Sovereign Default: Eaton-Gersovitz / Arellano (2008)

David Evans

2026-11-13

Introduction

From Domestic to International

Where We've Been

- **McCall (1970)**: an unemployed worker searches for a job — solve for a reservation wage
- **Rust (1987)**: a bus superintendent decides when to replace engines — estimate structural parameters
- **Kydland & Prescott (1982)**: a representative agent in a macroeconomy — generate business cycles
- **DMP (1982–85)**: workers and firms in a search-and-matching labor market — general equilibrium

- All of these models feature **domestic** agents making decisions within a single economy
- Today we go **international**: a sovereign government deciding whether to repay or default on its debts

Sovereign Default

- Sovereign default: a national government fails to meet its **debt obligations**
- Unlike corporate default, there is **no bankruptcy court** — enforcement relies on indirect costs

- Historical episodes:
  - Argentina (2001): defaulted on $100 billion in debt
  - Greece (2012): largest sovereign restructuring in history
  - Russia (1998): default triggered a global financial crisis
  - Ecuador (2008, 2020): serial defaulter

The Key Question

Why do countries **ever repay** their debts? Without legal enforcement, what prevents default? And what determines the **price** of sovereign bonds?

The Eaton-Gersovitz / Arellano Framework

- **Eaton & Gersovitz (1981)**: foundational model of sovereign debt
- **Arellano (2008, *AER*)**: quantitative implementation calibrated to Argentina

- Key ingredients:
  1. A small open economy with **stochastic income**
  2. A government that borrows from international markets
  3. Each period, the government chooses to **repay** or **default**
  4. Default triggers **exclusion from credit markets** and an **output cost**
  5. Competitive risk-neutral lenders set **bond prices** endogenously

Connection to Previous Lectures

The sovereign default model combines: **dynamic programming** (McCall, Rust, KP), **Markov chains** (income process), **value function iteration** (KP), and **equilibrium pricing** (DMP).

Roadmap

1. **The Model** — the government's problem, lenders, and equilibrium
2. **Setup in Julia** — structs, Rouwenhorst discretization, and grids
3. **Solving the Model** — value function iteration and bond price iteration
4. **Results** — policy functions, bond prices, and default regions
5. **Simulation** — debt dynamics, default episodes, and sovereign spreads

The Model

The Government

The Government's Problem

- A small open economy receives stochastic **endowment** y_t each period (no production)
- The endowment follows an AR(1) process in logs:

$$\log y_{t+1} = \rho \log y_t + \sigma_\varepsilon \varepsilon_{t+1}, \qquad \varepsilon_{t+1} \sim N(0,1)$$

- The government has **one-period discount bonds** with face value b (negative b = debt)
- Each period, the government starts with income y and outstanding debt $-b$ (if $b < 0$)

- The government maximizes expected lifetime utility of the representative household:

$$\mathbb{E}_0 \sum_{t=0}^{\infty} \beta^t u(c_t)$$

where $u(c) = \frac{c^{1-\sigma}}{1-\sigma}$ is CRRA utility

The Repayment Decision

- At the beginning of each period, the government **observes** income y and decides:
  - **Repay**: honor the debt, maintain access to credit markets
  - **Default**: refuse to pay, get excluded from borrowing

- If the government **repays**:

$$c = y + b - q(b', y) \cdot b'$$

- b: current bond holdings (negative = debt owed)
- b': new bonds issued (negative = new borrowing)
- $q(b', y)$: price per unit of debt, set by lenders
- $q(b', y) \cdot b'$: revenue from issuing new debt

Reading the Budget Constraint

If the government has debt ($b < 0$), it must pay $|b|$ to old creditors. It can borrow $q \cdot |b'|$ from new creditors. The net resources available for consumption are $y + b - q \cdot b'$.

The Default Decision

- If the government **defaults**:
  - It is **excluded** from international credit markets (cannot borrow or save)
  - It suffers an **output cost**: income drops to $y^{\text{def}} = h(y)$
- The output cost function (Arellano 2008):

$$h(y) = \min\{y,\ \hat{y}\}$$

- When income is **above** $\hat{y}$: the government loses the excess — default is costlier in good times
- When income is **below** $\hat{y}$: no additional cost
- Each period in autarky, the government regains access to credit markets with probability λ

Value Functions

- Let $V^R(b, y)$ = value of **repaying** and $V^D(y)$ = value of **defaulting**
- **Value of repayment** (choose new debt b' optimally):

$$V^R(b, y) = \max_{b'} \left\{ u(y + b - q(b', y) \cdot b') + \beta \, \mathbb{E}\left[V(b', y') \mid y \right] \right\}$$

- **Value of default** (consume $h(y)$, hope to regain access):

$$V^D(y) = u(h(y)) + \beta \, \mathbb{E}\left[\lambda \, V(0, y') + (1 - \lambda) \, V^D(y') \mid y \right]$$

- **The government's choice**:

$$\boxed{V(b, y) = \max \left\{ V^R(b, y), \; V^D(y) \right\}}$$

The Default Set

- For each debt level b, there is a set of income realizations where the government defaults:

$$D(b) = \left\{ y : V^D(y) > V^R(b, y) \right\}$$

- The **default probability** from the lenders' perspective:

$$\delta(b', y) = \sum_{y' \in D(b')} \Pr(y' \mid y)$$

When Does Default Happen?

Default is more likely when:

- **Debt is high** ($|b|$ large) — repayment is more burdensome
- **Income is low** (y small) — the government cannot afford to repay
- **Output cost is low** — default is less punishing

International Lenders

Bond Pricing

- International lenders are **risk-neutral** and competitive
- They face a risk-free rate r
- They lend to the sovereign knowing it may default

- Zero-profit condition: expected return on sovereign bonds = risk-free rate

$$q(b', y) = \frac{1 - \delta(b', y)}{1 + r}$$

- $\delta(b', y)$: probability the government defaults **next period** given new debt b' and current income y
- If default is certain ($\delta = 1$): $q = 0$ — no one will lend
- If no default risk ($\delta = 0$): $q = \frac{1}{1+r}$ — risk-free price

The Sovereign Spread

- Define the **sovereign spread** as the interest rate premium over the risk-free rate:

$$\text{spread}(b', y) = \frac{1}{q(b', y)} - (1 + r)$$

- Higher default risk → lower bond price → higher spread
- This is exactly what we observe in data: countries with more debt and lower income face higher borrowing costs

Feedback Loop

Higher spreads → borrowing is more expensive → debt grows faster → default risk rises → even higher spreads. This can generate **debt crises** that spiral out of control.

Recursive Equilibrium

Definition

Recursive Equilibrium

A recursive equilibrium consists of:

1. **Value functions** $V(b, y), V^R(b, y), V^D(y)$
2. **Policy functions** $b'(b, y)$ (borrowing) and $d(b, y) \in \{0, 1\}$ (default)
3. **Bond price function** $q(b', y)$

such that:

- Given q, the government optimizes (Bellman equations hold)
- Given the government's policy, lenders break even (q satisfies the zero-profit condition)

Comparing to Previous Models

	McCall	Rust	KP	Sovereign Default
State	Wage offer	Mileage	Capital $\times$ TFP	**Debt $\times$ Income**
Choice	Accept/Reject	Replace/Keep	Savings	**Repay/Default + Borrowing**
State space	Finite	Finite	Continuous $\times$ Finite	**Continuous $\times$ Finite**
Prices	Exogenous	—	Factor prices	**Bond price $q(b', y)$**
Equilibrium	Partial	—	First Welfare Theorem	**Fixed point: V and q**

The New Challenge

The bond price $q(b', y)$ depends on the government's **future** default decision, which depends on the bond price. We need to iterate on **both** V and q simultaneously until they are consistent.

