Supplementary A:

```
# ============================
# ROBUST ABM–SDE HYBRID FRAMEWORK
# ----------------------------------------------------------------------
# Topic:
# Sovereign-Backed Equity Stabilization (LPI-UMKM)
# versus Conventional MSME Financing
#
# Features:
# ✓ Regime-Switching GBM
# ✓ Jump-Diffusion Crisis Shock
# ✓ Monte Carlo Framework
# ✓ Agent-Based Tax Formalization Dynamics
# ✓ Vectorized High-Performance Computation
# ✓ Percentile Confidence Intervals
# ✓ Publication-Grade Visualization
# ✓ Internal Unit Consistency
# ✓ Endogenous Growth Dynamics
#
# IMPORTANT:
# Exploratory computational stress-testing framework only.
# Not a calibrated predictive macroeconomic forecasting model.
# ============================

import numpy as np
import matplotlib.pyplot as plt

# ============================
# 1. REPRODUCIBILITY
# ============================

RNG = np.random.default_rng(seed=42)

# ============================
# 2. GLOBAL PARAMETERS
# ============================

YEARS = 5
TRADING_DAYS = 252
DT = 1 / TRADING_DAYS
N_STEPS = int(YEARS / DT)

MONTE_CARLO_RUNS = 200

# Sovereign Fund
INITIAL_FUND = 500.0  # Trillion IDR

# MSME Sector
TOTAL_SHADOW_ECONOMY = 5129.0  # Trillion IDR
N_AGENTS = 5000

# Sectoral heterogeneity
SECTOR_RISK_LEVELS = np.array([0.8, 1.0, 1.3])
SECTOR_PROBABILITIES = np.array([0.3, 0.5, 0.2])

# ============================
# 3. STORAGE MATRICES
# ============================

fund_paths_mc = np.zeros((MONTE_CARLO_RUNS, N_STEPS))
tax_paths_mc = np.zeros((MONTE_CARLO_RUNS, YEARS))

# ============================
# 4. MONTE CARLO LOOP
# ============================

for run in range(MONTE_CARLO_RUNS):

    # ====================
    # A. SOVEREIGN FUND SDE INITIALIZATION
    # ====================

    V_t = np.zeros(N_STEPS)
    V_t[0] = INITIAL_FUND

    crisis_start = int(2.0 / DT)
    crisis_end = int(3.5 / DT)

    # ====================
    # B. REGIME-SWITCHING GBM + JUMP DIFFUSION
    # ====================

    for i in range(1, N_STEPS):

        in_crisis = crisis_start <= i <= crisis_end

        # Crisis regime
        if in_crisis:
            mu_current = -0.06
            sigma_current = 0.30
        # Normal regime
        else:

            mu_current = 0.07
            sigma_current = 0.12

        # Brownian motion
        dW = RNG.normal(0, np.sqrt(DT))

        # Euler-Maruyama GBM update
        V_t[i] = (
            V_t[i - 1]
            + (mu_current * V_t[i - 1] * DT)
            + (sigma_current * V_t[i - 1] * dW)
        )

        # Jump-diffusion shock at crisis onset
        if i == crisis_start:

            jump_loss = RNG.uniform(0.12, 0.22)

            V_t[i] *= (1 - jump_loss)

        # Prevent negative values
        V_t[i] = max(V_t[i], 0)

    fund_paths_mc[run] = V_t

    # ====================
    # C. ABM TAX FORMALIZATION MODEL
    # ====================

    # Initial revenue distribution
    agent_revenue = RNG.lognormal(
        mean=np.log(TOTAL_SHADOW_ECONOMY / N_AGENTS),
        sigma=0.55,
        size=N_AGENTS
    )

    # Sectoral heterogeneity
    sector_risk = RNG.choice(
        SECTOR_RISK_LEVELS,
        size=N_AGENTS,
        p=SECTOR_PROBABILITIES
    )

    # Agent states:
    # 1 = MSME informal
    # 2 = formalized corporate
    # 3 = exited/default

    agent_status = np.ones(N_AGENTS, dtype=int)

    annual_tax_revenue = []
```

```
for yr in range(YEARS):

    # ====================
    # MACROECONOMIC CONDITION
    # ====================

    crisis_year = 2 <= yr <= 3

    if crisis_year:
        macro_growth = RNG.normal(0.93, 0.06)
    else:
        macro_growth = RNG.normal(1.05, 0.04)

    # ====================
    # MICRO SHOCKS
    # ====================

    micro_growth = RNG.lognormal(
        mean=0,
        sigma=0.10,
        size=N_AGENTS
    )

    # ====================
    # ENDOGENOUS REVENUE GROWTH
    # ====================

    agent_revenue *= (
        macro_growth
        * micro_growth
        / sector_risk
    )

    # Prevent pathological negatives
    agent_revenue = np.clip(agent_revenue, 0.01, None)

    # ====================
    # FORMALIZATION TRANSITION
    # ====================

    eligible_formalization = (
        (agent_revenue > 2.2)
        & (agent_status == 1)
    )

    formalize_transition = (
        RNG.random(N_AGENTS) < 0.18
    )

    transitioning_agents = (
        eligible_formalization
        & formalize_transition
    )

    agent_status[transitioning_agents] = 2

    # ====================
    # EXIT / FAILURE DYNAMICS
    # ====================

    distress_threshold = np.percentile(
        agent_revenue,
        12
    )

    distressed_agents = (
        agent_revenue < distress_threshold
    )

    exit_probability = (
        0.08 if crisis_year else 0.03
    )

    exiting_agents = (
        distressed_agents
        & (RNG.random(N_AGENTS) < exit_probability)
    )

    agent_status[exiting_agents] = 3

    # ====================
    # RECOVERY / RE-ENTRY
    # ====================

    recovery_probability = (
        0.05 if crisis_year else 0.10
    )

    recovering_agents = (
        (agent_status == 3)
        & (RNG.random(N_AGENTS) < recovery_probability)
    )

    agent_status[recovering_agents] = 1

    # ====================
    # TAX COLLECTION
    # ====================

    tax_micro = (
        agent_revenue
        * 0.005
        * (agent_status == 1)
    )

    tax_corporate = (
        agent_revenue
        * 0.22
        * (agent_status == 2)
    )

    total_tax = np.sum(
        tax_micro + tax_corporate
    )

    annual_tax_revenue.append(total_tax)

  tax_paths_mc[run] = annual_tax_revenue

# ============================
# 5. PERCENTILE CONFIDENCE INTERVALS
# ============================

def percentile_ci(matrix):

  mean = np.mean(matrix, axis=0)
  lower = np.percentile(matrix, 2.5, axis=0)
  upper = np.percentile(matrix, 97.5, axis=0)

  return mean, lower, upper

# Sovereign fund statistics
mean_fund, lower_fund, upper_fund = percentile_ci(fund_paths_mc)

# Tax statistics
mean_tax, lower_tax, upper_tax = percentile_ci(tax_paths_mc)

# ============================
# 6. VISUALIZATION
# ============================

fig, ax = plt.subplots(1, 2, figsize=(16, 6))

# ====================
# PANEL A — SOVEREIGN FUND TRAJECTORY
# ====================

time_continuous = np.linspace(0, YEARS, N_STEPS)
```

```
ax[0].plot(
    time_continuous,
    mean_fund,
    linewidth=2.5,
    label='Mean Sovereign Fund Path'
)

ax[0].fill_between(
    time_continuous,
    lower_fund,
    upper_fund,
    alpha=0.20,
    label='95% Percentile Band'
)

ax[0].axvspan(
    2.0,
    3.5,
    alpha=0.15,
    label='Systemic Crisis Window'
)

ax[0].axhline(
    INITIAL_FUND,
    linestyle='--',
    linewidth=1.6,
    label='Initial Baseline'
)

ax[0].set_title(
    'A. Sovereign Fund Sustainability\n'
    '(Regime-Switching GBM + Jump Diffusion)',
    fontsize=12,
    fontweight='bold'
)

ax[0].set_xlabel('Years')
ax[0].set_ylabel('Fund Value (Trillion IDR)')
ax[0].grid(True, linestyle='--', alpha=0.6)
ax[0].legend()

# =====================
# PANEL B — TAX FORMALIZATION DYNAMICS
# =====================

year_axis = np.arange(1, YEARS + 1)

ax[1].plot(
    year_axis,
    mean_tax,
    linewidth=2.5,
    label='Mean Annual Tax Revenue'
)

ax[1].fill_between(
    year_axis,
    lower_tax,
    upper_tax,
    alpha=0.20,
    label='95% Percentile Band'
)

ax[1].set_title(
    'B. ABM Tax Formalization Dynamics',
    fontsize=12,
    fontweight='bold'
)

ax[1].set_xlabel('Simulation Year')
ax[1].set_ylabel('Annual Tax Revenue (Trillion IDR)')
ax[1].grid(True, linestyle='--', alpha=0.6)
ax[1].legend()

# =====================
# GLOBAL TITLE
# =====================

plt.suptitle(
    'Robust ABM–SDE Hybrid Stress Testing Framework:\n'
    'LPI-UMKM Sovereign Stabilization Dynamics',
    fontsize=14,
    fontweight='bold'
)

plt.tight_layout()
plt.show()

# ============================
# 7. SUMMARY STATISTICS
# ============================

print("\n============")
print("FINAL MONTE CARLO SUMMARY")
print("============")

print(
    f"\nFinal Mean Sovereign Fund Value: "
    f"{mean_fund[-1]:.2f} Trillion IDR"
)

print(
    f"95% Percentile Interval: "
    f"[{lower_fund[-1]:.2f}, {upper_fund[-1]:.2f}]"
)

print(
    f"\nFinal Mean Annual Tax Revenue: "
    f"{mean_tax[-1]:.2f} Trillion IDR"
)

print(
    f"95% Percentile Interval: "
    f"[{lower_tax[-1]:.2f}, {upper_tax[-1]:.2f}]"
)

print("\n============")
print("MODEL NOTICE")
print("============")
print("This framework is exploratory and intended")
print("for computational stress-testing only.")
print("It is NOT a calibrated macroeconomic")
print("forecasting model.")
print("============")
```

