In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
%matplotlib inline
%config InlineBackend.figure_formats = ['svg']
import scipy
```

L03

In the following experiments I use PySINDy to find the underlying dynamics of a system soly based on data. By first simulating a dynamical system with a conventional numerical integrator, I aim to show how one could (re)-discover the dynamics by taking looking at each snapshot in time from the output of the numerical integrator.

(E7.4) Finding the dynamics of the Lorenz system with PySINDy

In this experiment I show how to find the dynamics of the Lorenz system. To do this I first integrate the following system with scipy's solve_ivp. To test the quality of the solution found via PySINDy I also simulate a similar system, but with other initial starting parameters. If the system obtained from PySINDy is correct, then this result sould also predict this other system. To make this experiment more realistic noise will be added to the measurements.

$$x' = \sigma(y - x)$$

$$y' = x(\rho - z) - y$$

$$z' = xy - \beta z$$

In [ ]:
```
def lorenz(t, vars, sigma, rho, beta):
    x, y, z = vars
    dx = sigma * (y - x)
    dy = x*(rho-z)-y
    dz = x*y - beta*z
    return np.array([dx, dy, dz])
```

In [ ]:
```
dt = 0.002
ts = np.arange(0,100, dt)
t_span = (0,100)
y0_train = np.array([1,2,3])
y0_test = np.array([1,4,3])
```

```
constants = np.array([10, 28, 2.667])
```

In [ ]:
```
sol_train = scipy.integrate.solve_ivp(
    fun=lorenz,
    t_span=t_span,
    t_eval=ts,
    y0=y0_train,
    args=(constants)
)

sol_test = scipy.integrate.solve_ivp(
    fun=lorenz,
    t_span=t_span,
    t_eval=ts,
    y0=y0_test,
    args=(constants)
)
```

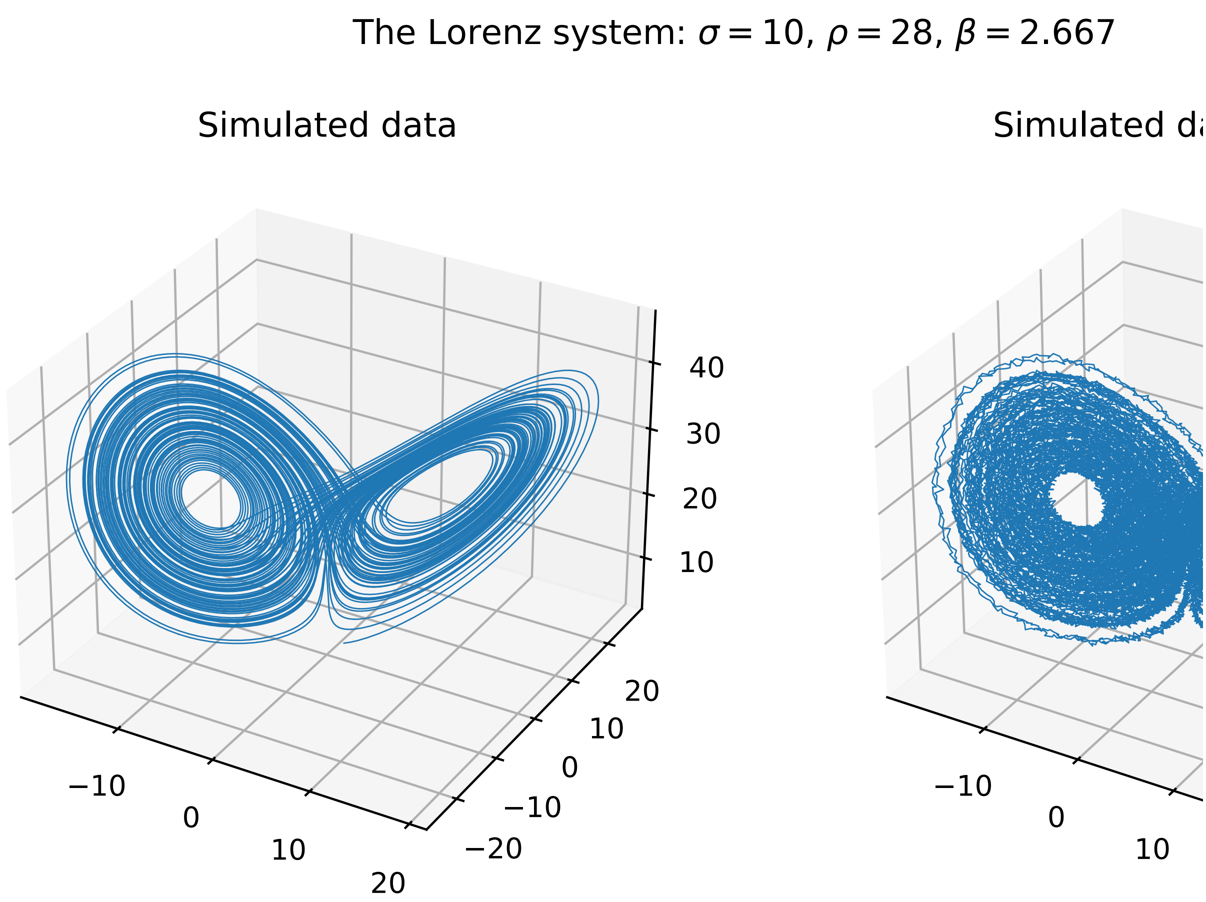
In [ ]:
```
x_train = sol_train.y.T
x_train_noise = x_train + np.random.normal(0,0.2,x_train.shape)

