Interim Report: 20 June 2025

I. Core System and Problem Definitions

A. *core.models.base.DynamicalSystem*

Abstract base class for dynamical systems:

$$\dot{x} = f(x, u, \theta, t) \quad \text{(Dynamics)}$$
$$y = g(x, \theta, t) \quad \text{(Observation)}$$

Interface & Capabilities:

- `__init__(n, m, p, params, ...)`: Constructor taking state (`n`), input (`m`), output (`p`) dimensions, and a JAX array of parameters (`params`).
- `f(...)`: Abstract method for the dynamics function.
- `g(...)`: Abstract method for the observation function.
- `simulate(...)`: High-level method that uses `diffrax` to solve the initial value problem and return the system's trajectory.

B. *core.idp.IdentificationProblem*

A high-level class designed to encapsulate all components of a system identification experiment into a single object. **Interface & Capabilities:**

- Bundles a `DynamicalSystem` instance with:
- $\mathtt{y}_m eas : Measuredoutputdata$

II. Model Implementations

`core.models.`

A. *linearCT.LinearSystem*

Implements a classical Linear Time-Invariant (LTI) system:

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t)$$

Interface & Capabilities:

- Inherits from `DynamicalSystem`
- Implements `is_controllable()` and `is_observable()`, which can use either the `Kalman` or `PBH` tests

B. *.controlAffine.ControlAffineSystem*

Implements a nonlinear system with dynamics affine in the control input:

$$\dot{x}(t) = f_d(x, \theta) + \sum_{i=1}^{m} g_i(x, \theta) u_i(t)$$
$$y(t) = h(x)$$

Interface & Capabilities:

- Inherits from `DynamicalSystem`
- Implements observability and controllability criteria based on Lie Algebra
- Basic operations of the Lie Algebra are implemented in `core.utils.algebra`

III. Algorithmic Toolkit

A. *Differential Geometry (core.utils.algebra)*

Provides fundamental operations for nonlinear system analysis. These functions are implemented using `jax.jacfwd` and `jax.jacrev` for efficient and accurate computation of Jacobians.

`lie_derivative(f, h)`: Computes the Lie derivative of a scalar field h along a vector field f: $L_f h = (\nabla h) \cdot f$.

`lie_bracket(f, g)`: Computes the Lie bracket of two vector fields f and g: $[f, g] = (\nabla g) f - (\nabla f) g$.

B. *Structural Identifiability Analysis*

These methods assess whether parameters can be uniquely determined from noise-free data.

Taylor Series Method (`analyze_diffalg`) :

- **Principle:** Parameters are identifiable if they uniquely affect the Taylor series expansion of the output $y(t)$ at $t = 0$. The coefficients of this series are functions of repeated Lie derivatives of $h(x)$ with respect to $f(x, \theta)$, evaluated at x_0.
- **Computation:** An identifiability matrix is formed by taking the Jacobian of the vector of output derivatives with respect to the parameters θ: $J = \frac{\partial [y(0), \dot{y}(0), \ddot{y}(0), \dots]}{\partial \theta}$.
- **Criterion:** Parameters are structurally locally identifiable if $\mathrm{rank}(J)$ equals the number of parameters.

Initial Condition-based Score (ICIS) (`QiuICIS.analyze`):

- **Principle:** For an unforced, diagonalizable LTI system, parameters in A are identifiable if and only if the initial condition x_0 excites all dynamic modes (i.e., has a non-zero projection onto every eigenvector).
- **Computation:** The score is the product of the squared magnitudes of the coordinates of x_0 in the eigenvector basis: ICIS $= \prod_{i=1}^{n} |(V^{-1}x_0)_i|^2$, where V is the eigenvector matrix.

C. Practical Identifiability and Sensitivity Analysis

All methods are part of the `QiuICIS` class.

Sensitivity Trajectory Computation (`compute_sensitivity_trajectory`):

- **Principle:** Computes the sensitivity of the state trajectory to changes in parameters, $S(t) = \frac{\partial x(t)}{\partial \theta}$. This is achieved by solving an augmented state-space system.
- **Computation:** The state vector is augmented as $z = [x^\top, \text{vec}(S)^\top]^\top$. The dynamics of the augmented system are:

$$\dot{x} = f(x, \theta)$$
$$\dot{S} = \frac{\partial f}{\partial x} S + \frac{\partial f}{\partial \theta}$$

  This augmented ODE system is solved once using `diffrax`, yielding both $x(t)$ and $S(t)$.

Likelihood Ratio Test (`analyze_LR`) :

- **Principle:** A statistical test to determine if a parameter is necessary for a model to fit the data. It compares the goodness-of-fit (Residual Sum of Squares, RSS) of the full model against a nested model where one parameter is fixed.
- **Computation:**
  1) Minimize RSS for the full model to find $\hat{\theta}$ and RSS_{full}.
  2) For each parameter θ_i, fix it to $\hat{\theta}_i$ and re-minimize the RSS over the remaining parameters to find $RSS_{reduced,i}$.
  3) Calculate the test statistic: $LR_i = N \log(RSS_{reduced,i}/RSS_{full})$.
- **Criterion:** The statistic LR_i is compared to a $\chi^2(1)$ distribution. A small p-value indicates the parameter is practically identifiable.

This script serves as a template for conducting in-silico experiments.

Objective: To evaluate the effect of control input magnitude on the practical identifiability of parameters in the `SimpleGlucoseInsulinModel`.

Workflow:
1) **Setup:** Define a ground truth model and generate synthetic data with varying input signal strengths.
2) **Encapsulation:** For each condition, create an `IdentificationProblem` instance.
3) **Optimization:** A cost function (Sum of Squared Errors) is defined, which wraps the model's `simulate` method. `scipy.optimize.minimize` is then used to find the maximum likelihood parameter estimates.
4) **Conclusion:** The estimated parameters are plotted against the true values for each input condition, allowing for a visual assessment of identifiability.