Iterations on the Slimplectic integrator and its applications to physically informed loss functions

J D Coles[1]

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

Abstract. The Slimplectic Integrator developed by Tsang *et al.* [1] provides fractional error bounds when integrating non-conservative Lagrangian systems. In this paper, we present iterations on this method using modern computational techniques improving its computational performance by a factor of $\approx 10^4$, with a small fixed cost, for long-term integrations on the order of $N_{\text{iter}} \approx 10^6$ and orders $r < 40$. We present applications of this work to physically informed loss functions for identifying physical systems from observational data and use these in the training of a physics-informed neural network. This network, when trained on damped harmonic oscillators, predicts Lagrangians that are accurate to an absolute RMS error of $\pm 5 \times 10^{-2}$ in $\mathbf{q}$ and $\pm 1 \times 10^{-1}$ in π.

1. Introduction

Neural networks (NNs) and other machine learning (ML) techniques are becoming an increasingly important tool in of modern research process. This is due to their wide application in data-intensive problems, and ability to make progress where prior computational methods have proved otherwise intractable.

The primary method of training ML models involves the construction of a loss function or equivalently a reward function, framing the training process as a high-dimensionality optimisation problem. In this setting, the parameters of the model are varied such as to minimise the aggregate loss of the model's action, when taken over a large collection of inputs, compared to known corresponding outputs. Thus it is the loss function that grounds the model in a given problem. The construction of loss functions varies across applications, but overall they exhibit some general characteristics as will be discussed.

Symplectic integrators are numerical integrators for Hamiltonian systems, which preserve the canonical symplectic 2-form of the system. This makes them widely applicable to physical fields such as orbital dynamics and molecular dynamics, among others [1,2]. Consequentially, they will preserve, or near preserve, the constants of motion of a system over a large number of integration steps.

The Slimplectic Integrator (SI) [3] is a non-conservative extension of a symplectic integrator, which enables numerical integration of non-conservative systems. As with the symplectic integrators, this exhibits well-defined bounds in the fractional error of the energy, and other conserved quantities of the system. These are based on the non-conservative action approach developed by Galley [4] and Galley *et al.* [5].

Automatic-differentiation techniques are a collection of computational methods for the determination of the derivatives of large classes of 'regular' scientific code, doing so in an efficient and accurate manner. In contrast, the Slimplectic Integrator is currently implemented using computer algebra systems such as SymPy [6], which represents and works with mathematical expressions symbolically, allowing for the computation of derivatives and integrals. These are general systems and are effective at working with a large range of expressions, but pose limitations when seeking to increase computational speed or to scale up to larger, more complex, physical systems.

In this paper, we focus on taking the existing mathematical framework of the Slimplectic Integrator and adapting it to use more advanced computational methods. In addition, making it amenable to automatic-differentiation which enables a wide class of new applications. These include scaling to larger systems, for example in molecular dynamics and condensed matter simulations, to its use as the foundation of several physics-based loss functions. These loss functions will evaluated for empirical and theoretical suitability, with the goal that these could be used in the creation of machine learning models which encode physical knowledge and insight, as discussed in the final part of the paper.

2. Preliminaries

2.1. Lagrangian Mechanics

Lagrangian mechanics is a formalism of classical mechanics which places emphasis on the energies and systems of the system and that allows flexibility in the exact variables used to parameterise the system. As it is traditionally phrased it works to solve the problem of determining the evolution of a physical system, parameterised by n generalised degrees of freedom, represented as the vector $\mathbf{q} = \{q_1, \ldots, q_n\} \in \mathbb{R}^n$, across the timespan $[t_i, t_f] \subset \mathbb{R}$ from some initially known configuration $\mathbf{q}_0 \in \mathbb{R}^n$. This determination is done by constructing a functional,

$$S[\mathbf{q}] = \int_{t_i}^{t_f} L(\mathbf{q}, \dot{\mathbf{q}}, t)\ \mathrm{d}t \tag{1}$$

where S is known as the action of the system, $\dot{\mathbf{q}}$ is the standard time derivative of the system's path, and L the Lagrangian. The Lagrangian is traditionally made up of two components T, the kinetic energy of the system in some configuration at some time, and V, the potential energy of the system. These are combined as,

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

A physical solution is arrived at by then employing Hamilton's principle of least action [7] which provides that the physical path, $\mathbf{q}(t)$, is the one that *extremises* the functional $S[\mathbf{q}]$ as defined in Equation (1), ie one where the variation δS with respect to any variation of the path $\delta\mathbf{q}(t)$ is zero. This equation can be found by solving the Euler-Lagrange equations,

$$\frac{\partial L}{\partial q_i} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} = 0 \tag{3}$$

for each generalised degree of q_i. To each we can also associate a conjugate momentum p_i and generalised force F_i,

$$p_i = \frac{\partial L}{\partial \dot{q}_i} \tag{4}$$

$$F_i = \frac{\partial L}{\partial q_i} \tag{5}$$

such that Equation (3) reads in line with Newton's second law, $F = \dot{p}_i$.

The focus on symmetries within the formalism comes to the fore however when we consider Noether's Theorem [8] which states that every continuous symmetry of the action S has a corresponding conserved quantity. A simple example of this can be seen for degrees of freedom which are said to be *cyclic* in that the coordinate value itself, q_i, does not appear directly in the Lagrangian. It can be readily seen from Equation (3) that if this is the case then,

$$\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial L}{\partial \dot{q}_i} = 0 \tag{6}$$

which implies that the conjugate momentum p_i is a conserved quantity of the system.

