A lagrangian mechanics oriented approach to neural networks

A S Roberts

Department of Physics, University of Bath, Claverton Down, Bath BA2 7AY, UK

Abstract. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

1. Introduction

1.1. Neural Networks in Physics

Machine learning has become the latest addition to the modern physicists toolkit with a broad range of applications from deep learning for imaging in medical physics [1] to large scale data analysis in astrophysics [2]. Machine learning and often neural networks are allowing physicists to tackle problems which were previously intractable due to the size and scale of the computational required. To understand how this tool can be used one must first understand how the tool works.

Neural networks are the most common form of machine learning and work by finding a generalised non-linear function (the neural network) between the input and output spaces. The input space is the space of data in which patterns aim to be discovered. For medical imaging this might be pixel values for a certain size image or in astrophysics it may be a range of luminosity values attributed to certain wavelengths of light. The output space is similar, a space of possible outputs such as possible classifications of an astrophysical object with respective confidence levels.

One of the main draws of neural networks is the ability to apply the function to any object in the input space, even if previously unseen. The neural network can be seen as a connected graph where each node and connection has a weighting (weights and biases respectively). To obtain a function that can accurately predict an outcome for any element of the input space the neural network has to be trained. This training involves taking a known subset of the input and output spaces called the training data set along with a loss function f_{loss}. The loss function compares the neural networks calculated outputs with the known (true) outputs in the training data. Starting with a randomly initialised neural network (randomised weights and biases) the following process can be repeated to slowly optimise the network:

(i) Pass an element of the training data through the neural network
(ii) Calculate losses for the output
(iii) Take the gradient of the loss function with respect to the weights and biases in the neural network
(iv) Update the weights and biases according to $-\nabla f_{\text{loss}}$ to minimise the loss of the function

By repeating this process local minima of the loss function can be found thereby optimising the neural network. Ideally, a global minima would be found but this is not guaranteed by the method outlined above. Optimisation algorithms such as Adaptive Moment Estimation (ADAM) optimiser [3] can be implemented in order to increase these chances by varying the size of the updates made when taking the gradient of the loss function.

With the ability to identify possibly unseen patterns in data, neural networks are an ideal tool for physicists. One area of neural network development related to this is that of physics informed neural networks (PINN's). These are neural networks that have been given a priori knowledge such as basic physical laws. This knowledge can be encoded into the model in a multitude of ways. Physics informed layers of the neural network are one such encoding such as layers that enforce conservation laws for fluid dynamics. Another encoding is via loss function formulation where the loss function incorporates terms derived from the governing physics equations. These terms will act to penalise the model for deviation from known physical laws.

1.2. Loss Functions

Loss functions play a crucial role in PINNs often acting as a conduit for incorporating known physical laws. PINN loss functions usually have two purposes:

- Data Fitting: This ensures that the network accurately predicts values, much like in more general neural networks.
- Physics Information: An encoding of physical laws such as ensuring specific PDE's are satisfied by penalising deviation from governing equations.

One issue that loss functions face is the requirement of being differentiable with respect to the weights and biases of a network. This often leads to simple regression functions such as mean squared error being selected in order to guarantee this property. A regression function may be paired with a simple physics informed term such as a penalisation of deviation from conservation terms. For example, conservation of mass within fluid dynamics for an ideal fluid can be expressed as

$$\nabla \cdot \dot{q} = 0 \tag{1}$$

so a penalisation term in the loss function could be written as Conservation term $= \Lambda|\nabla \cdot \dot{q}|^2$ thus penalising deviation from equation 1.

Another more applied issue that loss functions face is coding efficiency restraints. As on of the most called functions within the neural network training process a loss function is constrained to mathematically quick calculations and there often cannot afford to implement longer more involved calculations.

1.3. Google JAX

Google JAX [4] is a python machine learning framework developed by Google that collates the properties of Autograd [5] and XLA (Accelerated Linear Algebra) [6] which include just in time (JIT) compilation). Autograd allows for automatic differentiation of both `python` and `NumPy` functions, a beneficial property when performing gradient descent during training of neural networks. JIT compilation allows functionally written pieces of python code to be converted from interpreted to compiled code. By JIT compiling code is converted into mathematically more efficient "black-box" functions which map inputs to outputs in less time. This is especially beneficial for code that is frequently called as it need only be JIT compiled once before the "black-box" version is used thereafter.

1.4. Lagrangian Mechanics

Here we provide a brief overview of Lagrangian Mechanics, a formalism of classical mechanics prioritising the energy of a system as its initial reference point.

Lagrange's formalism considers a set of generalised coordinates $\{q_1, q_2, \ldots, q_n, \dot{q}_1, \dot{q}_2, \ldots, \dot{q}_n, t\}$(hereon denoted as the vectors $\mathbf{q}$ and $\dot{\mathbf{q}}$) which are used to express the kinetic, T, and potential, V, energies of the system. The Lagrangian

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) = T(\mathbf{q}, \dot{\mathbf{q}}, t) - V(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{2}$$

is defined as the difference between the two energies of the system. The Lagrangian provides the final object of interest the action, S, defined as

$$S = \int_{t_1}^{t_2} \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t) dt \tag{3}$$

a functional of which stationary points are of interest. Hamilton's principle [7] that these stationary points are where the Lagrangian models physical reality i.e. $\delta S = 0$.

