# Appendix C: Computational Methods

This appendix provides detailed documentation of the numerical methods used to solve the model's equilibrium. A complete, documented implementation in Julia is available in the online supplementary materials.

## 0.1 Mathematical Framework

### 0.1.1 Problem Statement and Computational Challenge

The goal is to solve the equilibrium system formally defined in Proposition 1. The primary computational challenge arises from a fixed-point problem: the optimal withdrawal times for agents depend on the perceived hazard rate $h(\tau)$, which in turn depends on the equilibrium bank collapse time, $\xi^*$. Simultaneously, the collapse time $\xi^*$ is determined by the aggregate withdrawals, which are a function of those same optimal withdrawal times. This interdependency makes a direct solver approach complex, as it would require iterating on the entire hazard rate function.

### 0.1.2 The Reversed Time Approach

Our solution hinges on a change of variables to "reversed time," a concept formalized in Lemma 2 of the proof appendix. Reversed time ($\bar{\tau} = \xi^* - \tau$) represents the time remaining until a potential collapse. This transformation is the cornerstone of our computational strategy, as it **decouples the fixed-point problem into a sequential one**. The key observation from Lemma 2 is that the hazard rate in reversed time, $\bar{h}(\bar{\tau})$, is completely independent of the crash time $\xi^*$. This allows us to solve for unconstrained optimal agent behavior *before* determining the equilibrium crash time.

This mathematical structure enables us to rewrite the equilibrium system as two separable sets of equations that can be solved sequentially rather than simultaneously, dramatically simplifying the computational problem.

### 0.1.3 The Two Separable Equations

In reversed time coordinates, we can rewrite the equilibrium system in the following way:

**Unconstrained Optimal Withdrawal Times** The unconstrained optimal withdrawal policies solve:

$$\bar{\tau}_{OUT}^{UNC} = \sup\{\bar{\tau} : \bar{h}(\bar{\tau}) > u\} \tag{1}$$

$$\bar{\tau}_{IN}^{UNC} = \inf\{\bar{\tau} : \bar{h}(\bar{\tau}) > u\} \tag{2}$$

**Constraint Application and Equilibrium** The actual withdrawal times incorporate the constraint that withdrawal decisions must occur before bank collapse. This leads to the aggregate withdrawal function at the moment of collapse, $\xi$:

$$AW(\xi) = G(\min(\xi, \bar{\tau}_{OUT}^{UNC})) - G(\min(\xi, \bar{\tau}_{IN}^{UNC})) \tag{3}$$

The equilibrium crash time is the first time at which aggregate withdrawals reach the fragility threshold:

$$\xi^* = \inf\{\xi : AW(\xi) \geq \kappa\} \tag{4}$$

While the final equation for $\xi^*$ is still highly non-linear, the system is now decoupled. We can first solve for the unconstrained $\bar{\tau}$ values and then take them as given when solving the final equation for $\xi^*$. Each step involves solving for one unknown in one equation, for which standard root-finding algorithms are well-suited.

### 0.1.4 Staged Computational Architecture

The solution method employs a three-tier staged approach that exploits this decoupled structure:

1. **Stage 1**: Solve learning dynamics $G(t)$.
2. **Stage 2**: Solve for unconstrained optimal withdrawal times $\bar{\tau}_{IN}^{UNC}$ and $\bar{\tau}_{OUT}^{UNC}$.
3. **Stage 3**: Solve the equilibrium condition for the crash time $\xi^*$ using the constrained $AW(\xi)$.

### 0.1.5 Remark on Computations

While some components of the baseline model admit closed-form solutions (e.g., the logistic $G(t)$), we employ a numerical approach. This provides a unified framework that readily accommodates model extensions discussed in the main text, such as the social learning model (Section 7.3) where learning dynamics lack an analytical solution.