x_test = sol_test.y.T

fig = plt.figure(figsize=(10,5))
ax1 = fig.add_subplot(1,2,1, projection="3d")
ax2 = fig.add_subplot(1,2,2, projection="3d")

plt.suptitle(r"The Lorenz system: $\sigma = 10$, $\rho = 28$, $\beta = 2.
ax1.plot(*x_train.T, lw=0.5)
ax1.set_title("Simulated data")
ax2.plot(*x_train_noise.T, lw=0.5)
ax2.set_title("Simulated data + noise")
```

Out[ ]:
```
Text(0.5, 0.92, 'Simulated data + noise')
```



In the PySINDy framework you can add a threshold in order to the resulting terms. In the next sections I experiment with this threshold and what to set it in order to find the best solution.

Without threshold

Without noise:

In [ ]:
```
# Code from https://github.com/dynamicslab/pysindy/blob/master/examples/1
import pysindy as ps

feature_names = ['x', 'y', 'z']
sparse_regression_optimizer = ps.STLSQ(threshold=0)
model = ps.SINDy(feature_names=feature_names, optimizer=sparse_regression
model.fit(x_train, t=dt)
model.print()
```

```
(x)' = -0.016 1 + -9.963 x + 9.975 y + 0.002 z + -0.001 x z + 0.001 y z
(y)' = 0.290 1 + 27.674 x + -0.873 y + -0.039 z + -0.006 x^2 + 0.006 x y +
-0.990 x z + -0.001 y^2 + -0.002 y z + 0.001 z^2
(z)' = -0.596 1 + 0.039 x + -0.019 y + -2.586 z + 0.012 x^2 + 0.988 x y +
-0.001 x z + 0.002 y^2 + -0.003 z^2
```

With noise:

In [ ]:
```
feature_names = ['x', 'y', 'z']
sparse_regression_optimizer = ps.STLSQ(threshold=0)
model = ps.SINDy(feature_names=feature_names, optimizer=sparse_regression
model.fit(x_train_noise, t=dt)
model.print()
```

```
(x)' = 1.727 1 + -9.306 x + 9.577 y + -0.221 z + -0.035 x^2 + 0.033 x y +
-0.018 x z + -0.008 y^2 + 0.010 y z + 0.007 z^2
(y)' = -0.126 1 + 27.108 x + -0.547 y + 0.016 z + 0.003 x^2 + -0.001 x y +
-0.974 x z + -0.011 y z + -0.001 z^2
(z)' = 0.666 1 + 0.149 x + -0.082 y + -2.755 z + -0.013 x^2 + 1.005 x y +
-0.004 x z + 0.001 y^2 + 0.002 y z + 0.003 z^2
```

Partial conclusion: The solutions without a threshold does not look like the real system.

With threshold

In [ ]:
```
feature_names = ['x', 'y', 'z']
sparse_regression_optimizer = ps.STLSQ(threshold=0.1)
model = ps.SINDy(feature_names=feature_names, optimizer=sparse_regression
model.fit(x_train, t=dt)
model.print()
```

