Solving the Schrödinger equation using program synthesis

Scott Habershon[1, a)]

Department of Chemistry, University of Warwick, Coventry, CV4 7AL, United Kingdom.

We demonstrate that a program synthesis approach based on a linear code representation can be used to generate algorithms which approximate the ground-state solutions of one-dimensional time-independent Schrödinger equations constructed with bound polynomial potential energy surfaces (PESs). Here, an algorithm is constructed as a linear series of instructions operating on a set of input vectors, matrices and constants which define the problem characteristics, such as the PES. Discrete optimization is performed using simulated annealing in order to identify sequences of code-lines, operating on the program inputs, which can reproduce the expected ground-state wavefunctions $\psi(x)$ for a set of target PESs. The outcome of this optimization is not simply a mathematical function approximating $\psi(x)$, but is instead a complete algorithm which converts the input vectors describing the system into a ground-state solution of the Schrödinger equation. These initial results point the way towards an alternative route for developing novel algorithms for quantum chemistry applications.

[a)]Electronic mail: S.Habershon@warwick.ac.uk

I. INTRODUCTION

The last two decades have witnessed a rapid growth in the availability of chemical datasets, particularly those derived from *ab initio* quantum chemical calculations;[1–8] this is turn has driven the further adoption of artificial intelligence (AI) and machine learning (ML) as tools to predict and explain the properties of complex chemical systems.[9–18] As a prototypical example, a wide range of schemes have been developed to provide accurate fitting of potential energy surfaces (PESs), typically using input data (such as molecular geometries and energies) calculated using *ab initio* quantum chemical methods;[5,19–31] different combinations of molecular descriptors, regression strategies and datasets yield a wide variety of fitting strategies, with recent examples from this domain emphasizing the importance of transferability across different chemical systems.[20,21,31,32] A further growth area of ML in chemistry over the last two decades has been the automated prediction of chemical reaction mechanisms and synthesis routes;[33–37] again, the availability of experimental or computational datasets, coupled to a wide range of AI/ML algorithms, has led to novel computational retrosynthesis approaches which, in some cases, compete with traditional human expertise. With the ongoing availability of low-cost/high-performance computing, these developments in AI/ML applied to chemical data would be expected to continue to grow.

Against the backdrop of AI/ML, a fundamentally different approach to computational science is *program synthesis* (PS).[38–45] Here, the central idea is to use enumeration, AI or other optimization strategies to generate complete computer programs which achieve a user-defined task subject to performance criteria; in other words, the focus in PS is on the generation of computer code, rather than the more common tasks of regression or classification encountered in applying AI/ML in the computational chemical sciences. The purpose of this Article is to explore the application of PS tools to derive novel algorithms to solve a benchmark problem in quantum chemistry, namely determination of solutions to the time-independent vibrational Schrödinger equation.

In the context of quantum chemistry, it is worth highlighting that previous automatic formula derivation methods and PS strategies have found success in the domain of *ab initio* electronic structure theory.[46–48] For example, Hirata's Tensor Contraction Engine (TCE),[46,47] takes as input an electronic wavefunction *ansatz*, such as that given in coupled-cluster schemes, and subsequently generates appropriate tensor contractions and associated instruction sets to enable highly-efficient computational implementation of the underlying theory. Here, working in the second quantization formalism, TCE operates by applying well-defined contraction rules to generate efficient formu-

lae for evaluating relevant operator matrix elements, and implements the resulting code for these contractions. This scheme provides an impressive demonstration of the power of PS in tackling numerically daunting tasks, but differs from the generalized inductive optimization strategy employed in this Article, where the focus is not on deployment of rigorous contraction rules to optimize equation implementation, but is instead aimed at exploring novel algorithms which can solve target problems in previously-unknown ways using previously-unknown methods.