## 0.2 Stage 1: Learning Dynamics Solver

### 0.2.1 Logistic Model Formulation

The learning process follows a logistic ODE describing information diffusion through the depositor population:

$$\frac{dG}{dt} = \beta G(t)(1 - G(t)) \tag{5}$$

where $G(t)$ represents the cumulative fraction of agents who have learned about potential bank fragility by time $t$, and $\beta > 0$ is the communication speed parameter.

### 0.2.2 Problem Setup

The ODE is solved over a time domain $[0, T]$ with initial condition $G(0) = G_0$, where $G_0$ is a small positive number (typically $10^{-4}$) to initialize the process. The time horizon $T$ is chosen such that learning approaches saturation, $G(T) > 0.999$.

### 0.2.3 Technical Implementation

The implementation uses standard adaptive ODE solvers (I use a 5th-order Runge-Kutta with automatic step size control). The logistic equation's multi-scale nature—rapid initial growth followed by slower convergence—is handled efficiently by the solver's automatic step-size adjustments. We set tolerances to machine epsilon ($\approx 1 \times 10^{-16}$) to ensure maximum precision for subsequent equilibrium calculations. The adaptive grid generated by the solver is inherited by all subsequent computational stages to maintain numerical consistency. The discrete solution points $(t_i, G(t_i))$ are converted to a continuous function $G(t)$ via linear interpolation.

### 0.2.4 PDF Computation

The learning probability density function is computed analytically from the ODE's right-hand side:

$$g(t) = \frac{dG}{dt} = \beta G(t)(1 - G(t)) \tag{6}$$

where $G(t)$ is the numerical solution from the ODE solver. We evaluate $g(t_i)$ at each grid point $\{t_i\}$ and create a corresponding interpolating function $g(t)$. This avoids numerical differentiation errors while preserving the adaptive grid structure.

## 0.3 Stage 2: Computing Unconstrained Optimal Withdrawal Times

### 0.3.1 Problem Statement and Approach

The goal of this stage is to find the unconstrained optimal withdrawal times that solve:

$$\bar{h}(\bar{\tau}_{IN}^{UNC}) = u \quad \text{and} \quad \bar{h}(\bar{\tau}_{OUT}^{UNC}) = u \tag{7}$$

We solve this by first computing $\bar{h}(\bar{\tau})$ over its full domain and then extracting the roots from the precomputed function, leveraging its known unimodal structure.

### 0.3.2 Hazard Rate Function Computation

**Grid Construction and Inheritance Strategy** The computation operates on a reversed time grid derived from the forward time grid $\{t_i\}$ from Stage 1, preserving the adaptive density. We filter this grid to retain only points within the awareness window, $\bar{\tau}_j \leq \eta$.

**Efficient One-Shot Integration** The hazard rate uses Lemma 2's closed-form expression, which contains an integral term that is computationally expensive to evaluate repeatedly:

$$\bar{h}(\bar{\tau}) = \frac{p e^{\lambda \bar{\tau}} g(\bar{\tau})}{p \int_0^{\bar{\tau}} e^{\lambda s} g(s) ds + (1-p) I_\eta} \tag{8}$$

We resolve this bottleneck by computing the integral for all grid points in a single pass using cumulative summation with the trapezoidal rule. This one-shot operation avoids the high cost of repeated numerical integration calls inside an iterative root-finder. Once the integral term is computed for all grid points, $\bar{h}(\bar{\tau})$ is evaluated efficiently using vectorized operations.

### 0.3.3 Root Extraction via Grid Search and Interpolation

**Exploiting Theoretical Structure** The grid-based approach exploits the unimodal structure of $\bar{h}(\bar{\tau})$ established in Lemma 2, which guarantees at most two solutions to $\bar{h}(\bar{\tau}) = u$.