From here the equations of motions (EoMs) can be obtained by applying the Euler-Lagrange equation which is equivalent to minimising the action (and thus finding its stationary points)

$$\frac{\partial \mathcal{L}}{\partial q_i} - \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) = 0 \tag{4}$$

where we obtain n differential equations for the n generalised coordinates. Equation 4 then provides a generalised form for the EoMs of a conservative system.

Much like generalised coordinates and velocity, generalised momenta is introduced here as

$$p_i = \frac{\partial \mathcal{L}}{\partial \dot{q}} \tag{5}$$

and is the fundamental basis for all forms of momentum, both linear and angular.

1.4.1. Example: Simple Harmonic Oscillator

A motivating example may be that of the 1-dimensional simple harmonic oscillator given by the differential equation

$$\ddot{x} = x \tag{6}$$

with position x, mass m and spring constant k. The kinetic energy of this system is then given by

$$T = \frac{1}{2} m \dot{x}^2 \tag{7}$$

and the potential energy by

$$V = \frac{1}{2} k x^2 \tag{8}$$

This gives the lagrangian

$$\mathcal{L} = \frac{1}{2} m \ddot{x}^2 - \frac{1}{2} k x^2 \tag{9}$$

which when inputted into equation 4 once again provides our known equation of motion

$$\ddot{x} = \frac{k}{m} x \tag{10}$$

Conservation laws can also be identified through the use of lagrangian mechanics via Noether's Theorem [7]. It states that every differentiable symmetry of the action for a conservative system has a corresponding conservation law. The conserved values are often known as Noether currents when considered as functions of time.

1.4.2. Nonconservative Lagrangians

Galley, Tsang and Stein [8] propose a generalisation of Noether's Theorem to allow for non-conservative systems. This works by generalising to the nonconservative lagrangian, Λ, of the form

$$\Lambda = \mathcal{L}(\mathbf{q_1}, \dot{\mathbf{q}}_\mathbf{1}, t) - \mathcal{L}(\mathbf{q_2}, \dot{\mathbf{q}}_\mathbf{2}, t) + K(\mathbf{q_1}, \dot{\mathbf{q}}_\mathbf{1}, \mathbf{q_2}, \dot{\mathbf{q}}_\mathbf{2}, t) \tag{11}$$

where $\mathcal{L}$ is the lagrangian as defined previously and K is an additional coupling term physically similar to a non-conservative potential. The generalised coordinates are doubled, as explained by Galley [9], from $\mathbf{q}, \dot{\mathbf{q}}$ to $\mathbf{q_1}, \dot{\mathbf{q}}_\mathbf{1}, \mathbf{q_2}, \dot{\mathbf{q}}_\mathbf{2}$. This allows for the conversion of Hamiltons principle from a boundary value problem (i.e. "minimise the action passing through given initial and final values") to an initial value problem involving twice the degrees of freedom. These new generalised coordinates must satisfy $q_1(t) = q_2(t)$ and $\dot{q}_1(t) = \dot{q}_2(t)$ for $t \in \{t_i, t_f\}$. The cartoon in figure 1.4.2 depicts this doubling of coordinates to create a free end point.

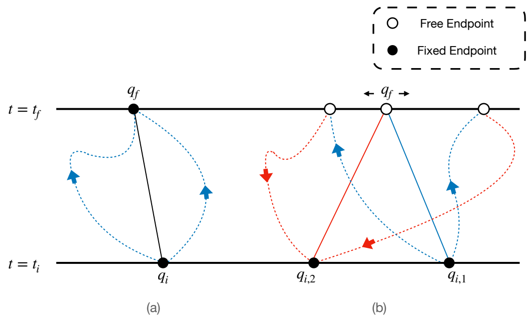

Figure 1. A cartoon depicting the doubling of coordinates posited by Galley [9]. (a) shows a conservative system where initial and end points are fixed such that the solid path between them minimises the Action and dashed paths represent paths that do not minimise the Action. (b) shows a system with doubled coordinates. the point on the $t = t_f$ line is now free to vary but both the $q_{1,i}$ and $q_{2,i}$ points are fixed on the $t = t_i$ line. Here the q_f point varies in order to globally minimise both paths. The physical limit occurs as $q_{2,i} \to q_{1,i}$ and, assuming the same system is modelled in both (a) and (b), rediscovers the path shown in (a).

Galley [9] also shows that K must be antisymmetric in swapping the indices and therefore must also vanish for $\mathbf{q_1} = \mathbf{q_2}, \dot{\mathbf{q}}_\mathbf{1} = \dot{\mathbf{q}}_\mathbf{2}$.

