

LEARNING EFFICIENT AND PROVABLY CONVERGENT SPLITTING METHODS*

LISA M. KREUSSER[†], HENRY E. LOCKYER[‡], EIKE H. MÜLLER[‡], AND PRANAV
SINGH[‡]

Abstract. Splitting methods are widely used for solving initial value problems (IVPs) due to their ability to simplify complicated evolutions into more manageable subproblems. These subproblems can be solved efficiently and accurately, leveraging properties like linearity, sparsity and reduced stiffness. Traditionally, these methods are derived using analytic and algebraic techniques from numerical analysis, including truncated Taylor series and their Lie algebraic analogue, the Baker–Campbell–Hausdorff formula. These tools enable the development of high-order numerical methods that provide exceptional accuracy for small timesteps. Moreover, these methods often (nearly) conserve important physical invariants, such as mass, unitarity, and energy. However, in many practical applications the computational resources are limited. Thus, it is crucial to identify methods that achieve the best accuracy within a fixed computational budget, which might require taking relatively large timesteps. In this regime, high-order methods derived with traditional methods often exhibit large errors since they are only designed to be asymptotically optimal. Machine Learning techniques offer a potential solution since they can be trained to efficiently solve a given IVP with less computational resources. However, they are often purely data-driven, come with limited convergence guarantees in the small-timestep regime and do not necessarily conserve physical invariants. In this work, we propose a framework for finding machine learned splitting methods that are computationally efficient for large timesteps and have provable convergence and conservation guarantees in the small-timestep limit. We demonstrate numerically that the learned methods, which by construction converge quadratically in the timestep size, can be significantly more efficient than established methods for the Schrödinger equation if the computational budget is limited.

Key words. Initial value problems, Geometric numerical integration, Operator splitting, Machine learning, Convergence, Computational efficiency, Schrödinger equation.

MSC codes. 34A26, 34L40, 65B99, 65L05, 65L20, 65Y20

1. Introduction. In this paper we consider first order initial value problems (IVPs) which arise in a wide range of physical applications. They encompass systems of ordinary differential equations (ODEs) with a finite dimensional state space, as well as the more general case of partial differential equations (PDEs) expressed as ODEs on an infinite dimensional Hilbert space \mathfrak{H} . Mathematically, first order IVPs can be written as

$$(1.1) \quad \dot{u}(t) = f(u, t), \quad u(0) = u_0, \quad u(t) \in \mathfrak{H}, \quad t \in [0, T],$$

for some $T > 0$, where u is some state that evolves in time t from a given initial state u_0 under the action of a vector field f and \dot{u} denotes the derivative of u with respect to time t . In general, it is not possible to solve IVPs like (1.1) analytically; even where closed form solutions do exist, their evaluation is often prohibitively expensive computationally. Thus we require numerical methods that are stable, accurate and computationally efficient. These are typically realised in terms of time-stepping methods where, for the sake of simplicity, we consider the evolution to the final time $T = Nh$ as being split into N equal steps of size $h \ll T$. A one-step numerical method

*Submitted to the editors November 15, 2024.

Funding: Henry Lockyer is supported by a scholarship from the EPSRC Centre for Doctoral Training in Statistical Applied Mathematics at Bath (SAMBa), under the project EP/S022945/1.

[†](Corresponding author).

[‡]Department of Mathematical Sciences, University of Bath, Bath, BA2 7AY, United Kingdom (lmk54@bath.ac.uk, hl785@bath.ac.uk, em459@bath.ac.uk, ps2106@bath.ac.uk).

is then uniquely defined by the *forward map* from the approximate solution at a given time t to the approximate solution at time $t + h$.

Splitting and composition methods [6] allow the separation of the evolution in (1.1) into simpler IVPs which can be solved efficiently and accurately; as a consequence they have been used successfully in many areas, see e.g. [32]. In particular, we assume that the vector field $f = f^{[1]} + f^{[2]}$ can be split into two components $f^{[1]}$ and $f^{[2]}$, each of which defines an IVP that is simpler to solve numerically; as shown in [18] this is indeed the case for a wide range of applications.

Traditionally, splitting and composition methods are derived using analytic and algebraic conditions, including truncated Taylor series and their Lie algebraic analogue, the Baker–Campbell–Hausdorff (BCH) formula [23, 9] to guarantee consistency (or local error) of high order. Once stability is ensured, this leads to high-order convergence (of the global error) in the asymptotic limit of small timesteps $h \rightarrow 0$, see e.g. [37, 28, 32, 5]. However, the asymptotic convergence of traditional methods implies that they are most efficient for small timestep h , which correspond to significant computational cost. It is also not obvious that a method which is designed to be fast in the limit $h \rightarrow 0$ will be the best choice for larger h . To make these points explicit we define two criteria to characterise an optimal numerical method:

C1: The method has the fastest decrease in error in the asymptotic regime $h \rightarrow 0$.

C2: The method has the smallest error for a given limited computational budget.

Traditionally, the focus has been on constructing methods that satisfy criterion **C1**. This, however, is not helpful if computational resources are limited and simulations have to be carried out at relatively large timestep sizes. This scenario, which is of significant practical interest, is the case we consider in this work. Our aim is to construct numerical methods with machine learning techniques such that they satisfy criterion **C2**, while being provably convergent in the limit $h \rightarrow 0$.

The exact solutions of IVPs often conserve a range of quantities that are known as invariants or first integrals. For example, for the Schrödinger equation the norm of the complex-valued solution does not change with time, which physically translates to the conservation of total probability, see [35]. More generally, for the Schrödinger equation the evolution of the IVP preserves the inner product of the underlying Hilbert space, a property known as unitarity, as well as the total energy, see [35]. Similarly, Hamiltonian systems conserve a symplectic two-norm in phase space, see [20]. Such invariants often have a physical interpretation and are typically related to conservation laws. It is highly desirable to construct numerical methods which preserve these invariants at least approximately. This usually also improves the stability of the method since it limits the trajectory to a sub-manifold of the Hilbert space. An important example is the class of symplectic integrators, see e.g. [20].

Machine Learning (ML) methods offer a promising alternative to traditional numerical methods for determining a numerical solution to an IVP, see e.g. [24] for a recent review. The key idea is to learn a forward solution map by training on a large set of initial and final values for the IVP. Provided the examples used for training are sufficiently representative of the (distribution of) initial conditions we are interested in solving, these methods can achieve high accuracy on unseen initial conditions from the same distribution and potentially also generalise to a wider class of IVPs. In their simplest form, ML methods are purely data-driven. They learn a fixed-cost forward solution map which is typically represented by a neural network. Unless measures are taken to specifically enforce inductive priors (see e.g. [33, 31, ?]), ML techniques do not guarantee the conservation of invariants such as unitarity and symplecticity which can be enforced with traditional splitting methods.

With few exceptions such as [11], ML methods suffer from their detachment from the differential equation framework: since they do not parametrise the forward map as a function of the timestep size h , there is no sense in which convergence in the limit $h \rightarrow 0$ can be quantified. In contrast to traditional numerical methods which can be made more accurate by reducing the timestep size, it is not possible to achieve higher accuracy for these ML methods in a controlled way. Even ML methods that learn the forward map with a variable timestep size h do not necessarily generalise to different timestep sizes. In particular they may fail to converge in the limit $h \rightarrow 0$. Moreover, the stability of the forward map – a crucial component for establishing the convergence – is exceptionally hard to guarantee.

However, provided sufficient training data is available, the model is sufficiently expressive, and the loss function weights large time step performance sufficiently, ML based methods have the potential to result in smaller overall errors, and thus superior performance for specific computational budgets and larger timesteps (in the sense of criterion **C2**) than traditional methods which aim to be efficient in the asymptotic regime (in the sense of criterion **C1**), see [24, 19].

1.1. Contributions. In this paper we combine techniques from numerical analysis and ML to find efficient splitting schemes in the sense of criterion **C2** while being provably convergent in the limit $h \rightarrow 0$. Specifically we use machine learning to find splitting schemes that are tailored for a specific distribution of initial conditions, yet maintain many of the advantages of classical numerical methods, such as interpretability, generalisability, convergence, and conservation. As our numerical results show, the learned methods also generalise to scenarios not contained in the training set.

More specifically, we use ML methods to learn coefficients of splitting methods, while algebraically enforcing conditions that guarantee desirable properties such as consistency, stability and reversibility. We demonstrate the utility of this framework by learning splittings of medium to long length which result in lower errors than classical methods for large step sizes h . Restricting our search to physically plausible methods allows us to search a significantly lower dimensional submanifold. This reduces the training time compared to naive black-box ML approaches while also guaranteeing convergence. We numerically verify that the learned methods are efficient for the Schrödinger equation with a double well potential and also demonstrate that they have the desirable conservation properties. While by construction the learned methods are formally only quadratically convergent in h , we also show that we can learn near fourth order methods.

2. Traditional numerical analysis methods. We restrict our attention to well-posed IVPs, using the definition of well-posedness found in [16]. Hence any IVP of the form (1.1) considered in this work has a unique solution $u(t)$ for all $t \in [0, T]$ and is differentiable with respect to time and initial conditions. Arguments about existence and uniqueness of strong solutions to PDEs of the form (1.1) are active areas of research and are beyond the scope of this paper. Specifically for the linear Schrödinger equation existence and uniqueness of solutions under time-varying potentials are discussed in [38, 34], while for the case of time-independent potentials the theory of one-parameter semi-groups can be applied, see [15]. Once these PDEs are semi-discretised into a system of ODEs with the method of lines, the Picard-Lindelöf theorem provides conditions on the existence of unique solutions, see for instance [17]. In the case of time-independent potentials a computable closed-form expression for this unique solution is provided by the matrix exponential. However, evaluating this directly can be prohibitively expensive in practice [29].

2.1. Flows. As we have assumed the well-posedness of our IVPs, we know there exists a unique solution $u(t)$ for all times $t \in [0, T]$. For a given time difference h and vector field f we define the analytic flow $\psi_h^{[f]} : \mathfrak{S} \rightarrow \mathfrak{S}$ as the solution map

$$(2.1) \quad \psi_h^{[f]}(u(t)) = u(t+h),$$

which maps a state $u(t)$ to its unique evolution $u(t+h)$ after time h under (1.1). As the analytic flow $\psi_h^{[f]}$ encodes the true solutions of a differential equation, its precise form is typically unavailable or computationally infeasible. Numerical time-stepping methods, see for example [20, 21], seek approximations to the analytic flow by approximating the solution at discrete times,

$$(2.2) \quad u_n \approx u(t_n), \quad n \in \{0, 1, \dots, N\},$$

where $N \in \mathbb{N}^+$ is the number of timesteps of size $h = \frac{T}{N} \in \mathbb{R}^+$ and $t_n = nh$, with $t_0 = 0$ and $t_N = T$. We consider one-step methods where the numerical solution u_{n+1} is the result of applying a numerical method (which may involve multiple evaluations of f , or its subcomponents $f^{[1]}$ and $f^{[2]}$ in the case of splitting methods) to the previous state u_n . This can be seen as a numerical *forward map* $\Psi_h^{[f]} : \mathfrak{S} \rightarrow \mathfrak{S}$,

$$(2.3) \quad \Psi_h^{[f]}(u_n) = u_{n+1}.$$

Inspired by the definition of the analytic flow in equation (2.1) we define the numerical flow as the map that sends the numerical state u_ν to its evolution under n timesteps of the forward map in (2.3),

$$(2.4) \quad \Psi_{t_n, h}^{[f]}(u_\nu) = \left(\Psi_h^{[f]}\right)^n(u_\nu) = u_{\nu+n}.$$

Note that we use the upper case Ψ for numerical flow and the lower case ψ for analytical flow. Unlike the exact flow in (2.1), the numerical flow in (2.4) is only defined at the discrete time points t_n , for $n \in \{0, 1, \dots, N\}$. A numerical method and its associated numerical flow $\Psi_{t_n, h}^{[f]}$ have order of convergence $p \in \mathbb{N}^+$ if for each sufficiently regular [20, 6] f and u_0 there exists a constant $C(f, u_0, T)$ independent of h such that

$$(2.5) \quad \left\| \Psi_{t_n, h}^{[f]}(u_0) - \psi_{t_n}^{[f]}(u_0) \right\| \leq Ch^p,$$

for all n as $h \rightarrow 0$. In other words, the numerical method is convergent of order p and the global error is of the order $\mathcal{O}(h^p)$. Convergence requires both consistency and stability which imply that the numerical and analytical flows agree up to the $p-1$ th term in the Taylor expansion in h .