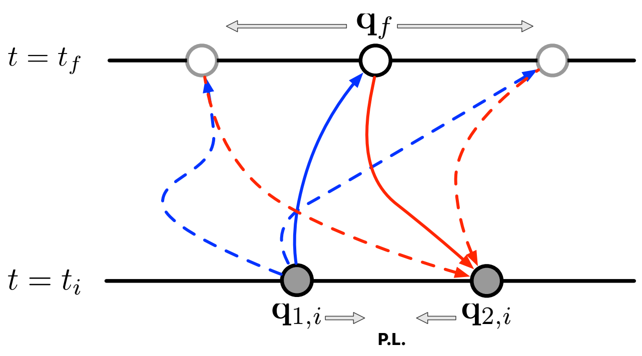

Figure 1. A cartoon based on a similar figure from [4] showing the two virtual paths. We can see that the two paths initiate at their respective initial configurations $\mathbf{q}_{i,\{1,2\}}$ at time t_i and are connected by the final configuration $\mathbf{q}_f$ at t_f. This final configuration is unknown and is allowed to vary within the problem (depicted as two alternative values shown in pale either side), depending on the value of $\mathbf{q}_{i,\{1,2\}}$. which are fixed during extremisation (displayed in grey). In this manner, the problem's boundary resides purely in the $t = t_i$ plane where physical information is known. As shown the physical limit (P.L.) consists of bringing the two virtual paths together such as $\mathbf{q}_{i,1} = \mathbf{q}_{i,2}$.

2.2. Non-Conservative Actions

Lagrangian mechanics as discussed applies only to conservative systems. While there exist other extensions to non-conservative systems, notably Rayleigh dissipation functions [9] for simple dissipative functions, we will focus on the method put forward by Galley *et al.* [4].

This method was originally developed to account for the fact that while we often employ Hamilton's principle and the Euler-Lagrange equations it produces as an initial value problem, it is formally a boundary value problem in time. To remedy this difficulty we formally double the degrees of freedom of our system into two new virtual paths $\mathbf{q}_{1,2}$, with an adapted non-conservative Lagrangian Λ given by,

$$\Lambda = L(\mathbf{q}_1, \dot{\mathbf{q}}_1, t) - L(\mathbf{q}_1, \dot{\mathbf{q}}_1, t) + K(\mathbf{q}_1, \mathbf{q}_2, \dot{\mathbf{q}}_1, \dot{\mathbf{q}}_2, t). \tag{7}$$

The form of this equation, as well as the visualisation shown in figure 1, shows we can consider this doubling as creating two paths, one forwards in time, the other backwards (and hence negative in (7)), with both of these paths contribute with terms as the traditional conservative Lagrangian. There is also an additional term K, the non-conservative potential, representing a non-conservative coupling between the two paths (if it could be broken down cleanly into a form of $V(\mathbf{q}_1) - V(\mathbf{q}_2)$ or similar, and thus being conservative, then these could be absorbed into L leaving $K = 0$). In this way, the system becomes a boundary value problem as needed, with both paths sharing the $\mathbf{q}_f$ point as one of their boundaries.

To solve this system we follow in the same form as before, aiming to extremise a new action defined now in terms of Λ, however crucially we do so in what is known as the 'physical limit' (P.L), where $\mathbf{q}_1 = \mathbf{q}_2$. This gives the non-conservative Euler Lagrange equation,

$$\left[\frac{\partial \Lambda}{\partial q_i} - \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial \Lambda}{\partial \dot{q}_i}\right]_{\mathrm{P.L}} = 0 \qquad (8)$$

In practical considerations, it is often useful to alter our choice of variables to instead be $\mathbf{q}_+ = (\mathbf{q}_1 + \mathbf{q}_2)/2$ and $\mathbf{q}_- = \mathbf{q}_1 - \mathbf{q}_2$. This reduced our P.L. condition to simply $\mathbf{q}_- \to \mathbf{0}$. Note that only the Euler-Lagrange equation involving differentiating with respect to $\mathbf{q}_-$ survives. We can of course also define a corresponding conjugate momenta,

$$\pi_{\mp,i} = \frac{\partial \Lambda}{\partial q_{\pm,i}}. \qquad (9)$$

A corresponding form of Noether's theorem can be shown to hold for these non-conservative systems where the Noether currents evolve in time due to the effect of the non-conservative coupling potential K [5]. There one can also find a more complete explanation of the process described above, along with a rigorous derivation.

2.3. The Slimplectic Integrator

The Slimplectic Integrator is an extension of a symplectic integrator to non-conservative Lagrangian mechanics, symplectic integrators being those which preserve the canonical symplectic 2-form of the system $dp \wedge dq$, and thus preserve many desired properties during integration.

Hence, to understand the Slimplectic Integrator method we first consider the application of a symplectic integrator to a traditional conservative system‡. As before consider a system with N degrees of freedom represented by some $\mathbf{q} \in \mathbb{R}^N$, governed by some Lagrangian, $L(\mathbf{q}, \dot{\mathbf{q}}, t) \in \mathbb{R}$.

‡ This explanation takes its path from an unpublished paper by Tsang *et al.* [10]

With this set up we now choose to apply two successive procedures to our extremal path $\mathbf{q}(t)$: first piecewise breakdown and then discretisation of the action itself. It is this ordering that differentiates this symplectic method from more traditional integrators such as Runge-Kutta [11] which solve similar systems by discretising the final equations of motion themselves rather than the action, before equations of motion are formed.

To start we break the trajectory down piecewise into a collection of M sub-paths $\gamma_{n\in[M-1]}$ where $[A] = \{0, \ldots, A\}$, each defined on some portion of the whole timespan $[t_n, t_{n+1}] \subset [t_i, t_f] \subset \mathbb{R}$ such that they cover it completely with overlaps only at the boundaries of the intervals. These together are such that the there is the correspondence,

$$\mathbf{q}(t) = \Big\{\gamma_n(t) \quad \text{for } t \in [t_n, t_{n+1}] \qquad (10)$$

between the total path and sub-paths for all $t \in [t_i, t_f]$. These paths define a collection of points $\mathbf{q}_{n\in[M]}$ which are their mutual values at the piecewise breakpoints (which we required to be equal) and the initial and final values for $n = 0$ and $n = M$ respectively.

As each piecewise curve constitutes in and of itself a physically attained trajectory we can state that any path $\mathbf{q}(t)$ which extremises the action for the whole timespan, must also extremise each piecewise portion, and visa-versa.

This does not bring us closer to a numerical solution directly but allows us to freely split our integral domain as needed without loss of accuracy, and thus limit the error of our numerical-integration method which will necessarily increase with the number of time-steps or their size.

