**Questions**

* In PINN, the NN already has access to the system dynamics, either via constraints or other via biases that are baked into model architecture. There are also multiple ‘levels’ of imposing physics, from generic symmetries (momentum, energy conservation) to specifying the explicit dynamics (e.g. if we’re solving a heat equation, we can explicitly apply as a constraint). These models already “know” what physics they’re trying to learn, and are essentially doing parameter fitting.
* ~~Should our model~~ *~~discover~~* ~~the dynamics (i.e. discover the form of the force law, vs , etc.) or “just” parameter-fit?~~

**Questions about papers**

~~- why does the 2019 PINN paper use 1 RK step with 1000s of stages, instead of iterated RK steps like a typical timestepper? is it just because the first approach is much easier to translate into an NN context? (while iterated is probably sth similar to an RNN/LSTM == state space model)~~

**Questions about spring-mass**

* Is it possible to identify & separately? It seems like it’s only possible to estimate their *ratio* at most (i.e. )
* ~~Is the “final objective” still to vary the number of masses? Are we treating the mass-springs as a discretization of a continuous PDE, and thinking of the mass nodes as sample points?~~

**Approaches**

Approach #1: fit neural network to the dynamics

Choice #1: continuous or discrete, i.e. does the m9ggggbgbgodel perform time integration by itself? or

Choice #2: 1st order or 2nd order system?

vs

Approach #2: NN approximates Hamiltonian (i.e. total energy)

Should be a blackbox function approximated by the NN, or do we specify the functional form (i.e. )? Can we/should we impose as constraints the properties that are known to have (symmetric, positive definite, etc.)?

~~(leads back to question of how much pre-existing knowledge to impose on NN)~~

**Architecture 1:**

At timestep , denote the mass positions by and velocities by . Dataset quantities are written with as  and quantities predicted by the network are denoted with a ‘hat’, . The network implements the discrete time-stepping scheme:

In other words, it takes the true position/velocity as input () and computes a prediction for the next position/velocity pair (). The input vector is and the output is .

Physics constraints are applied using the loss. The first term ensures that the trajectories in the dataset are matched,

If we use auto-differentiation to compute , we can add one more term to ensure that the accelerations match:

The second term conserves energy. Setting the initial energy as ,

is the spring energy from the supports, and can be written explicitly as:

where is the predicted spring constant, and are the supports’ positions and are the first/last masses’ positions.

For now, we treat and as parameters outside the main computation graph that are only updated via the loss backpropagation (is this possible/will this work?).

Optional additional constraints:

* Force law:
* is symmetric (), positive semi-definite (smallest eigenvalue >= 0), and tridiagonal
* is diagonal, time-independent, and positive definite

**Appendix**

The “real physics” is given by:

If we approximate the integral with an Euler step,

Alternatively, approximate the integral with a matrix exponential. Using math magic, it can be shown that,

The trained neural network should approach either of the green or blue expressions. By setting them to be equal, we could potentially “invert” the NN and obtain an estimate of (I don’t think this is a good idea).