In this Article, we consider the application of inductive PS to the problem of generating code and algorithms which can solve the time-independent vibrational Schrödinger equation. Within this inductive programming paradigm, we assume that we have a set of examples of expected outputs required from our code when executed on a corresponding set of inputs; these correct input/output examples are assumed to be provided by an *oracle* code which can exactly produce the desired output for any input. Within this inductive paradigm, the problem of PS can be cast as a challenge in optimization; one seeks the set of computer instructions which form a code to produce the expected output given each example input. In this sense, this inductive PS can be viewed as somewhat comparable to the usual training of an artificial neural network (ANN), where one optimizes network weights using a set of input/output examples; the key difference in PS is that the output is not a series of ANN weights, but is instead a complete computer program to perform the required processing. As such, by focussing on algorithm generation, this approach has the potential to automatically discover novel theories or functional patterns which might not have been otherwise anticipated; investigating whether this strategy can be ported into the realm of computational chemistry is an important goal of this article.

Here, we are interested not in function approximation or fast implementation of defined tensor contractions, but in generating complete *codes* which suggest general routes to solving the time-independent Schrödinger equation. With the challenge of PS framed as an optimization problem, a range of different approaches are commonly applied to auto-generate code which satisfies the target program performance. In this context, genetic programming (GP) has been used extensively.[43–45,49,50] Here, a code is typically represented as a tree structure constructed from a set of pre-defined primitive functions (such as $\times, \div, +, -$) and constants; arranging this set of functions and constants in a variable tree structure acting on a set of inputs creates a computer program which gives a specific output for each input which is presented. In GP applied to PS, a population of tree codes is evolved under the action of evolutionary operations; the fitness of each population member is given by an appropriate metric indicating how accurately the tree-code yields the ex-

pected target outputs, and the principle of Darwinian evolution is used to drive optimization of the population towards better-performing codes. As we describe below, we use a simpler simulated annealing (SA) protocol to perform PS here, in combination with a code representation which is comparable to that employed in Cartesian GP.[44,51] Our interest here is in exploring the challenges and possibilities for PS in the domain of quantum chemistry, and we leave to further development of targeted optimization strategies and code representations to future work.

In this Article, we investigate the use of a linear PS system to generate solutions of the time-independent Schrödinger equation. In this initial proof-of-concept study, we primarily investigate the extent to which PS enables solution of the one-dimensional (1-D) vibrational Schrödinger equation for bound polynomial PESs; while simple, such systems serve as a useful starting point for which numerically-exact solutions can be readily generated to enable evaluation of PS-generated algorithms. In contrast to schemes used in electronic structure theory (such as TCE[46,47]) or previous GP-based schemes,[50] our strategy does not optimize implementation of a specific target wavefunction *ansatz*, but instead uses a search in a space defined by possible instruction sequences, guided by an optimization function assessing code performance. The remainder of this Article is organized as follows. First, in Section II, we discuss our own implementation of a linear PS system, combined with a simulated annealing (SA) optimization protocol. In Section III, we then show how our PS strategy can readily generate novel algorithms (codes) which solve the Schrödinger equation for arbitrary bound polynomial PESs; for example, by optimizing code structures for a set of just 10 input/output examples, we find that the resulting codes give accurate ground-state wavefunctions for a further 10^3 random PES examples. We also investigate some aspects of our PS set-up in an attempt to better understand the impact of algorithm parameters on predictive performance. In Section V, we discuss current limitations of our PS approach, and offer some further discussion as to how the PS strategy might be extended to enable prediction of properties for more complex, many-dimensional systems. Finally, Section VI summarizes this contribution.

II. THEORY

Here, we first describe our target application, namely determination of the ground-state wavefunctions for one-dimensional Schrödinger equations with bound PESs. We then explain how algorithms are represented in our linear PS approach, before describing how the performance of

computer-generated code is optimized using a SA protocol. We also give details of the input, output and internal function sets which are used in our implementation to form solutions to Eq. 1.