We now turn to the discretisation method itself. This is done using the Galerkin-Gauss-Lobatto (GGL) quadrature method of order $r \in \mathbb{N}_0$ [3, 12] which approximates the integral from t_n to t_{n+1} using the intermediary points

$$t_n^{(i)} = t_n + (1 + x_i)\frac{\Delta t}{2} \qquad (11)$$

where $i \in [r + 1]$ and x_i are the ordered roots of the the derivative of the $(r+1)$th Legendre polynomial P_{r+1} and $\Delta t = t_f - t_i$. For a given choice of r, this provides slimplectic maps that are accurate up an order $2r + 2$ [3]. In turn we also define $\mathbf{q}_n^{(i)} = \mathbf{q}(t_n^{(i)})$ to be the value of each sub-path at each $t_n^{(i)}$, which we label interior points. This process can be seen visually in figure 2.

We now approximate the path of the system within this quadrature using the associated cardinal functions for the GGL quadrature, labelled $\phi(t)$, as $\mathbf{q}(t) = \phi(t) + \mathrm{O}((\Delta t)^{r+2})$. These provide a suitable

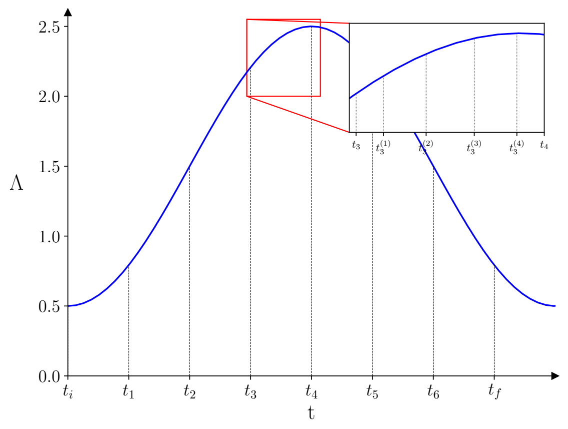

Figure 2. A depiction of the piecewise breakdown and then discretisation of the action. Each t_n represents shows the action over a subpath γ_n. In addition we focus on the sub-path spanning $[t_3, t_4]$ showing the interior times $t_3^{(i)}$ for $r = 4$.

approximation for the path derivative $\dot{\mathbf{q}}$ by the use of the derivative matrix,

$$D_{ij} = \begin{cases} \dfrac{(r+1)(r+2)}{2\Delta t} & i = j = 0 \\ -\dfrac{(r+1)(r+2)}{2\Delta t} & i = j = r+1 \\ 0 & i = j \wedge i \notin \{0, r+1\} \\ \dfrac{2P_{r+1}(x_i)}{P_{r+1}(x_j)(x - x_j)\Delta t} & i \neq j \end{cases} \tag{12}$$

which provides that,

$$\dot{\phi}(t_n^{(i)}) = \sum_{j=0}^{r+1} D_{ij} q_n^{(j)}. \tag{13}$$

In turn, this allows us to express the integral as,

$$\int_{t_n}^{t_{n+1}} L(\mathbf{q}, \dot{\mathbf{q}}, t) \mathrm{d}t \approx \sum_{i=0}^{r+1} w_i L(q_n^{(i)}, \dot{\phi}_n^{(i)}, t_n^{(i)}) \tag{14}$$

where the label the approximate expression L_d^n and where w_i are the quadrature weights given by

$$w_i = \frac{\Delta t}{(r+1)(r+2)(P_{r+1}(x_i))^2}. \tag{15}$$

Equation (14) strung together across the different piecewise sub-trajectories defined in equation (10) gives us the total discretised action of the system, shown in Equation (17), that sets symplectic integrators apart from their more general counterparts,

$$L_d^n = \sum_{i=0}^{r+1} w_i L(q_n^{(i)}, \dot{\phi}_n^{(i)}, t_n^{(i)}), \tag{16}$$

$$S_d = \sum_{n=0}^{M} L_d^n. \tag{17}$$

From here we then extremise this discretised action with respect to the mutual and interior points to obtain the equations of motion of the system as,

$$\frac{\partial L_d^{n-1}}{\partial q_n} + \frac{\partial L_d^n}{\partial q_n} = 0 \tag{18}$$

$$\frac{\partial L_d^n}{\partial q_n^{(i)}} = 0. \tag{19}$$

This first equation can be simplified further into two equivalent definitions for the discrete momentum π_n,

$$\pi_n = -\frac{\partial L_d^n}{\partial q_n} \tag{20}$$

$$\pi_{n+1} = \frac{\partial L_d^n}{\partial q_{n+1}} \tag{21}$$

and a continuity constraint that they both be equal at these mutual overlap points $\mathbf{q}_n$ in the same manner as we required for $\mathbf{q}_n$ itself. Together Equations (19), (20) can be solved to determine the values of $\mathbf{q}_n^{(i)}$, from which Equation (21) provides a form for determining π_{n+1}.

This derivation is provided in terms of the traditional conservative Lagrangian L however can be readily adapted to the non-conservative Lagrangian formalism discussed in section 2.2 by considering instead the non-conservative Euler-Lagrange Equation (8), in effect substituting L for the non-conservative Lagrangian Λ and $\mathbf{q}_n$ for $\mathbf{q}_{n,-}$.

Continuing the consideration of Noether currents from prior sections, it can be shown that the continuous symmetries of the conservative action evolve as due to this now discretised K_d. It should be noted that GGL discretisation does not preserve time-shift symmetry, and hence energy is not conserved under the operation, however, we will show, in line with previous work [3], that the fractional error is generally bounded over the integration.

2.4. Physics Informed Neural Networks

Physics-informed neural networks (PINNs) are neural networks which include physical knowledge in their training processes [13]. Their most common application is in solving partial differential equations (PDEs) [14, 15] where we take physically derived knowledge of the system as a prior when constructing our ML model. For example, one could train the neural network with a loss function incorporating the residuals from the initial and boundary conditions to (loss functions will be discussed in more depth in section 2.5).

