Readme for operator inference package

Renee Swischuk

May 29, 2019

Section 1 Operator inference package

Scripts: `opinf_demo.py`
Dependencies: operator_inference package

The first step is to download the operator inference module from pip. In the command prompt, type
`pip3 install -i https://test.pypi.org/simple/ operator-inference`
This is temporary

The `operator_inference` package contains a `model` class within the OpInf module, with functions defined in Section 1.1.1, and two helper scripts called `opinf_helper.py` and `integration_helpers.py`, with functions defined in Section 1.2.1 and Section 1.3.1, respectively.

Section 1.1 Model class

The following commands will initialize an operator inference model.

```
from operator_inference import OpInf
my_model = OpInf.model(degree, input)
```

where `degree` is the degree of the model with the following options

‘L’ – a linear model, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$

‘Lc’ – a linear model with a constant, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{c}$

‘LQ’ – a linear and quadratic model, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{F}\mathbf{x}^2$

‘LQc’ – a linear and quadratic model with a constant, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{F}\mathbf{x}^2 + \mathbf{c}$

‘Q’ – a quadratic model, $\dot{\mathbf{x}} = \mathbf{F}\mathbf{x}^2$

‘Qc’ – a quadratic model with a constant, $\dot{\mathbf{x}} = \mathbf{F}\mathbf{x}^2 + \mathbf{c}$

The `input` argument is a boolean (True or False) denoting whether or not there is an additive input term of the form $+\mathbf{BU}$.

The script, `opinf_demo.py` demonstrates the use of the operator inference model on data generated from the heat equation. See [?] for the problem setup.

Section 1.1.1 Model class functions

Functions can be called as `mymodel.function_name()`

1. `fit(r,reg,xdot,xhat,u=None)`
   Find the operators of the reduced-order model that fit the data
   Parameters:

   r – (integer) POD basis size

   reg – (float) L_2 regularization parameter. For no regularization, set to 0.

   xdot – ($r \times n_t$ array) the reduced time derivative data

   xhat– ($r \times n_t$ array) the reduced snapshot data

   u – ($p \times n_t$ array, optional) the input, if `model.input = True`

   Returns:

   None

2. `predict(init, n_timesteps, dt, u = None)`
   Simulate the learned model with a Runge Kutta scheme
   Parameters:

   init – ($r \times 1$) intial reduced state

   n_timesteps – (int) number of time steps to simulate

   dt– (float) the time step size

   u – ($p\times$ n_timesteps array) the input at each simulation time step

   Returns:

   projected_state – ($r\times$ n_timesteps array) the simulated, reduced states

   i – (int) the time step that the simulation ended on ($i <$n_timesteps only if NaNs occur in simulation)

3. `get_residual()`
   Get the residuals of the least squares problem
   Parameters:

   None

   Returns:

   residual – (float) residual of data fit, $\|\mathbf{D}\mathbf{O}^T - \dot{\mathbf{X}}^T\|_2^2$

   solution – (float) residual of the solution, $\|\mathbf{O}^T\|_2^2$

4. `get_operators()`
   Get the learned operators
   Parameters:

None

Returns:

ops – (tuple) containing each operator (as an array) as defined by `degree` of the model

Section 1.2 `opinf_helper.py`

Import the opinf helper script as
`from operator_inference import opinf_helper`.

Section 1.2.1 `opinf_helper.py` functions

The following functions are supported and called as `opinf_helper.function_name()`.

1. `normal_equations(D,r,k,num)`
   Solves the normal equations corresponding to the regularized least squares problem $\min_{\mathbf{o}_i} \|\mathbf{D}\mathbf{o}_i - \mathbf{r}_i\|_2^2 + k\|\mathbf{P}\mathbf{o}_i\|_2^2$
   Parameters:

   D – (nd array) data matrix

   r – (nd array) reduced time derivative data

   k – (float) regularization parameter

   num – (int) number of ls problem we are solving [1..r]

   Returns:

   $\mathbf{o}_i$ – (nd array) the solution to the least squares problem

2. `get_x_sq(X)`
   Compute squared snapshot data as in [**?**].
   Parameters:

   X – ($n_t \times r$ array) reduced snapshot data (transposed)

   Returns:

   X^2 – ($n_t \times \frac{r(r+1)}{2}$ array) reduced snapshot data squared without redundant terms.