**Efficient Boundary Detection and Refinement** With $\bar{h}(\bar{\tau})$ evaluated at all grid points, root extraction is a two-step process:

1. **Grid Scan**: Create an indicator array $\mathbb{I}[\bar{h}(\bar{\tau}_j) > u]$ to identify the grid points that bracket the roots. $\bar{\tau}_{IN}^{UNC}$ is bracketed by the first transition from 0 to 1, and $\bar{\tau}_{OUT}^{UNC}$ by the last transition from 1 to 0.

2. **Interpolation**: Once the bracketing grid points are found, a precise root is obtained via linear interpolation. This is computationally inexpensive while being more accurate than simply snapping to the nearest grid point.

3. **Boundary Cases**: If no grid points satisfy $\bar{h}(\bar{\tau}_j) > u$, both withdrawal times are set to the domain boundary $\eta$.

## 0.4 Stage 3: Equilibrium Solver via Adaptive Bisection

### 0.4.1 Problem Statement and Algorithm Motivation

The equilibrium crash time $\xi^*$ is found by solving the root problem $AW(\xi) = \kappa$, where:

$$AW(\xi) = G(\min(\xi, \bar{\tau}_{OUT}^{UNC})) - G(\min(\xi, \bar{\tau}_{IN}^{UNC})) \tag{9}$$

We use this general form because it extends robustly to multi-group extensions where it may not be obvious which constraints are binding. The equilibrium is defined as the *first* time withdrawals reach the threshold:

$$\xi^* = \inf\{\xi : AW(\xi) \geq \kappa\} \tag{10}$$

### 0.4.2 Notes on Well-Posedness

The piecewise structure of $AW(\xi)$ and the unimodal nature of the underlying learning density $g(\xi)$ imply that the equation $AW(\xi) = \kappa$ can have up to two solutions. Our definition of equilibrium as the *infimum* selects the smaller of the two, which lies on the increasing portion of the $AW(\xi)$ function.

### 0.4.3 Adaptive Bisection Algorithm

We solve $AW(\xi) = \kappa$ using a bisection algorithm guided by a finite-difference check to ensure we find the correct (first) root.

**Algorithm Setup**

- **Initialization**: We set search bounds, e.g., $\xi_{min} = 0$ and $\xi_{max} = T$.
- **Function evaluation**: $AW(\xi_{guess}) = G(\min(\xi_{guess}, \bar{\tau}_{OUT}^{UNC})) - G(\min(\xi_{guess}, \bar{\tau}_{IN}^{UNC}))$.
- **Error assessment**: $error = AW(\xi_{guess}) - \kappa$.

**Adaptive Bound Update Logic** The bisection logic is adapted to handle the unimodality. We check the local slope using a finite difference to ensure we are converging to a solution on an increasing segment of $AW(\xi)$.

1. **Case 1** ($error > tolerance$): $AW > \kappa$ (overshoot) $\rightarrow$ We've gone too far. Set $\xi_{max} = \xi_{guess}$.
2. **Case 2** ($error < -tolerance$ and $AW$ is locally increasing): $AW < \kappa$ and we are approaching the first root $\rightarrow$ Correct direction. Set $\xi_{min} = \xi_{guess}$.
3. **Case 3** ($error < -tolerance$ and $AW$ is locally decreasing): $AW < \kappa$ but we are on the far side of the peak $\rightarrow$ We've overshot both potential roots. Set $\xi_{max} = \xi_{guess}$.
4. **Case 4** ($|error| \leq tolerance$): Converged.

**The No-Run Case** The algorithm robustly handles the "no-run" case. If a run equilibrium does not exist for a given set of parameters (as discussed in the main text), it is because the condition $\max_\xi AW(\xi) < \kappa$ holds. In this situation, the bisection search will fail to find a root, with the search interval $|\xi_{max} - \xi_{min}|$ collapsing below tolerance without the error condition being met. This failure to converge correctly identifies the absence of a run equilibrium.

## 0.5 Extensions to the Baseline Model

This section documents the computational methods for the three extensions: heterogeneous learning speeds, interest rates on deposits and endogenous social learning from observed withdrawals. Each extension preserves the staged approach but requires specific modifications to handle the increased complexity.