```
(x)' = -10.004 x + 10.004 y
(y)' = 27.779 x + -0.948 y + -0.993 x z
(z)' = -2.667 z + 0.999 x y
```

In [ ]:
```
feature_names = ['x', 'y', 'z']
sparse_regression_optimizer = ps.STLSQ(threshold=0.1)
model = ps.SINDy(feature_names=feature_names, optimizer=sparse_regression
model.fit(x_train_noise, t=dt)
model.print()
```

```
(x)' = -9.952 x + 9.960 y
(y)' = 27.568 x + -0.883 y + -0.988 x z
(z)' = -2.665 z + 0.998 x y
```

Partial conclusion: These solutions looks a lot better than the previous.

Setting the correct threshold

To find the most optimal threshold I make a linear scan through 0 to 1. I use the noisy data and compare the resulting system to the real data. I use two different metrics: (a) the one provided by PySINDy and (b) the MSE between the found test system and the true simulated system without noise.

In [ ]:
```
from sklearn import metrics

threshold_scan = np.linspace(0, 1.0, 30)
```

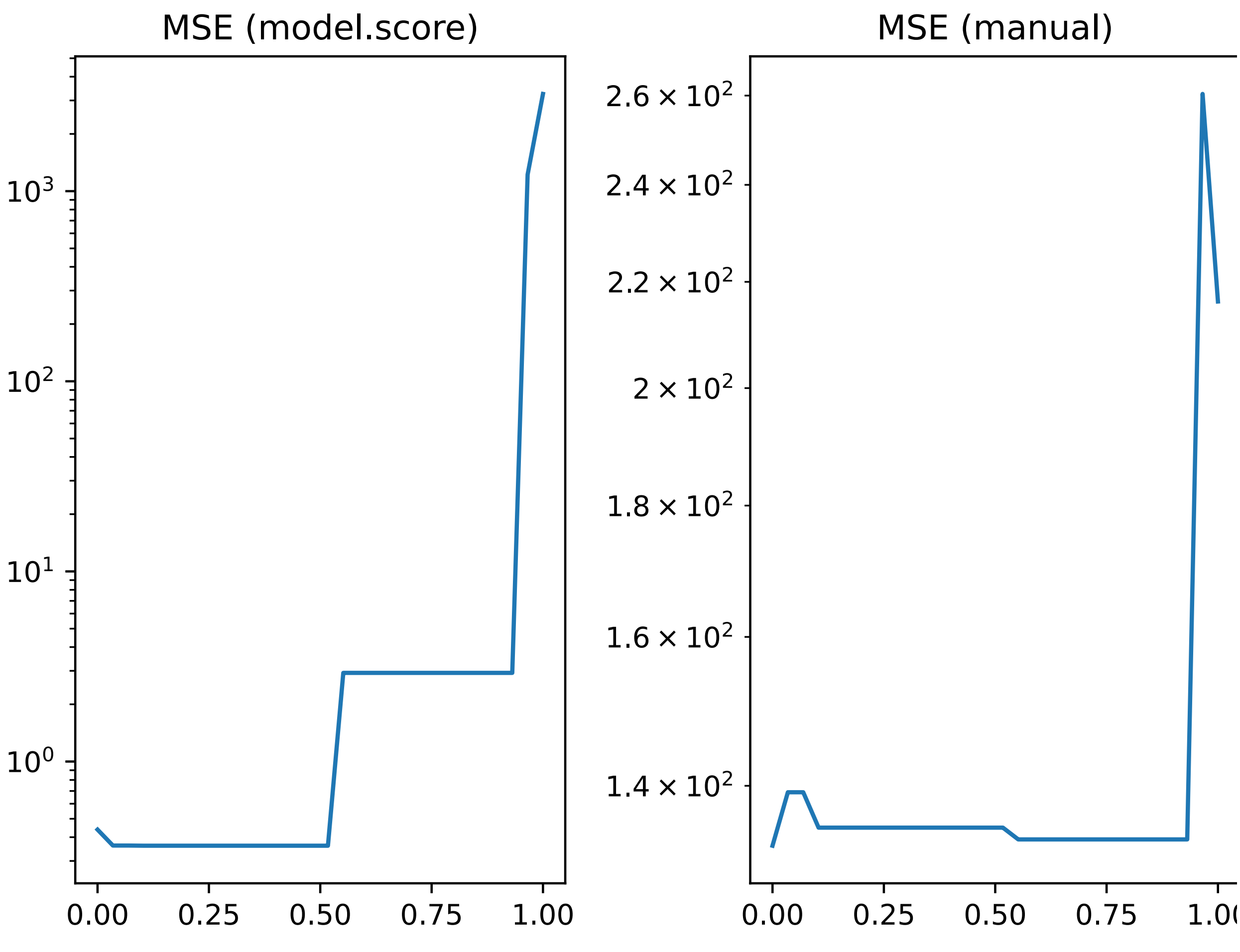
```
mse_sim = np.zeros(len(threshold_scan))
mse = np.zeros(len(threshold_scan))
for i, threshold in enumerate(threshold_scan):
    sparse_regression_optimizer = ps.STLSQ(threshold=threshold)
    model = ps.SINDy(
        feature_names=feature_names,
        optimizer=sparse_regression_optimizer
    )
    model.fit(x_train_noise, t=dt, quiet=True)

