Readme for operator inference on GEMS data

Renee Swischuk

May 29, 2019

Section 1 Preprocessing, lifting and scaling GEMS data

Scripts: `tec2h5.py`
Dependencies: `chemistry_conversions.py, scaling_tools.py`

The script *tec2h5.py* does the following

- Read in GEMS .dat files from the folder *OriginalGEMS*
- Convert mass fractions to molar concentrations
- Replace temperature with specific volume
- Stack the variables vertically and save the numVars*numElements $\times$ numSnaps dataset as *data_unscaled.h5* in the folder *OpInfdata*
- Scale to the range [-1,1]
- Save the scaled data as *data_minmax.h5* in the folder *OpInfdata*

To access either of the datasets, run the following commands:

```
import h5py
data_file = h5py.File('data_minmax.h5','r')
dataset = data_file['data'][:,:]
```

Section 2 Computing POD basis, reducing data and computing xdot

---

Scripts: `projection_helpers.py`
Dependencies:

Run the script `projection_helpers.py`.

1) You will be asked for the number of snapshots you want to compute the svd of/reduce.
Type this number in and click enter.

2) You will be asked if you want to compute the SVD.
Type True and click enter if you want to compute the SVD of the data (the data will be read from the file *OpInfdata/data_minmax.h5*).

2a) You will then be prompted for the truncation value for the randomized SVD algorithm.
Type this number in and click enter.

The SVD will be saved to the file *OpInfdata/svd_nt%d.h5* where %d = number of training snapshots you entered. There will be two datasets in the h5 file, one for the singular vectors, called 'U', and one for the singular values, called 'S'

Type False and click enter if you already have the SVD stored in *OpInfdata/svd_nt%d.h5* with datasets 'U' and 'S' for singular vectors and values.

After the prompts, the script `projection_helpers.py` will do the following:

- Load the scaled data from *OpInfdata/data_minmax.h5*)
- (optionally) Compute the SVD of the data and save
- Compute the time derivative of the scaled data
- Project data and time derivative onto POD basis of size r – multiple values for r can be given and must be input directly in the code at line 263 in the `main()` function.
- Save the projected data and xdot into files
  OpInfdata/reducedData/data_reduced_minmax_nt%d/data_reduced_%d.h5 %(nt,r) and
  OpInfdata/reducedData/xdot_reduced_minmax_nt%d/xdot_reduced_%d.h5 %(nt,r)

  The reduced datasets are read with the following commands:
  `df= h5py.File('OpInfdata/reducedData/data_reduced_minmax_nt5000/data_reduced_8.h5','r')`
  `data = df['data'][:,:]` – numVars*numElements × r array
  `xf = h5py.File('OpInfdata/reducedData/xdot_reduced_minmax_nt5000/xdot_reduced_8.h5','r')`
  `xdot = xf['xdot'][:,:]` – numVars*numElements × r array

  The SVD data is read with the following commands:
  `svd_file = h5py.File('svd_nt5000.h5','r')`
  `singular_values = svd_file['S']`
  `singular_vectors = svd_file['U']`

Section 3 Operator inference package

Scripts: `opinf_demo.py`
Dependencies: operator_inference package – see https://test.pypi.org/project/operator-inference/

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, with functions defined in **??**, and two helper scripts called `opinf_helper.py` and `integration_helpers.py`, with functions defined in **??** and **??**, respectively.

Section 3.1 Quick Start

```
from operator_inference import OpInf

#define the model
mymodel = OpInf.model('Lc',False) # a linear quadratic with no input

#fit the model
mymodel.fit(r,k,xdot,xhat)

#simulate the model for train and test time steps
xr,break_point = mymodel.predict(xhat[:,0], n_t, dt)

#reconstruct the predictions
xr_rec = U[:,:r]@xr
```

Section 3.2 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 a string denoting 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 $+\mathbf{B}\mathbf{U}$.

