CAS MA 539 — Methods of Scientific Computing

Name: Emmy Blumenthal Final Project Proposal **BU ID:** U87312711

Due Date: Nov 17, 2022 **Email:** emmyb320@bu.edu

GitHub Repository: `https://github.com/emmyb-bu/ma539-PINNs`

Physics-Informed Neural Networks

In this project, I will explore physics-informed neural networks (PINNs) in *Julia* using existing, extensible packages that implement neural networks (i.e., Lux.jl), automatic differentiation (e.g., Zygote.jl, ForwardDiff.jl, etc.), and optimization (e.g., Optimisers.jl). PINNs use neural networks to parameterize solutions by minimizing the difference between the left-hand and right-hand side of a PDE as a least-squares problem. Specifically, let $u(\vec{x},t)$ such that $\partial_t u + \mathcal{N}[u] = 0$ for $\vec{x} \in \Omega$ and $t \in [0,t_1]$ with boundary data specified as a collection of n points $((\vec{x}_i,t_i),u_i)_{i=1}^n$ with $\vec{x}_i \in \partial\Omega$. Next, the solution $u(\vec{x},t|\vec{\lambda})$ is modeled as a neural network that takes an array $(\vec{x},t)$ as an input and returns a value of u, where $\vec{\lambda} \in \mathbb{R}^p$ are the weights and biases of the neural network. The PINN is fit by solving the optimization problem:

$$\min_{\vec{\lambda}\in\mathbb{R}^p} \quad \frac{1}{t_1|\Omega|}\int_0^{t_1}\int_\Omega \left(|\partial_t u(\vec{x},t|\vec{\lambda}) + \mathcal{N}[u(\vec{x},t|\vec{\lambda})]|^2\right) d\vec{x}dt + \frac{\alpha}{n}\sum_{i=1}^n |u(\vec{x}_i,t_i|\vec{\lambda}) - u_i|^2. \tag{1}$$

Here, the first term enforces that $u(\vec{x},t|\vec{\lambda})$ obeys the PDE, and the second term enforces that the solution is fit to boundary data; $\alpha > 0$ is a hyper-parameter. To solve the problem on the computer, integrals will be discretized by choosing a set of well-spaced collocation points using a method like Latin hypercube sampling; the factor of $1/t_1|\Omega|$ indicates that after discretization, the first term is a mean-square-error-like term; the boundary data may be discretized boundary conditions. Derivatives like ∂_t and those involved in $\mathcal{N}$ will be constructed using automatic differentiation of the neural network's inputs. Additionally, the operator $\mathcal{N}_{\vec{\mu}}$ can be parameterized by additional parameters $\vec{\mu} \in \mathbb{R}^q$ which are optimized in addition to the parameters $\vec{\lambda}$ with the same objective function except the provided data belongs to the region Ω instead of as boundary conditions. In a non-ideal case, the data may constitute real-world/experimental data and the objective of the PINN is to learn the operator $\mathcal{N}_{\vec{\mu}}$. In this project, I will aim to implement this method in four cases:

1. Solving the 1D time-dependent Schrödinger equation (2) and comparing with solution found using the time-independent Schrödinger equation and exact diagonalization.

$$i\frac{\partial\psi}{\partial t} = -\frac{1}{2}\frac{\partial^2\psi}{\partial x^2} + V(x)\psi \tag{2}$$

The potential function will first be chosen to be $V(x) = 0$, and we will implement Dirichlet boundary conditions (i.e., the 'infinite square well'). Then, we will experiment with various other potentials which may include the quantum harmonic oscillator, the anharmonic oscillator, the double-well potential, and scattering potentials.

2. Discovering parameters in the potential function used in the 1D time-dependent Schrödinger; the potential may be parameterized in three ways:

   (a) The potential will have a closed analytical form with few parameters that will be discovered

   (b) The potential will have a closed analytical form where optimized parameters represent coefficients of a Fourier series expansion of the potential

   (c) The potential (maybe the entire Hamiltonian) will be parameterized entirely by a neural network

   The parameters will be discovered by training on data generated using other solution methods (e.g., exact diagonalization, Crank-Nicolson). The generated data could be obscured one step further by sampling from the probability distribution function (PDF) specified by the wave-function to roughly simulate experimental data. The loss function would then include a term minimizing the KL-divergence between the PDF specified by the wave-function found using the PINN method and the simulated data.

3. Solving the Liouville equation (3) from Hamiltonian and statistical mechanics for one particle

$$\frac{\partial \rho}{\partial t} = -\{\rho, H\} = \frac{\partial H}{\partial q}\frac{\partial \rho}{\partial p} - \frac{\partial H}{\partial p}\frac{\partial \rho}{\partial q} \tag{3}$$

   Solving the Liouville equation with PINNs is useful because it is often difficult to avoid negative (and thus unphysical) values of ρ when solving numerically, and parameterization that explicitly prevents negative values of ρ is easy to implement with a neural network. It may be interesting to compare some of the results of the Liouville equation (i.e., classical ensembles) to the results of the Schrodinger equation (i.e., quantum ensembles) for the same potentials.

4. Solving the Liouville equation (4) from Hamiltonian and statistical mechanics for $S > N$ particles

$$\frac{\partial \rho}{\partial t} = -\{\rho, H\} = \sum_{i=1}^{N} \frac{\partial H}{\partial q_i}\frac{\partial \rho}{\partial p_i} - \frac{\partial H}{\partial p_i}\frac{\partial \rho}{\partial q_i} \tag{4}$$

These four goals may be too ambitious, so I will proceed through them in order, completing as least item 1 and item 2(a). Throughout this project, I will additionally familiarize myself with existing packages implementing PINNs (e.g., ModelingToolkit.jl, NeuralPDEs.jl); items 3 and 4 may be implemented with these packages alone depending on difficulty and time available. Training may involve the use of GPUs which I will access using Boston University's Shared Computing Cluster. The final product of these explorations will be a set of *Julia* scripts which train the

PINNs, analyze the results, and visualize solutions along with a project summary which describes successful results, challenges, and how existing code might be expanded or generalized.

References:

- `https://arxiv.org/abs/1711.10561`
- `https://arxiv.org/abs/1711.10566`
- `https://en.wikipedia.org/wiki/Physics-informed_neural_networks`
- `https://en.wikipedia.org/wiki/Schrodinger_equation`
- `https://en.wikipedia.org/wiki/Liouville's_theorem_(Hamiltonian)`
- `http://lux.csail.mit.edu/stable/`
- `https://juliadiff.org/ForwardDiff.jl/stable/`
- `https://fluxml.ai/Optimisers.jl/dev/`
- `https://neuralpde.sciml.ai/stable/`