Setup in Julia

Model Structure

Loading Packages

```
using Plots
using LinearAlgebra
using Statistics
using Distributions
using Random
using Interpolations
```

Rouwenhorst Discretization

- We reuse the Rouwenhorst method from the KP lecture to discretize the income process:

```
function rouwenhorst(N, ρ, σ_ε)
    σ_z = σ_ε / sqrt(1 - ρ^2)
    p = (1 + ρ) / 2
    P = [p 1-p; 1-p p]
    for n in 3:N
        z = zeros(n - 1)
        P = p     * [P z; z' 0] + (1-p) * [z P; 0 z'] +
            (1-p) * [z' 0; P z] + p     * [0 z'; z P]
        P[2:end-1, :] ./= 2
    end
    z_grid = collect(LinRange(-sqrt(N-1) * σ_z,
                              sqrt(N-1) * σ_z, N))
    return z_grid, P
end
```

```
rouwenhorst (generic function with 1 method)
```

The `SovDefaultModel` Struct

```julia
@kwdef mutable struct SovDefaultModel
    β::Float64 = 0.980                      # discount factor (calibrated to 0.75% quarterly default)
    σ::Float64 = 2.0                        # risk aversion (CRRA)
    r::Float64 = 0.017                      # risk-free interest rate
    ρ::Float64 = 0.945                      # income persistence
    σ_ε::Float64 = 0.025                    # income innovation std dev
    λ::Float64 = 0.282                      # re-entry probability
    ŷ::Float64 = 0.969                      # default output ceiling
    τ::Float64 = 0.001                      # taste shock scale (smoothing)
    N_y::Int = 21                           # number of income states
    N_b::Int = 101                          # sparse grid (value function storage)
    N_b_fine::Int = 251                     # fine grid (optimization search)
    b_min::Float64 = -0.4                   # minimum bond holdings (max debt)
    b_max::Float64 = 0.4                    # maximum bond holdings (max savings)

    y_grid::Vector{Float64} = Float64[]     # filled by setup!
    P_y::Matrix{Float64} = zeros(0, 0)      # filled by setup!
    b_grid::Vector{Float64} = Float64[]     # sparse grid, filled by setup!
```

```
SovDefaultModel
```

The `setup!` Function

```
function setup!(m::SovDefaultModel)
    (; ρ, σ_ε, N_y, N_b, N_b_fine, b_min, b_max) = m
    log_y, P = rouwenhorst(N_y, ρ, σ_ε)
    m.y_grid = exp.(log_y)
    m.P_y = P
    m.b_grid = collect(LinRange(b_min, b_max, N_b))
    m.b_grid_fine = collect(LinRange(b_min, b_max, N_b_fine))
    return m
end
```

```
setup! (generic function with 1 method)
```

```
m = setup!(SovDefaultModel())
iy_set = round.(Int, LinRange(1, m.N_y, 5))
println("Income grid: ", round.(m.y_grid[iy_set], digits=3))
println("Sparse grid ($(m.N_b) pts): [$(round(m.b_grid[1], digits=3)), ..., $(round(m.b_grid[end], digits=3))]")
println("Fine grid ($(m.N_b_fine) pts): [$(round(m.b_grid_fine[1], digits=3)), ..., $(round(m.b_grid_fine[end], digits=3))]")
```

```
Income grid: [0.71, 0.843, 1.0, 1.186, 1.408]
Sparse grid (101 pts): [-0.4, ..., 0.4]
Fine grid (251 pts): [-0.4, ..., 0.4]
```

CRRA Utility

```
function u(c, σ)
    c <= 0 && return -1e14
    σ == 1.0 && return log(c)
    return c^(1 - σ) / (1 - σ)
end
```

```
u (generic function with 1 method)
```

- Returns a large penalty for negative consumption (infeasible)
- Special-cases $\sigma = 1$ as log utility (the limit of CRRA as $\sigma \to 1$)

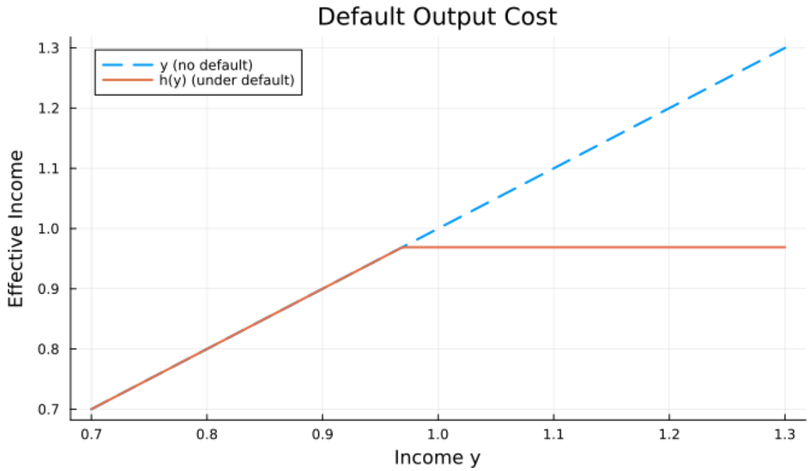
The Default Output Cost

- Arellano's asymmetric cost function:

```
1 h(y, ŷ) = min(y, ŷ)
```

```
h (generic function with 1 method)
```

```
1 y_fine = LinRange(0.7, 1.3, 200)
2 plot(y_fine, y_fine, linewidth=2, linestyle=:dash, label="y (no default)",
3      legend=:topleft, size=(700, 400))
4 plot!(y_fine, h.(y_fine, m.ŷ), linewidth=2, label="h(y) (under default)")
5 xlabel!("Income y")
6 ylabel!("Effective Income")
7 title!("Default Output Cost")
```



- Default is costlier when income is **high** — the government loses more output

Solving the Model

The Algorithm

Overview

1. **Initialize** value functions V^R, V^D and bond prices q
2. **Iterate**:
   a. Given q, update the government's value functions and policies
   b. Given the default policy, update bond prices q
3. **Repeat** until both V and q converge

> ⚠ **Two Nested Fixed Points**
>
> Unlike KP (where we only iterated on V), here we iterate on **both** V and q. The bond price depends on future default decisions, which depend on the bond price.

