Homework Set #7 – PHYS 6260
Prof. John Wise & GTA Alisha Vira
Due Wednesday, March 27th, 11:59pm (Submit github URL to Canvas; all code on github)

- Your assignment should be uploaded as a **Jupyter notebook** for both Problems.
- Please use the template notebook uploaded on Canvas and github as a starter.
- Comment your code through inline comments with # or markdown blocks, where the latter option is preferred.
- In the problem descriptions, "programs" are referring to single or multiple code blocks in a notebook.
- The materials that you are required to include are indicated at the end of each problem, next to the check symbol: ☑

---

1. **Physics Informed Neural Networks (80 points total):** Here you will complete a code that solves Burger's Equation with a traditional finite differencing PDE solver and then use that solution to train a Physics Informed Neural Network (PINN) to approximate the solution. We have provided most of the code in both approaches in the `HW7.ipynb` notebook, and you will complete the missing sections marked with `COMPLETE HERE`. If you desire, you may change other portions of the code and restructure it, but this is not necessary to complete the problem.

Burger's Equation is a fundamental non-linear PDE that has many applications in fluid dynamics, traffic flow, and applied mathematics, to name a few. We will consider the 1D case of its viscous version

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0, \tag{1}$$

where ν is the kinematic viscosity or diffusion coefficient. In the application of fluid flow, the field $u(x,t)$ describes the fluid velocity at some position x and time t. It is usually the prototypical PDE that one solves when testing hydrodynamics solvers before moving to more realistic PDEs, i.e. Euler's and Navier-Stokes equations. In its evolution, any smooth fluid flow will develop a discontinuity, approximating a shock.

(a) (10 points) In the `BurgerSolver` class, complete the missing code in the `solver` routine to implement the Forward Time, Upwind Space (FTUS) method that we used in Homework Set #4. Recall that it is a stable method for the advection equation, where any first spatial derivative is always evaluated "upwind," that is in the direction of the velocity, which is u for Burger's Equation,

$$\frac{\partial u}{\partial x} = \frac{1}{\Delta x}\left(u_i^n - u_{i-1}^n\right) \quad (u > 0) \tag{2}$$

or

$$\frac{\partial u}{\partial x} = \frac{1}{\Delta x}\left(u_{i+1}^n - u_i^n\right) \quad (u < 0), \tag{3}$$

for the i^{th} cell at timestep n. For the second derivative term, you can use a first-order accurate finite difference.

Given an initial condition of $u(x,0) = -\sin(\pi x)$, boundary conditions of $u(0,t) = 0$, a domain $[-1,1]$ that is discretized into $N = 1000$ points, and $\nu = 0.01$, evolve the system for 1 second.

- ☐ Plot $u(x)$ at a time $t = 0.2$ s and $t = 1$ s.
- ☐ Plot the space-time diagram $u(x,t)$ as an image colored by the values of u, which will be the basis for the PINN in the remaining questions.

(b) (20 points) Here we will be training a neural network with one hidden layer on the training set, provided by the solution in part (a). The `BurgerSolver.random_sample()` routine returns N random solutions u within the simulation (x,t) domain. The provided `NeuralNetwork` class is based on the one showed during lectures. Using a loss function provided in the "exact" method in `set_loss_function`,

$$\mathcal{L}_{\text{exact}} = z - y, \tag{4}$$

where z and y are the predicted and true values, respectively, train the neural network with 10,000 random data samples in 50 epochs with a learning rate $\eta = 0.1$ for $k = 2, 5, 10, 40$, and 100 nodes in the hidden layer.

- ☐ Plot the loss curve as a function of epochs for these values of k, which is returned by the `solve` routine. You should plot all five curves on a single plot.
- ☐ Describe if the model is adequately converging during the training and the differences between the models with different values of k.
- ☐ Create an image plot of the predicted solution $\hat{u}(x,t)$ with $k = 40$, using the `predict` routine in the `NeuralNetwork` class. *Note:* Your predicted model may not be that accurate, but it should be smooth.
- ☐ Create another image plot the relative differences with your solution in part (a).

(c) (20 points) Now we will incorporate the boundary and initial conditions into the loss function $\mathcal{L} = \mathcal{L}_{\text{exact}} + \mathcal{L}_{\text{bc}} + \mathcal{L}_{\text{ic}}$, where

$$\mathcal{L}_{\text{bc}} = \frac{1}{N_c}\sum_{i}^{N_c}[z_i - u(0,t)]^2, \quad \mathcal{L}_{\text{ic}} = \frac{1}{N_c}\sum_{i}^{N_c}[z_i - u(x,0)]^2, \tag{5}$$

where z_i are $N_c = 50$ predicted values on the boundaries and initial time. These are set in the routines `return_bc_loss` and `return_ic_loss`. Use $k = 40$ nodes in the hidden layer and the same hyperparameters as in part (b).

- ☐ Plot the loss curve for this model and compare it against your loss curve found in part (b) for $k = 40$.
- ☐ Create an image plot of the predicted solution $\hat{u}(x,t)$.

☐ Create another image plot the relative differences with your solution in part (a).

(d) (30 points) Finally we will include the residual of Burger's Equation in the loss function so that $\mathcal{L} = \mathcal{L}_{\text{exact}} + \mathcal{L}_{\text{bc}} + \mathcal{L}_{\text{ic}} + \mathcal{L}_{\text{res}}$, where

$$\mathcal{L}_{\text{res}} = \frac{\partial \hat{u}}{\partial t} + u\frac{\partial \hat{u}}{\partial x} - \nu\frac{\partial^2 \hat{u}}{\partial x^2}. \tag{6}$$

This is more difficult to compute because we need the derivatives of the predicted solution $\hat{u}$. For this you will need to modify the routine `return_deriv` to return these derivatives. Here you will predict nearby points at each combination of $(x \pm \Delta x/2, t \pm \Delta t/2)$ with $\Delta t = \Delta x = 10^{-3}$ and use central differencing to compute the derivatives.

☐ Plot the loss curve for this model and compare it against your loss curve found in part (c).
☐ Create an image plot of the predicted solution $\hat{u}(x, t)$.
☐ Create another image plot the relative differences with your solution in part (a).

In my experience, this model does no better than the model in part (c). What could be some solutions to better improve the model if given more freedom when constructing the neural network architecture?

☑ **For full credit**, you should include your program with comments and include the requested plots along with descriptions of them.

2. **Application question (20 points total):** There are four different methods that can be used to incorporate physics into neural networks:

1. Train the model on a large amount of data and let the NN learn the constraints. No physics laws fed into the NN so it may require a lot of training to obtain the results.
2. Enforce physical laws as a hard constraint in the NN architecture or during the optimization. But it can be difficult to train the NN with such constraints.
3. Use penalty methods and add the physics model as a residual to the loss. This is the method that you used for problem 1 of this homework. This is simpler to implement and works with any NN architecture.
4. Use a combination of observation data as well as physical constraints. These physical constraints are added as soft penalty terms to the residual loss function.

There are some problems in physics which are better suited for a particular method. For example, it would be useful to use the first method to classify different types of solar activity from satellite data where the physics driving these events are complex and unknown but there are distinctive patterns in the optical signatures. Pick one example physics problem and describe why this application is best suited to be solved using one of the four ways to use physics informed neural networks.

☑ **For full credit,** you should pick an example that is not the same as problem one or your final project. You should also explain how you would incorporate the physics into the NN in a quantitative way with the specific governing equations or laws.