Supplementary B:

```
# ========================================
# ADVANCED FISCAL-STABILITY SIMULATION FRAMEWORK
# ----------------------------------------------------------------
# COMPARATIVE MSME SUPPORT SYSTEMS:
#
# 1. KUR              : Subsidized Credit
# 2. PCGS             : Partial Credit Guarantee Scheme
# 3. Direct Grants    : Non-Recoverable Fiscal Transfers
# 4. LPI-UMKM         : Sovereign Equity Participation
#
# ========================================
# SCIENTIFIC OBJECTIVE
# ========================================
#
# Simulate how a government fully deploys a fixed 10-year MSME
# intervention envelope (500T IDR) under different policy structures.
#
# The comparison focuses on:
#
# ✓ Recoverable public value
# ✓ Fiscal sustainability
# ✓ Capital recycling efficiency
# ✓ MSME survival resilience
# ✓ Crisis robustness
# ✓ Dynamic adaptive deployment
#
# ========================================
# IMPORTANT ECONOMIC INTERPRETATION
# ========================================
#
# The 500T budget is NOT idle cash.
#
# It represents:
#
# "Total deployable 10-year MSME intervention capacity."
#
# Thus:
#
# - Government progressively deploys almost all funds.
# - Systems differ by:
#     • recovery mechanisms
#     • capital recycling
#     • sovereign ownership retention
#     • contingent liabilities
#
# ========================================

import numpy as np
import matplotlib.pyplot as plt

# ========================================
# 1. REPRODUCIBILITY
# ========================================

RNG = np.random.default_rng(42)

# ========================================
# 2. GLOBAL PARAMETERS
# ========================================

YEARS = 10
MC_RUNS = 400
N_AGENTS = 5000

INITIAL_PUBLIC_COMMITMENT = 500.0   # Trillion IDR

CRISIS_START = 3
CRISIS_END = 5

SYSTEMS = [
    "KUR",
    "PCGS",
    "Grant",
    "LPI-UMKM"
]

# ========================================
# 3. SYSTEM PARAMETERS
# ========================================

PARAMS = {

    "KUR": {

        "base_default": 0.05,
        "crisis_default": 0.18,

        "recovery_rate": 0.45,

        "deployment_intensity": 1.00,

        "subsidy_cost": 0.12,

        "credit_multiplier": 1.20
    },

    "PCGS": {

        "base_default": 0.07,
        "crisis_default": 0.28,

        "recovery_rate": 0.20,

        "deployment_intensity": 1.05,

        "guarantee_cost": 0.25,

        "credit_multiplier": 1.40
    },

    "Grant": {

        "base_default": 0.09,
        "crisis_default": 0.35,

        "recovery_rate": 0.00,

        "deployment_intensity": 1.10,

        "grant_cost": 1.00,

        "demand_stimulus": 0.10
    },

    "LPI-UMKM": {

        "base_default": 0.04,
        "crisis_default": 0.12,

        "recovery_rate": 0.72,

        "deployment_intensity": 0.95,

        "ownership_share": 0.40,

        "locked_capital_ratio": 0.65,

        "buyback_probability": 0.15,

        "secondary_exit_probability": 0.08,

        "portfolio_return_normal": 0.08,

        "portfolio_return_crisis": -0.03,

        "portfolio_vol_normal": 0.10,

        "portfolio_vol_crisis": 0.22,

        "monitoring_cost": 0.05
    }
}

# ========================================
# 4. STORAGE MATRICES
# ========================================

recoverable_value = {
```

```
    s: np.zeros((MC_RUNS, YEARS + 1))
    for s in SYSTEMS
}

survival_rate = {
    s: np.zeros((MC_RUNS, YEARS + 1))
    for s in SYSTEMS
}

cumulative_deployment = {
    s: np.zeros((MC_RUNS, YEARS + 1))
    for s in SYSTEMS
}

annual_deployment = {
    s: np.zeros((MC_RUNS, YEARS + 1))
    for s in SYSTEMS
}

# ========================================
# 5. PERCENTILE CONFIDENCE INTERVAL
# ========================================

def percentile_ci(matrix):

    mean = np.mean(matrix, axis=0)

    lower = np.percentile(matrix, 2.5, axis=0)

    upper = np.percentile(matrix, 97.5, axis=0)

    return mean, lower, upper

# ========================================
# 6. MAIN MONTE CARLO SIMULATION
# ========================================

for run in range(MC_RUNS):

    # ------------------------------------------------------------
    # HETEROGENEOUS MSME ECONOMIC BASELINES
    # ------------------------------------------------------------

    base_revenue = RNG.lognormal(
        mean=4.5,
        sigma=0.65,
        size=N_AGENTS
    )

    productivity = RNG.normal(
        1.0,
        0.15,
        size=N_AGENTS
    )

    sector_risk = RNG.choice(
        [0.85, 1.0, 1.35],
        size=N_AGENTS,
        p=[0.30, 0.50, 0.20]
    )

    for system in SYSTEMS:

        params = PARAMS[system]

        # ============================
        # GOVERNMENT PROGRAM ACCOUNTING
        # ============================

        remaining_commitment = INITIAL_PUBLIC_COMMITMENT

        recoverable_public_value = INITIAL_PUBLIC_COMMITMENT

        cumulative_used = 0.0

        # ============================
        # LPI SOVEREIGN ASSET ACCOUNT
        # ============================

        sovereign_assets = (
            INITIAL_PUBLIC_COMMITMENT
            if system == "LPI-UMKM"
            else 0
        )

        # ============================
        # MSME STATES
        # ============================

        active_agents = np.ones(
            N_AGENTS,
            dtype=bool
        )

        revenue = base_revenue.copy()

        # ============================
        # INITIAL RECORDS
        # ============================

        recoverable_value[system][run, 0] = recoverable_public_value

        survival_rate[system][run, 0] = 100

        cumulative_deployment[system][run, 0] = 0

        annual_deployment[system][run, 0] = 0

        # ============================
        # YEARLY SIMULATION
        # ============================

        for year in range(1, YEARS + 1):

            # ========================
            # REGIME SWITCHING MACROECONOMY