Interpolation in Sovereign Default

- In the KP lecture we introduced **linear interpolation** to handle continuous capital k
- Here we reuse the same tool — but with a twist

- In KP, the objective was **concave** in k', so gradient-based optimization found the global optimum reliably
- Here, the endogenous bond price $q(b', y)$ creates **kinks** in the objective — it is not globally concave
- We use **grid search** over a fine grid (251 pts) while storing value functions on a sparse grid (101 pts) — simple, robust, and requires no concavity

Where Interpolation Helps

We store V and q on a **sparse grid** (101 points) and build `linear_interpolation` per income state. In the Bellman operator, we search over a **fine grid** (251 points), evaluating interpolated V and q. This separates approximation accuracy from optimization accuracy. `V_itp[iy](0.0)` cleanly evaluates $V(0, y')$ for the default value.

Taste Shocks: Connecting to Rust

- In the Rust lecture, we added **Type I Extreme Value** shocks to the keep/replace decision
- The same idea applies here: the government's repay/default decision is perturbed by taste shocks

- **Without taste shocks**: $V(b, y) = \max\{V^R(b, y),\ V^D(y)\}$ — sharp cutoff
- **With taste shocks** (scale τ):

$$V(b, y) = \tau \log\left[\exp\left(\frac{V^R}{\tau}\right) + \exp\left(\frac{V^D}{\tau}\right)\right]$$

- Default probability becomes a **smooth logistic** (just like Rust's replacement probability):

$$\delta(b, y) = \frac{1}{1 + \exp\left(\frac{V^R(b,y) - V^D(y)}{\tau}\right)}$$

The Role of τ

As $\tau \to 0$, we recover the original hard-max model. We set $\tau = 0.001$ as our baseline: this is close to the hard-max model while still providing a smooth default probability for numerical stability. The parameter τ plays the same role as the implicit scale $= 1$ in the Rust lecture.

`smooth_max` and `default_prob`

- Two helper functions, using the **log-sum-exp trick** from the Rust lecture:

```
1 function smooth_max(v1, v2, τ)
2     vmax = max(v1, v2)
3     return vmax + τ * log(exp((v1 - vmax) / τ) + exp((v2 - vmax) / τ))
4 end
5
6 function default_prob(VR, VD, τ)
7     return 1.0 / (1.0 + exp((VR - VD) / τ))
8 end
```

```
default_prob (generic function with 1 method)
```

- `smooth_max`: numerically stable log-sum-exp (subtracts the max before exponentiating)
- `default_prob`: logistic sigmoid – $P(\text{default}) = \frac{1}{1+e^{(V^R - V^D)/\tau}}$

Step 1: The Value of Default

- For each income state y_j, the value of default is:

$$V^D(y_j) = u(h(y_j)) + \beta \sum_{j'} P_{j,j'} \left[\lambda V(0, y_{j'}) + (1-\lambda) V^D(y_{j'}) \right]$$

- The government consumes $h(y_j)$ today (with output cost)
- Next period: regains market access with probability λ (enters with zero debt), otherwise stays in autarky

Step 2: The Value of Repayment

- For each (b_i, y_j), find the optimal new debt b':

$$V^R(b_i, y_j) = \max_{b'} \left\{ u(y_j + b_i - q(b', y_j) \cdot b') + \beta \sum_{j'} P_{j,j'} V(b', y_{j'}) \right\}$$

- Search over the bond grid with interpolated V and q to find the best b'
- Consumption: $c = y_j + b_i - q(b', y_j) \cdot b'$

Step 3: Update Bond Prices

- Compute the **smooth** default probability for each (b', y) using the logistic:

$$\delta(b', y_j) = \sum_{j'} P_{j,j'} \cdot \frac{1}{1 + \exp\left(\frac{V^R(b', y_{j'}) - V^D(y_{j'})}{\tau}\right)}$$

- Update the bond price:

$$q(b', y_j) = \frac{1 - \delta(b', y_j)}{1 + r}$$

Julia Implementation

`optimal_default`: Pseudocode

- The core building block for **default**: given income index `iy`, the current value function, and VD, compute the value of defaulting today

```
function optimal_default(m, iy, V_itp, VD)
    # Consume h(y) (income with output cost)
    # Continuation: regain access (V(0, y')) with prob λ, stay in autarky (VD(y')) otherwise
    # Return the value of default
end
```

`optimal_default`: Full Function

```
function optimal_default(m, iy, V_itp, VD)
    (; β, σ, λ, ŷ, N_y, y_grid, P_y) = m
    y = y_grid[iy]
    c_def = h(y, ŷ)                                  # income under default
    EV_def = sum(P_y[iy, iy2] * (λ * V_itp[iy2](0.0) +    # regain access at b=0
                 (1 - λ) * VD[iy2]) for iy2 in 1:N_y)      # or stay in autarky
    return u(c_def, σ) + β * EV_def
end
```

```
optimal_default (generic function with 1 method)
```

- `V_itp[iy2](0.0)` evaluates the continuation value at **zero debt** using interpolation — no need to find a grid index
- The government consumes $h(y)$ today and hopes to regain market access with probability λ

A Simple `find_max` Helper

- Before writing `optimal_repayment`, let's build a tiny utility: **grid search** over a vector to maximize a function

```
function find_max(f, grid)
    best_val = -Inf
    best_x   = grid[1]
    for x in grid
        val = f(x)
        if val > best_val
            best_val = val
            best_x   = x
        end
    end
    return (val = best_val, x = best_x)
end
```

```
find_max (generic function with 1 method)
```

- Evaluates `f` at every point in `grid` and returns the best `(val, x)`
- Simple, transparent, no external packages needed
- We search over the **fine grid** (`b_grid_fine`, 251 points) for accuracy, while storing values on the **sparse grid** (`b_grid`, 101 points)

`optimal_repayment`: Pseudocode

- The core building block for **repayment**: given current debt b, income index `iy`, and interpolated V and q, find the optimal new borrowing b'

```
function optimal_repayment(m, b, iy, V_itp, q_itp)
    # Define payoff(b'): flow utility + continuation value
    #   Consumption: c = y + b - q(b', y) × b'
    #   Continuation: E[V(b', y')] using interpolation
    # Search over fine grid using find_max
    # Return value and optimal b' as named tuple
end
```

`optimal_repayment`: Full Function

```julia
function optimal_repayment(m, b, iy, V_itp, q_itp)
    (; β, σ, N_y, y_grid, P_y, b_grid_fine) = m
    y = y_grid[iy]
    function payoff(bp)
        c = y + b - q_itp[iy](bp) * bp
        c <= 0 && return -Inf
        EV = sum(P_y[iy, iy2] * V_itp[iy2](bp) for iy2 in 1:N_y)
        return u(c, σ) + β * EV
    end
    result = find_max(payoff, b_grid_fine)
    return (V = result.val, bp = result.x)
end
```

```
optimal_repayment (generic function with 1 method)
```

- `payoff(bp)` computes the **flow utility + discounted continuation** for any candidate b'
- `find_max` searches the **fine grid** (251 points) — more resolution than the sparse grid where we store V
- Returns a named tuple `(V, bp)` — the optimal value and borrowing level

The Bellman Operator

- Now the Bellman operator simply **calls** our two building blocks and assembles the results:

```
function bellman_operator(m, V, VR, VD, q)
    # Build interpolants for V and q (once per iteration)
    # Call optimal_default for each income state
    # Call optimal_repayment for each (b, y) state
    # Return updated value functions and policies
end
```