    x_test_sim = model.simulate(y0_test, t=ts)
    mse[i] = model.score(x_test, ts, metric=metrics.mean_squared_error)
    mse_sim[i] = np.mean((x_test - x_test_sim) ** 2)
```

In [ ]:
```
fig, axs = plt.subplots(1,2)
fig.set_tight_layout(True)

axs[0].semilogy(threshold_scan, mse)
axs[0].set_title("MSE (model.score)")
axs[1].semilogy(threshold_scan, mse_sim)
axs[1].set_title("MSE (manual)")
```

Out[ ]: `Text(0.5, 1.0, 'MSE (manual)')`



The big spike towards one is a consequence of the many of the terms in the last two equation diaspere. Cruriosly the two methods of calculating the error does not give the same answer. The left (Based on PySINDy) give the best threshold ≈ 0.25. The right (calculating the MSE manually) give the best threshold at ≈ 0.75.

The below model shows what happens when threshold is set to 1. It can be seen that many of the terms is not pressent in the model:

In [ ]:
```
feature_names = ['x', 'y', 'z']
sparse_regression_optimizer = ps.STLSQ(threshold=1)
model = ps.SINDy(feature_names=feature_names, optimizer=sparse_regression
model.fit(x_train_noise, t=dt)
model.print()
```

```
(x)' = -9.952 x + 9.960 y
(y)' = -2.824 x
(z)' = 0.000
```

```
/Users/louiss/code/uni/master/SML/.venv/lib/python3.11/site-packages/pysin
dy/optimizers/stlsq.py:201: UserWarning: Sparsity parameter is too big (1)
and eliminated all coefficients
  warnings.warn(
```

According to the the left plot the best solution is found when the threshold is set to ca: 0.25:

In [ ]:
```
feature_names = ['x', 'y', 'z']
sparse_regression_optimizer = ps.STLSQ(threshold=0.25)
model = ps.SINDy(feature_names=feature_names, optimizer=sparse_regression
model.fit(x_train_noise, t=dt)
model.print()
```