            # ========================

            crisis = (
                CRISIS_START <= year <= CRISIS_END
            )

            if crisis:

                macro_growth = RNG.normal(0.88, 0.07)

                macro_volatility = 0.28

                crisis_multiplier = 2.2

            else:

                macro_growth = RNG.normal(1.04, 0.03)

                macro_volatility = 0.10

                crisis_multiplier = 1.0

            # ========================
            # MSME REVENUE EVOLUTION
            # ========================

            micro_shock = RNG.lognormal(
                mean=0,
                sigma=macro_volatility,
                size=N_AGENTS
            )

            revenue *= (
                macro_growth
                * micro_shock
                * productivity
                / sector_risk
            )

            revenue = np.clip(revenue, 0.01, None)

            # ========================
            # DISTRESS IDENTIFICATION
            # ========================

            distress_threshold = np.percentile(
```

```
    revenue,
    25
)

distressed = (
    revenue < distress_threshold
)

distress_ratio = np.mean(distressed)

# ========================
# DYNAMIC DEPLOYMENT POLICY
# ========================
#
# Government progressively deploys nearly all
# fiscal commitment across 10 years.
#
# Crisis → accelerated deployment
# Distress → intensified intervention
# Remaining years constrain allocation
#
# ========================

years_left = YEARS - year + 1

baseline_yearly_target = (
    remaining_commitment / years_left
)

adaptive_multiplier = (
    1
    + 0.80 * distress_ratio
    + 0.70 * (crisis_multiplier - 1)
)

yearly_deployment_target = (
    baseline_yearly_target
    * adaptive_multiplier
    * params["deployment_intensity"]
)

yearly_deployment_target = min(
    yearly_deployment_target,
    remaining_commitment
)

# ========================
# DEFAULT HAZARD
# ========================

base_default = (
    params["crisis_default"]
    if crisis
    else params["base_default"]
)

intervention_strength = (
    yearly_deployment_target
    / INITIAL_PUBLIC_COMMITMENT
)

effective_default = (
    base_default
    * (1 - 0.60 * intervention_strength)
)

effective_default += (
    0.08 * distressed
)

effective_default = np.clip(
    effective_default,
    0.01,
    0.95
)

# ========================
# DEFAULT EVENTS
# ========================

default_events = (
    active_agents
    & (
        RNG.random(N_AGENTS)
        < effective_default
    )
)

active_agents[default_events] = False

defaults_this_year = np.sum(default_events)

active_count = np.sum(active_agents)

# ========================
# SYSTEM-SPECIFIC FISCAL DYNAMICS
# ========================

recovery = 0.0

# ----------------------------------------------------
# KUR
# ----------------------------------------------------

if system == "KUR":

    subsidy_spending = (
        yearly_deployment_target
        * params["subsidy_cost"]
    )

    banking_recovery = (
        subsidy_spending
        * params["recovery_rate"]
    )

    default_losses = (
        defaults_this_year
        * 0.002
    )

    recovery = (
        banking_recovery
        - default_losses
    )

# ----------------------------------------------------
# PCGS
# ----------------------------------------------------

elif system == "PCGS":

    guarantee_claims = (
        defaults_this_year
        * 0.005
        * crisis_multiplier
    )

    guarantee_fees = (
        yearly_deployment_target
        * 0.05
    )

    recovery = (
        guarantee_fees
        - guarantee_claims
    )

# ----------------------------------------------------
# GRANTS
# ----------------------------------------------------

elif system == "Grant":

    recovery = 0.0

# ----------------------------------------------------
# LPI-UMKM
# ----------------------------------------------------

elif system == "LPI-UMKM":
```

```
# ----------------------------------------------------
# DEPLOYMENT INTO EQUITY POSITIONS
# ----------------------------------------------------

sovereign_assets += (
    yearly_deployment_target
)

# ----------------------------------------------------
# PORTFOLIO REGIME SWITCHING
# ----------------------------------------------------

if crisis:

    portfolio_return = RNG.normal(
        params["portfolio_return_crisis"],
        params["portfolio_vol_crisis"]
    )

else:

    portfolio_return = RNG.normal(
        params["portfolio_return_normal"],
        params["portfolio_vol_normal"]
    )

sovereign_assets *= (
    1 + portfolio_return
)

# ----------------------------------------------------
# IMPAIRMENT LOSSES
# ----------------------------------------------------

impairment = (
    defaults_this_year
    * 0.003
)

sovereign_assets -= impairment

# ----------------------------------------------------
# BUYBACKS
# ----------------------------------------------------

healthy_agents = (
    active_agents
    & (
        revenue
        > np.percentile(revenue, 70)
    )
)

buyback_events = (
    healthy_agents
    & (
        RNG.random(N_AGENTS)
        < params["buyback_probability"]
    )
)

n_buybacks = np.sum(
    buyback_events
)

buyback_proceeds = (
    n_buybacks
    * 0.0025
)

# ----------------------------------------------------
# SECONDARY INVESTOR EXITS
# ----------------------------------------------------

secondary_events = (
    healthy_agents
    & (
        RNG.random(N_AGENTS)
        < params["secondary_exit_probability"]
    )
)

    n_secondary = np.sum(
        secondary_events
    )

    secondary_proceeds = (
        n_secondary
        * 0.002
    )

    # ----------------------------------------------------
    # MONITORING COST
    # ----------------------------------------------------

    monitoring_cost = (
        yearly_deployment_target
        * params["monitoring_cost"]