2.2. Splittings. For the differential equation of the form (1.1) we assume that the vector field, $f = f^{[1]} + f^{[2]}$, can be split into two (sub-)components, $f^{[1]}$ and $f^{[2]}$. We further assume that the two IVPs, defined by the flow under the individual vector fields $f^{[1]}$ and $f^{[2]}$, are well-posed and have known analytic solutions which can readily be computed. The corresponding analytic flows are called the “subflows” and denoted by $\psi_t^{[1]}$ and $\psi_t^{[2]}$ respectively. A numerical method for the IVP in (1.1) can be constructed by interleaving the sub-flows. Two such well known splitting methods are

$$(2.6) \quad \text{Trotter} : \Psi_h^{[f]} = \psi_h^{[2]} \circ \psi_h^{[1]} \quad \text{and}$$

$$(2.7) \quad \text{Strang} : \Psi_h^{[f]} = \psi_{0.5h}^{[1]} \circ \psi_h^{[2]} \circ \psi_{0.5h}^{[1]}.$$

Trotter and Strang are methods of order one and two respectively, in the sense of (2.5), see [6]. As a consequence, the Taylor coefficients of the analytical and numerical flows for all points in the temporal discretisations agree for the constant, h^0 term for both methods; for Strang the coefficients of the h^1 terms also agree. Both methods are part of the larger family of splitting schemes that can be parameterised as follows,

$$(2.8) \quad \Psi_{T,h}^{[f]}(\cdot; \alpha, \beta) = \bigcirc_{n=1}^N \Psi_h^{[f]}(\cdot; \alpha, \beta), \text{ where } \Psi_h^{[f]}(\cdot; \alpha, \beta) = \bigcirc_{k=1}^K \psi_{\beta_k h}^{[2]} \circ \psi_{\alpha_k h}^{[1]}.$$

Here, \bigcirc represents repeated composition, i.e. $\bigcirc_{n=1}^N \varphi_n = \varphi_N \circ \dots \circ \varphi_2 \circ \varphi_1$ and $\Psi_{T,h}^{[f]}(\cdot; \alpha, \beta)$, $\Psi_h^{[f]}(\cdot; \alpha, \beta)$, $\psi_{\alpha_k h}^{[1]}$ and $\psi_{\beta_k h}^{[2]}$ map from \mathfrak{H} to \mathfrak{H} . We call a splitting of the form (2.8) a K -stage method with N individual steps and a timestep of size $h = T/N$. Observe that in general (2.8) requires a total of $2KN$ sub-flow evaluations. The vector of $2K$ parameters $[\alpha, \beta] = [\alpha_1, \alpha_2, \dots, \alpha_K, \beta_1, \beta_2, \dots, \beta_K]$ continuously parameterises all¹ possible splittings of maximum length K .

We assume differentiability of the parameterised numerical flow $\Psi_{T,h}^{[f]}(\cdot; \alpha, \beta)$ with respect to the parameters α_k, β_k . To ensure this, it suffices to assume the differentiability of the analytical sub-flows $\psi_{\alpha_k h}^{[1]}$ and $\psi_{\beta_k h}^{[2]}$ with respect to time and initial conditions which naturally follows from the assumed well-posedness of the sub-flows.

2.3. Order conditions. While some particular IVPs allow exact splittings [4, 3], in general splitting methods introduce errors which depend on the particular IVP that is solved. Nevertheless, some splitting schemes do have (near) universal desirable properties. For example, imposing so-called order conditions on the parameters α and β ensures that the corresponding numerical methods have a particular order. The ‘first order consistency’ order condition, henceforth simply called “consistency”, is given by,

$$(2.9) \quad \sum_{k=1}^K \alpha_k = 1 \text{ and } \sum_{k=1}^K \beta_k = 1.$$

It can then be shown that (2.9) implies convergence with order $p = 1$ in the sense of (2.5) if the scheme is also stable. Moreover, any splitting scheme with convergence of order $p \geq 1$ necessarily needs to satisfy the consistency condition (2.9). Thus, without loss of generalisation, we assume in the remainder of the manuscript that the consistency condition (2.9) is always satisfied.

The simplest consistent splitting scheme with the smallest value of K is the Trotter method defined by the parameters $[\alpha_1, \beta_1] = [1.0, 1.0]$, which satisfy (2.9). For longer splittings with larger K we can impose further order conditions on the parameters α_k and β_k . For example, enforcing symmetry under time-reversal ensures that the splitting error is an even function of the timestep size, i.e. all odd powers of h in the Taylor expansion in h vanish, and hence,

$$(2.10) \quad \Psi_{T,h}^{[f]} = \left(\Psi_{-T,-h}^{[f]} \right)^{-1} \Rightarrow \Psi_{T,h}^{[f]}(u) = \psi_T^{[f]}(u) + \mathcal{O}(h^p), \quad p \in 2\mathbb{Z}.$$

When combined with stability, symmetry under time-reversal, henceforth simply called “symmetry”, implies even order convergence. Since we assume consistency, symmetry ensures convergence of order at least $p = 2$. As shown in [6], symmetry can

¹Technically in (2.8) we have imposed that the flow $\psi_h^{[1]}$ associated with $f^{[1]}$ comes first, precluding some splittings, but this can be solved by relabelling $f^{[1]} \leftrightarrow f^{[2]}$.

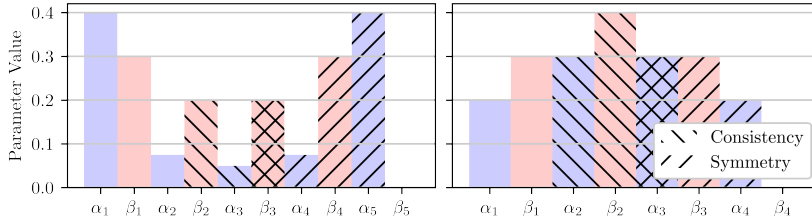


FIG. 1. Visualisation of the α and β for two symmetric and consistent splitting schemes with $K = 5$ (left) and $K = 4$ (right) stages. Note how symmetry constrains half the parameters, including the trailing β_K that is symbolically zero. Consistency fixes two additional parameters to ensure that the α and β sum to one as in (2.9).

be guaranteed if the coefficient vectors that define the scheme in (2.8) are palindromic, i.e. of the form $(\alpha_1, \alpha_2, \dots, \alpha_K) = (\alpha_K, \alpha_{K-1}, \dots, \alpha_1)$ and $(\beta_1, \beta_2, \dots, \beta_{K-1}, 0) = (\beta_{K-1}, \beta_{K-2}, \dots, \beta_1, 0)$ with the symbolic zero $\beta_K = 0$. Enforcing consistency and symmetry imposes constraints on α and β and hence reduces the number of degrees of freedom that define the splitting scheme: to parameterise all possible symmetric and consistent splittings we only require $K - 2$ independent parameters rather than the $2K$ parameters required to parameterise all splittings. Since these conditions are straightforward to impose and reduce the dimension of the search space, we restrict our attention to symmetric and consistent methods for the rest of the manuscript. The parameter of two such methods are visualised in Figure 1.

2.4. Composing splitting methods. The simplest, and indeed unique, splitting scheme of length two that is both consistent and symmetric (and hence second order) is Strang, defined by the parameters $[0.5, 0.5, 1.0, 0.0]$. However, finding higher-order conditions requires significant work. An alternative is to build up higher-order methods by composing multiple copies of lower order methods, see [20, II.4]. An example of a composition method is the triple jump technique which we discuss in more detail in Appendix A and which can be applied to Strang to construct a splitting method of order four, referred to as Yoshida in this paper. A disadvantage with using composition methods to find new splittings is that they implicitly restrict the methods that can be found, and the number of stages grows exponentially with the desired order.

2.5. Cost estimates. The symbolic zero $\beta_K = 0$ in symmetric methods is worth remarking on as the costs for the evaluation of symmetric and general methods differs significantly. We denote the cost of a single evaluation of the subflows $\psi_{\alpha_k h}^{[1]}$, $\psi_{\beta_k h}^{[2]}$ by $\mathcal{C}^{[1]}$ and $\mathcal{C}^{[2]}$ respectively. For evaluating N individual steps of a general K -stage method, the total cost is given by $NK(\mathcal{C}^{[1]} + \mathcal{C}^{[2]})$. For symmetric K -stage methods, composing N individual steps does not require the evaluation of the identity flows associated with the symbolic zeros $\beta_K = 0$, while adjacent sub-flows associated with $f^{[1]}$ can be combined since $\psi_{\alpha_1 h}^{[1]} \circ \psi_{\alpha_K h}^{[1]} = \psi_{(\alpha_1 + \alpha_K)h}^{[1]}$. For symmetric methods, the total cost therefore satisfies $N(K\mathcal{C}^{[1]} + (K - 1)\mathcal{C}^{[2]}) - (N - 1)\mathcal{C}^{[1]} = N(K - 1)(\mathcal{C}^{[1]} + \mathcal{C}^{[2]}) + \mathcal{C}^{[1]}$, where the subtraction of $(N - 1)\mathcal{C}^{[1]}$ accounts for the reduction of the cost due to combination of sub-flows. Overall, this leads to relative cost savings of a factor $(K - 1)/K + \mathcal{O}(h)$ per timestep. To summarise, the total cost of numerically integrating the IVP (1.1) to the final time T with a K -stage splitting method and a

timestep size of $h = T/N$ is given by,

$$(2.11) \quad \mathcal{C} = \left\{ \begin{array}{ll} N(K-1)(\mathcal{C}^{[1]} + \mathcal{C}^{[2]}) + \mathcal{C}^{[1]} & \text{if symmetric} \\ NK(\mathcal{C}^{[1]} + \mathcal{C}^{[2]}) & \text{in general} \end{array} \right\} \geq C'(K, T)h^{-1},$$

where the constant,

$$(2.12) \quad C'(K, T) = T \cdot \left\{ \begin{array}{ll} (K-1)(\mathcal{C}^{[1]} + \mathcal{C}^{[2]}) & \text{if symmetric,} \\ K(\mathcal{C}^{[1]} + \mathcal{C}^{[2]}) & \text{in general,} \end{array} \right.$$

is independent of h and the bound in (2.11) is sharp up to corrections of relative $\mathcal{O}(h)$. As a consequence, up to corrections of $\mathcal{O}(h)$, which are amortised for $N \gg 1$, a single step of symmetric Strang ($K = 2$) costs the same as a step of non-symmetric Trotter ($K = 1$).

2.6. Computational efficiency. To choose an optimal numerical time-stepping method, we aim to minimise the computational cost $\mathcal{C} = \mathcal{C}(\varepsilon)$ for a given tolerance ε on the numerical error $\left\| \Psi_{T,h}^{[f]}(u_0) - \psi_T^{[f]}(u_0) \right\|$. If the method $\Psi_{T,h}^{[f]}(u_0)$ is convergent of order p , then according to (2.5) the numerical error at the final time is smaller than the tolerance ε provided that the timestep size h is chosen such that

$$(2.13) \quad C(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])h^p \leq \varepsilon.$$

Here we have made the dependence of the constant C on the timestepping method explicit through the parameters $[\alpha, \beta]$. Combining (2.11) and (2.13), we see that the total cost as a function of the tolerance ε is bounded from below by

$$(2.14) \quad \mathcal{C}(\varepsilon) \geq C'(K, T)C(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])^{1/p}\varepsilon^{-1/p}.$$

Provided that the bounds in (2.11) and (2.13) are sharp, this implies that in the limit $\varepsilon, h \rightarrow 0$ high-order methods will be most cost effective. However, in practical applications choosing a numerical method based on this argument (which corresponds to criterion **C1**) might be premature for two reasons: firstly, the constants that appear in (2.14) also depend on the timestepping method and might be of a similar size as the factor $\varepsilon^{-1/p}$ for moderate tolerances ε . Consider, for example, the discussion in Appendix A: composing splitting methods with the triple-jump technique leads to a $\mathcal{O}(3^p)$ exponential growth in the number of stages, K , and thus to a similar increase of $C'(K, T)$. As a consequence, for a fixed computational budget (criterion **C2**) and thus moderately large step size h , a smaller error might be obtained with a lower-order splitting method with a small error constant $C(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])$ than by resorting to a higher-order method which may have a larger error constant. This effect is particularly pronounced when low to moderate levels of accuracy ε are required or acceptable, a typical scenario in applications which are either significantly constrained by computational cost or where the IVP only provides a moderately accurate model of the underlying physical system. Hence, at any given order it is highly desirable to minimise the constant $C(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])$.

2.7. Identifying efficient splitting methods. To a certain extent studies such as [32] attempt to obtain a lower error constant by considering a large number of splittings, typically obtained by solving the algebraic order conditions, and identifying the splittings that feature the smallest coefficients accompanying the leading commutators

terms in the residual BCH expansion. By design, this strategy is only asymptotically relevant, i.e. for small step size h . Moreover, it ignores the magnitude of the commutators and hence does not minimize the constant $C(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])$ that also depends on the particular IVP and its splitting given by $f^{[1]}, f^{[2]}$, the initial condition u_0 and the final time T . These factors are not straightforward to incorporate in an algebraically motivated approach.

When the subflows are unitary or symplectic, a splitting method guarantees the exact conservation of a shadow Hamiltonian or near conservation of the true Hamiltonian, and the error in energy is sometimes more relevant than the L_2 -error of the solution. In the context of the Hamiltonian Monte Carlo sampling method, for instance, it becomes possible to obtain optimal splittings for larger step sizes by minimizing the expected error in energy under certain analytic priors on the problem [1, 7].

2.8. A data-driven approach for minimising the error constant. Our goal is to construct a method which is robust in the sense that it produces a small error for a range of different vectors fields f and splittings $f^{[1]}, f^{[2]}$, as well as different initial conditions u_0 and final times T . For example, keeping $f^{[1]}, f^{[2]}$ and T fixed, we might be interested in,

$$(2.15) \quad \overline{C}(f^{[1]}, f^{[2]}, T; [\alpha, \beta]) = \sup_{u_0 \in \mathbb{U}} \left\{ C(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta]) \right\},$$

for all possible initial conditions u_0 from some given set \mathbb{U} . This would allow replacing the specific bound in (2.13) by the more general bound,

$$(2.16) \quad \overline{C}(f^{[1]}, f^{[2]}, T; [\alpha, \beta]) h^p \leq \varepsilon,$$

which holds for all $u_0 \in \mathbb{U}$. Similarly, we might want to make the bound problem-independent by considering the supremum over a range of f and splittings $f^{[1]}, f^{[2]}$. However, it is usually not possible to write down the dependence of C (let alone \overline{C}) on $f^{[1]}, f^{[2]}, u_0$ and T in closed form, so providing analytical bounds of the form in (2.16) is typically not feasible. Instead, we could choose to quantify the error as the *expected* error when drawing the initial condition u_0 from some probability distribution \mathcal{U} . In practice, we would then estimate the expected error of the distribution \mathcal{U} with a large, but finite sample \mathbb{U} drawn from \mathcal{U} . This is the approach which we will take in this work. As will be discussed in the following section, it fits naturally with the machine learning framework that we propose.

