TDAstro: Community-driven light curve modeling for LSST

A.I. Malz, M. Dai, K. Malanchev, J. Kubica, O. Lynn, M. Tauraso, the LINCC-Frameworks Team

LSST Interdisciplinary Network for Collaboration and Computing (LINCC) Frameworks

Time-series forward modeling infrastructure is a Rubin-wide need.

Realistic light curve simulations are essential to many aspects of time-domain science with LSST, before, during, and after the survey itself.

- Classifier stress-testing
- Observing strategy optimization
- Forecasting for new physical theories
- Validating analysis pipelines
- Simulation-based inference
- . . . and more!

A **shared infrastructure** with **broad applicability** across time-domain phenomena that produces **realistic** LSST-like light curves **at scale** with **ease of use and contribution** would play a key role in enabling Rubin community science!

LINCC-Frameworks builds software infrastructure for LSST science.

Supported by Schmidt Sciences, the professional software engineers and astronomy researchers of the LINCC-Frameworks team build tools to broadly enable LSST science beyond what Rubin alone can provide.

	Cross-matching	Photo-z	Selection functions	Time series	Image reprocessing	Image analysis
Cosmology	✓✓	✓✓	✓✓	✓✓	✓	✓
Extragalactic static	✓✓	✓✓	✓✓		✓✓	✓
Extragalactic transient	✓✓	✓✓	✓	✓✓	✓	✓
Extragalactic variable	✓✓	✓	✓	✓✓	✓	✓
Local Universe transient & variable	✓✓		✓	✓✓		
Local Universe static	✓✓		✓✓		✓	✓
Solar system	✓		✓✓	✓✓	✓	✓✓

Figure: The *Data to Software to Science Workshop* (arXiv:2208.02781) tasked domain experts across the Rubin community with identifying cross-cutting software infrastructure needs for the Rubin community, highlighting infrastructure supporting time-domain simulation as a key need across fields.

`TDAstro` is designed to address technical needs for time-series data across multiple science use cases, guided by the following priorities.

- Usability: easy to install, use, and contribute to
- Flexibility: supports inclusion of many models
- Integration: instruments, surveys, data types, etc.
- Robustness: self-consistent parameter sampling
- Efficiency: usable with and without supercomputer resources
- Reliability: continuous integration, unit tests, documentation

Structure of the TDAstro workflow

`TDAstro` supports end-to-end forward-modeling pipelines as well as a la carte usage of individual components to embed in external pipelines.

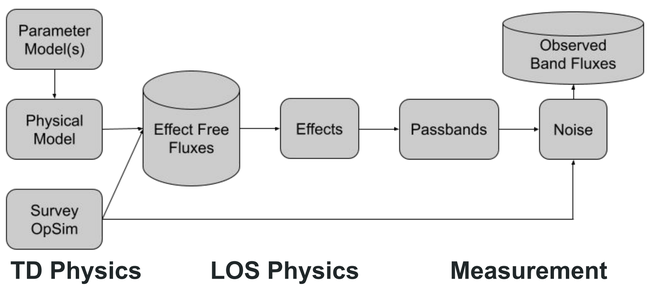

Figure: `TDAstro`'s modular structure enables users to build end-to-end pipelines for many time-domain simulation applications.

- Models refer to physical sources of light curve data including the time-variable phenomena such as supernovae, AGN, and variable stars, as well as static contributions such as host galaxies for extragalactic objects.
- Effects apply line-of-sight physics, such as Milky Way extinction and redshifting of extragalactic sources, to error-free fluxes at execution time.
- Measurements of a source require the observing strategy in LSST's OpSim format and a PassbandGroup of the desired survey's photometric filters' transmission curves.

TDAstro usage demonstration: SN Ia as a case study

This poster illustrates the usage of `TDAstro` to simulate Type Ia supernovae (SNe Ia) as a well-investigated extragalactic transient, but other models are available, with more in progress.

Intuitively build a graphical model for your physical source.

Parameters for a physical model are self-consistently drawn from a directed acyclic graph (DAG) relating the user's distributions for the physical model parameters. The physical model for the SN Ia example can be represented as a DAG relating parameters about the transient, its host galaxy, and other physics affecting its underlying time-dependent spectrum.