A. Problem representation

As a benchmark problem, we will seek to identify codes which can produce the ground-state solution $\psi(x)$ for the 1-D time-independent Schrödinger equation:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2}+V(x)\psi(x)=E_0\psi(x). \tag{1}$$

Here, m is the particle mass, and E_0 is the total energy of the eigenstate $\psi(x)$; in what follows, we adopt atomic units such that $\hbar=1$, and we will only consider systems for which $m=1$. As emphasized below, and in contrast to previous research,[50] we are aiming to generate entire computer programs which can solve the eigenproblem of Eq. 1 to give $\psi(x)$ for more general $V(x)$.

In all of the following discussion, we will aim to generate algorithms for PESs $V(x)$ which are of a general polynomial form given by

$$V(x)=\sum_{k=1}^{N_p}a_k x^k, \tag{2}$$

where N_p is the polynomial order and $\mathbf{a}$ is a set of random coefficients chosen in the range $[-5,+5]$. To encourage generation of algorithms which are transferrable across different polynomial orders, the order N_p is randomly-selected $N_p\in[2,6]$ whenever a PES is required as input during our PS strategy. Furthermore, we require that any $V(x)$ is bound, such that it approaches a sufficiently large value (chosen to be $V(x)=5$ here) at the boundaries of the coordinate range-of-interest (see below). This set-up for $V(x)$ is chosen because, by choosing different random coefficients $\mathbf{a}$, it allows us to generate arbitrary numbers of different $V(x)$ examples which can be used as target inputs to verify the accuracy of codes generated by PS.

Based on the above definitions, our goal is to now automatically synthesize algorithms which output $\psi(x)$ for a given $V(x)$. However, to make progress, we must first define how the mathematical problem captured by Eq. 1 will be represented in our computer-generated codes; here, we adopt a computationally-simple and practical approach, representing both $V(x)$ and $\psi(x)$ as real-valued numbers on a uniform grid $\mathbf{x}$. The vectors denoting these properties on the grid-points

will be labelled $\mathbf{V}$ and ψ, respectively; in all calculations reported below, the uniform grid comprises $n_g = 101$ grid-points uniformly distributed on the range $x_i \in [-5, +5]$. This grid-based approach is well-known in the field of quantum simulations, as in the discrete variable representation (DVR[52–55]) which forms the basis of common strategies for solving Eq. 1; the connection between DVR methods and the algorithms generated here is discussed further below. Alternative representations of the input $V(x)$ and output $\psi(x)$ can clearly be explored in the future, whereas the focus of the current article is in providing an initial proof-of-concept.

B. Program representation

Previous work on PS has employed a variety of different structures for representing computer programs, such as tree structures, directed acyclic graphs, and fixed-size grids.[43–45,49,50] In this article, we choose to use a program representation which is analogous to that used in Cartesian GP, namely using a grid of functional nodes which operate on input vectors, constants and matrices.[44,56,57] Although we anticipate that a tree structure, as commonly employed in GP, can be used to solve the same problem, a grid-like code representation is selected here for its simplicity, flexibility and ease of interpretation. Of course, there is no guarantee that the chosen program representation used here is optimal in any way, and the impact of the program structure will be a subject of interest in future work.

In our approach, illustrated in Fig. 1, a program is represented on a grid comprising columns of n_r primitive functions organized in a sequentially-labelled column vector; a set of n_c of these columns are then arranged to comprise a $n_r \times n_c$ matrix of functions. In addition to this grid of functions, our program structure also comprises a set of n_i possible input nodes which define different possible problem-related characteristics, such as $\mathbf{V}$ and $\mathbf{x}$, as described in Section II D. Furthermore, as is common in GP, an additional input vector is also supplied, comprising any suitable input constants relevant to the problem domain.

Each of the n_r entries in each column represents a primitive mathematical or programmatic function which operates directly on the inputs to produce a new output. In this article, we constrain the PS system to generation of linear codes. Here, after selection of one of the n_i inputs from the possible input set, a series of n_c instructions operate on these inputs sequentially; in other words, the program state moves from left to right in Fig. 1, performing operations on the inputs. As highlighted in Fig. 1, the program inputs are typically a workspace vector $\mathbf{y}$ and a workspace

matrix $\mathbf{M}$, initialized to some input values depending on the choice of input function.