3. Learned splitting schemes. We now explain how we use Machine Learning (ML) to construct efficient splitting schemes which are guaranteed to have desirable properties such as consistency and symmetry. To do this, we first review some key concepts in ML in the context of the problem we consider in this paper. After this, we discuss how to enforce consistency and time reversal symmetry through a parameter transformation and outline our training algorithm.

3.1. ML notation and concepts. In the setting considered here, a general supervised machine learning approach requires three ingredients:

- A dataset \mathbb{U} , which consists of initial conditions for the IVP in (1.1). For each $u_0 \in \mathbb{U}$, we assume that we can approximate the solution $u(T) = \psi_T^{[f]}(u_0)$ of the IVP at time T to high precision, i.e. $u_{\text{Ref}} \approx u(T)$, and we therefore construct labelled pairs (u_0, u_{Ref}) .
- A numerical flow function $\Phi(\cdot; \theta)$ which encodes a numerical method and is parametrised by learnable parameters $\theta \in \mathbb{R}^d$ for some given $d \in \mathbb{N}$.

- A loss function $\mathcal{L}(\theta)$ which measures how well $\Phi(\cdot; \theta)$ approximates the analytic flow $\psi_T^{[f]}$ for initial conditions $u_0 \in \mathbb{U}$.

During training, the parameters θ are tuned such that $\Phi(\cdot; \theta)$ approximates $\psi_T^{[f]}$. Ideally, we would like to choose \mathbb{U} to be the set of *all* possible physical initial conditions. However, this is typically impractical and in the case of PDEs, where this set is parametrised by an infinite number of degrees of freedom, impossible. Instead, we assume here that the initial conditions $u_0 \sim \mathcal{U}$ are drawn from a probability distribution which we denote by \mathcal{U} . We define the loss function $\mathcal{L}(\theta)$ as the expectation of the mean squared error over the distribution \mathcal{U} which is given by,

$$(3.1) \quad \mathcal{L}(\theta) = \mathbb{E}_{u_0 \sim \mathcal{U}} \|\Phi(u_0; \theta) - u_{\text{Ref}}\|_2^2.$$

In practice, only a finite number of samples is considered so that the expectation in the loss function (3.1) can be replaced by a sum. The optimal parameters θ^* are then obtained by minimising the loss, i.e.

$$(3.2) \quad \theta^* = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta).$$

Our aim is therefore to determine θ^* results in the learned function $\Phi(\cdot; \theta^*)$ whose evaluation is efficient in the sense of **C2**, and which generalises to unseen inputs u_0 . This includes initial conditions with low probability or even u_0 which are not in the distribution \mathcal{U} . Usually this is possible if the function $\Phi(\cdot; \theta)$ is both sufficiently expressive and enough samples $u_0 \sim \mathcal{U}$ are used for training to find the optimal θ^* while avoiding overfitting. Overfitting can be mitigated by adding regularisers or by including inductive biases in the construction of the function $\Phi(\cdot; \theta)$.

To determine θ^* , brute force grid search or parameter sweep methods can be used to find an approximate minimiser of the loss function (3.1) provided the parameters are contained in a bounded subset of \mathbb{R}^d for some small d . For large d , these approaches are too expensive since the cost grows exponentially in d . For high-dimensional search spaces, θ^* is usually determined using gradient-based methods which only require differentiability of \mathcal{L} with respect to θ . If $u_0 \sim \mathcal{U}$ is randomly drawn for each gradient computation we call this stochastic optimisation (SO). Examples of SO include stochastic gradient descent (SGD) or improved variants like Adam [25].

3.2. Incorporating flows into ML. We let our learnable numerical flows $\Phi(\cdot; \theta)$ be the splitting methods introduced in Section 2. Hence $\Phi(\cdot; \theta)$ is obtained by composing multiple applications of the forward map $\Psi_h^{[f]}(\cdot; \alpha, \beta)$ and the learnable parameters θ are the coefficients α, β that define the splitting scheme. In symbols, we define the learnable flow as,

$$(3.3) \quad \Phi(u_0; \theta) = \Psi_{T,h}^{[f]}(u_0; \alpha, \beta) := \left(\Psi_h^{[f]}(u_0; \alpha, \beta) \right)^N.$$

By restricting the values of the learnable parameters α, β as in (2.9) we can guarantee the consistency order condition. Provided the IVP is well-defined and the method is stable, this will imply that the learned flow will produce provably convergent solutions in the limit $h \rightarrow 0$. Higher-order consistency can be enforced by imposing further order conditions, such as symmetry under time reversal. Other constraints from classical numerical analysis such as unitarity or symplecticity can also be enforced in this ML

framework by enforcing the very same classical conditions from pre-existing numerical analysis. With the learned flow $\Phi(u_0; \theta)$ in (3.3) the loss function in (3.1) becomes,

$$(3.4) \quad \mathcal{L}(\alpha, \beta) = \mathbb{E}_{u_0 \sim \mathcal{U}} \left\| \Psi_{T,h}^{[f]}(u_0; \alpha, \beta) - u_{\text{Ref}} \right\|_2^2.$$

By construction, the loss function in (3.4) is differentiable in α and β , as the numerical flow was differentiable in α and β . As a consequence, it can be minimised with gradient-based methods as discussed in Section 3.1. In analogy to (3.2), we obtain the optimal parameters α^*, β^* as

$$(3.5) \quad \alpha^*, \beta^* = \underset{\alpha, \beta}{\operatorname{argmin}} \mathcal{L}(\alpha, \beta) \text{ for } \mathcal{L} \text{ defined in (3.4).}$$

The minima of the loss function (3.5) depends on the distribution \mathcal{U} . In other words, the learned splittings are tailored to the specific distribution of initial values, and hence a specific distribution of IVPs. Minimizing the loss function (3.4) minimises the expected error. For an order p splitting the error is of the form

$$(3.6) \quad \left\| \Psi_{T,h}^{[f]}(u_0; \alpha, \beta) - u_{\text{Ref}} \right\| = C_0(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])h^p \\ + \sum_{m=1}^{\infty} C_m(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])h^{p+m}$$

Observe that for small timestep sizes h the dominant contribution to the error will come from the first term in (3.6). Minimizing the loss function (3.4) will minimise the expected value of all error constants $C_m(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])$ and in particular the expected value of the leading constant $C_0(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])$. As a consequence, the error of the learned method is potentially significantly smaller than that of a method constructed with traditional techniques from numerical analysis, which rely purely on algebraic and analytic techniques, in particular for larger values of h where asymptotic analysis typically breaks down. In other words, the learned method is efficient in the sense of criterion C2.

3.3. Enforcing consistency and symmetry through parameter transformations. Training on the landscape defined by the loss function (3.4) for a splitting method $\Psi_{T,h}^{[f]}(u_0; \alpha, \beta)$ is non-trivial since it is non-convex and ill-conditioned, in particular for small values of h . To see this, consider the expansion of the error in (3.6). For sufficiently long splitting methods the manifold of all parameters α, β will likely contain a sub-manifold of parameters for which the method is of order $p + 1$, i.e. the constant $C_0(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])$ will be identically zero on this sub-manifold. Hence, varying the parameters within this sub-manifold will only lead to small changes in the loss function due to changes in the error constants $C_m(f^{[1]}, f^{[2]}, u_0, T; [\alpha, \beta])$ with $m \geq 1$. However, moving away from the sub-manifold will at least reduce the order of the method from $p + 1$ to p , and thus incur changes of $\mathcal{O}(h^{-1})$ in the loss function. This argument can be applied recursively: sub-manifolds of higher-order methods are nested within manifolds that describe lower-order methods. As a consequence, the condition number of the Hessian grows with some power of the inverse timestep size h^{-1} . Furthermore, this problem is likely to get worse for splitting methods with more sub-flows K , i.e. higher-dimensional parameter spaces. To compensate for the poor conditioning, we would have to severely restrict the learning rate which renders SO methods very inefficient.

Given that the consistency order condition (2.9) is easy to enforce and required for provable convergence guarantees of the learned method, it would be unwise not to incorporate it. The consistency order condition (2.9) restricts the parameters α, β to an affine hyperplane. Symmetric splitting methods can be obtained by imposing an additional set of linear constraints on α, β , which – together with (2.9) – reduce the parameters to a lower-dimensional hyperplane that contains methods of even order.

To restrict ourselves to the sub-manifold that enforces consistency and symmetry, we express the original $\alpha, \beta \in \mathbb{R}^K$ in terms of suitable coordinates $\gamma_\alpha \in \mathbb{R}^{\lfloor \frac{K-1}{2} \rfloor}$, $\gamma_\beta \in \mathbb{R}^{\lfloor \frac{K-2}{2} \rfloor}$ on the sub-manifold and combine reduced parameters as $\gamma := [\gamma_\alpha, \gamma_\beta] \in \mathbb{R}^{K-2}$. For K even, both sub-parameterisations $\gamma_\alpha, \gamma_\beta$ are of equal size, i.e. $|\gamma_\alpha| = |\gamma_\beta| = \frac{K-2}{2}$. However, if K is odd, the sub-parameterisations cannot be of equal size. The symbolic zero $\beta_K = 0$ suggests that γ_β has fewer degrees of freedom and hence γ_β has a smaller sub-parameterisation than γ_α , i.e. $|\gamma_\alpha| = \frac{K-1}{2}$ and $|\gamma_\beta| = \frac{K-3}{2}$. The corresponding linear parameterisation transform $g: \mathbb{R}^{K-2} \rightarrow \mathbb{R}^{2K}$ which will be used in the following can be written in matrix form as,

$$(3.7) \quad [\alpha, \beta]^\top = g([\gamma_\alpha, \gamma_\beta]) = \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix} \begin{bmatrix} \gamma_\alpha \\ \gamma_\beta \end{bmatrix} + \begin{bmatrix} C \\ D \end{bmatrix} \Leftrightarrow A\gamma_\alpha + C = \alpha, B\gamma_\beta + D = \beta.$$

Explicit expressions for the matrices $A \in \mathbb{R}^{K \times \lfloor \frac{K-1}{2} \rfloor}$, $B \in \mathbb{R}^{K \times \lfloor \frac{K-2}{2} \rfloor}$ and the vectors $C, D \in \mathbb{R}^K$ can be found in Appendix B. The loss function in (3.4) and optimisation problem in (3.5) becomes a function of the parameters $\gamma \in \mathbb{R}^{K-2}$, namely

$$(3.8) \quad \mathcal{L}(\gamma) = \mathbb{E}_{u_0 \sim \mathcal{U}} \left\| \Psi_{T,h}^{[f]}(u_0; g(\gamma)) - u_{\text{Ref}} \right\|_2^2 \quad \text{and} \quad \gamma^* = \underset{\gamma}{\operatorname{argmin}} \mathcal{L}(\gamma).$$

In summary, restricting learning to a sub-manifold (1) reduces the dimension of the search space, (2) improves conditioning and (3) guarantees that the learned splitting method is provably consistent.

3.4. Training algorithm. Having set out the machine learning problem, we now discuss our training algorithm for minimising the loss function (3.8). It is natural to expect that the coefficients of known classical splittings, such as Trotter in (2.6) and Strang in (2.7), are local minima of the loss function. For higher-dimensional coefficient spaces, local minima might also correspond to compositions of these methods, as discussed in Section 2.4. Our goal is to beat preexisting numerical methods for our choice of initial conditions and step sizes, thus care must be taken to avoid poor local minima that may correspond to pre-existing methods.

Although they are usually the standard choice in machine learning applications, pure gradient based methods such as Adam [25] risk getting stuck in such local minima which are more prevalent in low- to moderate-dimensional parameter spaces that we consider here. For this reason, we opt to incorporate a global optimisation aspect which attempts to cover a significant fraction of the parameter space with the aim to identify the minima with very low loss values. As described in Algorithm 3.1, the key idea is to create a set of candidate parameter values that are distributed widely over the search space, screen these candidates with a simple heuristic to reduce them to a manageable number and then run a SO algorithm with the candidates as starting points to fine-tune the parameter values. We used Adam in all numerical experiments (see Appendix D.1 for a discussion of other first- and second-order stochastic optimisers).

Let $\mathcal{S}_{\text{train}} = \{(u_0^{(j)}, u_{\text{Ref}}^{(j)})\}_{j=1}^{M_{\text{train}}}$ with $u_0^{(j)} \sim \mathcal{U}$ be the training dataset with M_{train} data points. A validation dataset $\mathcal{S}_{\text{valid}}$ with M_{valid} samples is constructed in the

same way. For each subset $\mathcal{B} \subseteq \mathcal{S}_{\text{train}}$ (or equivalently $\mathcal{B} \subseteq \mathcal{S}_{\text{valid}}$), we define the loss function

$$(3.9) \quad \mathcal{L}(\gamma; \mathcal{B}) = \frac{1}{|\mathcal{B}|} \sum_{(u_0, u_{\text{Ref}}) \in \mathcal{B}} \left\| \Psi_{T,h}^{[f]}(u_0; g(\gamma)) - u_{\text{Ref}} \right\|_2^2$$

which approximates the “true” loss in (3.8) with a finite sample of size $|\mathcal{B}|$.

