Bridging System Dynamics and Deep Learning: Macroeconomic Policy Design via Differentiable Stock-Flow Consistent (SFC) Models

Abstract

Stock-Flow Consistent (SFC) models have long served as the rigorous backbone for heterodox macroeconomic analysis, ensuring accounting integrity across institutional sectors. However, their traditional implementation in discrete-time simulation environments limits their utility for complex sensitivity analysis and automated policy optimization. This paper introduces a novel framework for implementing SFC models as differentiable programs using modern computational engines such as JAX, PyTorch, and TensorFlow. By treating the macroeconomic system as a computational graph, we enable the use of automatic differentiation (AD) to compute exact analytical gradients across entire time horizons. We demonstrate that this approach allows for (1) instantaneous sensitivity analysis of long-term stability to parameter shifts, (2) the use of gradient-based optimization to discover "optimal control" policy knobs that minimize systemic volatility, and (3) high-performance automated calibration against historical datasets. We provide a minimal implementation of the "SIM" model as a proof-of-concept, showcasing a significant leap in the analytical depth and scalability of the SFC methodology.

1. Introduction: The Limits of Black-Box Simulation

Context: The history of SFC modeling (Godley, Lavoie) and its importance in systemic risk and institutional accounting.

Problem: Traditional models are "run-and-see." Sensitivity analysis requires expensive Monte Carlo simulations; policy discovery is a process of trial-and-error where researchers manually tune "trigger values" to find stability.

The Opportunity: The rise of "Differentiable Programming" in machine learning offers a path to turn these simulations into transparent, optimizable manifolds. By making the transition matrix differentiable, we move from observing the system to navigating it via gradients.

2. Methodology: From Loops to Graphs

Defining the Computational Graph

We represent the SFC transaction-flow matrix as a series of differentiable tensor operations. Each time step t is a node in a graph, and the "Stock" variables (Wealth, Capital, Debt) act as the hidden state that propagates forward.

Automatic Differentiation (AD)

We apply Backpropagation Through Time (BPTT). Unlike manual calculus, AD tracks the partial derivatives of every accounting identity simultaneously. If a tax rate changes in year 1, the engine calculates the exact impact on firm liquidity in year 50 without a re-simulation.

Framework Selection

- **JAX:** Ideal for research due to vmap (parallel simulations) and jit (compilation to XLA).
- **PyTorch/TensorFlow:** Superior for integrating neural networks (e.g., using a neural net as a "smart" central bank policy controller).

3. The "SIM-Diff" Toy Model

Implementation

A differentiable version of the basic SIM model (Service-Induced Macroeconomics).

Analytical Gradients

Visualizing the gradient of GDP stability with respect to tax rates ($\nabla_\theta Y$). We show that the "optimal" tax rate is not a point, but a trajectory that can be solved for.

Sensitivity Manifolds

Mapping how small parameter changes (propensity to consume, tax thresholds) cascade through institutional balance sheets over T steps, identifying "stiffness" in the system where small errors lead to explosive debt.

4. Enhanced Economic Insights: The Loss Function ()

The core benefit of a differentiable model is the ability to define a Loss Function that represents the "Health" of the economy and then use gradients to minimize it.

4.1. Definition of the Multi-Objective Loss Function

We propose a Loss Function that penalizes three primary systemic failures:

$$ {}^{T}{}_{t}{}^{t} \qquad {}^{T}{}_{t} Y_t \qquad {}^{T}{}_{t}{}^{t} $$

Where:

- **Unemployment Gap** ($_t$): Penalizes deviations from target unemployment.
- **Volatility Penalty** (Y_t): Penalizes "chattering" or high-frequency oscillations in GDP (the "oscillation problem" in many SFC models).

- **Financial Fragility** ($_t$): A penalty for unsustainable debt-to-equity ratios or net lending imbalances that exceed a safety threshold.

4.2. Policy Optimization via Gradients

By computing ∇, we can update policy parameters (like government spending triggers or interest rate rules) using an optimizer (e.g., Adam):

$$\nabla$$

This allows the model to "discover" the most stabilizing policy rules automatically.

5. Automated Calibration & Inverse Modeling

Data Fitting

Treating historical national accounts (SNA) as training data.

Parameter Recovery

Instead of manually setting behavioral constants (e.g.,), we treat them as "learnable weights." The model "learns" the behavior of households by minimizing the error between the model's predicted flows and actual historical transaction-flow matrices.

6. Discussion and Future Work

Scalability

Transitioning from the SIM model to complex multi-sector models with thousands of variables.

Closing the Loop

The potential for real-time policy "cockpits" where policy-makers can see the "gradient" of their decisions in real-time.

Technical Appendix: Reference Implementation

Pseudo-code

```
def sfc_loss(params, initial_state):
    states = simulate_sfc(params, initial_state)  # Differentiable
simulation
    U = calculate_unemployment(states)
    Y = calculate_gdp(states)
    return jnp.mean((U - 0.05)**2) + jnp.var(Y)  # Minimize gap & volatility
```

```
# The magic 'grad' function
policy_gradients = grad(sfc_loss)(current_params, start_state)
```

Instructions for Calculating Hessians

Methods for finding second-order stability regions and analyzing the curvature of the loss landscape.