```
(x)' = -9.952 x + 9.960 y
(y)' = 27.568 x + -0.883 y + -0.988 x z
(z)' = -2.665 z + 0.998 x y
```

According to the the left plot the best solution is found when the threshold is set to ca: 0.75:

In [ ]:
```
feature_names = ['x', 'y', 'z']
sparse_regression_optimizer = ps.STLSQ(threshold=0.75)
model = ps.SINDy(feature_names=feature_names, optimizer=sparse_regression
model.fit(x_train_noise, t=dt)
model.print()
```

```
(x)' = -9.952 x + 9.960 y
(y)' = 25.415 x + -0.946 x z
(z)' = -2.665 z + 0.998 x y
```

With the true system being:

$$(x)' = -10x + 10y$$

$$(y)' = 10x + -xz - y$$

$$(z)' = -2.667z + xy$$

the best model is seen to be with ≈ 0.25. I dont know why my manual MSE gives the wrong answer.

The conclusion for this experiment is that PySINDy is able to find the governing equations for a dynamical system on the conditions that there are not to much noise, that there is enough data and that the threshold is set correctly.

(E.7.7)

In this experiment I tried to use the PDE module in PySINDy. After a long time trying to fix issues I unfortunately had to give up.

$$u_t + u_{xxx} - 6uu_z = 0$$

$$u(x,t) = -\frac{c}{2} sech^2 \left(\frac{\sqrt{c}}{2} (x - ct) \right)$$

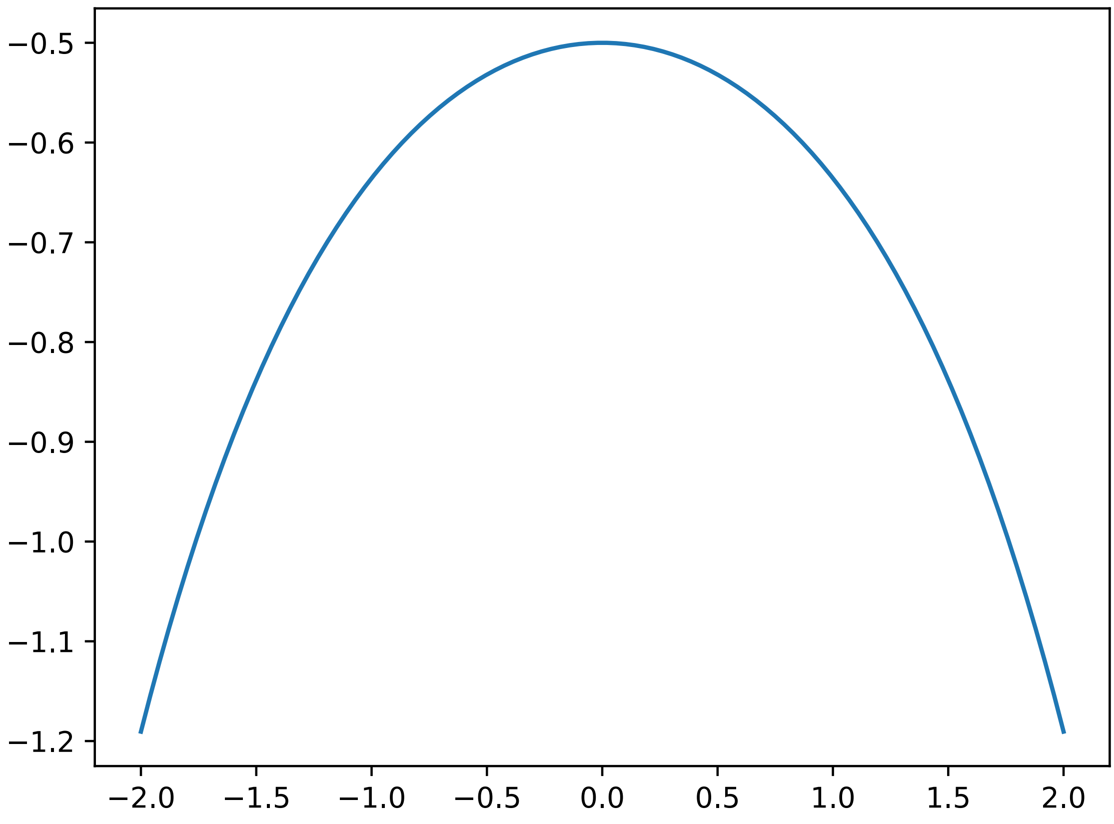
In [ ]:
```
sech = lambda x: np.cosh(x)