A sequence of $N = n_c + 2$ integers define a path through the network illustrated in Fig. 1. The identity of each function node at the input, output and each of the n_c code-layers can be treated as a discrete variable, such that a program can be defined by a sequence of N integers; modification of these integer entries gives different sequences of operations (Fig. 1), hence different computer programs. As described in Section II C, proposal of a computer program to solve Eq. 1 then represents an optimization problem in which the instruction identities at the n_c layers are the variables.

Each operation at each layer acts on the inputs (*i.e.* defined workspace vector and matrix) passed from the previous code layer. We note that, at each code layer, the applied instruction is not required to modify all inputs, and can operate on just one of the inputs at a time; an alternative way of viewing this operation is that the sequence of n_c operations modify the set of input vectors and matrices "in place". After the sequence of n_c operations are complete, the final output of the code is generated by an output layer comprising n_o possible output instructions which ensure that a real-valued vector, $\mathbf{w}$, is output from the code-sequence, as described in Section II F. The vector $\mathbf{w}$ is subsequently interpreted as the current program's proposed solution $\psi(x)$ to the Schrödinger equation defined by the input PES $V(x)$.

C. Program optimization

To generate computer programs which solve Eq. 1, we use inductive PS. Here, we assume that we have available a set of M input PESs $\{\mathbf{V}_i\}_{i=1}^{M}$ (evaluated on the underlying coordinate grid $\mathbf{x}$) and corresponding output values $\{\psi_i\}_{i=1}^{M}$ provided by a so-called *oracle* code which can solve the required problem exactly. In the current example, we use the standard Colbert-Miller DVR method[58] to generate the exact $\psi(x)$ for each target $V(x)$.

To identify new computer codes which solve the target problem, we seek to minimize the errors in the program's proposed solutions of Eq. 1 for the set $\{\mathbf{V}_i\}_{i=1}^{M}$. To do so, we define an error function, here given as the norm of the difference between the targets $\{\psi_i\}_{i=1}^{M}$ and the current code outputs $\{\mathbf{w}_i\}_{i=1}^{M}$:

$$F = \frac{1}{M}\sum_{j=1}^{M} |\psi_j - \left(S_j \mathbf{w}_j\right)|. \tag{3}$$

Here, ψ_j is the vector containing the exact ground-state wavefunction for the Schrödinger equa-