### 0.5.1 Heterogeneous Learning Speeds

**Mathematical Framework** Population divided into $K$ groups with different communication speeds $\beta_k$ and population shares $\rho_k$, where $\sum_{k=1}^{K} \rho_k = 1$. The learning dynamics follow an SI network model where information spreads through cross-group interactions. Let $G_k(t)$ denote the fraction of informed agents in group $k$.

**Modified Learning Dynamics** The key computational change is the coupled system of ODEs governing information diffusion:

$$\frac{dG_k}{dt} = \beta_k(1 - G_k(t)) \sum_{j=1}^{K} \rho_j G_j(t) \tag{11}$$

This represents agents in group $k$ learning from the overall pool of informed agents $\omega(t) = \sum_j \rho_j G_j(t)$. The closed-form solution exists for certain distibution[1] but involves complex expressions and do not extend to the social learning setting bellow. We solve this system numerically using the same adaptive ODE solver as the baseline model.

**Aggregate Withdrawal Computation** The aggregate withdrawal function becomes a weighted sum across groups:

$$AW(\xi) = \sum_{k=1}^{K} \rho_k \left[ G_k(\min(\xi, \bar{\tau}_{OUT,k}^{UNC})) - G_k(\min(\xi, \bar{\tau}_{IN,k}^{UNC})) \right] \tag{12}$$

Each group $k$ has its own learning CDF $G_k(t)$ but shares the same hazard rate function structure. The unconstrained optimal withdrawal times $\bar{\tau}_{IN,k}^{UNC}$ and $\bar{\tau}_{OUT,k}^{UNC}$ are computed using the same root-finding approach as the baseline model, applied to each group's hazard rate. One key advantage of the approach we described above is that we can remain agnostic on wich group will have binding constraint and which will not (in particular it is possible for a group to have both $0 < \bar{\tau}_{OUT}^{UNC}, \bar{\tau}_{OUT}^{UNC} < \xi$).

**Computational Workflow**

1. Solve the coupled ODE system for $\{G_k(t)\}_{k=1}^{K}$ on the adaptive grid.
2. Extract optimal withdrawal times for each group using the baseline algorithm.
3. Solve for equilibrium $\xi^*$ using the weighted aggregate withdrawal function.

The equilibrium solver (Stage 3) remains unchanged except for the modified $AW(\xi)$ calculation.

### 0.5.2 Interest Rates and Value Functions

**Problem Formulation** With positive interest rate $r > 0$, agents maximize expected discounted utility. This introduces a value function $V(\tau)$ representing the continuation value of holding deposits, where $\tau = \xi - t$ is reversed time (time remaining until potential collapse). The optimal holding decision becomes:

$$\text{Hold deposits if and only if } h(\tau) < u + rV(\tau) \tag{13}$$

This modifies the threshold condition from $h(\tau) < u$ to $h(\tau) - rV(\tau) < u$.

[1]The closed form solution invovles the moment-generating functions of the distribution

**Value Function Computation** The value function satisfies an ordinary differential equation derived from the Hamilton-Jacobi-Bellman equation:

$$V'(\tau) = (\delta - r)V(\tau) - u + \max\{0, h(\tau) - rV(\tau) - u\} \tag{14}$$

where $\delta$ is the discount/recovery rate. The boundary condition is $V(0) = \delta/(\delta - r)$, representing in normal time the terminal value at $\xi$ when the agent is certain that a surviving bank is safe.

We solve this ODE backward from $\tau = \eta$ to $\tau = 0$ using the same adaptive grid inherited from the learning dynamics solver. This ensures consistency across all computational stages and avoids interpolation errors.

