ME964: Scientific Computing & Machine Learning
Final Project

Consider 1D Burgers equation given by:

$$\begin{aligned} &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \frac{1}{Re}\frac{\partial^2 u}{\partial x^2}, \quad x \in (0,1) \\ &u(0,t) = 0, \quad u(1,t) = 0, \quad t \geq 0 \\ &u(x,0) = \frac{x}{1+\sqrt{\frac{1}{t^o}}\exp(Re\frac{x^2}{4})}, \quad x \in (0,1) \end{aligned} \tag{1}$$

where Re is the Reynolds number and the parameter $t^o = \exp(\frac{Re}{8})$.

The analytical solution of Eq. (1) is given by:

$$u(x,t) = \frac{\frac{x}{t+1}}{1+\sqrt{\frac{t+1}{t^o}}\exp(Re\frac{x^2}{4t+4})} \, . \tag{2}$$

The analytical solution in Eq. (2) can be used to generate training data and validate your results.

Problem 1: Solving PDE by deep learning

a) An example code (`burger.py`) is provided to solve the PDE in Eq. (1) using neural network. You can adapt the code to get solutions at different values of $Re = 1, 50, 100, 300$. Use the analytical solution to validate your results. (**40 Points**)

b) Solve the combined inverse-forward problem, assuming the value of Re in Eq. (1) is unknown. (**20 Points**)

To work on this project, you may need to install the following packages:

Packages needed:

- Matplotlib (`https://matplotlib.org/`)
- Numpy (`https://numpy.org/`)
- scikit-learn (`https://scikit-learn.org/stable/`)
- scikit-optimize (`https://scikit-optimize.github.io/stable/`)
- Scipy (`https://scipy.org/`)
- And one of the two following deep learning packages:
  1. TensorFlow 2.x (`https://www.tensorflow.org/`)
  2. PyTorch (`https://pytorch.org/`)
- DeepXde (`https://github.com/lululxvi/deepxde` and `https://deepxde.readthedocs.io/en/latest/`)

Installation: Generally, you can run:
`pip install package's name`
in the command line to install all the packages above. For example:
`pip install deepxde`
to install the Deepxde package.

You can also find the details about installation and other user documents from the websites provided above.

Problem 2: Data-driven Reduced Order Modeling

Use the analytical solution of the above 1D burgers equation at $Re = 100$ as training data, $u(t_i, x_j)$ at time t_i and spatial position x_j, with $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. We can express $u(t_i, x_j)$ at different time instances and spatial positions as a matrix $\mathbf{A}$ with $A_{ij} = u(t_i, x_j)$. Computing the SVD of $\mathbf{A}$ leads to:

$$\mathbf{A} \approx \mathbf{A}_k = \sum_{q=1}^{k} S_q \mathbf{u}_q \otimes \mathbf{v}_q \ . \tag{3}$$

$$\epsilon = \sqrt{\frac{\sum_{q=k+1}^{m} S_q^2}{\sum_{q=1}^{k} S_q^2}} \ . \tag{4}$$

By requiring the error $\epsilon \leq \alpha^{\text{SVD}}$ with α^{SVD} the preset threshold (e.g., $\alpha^{\text{SVD}} = 0.01$), we can determine the value of k. (Just for your reference, from our preliminary studies, requiring $\epsilon \leq 0.01$ leads to $k = 2$ for this problem.)

Thus, we have

$$u(t_i, \mathbf{x}_j) \approx \sum_{q=1}^{k} S_q u_q(t_i) v_q^T(x_j) \ .$$

Now let's do regression for the discrete data $u_q(t_i)$ using Gaussian process regression (GPR), we obtain the following reduced-order model (ROM):

$$u(t, x_j) \approx u^{\text{ROM}}(t, x_j) = \sum_{q=1}^{k} S_q \hat{u}_q(t) v_q(x_j) \ . \tag{5}$$

Eq. (5) provides an efficient way to determine the full-order solution $u(t, x_j)$ at any given time t and for all grid points x_j.

When the numerical simulations or experimental measurements are demanding or expensive for a dynamical system, one can employ the ROM to forecast the full-order solutions at future times beyond the database range. However, the furthest forecast time is constrained by the predictive capability of the ROM. Thus, after the furthest forecast time of the ROM is reached, the consecutive prediction would still need numerical simulations or experimental measurements. Thus, for long-time prediction of a dynamical system, the numerical simulations/experimental measurements and the ROM can be called alternatively in an automated process:

1. Generate m snapshot (training) data (i.e., full-order solutions from t_1 to t_m). For example, with the time step $\Delta t = 10^{-2}$ and spatial grid length $\Delta x = 10^{-3}$ and $m = 20$.

2. Construct a ROM from the m snapshot data and determine the furthest permissible forecast time t^* as follows. (**20 points**)

   Since the standard deviation $\sigma_q(t')$ for the GPR model $\hat{u}_q(t)$ grows when the forecast time t' goes further beyond t_m, and hence it poses a constraint on the furthest forecast time t^* permitted by GPR. Considering there are k GPR models, we can take an average of the standard deviations as $\bar{\sigma}(t') = \frac{1}{k} \sum_1^k \hat{\sigma}_q(t')$. We require $\bar{\sigma}(t') \leq \sigma_{\text{tolerance}}$ until t^*. Then, t^* would be the furthest permissible forecast time.

   If you have difficulty of using GPR, you can instead use the least squares method for regression, then determine l empirically from your experimental experiments such that $t^* = l\Delta t$ and keep using the same l in the following steps.

3. Call the ROM to predict the full-order solutions until $t = t^*$. The full-order solution at t^* is taken as the initial condition for obtaining the consecutive full-order solutions.

4. Repeat Steps 1-3 until the target prediction time. (**20 points**)

This process adaptively combines full-order solutions (usually obtained from expensive numerical simulations or experimental measurements) with ROMs and hence optimizes the efficiency for long-time prediction of a dynamical system. Apply this methodology to make a long-time prediction for the solution of the above 1D Burgers equation at $Re = 100$ until $T = 2.0$.