Neural networks more generally are a collection of linear and non-linear components composed together to form complex non-linear functions. At their base level the linear components can be expressed as the simple linear equation $\mathbf{y} = W\mathbf{x} + \mathbf{b}$ where $\mathbf{x}$ is the input for this component, W is a matrix of weights and $\mathbf{b}$ an offset with components known as biases. The non-linear components are functions such as sigmoid or tanh, which introduce non-linearity into the model, allowing for non-trivial behaviour. Without these non-linear components, the whole model could be expressed as a single linear operation. Collectively, these weights, biases, and other variables within the model constitute the model's *parameters*.

This composition is useful as it can be shown that, with sufficient complexity, these forms are dense in the space of Borel measurable functions and hence can take the place of almost any physical function or mathematical procedure [16].

2.5. Loss functions

The primary method of training ML models involves the construction of a loss function or equivalently a reward function, this frames the training process as a high-dimensionality optimisation problem. In this setting, the parameters of the model are varied so as to minimise the aggregate loss of the model's action over a large collection of inputs when compared to known corresponding outputs. It is thus the loss function that grounds the model in a given problem, in PINNs, where physics steps in to connect our computational model to reality. Their construction varies but they exhibit some general characteristics as will be discussed.

A loss function traditionally takes the form, loss $= f(y_{\text{true}}, y_{\text{pred}})$ where f is some chosen function and y_{pred} and y_{true} are both real vectors in the output space of the model, representing the model's output and the known true output respectively.

Loss functions in PINNs are often comprised of two components. First, there is the prediction or physical loss which may, as mentioned, be made up of the residuals of the model under PDE, boundary, and initial conditions of the system. Second, there is a regularisation loss term which helps to penalise overfitting to the training data, expressed as 'complexity', within the model. This might be implemented as an L^2 of a vector containing all parameters of the model.

Stochastic gradient descent and Adam [17] are both common choices for the optimisation algorithm used in loss-function based training. These impose various requirements on the loss function, for example, that it is differentiable § and benefits from other properties such as higher-order differentiability, and convexity [19]. The rate at which these methods apply gradient-based updates is determined by a provided learning rate. This will often be varied through training to quickly move towards minima during the initial stages, then tuned down to avoid 'jumping' over the minima as we close in on the optimal solution.

2.6. Lagrangian Embeddings

To work with non-conservative Lagrangians Λ computationally, for example as the output of a PINN, or one of the inputs of a loss function, we must define some mapping from a collection of real numbers, to a form of $\Lambda(\mathbf{q}, \dot{\mathbf{q}}, t)$. This can be thought of as a function,

$$E : \mathbb{R}^n \to \mathbb{L} \tag{22}$$

where E is known as an 'embedding function' and $\mathbb{L}$ is a subset of the overall function space of possible Λ values. The choice of embedding necessarily restricts and dictates the physical systems which the model can deal with, and hence our choice of embedding is crucial. Discussions of different embeddings used in this paper can be found in section 3.1.

There are two broad approaches to designing an embedding function. Either one can be chosen to represent a specific system by its physical parameters, or a more broad class of systems can be represented by some arbitrary function approximation scheme, such as Fourier series or Taylor expansion around a point.

It should be noted that these embeddings often will not have a unique correspondence with physical Lagrangians, i.e. the map is not injective up to physical behaviour. This has implications for optimisation in creating multiple minima, as will be discussed.

2.7. Automatic-Differentiation, XLA, and Google's JAX

Automatic-differentiation is the process of computing the differential of 'regular' code with respect to one or more of its arguments. Google's JAX library [20] contains one implementation of this in Python which works by passing 'tracers' into the Python code that records the actions done to them such that an overall differential can be calculated.

JAX also incorporates the XLA (Accelerated Linear Algebra) library [21], where it uses similar

§ Although methods exist for non-differentiable functions [18].

tracing methods to translate a restricted, but large, subset of Python code into a series of low-level efficient operations which can be executed at speed on the GPU or CPU.

These two technologies together show great promise in the fields of machine learning and physics simulation, as they allow for code that would previously have to be written in low-level languages, such as C or C++, to be expressed in Python which makes high-performance simulations much easier to develop and iterate on.

3. Method

3.1. Damped Forced Harmonic Oscillators

The main physical system used in the testing, and evaluation, of our model is the damped forced harmonic oscillator. It is one of the simplest non-conservative systems, having a non-conservative Lagrangian given by [5],

$$\begin{aligned} \Lambda &= L - K \\ L &= \frac{1}{2} m \dot{\mathbf{q}}^2 - \frac{1}{2} m \dot{\mathbf{q}} - \\ K &= -\lambda \mathbf{q}_- \cdot \dot{\mathbf{q}}_+ + \mathbf{q}_- \cdot \mathbf{F}(t) \end{aligned} \tag{23}$$

for a system with mass m, spring constant k, damping constant λ and time dependent force $\mathbf{F}(t)$. This corresponds with the standard equation of motion,

$$m\ddot{\mathbf{x}} + \lambda \ddot{\mathbf{x}} + k\mathbf{x} = \mathbf{F}(t). \tag{24}$$

3.2. Improvements to the Slimplectic Integrator and their Physical Applications

Initially, the existing `slimplectic` codebase from Tsang *et al.* [22], which we will refer to as the *original implementation*, was rewritten using the JAX framework, known as the *updated implementation.* We chose two key metrics, run-time duration and accuracy, to judge our model against the *original implementation* and Runge-Kutta to various orders.

First, to ensure that we maintained the error bounds expected for our model (as per section 2.3) we compared the fractional energy and momentum error for Runge-Kutta, the *original implementation*, and the *updated implementation* across the time evolution of a physical system.

Next, we create two test cases to measure the performance of the integrator and its consequences on experimental capability. These were,

(i) Modelling systems for different timespans varying the number of iterations performed by the integrator.

(ii) Modelling the same system across different orders of integration r to determine how the order of our method affects the performance.

These are designed to test the scaling behaviours of our model to provide insight into its applicability to larger systems.

3.3. Applications to Loss Functions and Optimisation

Taking this *updated implementation* we then explore its different uses as a loss function for physical systems, in combination with different embedding functions.

While these are distinct components of the model, they are heavily linked, as ideally, we would define the physical components of our loss function primarily in terms of the trajectories, rather than the values of the embedding vector for the non-conservative Lagrangian itself. Hence the embedding function and Slimplectic integrator must be chained together, before the physical loss component, to provide it with the trajectories.