    )

    recovery = (
        buyback_proceeds
        + secondary_proceeds
        - monitoring_cost
    )

    recoverable_public_value = (
        sovereign_assets
    )

# ========================
# GOVERNMENT ACCOUNTING IDENTITY
# ========================

cumulative_used += yearly_deployment_target

remaining_commitment -= yearly_deployment_target

remaining_commitment = max(
    remaining_commitment,
    0
)

# ========================
# RECOVERABLE PUBLIC VALUE
# ========================

if system != "LPI-UMKM":

    recoverable_public_value = max(
        (
            INITIAL_PUBLIC_COMMITMENT
            - cumulative_used
            + recovery
        ),
        0
    )

# ========================
# STORE RESULTS
# ========================

recoverable_value[system][run, year] = (
    recoverable_public_value
)

survival_rate[system][run, year] = (
    active_count / N_AGENTS
) * 100

cumulative_deployment[system][run, year] = (
    cumulative_used
)

annual_deployment[system][run, year] = (
    yearly_deployment_target
)

# ======================================
# 7. VISUALIZATION
# ======================================
```

```
fig, ax = plt.subplots(2, 2, figsize=(18, 12))

time_axis = np.arange(YEARS + 1)

colors = {
    "KUR": "blue",
    "PCGS": "red",
    "Grant": "orange",
    "LPI-UMKM": "forestgreen"
}

# ======================================
# PANEL A — RECOVERABLE PUBLIC VALUE
# ======================================

for system in SYSTEMS:

    mean, low, high = percentile_ci(
        recoverable_value[system]
    )

    ax[0,0].plot(
        time_axis,
        mean,
        label=system,
        color=colors[system],
        linewidth=2.5
    )

    ax[0,0].fill_between(
        time_axis,
        low,
        high,
        color=colors[system],
        alpha=0.12
    )

ax[0,0].axvspan(
    CRISIS_START,
    CRISIS_END,
    color='gray',
    alpha=0.15
)

ax[0,0].set_title(
    'A. Recoverable Public Fiscal Value',
    fontweight='bold'
)

ax[0,0].set_ylabel(
    'Recoverable Value (Trillion IDR)'
)

ax[0,0].grid(True, linestyle='--', alpha=0.6)

ax[0,0].legend()

# ======================================
# PANEL B — MSME SURVIVAL
# ======================================

for system in SYSTEMS:

    mean, low, high = percentile_ci(
        survival_rate[system]
    )

    ax[0,1].plot(
        time_axis,
        mean,
        label=system,
        color=colors[system],
        linewidth=2.5
    )

    ax[0,1].fill_between(
        time_axis,
        low,
        high,
        color=colors[system],
        alpha=0.10
    )

ax[0,1].axvspan(
    CRISIS_START,
    CRISIS_END,
    color='gray',
    alpha=0.15
)

ax[0,1].set_title(
    'B. MSME Survival Rate',
    fontweight='bold'
)

ax[0,1].set_ylabel(
    'Survival Rate (%)'
)

ax[0,1].set_ylim(0, 105)

ax[0,1].grid(True, linestyle='--', alpha=0.6)

ax[0,1].legend()

# ======================================
# PANEL C — CUMULATIVE DEPLOYMENT
# ======================================

for system in SYSTEMS:

    mean, low, high = percentile_ci(
        cumulative_deployment[system]
    )

    ax[1,0].plot(
        time_axis,
        mean,
        label=system,
        color=colors[system],
        linewidth=2.5
    )

    ax[1,0].fill_between(
        time_axis,
        low,
        high,
        color=colors[system],
        alpha=0.10
    )

ax[1,0].set_title(
    'C. Cumulative Government MSME Deployment',
    fontweight='bold'
)

ax[1,0].set_ylabel(
    'Cumulative Deployment (Trillion IDR)'
)

ax[1,0].grid(True, linestyle='--', alpha=0.6)

ax[1,0].legend()

# ======================================
# PANEL D — YEARLY DEPLOYMENT
# ======================================

for system in SYSTEMS:

    mean, low, high = percentile_ci(
        annual_deployment[system]
    )

    ax[1,1].plot(
        time_axis,
        mean,
        label=system,
        color=colors[system],
        linewidth=2.5
    )
```

```
    ax[1,1].fill_between(
        time_axis,
        low,
        high,
        color=colors[system],
        alpha=0.10
    )

ax[1,1].axvspan(
    CRISIS_START,
    CRISIS_END,
    color='gray',
    alpha=0.15
)

ax[1,1].set_title(
    'D. Dynamic Annual Fiscal Deployment',
    fontweight='bold'
)

ax[1,1].set_ylabel(
    'Deployment (Trillion IDR / Year)'
)

ax[1,1].grid(True, linestyle='--', alpha=0.6)

ax[1,1].legend()

# ========================================
# GLOBAL TITLE
# ========================================

plt.suptitle(
    'Adaptive Fiscal Deployment and Recoverability of MSME Support Systems\n'
    'Under Regime-Switching Macroeconomic Stress',
    fontsize=16,
    fontweight='bold'
)

plt.tight_layout()

plt.show()

# ========================================
# 8. FINAL SUMMARY
# ========================================

print("\n========================")
print("FINAL SIMULATION SUMMARY")
print("========================")

for system in SYSTEMS:

    final_value = np.mean(
        recoverable_value[system][:, -1]
    )

    final_survival = np.mean(
        survival_rate[system][:, -1]
    )

    final_deployment = np.mean(
        cumulative_deployment[system][:, -1]
    )

    print(f"\nSYSTEM: {system}")

    print(
        f"Total Fiscal Deployment: "
        f"{final_deployment:.2f}T IDR"
    )

    print(
        f"Recoverable Public Value: "
        f"{final_value:.2f}T IDR"
    )

    print(
        f"Final MSME Survival: "
        f"{final_survival:.2f}%"
    )

print("\n========================")
print("MODEL CHARACTERISTICS")
print("========================")
print("✓ Full 10-year deployment framework")
print("✓ Adaptive crisis-response allocation")
print("✓ Regime-switching macroeconomy")
print("✓ Endogenous fiscal recycling")
print("✓ Sovereign equity recoverability")
print("✓ Buyback and secondary exit dynamics")
print("✓ Monte Carlo uncertainty propagation")
print("========================")
```