**Modified Optimal Buffer Computation** Although the value function implicitly contains the optimal holding strategy, we explicitly recompute the withdrawal times for clarity and numerical stability:

1. Define the modified threshold function: $\tilde{h}(\tau) = h(\tau) - rV(\tau)$
2. Apply the baseline root-finding algorithm to find:

$$\bar{\tau}_{IN}^{UNC} = \inf\{\tau : \tilde{h}(\tau) > u\} \tag{15}$$

$$\bar{\tau}_{OUT}^{UNC} = \sup\{\tau : \tilde{h}(\tau) > u\} \tag{16}$$

This approach reuses the efficient grid-based root extraction from the baseline model while incorporating the value function effects. The subsequent equilibrium computation (Stage 3) proceeds unchanged with these modified withdrawal times.

### 0.5.3 Social Learning from Withdrawals

**The Endogenous Learning Challenge** Social learning fundamentally alters the model's structure by making learning dynamics depend on equilibrium behavior. Agents learn about bank fragility by observing others' withdrawals, creating a fixed-point problem: aggregate withdrawals determine learning, learning determines optimal behavior, and optimal behavior determines aggregate withdrawals.

Mathematically, this circular dependency manifests as:

$$\frac{dG}{dt} = \beta(1 - G(t)) \times AW(t) \tag{17}$$

where $AW(t)$ itself depends on the learning distribution $G(t)$ through agents' optimal strategies.

**Fixed-Point Problem Formulation** The equilibrium must satisfy the following coupled system:

1. Given $AW(t)$, solve the learning ODE: $dG/dt = \beta(1 - G)AW$
2. Given $G(t)$, compute hazard rates and optimal withdrawal times
3. Given withdrawal times, compute: $AW(t) = G(\min(t, \bar{\tau}_{OUT}^{UNC})) - G(\min(t, \bar{\tau}_{IN}^{UNC}))$
4. Verify consistency: the $AW(t)$ from step 3 must equal the $AW(t)$ assumed in step 1

**Iterative Solution via Damped Fixed-Point Iteration** We solve this fixed-point problem through a damped fixed-point iteration scheme on the aggregate withdrawal function:

1. **Initialization**: Obtain initial guess $AW^{(0)}(t)$ by solving the baseline model (word-of-mouth learning) without social learning effects.

2. **Iteration**: At each iteration $n \geq 1$:

   - Solve learning ODE with forcing term $AW^{(n-1)}(t)$: $\frac{dG}{dt} = \beta(1-G(t)) \times AW^{(n-1)}(t)$
   - Given $G(t)$, compute hazard rates and optimal withdrawal times $\bar{\tau}_{IN}^{UNC}$, $\bar{\tau}_{OUT}^{UNC}$
   - Compute implied equilibrium collapse time $\xi^*$ and withdrawal function: $\tilde{AW}^{(n)}(t)$ (I reuse the bisection algorithm on $\xi$ from the baseline but replace the withdrawal times and learning functions by what is obtained in step 1 and step 2).
   - Apply damping with fixed parameter $\alpha = 0.5$:

     $$AW^{(n)}(t) = (1-\alpha)AW^{(n-1)}(t) + \alpha\tilde{AW}^{(n)}(t) \tag{18}$$

     Alternative damping schemes are possible but practice shows a uniform rule works well.
   - Check convergence: $\|AW^{(n)} - AW^{(n-1)}\|_\infty < \epsilon$

3. **Convergence**: Stop when iteration error falls below tolerance, or maximum iterations is reached.

**Inner Loop Failure** For a given $AW^{(n-1)}(t)$, the equilibrium solver (Stage 3 from the baseline model) may find no crash time $\xi$ satisfying $AW(\xi) = \kappa$. Without a crash time, the next iteration's aggregate withdrawal function cannot be constructed. We address this by setting $\xi^{(n)} = \xi^{(n-1)} + \eta/500$, a small increment that allows the iteration to continue. While this lacks theoretical foundation, it permits the algorithm to continue exploring for a potential solution.