The Bellman Operator

```julia
function bellman_operator(m, V, VR, VD, q)
    (; N_y, N_b, b_grid) = m
    VR_new = similar(VR)
    VD_new = similar(VD)
    bp_policy = zeros(N_b, N_y)
    V_itp = [linear_interpolation(b_grid, V[:, iy]) for iy in 1:N_y]
    q_itp = [linear_interpolation(b_grid, q[:, iy]) for iy in 1:N_y]
    for iy in 1:N_y
        VD_new[iy] = optimal_default(m, iy, V_itp, VD)
    end
    for iy in 1:N_y
        for ib in 1:N_b
            result = optimal_repayment(m, b_grid[ib], iy, V_itp, q_itp)
            VR_new[ib, iy] = result.V
            bp_policy[ib, iy] = result.bp
        end
    end
    return VR_new, VD_new, bp_policy
```

```
bellman_operator (generic function with 1 method)
```

- `V_itp` and `q_itp` are built **once per iteration** (not once per state) for efficiency
- The Bellman operator is now clean and readable — each piece of logic lives in its own function

The Bond Price Update

```
function update_bond_price(m, VR, VD)
    # For each (b', y), compute smooth default probability next period
    # Bond price = (1 - default_prob) / (1 + r)
    # Return updated q matrix
end
```

The Bond Price Update

```julia
function update_bond_price(m, VR, VD)
    (; r, τ, N_y, N_b, P_y) = m
    q_new = zeros(N_b, N_y)
    for iy in 1:N_y
        for ibp in 1:N_b
            δ = sum(P_y[iy, iy2] * default_prob(VR[ibp, iy2], VD[iy2], τ)
                    for iy2 in 1:N_y)
            q_new[ibp, iy] = (1 - δ) / (1 + r)
        end
    end
    return q_new
end
```

```
update_bond_price (generic function with 1 method)
```

- `default_prob(VR, VD, τ)` returns the **smooth** logistic probability — no sharp 0/1 discontinuity
- This is the key benefit of taste shocks: bond prices vary **continuously** with debt and income

The Full Solver

```
function solveSovDefault(m; tol=1e-6, maxiter=1500, verbose=true, α=0.5)
    # Initialize value functions and bond prices
    # Iterate: Bellman update + damped bond price update
    # V uses smooth_max (log-sum-exp with τ)
    # q updated with mixing: q = α·q_new + (1-α)·q_old (prevents oscillation)
    # Build interpolated policies for simulation
    # Return solution tuple with smooth default probabilities
end
```

The Full Solver

```
function solveSovDefault(m; tol=1e-6, maxiter=1500, verbose=true, α=0.5)
    (; N_b, N_y, r, τ, b_grid) = m
    VR = zeros(N_b, N_y)
    VD = zeros(N_y)
    V = zeros(N_b, N_y)
    q = fill(1 / (1 + r), N_b, N_y)            # initial guess: risk-free price
    bp_policy = zeros(N_b, N_y)
    for iter in 1:maxiter
        VR_new, VD_new, bp_policy = bellman_operator(m, V, VR, VD, q)
        V_new = [smooth_max(VR_new[ib, iy], VD_new[iy], τ)
                 for ib in 1:N_b, iy in 1:N_y]
        q_lenders = update_bond_price(m, VR_new, VD_new)
        q_new = α .* q_lenders .+ (1 - α) .* q      # damped update
        dist_V = maximum(abs.(V_new - V))
        dist_q = maximum(abs.(q_new - q))
        if max(dist_V, dist_q) < tol
            verbose && println("Converged in $iter iterations (V: $(round(dist_V, sigdigits=3)), q: $(round(dist_q, sigdigits=3)))")
            VR = VR_new; VD = VD_new; V = V_new; q = q_new
```

```
solveSovDefault (generic function with 1 method)
```

- The **damping parameter** $\alpha = 0.5$ mixes new and old bond prices: $q \leftarrow \alpha\, q^{\text{new}} + (1-\alpha)\, q^{\text{old}}$
- With small τ, default probabilities are nearly 0/1, so q can jump dramatically between iterations
- Damping prevents oscillation and ensures convergence

Why Damping?

The bond price q depends on future default, which depends on q itself. With near-hard-max taste shocks ($\tau = 0.001$), small changes in V near the default boundary cause large swings in q. Mixing old and new prices ($\alpha = 0.5$) smooths this feedback loop.

Calibration

Calibration Strategy

- Following Arellano (2008), we calibrate the model to **quarterly Argentine data**
- Parameters are either set **externally** (from data / standard values) or calibrated **internally** to match moments

Parameter	Value	Source
σ	2.0	Standard CRRA
r	0.017	US T-bill rate (quarterly)
ρ	0.945	Argentine GDP AR(1) persistence
σ_ε	0.025	Argentine GDP innovation std dev
λ	0.282	Avg. exclusion $\approx$ 3.5 quarters
$\hat{y}$	0.969	Arellano (2008) asymmetric output cost

Calibrating β

- β controls how much the government values **future** market access
- Higher $\beta \rightarrow$ repayment more attractive $\rightarrow$ fewer defaults
- Arellano (2008) calibrates $\beta = 0.953$ to match Argentina's quarterly default rate of $\approx 0.75\%$ ($\approx 3\%$ annualized)
- With $\tau = 0.001$ we are essentially at the hard-max model, so we use Arellano's value directly

Why $\beta < 1$?

A discount factor of 0.953 is **low** compared to standard macro models ($\beta \approx 0.99$). This reflects the impatience needed to generate realistic borrowing and default. A very patient government would rarely accumulate enough debt to default.

Running the Solver

```
simulateSovDefault (generic function with 1 method)
```

```
m = setup!(SovDefaultModel())
sol = solveSovDefault(m)
println("Model: β = $(m.β), τ = $(m.τ)")
```

```
iter 25: dist_V = 0.683, dist_q = 0.0158
iter 50: dist_V = 0.384, dist_q = 8.46e-5
iter 75: dist_V = 0.228, dist_q = 3.94e-6
iter 100: dist_V = 0.137, dist_q = 1.81e-7
iter 125: dist_V = 0.0825, dist_q = 8.17e-9
iter 150: dist_V = 0.0498, dist_q = 3.7e-10
iter 175: dist_V = 0.03, dist_q = 1.69e-11
iter 200: dist_V = 0.0181, dist_q = 9.18e-13
iter 225: dist_V = 0.0109, dist_q = 2.6e-13
iter 250: dist_V = 0.0066, dist_q = 3.82e-13
iter 275: dist_V = 0.00398, dist_q = 6.5e-13
iter 300: dist_V = 0.0024, dist_q = 1.54e-12
iter 325: dist_V = 0.00145, dist_q = 1.29e-12
iter 350: dist_V = 0.000876, dist_q = 3.84e-13
```

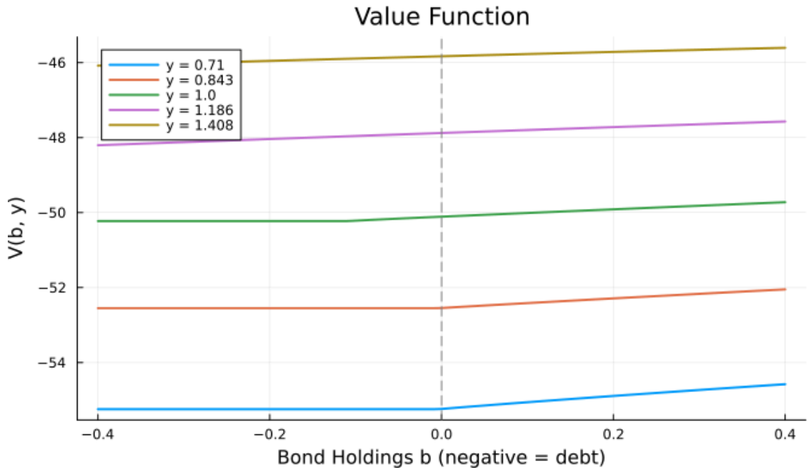
Patience

This takes longer than the KP model because we iterate on **both** value functions and bond prices. Each iteration optimizes over b' for every (b, y) state — $N_b \times N_y$ evaluations per iteration.