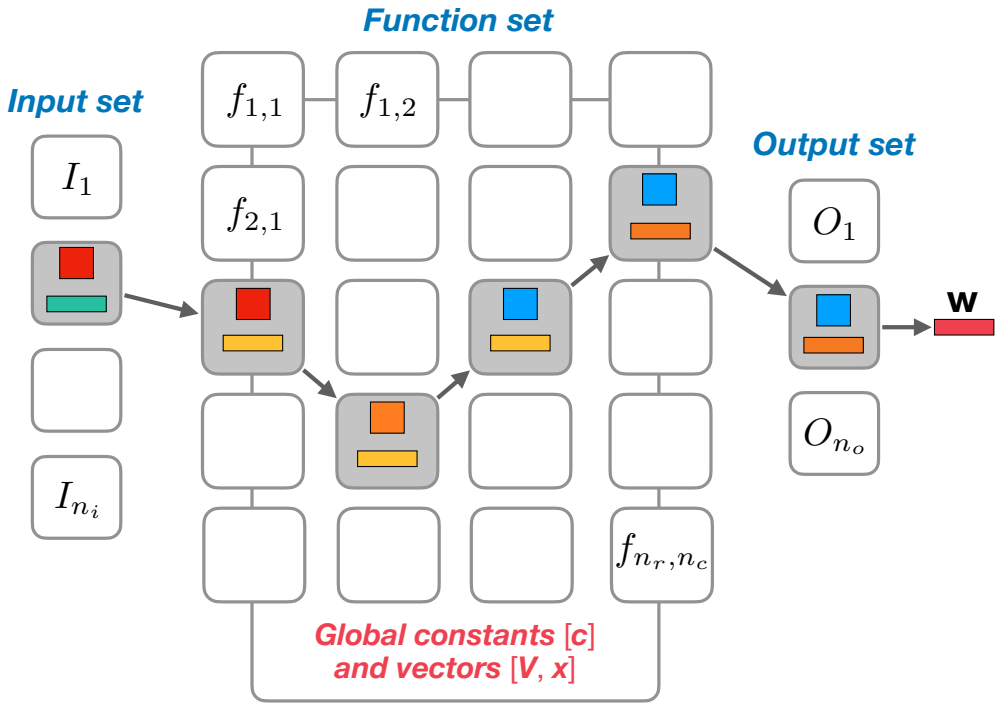

FIG. 1. Schematic representation of the program synthesis used to generate algorithms which solve Eq. 1. In the example shown, there are four possible input sets and three possible output functions; there are also n_r internal functional choices (with $n_r = 5$ in this example) and n_c internal code-lines (with $n_c = 4$ in this example). On the left, the different available input functions define the input workspace vector $\mathbf{y}$ (shown as a horizontal bar) and workspace matrix $\mathbf{M}$ (shown as a square). Moving from left to right, these objects are then modified by the n_c internal functions; here, one particular pathway through the internal functions is highlighted, corresponding to one particular algorithm. At the output function set on the right, the final vector output $\mathbf{w}$ from the proposed algorithm is generated; this is interpreted as the current algorithm's approximation of the ground-state wavefunction of Eq. 1. Our program synthesis strategy can be viewed as the search for the pathway through the network of interconnected instructions which produces outputs $\mathbf{w}$ that match a target output provided by an existing oracle code.

tion with PES $\mathbf{V}_j$, evaluated on the coordinate grid, and w_j is the vector output predicted by the current program structure for the same PES $\mathbf{V}_j$. We note that, because the overall phase-factor of a predicted wavefunction $\mathbf{w}_j$ may be different to that of the target ψ_j, without changing energy expectation values, the sign $S_j = \pm 1$ is independently chosen as the value giving the best agreement for each target wavefunction.

To minimize the error F, we use a SA protocol in which the function identities at the input

layer, n_c internal layers, and the output layer are iteratively modified. In other words, the variables to be optimized are the (n_c+2) integer identities which define the instructions to be used at the input, internal and output layers. At each SA iteration, we randomly change a small number of functions $m \in [1,4]$ in randomly selected program layers, as shown in Fig. 2. After updating the current code, the new error function, F_{new}, is evaluated using the test data, $\{\mathbf{V}_i\}_{i=1}^{M}$ and $\{\psi_i\}_{i=1}^{M}$. The new code is then accepted or rejected using the usual Metropolis criterion, with probability $P = \min(1, e^{-\beta[F_{new}-F_{old}]})$, where $\beta = 1/(k_B T)$, T is the current annealing temperature, and F_{old} is the error value for the original code. This procedure is repeated for a fixed number of SA iterations, with the temperature T linearly decreased in order to encourage increasingly localized searching amongst minima on the landscape defined by F. After a pre-defined number of SA iterations, the optimization is stopped, and the final code can be further studied and assessed.

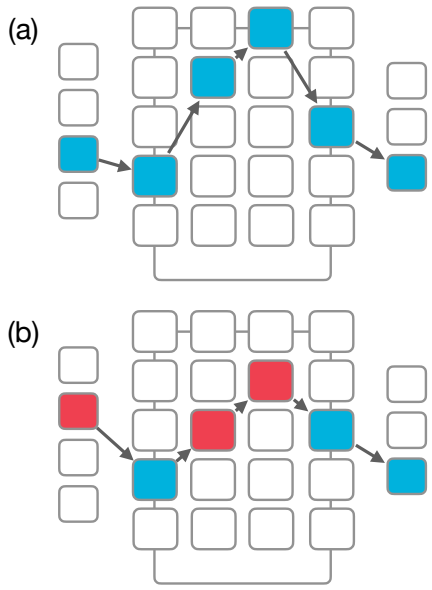

FIG. 2. A representative SA move used during code optimization. Here, an initial algorithm in (a) is updated by modifying selected instructions, as highlighted by the red blocks. The new algorithm in (b) will then be assessed by evaluating the optimization function F, with the code changes accepted in the usual Metropolis acceptance/rejection scheme.