We consider a number of choices for these two functions, comparing their behaviour near known minima, and looking for convexity and smoothness. In addition, we also empirically test the suitability of these spaces for optimisation, by directly performing gradient descent to the loss function + embedding composition attempting to converge to a known embedding.

3.4. Loss Function in the Training of PINNs

Finally, we employ a loss and embedding function combination to a, simplified, stand-in neural network, observing the correctness of the outputs.

This model was based on the long short-term memory (LSTM) layer [23] which is frequently used in time-series analysis, and thus chosen as a suitable basis for our model. This involves generating large datasets of different physical systems within the domain of the chosen embedding as will be discussed.

4. Results and Specific Discussion

4.1. Improvements to the Slimplectic Integrator and their Physical Applications

As discussed we first verify that the *updated implementation* retains the required fractional error bounds of the formal method. This can be seen in figure 3 where the fractional error of the *updated implementation* with respect to that of the true analytic solution is compared directly against that of Runge-Kutta order 2 and 4. We clearly observe that the fractional energy error remains bounded across the timeframe of iteration. This

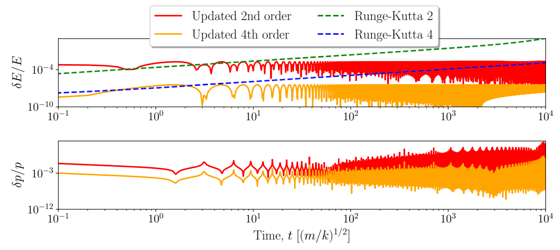

Figure 3. A comparison of the error in the energy, top, and momenta, bottom, as a fraction of the known analytic solution of the system, of the *updated implementation* simulating a damped harmonic oscillator. One can observe that the fractional energy error is bounded and displays the oscillating behaviour that we expect from previous work, whereas the momenta error does not display such behaviour. This plot is based on its equivalent in the original paper [3, Figure 2, bottom].

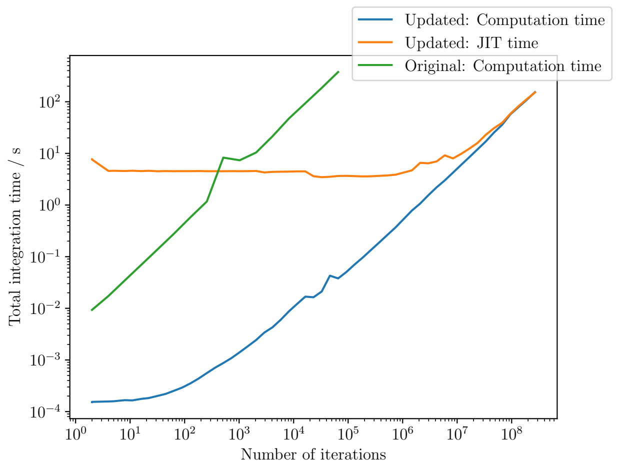

Figure 4. A comparison of running a damped harmonic oscillator system for various numbers of iterations, with $r = 2, \Delta t = 0.1$. The *updated implementation* runtime is split into two components where 'JIT time' represents the one-time fixed cost of compiling the function (see section 2.7) and "Computation time" represents the actual time spent on computation. Each value is the mean of 4 runs, with the *original implementation* being cut off early due after $>$ 20 minutes runtime for the next sample.

successfully replicated the behaviour of the original implementation to an absolute root mean squared residual over the simulation duration of $\Delta = 1.42 \times 10^{-13}$ and $\Delta = 4.50 \times 10^{-12}$ for $r = 2$ and $r = 4$ respectively. This is not the case for the fractional momenta error however as fixed-time-step variational methods such as those implemented cannot be both slimplectic-momentum and momentum-energy preserving [24].

Next, we explored the time complexity of the system in terms of the iteration count, N_{iter}, and GGL quadrature order, r. These are important to physical applications as they determine the limits of iteration accuracy when iterating over large timescales – with error scaling as $(\Delta t)^{2r+2}$ and total timeframe being

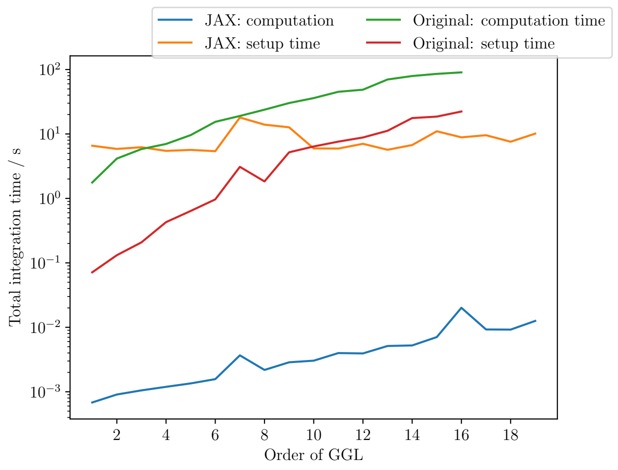

Figure 5. A comparison of running a damped harmonic oscillator system for various values of r, the order of the GGL quadrature, with $N_{\text{iter}} = 500, \Delta t = 0.1$. Each result is the mean of 4 independent runs. For both implementations, we split the overall time into setup and computation as changing the method order requires re-discretising the Lagrangian and thus non-trivial work under both implementations.

$N_{\text{iter}}\Delta t$.

Focusing first on the time complexity in N_{iter} as shown in figure 4, we note that the computation time of the *updated implementation* is much lower than that of the *original implementation*, with the *updated implementation* growing as $\mathrm{O}((N_{\text{iter}})^a)$ where $a = 1.0140 \pm 0.0042$. In addition, we note that the fixed cost JIT compilation remains roughly constant until 10^7 iterations after which it begins to grow linearly.