Algorithm 3.1 Learning Pipeline for (approximately) minimising $\mathcal{L}(\gamma)$

- 1: Generate candidate coefficients $\Gamma = \{\gamma_1, \gamma_2, \dots\}$, by either choosing $\gamma_i \in \mathbb{R}^{K-2}$ randomly or as the vertices of a regular grid that covers the bounded domain $\Omega \subset \mathbb{R}^{K-2}$ in which we expect to find the global minimum.
 - 2: **for** all candidates $\gamma_i \in \Gamma$ **do**
 - 3: Compute loss $\ell_i := \mathcal{L}(\gamma_i; \mathcal{S}_{\text{valid}})$.
 - 4: **end for**
 - 5: Remove all candidates γ_j from Γ for which $\ell_j > \min_i \{\ell_i\} + \epsilon$ for some $\epsilon > 0$.
 - 6: Remove all candidates γ_j from Γ for which $\exists i$ s.t. $\|\gamma_j - \gamma_i\| \leq \delta$ and $\ell_j > \ell_i$ for some minimum distance $\delta > 0$.
 - 7: **for** all candidates $\gamma_i \in \Gamma$ **do**
 - 8: Improve γ_i by applying a fixed number of batched SO steps; for this use different randomly chosen batches $\mathcal{B} \subset \mathcal{S}_{\text{train}}$ at each SO step.
 - 9: **end for**
 - 10: **return** $\gamma_{\min} = \operatorname{argmin}_{\gamma_i \in \Gamma} \{\mathcal{L}(\gamma_i; \mathcal{S}_{\text{valid}})\}$.
-

Observe that the variance of the estimator in (3.9) decreases $\propto |\mathcal{B}|^{-1}$. As a consequence, for small $|\mathcal{B}|$ the value of $\mathcal{L}(\gamma; \mathcal{B})$ for fixed γ can vary significantly between different batches \mathcal{B} . This is not surprising since the learned splitting method will perform differently for different initial conditions. Hence, when making comparative judgments between different parameters, such as in lines 3 and 10 of Algorithm 3.1, the loss function $\mathcal{L}(\gamma_i; \cdot)$ should be evaluated with a single fixed batch, chosen to be $\mathcal{S}_{\text{valid}}$, for all parameters γ_i .

Each minimum of the loss function will have some basin of attraction under SO. Because of this, it would be computationally inefficient to fine-tune all candidates identified in the first step of Algorithm 3.1 with expensive SO iterations. We therefore only pursue candidates that have a low loss (compared to all other candidates) and which are separated by some distance $\delta > 0$. This distance is chosen heuristically to be smaller than the average diameter of all basins of attraction. While alternative techniques such as particle swarm methods [30, 22] have been suggested in the literature, we find that empirically Algorithm 3.1 produces good results.

Because second-order convergence, in the limit $h \rightarrow 0$, of our learned methods is guaranteed by their construction in Section 3.3, the learned method will generalise to other initial conditions outside the training distribution \mathcal{U} . This is a distinctive advantage compared to naive ML approaches, which cannot be expected to generalise in this sense.

4. Numerical results. In this section we numerically show the efficiency of learned splitting methods for a representative model problem.

4.1. Schrödinger’s equation. To demonstrate the advantages of our approach, we consider the one-dimensional Schrödinger equation $i\dot{u}(x, t) = [V(x) - \Delta]u(x, t)$

where $\Delta = \partial^2/\partial x^2$ is the Laplace operator and $V(x)$ is a real-valued potential which we assume to be monotonically increasing for $|x| > L$. While typically the problem is defined on $\mathbb{R} \times [0, T]$, discretisation of the unbounded spatial domain with an equidistant grid would require an infinite number of unknowns and make the solution untractable on a computer with finite memory. Because of this, we restrict the domain to $\Omega \times [0, T]$, where $\Omega = [-L, +L]$ for some $L > 0$, and impose periodic boundary conditions in space; these boundary conditions render the Laplace operator diagonal in Fourier space [36]. The solution on $\Omega \times [0, T]$ differs from the one on $\mathbb{R} \times [0, T]$ by terms that are exponentially suppressed. These differences are small provided L is sufficiently large (to see this, note that the eigenfunction corresponding to energy E can be bounded by $C \exp[-\sqrt{V(L) - E}]$ for $|x| > L$ and some constant C).

The Schrödinger equation can be seen as an example of a wider class of one-dimensional autonomous PDEs of the form

$$(4.1) \quad \dot{u}(x, t) = \mathcal{F}(u(x, t)), \quad u(x, 0) = u_0(x), \quad u(x, t) \in \mathbb{C}, \quad x \in \Omega \subset \mathbb{R}, \quad t \in [0, T]$$

with suitable boundary conditions on $\partial\Omega$. Here, \mathcal{F} (which is $i[\Delta - V(x)]$ for the Schrödinger equation), is the differential operator acting on the solution $u(x, t)$. The method of lines is used to convert the problem (4.1) into a system of coupled ODEs of the form (1.1) which can be solved numerically. For this, we pick a discretisation of the spatial domain Ω defined by a set of M equally spaced points $\{x_m\}_{m=1}^M$, $x_m = (\frac{2m-1}{M} - 1)L \in \Omega$, discretise the differential operator \mathcal{F} and solve for the time-dependent solution vector $u(t) = (u_1(t), \dots, u_M(t)) \in \mathbb{C}^M$ with $u_m(t) \approx u(x_m, t)$ for $m = 1, \dots, M$. For the Schrödinger equation, this allows us to write the problem in the form of the IVP in (1.1) as

$$(4.2) \quad \dot{u}(t) = i[\widehat{\Delta} - \widehat{V}]u(t), \quad u(0) = u_0, \quad u(t) \in \mathbb{C}^M, \quad t \in [0, T]$$

for some initial condition $u_0 \in \mathbb{C}^M$. The Laplace operator Δ , and indeed the entire Hamiltonian $H = V(x) - \Delta$, is self-adjoint. The Hermitian matrix $\widehat{\Delta} \in \mathbb{C}^{M \times M}$ approximates Δ , and is diagonal in discrete Fourier space. Additionally, $\widehat{V} \in \mathbb{R}^{M \times M}$ is a diagonal matrix, where the diagonal entries are given by $\widehat{V}_{m,m} = V(x_m)$. The natural splitting for (4.2) is to use $f^{[1]} = -i\widehat{V}$ and $f^{[2]} = i\widehat{\Delta}$ as this lets us exploit the diagonalisability of both these flows. We let $\widehat{\Delta} = U^\dagger \widehat{\Delta}_{\text{diag}} U$ where $\widehat{\Delta}_{\text{diag}} \in \mathbb{C}^{M \times M}$ denotes a diagonal matrix and $U \in \mathbb{C}^{M \times M}$ is the unitary Fourier transform matrix. Note that applying U and U^\dagger with the Fast Fourier Transformation [13] costs $\mathcal{O}(M \log(M))$ operations. As a consequence, the sub-flows corresponding to $f^{[1]}$ and $f^{[2]}$ are given by

$$(4.3) \quad \psi_t^{[1]} = e^{-it\widehat{V}} \quad \text{and} \quad \psi_t^{[2]} = e^{it\widehat{\Delta}} = U^\dagger e^{it\widehat{\Delta}_{\text{diag}}} U,$$

respectively. The sub-flows can be evaluated efficiently since $e^{-it\widehat{V}}$ and $e^{it\widehat{\Delta}_{\text{diag}}}$ are exponentials of diagonal matrices which can be computed in $\mathcal{O}(M)$ time. Hence, the total cost for one evaluation of $\psi_t^{[1]}$ and $\psi_t^{[2]}$ is $\mathcal{O}(M)$ and $\mathcal{O}(M \log M)$, respectively. In our numerical experiments we fix $M = 200$.

4.2. Implementation. Our Python code is publicly available on [Zenodo](#) and datasets and scripts to recreate all figures are available on [Github](#). All training and inference algorithms were implemented in the JAX library [10], which we found to be significantly faster than PyTorch since it allows the very efficient automatic differentiation with respect to the parameters α, β in products of exponentials that arise

from (2.8). Double precision arithmetic was used for all calculations. We used the Optax package [8] for the implementation of Stochastic Optimisation in line 8 of Algorithm 3.1. The Fast Fourier Transform that is required in the evaluation of the flow $\psi_t^{[2]} = e^{it\hat{\Delta}}$ in (4.3) employs the efficient `jax.numpy.fft` method to mimic the implementation in the Expsolve package [36]. During training and validation the analytical flow $\psi_T^{[f]} = e^{iT[\hat{\Delta}-\hat{V}]}$ needs to be computed. For this, the matrix exponential is evaluated by exponentiating the (dense) matrix $\hat{\Delta} - \hat{V}$ with the `scipy.linalg.expm` function, which employs the scaling and squaring algorithm given in [2]. Since this is only required during training and will not have any impact on the performance of the learned timestepping methods, no attempt was made to optimise this operation.

4.3. Learning splitting coefficients for Schrödinger’s equation. During training we learn optimal splitting coefficients for the Schrödinger equation given in (4.2) where $V(x) = x^4 - 10x^2$ is a quartic function that describes a double-well potential. The minima of the potential are located at $x_{\pm} = \pm\sqrt{5}$ in this case. The parameter transform described in Section 3.3 is used to enforce consistency and symmetry of the learned splitting methods.

4.3.1. Data generation. The training dataset $\mathcal{S}_{\text{train}} = \{(u_0^{(j)}, u_{\text{Ref}}^{(j)})\}_{j=1}^{M_{\text{train}}}$ consists of initial conditions and reference solutions, where $u_0^{(j)}$ and $u_{\text{Ref}}^{(j)}$ are functions evaluated on the spatial grid $\{x_m\}_{m=1}^M$ introduced in Section 4.1. The reference solutions are calculated using the matrix exponential of the initial conditions. While we have not specified the distribution \mathcal{U} of initial conditions explicitly, this distribution is defined implicitly by a sampling algorithm that is controlled by three parameters: the centre mean x_{cent} , the centre standard deviation x_{stdDev} , and the basis standard deviation σ . We let $x_{\text{cent}} = -\sqrt{5}$, $x_{\text{stdDev}} = 0.1$, and $\sigma = 0.5$ for training.

We recursively generate $u_0^{(j)}$ by starting from the previous reference solution $u_{\text{Ref}}^{(j-1)}$. With a moderate probability we perturb $u_{\text{Ref}}^{(j-1)}$ by adding the discrete Gaussian $\mathcal{N}(x_0^{(j)}, \sigma)$ then renormalising and/or a random phase shift, or with a low probability we set $u_0^{(j)}$ to be the discrete Gaussian $\mathcal{N}(x_0^{(j)}, \sigma)$, where $x_0^{(j)} \sim \mathcal{N}(x_{\text{cent}}, x_{\text{stdDev}})$. As $x_{\text{cent}} = x_-$, these discrete Gaussians are clustered around the centre of the left well of the potential V . For $j = 0$, we proceed as above but start with the discrete Gaussian $\mathcal{N}(\bar{x}_0, \sigma)$ for some $\bar{x}_0 \sim \mathcal{N}(x_{\text{cent}}, x_{\text{stdDev}})$ instead of the previous reference solution. The recursive augmentation method is written down explicitly in Algorithm C.1, along with some representative initial conditions $u_0^{(j)}$ in Figure 11, in Appendix C.

A fixed validation dataset $\mathcal{S}_{\text{valid}}$ is constructed in the same way. During training, we fix $T = 10$ and $h = \frac{1}{7}$ (which implies $N = 70$), and use the approximate loss function in (3.9). We find that training with a fixed step size results in learned splitting methods which generalise to other timestep sizes h and final times T (see Section 4.3.4).