D. Program input set

As shown in Fig. 1, the input layer for our code structure comprises a set of n_i terminals, each of which provide different input workspace vectors and matrices which can be operated on by the rest of the linear code structure.

In the calculations reported below, we provide a set of $n_i = 6$ different input possibilities, as shown in Table I. These inputs comprise different choices for a workspace vector $\mathbf{y}$ and matrix $\mathbf{M}$, which are subsequently operated on by the remaining $n_c + 1$ operations in the code structure (Fig. 1). The inputs selected here are far from exhaustive, but are instead selected to be representative of the vectors and matrices which are commonly encountered in the mathematics and numerical analysis of differential equations such as Eq. 1. We simply note here that an interesting aspect for future work would be the automated identification or generation of problem-specific input sets, but this lies beyond the scope of this initial investigation.

TABLE I. Definition of input vectors and matrices defined for the $n_i = 6$ input terminals used in PS simulations to solve Eq. 1. Here, $\mathbf{V}$ is the PES evaluted on the input coordinate grid, $\mathbf{x}$ contains the coordinate grid itself, and Δx is the coordinate grid spacing.

Input index	**Input vector**	**Input matrix**
1	$\mathbf{y} = \mathbf{1}$	$\mathbf{M} = \mathbf{I}$
2	$\mathbf{y} = \mathbf{1}$	$M_{ij} = 1$, for all i, j
3	$\mathbf{y} = \mathbf{V}$	$\mathbf{M} = \mathbf{I}$
4	$\mathbf{y} = \mathbf{x}$	$\mathbf{M} = \mathbf{I}$
5	$\mathbf{y} = \mathbf{1}$	$M_{ij} = \pm\Delta x$, if $i = j \pm 1$, $M_{ij} = 0$ otherwise
6	$\mathbf{y} = \mathbf{V}$	$M_{ij} = \pm\Delta x$, if $i = j \pm 1$, $M_{ij} = 0$ otherwise

E. Function set

The internal function set used in our PS simulations is shown in Table II. In each of the n_c columns (or code-lines) of our PS approach, the same n_r functions are available (Fig. 1); considering all possible combinations of constants and operations, there are a total of $n_r = 97$ internal functional possibilities at each of the n_c columns. These functions operate on the workspace vector and matrix which are initially defined in the input layer, performing operations and passing on the resulting vector and matrix to the next code-layer.

We note that most functions operate on either the workspace vector or matrix, with one or two exceptions performing operations on both, such as matrix-vector multiplication. Our PS code is written in *python* which dramatically simplifies the coding of the set of internal functions; furthermore, we emphasize that the operations given in Table II generally operate in an element-wise fashion.

The function set shown in Table II span a broad range of operations, range from simple scaling or shifting of vector and matrix elements, to more complex operations such as matrix diagonalization. The choice of this initial function set is somewhat arbitrary, but the general aim in selecting the allowed functions was to provide mathematical flexibility while avoiding the definition of overly-complex transformations; as such, the majority of the defined functions represent just one or two lines of *python* code. We note that there is, of course, enormous potential in exploring new functions and transformations in our PS approach; for example, there is the possibility of adopting approaches using automatically defined functions (ADFs), a well-known strategy for developing modular and re-useable computer-generated codes which has been employed extensively in the context of GP.[59,60]