Results

The Value Function

```
iy_set = round.(Int, LinRange(1, m.N_y, 5))
plot(legend=:topleft, size=(700, 400))
for iy in iy_set
    plot!(m.b_grid, sol.V[:, iy], linewidth=2,
          label="y = $(round(m.y_grid[iy], digits=3))")
end
xlabel!("Bond Holdings b (negative = debt)")
ylabel!("V(b, y)")
title!("Value Function")
vline!([0.0], color=:gray, linestyle=:dash, label="")
```



- Higher income → higher value at every debt level
- Value is increasing in b (less debt = better)

Breaking Down the Repayment Objective

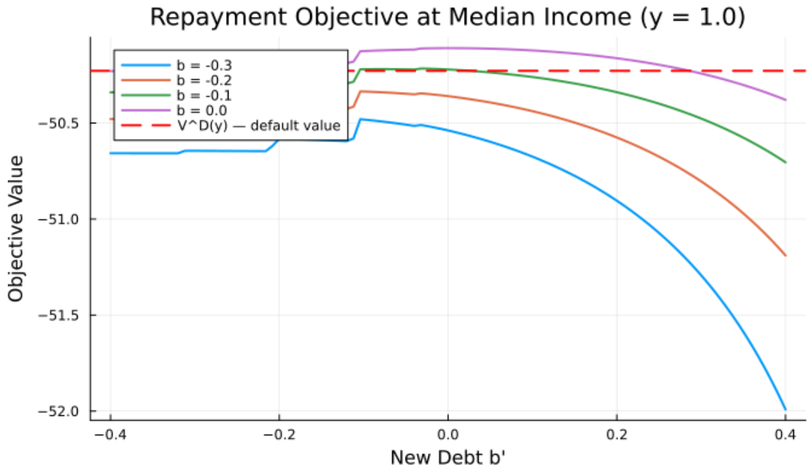
- The government's repayment problem chooses new debt b' to solve:

$$V^R(b, y) = \max_{b'} \underbrace{\left\{ u\left(y + b - q(b', y) \cdot b'\right) + \beta \, \mathbb{E}\left[V(b', y') \mid y \right] \right\}}_{\text{obj}(b';\, b, y)}$$

- Let's fix income at the **median** and plot $\text{obj}(b')$ for various current debt levels b
- This reveals the **shape** of the optimization problem and when the government **prefers default**

The Objective at Converged V and q

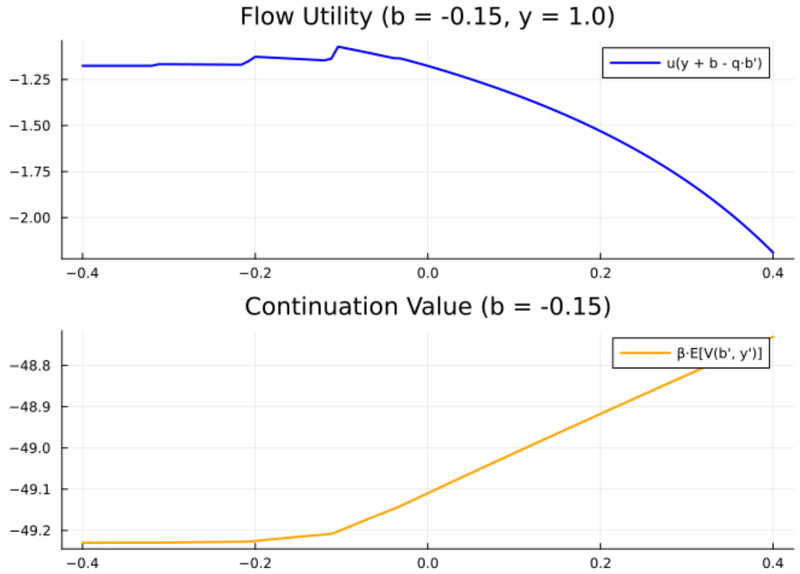
```julia
iy_mid = (m.N_y + 1) ÷ 2
y_mid = m.y_grid[iy_mid]
V_itp_plot = [linear_interpolation(m.b_grid, sol.V[:, iy]) for iy in 1:m.N_y]
q_itp_plot = [linear_interpolation(m.b_grid, sol.q[:, iy]) for iy in 1:m.N_y]

b_show = [-0.3, -0.2, -0.1, 0.0]
plot(legend=:topleft, size=(700, 400))
for bv in b_show
    obj = [let c = y_mid + bv - q_itp_plot[iy_mid](bp) * bp
           c > 0 ? u(c, m.σ) + m.β * sum(m.P_y[iy_mid, j] * V_itp_plot[j](bp)
                    for j in 1:m.N_y) : NaN
           end for bp in m.b_grid]
    plot!(m.b_grid, obj, linewidth=2, label="b = $bv")
end
hline!([sol.VD[iy_mid]], color=:red, linestyle=:dash, linewidth=2,
       label="V^D(y) — default value")
xlabel!("New Debt b'")
ylabel!("Objective Value")
```



- Each curve peaks at the **optimal** b' for that debt level
- The red dashed line is $V^D(y)$: if the best repayment value lies below it, the government defaults