For convenience a change of coordinates is often performed due to physical motivation. These are $q_- = (q_1 - q_2)/2$ and $q_+ = (q_1 + q_2)/2$ where the former can be considered a hypothetical displacement that vanishes in the physical limit $q_- \to 0$ and the latter the physically relevant component of the coordinates. Note here that equation 11 continues to satisfy the Euler Lagrange equation (equation 4) when evaluated in the physical limit (PL). This is formulated as

$$\frac{d}{dt}\left[\frac{\partial \Lambda}{\partial \dot{q_-}}\right]_{PL} - \left[\frac{\partial \Lambda}{\partial q_-}\right]_{PL} = 0 \tag{12}$$

1.4.3. Hamiltonian Mechanics

Another formulation of classical mechanics similar to lagrangian mechanics is that of Hamiltonian mechanics. As the name suggests the object of interest here is the classical Hamiltonian H defined as

$$H = T + V \tag{13}$$

Hamiltonian mechanics exchanges the use of generalised velocities ($\dot{q}$) in favour of generalised momenta p. This is sometimes, but not always, equivalent to momenta in classical mechanics in the same way that $\dot{\theta}$ could be considered a velocity (time derivative of a coordinate θ) but would not likely be ones initial idea for a velocity. This exchange does not affect the class of physical phenomena that can be described by Hamiltonian mechanics compared to

Lagrangian mechanics; that is to say that these classes of phenomena are equivalent.

The Hamiltonian (equation 13) produces first order ordinary differential equations (ODEs) as the equations of motion

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i} \tag{14}$$

This formulation is often useful for symplectic integrators [10] as discussed in section 2.

2. Slimplectic Integrator

Integrators are another tool within the modern physicists toolbox. Many physics problems have non-analytic solutions and this is where integrators make their appearance. Many integrators exist and are specialised for different applications. The Runge-Kutta method [11] is a well known and widely applied group of methods used for a balanced trade off between accuracy and efficiency. RK45 is perhaps paramount of these with an efficient method of finding solutions to problems with stronger than 4th order accuracy. One issue that Runge-Kutta faces however is that of compounding error. Whilst any one term has bounded error when using Runge-Kutta for integration, the total accumulation of error is not bounded causing issues for systems evolved over longer time periods.

A remedy to this is another class of integrators called Symplectic integrators. These preserve (up to computational rounding) the differential 2-form called the symplectic form. Preservation of the symplectic form is analogous to preservation of certain physical constants. Due to this preservation the symplectic integrator is often used in integration of systems over long time scales. This is particularly common in orbital mechanics [12, 13] but also finds uses in statistical algorithms [14].

One construction of symplectic integrators is via the means of variational integrators which are determined by the variation of a discretised action. This differs from traditional numerical integrators, such as Runge-Kutta, which discretises the equations of motion themselves.

igure

Symplectic integrators are often used to solve Hamiltonian systems numerically. One common approach used for symplectic integrators is splitting methods such as the leapfrog and Verlet [15] methods. These work by splitting the Hamiltonian into smaller, more manageable parts and iteratively solving the system. By splitting the Hamiltonian like this the integrator can approximate solutions whilst preserving important geometric properties (such as the symplectic form, volume or energy) at a global level.

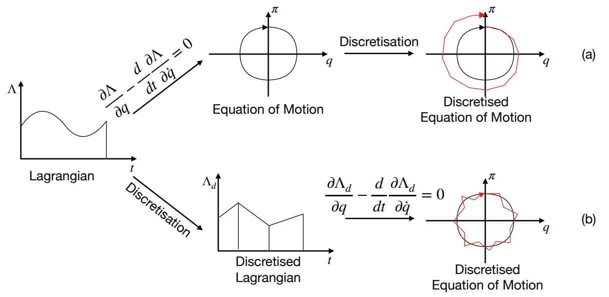
Λ
Lagrangian
t
$\frac{\partial\Lambda}{\partial q} - \frac{d}{dt}\frac{\partial\Lambda}{\partial\dot{q}} = 0$
π
q
Equation of Motion
Discretisation
Discretised
Equation of Motion
(a)
Discretisation
Λ_d
Discretised
Lagrangian
$\frac{\partial\Lambda_d}{\partial q} - \frac{d}{dt}\frac{\partial\Lambda_d}{\partial\dot{q}} = 0$
Discretised
Equation of Motion
(b)

Figure 2. The difference between discretising a standard integrator and a variational integrator. In discretising a standard integrator the discretisation acts directly on the EoM's. However, a variational integrator discretises the action before determining the discretised EoM's.

2.1. The "Slimplectic" Integrator

Tsang et al. [16] have developed a variational integrator, the "Slimplectic" integrator, from the nonconservative action principle [8]. Starting with equation 12 the action integral $S = \int_{t_i}^{t_f} \Lambda(q_\pm, \dot{q_\pm}, t)dt$ is discretised using Galerkin-Gauss-Lobatto (GGL) quadrature [17] as shown in figure 2.1. The GGL quadrature is chosen as it is an even order method and therefore symmetric under time reversal ($t \to -t$). The interval $t \in [t_n, t_{n+1}]$ has $r+2$ quadrature points given by

$$t_n^{(i)} \equiv t_n + (1 + x_i)\frac{\Delta t}{2} \tag{15}$$

where $\Delta t = (t_{n+1} - t_n)/2$, $x_0 \equiv -1, x_{r+1} \equiv 1$ and x_i is the ith root of dP_{r+1}/dx, the derivative of the $(r+1)$th Legendre polynomial $P_{r+1}(x)$.