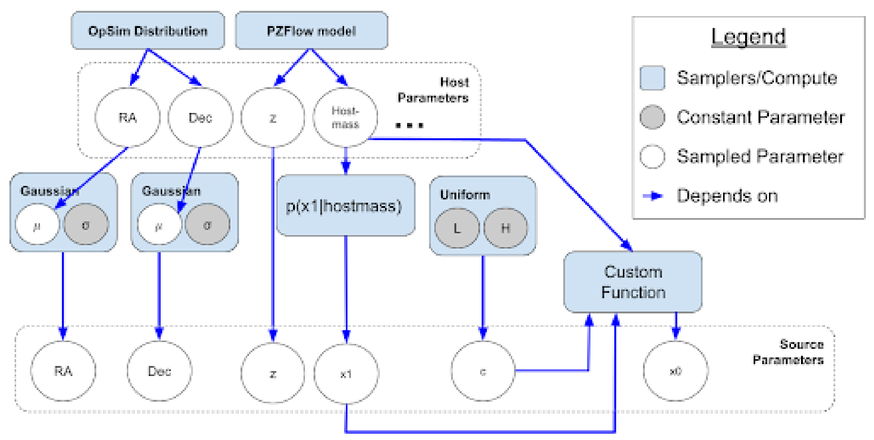

Figure: A user-defined model in terms of a directed acyclic graph relating parameters and distributions from which they are sampled.

Chain modular nodes to define a hierarchical source model.

Users define models using familiar Python programming syntax. `TDAstro`'s flexible parameter definitions enable users to build sophisticated models from simple Node components.

```
>> source = SALT2JaxModel(                          # User-selected model
    t0=NumpyRandomFunc("uniform",                   # t0 parameter is a call to sample
      low=t_min, high=t_max),                       # from a provided distribution.
    x0=x0_func,                                     # x0 and x1 parameters are calls to
    x1=x1_func,                                     # sample from user-defined functions.
    c=0.01,                                         # c parameter is a constant.
    ra=NumpyRandomFunc("normal",                    # ra and dec parameters are defined
      loc=host.ra, scale=0.01),                     # as calls to sample from provided
    dec=NumpyRandomFunc("normal",                   # distributions that themselves depend
      loc=host.dec, scale=0.01),                    # on other model node parameters.
    redshift=host.redshift)  # redshift parameter is from another model node.
```

Easily make OpSim-specified, survey-specific observations of a model.

An OpSim-formatted observing plan specifies photometric filters by name, but actual filters vary across instruments even if they share names. While specifying the actual transmission curves corresponding to the survey they want to simulate, users may also subselect observations from an OpSim for faster processing by ignoring filters less relevant to their science case. `TDAstro` includes transmission curves for upcoming surveys and will connect to archival filter sets for users to more easily conduct cross-survey simulations.

```
>> opsim_db = OpSim.from_url(opsim_url)      # Load an OpSim from file or URL.
>> passband_group = PassbandGroup.from_preset(preset="LSST",
     filters_to_load=["g", "r", "i", "z"])   # Specify passbands for survey.
>> filter_mask = passband_group.mask_by_filter(opsim_db["filter"])
>> ops_data = opsim_db.filter_rows(filter_mask)   # Isolate the desired bands.
```

Running a simulation with TDAstro is efficient and easy!

Instead of manually sampling parameters, passing a table of parameters into physical models, saving oversampled light curves, then whittling them down and deriving errors according to an observing plan, `TDAstro` enables users to perform a streamlined simulation without necessarily stopping at intermediate steps, saving time and storage space.

```
>> lightcurves = simulate_lightcurves(
    source_model,                          # Physical source and effects model
    1_000,                                 # Number of simulations
    ops_data,                              # Observations specification
    passband_group)                        # Passband data
```

`TDAstro` enables anyone to make a PLAsTiCC-like data challenge simulation tuned to their science goals.