Decomposing Flow Utility vs. Continuation

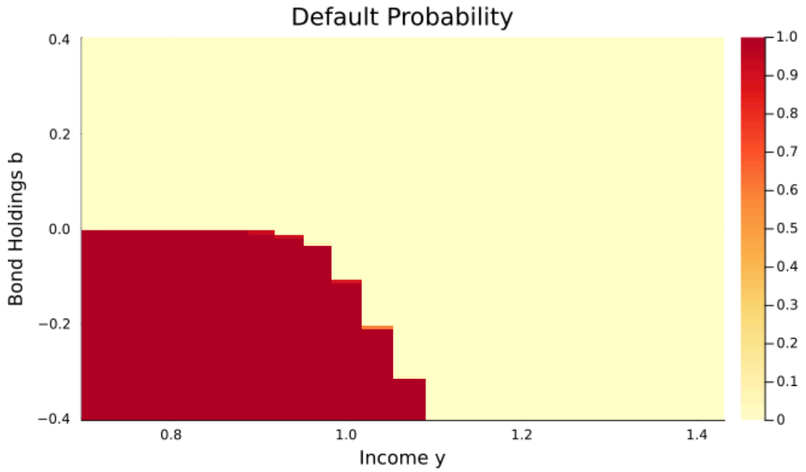
```julia
bv = -0.15                                          # moderate debt
flow = [let c = y_mid + bv - q_itp_plot[iy_mid](bp) * bp
        c > 0 ? u(c, m.σ) : NaN end for bp in m.b_grid]
cont = [m.β * sum(m.P_y[iy_mid, j] * V_itp_plot[j](bp)
        for j in 1:m.N_y) for bp in m.b_grid]
p1 = plot(m.b_grid, flow, linewidth=2, color=:blue,
          label="u(y + b - q·b')", legend=:topright, size=(700, 250),
          title="Flow Utility (b = $bv, y = $(round(y_mid, digits=3)))")
p2 = plot(m.b_grid, cont, linewidth=2, color=:orange,
          label="β·E[V(b', y')]", legend=:topright, size=(700, 250),
          title="Continuation Value (b = $bv)")
plot(p1, p2, layout=(2, 1), size=(700, 500))
```



- **Flow utility** peaks at moderate borrowing: past a point, more debt drives down q faster than it raises revenue (the sovereign "Laffer curve")
- **Continuation value** falls steadily with borrowing: more debt today $\longrightarrow$ worse starting point tomorrow

Default Probability (Smooth)

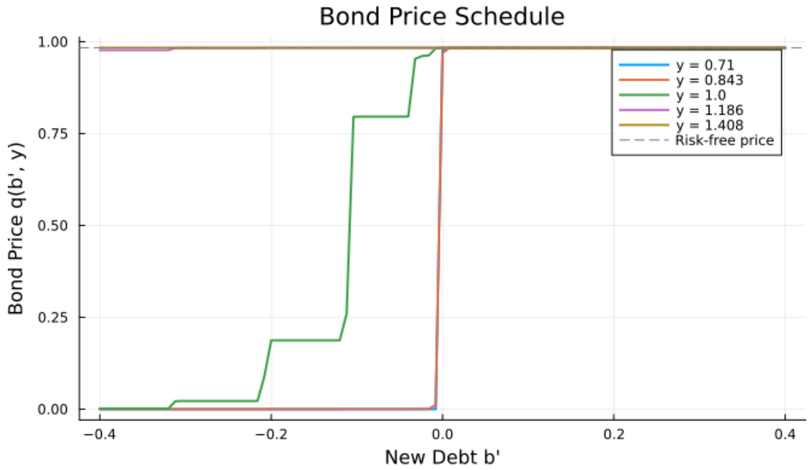
```
heatmap(m.y_grid, m.b_grid, sol.def_prob,
        xlabel="Income y", ylabel="Bond Holdings b",
        title="Default Probability",
        color=:YlOrRd, clims=(0, 1), size=(700, 400))
```



- Default probability transitions **smoothly** from 0 to 1 (thanks to taste shocks)
- The gradient traces out the **default boundary** — no sharp jump

Bond Price Schedule

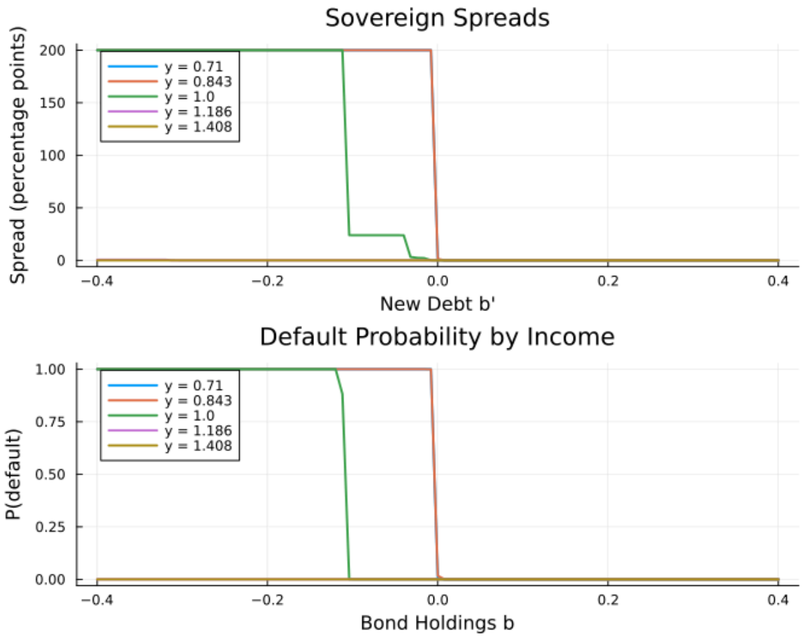
```
iy_set = round.(Int, LinRange(1, m.N_y, 5))
plot(legend=:topright, size=(700, 400))
for iy in iy_set
    plot!(m.b_grid, sol.q[:, iy], linewidth=2,
          label="y = $(round(m.y_grid[iy], digits=3))")
end
hline!([1 / (1 + m.r)], color=:gray, linestyle=:dash, label="Risk-free price")
xlabel!("New Debt b'")
ylabel!("Bond Price q(b', y)")
title!("Bond Price Schedule")
```



- Bond prices **fall** as the government borrows more — lenders demand compensation for higher default risk
- Higher income → higher bond prices (lower default risk)

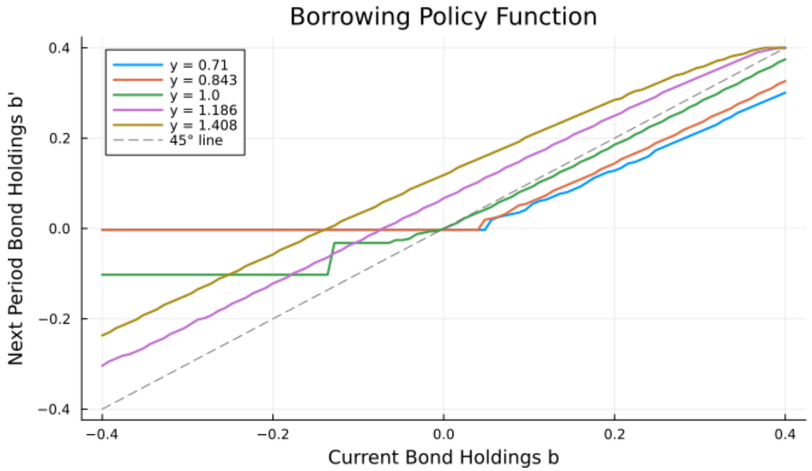
Sovereign Spreads

```julia
iy_set = round.(Int, LinRange(1, m.N_y, 5))
p1 = plot(legend=:topleft, size=(700, 300), title="Sovereign Spreads")
for iy in iy_set
    spread = (1 ./ max.(sol.q[:, iy], 1e-8)) .- (1 + m.r)
    spread = clamp.(spread, 0, 2.0)                          # cap at 200%
    plot!(p1, m.b_grid, 100 .* spread, linewidth=2,
          label="y = $(round(m.y_grid[iy], digits=3))")
end
xlabel!(p1, "New Debt b'")
ylabel!(p1, "Spread (percentage points)")

p2 = plot(legend=:topleft, size=(700, 300), title="Default Probability by Income")
for iy in iy_set
    plot!(p2, m.b_grid, sol.def_prob[:, iy], linewidth=2,
          label="y = $(round(m.y_grid[iy], digits=3))")
end
xlabel!(p2, "Bond Holdings b")
ylabel!(p2, "P(default)")
```