Supplementary C:

```
# ROBUST MONTE CARLO ABM–SDE–MARKOV SIMULATION FRAMEWORK
# ------------------------------------------------------------------
#
# Features:
# ✓ Monte Carlo Simulation
# ✓ Agent-Based Modeling (ABM)
# ✓ Regime-Switching Stochastic Differential Equation (SDE)
# ✓ Markov State Transitions
# ✓ Sectoral Heterogeneity
# ✓ Recovery & Restructuring Dynamics
# ✓ Sovereign Fund Asset Dynamics
# ✓ Percentile Confidence Intervals
# ✓ Vectorized High-Performance Computation
# ✓ Publication-Grade Visualization
#
# Intended Use:
# Exploratory computational stress-testing framework only.
# Not intended as predictive macroeconomic forecasting.
# ====================

import numpy as np
import matplotlib.pyplot as plt

# ====================
# 1. REPRODUCIBILITY
# ====================

RNG = np.random.default_rng(seed=42)

# ====================
# 2. GLOBAL PARAMETERS
# ====================

NUM_AGENTS = 5000
YEARS = 10
MONTE_CARLO_RUNS = 200

INITIAL_SOVEREIGN_FUND = 500.0  # Trillion IDR

# MSME States
STATE_PERFORMING = 0
STATE_DISTRESSED = 1
STATE_RESTRUCTURED = 2
STATE_DEFAULT = 3

# ====================
# 3. STORAGE MATRICES
# ====================

surv_kur_mc = np.zeros((MONTE_CARLO_RUNS, YEARS))
 surv_lpi_mc = np.zeros((MONTE_CARLO_RUNS, YEARS))

fund_lpi_mc = np.zeros((MONTE_CARLO_RUNS, YEARS))

# ====================
# 4. SECTORAL HETEROGENEITY
#
------------------------------------------------------------------
# Sector Risk Multipliers:
# Low Risk     = 0.8
# Medium Risk  = 1.0
# High Risk    = 1.35
# ====================

 sector_risk_levels = np.array([0.8, 1.0, 1.35])
sector_probabilities = np.array([0.3, 0.5, 0.2])

# ====================
# 5. MONTE CARLO
SIMULATION #
====================

for run in range(MONTE_CARLO_RUNS):

# ------------------------------------------------------------------
# INITIAL MSME REVENUE DISTRIBUTION
# ------------------------------------------------------------------
# Lognormal approximates fat-tailed MSME income dispersion
# ------------------------------------------------------------------

base_revenue_kur = RNG.lognormal(
   mean=4.6,
   sigma=0.55,
   size=NUM_AGENTS
)

base_revenue_lpi = RNG.lognormal(
   mean=4.6,
   sigma=0.55,
   size=NUM_AGENTS
)

# Operating costs
operating_cost = RNG.normal(
   loc=38,
   scale=6,
   size=NUM_AGENTS
)

operating_cost = np.clip(operating_cost, 5, None)

# Sector heterogeneity
sector_risk = RNG.choice(
   sector_risk_levels,
   size=NUM_AGENTS,
   p=sector_probabilities
)

# ------------------------------------------------------------------
# FINANCING PARAMETERS
# ------------------------------------------------------------------

kur_fixed_installment = 28

lpi_profit_share_ratio = 0.28

# Dynamic monitoring/restructuring cost
base_monitoring_cost = 4.0

# Initial sovereign fund
sovereign_fund = INITIAL_SOVEREIGN_FUND

# ------------------------------------------------------------------
# INITIAL MSME STATES
# ------------------------------------------------------------------

state_kur = np.full(NUM_AGENTS, STATE_PERFORMING)
state_lpi = np.full(NUM_AGENTS, STATE_PERFORMING)

# ========================
# YEARLY SIMULATION LOOP
# ========================

for year in range(YEARS):

   # ====================
   # A. STOCHASTIC MACROECONOMIC REGIME SWITCHING
   # ====================

   crisis_occurs = RNG.random() < 0.15

   if crisis_occurs:

      # Crisis Regime
      macro_factor = RNG.uniform(0.45, 0.70)

      mu_fund = -0.10
```

```
    sigma_fund = 0.25

    restructuring_probability = 0.45

else:

    # Normal Regime
    macro_factor = RNG.normal(1.02, 0.04)

    mu_fund = 0.06
    sigma_fund = 0.12

    restructuring_probability = 0.18

# =====================
# B. SOVEREIGN FUND STOCHASTIC DYNAMICS (SDE)
# =====================

dW = RNG.normal(0, 1)

market_return = (
    mu_fund
    + sigma_fund * dW
)

sovereign_fund *= (1 + market_return)

sovereign_fund = max(sovereign_fund, 0)

# =====================
# C. MSME MICRO SHOCKS
# =====================

micro_shock_kur = RNG.lognormal(
    mean=0,
    sigma=0.12,
    size=NUM_AGENTS
)

micro_shock_lpi = RNG.lognormal(
    mean=0,
    sigma=0.12,
    size=NUM_AGENTS
)

# =====================
# D. ENDOGENOUS BUSINESS GROWTH
# =====================

endogenous_growth = RNG.normal(
    loc=1.03,
    scale=0.04,
    size=NUM_AGENTS
)

base_revenue_kur *= endogenous_growth
base_revenue_lpi *= endogenous_growth

# =====================
# E. CURRENT REVENUE REALIZATION
# =====================

curr_rev_kur = (
    base_revenue_kur
    * macro_factor
    * micro_shock_kur
    / sector_risk
)

curr_rev_lpi = (
    base_revenue_lpi
    * macro_factor
    * micro_shock_lpi
    / sector_risk
)

# =====================
# F. KUR SYSTEM DYNAMICS
# =====================

active_kur = state_kur != STATE_DEFAULT

kur_failure_threshold = (
    kur_fixed_installment
    + operating_cost
)

kur_distressed = (
    active_kur
    & (curr_rev_kur < kur_failure_threshold)
)

# Transition Performing -> Distressed
state_kur[kur_distressed] = STATE_DISTRESSED

# Distressed -> Default
default_transition_kur = (
    (state_kur == STATE_DISTRESSED)
    & (RNG.random(NUM_AGENTS) < 0.45)
)

state_kur[default_transition_kur] = STATE_DEFAULT

# Distressed -> Recovery
recovery_transition_kur = (
    (state_kur == STATE_DISTRESSED)
    & (RNG.random(NUM_AGENTS) < 0.18)
)

state_kur[recovery_transition_kur] = STATE_RESTRUCTURED

# Restructured -> Performing
normalize_transition_kur = (
    (state_kur == STATE_RESTRUCTURED)
    & (RNG.random(NUM_AGENTS) < 0.35)