This pattern is seen also in figure 5 when investigating the time complexity in the order of the method. Again the JIT time remains roughly constant across the domain tested, and though it starts initially higher than the *original implementation*'s setup costs, the *original implementation*'s computation time quickly swamps this fixed cost. Similar to the behaviour in N_{iter}, the computation time for the *updated implementation* is also linear r, remaining insignificant in the overall runtime. This is in comparison with the *original implementation* where the setup time grows as $\mathrm{O}(r^n)$ with $n = 2.31 \pm 0.15$ and the computation time as $\mathrm{O}(r^m)$ with $m = 1.490 \pm 0.070$.

It should be noted that while increasing the order of the method will increase the precision, at higher orders we start encountering limitations with the fixed precision `float64` type used for calculation in the *updated implementation* compared to the arbitrary precision numerics employed in the *original implementation*. Still, however, this represents an overall improvement in accuracy in physically meaningful simulations as the required precision could be more readily attained by decreasing Δt rather than increasing r, avoiding the large increase in runtime observed in the *original implementation*, or the degradation of precision at high r in the *updated implementation.* This is discussed further in section 4.1

4.2. Loss Functions and Optimisation

Moving now onto the loss function and choice of Lagrangian embedding function. We explored several loss functions through a combination of gradient descent and visual inspection.

A wide number of loss functions were tested, however focusing on those surrounding systems of harmonic oscillators, we settled on damped harmonic oscillator (DHO) system-specific embedding shown in equation (23), labelled DHO 3. As well as an adaption where an additional embedding parameter α was introduced as a pre-factor on the entire non-conservative Lagrangian Λ, labelled DHO 4. These were paired with a number of simple loss functions, table 4.2, drawing from physical knowledge and existing conventions for neural network loss functions.

First, we perform a visual inspection of the 'Simple RMS' loss function at various scales with both embedding functions, as seen in figure 6. Here we observe mild non-convexity around the minima, as impulse like behaviour around the highly non-physical $m = 0$ point.

To test how this mild non-convexity translates into optimisation performance we subject a number of combinations to empirical optimisation tests where we attempt to attain a known true embedding from a number of random initial embedding values. Table 4.2 (see caption for details) shows that the addition of the global pre-factor α substantially increases the probability of successful convergence. In addition, we see that both the $\mathbf{q}$ and π values are important in the fitting process from the complete lack of convergence when only π is included and that their relative weight seems to be of little importance at this stage.

This is promising as, while optimisation directly in the embedding space is a strictly simpler problem than with respect to the parameters of an associated neural network, it suggests that this mild non-convexity observed during visual inspection is an obstacle we can overcome.

4.3. PINNs and Approximating Damped Harmonic Oscillators

Finally, we applied these techniques to a PINN. For our initial explorations in this space, we focused on fitting to systems of damped harmonic oscillators as has been the through line of the work thus far. We chose the 3-parameter DHO embedding over the 4-parameter embedding with a pre-factor as while DHO 4 was more effective when subject to direct gradient descent, we were concerned that the additional redundant embedding parameter would make learning the systems more complex and error-prone.

The dataset used in training ($N \approx 5 \times 10^5$) was a mixed selection. Primarily it was comprised of uniformly distributed embeddings within the subset of the physical region of the embedding space (positive embedding values). In addition a fraction, approximately 5% for both, of the spring and damping constants were set to zero to ensure that the model was exposed to non-damped simple harmonic motion and kinetic energy only systems.

For PINN training, accounting for the increased complexity of the optimisation problem now being with respect to the parameters of the model, we made two alterations to our loss functions. First, we added a strong weight against non-physical negative embedding values in our loss function, taking the form of,

$$f(e_i) = \begin{cases} 0 & e_i \geq \delta \\ \exp\{-\gamma e_i\} & e_i < \delta \end{cases} \tag{25}$$

where $\delta = 0.1$ is a tolerance to allow for zero to be a non-penalised output and $\gamma \approx 10$ is the strength of the penalisation. This was done as the non-physical behaviour of negative embedding values is less visible when considering the larger whole model optimisation

Table 1. Different loss functions used, their names and descriptions of their function. Key: RMS = Root Mean Squared, a standard loss function in machine learning.

Name	Description
Simple RMS	Sum of RMS for $\mathbf{q}$ and π in equal weighting.
RMS-(a, b)	Sum of a times the RMS for $\mathbf{q}$ and b times the RMS for π.

Table 2. Results of optimising the chosen loss function from table 4.2 for 200 initially random embeddings (distributed uniformly in $[0, 10]^n$) in the input space of the chosen embedding function, where convergence is defined as being within $\epsilon = 0.05$ of the true embedding at the end of a maximum of 250 iterations of the optimiser. Note that for DHO 4 an additional normalisation step of dividing by α was applied to remove the redundancy introduced by the pre-factor when determining if convergence had been achieved. The true embedding that was optimised towards, in DHO 3 form, was $(m = 2, k = 3, \lambda = 1)$ and systems were compared with $N_{\text{iter}} = 50, r = 2, \Delta t = 0.1, q_0 = 0, \pi_0 = 1$.

Loss function	Embedding	Convergence %
Simple RMS	DHO 3	33.5%
Simple RMS	DHO 4	97.5%
RMS-$(2, 1)$	DHO 4	98%
RMS-$(0, 1)$	DHO 4	0%