def pde(x,t,c):
    return -(c/2)*(sech((np.sqrt(c) / 2)*(x-c*t))**2)
```

In [ ]:
```
import pysindy as ps
ts = np.linspace(-2,2,100)
xs = pde(ts, 0, 1)
us = ps.AxesArray(np.ones((len(xs) * len(ts), 2)),{"ax_coord":1})

plt.plot(ts, xs)
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x127f07690>]
```



In [ ]:
```
library_functions = [lambda x: x]
pde_lib = ps.PDELibrary(
    library_functions=library_functions,
    derivative_order=2,
    spatial_grid=xs,
).fit([us])

optimizer = ps.STLSQ()

model = ps.SINDy(feature_library=pde_lib, optimizer=optimizer)
model.fit(xs, t=ts)
# model.print()
```

```
---------------------------------------------------------------------------
LinAlgError                               Traceback (most recent call last)
Cell In[16], line 11
      8 optimizer = ps.STLSQ()
     10 model = ps.SINDy(feature_library=pde_lib, optimizer=optimizer)
---> 11 model.fit(xs, t=ts)
     12 # model.print()

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/pysindy/pysindy.py:414, in SINDy.fit(self, x, t, x_dot, u, multiple_trajectories, unbias, quiet, ensemble, library_ensemble, replace, n_candidates_to_drop, n_subset, n_models, ensemble_aggregator)
    412     warnings.filterwarnings(action, category=LinAlgWarning)
    413     warnings.filterwarnings(action, category=UserWarning)
```

```
--> 414     self.model.fit(x, x_dot)
    416 # New version of sklearn changes attribute name
    417 if float(__version__[:3]) >= 1.0:

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/sklearn/base.py:1474, in _fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/sklearn/pipeline.py:471, in Pipeline.fit(self, X, y, **params)
    428 """Fit the model.
    429 
    430 Fit all the transformers one after the other and sequentially transform the
   (...)
    468     Pipeline with fitted steps.
    469 """
    470 routed_params = self._check_method_params(method="fit", props=params)
--> 471 Xt = self._fit(X, y, routed_params)
    472 with _print_elapsed_time("Pipeline", self._log_message(len(self.steps) - 1)):
    473     if self._final_estimator != "passthrough":

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/sklearn/pipeline.py:408, in Pipeline._fit(self, X, y, routed_params)
    406     cloned_transformer = clone(transformer)
    407 # Fit or load from cache the current transformer
--> 408 X, fitted_transformer = fit_transform_one_cached(
    409     cloned_transformer,
    410     X,
    411     y,
    412     None,
    413     message_clsname="Pipeline",
    414     message=self._log_message(step_idx),
    415     params=routed_params[name],
    416 )
    417 # Replace the transformer of the step with the fitted
    418 # transformer. This is necessary when loading the transformer
    419 # from the cache.
    420 self.steps[step_idx] = (name, fitted_transformer)

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/joblib/memory.py:312, in NotMemorizedFunc.__call__(self, *args, **kwargs)
    311 def __call__(self, *args, **kwargs):
--> 312     return self.func(*args, **kwargs)
```

```
File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/sklearn/pipeline.py:1303, in _fit_transform_one(transformer, X, y, weight, message_clsname, message, params)
   1301 with _print_elapsed_time(message_clsname, message):
   1302     if hasattr(transformer, "fit_transform"):
-> 1303         res = transformer.fit_transform(X, y, **params.get("fit_transform", {}))
   1304     else:
   1305         res = transformer.fit(X, y, **params.get("fit", {})).transform(
   1306             X, **params.get("transform", {})
   1307         )