Borrowing Policy

```
iy_set = round.(Int, LinRange(1, m.N_y, 5))
plot(legend=:topleft, size=(700, 400))
for iy in iy_set
    plot!(m.b_grid, sol.bp[:, iy], linewidth=2,
          label="y = $(round(m.y_grid[iy], digits=3))")
end
plot!(m.b_grid, m.b_grid, color=:gray, linestyle=:dash, label="45° line")
xlabel!("Current Bond Holdings b")
ylabel!("Next Period Bond Holdings b'")
title!("Borrowing Policy Function")
```



- Below the 45° line: the government is **accumulating debt** ($b' < b$)
- Above the 45° line: the government is **saving** or **reducing debt**

Simulation

Simulating the Economy

The Simulation Function

```
function simulateSovDefault(m, sol, T; seed=42)
    # Initialize at zero debt and median income
    # Each period: compute smooth default probability, draw Bernoulli shock
    # If not in default: use interpolated policy bp_itp for borrowing
    # If in default: consume h(y), try to regain access
    # Return named tuple
end
```

The Simulation Function

```
function simulateSovDefault(m, sol, T; seed=42)
    Random.seed!(seed)
    (; r, λ, τ, ŷ, σ, y_grid, P_y, b_grid, N_y) = m
    b_sim = zeros(T);  y_sim = zeros(T);  c_sim = zeros(T)
    spread_sim = zeros(T);  default_sim = zeros(Int, T)
    b = 0.0                                  # start at zero debt
    iy = (N_y + 1) ÷ 2                       # start at median income
    in_default = false
    for t in 1:T
        y = y_grid[iy]
        b_sim[t] = b;  y_sim[t] = y
        if in_default
            c_sim[t] = h(y, ŷ)
            default_sim[t] = 1
            spread_sim[t] = NaN
            in_default = rand() > λ          # try to regain access
            if !in_default;  b = 0.0;  end
        else
```

```
simulateSovDefault (generic function with 1 method)
```

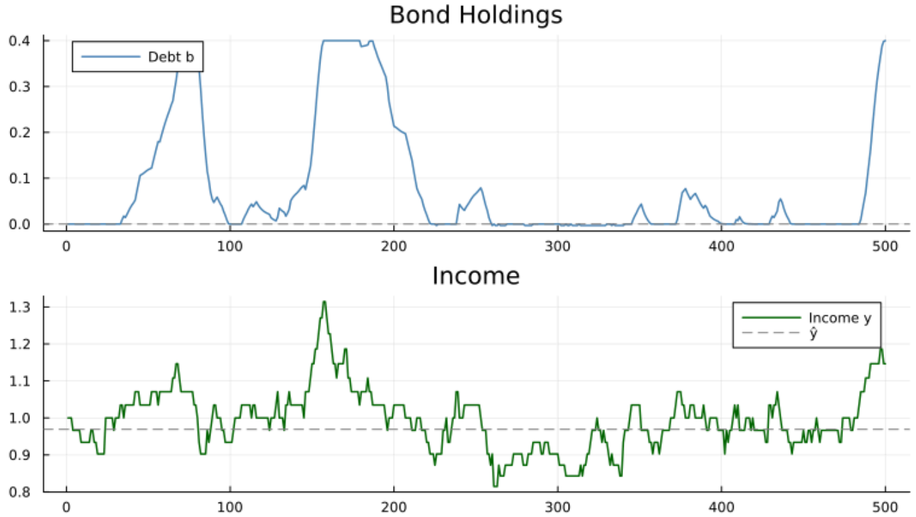
Running the Simulation

```
1  sim = simulateSovDefault(m, sol, 500)
2  println("Default rate: ", round(mean(sim.default), digits=3))
```

```
Default rate: 0.08
```

Debt and Income Dynamics

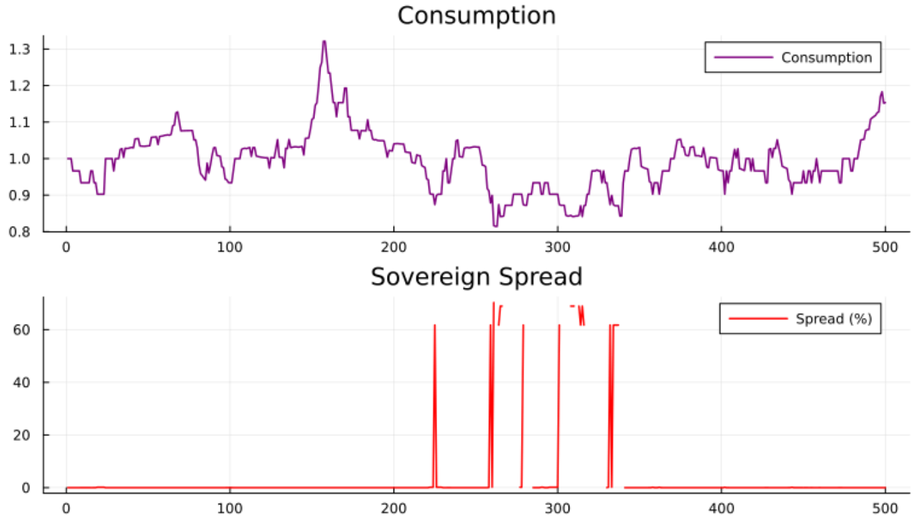
```
p1 = plot(sim.b, linewidth=1.5, color=:steelblue, label="Debt b",
          title="Bond Holdings", size=(800, 200))
hline!(p1, [0.0], color=:gray, linestyle=:dash, label="")
p2 = plot(sim.y, linewidth=1.5, color=:darkgreen, label="Income y",
          title="Income", size=(800, 200))
hline!(p2, [m.ŷ], color=:gray, linestyle=:dash, label="ŷ")
plot(p1, p2, layout=(2, 1), size=(800, 450))
```



- The government borrows during normal times and defaults when income drops sharply
- After default, debt resets to zero upon regaining market access

Consumption and Spreads

```
p1 = plot(sim.c, linewidth=1.5, color=:purple, label="Consumption",
          title="Consumption", size=(800, 200))
p2 = plot(100 .* sim.spread, linewidth=1.5, color=:red, label="Spread (%)",
          title="Sovereign Spread", size=(800, 200))
plot(p1, p2, layout=(2, 1), size=(800, 450))
```



- Consumption drops sharply at default (output cost)
- Spreads spike before default events — lenders anticipate the crisis