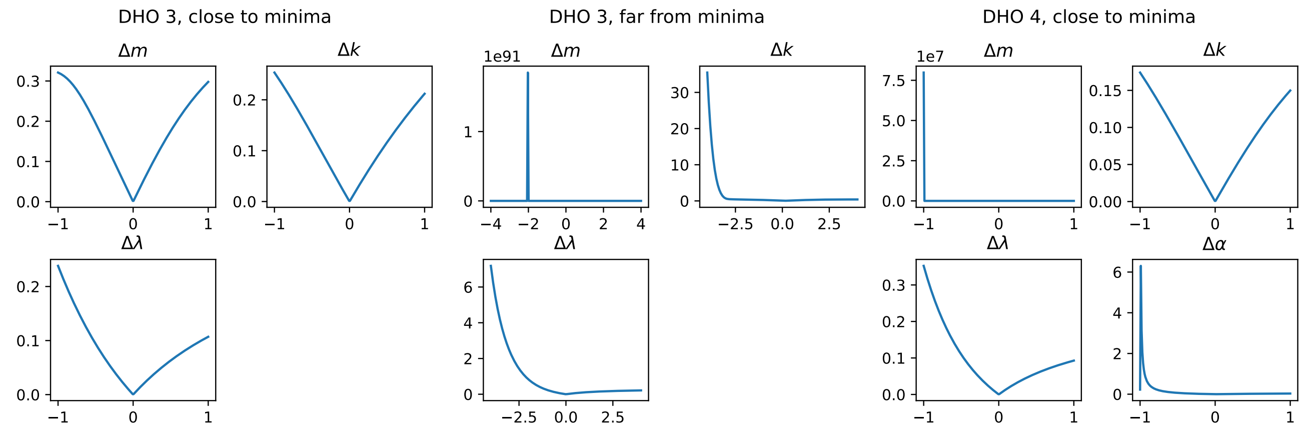

Figure 6. The local behaviour of an equally weighted $\mathbf{q}$ and π RMS loss function, when viewed as a function of variation in each of the embedding variables independently, with others kept fixed. We observe roughly correct behaviour in all variables bar mass m and subsequently α for the DHO 4 embedding where α acts as a pre-factor for the whole non-conservative Lagrangian Λ. This behaviour is to be expected as a system without mass is non-physical and explodes to infinity.

problem. In addition, we also experimented with capping the values of the loss at large values to mitigate overflows when dealing with particularly bad fits, such as the initially random initialised weights before any training had commenced. Both of these alterations notably improved training stability and decreased the probability of floating-point related errors.

Model training was done in batches and was prone to explosion possibly due to the non-convexity in some regions as discussed in section 4.2. Overall it was found that training could be made more stable by increasing batch sizes to 512 from 32, and manually tweaking learning rates and loss weightings (for example between the $\mathbf{q}$ and π RMS terms where more progress was made with weighting towards $\mathbf{q}$ error to counteract observed larger tendency for error in this term) as training progressed.

Other loss functions, such as RMS in π only (RMS-(0, 1) in table 4.2), were once again investigated. This initially looked promising, producing good losses on the order of 10^{-4}, however on further inspection it was found that these resulted from a finding a false minimum of $m \approx k \approx \lambda$, highlighting the complexity of the embedding space in representing physical systems. This also underlined the utility of the previous direct embedding space optimisation done prior in informing us about the behaviour of the loss function itself.

By the training's conclusion we obtained an absolute RMS error of $\pm 5 \times 10^{-2}$ in $\mathbf{q}$ and $\pm 1 \times 10^{-1}$ in π. This model was able to predict systems with some accuracy, clearly able to fit to certain features as shown in figure 7.

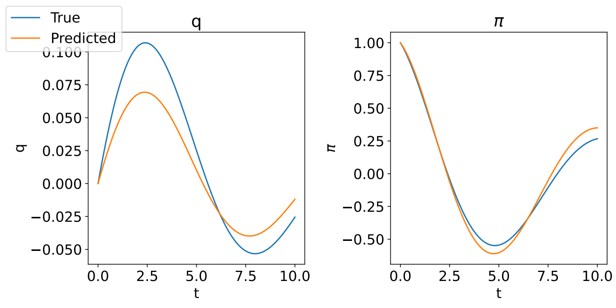

Figure 7. A comparison of the behaviour of a Lagrangian embedding predicted by a PINN. True embedding, $(m = 6, k = 2, \lambda = 3)$, predicted embedding, $(m = 9.401553, k = 3.3729377, \lambda = 3.9162085)$.

5. Discussion

5.1. Slimplectic Integrator

We have presented a number of results on the performance characteristics of our *updated implementation* of the Slimplectic Integrator method applied to physical problems. These warrant additional discussion to place them in context and examine their implications for experimental work.

Starting with the effect of changing the method order r, as presented in figure 5. Roughly constant-time behaviour is observed and is expected to continue for higher values of r, as this reflects very little change in the size of the underlying calculation (due to data size and calculation cost) involved in the integration process. This, coupled with the ability to split long integration runs (high N_{iter} values) into multiple successive runs, means, that with careful management of errors (especially with regards to round-off build-up due to fixed precision floats), we should be able to limit the degradation to the linear performance observed in figure 4 for $N_{\rm iter} > 10^7$. From this $N_{\rm iter}$ value the method's internal arrays become greater than 1Gb in size possibly hitting JAX de-optimisation.

It is important to note however, that in the high order domain $r \gtrsim 40$ the computation of the derivative matrix, given in Equation (12), begins to fail as we encounter issues with the `float64` floating point precision used in JAX calculations. Unfortunately, this is unavoidable as it is the maximum precision offered by JAX. Nonetheless, this limitation can be mitigated, whilst retaining the desired error characteristics by employing alternative strategies, such as reducing the time step or using a higher-order method with a lower value.

Overall, however, this approach is made more fruitful by noting that we can avoid the fixed cost entirely if we reuse the same of form non-conservative Lagrangian. This allows us to vary not only initial conditions, for example exploring the same system in different circumstances or restarting after multiple runs, but also sample across a range of physical parameters – for example those parameterising the DHO 3 embedding as defined in Equation (23). In particular, this scaling without fixed costs applies effectively to systems composed of many repeated sub-systems, such as field theory or molecular dynamics simulations [25].

As shown in figure 4, we observe a transient increase in runtime over the range $r \in [7, 9]$. This phenomenon is repeatable and its origin is uncertain. However, informed by the size of the quadrature array being $r + 2$, we suspect that this behaviour is due to optimisation cliffs in the JAX internal code, as we transition between two internal implementations optimised for small vs large array sizes. Finally, it is important to consider that for less trivial systems, where evaluating the Lagrangian may bring its own performance considerations, the method may require further analysis and optimisation.

Looking forward, we noted that the use of fixed time steps in the method limits our ability to maintain energy and momentum fractional error bounds. Moving to an adaptive time step approach would be a promising avenue for improvement in future work.