As well as the functions listed in Table II, our code structure also contains a set of global constants and vectors which are available to all internal functions, as appropriate depending on the function type (Fig. 1). As in standard GP approaches, the constants can be defined in a few different ways, most notably as a set of random floating-point or integer numbers, or by defining a fixed library of constants. For simplicity we take the latter approach here, with the constants available to the internal function set being $[m, \Delta x, 2, 3, \pi, 4]$. In other words, our constant set contains a mixture of problem-dependent definitions, namely the mass m and the grid-spacing Δx, as well as a small number of numerical constants. Given that it is not clear, from the outset, exactly how these constants should be chosen in a more general fashion, we leave that problem

for future work and optimization, but note that this small set of constants is sufficient to enable discovery of interesting new algorithms for solving Eq. 1, as shown in Section III. In addition, the vectors containing the coordinate grid **x** and the PES evaluated on the grid **V** are also provided as accessible inputs to all internal functions (although we note that many internal functions do not actually require these).

F. Program output

After selection of an input vector and matrix workspace, and application of the sequence of n_c internal functions, the final operation required of any proposed code is to output an appropriate vector **w**; this vector is required to be the same size as the input vector and matrices, as defined by the grid-size discussed above. As such, the final output function set (Fig. 1) must exclusively contain functions which result in vector outputs. As in the case of the input set and the internal function set, the current choice of output functions is somewhat arbitrary, and judged predominantly by the sorts of operations which are often observed in the mathematics of differential equations of the form of Eq. 1. In this proof-of-concept work, we use the $n_o = 6$ functions shown in Table III as the possible output set; as we discuss further below, the presence of the calculation of matrix eigenvectors proves to be the "easy way out" for our computational PS approach, providing a clear route to generating novel solutions satisfying Eq. 1. Furthermore, we also note that the final vector approximation to the target wavefunction is normalized before calculation of the cost function F.

A final important point concerns the handling of errors in the code outputs. During the course of our SA optimizations, we find that errors can be generated by proposed codes, particularly in the early stages of optimization. Closer study shows that many of these errors arise from numerical problems caused by operations on incompatible vectors or matrices. For example, attempting to take the exponential of vector elements which already have very large numerical values as a result of previous operations will inevitably lead to overflow problems, ultimately resulting in the output vector adopting nonsensical values. In such cases, we simply reject such error-producing codes immediately during SA optimization.

TABLE II. Internal functions set used in code generation to solve Eq. 1. Here, $\mathbf{y}$ indicates the current workspace vector, $\mathbf{M}$ indicates the workspace matrix, and Δx is the uniform grid-spacing in the wavefunction representation. Explicit elements of the vector and matrix are written y_i and M_{ij} respectively, with M_{ii} representing the matrix diagonal elements. Except where explicitly indicated, all operations act in an element-wise manner on all entries in the workspace vector or matrix. We note that DCT and DCT^{-1} indicate the discrete cosine transform and its inverse, respectively. Considering all combinations of constants, $c = [m, \Delta x, 2, 3, \pi, 4]$, and functions, we have a total of $n_r = 97$ possible operations at each internal code line.