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 3.2.1 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 3.3 `opinf_helper.py`

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

Section 3.3.1 `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 3.4 `integration_helpers.py`

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

Section 3.4.1 `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

Section 4 Operator inference on GEMS data

Scripts: `main.py`
Dependencies: operator_inference package

The script `main.py` performs operator inference on the GEMS data, simulates the learned system and plots some results. There are a number of user inputs that must be defined within the script. They are as follows

k_ridge – (list of floats) The value or values to use for regularization parameter. If multiple values are given, the script loops, generating results for each value.

r_vals – (list of ints) The POD basis size or sizes. If multiple values are given, the script loops, generating results for each value. You must have an SVD dataset and a reduced dataset corresponding to each r_val.

train_time – (int) Number of snapshots for learning the operators.

forecast_time – (int) Number of time steps to predict. The script will predict train_time + forecast_time time steps.

plot_svd – (bool) True = plot singular value decay. False = don't plot.

compute_species_integral – (bool) True = compute sum of each species at each time step and save to file. False = Dont compute sum

plot_forecast – (bool) True = plot the time trace of predictions at element_to_plot locations in the domain. You can also save these plots by uncommenting the line in the script ($\approx$ line 350 and 351). False = dont plot it or save it.

element_to_plot – (list of ints between 0 and 38523) Which element to plot time trace for.

output_filename – (string) If you decide to save the time traces, this is the filename to save them to.

save_forecast – (bool) True = save data for prediction in text file. False = dont save data

save_data – (bool) True = save the predicted, full dimensional state vector (p,u,v,T,ch4 molar, o2 molar, co2 molar, h2o molar), and error at time step time_to_save. False = dont save.

time_to_save – (int) The time step to save, must be less than train_time + forecast_time. Currently can only accept one time step to save at a time.

monitor_unphysical_values – (bool) True = count the negative values at each time step occuring in each full dimensional variable and save to "Unphysical_Values_Timestep.txt". False = don't do this.

datafilefolder – (string) The folder containing the projected data. Files inside folder should be called "data_reduced_%d.h5" %d is the POD basis size. Should have datasets 'data'. This is the format used by `projection_helpers.py`.

xdotfilefolder – (string) The folder containing the projected xdot. Files inside folder should be called "xdot_reduced_%d.h5" %d is the POD basis size. Should have dataset 'xdot'. This is the format used by `projection_helpers.py`.

fulldatapath – (string) Full filepath for the full dimensional data, should be in dataset 'data' in an hdf5 file.

svd_filepath – (string) Full filepath for the SVD, should be in a data set 'U' and 'S' in an hdf5 file.

Once these inputs are chosen, simply run the file from the command prompt as
`python3 main.py`

The code also outputs the timing for the steps of operator inference.

Section 5 Preparing results for tecplot

Scripts: `csv2tec.py`
Dependencies: `save_as_tec.m`, `importGridFile.m`, `importTecASCIIdata.m`, `OutputTecASCIIdata.m` `grid.dat`, matlab package

If save_data == True when you run `main.py`, then you can plot the resulting field plots in tecplot. You will need TecPlot 360.

Install the matlab package, so we can call the matlab formatting scripts. In the command line type

```
pip install matlab
```

You must install this using python2's pip.

In the script `csv2tec.py`, there is a main function with some user inputs you must define corresponding to the data you saved. Once thats done, save the csv results files to a tecplot formatted .dat file by typing the command

```
python csv2tec.py
```

This must be run using python 2

This will produce .dat files with the same name plus an additional "_formatted". In TecPlot, go to File >> Load Data >> Choose the file >> Choose Tecplot Data Loader >> OK.

Once the domain is visible on the screen, View >> Plot Sidebar. On the side bar, select the contour box and click details to choose different variables and colorings.

References

[1] Peherstorfer, Benjamin and Willcox, Karen, Data-driven operator inference for nonintrusive projection-based model reduction, *Computer Methods in Applied Mechanics and Engineering*, volume 306, pages 196–215, 2016

[2] Swischuk, Renee, Physics-based machine learning and data-driven reduced-order modeling, *MIT Master's Thesis*, 2019