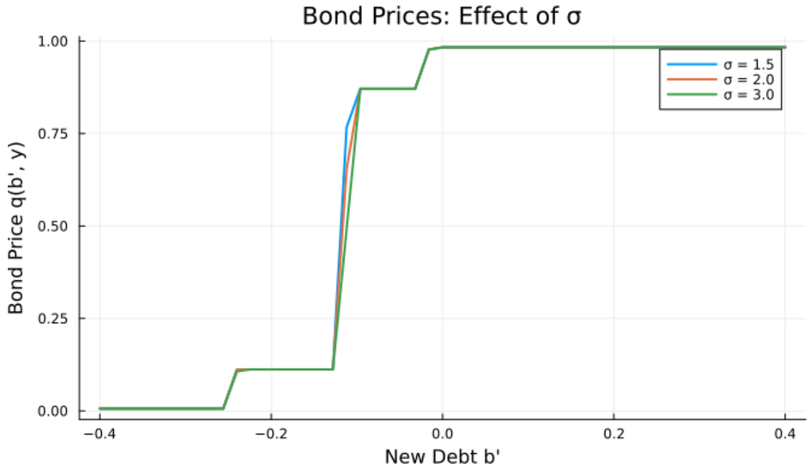
Default Episodes

```
Default episodes at periods: [262, 267, 280, 302, 306]
Total defaults in 500 periods: 8
```

Comparative Statics

Effect of Risk Aversion (σ)

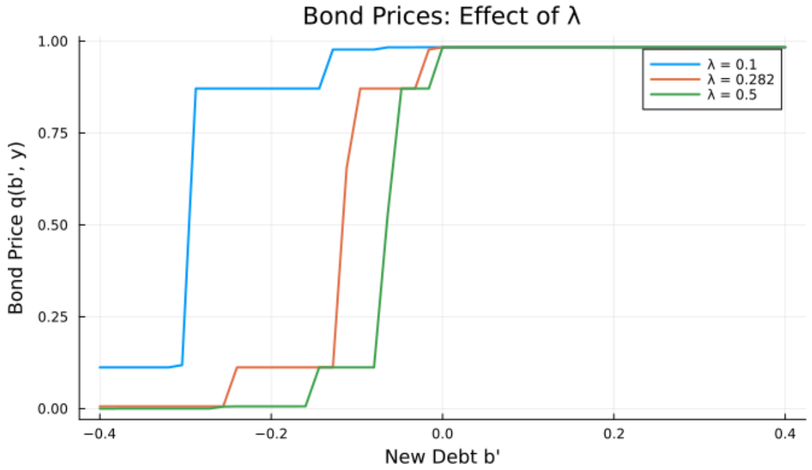
```julia
plot(legend=:topright, size=(700, 400), title="Bond Prices: Effect of σ")
for σ_val in [1.5, 2.0, 3.0]
    m_trial = setup!(SovDefaultModel(β=0.980, σ=σ_val, N_b=51, N_b_fine=101, N_y=11))
    sol_trial = solveSovDefault(m_trial, verbose=false)
    iy_mid = (m_trial.N_y + 1) ÷ 2
    plot!(m_trial.b_grid, sol_trial.q[:, iy_mid], linewidth=2, label="σ = $σ_val")
end
xlabel!("New Debt b'")
ylabel!("Bond Price q(b', y)")
```



- Higher risk aversion → government values consumption smoothing more → different borrowing patterns

Effect of Re-entry Probability (λ)

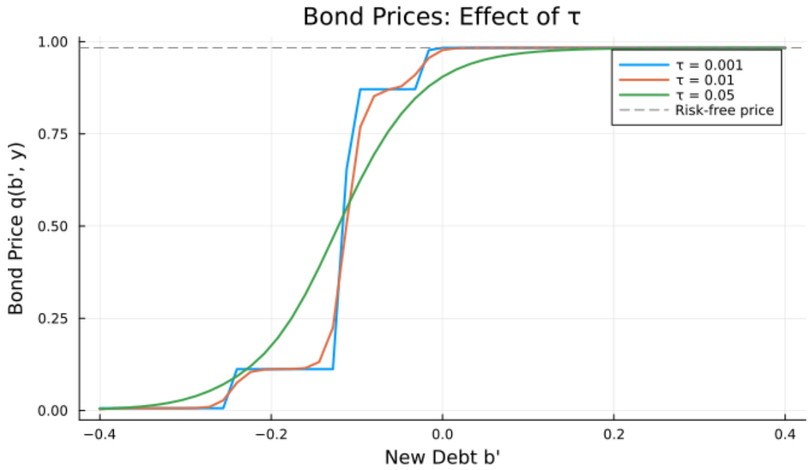
```julia
plot(legend=:topright, size=(700, 400), title="Bond Prices: Effect of λ")
for λ_val in [0.1, 0.282, 0.5]
    m_trial = setup!(SovDefaultModel(β=0.980, λ=λ_val, N_b=51, N_b_fine=101, N_y=11))
    sol_trial = solveSovDefault(m_trial, verbose=false)
    iy_mid = (m_trial.N_y + 1) ÷ 2
    plot!(m_trial.b_grid, sol_trial.q[:, iy_mid], linewidth=2, label="λ = $λ_val")
end
xlabel!("New Debt b'")
ylabel!("Bond Price q(b', y)")
```



- Higher λ → shorter exclusion from credit → default is less costly → more default → lower bond prices

Effect of Taste Shock Scale (τ)

```julia
plot(legend=:topright, size=(700, 400), title="Bond Prices: Effect of τ")
for τ_val in [0.001, 0.01, 0.05]
    m_trial = setup!(SovDefaultModel(β=0.980, τ=τ_val, N_b=51, N_b_fine=101, N_y=11))
    sol_trial = solveSovDefault(m_trial, verbose=false)
    iy_mid = (m_trial.N_y + 1) ÷ 2
    plot!(m_trial.b_grid, sol_trial.q[:, iy_mid], linewidth=2, label="τ = $τ_val")
end
hline!([1 / (1 + m.r)], color=:gray, linestyle=:dash, label="Risk-free price")
xlabel!("New Debt b'")
ylabel!("Bond Price q(b', y)")
```



- $\tau = 0.001$: our baseline — very close to the hard-max model, sharp default boundary
- $\tau = 0.01$: light smoothing — slightly blurred default boundary
- $\tau = 0.05$: heavier smoothing — default boundary blurs, useful when convergence is difficult

Summary

Key Takeaways

Concepts

- Sovereign default as a dynamic discrete choice
- Eaton-Gersovitz / Arellano model
- Endogenous bond price schedule and credit limits
- Default region: high debt, low income
- Sovereign spreads and the feedback loop
- Taste shocks: log-sum-exp and logistic probabilities (from Rust)
- Output costs and exclusion from credit markets

Julia Tools

- Rouwenhorst discretization (reused from KP)
- Mutable structs with `setup!` and `@kwdef`
- Two-dimensional value function iteration
- `smooth_max` and `default_prob` (log-sum-exp + logistic, from Rust)
- Linear interpolation for simulation policies (reused from KP)
- Bernoulli draws for stochastic default (as in Rust simulation)

Remember

The sovereign default model shows why countries face **higher borrowing costs** when they carry more debt or have weaker economies. The key mechanism is that bond prices are set by risk-neutral lenders who anticipate the government's future default decisions — creating a **feedback loop** between borrowing costs and default incentives. Unlike domestic models, there is **no enforcement mechanism** — the government repays only because default is costly (output loss + market exclusion).