**Handling No-Run Cases** The algorithm identifies no-run cases by failure to converge. In contrast to the baseline case, the identification is not fully grounded in theory and simply a pragmatic solution. Empirically, non-convergence reliably indicates that no social learning run equilibrium exists for the given parameters, as convergent sequences are observed whenever equilibria exist. Non-convergent sequences typically oscillate or stay at $AW = 0$. The theoretical shortcoming is there is a theoretical possibility that we fail to capture a run equilibrium (and label it as non-run), in particular because the search path depends on the initial guess and the update rule in case of non-run in the inner loop.

**Computational Cost** Convergence typically requires approximately 40 iterations. Each iteration involves solving a complex ODE with the time-varying forcing term $AW^{(n-1)}(t)$, which is computationally more demanding than the baseline logistic ODE. Total computation time is approximately 20 seconds, compared to 0.5 seconds for the baseline model.

## 0.6 Implementation Details and Figure Parameters

This section documents the exact parameter choices used to generate the figures in the main text. All computations use the Julia implementation available in the online supplementary materials.

### 0.6.1 Baseline Model Parameters

The baseline model parameters, used as the foundation for all figures unless otherwise specified, are:

| Parameter | Symbol | Value |
|---|---|---|
| Communication speed | $\beta$ | 1.0 |
| Raw awareness window | $\bar{\eta}$ | 15.0 |
| Awareness window | $\eta$ | 15.0 ($= \bar{\eta}/\beta$) |
| Utility flow from deposits | $u$ | 0.1 |
| Prior fragility probability | $p$ | 0.5 |
| Solvency threshold | $\kappa$ | 0.6 |
| Exponential rate for $t_0$ | $\lambda$ | 0.01 |
| Initial learning condition | $G(0)$ | 0.0001 |
| Time span | – | $(0, 2\eta) = (0, 30)$ |

Table 1: Baseline model parameters used throughout the paper.

### 0.6.2 Figure-Specific Parameters

Table **??** summarizes the parameter values used for each figure in the main text. When a cell shows a range (e.g., $[0.001, 0.2]$), this indicates comparative statics over that parameter range. All other parameters are held at baseline values unless otherwise specified.

| Figure | $\beta$ | $u$ | $\kappa$ | Description |
|---|---|---|---|---|
| Fig. 1: Learning | $\{0.5, 1.0, 2.0\}$ | 0.1 | 0.6 | Multiple $\beta$ comparison |
| Fig. 2: Hazard Rate | 1.0 | 0.1 | 0.6 | Baseline scenario |
| Fig. 3a: Main Eq. | 1.0 | 0.1 | 0.6 | Main scenario |
| Fig. 3b: Fast Comm. | 3.0 | 0.1 | 0.6 | High $\beta$ case |
| Fig. 3c: Low Utility | 1.0 | 0.01 | 0.6 | Low $u$ case |
| Fig. 4: $u$ Effects | 1.0 | $[0.001, 0.2]$ | 0.6 | Comp. statics in $u$ |
| Fig. 5: Cross-Effects | $[1, 10^4]$ | $[0.001, 1]$ | 0.6 | 2D parameter sweep |

Table 2: Parameter values for main text figures. All use $p = 0.5$, $\lambda = 0.01$, $\eta = 15/\beta$, time span $(0, 2\eta)$ unless noted. Ranges indicate comparative statics. Figure 4 uses 5000 grid points, Figure 5 uses $500 \times 500$ grid with $\beta = 1/\text{meeting time}$ where meeting time $\in [0.0001, 1]$.

### 0.6.3 Computational Settings

All computations use machine precision tolerances ($\approx 10^{-16}$) for ODE solving and $10^{-12}$ relative tolerance for equilibrium root-finding. The Julia implementation exploits the staged computational architecture to minimize redundant calculations, particularly by reusing learning dynamics across multiple equilibrium computations within parameter sweeps.