)

state_kur[normalize_transition_kur] = STATE_PERFORMING

# =====================
# G. LPI SYSTEM DYNAMICS
# =====================

active_lpi = state_lpi != STATE_DEFAULT

# Dynamic burden contracts during downturns
dynamic_profit_share = (
    curr_rev_lpi
    * lpi_profit_share_ratio
)

# Monitoring cost rises during crises
monitoring_cost = (
    base_monitoring_cost
    * (1.5 if crisis_occurs else 1.0)
)

lpi_failure_threshold = (
    operating_cost
    + monitoring_cost
)

lpi_distressed = (
    active_lpi
    & (curr_rev_lpi < lpi_failure_threshold)
)

state_lpi[lpi_distressed] = STATE_DISTRESSED
```

```
    # ====================
    # H. SOVEREIGN FUND IMPAIRMENT & RETURN LOGIC
    # ====================

    distressed_count = np.sum(state_lpi == STATE_DISTRESSED)

    restructuring_cost = distressed_count * 0.002

    sovereign_fund -= restructuring_cost

    # Profit-sharing yield from healthy firms
    performing_lpi = (
        state_lpi == STATE_PERFORMING
    )

    sovereign_yield = np.sum(
        dynamic_profit_share[performing_lpi]
    ) * 0.00008

    sovereign_fund += sovereign_yield

    # Default transitions
    default_transition_lpi = (
        (state_lpi == STATE_DISTRESSED)
        & (
            RNG.random(NUM_AGENTS)
            < (0.20 + 0.10 * crisis_occurs)
        )
    )

    state_lpi[default_transition_lpi] = STATE_DEFAULT

    # Recovery transitions
    recovery_transition_lpi = (
        (state_lpi == STATE_DISTRESSED)
        & (
            RNG.random(NUM_AGENTS)
            < restructuring_probability
        )
    )

    state_lpi[recovery_transition_lpi] = STATE_RESTRUCTURED

    # Restructured normalization
    normalize_transition_lpi = (
        (state_lpi == STATE_RESTRUCTURED)
        & (RNG.random(NUM_AGENTS) < 0.45)
    )

    state_lpi[normalize_transition_lpi] = STATE_PERFORMING

    sovereign_fund = max(sovereign_fund, 0)

    # ====================
    # I. RECORD RESULTS
    # ====================

    surv_kur_mc[run, year] = (
        np.mean(state_kur != STATE_DEFAULT)
        * 100
    )

    surv_lpi_mc[run, year] = (
        np.mean(state_lpi != STATE_DEFAULT)
        * 100
    )

    fund_lpi_mc[run, year] = sovereign_fund

# ====================
# 6. PERCENTILE CONFIDENCE INTERVALS
# ====================

def percentile_ci(matrix):

    mean = np.mean(matrix, axis=0)

    lower = np.percentile(matrix, 2.5, axis=0)

    upper = np.percentile(matrix, 97.5, axis=0)

    return mean, lower, upper

mean_kur, lower_kur, upper_kur = percentile_ci(surv_kur_mc)

mean_lpi, lower_lpi, upper_lpi = percentile_ci(surv_lpi_mc)

mean_fund, lower_fund, upper_fund = percentile_ci(fund_lpi_mc)

# ====================
# 7. VISUALIZATION
# ====================

years_axis = np.arange(1, YEARS + 1)

fig, (ax1, ax2) = plt.subplots(
    1,
    2,
    figsize=(15, 6)
)

# ----------------------------------------------------------------------
# PANEL A: MSME SURVIVAL DYNAMICS
# ----------------------------------------------------------------------

ax1.plot(
    years_axis,
    mean_kur,
    linewidth=2.2,
    label='Conventional Debt (KUR)'
)

ax1.fill_between(
    years_axis,
    lower_kur,
    upper_kur,
    alpha=0.20
)

ax1.plot(
    years_axis,
    mean_lpi,
    linewidth=2.6,
    label='Sovereign Equity (LPI-UMKM)'
)

ax1.fill_between(
    years_axis,
    lower_lpi,
    upper_lpi,
    alpha=0.20
)

ax1.set_title(
    'A. MSME Survival Dynamics\n(95% Percentile Confidence Bands)',
    fontsize=12,
    fontweight='bold'
)

ax1.set_xlabel('Simulation Year')

ax1.set_ylabel('Survival Rate (%)')

ax1.set_ylim(0, 105)

ax1.grid(True, linestyle='--', alpha=0.6)
```

```
ax1.legend()

# ------------------------------------------------------------------
# PANEL B: SOVEREIGN FUND TRAJECTORY
# ------------------------------------------------------------------

ax2.plot(
    years_axis,
    mean_fund,
    linewidth=2.5,
    label='LPI Sovereign Fund'
)

ax2.fill_between(
    years_axis,
    lower_fund,
    upper_fund,
    alpha=0.20
)

ax2.axhline(
    INITIAL_SOVEREIGN_FUND,
    linestyle='--',
    linewidth=1.8,
    label='Initial Baseline'
)

ax2.set_title(
    'B. Sovereign Fund Trajectory\n(Regime-Switching SDE)',
    fontsize=12,
    fontweight='bold'
)

ax2.set_xlabel('Simulation Year')

ax2.set_ylabel('Fund Value (Trillion IDR)')

ax2.grid(True, linestyle='--', alpha=0.6)

ax2.legend()

# ------------------------------------------------------------------
# GLOBAL TITLE
# ------------------------------------------------------------------

plt.suptitle(
    'Robust Monte Carlo ABM–SDE–Markov Stress Test:\n'
    'Debt-Based vs Sovereign Equity MSME Financing Systems',
    fontsize=14,
    fontweight='bold'
)

plt.tight_layout()

plt.show()

# ======================
# 8. SUMMARY STATISTICS
# ======================

print("\n======================")
print("FINAL SIMULATION SUMMARY")
print("======================")

print(f"\nFinal Mean Survival Rate (KUR): {mean_kur[-1]:.2f}%")
print(f"Final Mean Survival Rate (LPI): {mean_lpi[-1]:.2f}%")

print(f"\nFinal Mean Sovereign Fund Value: "
      f"{mean_fund[-1]:.2f} Trillion IDR")

survival_advantage = mean_lpi[-1] - mean_kur[-1]

print(f"\nLPI Survival Advantage: "
      f"{survival_advantage:.2f} percentage points")

print("\n======================")
print("NOTE:")
print("This framework is exploratory and intended for")
print("computational stress-testing only.")
print("It is NOT a calibrated macroeconomic forecasting model.")
print("======================")
```