Replace Ima

To obtain approximations of values for $\dot{q}_{n,\pm}^{(i)}(t)$ at the quadrature points the derivative matrix [18]

$$D_{ij} = \begin{cases} -(r+1)(r+2)/(2\Delta t) & i = j = 0 \\ (r+1)(r+2)/(2\Delta t) & i = j = r+1 \\ 0 & i = j \notin \{0, r+1\} \\ \frac{2P_{r+1}(x_i)}{P_{r+1}(x_j)(x_i - x_j)\Delta t} & i \neq j \end{cases} \tag{16}$$

is used to give

$$\dot{q}_{n,\pm}(t_n^{(i)}) \simeq \sum_{j=0}^{r+1} D_{ij} q_{n,\pm}^{(j)} \tag{17}$$

where Tsang et al. [16] provides more explanation as to the reasoning for this approximation.

GGL quadrature allows any functional $\int F dt$ to be approximated over an interval of time by a discrete

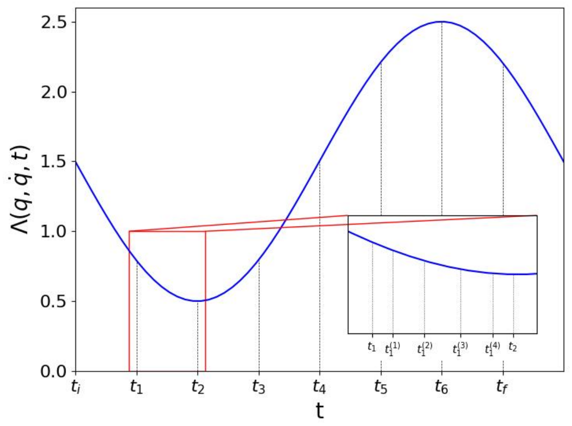

Figure 3. The discretisation of time used within the GGL method. This is used to approximate the action to N timesteps between t_i and t_f. Between t_n and t_{n+1} the quadrature discretises r times as according to equation 15.

functional F_d given by

$$\begin{aligned} F_d^n &\equiv F_d\left(q_{n,\pm}, \left\{q_{n,\pm}^{(i)}\right\}_{i=1}^{r}, q_{n+1,\pm}, t_n\right) \\ &\equiv \sum_{i=0}^{r+1} w_i F\left(q_{n,\pm}^{(i)}, \dot{\phi}_{n,\pm}^{(i)}, t_n^{(i)}\right), \end{aligned} \tag{18}$$

where w_i is a weighting given by

$$w_i \equiv \frac{\Delta t}{(r+1)(r+2)[P_{r+1}(x_i)]^2}. \tag{19}$$

With a discretisation of $\dot{q}_\pm$ the action can be discretised by applying the methodology of equation 18 to equation 11 defined as

$$\mathcal{S}_d\left[t_0, t_{N+1}\right] \equiv \sum_{n=0}^{N} \Lambda_d\left(q_{n,\pm}, \left\{q_{n,\pm}^{(i)}\right\}_{i=1}^{r}, q_{n+1,\pm}, t_n\right). \tag{20}$$

This can then be applied to the the Euler Lagrange equation to obtain discretised equations of motion

$$\left[\frac{\partial \Lambda_d^{n-1}}{\partial q_{n,-}} + \frac{\partial \Lambda_d^{n}}{\partial q_{n,-}}\right]_{PL} = 0 \tag{21a}$$

$$\left[\frac{\partial \Lambda_d^{n}}{\partial q_{n,-}^{(i)}}\right]_{PL} = 0. \tag{21b}$$

Taking the definition of generalised momenta in equation 5, discretised momenta can now be introduced by splitting equation 21a into

$$\pi_n \equiv -\left[\frac{\partial \Lambda_d^n}{\partial q_{n,-}}\right]_{PL} \tag{22a}$$

$$\pi_{n+1} \equiv \left[\frac{\partial \Lambda_d^n}{\partial q_{n+1,-}}\right]_{PL} \tag{22b}$$

$$0 \equiv \left[\frac{\partial \Lambda_d^n}{\partial q_{n,-}^{(i)}}\right]_{PL} \tag{22c}$$

which finally produces a set of discretised equations that can be computationally solved for. By taking initial conditions for q_0, π_0 the integrator can then forward integrate the system for N iterations whilst preserving (up to a constant error bound)

Tsang et al. [16] have developed a `python` code `slimplectic` which makes use of the `SymPy` package to include a computational solver using the slimplectic method as described in equation 22. The package takes initial conditions; a value for r; a number of timesteps, N, for which to iterate; and a lagrangian to determine values for q_n, π_n with $n \in \{1, N\}$.