4.3.2. Training procedure. The landscape defined by the loss function (3.9) for splittings of length $K = 5$ and evaluated on a fixed validation set $\mathcal{B} = \mathcal{S}_{\text{valid}}$ of size 200, has three free parameters γ and is visualised in Figure 2. As this figure shows, the landscape is clearly non-convex and has multiple minima, as expected. Observe that locations of lower values of the loss function appear to be close to lower-dimensional sub-manifolds that define higher order methods, which is consistent with the discussion in Section 3.3. The minimum at $\gamma_{\text{Strang}} = [0.125, 0.25, 0.25]$ is readily identified as the composition of four Strang splittings (resulting in a five-stage method) with a validation loss value of $\mathcal{L}(\gamma_{\text{Strang}}) = 0.2917$. However, by using the

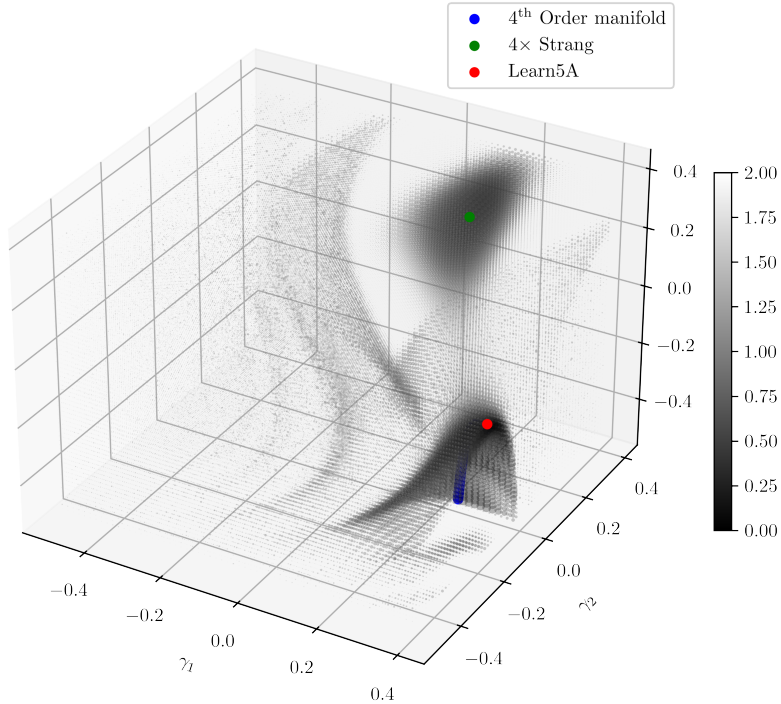


FIG. 2. Plot of the loss function $\mathcal{L}(\gamma)$ in (3.9) for the Schrödinger equation in (4.2) with $T = 10$ and $h = \frac{1}{7}$. A section of the one-dimensional manifold of fourth-order accurate methods can be seen in the lower right corner. To obtain this figure, the loss function was evaluated on the fixed validation set $\mathcal{B} = \mathcal{S}_{\text{valid}}$ with 200 members and for a uniform grid for the values of γ in the box $[-0.5, 0.4]^3$. Lower loss values are plotted as darker and larger points.

full training algorithm in Algorithm 3.1, we also identified a novel learned splitting, referred to as Learn5A, with $\gamma_{\text{learned}} = [0.3627, -0.1003, -0.1353]$ and a substantially lower validation loss value of $\mathcal{L}(\gamma_{\text{learned}}) = 0.02106$.

For the training dataset, the landscape will vary between batches, but we expect the qualitative behaviour shown in Figure 2 to be representative for most batches. Due to this stochasticity γ_{learned} will not be a minimum of the loss function when evaluated on $\mathcal{S}_{\text{valid}}$. Instead we consider the nearby validation loss minimum found at $\gamma_{\text{valid}} = [0.3314, -0.07304, -0.1821]$. To gain further insight into the loss function $\mathcal{L}(\gamma)$, we plot the validation loss in the proximity of the two minima γ_{Strang} and γ_{valid} in Figure 3. The shape of the minima is characterised by their Hessian matrices. The condition numbers of the Hessians at γ_{Strang} and γ_{valid} are 15.5 and 8563.1, respectively. The high condition number of the Hessian matrix at γ_{valid} is consistent with the discussion in Section 3.3. This large condition number implies that the naive SGD will be very inefficient, we explored a variety of improved stochastic optimisers (SOs), including both first- and second-order methods, in our training pipeline (line 8 of Algorithm 3.1); further details can be found in Appendix D.1. However, all results reported in the rest of this section were obtained with the widely used Adam method [25] which we find to be efficient and robust. The minimum $\gamma_{\text{learned}} = [0.3627, -0.1003, -0.1353]$ was found with a fixed learning rate of 0.01, where the parameters are initialised with the candidates identified in the initial exploration of the parameter space (lines 1-6

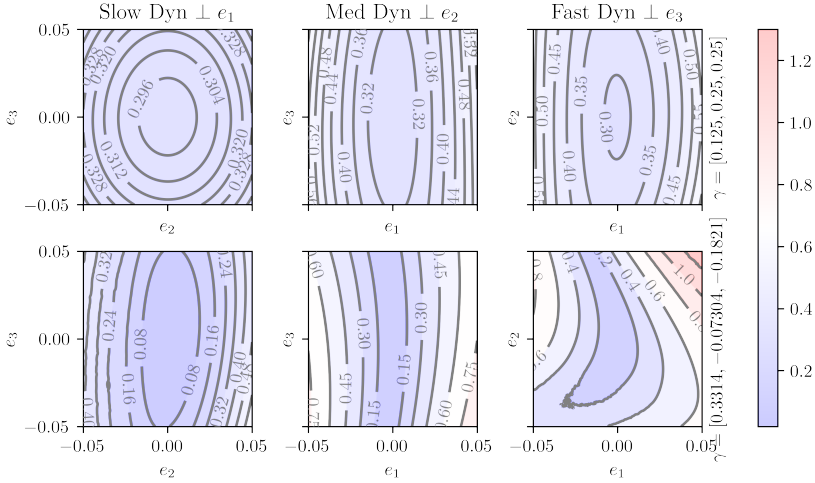


FIG. 3. Local environment of the validation loss visualised in Figure 2 around the two minima γ_{Strang} and γ_{valid} . The function is plotted in the three planes that are perpendicular to the largest (left), middle (centre), and smallest (right), eigenvalues of the Hessian matrix at the local minima.

of Algorithm 3.1). The evolution of γ_{learned} during training is shown in Figure 12 in Appendix D.

4.3.3. Learning longer splitting methods. We also employed the training pipeline described in Section 3.4 to find longer learned splittings with $K = 8$ stages and six free parameters. For this, we generated 75,000 random parameter candidates, where we implicitly ensured that some of the parameter values are negative and hence potentially near methods higher than second order. We then evaluated the loss on a consistent validation set (for fairness of comparison), selected the 100 parameter candidates with the lowest losses and removed all candidates that were within an (Euclidean) distance of less than 0.75 to parameter candidates with lower loss values. This resulted in nine candidates that were explored further. For these, the Adam optimisation step in line 8 of Algorithm 3.1 was performed for 250 iterations with a learning rate schedule starting at 0.02 and decreasing exponentially with a decay rate of 0.995. As before, the parameter transform $g(\gamma)$ in (3.7) was used to ensure that the learned methods are consistent and symmetric. Two of the nine parameter candidates converged to unknown splittings with low losses, which we refer to as Learn8A and Learn8B. Along with Learn5A found in Section 4.3.2, the splitting coefficients γ after training are given in Table 1 and their evolution during training is shown in Figure 12 in Appendix D.

Naturally, the question arises whether the learned methods are similar to known, existing methods. Since this is difficult to infer from the numerical value of γ , we describe a graphical technique for visualising different splitting methods. For this, a method defined by the coefficients $(\alpha_1, \dots, \alpha_K, \beta_1, \dots, \beta_K)$ is represented by the continuous curve obtained by joining line segments of (oriented) lengths $\alpha_1, \beta_1, \alpha_2, \beta_2, \dots, \alpha_K, \beta_K$ in this order such that the segments associated with α_j are parallel to the horizontal axis while the segments corresponding to β_j are parallel to the vertical axis. Figure 4 uses this technique to visualise splitting methods discussed in this work. As explained in Appendix E, this construction can be interpreted as the solution of an

Splitting	K	γ
Trotter	1	—
Strang	2	\square
Yoshida	4	[0.67560, 1.35120]
4× Strang	5	[0.125, 0.25, 0.25]
Learn5A	5	[0.3627, -0.1003, -0.1353]
Learn8A	8	[0.2135, -0.0582, 0.4125, -0.1352, 0.4443, -0.0251]
Learn8B	8	[0.1178, 0.3876, 0.3660, 0.2922, 0.0564, -0.0212]

TABLE 1

Coefficients γ of existing and learned splitting methods with different numbers of stages K . The non-symmetric Trotter given in (2.6) can not be symmetrically parameterised. The Strang method, given in (2.7), is completely defined by symmetry and consistency.

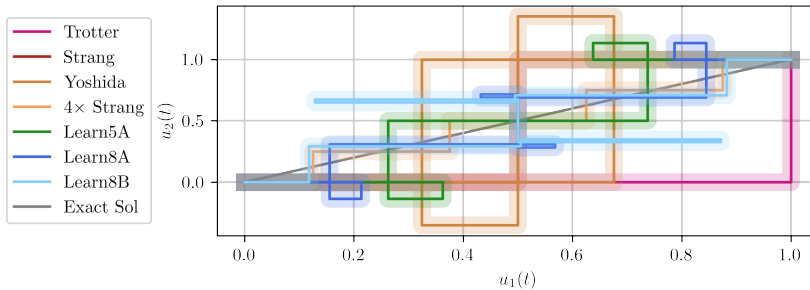


FIG. 4. Comparison of the paths that visualise the fluxes (E.1) for the IVP $\dot{u}(t) = [1, 1]^T$, $u(0) = [0, 0]^T$, $t \in [0, 1]$. Results are shown for known numerical methods and our new learned splitting methods.

initial value problem. As none of the learned paths are covered by another path in their entirety in Figure 4, we conclude that the learned methods are novel since they differ from existing methods. We expect that the associated splitting methods also behave fundamentally different for more complicated IVPs, in particular the Schrödinger equation considered here.

Next, we assess the performance and efficiency of our learned methods by comparing them to three well-known splittings, namely Trotter, Strang and Yoshida. As can be seen from Table 1 and Figure 4, the different methods involve different numbers of stages K . This makes comparisons harder, as comparing Strang with 4× Strang is not fair unless their different cost is taken into account. To address this, we plot the error against the number of exponential evaluations, i.e. the number of sub-flow evaluations. Since the number of sub-flow evaluations can also be regarded as a proxy for the computational cost, this allows a fair comparison of the methods with different numbers of stages as we can no longer “improve” a method by repeating it many times in a single step of a longer method. Figure 5 shows the expected L_2 -error at the final times $T = 10$ over the validation set as a function of the number of exponential evaluations (which is proportional to the inverse of the timestep size h) for various methods. As expected we observe from Figure 5 that the learned splittings Learn5A, Learn8A and Learn8B converge quadratically for $h \rightarrow 0$. Interestingly, they show a faster initial decay for larger h and this is discussed in more detail in Section 4.3.5.

While the fourth-order Yoshida integrator performs better at high resolutions,

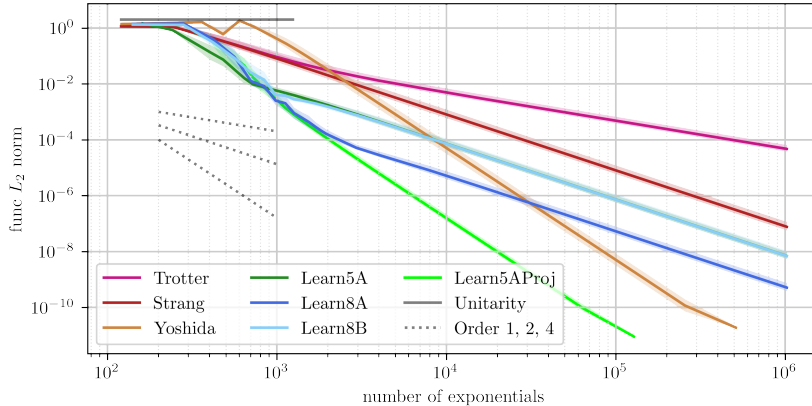


FIG. 5. Average L_2 -error $\|\Psi_{10,h}^{[f]}(u_0) - \psi_{10}^{[f]}(u_0)\|_2$ at the final time $T = 10$ on a validation set of size 200. The median is shown with a line, with a shaded envelope indicating the 15.9% and 84.1% quantile. The solid horizontal black vertical line corresponds to the unitarity bound which arises from the fact that the L_2 -norm of the numerical flow cannot exceed 1. Learn5AProj is introduced in Section 4.3.5.

Figure 5 demonstrates that our learned methods outperform all other classical methods for smaller computational budgets: one can either achieve a significant speed-up by using larger step sizes or improve accuracy by orders of magnitude with the same step size. We illustrate this in Figure 6, where we plot the relative advantage (in terms of accuracy) in comparison to Yoshida as a function of the number of exponential evaluations. The figure confirms that our learned splitting methods can be up to two orders of magnitude more accurate than Yoshida at the same cost. This is further quantified in Table 2 where we estimated the loss values after 2506 subflow evaluations; our learned methods have the highest relative advantage for this timestep size. The final two columns of the table show the relative accuracy of the methods as well as the gain in speed. The latter is defined as the relative decrease in the number of subflow evaluations compared to Yoshida when both methods reach the same L_2 error. We conclude that we have indeed learned splitting methods that outperform classical numerical methods such as Trotter, Strang, and Yoshida for small computational budgets.

Splitting	L_2 -error for 2506 subflows	Rel accuracy vs Yoshida	Rel speed vs Yoshida
Trotter	0.023247	0.55	0.84
Strang	0.012862	1.00	1.00
Yoshida	0.012864	1.00	1.00
Learn5A	0.001121	11.47	1.84
Learn8A	0.000081	158.75	3.55
Learn8B	0.001029	12.50	1.88

TABLE 2

Comparison of loss values after 2506 subflow evaluations, corresponding relative accuracy and gain in speed for various splitting methods.

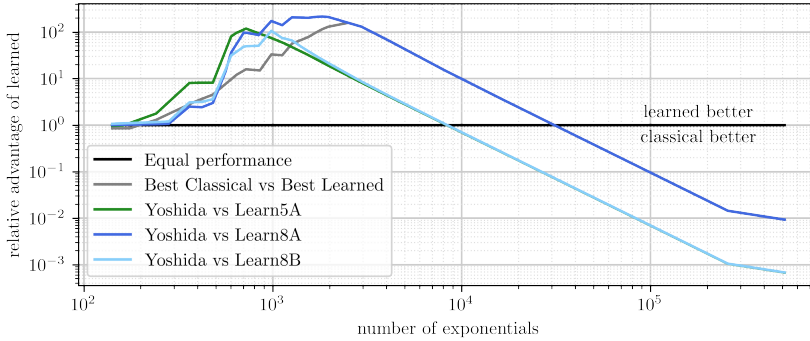


FIG. 6. Comparison of the classical vs learned methods relative expected L_2 -errors, as plotted in Figure 5. The plot shows the L_2 -error of a classical method divided by the L_2 -error of a learned method. Hence, values larger than one represent the learned method being superior. We define the best classical and learned method as the element-wise minimum of the respective families.