Supplementary D

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gridspec
from matplotlib.lines import Line2D
import matplotlib.patches as mpatches
from matplotlib.patches import FancyBboxPatch

# ==============================
# 1.  REPRODUCIBILITY
# ==============================

RNG = np.random.default_rng(42)

# ==============================
# 2.  GLOBAL PARAMETERS
# ==============================

YEARS        = 10
MC_RUNS      = 250
N_AGENTS     = 5000
INIT_CAPITAL = 500.0   # Trillion IDR

CRISIS_START = 2
CRISIS_END   = 3

SYSTEMS = ["KUR", "PCGS", "Grant", "LPI-UMKM"]

# MARKOV STATES
S_PERFORMING   = 0
S_DISTRESSED   = 1
S_RESTRUCTURED = 2
S_DEFAULT      = 3
S_EXITED       = 4
S_INDEPENDENT  = 5

# ==============================
# 3.  LPI-UMKM STRUCTURAL MECHANISM PARAMETERS
# ==============================

ETA                  = 0.82
HAZARD_REDUCTION_FACTOR = 0.38
ASSET_BACKING_RATIO    = 0.68
ASSET_RECOVERY_RATE    = 0.62
LPI_RECOVERY_STRENGTH          =     ASSET_BACKING_RATIO     *
      ASSET_RECOVERY_RATE * ETA

FIRST_LOSS_TRANCHE     = 0.15
BUYBACK_PROB           = 0.06
BUYBACK_MARKUP         = 1.12
THIRD_PARTY_YEAR       = 4
THIRD_PARTY_SHARE      = 0.25
TAX_CLIFF_EXIT_PROB    = 0.18
REVENUE_GRADUATION_MULTIPLE = 2.2

# ==============================
# 4.  MORAL HAZARD PROFILES
# ==============================

MH = {
   "KUR": {
      "capital_diversion_rate": 0.20,
      "bank_risk_inflation": 1.28,
      "productivity_discount": 0.72,
      "fiscal_leakage_multiplier": 1.32,
      "recovery_discount": 0.78,
   },
   "PCGS": {
      "capital_diversion_rate": 0.15,
      "bank_risk_inflation": 1.42,
      "productivity_discount": 0.74,
      "fiscal_leakage_multiplier": 1.38,
      "recovery_discount": 0.70,
   },
   "Grant": {
      "capital_diversion_rate": 0.38,
      "bank_risk_inflation": 1.00,
      "productivity_discount": 0.58,
      "fiscal_leakage_multiplier": 1.48,
      "recovery_discount": 1.00,
   },
   "LPI-UMKM": {
      "capital_diversion_rate": (1.0 - ETA) * 0.18,
      "bank_risk_inflation": 1.08,
      "productivity_discount": 0.92,
      "fiscal_leakage_multiplier": 1.12,
      "recovery_discount": 0.88,
   },
}

# ==============================
# 5.  BASE SYSTEM PARAMETERS
# ==============================

PARAMS = {
   "KUR": {
      "burden_ratio": 0.32, "productivity_boost": 0.035, "banking_multiplier":
      0.02,     "operational_cost":     0.012,     "recovery_strength":     0.14,
      "debt_overhang_rate": 0.08,
   },
   "PCGS": {
      "burden_ratio": 0.28, "productivity_boost": 0.030, "banking_multiplier":
      0.04,     "operational_cost":     0.015,     "recovery_strength":     0.12,
      "debt_overhang_rate": 0.05,
   },
   "Grant": {
      "burden_ratio": 0.18, "productivity_boost": 0.045, "banking_multiplier":
      0.01,     "operational_cost":     0.035,     "recovery_strength":     0.18,
      "debt_overhang_rate": 0.00,
   },
   "LPI-UMKM": {
      "burden_ratio": 0.28, "productivity_boost": 0.040, "banking_multiplier":
      0.03,     "operational_cost":     0.016     +     (1.0     -     ETA)     *     0.012,
      "recovery_strength":                    LPI_RECOVERY_STRENGTH,
      "debt_overhang_rate": 0.00,
   },
}

for s in SYSTEMS:
   mh = MH[s]
   bp = PARAMS[s]
   bp["eff_productivity"]          =          bp["productivity_boost"]          *
      mh["productivity_discount"] * (1.0 - mh["capital_diversion_rate"])
   bp["eff_op_cost"]          =          bp["operational_cost"]          *
      mh["fiscal_leakage_multiplier"]
   bp["eff_recovery"] = bp["recovery_strength"] * mh["recovery_discount"]

# ==============================
# 6.  STORAGE MATRICES
# ==============================

budget_arr   = {s: np.zeros((MC_RUNS, YEARS + 1)) for s in SYSTEMS}
survival_arr = {s: np.zeros((MC_RUNS, YEARS + 1)) for s in SYSTEMS}
gdp_arr      = {s: np.zeros((MC_RUNS, YEARS + 1)) for s in SYSTEMS}
lpi_asset_pool    = np.zeros((MC_RUNS, YEARS + 1))
lpi_buyback_cumul = np.zeros((MC_RUNS, YEARS + 1))
lpi_formal_rate   = np.zeros((MC_RUNS, YEARS + 1))
lpi_net_default   = np.zeros((MC_RUNS, YEARS + 1))

# ==============================
# 7.  MONTE CARLO SIMULATION LOOP
# ==============================

for run in range(MC_RUNS):
   sector_risk = RNG.choice([0.80, 1.00, 1.35], size=N_AGENTS, p=[0.30,
      0.50, 0.20])
```

```
initial_revenue = RNG.lognormal(mean=4.5, sigma=0.55, size=N_AGENTS)
revenue_threshold = np.exp(4.5) * REVENUE_GRADUATION_MULTIPLE

for system in SYSTEMS:
    p = PARAMS[system]
    mh = MH[system]
    budget = INIT_CAPITAL
    revenue = initial_revenue.copy()
    state = np.full(N_AGENTS, S_PERFORMING, dtype=np.int8)
    tax_reg = np.zeros(N_AGENTS, dtype=np.int8)
    asset_pool = revenue * ASSET_BACKING_RATIO * 0.001 if system == "LPI-UMKM" else None
    cumul_buyback = 0.0

    budget_arr[system][run, 0] = budget
    survival_arr[system][run, 0] = 100.0
    gdp_arr[system][run, 0] = np.sum(revenue)

    for year in range(1, YEARS + 1):
        crisis = CRISIS_START <= year <= CRISIS_END
        macro_growth = RNG.normal(0.88, 0.07) if crisis else RNG.normal(1.04, 0.04)
        macro_vol = 0.28 if crisis else 0.12
        systemic_stress = 1.35 if crisis else 1.00
        micro_shock = RNG.lognormal(mean=0.0, sigma=macro_vol, size=N_AGENTS)

        if system == "LPI-UMKM":
            eff_burden = p["burden_ratio"] * (1.0 - ETA * ASSET_BACKING_RATIO) * (1.10 if crisis else 1.0)
        elif system == "KUR" and crisis:
            eff_burden = p["burden_ratio"] * (1.0 + p["debt_overhang_rate"])
        else:
            eff_burden = p["burden_ratio"]