3. Code and Method

Here we outline the two pieces of code developed and testing framework to identify strengths and weaknesses with each.

3.1. *Slimplectic Integrator 2.0*

Using the Google JAX package we have developed an "autodiffable" version of the Slimplectic integrator (hereon referred to as the JAX-based integrator). This follows the generation method outlined in section 2 which matches that of Tsang et al [16] but uses the JAX implementations of `Autograd` and `NumPy` in place of the `SymPy` package used in the original version. Similar to the original Slimplectic integrator, the JAX-based integrator takes values for r, the number of timesteps along with Δt and initial conditions for q and π to forward integrate. In place of the `SymPy` expressions for L and K, a python function with arguments (q, π) with the form of the Lagrangian is inputted.

Two examples (sections 3.1.1, 3.1.2) are used to compare the SymPy-based and JAX-based Slimplectic Integrators over two metrics. The first of these being the time complexity of the integrator measured by determining q and π for a large number of timesteps. The second being the order of the method, determined by varying r and observing the time taken to generate values for a fixed number of timesteps.

3.1.1. Damped Harmonic Oscillator

Using the format of equation 12, a one-dimensional damped harmonic oscillator (DHO) has a lagrangian

$$\Lambda(\mathbf{q}_\pm, \dot{\mathbf{q}}_\pm, t) = \frac{1}{2}m\dot{q}^2 - \frac{1}{2}kq^2 - c\dot{q}_+ q_- \tag{23}$$

where m is the mass of the oscillating object, k is the spring constant of the oscillator, and c is a damping constant.

We choose a DHO as it allows us to identify non-conservative behaviour being correctly predicted at high levels of accuracy due to its simplicity.

3.1.2. Poynting-Robertson Drag

As with Tsang et al. [16] we also examine the orbital motion of a small dust particle being acted on by radiation from a solar type star. This particle would experience Poynting-Robertson drag [19] which would give the system a conservative Lagrangian of

$$L = \frac{1}{2}m\dot{\mathbf{q}}^2 + (1-\beta)\frac{GM_\odot m}{|\mathbf{q}|} \tag{24}$$

with $\mathbf{q}$ the particles position and m the particles mass. The β term represents a dimensionless quantity which is the ratio between forces due to radiation pressure and gravity given by

$$\beta \equiv \frac{3L_\odot}{8\pi c\rho GM_\odot d}. \tag{25}$$

Here ρ and d are the density and diameter of the dust particle and $L_\odot$ and $M_\odot$ are the solar luminosity and mass.

The Poynting-Robertson drag manifests itself as the non-conservative term

$$K = \frac{\beta GM_\odot m}{c\mathbf{q}_+^2}\left[\dot{\mathbf{q}}_+ \cdot \mathbf{q}_- + \frac{1}{\mathbf{q}_+^2}(\dot{\mathbf{q}}_+ \cdot \mathbf{q}_+)(\mathbf{q}_+ \cdot \mathbf{q}_-)\right]. \tag{26}$$

This example is chosen due to its use in Tsang et al. [16] to compare the Runge-Kutta 4th order method.

3.2. Neural Network and Loss Function

The second piece of code is a neural network written using the Keras [20] and Tensorflow [21] frameworks. The network uses a simple, multi-layer LSTM model [22] combined with a novel loss function which implements the JAX-based slimplectic integrator. The NN is built to identify coefficients of terms within the Lagrangian for Lagrangians of a specific 'family'. A 'family' of Lagrangians is defined to be the set of all Lagrangians with the same terms. For example, the DHO lagrangian in equation 23 would be part of the family of Lagrangians with the form

$$\Lambda = c_1\dot{q}^2 - c_2 q^2 - c_3\dot{q}_+ q_- \tag{27}$$

where the constant term has been omitted as Lagrangians differing by a constant term will produce the same equations of motion. As an input, the NN takes observational data, such as (q, π) or $(q, \dot{q})$ at sequential time-steps. The relationship between the input and output here highlights the motivation for creating such a NN. We aim to create a program that has the ability to correctly identify conserved quantities (symmetries) alongside other physical phenomena (e.g. energy decay in damped systems) without knowing any information about the underlying physics governing the system.

The loss function comprises of terms akin to the L^2 norms in the position, q, and velocity, $\dot{q}$, spaces and has the form

$$\begin{aligned}\text{loss} = &\frac{1}{2}\left(\sum_{i=0}^{N}(q_{\text{true}}(t_i) - q_{\text{pred}}(t_i))^2\right)^{1/2} \\ &+\frac{1}{2}\left(\sum_{i=0}^{N}(\dot{q}_{\text{true}}(t_i) - \dot{q}_{\text{pred}}(t_i))^2\right)^{1/2}\end{aligned} \tag{28}$$

where $q_{\text{pred}}, \dot{q}_{\text{pred}}$ are q and $\dot{q}$ values generated by taking the NN's current predictions for the Lagrangian coefficients and passing them through the JAX-based slimplectic integrator. $q_{\text{true}}, \dot{q}_{\text{true}}$ are the NN input values (i.e. the known observational data). This form was chosen as to not specify more importance for either the position or velocity spaces as well as minimising total deviance from the "true" values.