4.3.4. Generalisation to other setups. Due to the cost associated with training, learning an optimised splitting method is most justified if it generalises to other setups. Sections 4.3.2 and 4.3.3 demonstrate that our learned methods are convergent and extrapolate from the training set to the unseen validation set drawn from the same distribution, as expected for any sensible ML approach. However, we note that all methods were trained for a fixed final time $T = 10$ and timestep size $h = \frac{1}{7}$, which corresponds to 561 sub-flow evaluations for Learn5A and 981 sub-flow evaluations for Learn8A and Learn8B. As shown in Figure 6, our learned methods maintain and indeed improve their advantages when generalised to other numbers of subflow evaluations. Next, we investigate the generalisation to other final times, different initial conditions and for perturbations of the double-well potential. More specifically, we consider the following setups:

- **Final time:** The final time T was set to either $T_1 = 10$ or $T_2 = 30$.
- **Initial conditions:** Recall that the (distribution of) initial conditions in Section 4.3.1 were controlled by three parameters x_{cent} , x_{stdDev} and σ and we denote the distribution of initial conditions by $\mathcal{U}(x_{\text{cent}}, x_{\text{stdDev}}, \sigma)$. Here we vary the (distribution of) initial conditions by either setting $\mathcal{U}_1 = \mathcal{U}(-\sqrt{5}, 0.1, 0.5)$, $\mathcal{U}_2 = \mathcal{U}(\sqrt{5}, 0.2, 0.5)$, or $\mathcal{U}_3 = \mathcal{U}(-\sqrt{15}, 0.05, \sqrt{0.1})$
- **Shape of the potential:** The shape of the double-well potential $V(x)$ is set to either $V_1(x) = x^4 - 10x^2$, $V_2(x) = x^4 - 10x^2 - 10x$, $V_3(x) = 3x^4 - 50x^2 + 20x$, or $V_4(x) = x^4 - 30x^2$; note that this includes non-symmetric potentials.

Figure 7 illustrates the L_2 -error for different final time, initial conditions and the shape of the potential. Both variations of individual parameters (left) and of all parameters (right) are shown. We omit to plot the L_2 -error for the case where only the initial conditions are changed, i.e. $T = T_1$, $\mathcal{U} = \mathcal{U}_2$, and $V = V_1$, as it is visually indistinguishable from the L_2 error on the original training data. We conclude that our learned methods do indeed generalise to IVPs with different final times, initial conditions and potentials and maintain their advantage compared to classical integrators for larger timestep sizes under these variations.

To demonstrate that we have indeed adapted to our distribution of IVPs (characterised by different distributions of initial conditions and potentials) and have not simply learned universal splittings (that we may have been able to find with traditional techniques) we illustrate that the loss landscape is significantly distribution

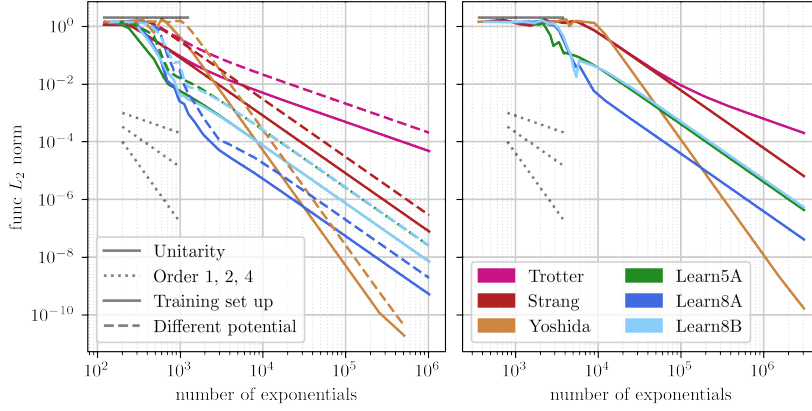


FIG. 7. L_2 error for generalisations to additional datasets with different final times, initial conditions and potentials. The same quantities as in Figure 5 are plotted. The solid line in the left figure shows the L_2 -error for the original training data, i.e. $T = T_1$, $\mathcal{U} = \mathcal{U}_1$, and $V = V_1$, for reference. The dashed line in the left figure shows L_2 -error for a changed potential, i.e. $T = T_1$, $\mathcal{U} = \mathcal{U}_1$, and $V = V_2$. In the right figure all parameters are changed simultaneously: the solid line shows the L_2 -error for different final time $T = T_2$, initial conditions $\mathcal{U} = \mathcal{U}_2$, and potential $V = V_3$.

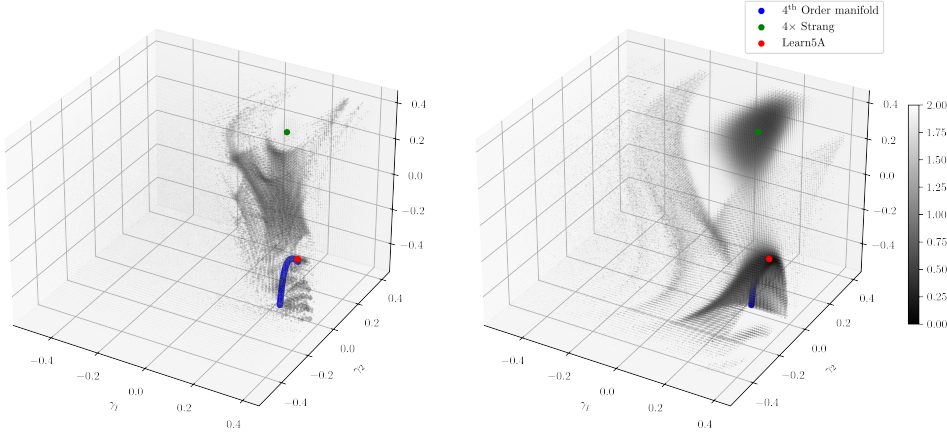


FIG. 8. Left error landscape generated with the distribution of IVPs defined by $T = T_1$, $\mathcal{U} = \mathcal{U}_3$, and $V = V_4$. By comparing with Figure 2, reproduced right, we clearly see that the landscape, and therefore gradients and minima are problem-dependent. This shows that our pipeline and the learned methods are indeed adapted to the training set of IVP distributions.

dependant. To see this we consider a distribution of initial conditions with sufficiently deep wells and sufficiently low energy initial conditions so that the training dataset illustrates only negligible quantum tunneling, i.e. $T = T_1$, $\mathcal{U} = \mathcal{U}_3$, and $V = V_4$. The associated landscape is shown in Figure 8 and significantly different from the landscape in Figure 2. In particular, we note that $4\times$ Strang has ceased to be a local minimum.

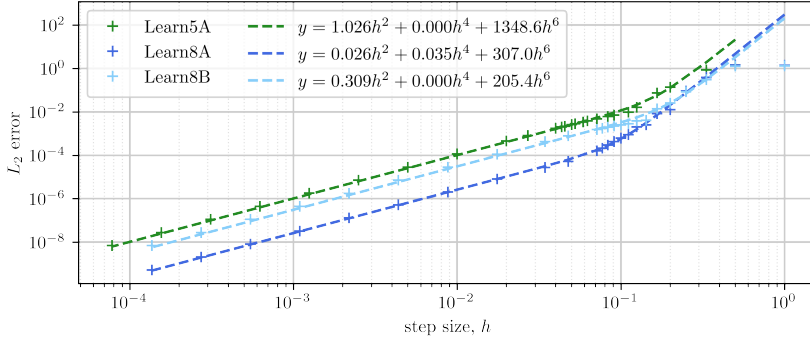


FIG. 9. Fit of the coefficients C_{2j} for $j = 1, 2, 3$ in the Taylor expansion (4.4) to the data points (x_i, y_i) as a function of the timestep size h .

4.3.5. Order of learned methods. Recall that for a given method, h is inversely proportional to the number of exponentials N required to integrate to a fixed final time T . With this in mind, Figure 5 shows that asymptotically the L_2 -error of the learned methods is quadratically convergent in the timestep size h . This is of course to be expected since the methods were explicitly constructed to be consistent and symmetric. Interestingly, before reaching this asymptotic convergence rate, there is a significant drop of the error for larger values of h . To quantify this, we consider a Taylor expansion of the L_2 -error. As the odd terms of the Taylor expansion are symbolically zero due to the parameter transform enforcing symmetry, we expand

$$(4.4) \quad E(h) = C_2 h^2 + C_4 h^4 + C_6 h^6 + \mathcal{O}(h^8),$$

for some non-negative constants $C_2, C_4, C_6 \geq 0$. If $0 < C_2 \ll C_4, C_6$, then the methods are formally second order ($C_2 \neq 0$), but the error will decrease approximately with a higher power of h for larger stepsizes where the term $C_2 h^2$ in (4.4) is small relative to the higher order terms. Figure 9 shows a fit of the leading terms in the expansion (4.4) to the L_2 -error, excluding the two data points with the largest timesteps where the sixth order polynomial approximation is not appropriate. This fit is based on the assumption that our data is of the form (x_i, y_i) and obeys the model $y_i = \sum_{j=1}^3 C_{2j} x_i^{2j}$ with coefficients C_{2j} . A least-squares fit leads to the minimisation problem $\operatorname{argmin}_{C \in \mathbb{R}^3} \mathcal{L}(C)$ where $C = (C_2, C_4, C_6) \in \mathbb{R}^3$ and $\mathcal{L}(C) = \sum_i (\sum_{j=1}^3 C_{2j} \phi_j(x_i) - y_i)^2$. To enforce non-negativity of the coefficients, we set $C_{2j} = (\tilde{C}_{2j})^2$ and fit the logarithm of the error, i.e. we find $\operatorname{argmin}_{\tilde{C} \in \mathbb{R}^3} \tilde{\mathcal{L}}(\tilde{C})$ with $\tilde{C} = (\tilde{C}_2, \tilde{C}_4, \tilde{C}_6)$ and $\tilde{\mathcal{L}}(\tilde{C}) = \sum_i (\log(\sum_{j=1}^3 (\tilde{C}_{2j})^2 \phi_j(x_i)) - \log(y_i))^2$. The results of this fit in Figure 9 show that visually the agreement with the data looks very reasonable for the Learn8A method, while it is less good for the other two methods. This is because the expansion in (4.4) does not consist of orthogonal basis functions and hence will be poor for larger values of h which constrain the terms C_4 and C_6 as higher order terms will have non-trivial effects. We stress, however, that our goal is not to find the exact values of the fit coefficients but rather to get an indication of the relative sizes of C_2, C_4 and C_6 . The numerical values of the fit coefficients C_2, C_4, C_6 given in Table 3 show that for all three learned methods we have $C_2 \ll C_6$. This confirms that although our learned methods are formally of order two, they are very close to methods of higher order.

method	C_2	C_4	C_6
Learn5A	1.026	0.000	1348.6
Learn8A	0.026	0.035	307.0
Learn8B	0.309	0.000	205.4

TABLE 3

Fit coefficients of the L_2 -error expansion in (4.4) as a function of the timestep size h for the three learned methods, based on the fit shown in Figure 9.

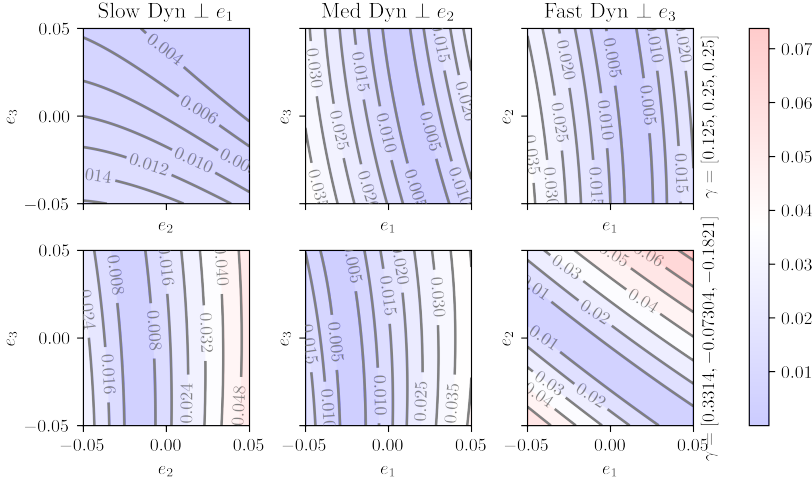


FIG. 10. Function $w(\gamma) := |w_{112}| + |w_{122}|$ which acts as a proxy of the violation of the fourth order condition as a function of the parameters γ . The function is plotted in the vicinity of the minima of the validation loss shown in Figure 3.