3. `F2H(F)`
   Convert quadratic operator $\mathbf{F}$ to symmetric quadratic operator $\mathbf{H}$ for simulating the learned system.
   Parameters:

   F – ($r \times \frac{r(r+1)}{2}$ array) learned quadratic operator

   Returns:

   H – ($r \times r^2$ array) symmetric quadratic operator

Section 1.3 `integration_helpers.py`

Import the integration helper script as
`from operator_inference import integration_helpers`.

Section 1.3.1 `integration_helpers.py` functions

The following functions are supported and called as `integration_helpers.function_name()`.

1. `rk4advance_L(x,dt,A,B=0,u=0)`
   One step of 4th order runge kutta integration of a system of the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$

   Parameters:

   x – ($r \times 1$ array) current reduced state

   dt – (float) time step size

   A – ($r \times r$ array) linear operator

   B – ($r \times p$ array, optional default $= 0$) input operator (only needed if `input = True`).

   u – ($p \times 1$ array, optional default $= 0$) the input at the current time step (only needed if `input = True`).

   Returns:

   x – ($r \times 1$ array) reduced state at the next time step

2. `rk4advance_Lc(x,dt,A,c,B=0,u=0)`
   One step of 4th order runge kutta integration of a system of the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{c}$

   Parameters:

   x – ($r \times 1$ array) current reduced state

   dt – (float) time step size

   A – ($r \times r$ array) linear operator

   c – ($r \times 1$ array) constant term

   B – ($r \times p$ array, optional default $= 0$) input operator (only needed if `input = True`).

   u – ($p \times 1$ array, optional default $= 0$) the input at the current time step (only needed if `input = True`).

   Returns:

   x – ($r \times 1$ array) reduced state at the next time step

3. `rk4advance_LQ(x,dt,A,H,B=0,u=0)`
   One step of 4th order runge kutta integration of a system of the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{H}(\mathbf{x} \otimes \mathbf{x})$

   Parameters:

   x – ($r \times 1$ array) current reduced state

   dt – (float) time step size

   A – ($r \times r$ array) linear operator

   H – ($r \times r^2$ array) quadratic operator

   B – ($r \times p$ array, optional default $= 0$) input operator (only needed if `input = True`).

u – ($p \times 1$ array, optional default = 0) the input at the current time step (only needed if `input = True`).

Returns:

x – ($r \times 1$ array) reduced state at the next time step

4. `rk4advance_LQc(x,dt,A,H,c,B=0,u=0)`
One step of 4th order runge kutta integration of a system of the form $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{H}(\mathbf{x} \otimes \mathbf{x}) + \mathbf{c}$

Parameters:

x – ($r \times 1$ array) current reduced state

dt – (float) time step size

A – ($r \times r$ array) linear operator

H – ($r \times r^2$ array) quadratic operator

c – ($r \times 1$ array) constant term

B – ($r \times p$ array, optional default = 0) input operator (only needed if `input = True`).

u – ($p \times 1$ array, optional default = 0) the input at the current time step (only needed if `input = True`).

Returns:

x – ($r \times 1$ array) reduced state at the next time step

5. `rk4advance_Q(x,dt,H,B=0,u=0)`
One step of 4th order runge kutta integration of a system of the form $\dot{\mathbf{x}} = \mathbf{H}(\mathbf{x} \otimes \mathbf{x})$

Parameters:

x – ($r \times 1$ array) current reduced state

dt – (float) time step size

H – ($r \times r^2$ array) quadratic operator

B – ($r \times p$ array, optional default = 0) input operator (only needed if `input = True`).

u – ($p \times 1$ array, optional default = 0) the input at the current time step (only needed if `input = True`).

Returns:

x – ($r \times 1$ array) reduced state at the next time step

6. `rk4advance_Qc(x,dt,H,c,B=0,u=0)`
One step of 4th order runge kutta integration of a system of the form $\dot{\mathbf{x}} = \mathbf{H}(\mathbf{x} \otimes \mathbf{x}) + \mathbf{c}$

Parameters:

x – ($r \times 1$ array) current reduced state

dt – (float) time step size

H – ($r \times r^2$ array) quadratic operator

c – ($r \times 1$ array) constant term

B – ($r \times p$ array, optional default = 0) input operator (only needed if `input = True`).

u – ($p \times 1$ array, optional default = 0) the input at the current time step (only needed if `input = True`).

Returns:

x – ($r \times 1$ array) reduced state at the next time step