We use the examples in section 3.1 once again to study the behaviour of the NN. Here we generate a sample dataset from the family of Lagrangians the NN has been trained on. This sample dataset can then be passed into the trained NN to obtain predicted lagrangians. These predicted lagrangians can then be compared via the loss function and statistical analysis in order to identify whether or not the neural network has accurately predicted lagrangians that governs the system and thus identified the equations of motion.

4. Results

4.1. Integrating for N timesteps

We first focus on the JAX-based slimplectic integrator and look to its behaviour for large numbers of iterations. Figure 4.1 compares the time taken to forward integrate a DHO system for N iterations with $r = 2, \Delta t = 0.1$ for both the original and JAX-based integrators.The JAX-based integrator performs

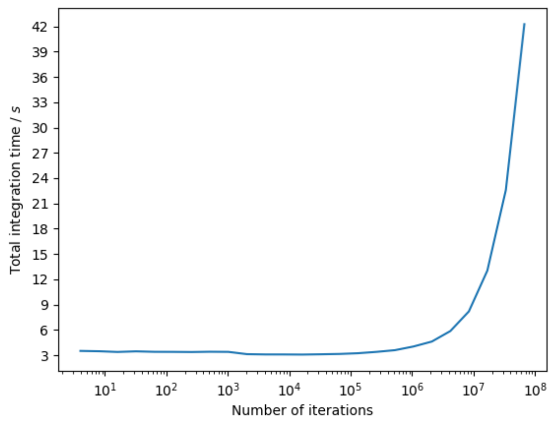

Figure 4. Time taken to integrate a simple DHO system for N timestep. The system has the parameters $r = 4, \Delta t = 0.01$s and has a lagrangian with the same form as equation 27 with $c_1 = c_2 = c_3 = 1$. Note that the number of iterations, N, is reported against a logarithmic scale and therefore the exponential growth seen for $N > 10^6$ is a linear relationship between N and total integration time for N in this region.

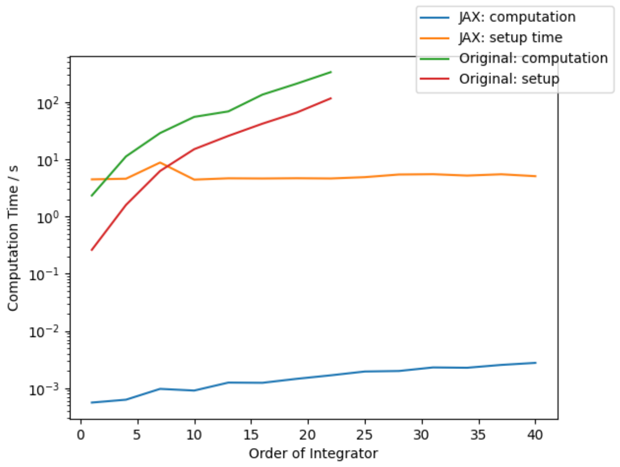

Figure 5. A comparison of the order of the integrator and the time taken to integrate the system. Time is split by basis of the model (original vs JAX) and by setup/computation calculations. Here we observe the fixed compiling cost once again dominating the total time for the JAX-based integrator effectively providing a fixed computation time regardless of integrator order. Times for order > 25 were not recorded for the original integrator due to extreme computation time.

this task in approximately constant time for $N < 10^6$. This is due to the fixed time cost of JIT compiling the system with the integration time being linearly proportional to N, the number of time-steps. The total integration time for the integrator can therefore be seen as $t_{\text{tot}} = t_{\text{comp}} + t_{\text{int}}$. For $N < 10^6$ we have $t_{\text{comp}} \gg t_{\text{int}}$ which exhibits itself as approximately constant time. For $N > 10^6$ we have $t_{\text{compile}} \ll t_{\text{int}}$ which produces the observed linear growth in t_{tot}. Note that the exact value at which t_{tot} transitions from the constant to linear domain is dependant on the specific family of lagrangians. A lagrangian defined by 4 coefficients will differ from one defined by 12 coefficients due to the increased complexity in the mathematical calculations performed.

We observe that the original slimplectic integrator performs integration for all values of N in a manner such that $t_{\text{tot}} \propto N$. For $N < 10^2$ we observe that the original integrator performs the integrations quicker than the JAX-based version due to the lack of compile time. We observe that for all values of N, $t_{\text{int,original}} \propto 10^2 \times t_{\text{int,JAX}}$ which for large values of N represents an hundred-fold increase in the computation speed of the JAX-based version when compared to the original.

4.2. Changing the order of the Integrator

The order of the Slimplectic integrator, [16], is $r+2$ and allows us to approximate the error in the integrators predictions as

$$|\text{Err}(\Delta t)| \approx a(\Delta t)^{r+2}. \tag{29}$$

We can therefore increase the accuracy of the integrator by increasing r. Figure 4.2 shows the effect of varying this for a Poynting-Robinson Drag system. The compile time dominates the JAX-based integrators total integration time leading to an effective fixed cost computation for r$\sim 10^1$. The JAX-based integrator cannot currently discretise order > 85 due to numerical instability in the Legendre polynomial root finding used in 15.

