Keister 実験をベースにした我々の実験の概要をまとめ，それに基づいた実験環境を整えた．

本実験の性格上，実験施設である関西大学経済実験センターにインストールされてある Z-tree などのプログラムを使うのではなく，フルスクラッチでウェブ・アプリとして書かれた実験プログラムをウェブ・サーバー（Apache 2）を運用して使用することとしていたが，その実験プログラムの実行環境に加えて，そのアプリケーション開発で直面した問題，例えば，ランダムなグループの割り当てや流動性需要の抽選などのプログラミングの問題を，実際のラボにおける何度かの試運転を経て解消することができた．そして，それらの問題をクリアすることで実際に実験を行い，データを得ることができた．他編ではその実行結果の分析を行うとともに，実験に基づいた理論モデルの再構築と分析を行い，その含意を考える．

注

1） 日本大学経済学部准教授
2） 関西大学経済学部教授
3） 日本大学経済学部専任講師
4） 関西大学経済学部教授
5） 厳密には，現代の銀行システムは通常，個々の銀行が中央銀行に準備預金を持ち，さらに中央銀行は最後の貸し手となるべく待機し，急な流動性需要に耐えられるように支払い能力を金融システム全体として担保している．
6） 追加的な UI は，クリック音が情報伝達チャンネルになっている可能性を考慮したものである．詳細は他篇に記すこととする．
7） ネットワーク・セキュリティに関する本件のような事項の多くは機密事項であるため，担当者に対しての詳細確認は行っていない．
8） 外部との接続は全て遮断した状態にした．
9） 本実験のネットワーク環境構築は，本実験の実施場所である関西大学経済実験センターの管理者と協議の上で行った．
10）後で詳述する通り，Ajax を用いてウェブ・ブラウザーのみで操作できるように設計し，クライアント側はサーバーへの問い合わせと結果等の描画のみを行わせるようにしている．
11）SQLite はデーモンを起動してポートを監視しているわけではないため，厳密には SQL サーバーではない．また，最も普及している MySQL や PostgreSQL と比べ，データが多くなるに伴って，検索速度が急速に落ちることが知られている．この問題はインデックスを適切に定義することで大幅に改善することも知られている．
12）配列の格納番号は使用する言語の仕様により 0 からスタートするので，サイズが n の配列の場合，その最後尾の格納番号は $n - 1$ となる．
13）シャッフルした配列の先頭から，配列の格納番号を利用して，必要な人数を抽出して割り当てる．
14）上記の統制実験の他に，準備段階では設備の確認やネットワークの状況，実験プログラムの動作を確認する作業を，関西大学経済実験センターと日本大学経済学部にて，何度か行っている．

参照文献

Diamond, Douglas W. and Philip H. Dybvig（1983）. Bank Runs, Deposit Insurance, and Liquidity. *Journal of Political Economy*, 91（5）, 401–19.

Garratt, Rod and Todd Keister （2009）. Bank Runs as Coordination Failures: An Experimental Study. *Journal of Economic Behavior & Organization*, 71（2）, 300-17.

Gertler, Mark and Nobuhiro Kiyotaki （2015）. Banking, Liquidity, and Bank Runs in an Infinite Horizon Economy. *American Economic Review*, 105（7）, 2011-43.

Kehoe, Timothy J., Nobuhiro Kiyotaki, and Randall Wright （1993）. More on Money as a Medium of Exchange, *Economic Theory*, 3（2）, 297-314.

Kiyotaki, Nobuhiro and Randall Wright （1989）. On Money as a Medium of Exchange, *Journal of Political Economy*, 97（4）, 927-54.

Kiyotaki, Nobuhiro and Randall Wright （1990）. Search for a Theory of Money, *NBER Working Paper*, No. 3482.

Kiyotaki, Nobuhiro and Randall Wright （1991）. A Contribution to the Pure Theory of Money, *Journal of Economic Theory*, 53（2）, 215-35.

Kiyotaki, Nobuhiro and Randall Wright （1993）. A Search-Theoretic Approach to Monetary Economics, *American Economic Review*, 83（1）, 63-77.

Lagos, Ricardo and Randall Wright （2005）. A Unified Framework for Monetary Theory and Policy Analysis, *Journal of Political Economy*, 113（3）, 463-84.

Nakamoto, Satoshi （2008）. Bitcoin: A Peer-to-Peer Electronic Cash System. *Mimeo*.（Available at https://bitcoin.org/bitcoin.pdf）

Saito, Tetsuya （2018）. When Does Bitcoin Rise? *Mimeo.*
（初稿は東京大学金融教育研究センター・日本銀行決済機構局共催コンファランス『フィンテックと貨幣の将来像』にて報告
https://www.boj.or.jp/announcements/release_2016/rel161201a.htm/）

Trejos, Alberto and Randall Wright（1995）. Search, Bargaining, Money, and Prices, *Journal of Political Economy*, 103（1）, 118-41.