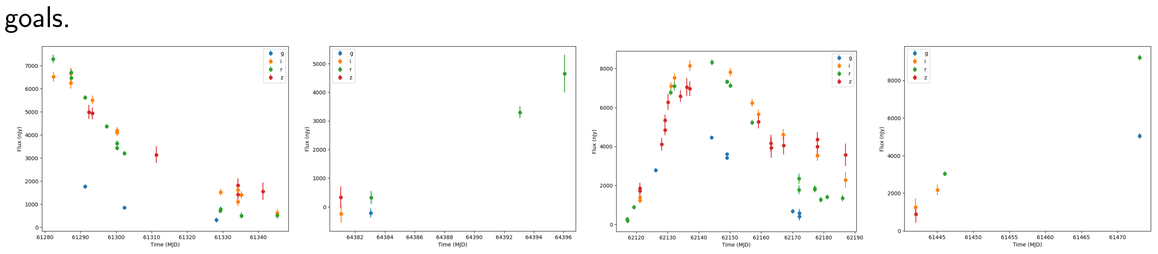

Figure: SN Ia light curves generated using `TDAstro` with PLAsTiCC priors (arXiv:1903.11756) under the `baseline_v3.4` OpSim modified to only include LSST's *griz* passbands.

A node chain for hyperparameters

Transient and variable rates and associations are key components to generate and use simulated light curves. In the SN Ia example, the transient inherits its redshift from its host galaxy, so `TDAstro` can sample redshifts from a realistic distribution.

```
>> pz_node = PZFlowNode.from_file(
      path_to_trained_flow,
      node_label="pznode")
>> host = SNIaHost(
       ra=pz_node.RA_GAL,
       dec=pz_node.DEC_GAL,
       hostmass=pz_node.LOGMASS,
       redshift=NumpyRandomFunc("uniform",
         low=0.1, high=0.6),
       node_label="host")
```

Adding observational effects to a model

Line-of-sight effects are key to realism of simulated light curves, and their implementation has posed a challenge to decentralized efforts at light curve simulation. `TDAstro` provides a structure for such effects to be applied to any light curve simulation as an addition into the physical model.

```
>> model = SinWaveSource(
     brightness=100.0,        # Define the
     frequency=20.0)     # physical model.
>> ext_effect = ExtinctionEffect(
     extinction_model="CCM89",
     ebv=dust_map_node,
     Rv=3.1) # Specify extinction effect.
>> model.add_effect(ext_effect)
          # Add the effect to the model.
```

`TDAstro` can thus serve as a central repository for implementations of effects to reduce duplication of effort across the community while making efficient implementations available to everyone.

Testing realism of effects against DP1

`TDAstro` can be used to model how a hypothetical source would have been observed in DP1.

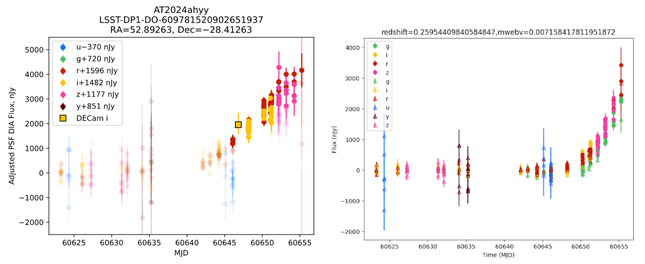

Figure: Left: Actual unclassified transient observed during DP1 (arXiv:2506.23955). Right: Synthetic SN Ia made with `TDAstro` using the DP1 OpSim.

The comparable observational errors demonstrate the realism of the `TDAstro` simulation process.

How to get started and contribute

`TDAstro` is developed publicly on GitHub at **github.com/lincc-frameworks/tdastro**.
API documentation and demos may be found on ReadTheDocs at **tdastro.readthedocs.io**.
Get started with `>> pip install tdastro`.

`TDAstro` meets the Rubin community's need for easy-to-use software infrastructure to centralize resources for light curve simulation. Its value to the broader time-domain astronomical community grows with the depth of its library of physical models and observational effects — TDAstro needs you!

`TDAstro` is designed to make it easy to contribute your models, effects, and survey definitions. Connect with the `TDAstro` team by posting an issue on GitHub or joining the **#lincc-frameworks-tdastro** channel of the LSST-DA Slack workspace.