Given that empirically the learned methods appear to be “close to” fourth order, it is natural to look for higher order methods in the space parametrised by γ . The key technique for finding such higher order methods is to construct a smooth function of γ which vanishes on the manifold that describes higher order methods. As explained in [6], the manifold of methods of at least order four can be implicitly defined with the help of two polynomials w_{112}, w_{122} of the splitting coefficients (α, β) . By using the parameter transform $g(\gamma)$ described in Section 3.3, we express w_{112}, w_{122} as a function of the independent splitting coefficients γ . A symmetric method with splitting coefficients γ is of order four or higher if $w_{112}(\gamma) = 0$ and $w_{122}(\gamma) = 0$. For five-stage methods with three free parameters γ , we can explicitly parametrise the one-dimensional manifold defined by $w_{112}(\gamma) = w_{122}(\gamma) = 0$ which contains higher order methods. This manifold is depicted in Figure 2 and Figure 8. In Figure 10, we plot the value of $w(\gamma) := |w_{112}(\gamma)| + |w_{122}(\gamma)|$ as a function of the parameters γ in the vicinity of the minima that are considered in Figure 3. Observe that $w(\gamma) = 0$ is an equivalent definition of the higher-order manifold $w_{112}(\gamma) = w_{122}(\gamma) = 0$. As Figure 10 shows, the manifold of methods of at least order four is close to the minima γ_{valid} of the validation loss, visualised in Figure 3, that was associated with Learn5A. This confirms that our learned methods are indeed close to methods of at least order four.

Next, we further improve the learned method Learn5A by projecting the coeffi-

cient $\gamma_{\text{learned}} = [0.363, -0.100, -0.135]$ to the closest point on the manifold $w(\gamma) = 0$. We call this new method, defined by $\gamma_{\text{proj}} = [0.346, -0.112, -0.132]$, Learn5AProj. The L_2 -error of Learn5A and Learn5AProj is shown in Figure 5. As expected the Learn5AProj shows fourth-order convergence. Note, however, that our gradient based method was correct in moving away from the manifold of fourth-order methods as Learn5A does indeed have a smaller error than Learn5AProj for larger timesteps. Overall, this shows that our approach is able to find the most efficient method for a limited, fixed cost budget (i.e. number of exponential evaluations), that these efficient methods may be lower order and lower error constant methods, but also that gradient based methods can approximately recover the classically derived order conditions.

5. Conclusion and future work. In this paper, we have developed a machine learning-based framework to find computationally efficient splitting schemes tailored for limited computational budgets and adapted to classes of IVPs defined by specific right-hand sides, final times and distributions of initial conditions. Our approach maintains common advantages of classical methods, such as interpretability, generalisability, convergence, and conservation properties, which are usually not guaranteed for machine learning-based approaches. We showed that our framework can learn medium to long splittings with lower errors than classical splittings. It is particularly efficient for large timestep sizes. It should be stressed that our learned methods are novel which was confirmed by comparing them to existing splitting methods. There are several ways in which this work can be extended. This includes exploiting the generalisation abilities of a specific learned method. For instance, a learned method could be found by training on a low-dimensional system, which is cheap to simulate, and then applying it to more sophisticated higher-dimensional problems. One could also investigate longer splittings with a larger number of subflows. Further work could also be done to remove the dependence on labelled training data or augment the loss function with regularisers, which penalise the violation of higher-order conditions.

Acknowledgements. The computational studies made use of the Nimbus High Performance Computing (HPC) Services at the University of Bath (University of Bath’s Research Computing Group (doi.org/10.15125/b6cd-s854)).

REFERENCES

- [1] E. AKHMATSKAYA, M. FERNÁNDEZ-PENDÁS, T. RADIVOJEVIĆ, AND J. M. SANZ-SERNA, Adaptive splitting integrators for enhancing sampling efficiency of modified hamiltonian monte carlo methods in molecular simulation, *Langmuir*, (2017).
- [2] A. H. AL-MOHY AND N. J. HIGHAM, A new scaling and squaring algorithm for the matrix exponential, *SIAM Journal on Matrix Analysis and Applications*, 31 (2010), pp. 970–989.
- [3] J. BERNIER, Exact splitting methods for semigroups generated by inhomogeneous quadratic differential operators, 2020.
- [4] J. BERNIER, N. CROUSEILLES, AND Y. LI, Exact splitting methods for kinetic and schrödinger equations, 2019.
- [5] S. BLANES, F. CASAS, AND A. MURUA, Splitting and composition methods in the numerical integration of differential equations, *Boletín de la Sociedad Española de Matemática Aplicada*, (2008).
- [6] S. BLANES, F. CASAS, AND A. MURUA, Splitting methods for differential equations, *Acta Numer.*, (2024).
- [7] S. BLANES, F. CASAS, AND J. M. SANZ-SERNA, Numerical integrators for the hybrid monte carlo method, *SIAM Journal on Scientific Computing*, 36 (2014), p. A1556–A1580.
- [8] M. BLONDEL, Q. BERTHET, ET AL., Efficient and modular implicit differentiation, *arXiv:2105.15183*, (2021).
- [9] A. BONFIGLIOLI AND R. FULCI, Topics in Noncommutative Algebra: The Theorem of Campbell, Baker, Hausdorff and Dynkin, vol. 2034, Springer Berlin, Heidelberg, 01 2012.

- [10] J. BRADBURY, R. FROSTIG, P. HAWKINS, ET AL., JAX: composable transformations of Python+NumPy programs, 2018.
- [11] R. T. Q. CHEN, Y. RUBANOVA, J. BETTENCOURT, AND D. DUVENAUD, Neural ordinary differential equations, in Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, Curran Associates Inc., 2018, pp. 6572–6583.
- [12] X. CHEN, C. LIANG, D. HUANG, ET AL., Symbolic discovery of optimization algorithms, Advances in neural information processing systems, 36 (2024).
- [13] J. W. COOLEY AND J. W. TUKEY, An algorithm for the machine calculation of complex fourier series, Mathematics of computation, 19 (1965), pp. 297–301.
- [14] J. DUCHI, E. HAZAN, AND Y. SINGER, Adaptive subgradient methods for online learning and stochastic optimization., Journal of machine learning research, 12 (2011).
- [15] K. ENGEL, S. BRENDLE, R. NAGEL, ET AL., One-Parameter Semigroups for Linear Evolution Equations, Graduate Texts in Mathematics, Springer New York, 1999.
- [16] L. EVANS, Partial Differential Equations, Graduate studies in mathematics, American Mathematical Society, 2010.
- [17] P. GIORDANO AND L. L. BAGLINI, A Picard-Lindelöf theorem for smooth PDE, 2022.
- [18] R. GLOWINSKI, S. OSHER, AND W. YIN, Splitting Methods in Communication, Imaging, Science and Engineering, Springer Cham, 01 2017.
- [19] T. G. GROSSMANN, U. J. KOMOROWSKA, J. LATZ, AND C.-B. SCHÖNLIEB, Can physics-informed neural networks beat the finite element method?, 2023.
- [20] E. HAIRER, C. LUBICH, AND G. WANNER, Geometric numerical integration, vol. 31 of Springer Series in Computational Mathematics, Springer-Verlag, Berlin, second ed., 2006.
- [21] E. HAIRER, S. NØRSETT, AND G. WANNER, Solving Ordinary Differential Equations I: Nonstiff Problems, Springer Series in Computational Mathematics, Springer, 2008.
- [22] R. HORST AND H. TUY, Global Optimization: Deterministic Approaches, Springer, 1996.
- [23] A. ISERLES, H. Z. MUNTHE-KAAS, S. P. NØRSETT, AND A. ZANNA, Lie-group methods, Acta Numer., 9 (2000), pp. 215–365.
- [24] G. E. KARNIADAKIS, I. G. KEVREKIDIS, L. LU, P. PERDIKARIS, S. WANG, AND L. YANG, Physics-informed machine learning, Nature Reviews Physics, 3 (2021), pp. 422–440.
- [25] D. P. KINGMA AND J. BA, Adam: A method for stochastic optimization, arXiv:1412.6980, (2014).
- [26] K. LEVENBERG, A method for the solution of certain non-linear problems in least squares, Quarterly of applied mathematics, 2 (1944), pp. 164–168.
- [27] D. W. MARQUARDT, An algorithm for least-squares estimation of nonlinear parameters, Journal of the society for Industrial and Applied Mathematics, 11 (1963), pp. 431–441.
- [28] R. I. MCLACHLAN AND G. R. W. QUISPTEL, Splitting methods, Acta Numer., 11 (2002), pp. 341–434.
- [29] C. MOLER AND C. VAN LOAN, Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later, SIAM Review, 45 (2003), pp. 3–49.
- [30] M. M. NOEL, A new gradient based particle swarm optimization algorithm for accurate computation of global minimum, Applied Soft Computing, 12 (2012), pp. 353–359.
- [31] C. OFFEN AND S. OBER-BLÖBAUM, Symplectic integration of learned hamiltonian systems, Chaos: An Interdisciplinary Journal of Nonlinear Science, 32 (2022).
- [32] I. OMELYAN, I. MRYGLOD, AND R. FOLK, Symplectic analytically integrable decomposition algorithms: classification, derivation, and application to molecular dynamics, quantum and celestial mechanics simulations, Comput. Phys. Comms., 151 (2003), pp. 272–314.
- [33] M. RAISSI, P. PERDIKARIS, AND G. KARNIADAKIS, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, Journal of Computational Physics, 378 (2019), pp. 686–707.
- [34] M. RUGGENTHALER, M. PENZ, AND R. VAN LEEUWEN, Existence, uniqueness, and construction of the density-potential mapping in time-dependent density-functional theory, Journal of Physics: Condensed Matter, 27 (2015), p. 203202.
- [35] J. J. SAKURAI AND J. NAPOLITANO, Modern Quantum Mechanics, Addison Wesley, 2 ed., 1985.
- [36] P. SINGH, A PyTorch compatible Differentiable Numerical Algorithms package for computational quantum mechanics. <https://pypi.org/project/expsolve/>.
- [37] M. SUZUKI, Fractal decomposition of exponential operators with applications to many-body theories and monte carlo simulations, Physics Letters A, 146 (1990), pp. 319–323.
- [38] K. YAJIMA, Existence of solutions for Schrödinger evolution equations, Communications in Mathematical Physics, 110 (1987), pp. 415 – 426.

Appendix A. Triple jump and composition methods. We describe the triple jump technique [20, II: 4.2] as an example of a composition method. Given a numerical method $\Psi_h^{[f]}(\cdot; \alpha, \beta)$ of order p , we can construct a higher-order numerical method by writing

$$(A.1) \quad \Psi_h^{[f]}(\cdot; \tilde{\alpha}, \tilde{\beta}) = \Psi_{\mu_3 h}^{[f]}(\cdot; \alpha, \beta) \circ \Psi_{\mu_2 h}^{[f]}(\cdot; \alpha, \beta) \circ \Psi_{\mu_1 h}^{[f]}(\cdot; \alpha, \beta),$$

where,

$$(A.2) \quad \mu_1 = \mu_3 = \frac{1}{2 - 2^{1/(p+1)}}, \quad \mu_2 = 1 - \mu_1 - \mu_3.$$

Due to the symmetry of the parameters that define the triple jump, $\mu = [\mu_1, \mu_2, \mu_3]$, within the larger class of composition methods, applying the triple jump to a symmetric method results in a new symmetric method. Hence the new method defined by (A.1) is of order $p + 1$ in general and of order $p + 2$ if the original method is symmetric. The coefficients $[\tilde{\alpha}, \tilde{\beta}]$ that define the new method can be constructed from the coefficients $[\alpha, \beta]$ of the original method in a straightforward way.

Applying the triple jump technique to the second order Strang results in a splitting method of order four from the Yoshida family, referred to as Yoshida in this paper. The triple jump technique can be inductively repeated to construct higher-order splitting methods. However, the number of stages grows exponentially in this approach: constructing a method of order $p \gg 1$ will result in $\mathcal{O}(3^p)$ stages.

Appendix B. Explicit parameter transform. To define the matrices $A \in \mathbb{R}^{K \times \lfloor \frac{K-1}{2} \rfloor}$, $B \in \mathbb{R}^{K \times \lfloor \frac{K-2}{2} \rfloor}$ and the vectors $C, D \in \mathbb{R}^K$ for the parameter transform in (3.7), we distinguish between K even and odd. For K even, we have

$$A = \begin{pmatrix} I_{\frac{K-2}{2}} \\ -1_{2, \frac{K-2}{2}} \\ J_{\frac{K-2}{2}} \end{pmatrix} \in \mathbb{R}^{K \times \frac{K-2}{2}}, \quad C = \begin{pmatrix} 0_{\frac{K-2}{2}, 1} \\ 0.5_{2, 1} \\ 0_{\frac{K-2}{2}, 1} \end{pmatrix} \in \mathbb{R}^K,$$

$$B = \begin{pmatrix} I_{\frac{K-2}{2}} \\ -2_{1, \frac{K-2}{2}} \\ J_{\frac{K-2}{2}} \\ 0_{1, \frac{K-2}{2}} \end{pmatrix} \in \mathbb{R}^{K \times \frac{K-2}{2}}, \quad D = \begin{pmatrix} 0_{\frac{K-2}{2}, 1} \\ 1 \\ 0_{\frac{K-2}{2}, 1} \\ 0 \end{pmatrix} \in \mathbb{R}^K,$$

while for K odd, we obtain,

$$A = \begin{pmatrix} I_{\frac{K-1}{2}} \\ -2_{1, \frac{K-1}{2}} \\ J_{\frac{K-1}{2}} \end{pmatrix} \in \mathbb{R}^{K \times \frac{K-1}{2}}, \quad C = \begin{pmatrix} 0_{\frac{K-1}{2}, 1} \\ 1 \\ 0_{\frac{K-1}{2}, 1} \end{pmatrix} \in \mathbb{R}^K,$$

$$B = \begin{pmatrix} I_{\frac{K-3}{2}} \\ -1_{2, \frac{K-3}{2}} \\ J_{\frac{K-3}{2}} \\ 0_{1, \frac{K-3}{2}} \end{pmatrix} \in \mathbb{R}^{K \times \frac{K-3}{2}}, \quad D = \begin{pmatrix} 0_{\frac{K-3}{2}, 1} \\ 0.5_{2, 1} \\ 0_{\frac{K-3}{2}, 1} \\ 0 \end{pmatrix} \in \mathbb{R}^K.$$

Here, $I_s \in \mathbb{R}^{s \times s}$ denotes the identity matrix and we write $J_s \in \mathbb{R}^{s \times s}$ for the exchange matrix. Further, $r_{p,q} \in \mathbb{R}^{p \times q}$ is filled with the scalar r .