5.2. PINN, Current and Future

The trained neural network represents a promising first step in predicting non-conservative Lagrangians from observational data, successfully capturing the general characteristics of the data. During our experiments, we frequently observed predicted embeddings with values of roughly correct proportion, but off by a constant factor. This suggests that the 4 parameter embedding, or other normalisation methods, may have been useful as the model was unable to move past this local minima in phase space, representing a sufficiently similar physical system. This poses questions on the handling of degeneracy in our Lagrangian space, where multiple Lagrangians can result in the same physical system, especially within the image of any chosen embedding function, as this degeneracy will present issues when optimising within the space.

Furthermore, we noted that the model was substantially more accurate with comparatively high masses. We theorise that this may be due to difficulties in fitting larger values of $\mathbf{q}$, possibly indicating the need for a larger and more varied dataset. It is important to acknowledge that this was a simplified test network, and we anticipate that future work will achieve improvements by utilising a larger parameter count and datasets.

To enhance the model's performance, incorporating physically known a priori information into the loss function may be beneficial. For example, a stricter restriction on non-negativity for the DHO 3 parameters could be implemented by first taking the absolute value of the outputs as inputs to the Slimplectic Integrator. This would have the effect of cleanly solving the problem of non-physical embedding values, at the cost of introducing additional degeneracy into our space possibly confusing the model.

It should be noted, however, that this and similar approaches, present a trade-off between specialising our embedding and loss functions towards a particular class of physical systems where we have more physical knowledge, and on the other hand, creating models which are more widely applicable, where we are unable to draw such physically derived conclusions (for example when using Taylor series embedding functions). Identifying physical laws that can be enforced in the Lagrangian would be a fruitful avenue for future research.

Following this theme, a suitably trained neural network could be constructed using this method to identify symmetries in observational data or recognise patterns it has previously been exposed to in new observational data. This suggests potential applications in domains such as astrophysics, where there is a large availability of data to train a model, but also difficulties in obtaining a complete dataset for any one experiment.

6. Conclusion

In this paper, we have put forward a more optimal implementation of the Slimplectic Integrator, which is capable of scaling to large-scale physical simulations while preserving bounded fractional error in the energy.

We have shown this implementation has applications in the construction of a loss function for physics-informed neural networks, and presented a simple model which shows that this approach has promise for identifying non-conservative Lagrangians from observational data. More work is required to explore better encodings of physical invariants in the loss function and model structure.

Acknowledgments

My gratitude is given to my project partner Aengus Roberts who was a guiding light throughout the project, and who spent many hours designing, training and monitoring the network. Without him, this would not have been possible. I would also like to thank Dr David Tsang for his constant support and discussion as we have moved through this project.

References

[1] Gladman B, Duncan M and Candy J 1991 *Celestial Mechanics and Dynamical Astronomy* **52** 221–240 ISSN 0923-2958, 1572-9478

[2] Wisdom J and Holman M 1991 *The Astronomical Journal* **102** 1528–1538 ISSN 0004-6256

[3] Tsang D, Galley C R, Stein L C and Turner A 2015 *The Astrophysical Journal* **809** L9 ISSN 2041-8213

[4] Galley C R 2013 *Physical Review Letters* **110** 174301 ISSN 0031-9007, 1079-7114

[5] Galley C R, Tsang D and Stein L C 2014

[6] Meurer A, Smith C P, Paprocki M, Čertík O, Kirpichev S B, Rocklin M, Kumar A, Ivanov S, Moore J K, Singh S, Rathnayake T, Vig S, Granger B E, Muller R P, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry M J, Terrel A R, Roučka v, Saboo A, Fernando I, Kulal S, Cimrman R and Scopatz A 2017 *PeerJ Computer Science* **3** e103 ISSN 2376-5992 URL `https://doi.org/10.7717/peerj-cs.103`

[7] Goldstein H 2000 *Classical Mechanics* 2nd ed Addison-Wesley Series in Physics (Reading, Mass.: Addison-Wesley) ISBN 978-0-201-02918-5

[8] Noether E and Tavel M A 1971 *Transport Theory and Statistical Physics* **1** 186–207 ISSN 0041-1450, 1532-2424 (*Preprint* `physics/0503066`)

[9] Strutt J W 1871 *Proceedings of the London Mathematical Society* **s1-4** 357–368 ISSN 00246115

[10] Tsang D and Hamilton D P

[11] DeVries P L and Hasbun J E 2011 *A First Course in Computational Physics* 2nd ed (Sudbury, Mass.: Jones and Bartlett Publ) ISBN 978-0-7637-7314-4

[12] Farr W M and Bertschinger E 2007 *The Astrophysical Journal* **663** 1420 ISSN 0004-637X

[13] Raissi M, Perdikaris P and Karniadakis G E 2017 Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations (*Preprint* `1711.10566`)

[14] Lu L, Meng X, Mao Z and Karniadakis G E 2021 *SIAM Review* **63** 208–228 ISSN 0036-1445

[15] Meng X and Karniadakis G E 2020 *Journal of Computational Physics* **401** 109020 ISSN 0021-9991

[16] Hornik K, Stinchcombe M and White H 1989 *Neural Networks* **2** 359–366 ISSN 0893-6080

[17] Kingma D P and Ba J 2017 Adam: A Method for Stochastic Optimization (*Preprint* `1412.6980`)

[18] Daubechies I, Defrise M and De Mol C 2003 An iterative thresholding algorithm for linear inverse problems with a sparsity constraint (*Preprint* `math/0307152`)

[19] Sra S, Nowozin S and Wright S J (eds) 2012 *Optimization for Machine Learning* Neural Information Processing Series (Cambridge, Mass: MIT Press) ISBN 978-0-262-01646-9

[20] 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`

[21] OpenXLA Project and collaborators 2022 XLA: Accelerated linear algebra `https://github.com/openxla/xla`

[22] Tsang D 2016 Slimplectic integrator URL `https://github.com/davtsang/slimplectic`

[23] Hochreiter S and Schmidhuber J 1997 *Neural Computation* **9** 1735–1780 ISSN 0899-7667, 1530-888X

[24] Zhong G and Marsden J E 1988 *Physics Letters A* **133** 134–139 ISSN 0375-9601

[25] Tuckerman M E and Martyna G J 2000 *The Journal of Physical Chemistry B* **104** 159–178 ISSN 1520-6106