        revenue = revenue * macro_growth * (1.0 + p["eff_productivity"]) * (1.0 + p["banking_multiplier"]) * micro_shock / sector_risk
        revenue = np.clip(revenue, 0.01, None)
        distress_threshold = (revenue * eff_burden) * systemic_stress * mh["bank_risk_inflation"]
        base_hazard = 0.05 + 0.22 * (revenue < distress_threshold).astype(float) + 0.08 * float(crisis)

        to_distress = (state == S_PERFORMING) & (RNG.random(N_AGENTS) < base_hazard)
        state[to_distress] = S_DISTRESSED

        p_def = (0.22 + 0.08 * float(crisis)) * (1.0 - ETA * HAZARD_REDUCTION_FACTOR) if system == "LPI-UMKM" else (0.22 + 0.08 * float(crisis))
        to_default = (state == S_DISTRESSED) & (RNG.random(N_AGENTS) < p_def)
        state[to_default] = S_DEFAULT
        to_restructure = (state == S_DISTRESSED) & ~to_default & (RNG.random(N_AGENTS) < p["eff_recovery"])
        state[to_restructure] = S_RESTRUCTURED
        state[(state == S_RESTRUCTURED) & (RNG.random(N_AGENTS) < 0.40)] = S_PERFORMING

        buyback_inflow = 0.0; net_default_cost = 0.0; tp_offset = 0.0
        if system == "LPI-UMKM":
            active_mask = (state != S_DEFAULT) & (state != S_EXITED)
            asset_pool[active_mask] *= np.clip(RNG.normal(macro_growth, macro_vol * 0.5, N_AGENTS)[active_mask], 0.05, None)
            newly_defaulted = (state == S_DEFAULT) & to_default
            if np.any(newly_defaulted):
                gl = np.sum(asset_pool[newly_defaulted]) / ASSET_BACKING_RATIO
                net_default_cost = max(gl * (1 - FIRST_LOSS_TRANCHE) - np.sum(asset_pool[newly_defaulted]) * ASSET_RECOVERY_RATE * mh["recovery_discount"], 0.0)
                asset_pool[newly_defaulted] = 0.0
            do_buyback = ((state == S_PERFORMING) | (state == S_RESTRUCTURED)) & (RNG.random(N_AGENTS) < BUYBACK_PROB)
            if np.any(do_buyback):
                buyback_inflow = np.sum(asset_pool[do_buyback]) * BUYBACK_MARKUP
                state[do_buyback] = S_INDEPENDENT; asset_pool[do_buyback] = 0.0
                cumul_buyback += buyback_inflow
            if year >= THIRD_PARTY_YEAR: tp_offset = np.sum(state == S_RESTRUCTURED) * ASSET_BACKING_RATIO * THIRD_PARTY_SHARE * 0.00008
            graduating = (state != S_DEFAULT) & (state != S_EXITED) & (tax_reg == 0) & (revenue > revenue_threshold)
            cliff_exits = graduating & (RNG.random(N_AGENTS) < TAX_CLIFF_EXIT_PROB)
            state[cliff_exits] = S_EXITED; tax_reg[graduating & ~cliff_exits] = 1
            lpi_formal_rate[run, year] = np.mean(tax_reg == 1)

        active = (state != S_DEFAULT) & (state != S_EXITED)
        n_active = np.sum(active); n_default = np.sum(state == S_DEFAULT); gdp_p = np.sum(revenue[active])
        op_c = n_active * p["eff_op_cost"] * 0.001; def_c = n_default * 0.004

        if system == "KUR": budget += gdp_p * 0.000018 + n_active * 0.00035 - op_c - def_c
        elif system == "PCGS": budget += gdp_p * 0.000022 + n_active * 0.00050 - n_default * 0.006 * mh["bank_risk_inflation"] - op_c
        elif system == "Grant": budget += gdp_p * 0.000024 - (n_active * 0.0015 if year <= 3 else 0) - op_c - def_c
        elif system == "LPI-UMKM":
            mu_f, sig_f = (-0.03, 0.24) if crisis else (0.06, 0.12)
            budget *= (1.0 + mu_f + sig_f * RNG.normal(0, 1))
            if year == CRISIS_START: budget *= (1.0 - RNG.uniform(0.10, 0.18))
            budget += gdp_p * 0.000020 + buyback_inflow - net_default_cost - op_c - tp_offset
        budget = max(budget, 0.0)
        budget_arr[system][run, year] = budget; survival_arr[system][run, year] = np.mean(active) * 100; gdp_arr[system][run, year] = gdp_p
        if system == "LPI-UMKM": lpi_asset_pool[run, year] = np.sum(asset_pool); lpi_buyback_cumul[run, year] = cumul_buyback; lpi_net_default[run, year] = net_default_cost

# ==============================
# 8. VISUALIZATION
# ==============================

COLORS = {"KUR": "#3A7BD5", "PCGS": "#E03B24", "Grant": "#E8900D", "LPI-UMKM": "#2E8B57"}
t = np.arange(YEARS + 1)

def pci(matrix): return np.mean(matrix, axis=0), np.percentile(matrix, 2.5, axis=0), np.percentile(matrix, 97.5, axis=0)

fig1, axes1 = plt.subplots(1, 3, figsize=(21, 7))
for ax, data, title, ylabel in zip(axes1, [budget_arr, survival_arr, gdp_arr], ["A. Fiscal Dynamics", "B. Survival Rate", "C. GDP Proxy"], ["T IDR", "%", "Index"]):
    ax.set_facecolor("#FDFCFA"); ax.grid(True, linestyle="--", alpha=0.5)
    for s in SYSTEMS:
        m, lo, hi = pci(data[s])
        ax.plot(t, m, label=s, color=COLORS[s], lw=2.3)
        ax.fill_between(t, lo, hi, color=COLORS[s], alpha=0.12)
    ax.axvspan(CRISIS_START, CRISIS_END, color="gray", alpha=0.1)
    ax.set_title(title, fontweight="bold"); ax.set_ylabel(ylabel)
axes1[0].legend()
plt.tight_layout(); plt.savefig("core_results.png"); plt.show()

fig2, axes2 = plt.subplots(2, 2, figsize=(18, 12))
lpi_data = [(lpi_asset_pool, "Asset Pool (T IDR)"), (lpi_buyback_cumul, "Buyback (T IDR)"), (lpi_formal_rate*100, "Formalization (%)"), (lpi_net_default, "Net Default (T IDR)")]
```

```
for ax, (data, label) in zip(axes2.flat, lpi_data):
    ax.set_facecolor("#FDFCFA"); ax.grid(True, linestyle="--", alpha=0.5)
    m, lo, hi = pci(data)
    ax.plot(t, m, color=COLORS["LPI-UMKM"], lw=2.3)
    ax.fill_between(t, lo, hi, color=COLORS["LPI-UMKM"], alpha=0.15)
    ax.set_title(label, fontweight="bold")
plt.tight_layout(); plt.savefig("lpi_mechanisms.png"); plt.show()
```