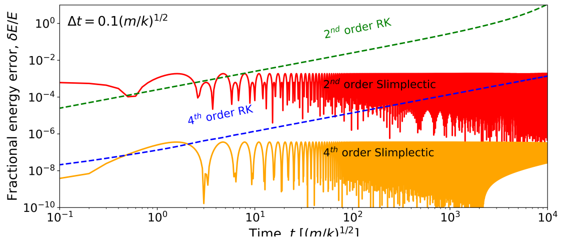

Figure 6. ee

4.3. Bounding the Integrator Error

As with the original slimplectic integrator we investigate the fractional energy error relative to the analytic solutions energy. Here we observe almost identical behaviour to the original integrator. Mirroring figure 2 in Tsang et al. [16] we compare the fractional energy error with the system evolution time for the JAX-based integrator and Runge-Kutta 2 and 4 methods. This is shown in figure [?] and almost directly mirrors the original slimplectic integrators version of this figure.

4.4. Approximating DHO Functions

We now move to the NN which was trained on a dataset of 5.0×10^5 randomly generated non-conservative lagrangians within the DHO family given by equation 27. For each generated set of lagrangian coefficients, $\{c_1, c_2, c_3\}$, the conditions $c_1 > 0$ and $c_2, c_3 \geq 0$ were applied. Approximately 5% of c_2 and c_3 values were randomly set to 0 such that the training dataset was guaranteed to have systems with $V, K = 0$. Negative values were discouraged through the implementation of an additional term in the loss function penalising coefficient values < -0.1. This limit was chosen as to not discourage the neural network from choosing zero valued coefficients whilst still discouraging non-physical systems.

Training of the model was temperamental and often susceptible to random permutations of the dataset causing diverging loss values which eventually broke models. By increasing batch sizes and introducing loss caps this was eventually remedied to produce a model that identifies trends within the data. Although not able to perfectly reproduce equations of motion as yet, figure 4.4 shows promising progress for the NN. The model produced a RMS error value of ± 0.050 in q and ± 0.10 in π. The loss function was also varied during the training process, initially weighting more in favour of the RMS error in q to compensate for the fact that a majority of systems produced larger π-values than q-values.

Biasing the loss function only use RMS error in π produced results of interest, initially reducing loss to 10^{-4}, a before unseen value. However, this was accomplished by the model choosing $c_1 \approx c_2 \approx c_3$ and then varying these values by small quantities to appease the loss function, thereby not producing physically accurate predictions.

Another result of interest was the models ability to train to an accuracy greater than that used to produce figure 4.4. The training process frequently approached a validation loss ≈ 0.20 but often struggled to become more accurate than this with the loss soon-after "blowing up", possibly due to the complexity of the loss function below this resolution. In figure 4.4 we can see the phase of q has been matched but the predicted amplitude is smaller than the true amplitude. Remedying this would require scaling of both potentials V and K by the same quantity. If one of these parameters were to change but not the other the model would see a "de-syncing" of the phase thereby increasing loss. Therefore the complexity in the loss function here may be down to the requirement to scale both coefficients equally. This issue was identified after exploring gradient descent in the embedding space to study the loss function. Figure 4.4 identifies this behaviour by identifying the behaviour of the loss function about the global minima, i.e. the true embedding of the function.

5. Discussion

6. Conclusion

Acknowledgments

I would like to thank my project partner Joshua Coles for his indispensable coding abilities, ideas and overall support throughout this project. Without him I would still probably be stuck trying to set up the correct python environment. I would also like to thank my project supervisor Dr David Tsang. His advice and mentorship have been invaluable and I have very much enjoyed our often wandering discussions during project meetings.

References

[1] Sahiner B, Pezeshk A, Hadjiiski L M, Wang X, Drukker K, Cha K H, Summers R M and Giger M L 2019 *Med. Phys.* **46** e1–e36

[2] Vanderplas J, Connolly A, Ivezić Ž and Gray A 2012 Introduction to astroml: Machine learning for astrophysics *Conference on Intelligent Data Understanding (CIDU)* pp 47 –54

[3] Kingma D P and Ba J 2017 Adam: A method for stochastic optimization (*Preprint* `1412.6980`)

[4] Bradbury J, Frostig R, Hawkins P, Johnson M J, Leary C, Maclaurin D, Necula G, Paszke A, VanderPlas J, Wanderman-Milne S and Zhang Q 2018 JAX: composable transformations of Python+NumPy programs URL `http://github.com/google/jax`

[5] Maclaurin D, Duvenaud D and Adams R P 2015 Autograd: Effortless gradients in numpy *ICML 2015 AutoML Workshop* vol 238 p 5

[6] Openxla `https://openxla.org` accessed: 2024-03-12

[7] Goldstein H 1980 *Classical Mechanics* (Addison-Wesley)