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/sklearn/utils/_set_output.py:295, in _wrap_method_output.<locals>.wrapped(self, X, *args, **kwargs)
    293 @wraps(f)
    294 def wrapped(self, X, *args, **kwargs):
--> 295     data_to_wrap = f(self, X, *args, **kwargs)
    296     if isinstance(data_to_wrap, tuple):
    297         # only wrap the first output for cross decomposition
    298         return_tuple = (
    299             _wrap_data_with_container(method, data_to_wrap[0], X, self),
    300             *data_to_wrap[1:],
    301         )

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/sklearn/base.py:1101, in TransformerMixin.fit_transform(self, X, y, **fit_params)
   1098     return self.fit(X, **fit_params).transform(X)
   1099 else:
   1100     # fit method of arity 2 (supervised transformation)
-> 1101     return self.fit(X, y, **fit_params).transform(X)

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/sklearn/utils/_set_output.py:295, in _wrap_method_output.<locals>.wrapped(self, X, *args, **kwargs)
    293 @wraps(f)
    294 def wrapped(self, X, *args, **kwargs):
--> 295     data_to_wrap = f(self, X, *args, **kwargs)
    296     if isinstance(data_to_wrap, tuple):
    297         # only wrap the first output for cross decomposition
    298         return_tuple = (
    299             _wrap_data_with_container(method, data_to_wrap[0], X, self),
    300             *data_to_wrap[1:],
    301         )

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/pysindy/feature_library/base.py:191, in x_sequence_or_item.<locals>.func(self, x, *args, **kwargs)
    189 if isinstance(x, Sequence):
    190     xs = [AxesArray(xi, comprehend_axes(xi)) for xi in x]
```

```
--> 191     result = wrapped_func(self, xs, *args, **kwargs)
    192     if isinstance(result, Sequence):  # e.g. transform() returns x
    193         return [AxesArray(xp, comprehend_axes(xp)) for xp in result]

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/pysindy/feature_library/pde_library.py:429, in PDELibrary.transform(self, x_full)
    422         s[axis] = slice(self.spatiotemporal_grid.shape[axis])
    423         s[-1] = axis
    425         derivs = self.differentiation_method(
    426             d=multiindex[axis],
    427             axis=axis,
    428             **self.diff_kwargs,
--> 429         )._differentiate(derivs, self.spatiotemporal_grid[tuple(s)])
    430 library_derivatives[
    431     ..., library_idx : library_idx + n_features
    432 ] = derivs
    433 library_idx += n_features

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/pysindy/differentiation/finite_difference.py:251, in FiniteDifference._differentiate(self, x, t)
    249             interior = interior + x[tuple(s)] * coeffs[i]
    250 else:
--> 251     coeffs = self._coefficients(t)
    252     interior = self._accumulate(coeffs, x)
    253 s[self.axis] = slice((self.n_stencil - 1) // 2, -(self.n_stencil - 1) // 2)

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/pysindy/differentiation/finite_difference.py:102, in FiniteDifference._coefficients(self, t)
    100 b = np.zeros(self.n_stencil)
    101 b[self.d] = np.math.factorial(self.d)
--> 102 return np.linalg.solve(matrices, [b])

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/pysindy/utils/axes.py:98, in AxesArray.__array_function__(self, func, types, args, kwargs)
     96 def __array_function__(self, func, types, args, kwargs):
     97     if func not in HANDLED_FUNCTIONS:
---> 98         arr = super(AxesArray, self).__array_function__(func, types, args, kwargs)
     99         if isinstance(arr, np.ndarray):
    100             return AxesArray(arr, axes=self.__dict__)

File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/numpy/linalg/linalg.py:409, in solve(a, b)
    407 signature = 'DD->D' if isComplexType(t) else 'dd->d'
    408 extobj = get_linalg_error_extobj(_raise_linalgerror_singular)
--> 409 r = gufunc(a, b, signature=signature, extobj=extobj)
    411 return wrap(r.astype(result_t, copy=False))
```

```
File ~/code/uni/master/SML/.venv/lib/python3.11/site-packages/numpy/linal
g/linalg.py:112, in _raise_linalgerror_singular(err, flag)
    111 def _raise_linalgerror_singular(err, flag):
--> 112     raise LinAlgError("Singular matrix")

LinAlgError: Singular matrix
```