Appendix C. Construction of training datasets. The procedure for generating a single batch \mathcal{B} of random initial conditions which can be used for training

and validation is explicitly described in Algorithm C.1. In lines 5, 8 and 11 of Algorithm C.1, the function $g(\cdot; \bar{x}_0, \sigma)$ is a Gaussian with mean \bar{x}_0 and standard deviation σ , namely

$$(C.1) \quad g(x; x_0, \sigma) := \mathcal{Z} \exp \left[-\frac{1}{2} \left(\frac{x - \bar{x}_0}{\sigma} \right)^2 \right].$$

Here, \mathcal{Z} is a suitable normalisation constant such that $\sum_{m=1}^M |g(x_m; x_0, \sigma)|^2 = 1$, where x_m , $m = 1, \dots, M$, denotes the spatial discretisation introduced in Section 4.1. As already remarked in Section 4.3.1, Algorithm C.1 implicitly defines the distribution \mathcal{U} of initial conditions used for training. Figure 11 shows six randomly chosen initial

Algorithm C.1 Generation of training batch $\mathcal{B} = \{u_0^{(0)}, u_0^{(1)}, \dots, u_0^{(b-1)}\}$.

- 1: Set width $\sigma = 0.5$.
 - 2: **for** $j = 0, 1, \dots, b - 1$ **do**
 - 3: Draw normally distributed random mean $x_0^{(j)} \sim \mathcal{N}(-\sqrt{5}, 0.1)$.
 - 4: Draw uniformly distributed random numbers $\xi_j^{(1)}, \xi_j^{(2)}, \xi_j^{(3)}, \xi_j^{(4)} \sim \mathcal{U}(0, 1)$.
 - 5: Set $\tilde{\phi} = \begin{cases} g(\cdot; \bar{x}_0, \sigma) & \text{with } \bar{x}_0 \sim \mathcal{N}(-\sqrt{5}, 0.1), \text{ if } j = 0, \\ \psi_T^{[f]}(u_0^{(j-1)}), & \text{otherwise.} \end{cases}$
 - 6: **if** $\xi_j^{(1)} < 0.5$ **then**
 - 7: Set $\tilde{\phi} \mapsto \tilde{\phi} + g(\cdot; x_0^{(j)}, \sigma)$ (add Gaussian).
 - 8: **end if**
 - 9: **if** $\xi_j^{(2)} < 0.5$ **then**
 - 10: $\tilde{\phi} \mapsto \exp[2\pi i \xi_j^{(3)}] \tilde{\phi}$ (apply a random phase shift).
 - 11: **end if**
 - 12: **if** $\xi_j^{(4)} < 0.01$ **then**
 - 13: $\tilde{\phi} = g(\cdot; x_0^{(j)}, \sigma)$ (reset to Gaussian).
 - 14: **end if**
 - 15: Set $u_0^{(j)} = \tilde{\phi} / \tilde{\mathcal{Z}}$ with the normalisation constant $\tilde{\mathcal{Z}} = \sum_{m=1}^M |\tilde{\phi}(x_m)|^2$.
 - 16: **end for**
-

conditions $u_0^{(j)}$ that were generated with Algorithm C.1. Note that the functions $u_0^{(j)}$ are concentrated around the left minimum $x_- = -\sqrt{5}$ of the double-well potential.

Appendix D. Further details on the choice of the stochastic optimisation algorithm. Figure 12 (left) shows the evolution of the training- and validation loss function in (3.9) for the Schrödinger equation (4.2) using Adam for the stochastic optimisation in line 8 of the learning pipeline in Algorithm 3.1. The loss is plotted as a function of the iterations for learning the optimal splittings with lengths $K = 5$ and $K = 8$. For $K = 8$, two different minima were identified, and these are labelled “Learn8A” and “Learn8B” respectively; the learned splitting method for $K = 5$ is referred to as “Learn5A”. Both the training loss and the validation loss (which is much smoother since it is always evaluated on the same batch) are shown. The corresponding evolution of the splitting parameters γ is shown in Figure 12 (right). As the plot demonstrates, all parameters converge after 250 iterations.

D.1. Comparison of different optimisers. Many different optimisers such as Adam [25], AdaGrad [14] or Lion [12] could be chosen in the SO step in line 8 of

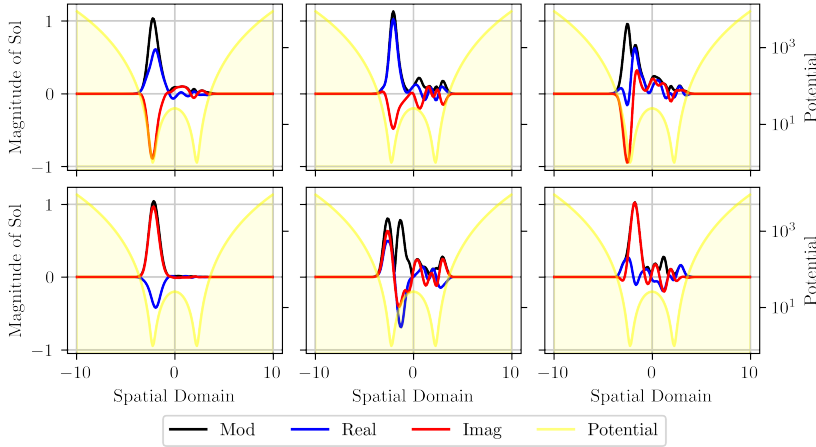


FIG. 11. Six randomly chosen initial conditions $u_0^{(j)}$ generated with Algorithm C.1, with modulus, real and imaginary part of $u_0^{(j)}$ depicted in black, blue and red, respectively. In each of the plots, the double-well potential is also shown on a logarithmic scale in the background.

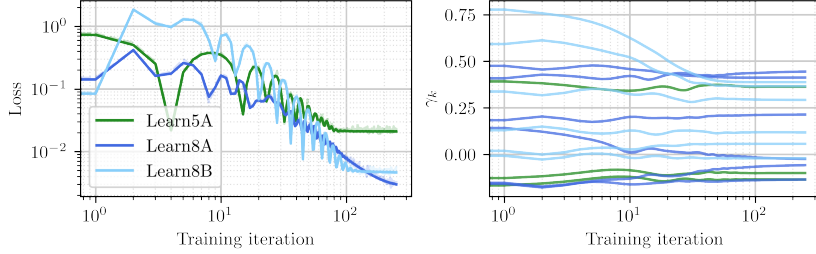


FIG. 12. Training- and validation loss (left) and evolution of the splitting parameters for the three learned methods Learn5A, Learn8A and Learn8B where we used Adam for the stochastic optimisation.

Algorithm 3.1. Since the number of learned parameters is low, this includes second-order methods such as Levenberg-Marquardt [26, 27] which require the evaluation of the Hessian. For each of these methods performance can be tuned by varying the relevant hyperparameters such as the learning rate, momentum and weight decay in the stochastic gradient descent on batched training data.

Figure 13 illustrates the performance of AdaGrad, Adam, Lion and Levenberg-Marquardt on our training data when trying to find the minimum for a splitting method with three free parameters γ_1 , γ_2 and γ_3 . To obtain this figure, we started near the novel learned splitting Learned5A by setting the initial values of the parameters to $\gamma_1 = 0.374129 \pm \delta$, $\gamma_2 = -0.109994 \pm \delta$, $\gamma_3 = -0.123223 \pm \delta$ where $\delta \in \{-0.05, +0.05, -0.1, +0.1\}$ and explored the loss landscape. The loss function in the upper left plot in Figure 13 shows that all optimisers apart from Lion, converged to numerical methods with error of around 0.02. For reference, the exact final values of the parameters obtained with the different methods and the corresponding value of the loss function are tabulated in Table 4. We conclude that they are all good candidates for optimisers to be used in line 8 of Algorithm 3.1. As Adam is known to be an efficient and widely used algorithm, all other numerical experiments employ

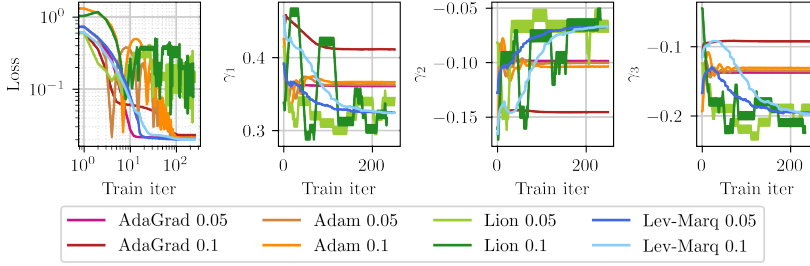


FIG. 13. Comparison of different optimisers with associated learning rates 0.05 and 0.1 for finding a splitting method with three free parameters. The plot in the top left corner shows the evolution of the loss function; the other three plots visualise the corresponding evolution of the parameters γ_1 , γ_2 and γ_3 .

Adam.

optimizer	LR	γ_1	γ_2	γ_3	loss \mathcal{L}
AdaGrad	0.05	0.3609	-0.09857	-0.1375	0.02096
	0.1	0.4114	-0.1455	-0.09246	0.2316
Adam	0.05	0.3627	-0.1003	-0.1353	0.02106
	0.1	0.3665	-0.1036	-0.1309	0.0212
Lion	0.05	0.3446	-0.07162	-0.1938	0.1275
	0.1	0.3169	-0.06039	-0.2246	0.07843
Levenberg-Marquardt	0.05	0.3245	-0.0671	-0.1963	0.02011
	0.1	0.3242	-0.06704	-0.1966	0.2013

TABLE 4

Final values of the parameters γ_1 , γ_2 and γ_3 and loss function when trained over 250 epochs with different optimisers and learning rates LR.

Appendix E. Visualisation of splitting methods. In the following we describe how the visualisation of splitting methods described in Section 4.3.3 arises from the solution of a the simple two-dimensional IVP $\dot{u}(t) = [1, 1]^\top$, $u(0) = [0, 0]^\top$, $t \in [0, 1]$ for $u(t) = (u_1(t), u_2(t))^\top \in \mathbb{R}^2$. The exact analytical solution $u(t) = [t, t]^\top$ corresponds to a diagonal line from $[0, 0]$ to $[1, 1]$. The natural splitting is $f^{[1]} = [1, 0]^\top$, $f^{[2]} = [0, 1]^\top$. The corresponding sub-flows $\psi_{\delta t}^{[1]} : (u_1(t), u_2(t))^\top \mapsto (u_1(t + \delta t), u_2(t))^\top$ and $\psi_{\delta t}^{[2]} : (u_1(t), u_2(t))^\top \mapsto (u_1(t), u_2(t + \delta t))^\top$ translate the state parallel to the coordinate axes. For $0 \leq \tau \leq 1$ we can now define the flux

$$\Psi_{\tau h}^{[f]}(\cdot; \alpha, \beta) = \begin{cases} \psi_{(\tau - S_{k'})h}^{[1]} \circ \bigcirc_{k=1}^{k'-1} \psi_{\beta_k h}^{[2]} \circ \psi_{\alpha_k h}^{[1]} & \text{for } S_{k'} \leq \tau \leq S_{k'} + \alpha_{k'} \\ \psi_{(\tau - \alpha'_k - S_{k'})h}^{[2]} \circ \psi_{\alpha'_k h}^{[1]} \circ \bigcirc_{k=1}^{k'-1} \psi_{\beta_k h}^{[2]} \circ \psi_{\alpha_k h}^{[1]} & \text{for } S_{k'} + \alpha_{k'} \leq \tau \leq S_{k'+1} \end{cases}$$

where the partial sums $S_{k'}$ are given by

$$S_{k'} = \sum_{k=1}^{k'-1} \beta_k + \alpha_k \quad \text{for } k' = 1, 2, \dots, K.$$

As τ increases from 0 to 1, this flux will translate a given state u to $\Psi_h^{[f]}(u; \alpha, \beta)$ along

a connected path of straight line segments that are parallel to the coordinate axes and whose (orientated) lengths are given by β_k and α_k respectively. This path can thus be regarded as a visualisation of the numerical time-stepping method defined by (α, β) which resolves the evolution within a single timestep of size h . Figure 4 visualises the evolution of the initial state $[0, 0]$ under different learned and reference methods for $h = 1$. Consistency of a method is equivalent to the path ending at $[1, 1]$, and for consistent methods symmetry is akin to having discrete 180° rotational symmetry about the point $[0.5, 0.5]$. By comparing Strang with $4 \times$ Strang in Figure 4 it can be seen that decreasing the step size of a method brings the path closer to the diagonal line which represents the analytic solution at the cost of increasing the number of line segments.