[8] Galley C R, Tsang D and Stein L C 2014 Publisher: arXiv Version: 1 URL `https://arxiv.org/abs/1412.3082`

[9] Galley C R 2013 *Phys. Rev. Lett.* **110**(17) 174301 URL `https://link.aps.org/doi/10.1103/PhysRevLett.110.174301`

[10] Sanz-Serna J 1992 *Acta Numerica* **1** 243 – 286

[11] DeVries P and Hasbun J 2011 *A First Course in Computational Physics* (Jones & Bartlett Learning) p 215 ISBN 9780763773144

[12] Wisdom J and Holman M 1991 *The Astronomical Journal* **102** 1528–1538

[13] Rein H and Tamayo D 2015 *Monthly Notices of the Royal Astronomical Society* **452** 376–388 ISSN 0035-8711

[14] Brooks S, Gelman A, Jones G and Meng X L 2011 *Handbook of Markov Chain Monte Carlo* (Chapman and Hall/CRC) ISBN 9780429138508 URL `http://dx.doi.org/10.1201/b10905`

[15] Verlet L 1967 *Phys. Rev.* **159**(1) 98–103 URL `https://link.aps.org/doi/10.1103/PhysRev.159.98`

[16] Tsang D, Galley C R, Stein L C and Turner A 2015 *The Astrophysical Journal Letters* **809** L9 URL `https://dx.doi.org/10.1088/2041-8205/809/1/L9`

[17] Farr W M and Bertschinger E 2007 *The Astrophysical Journal* **663** 1420–1433 (*Preprint* `astro-ph/0611416`)

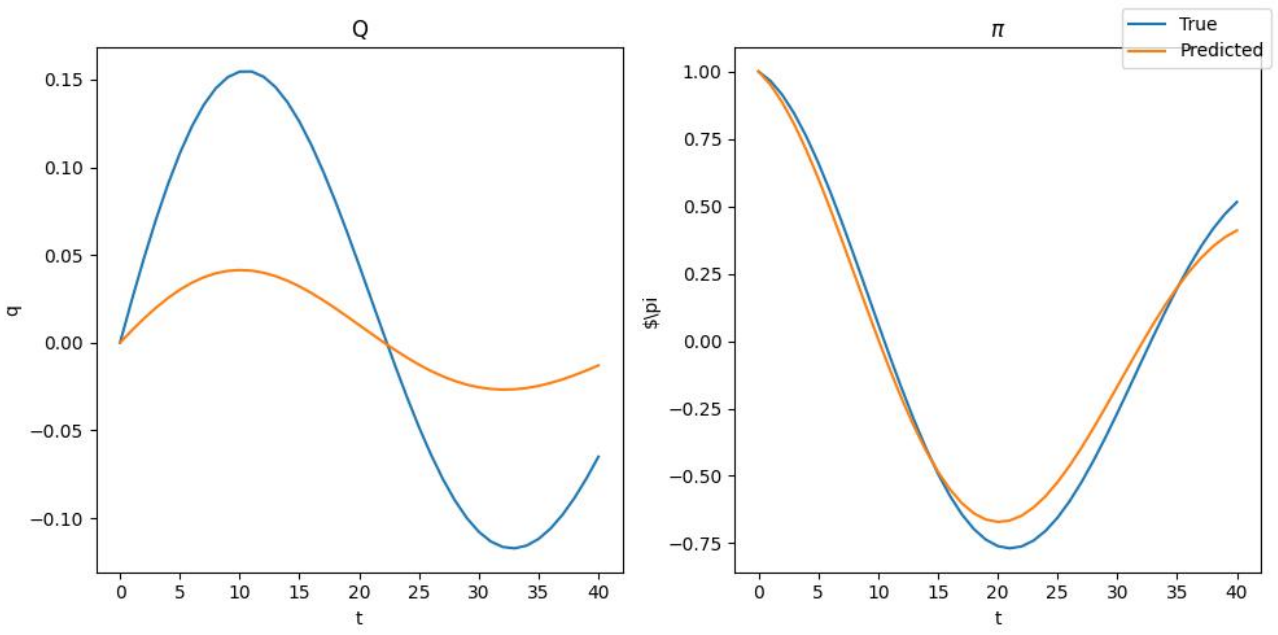

Figure 7. ee

Figure 8. ee

[18] Boyd J P 2001 *Chebyshev and Fourier spectral methods* (Courier Corporation)

[19] Burns J A, Lamy P L and Soter S 1979 *Icarus* **40** 1–48 ISSN 0019-1035 URL `https://www.sciencedirect.com/science/article/pii/0019103579900502`

[20] Chollet F *et al.* 2015 Keras `https://keras.io`

[21] Abadi M *et al.* 2015 TensorFlow: Large-scale machine learning on heterogeneous systems software available from tensorflow.org URL `https://www.tensorflow.org/`

[22] Goodfellow I, Bengio Y and Courville A 2016 *Deep Learning* (MIT Press) `http://www.deeplearningbook.org`