Operation(s)	**Operation(s)**
$\mathbf{y} \to \mathbf{y} \times c$	$M_{ii} \to M_{ii} + c$
$\mathbf{y} \to \mathbf{y} + c$	$M_{ii} \to M_{ii} - c$
$\mathbf{y} \to \mathbf{y} - c$	$M_{ii} \to M_{ii} \times c$
$\mathbf{y} \to \mathbf{y}/c$	$M_{ii} \to M_{ii}/c$
$\mathbf{y} \to -\mathbf{y}$	$\mathbf{y} \to \frac{d\mathbf{y}}{dx}$
$M_{ij} \to \frac{M_{ij}}{(i-j)}, i \neq j$	$\mathbf{y} \to \mathrm{DCT}[\mathbf{y}]$
$M_{ij} \to M_{ij} \times (i-j), i \neq j$	$\mathbf{y} \to \mathrm{DCT}^{-1}[\mathbf{y}]$
$\mathbf{y} \to \mathbf{y}, \mathbf{M} \to \mathbf{M}$	$\mathbf{y} \to \mathbf{My}$
$M_{ii} \to M_{ii} \times y_i$	$\mathbf{y} \to \mathbf{MMy}$
$M_{ii} \to M_{ii} + y_i$	$\mathbf{y} \to \sin(\mathbf{y})$
$M_{ii} \to M_{ii} - y_i$	$\mathbf{M} \to \mathbf{MM}$
$M_{ii} \to M_{ii}/y_i$	$y_i \to M_{ii}$
$M_{ii} \to M_{ii} \times V_i$	$\mathbf{y} \to \cos(\mathbf{y})$
$M_{ii} \to M_{ii} + V_i$	$M_{ij} \to \sin(M_{ij})$
$M_{ij} \to M_{ij} + c$	$M_{ij} \to \cos(M_{ij})$
$M_{ij} \to M_{ij} - c$	$y_i \to y_i \cos(i\Delta x)$
$M_{ij} \to M_{ij} \times c$	$y_i \to y_i \sin(i\Delta x)$
$M_{ij} \to M_{ij}/c$	$M_{ij} \to M_{ij} \times e^{[i\Delta x - j\Delta x]^2}$
$M_{ij} \to e^{M_{ij}}$	$M_{ij} \to M_{ij} \times e^{-[i\Delta x - j\Delta x]^2}$
$\mathbf{M} \to -\mathbf{M}$	$\mathbf{y} \to \frac{d^2\mathbf{y}}{dx^2}$

TABLE III. Definition of output vector operations defined for the $n_o = 6$ output terminals used in PS simulations to solve Eq. 1. Here, **M** is the final workspace matrix generated after the n_c internal operations, and **y** is the workspace vector. Note that operation 2 defines calculation of the eigenvector of **M** corresponding to the lowest eigenvalue, such that **U** contains the corresponding eigenvectors.

Output index	**Operation**
1	$\mathbf{w} = \mathbf{y}$
2	$w_i = U_{i1}$, where $\lambda = \mathbf{U}^T\mathbf{M}\mathbf{U}$
3	$w_i = M_{ii}$
4	$\mathbf{w} = \mathbf{M}\mathbf{y}$
5	$w_i = M_{1i}$
6	$w_i = M_{i1}$

III. APPLICATIONS, RESULTS AND DISCUSSION

The PS system described above was used to seek out algorithms which could identify the ground-state wavefunction of Eq. 1 for bound polynomial PESs given by Eq. 2. SA optimizations were performed using $M = 10$ examples of correct ground-state wavefunctions calculated for randomly generated PESs, as described in Section II A. The exact ground-state wavefunctions were obtained using the Colbert-Miller DVR method;[58] all calculations reported below used a grid-size $n_g = 101$. The Colbert-Miller DVR method serves as our oracle code, providing a set of correct solutions which can be used to judge the performance of any automatically-generated algorithm according to Eq. 3. All SA optimizations were performed for a total of 10^4 iterations.

To test the impact of code size, we performed initial simulations for four different algorithm sizes, $N = n_c + 2$ = 7, 10, 15 and 20. For each code-size, we performed 50 independent SA optimizations for a set of $M = 10$ randomly-generated PESs; as a further assessment of each final generated algorithm, we also calculate F for an additional 10^3 random PESs at the end of each calculation, providing an independent check of algorithm performance against problems which did not explicitly appear in the target data.

In the following, we give an overview of the code generation calculations, before giving examples of successful auto-generated codes.

A. Overview of code generation

Figure 3 shows the evolution of the error function F for the full set of SA optimizations performed for $N = 10$, 15 and 20. From visual inspection of solutions (see below), we have found that algorithms with $F < 0.1$ typically give good approximations to ground-state wavefunctions for most problems within the target set, and within the set of 10^3 randomly-generated PESs (created as an independent test, as noted above). Notably, we found that no successful codes, as judged by the final values of F, were found for $N = 7$; it appears that this number of code instructions is too small to develop a sufficiently accurate approximation of the target wavefunctions using the defined function sets. As such, $N = 7$ results will not be discussed further.

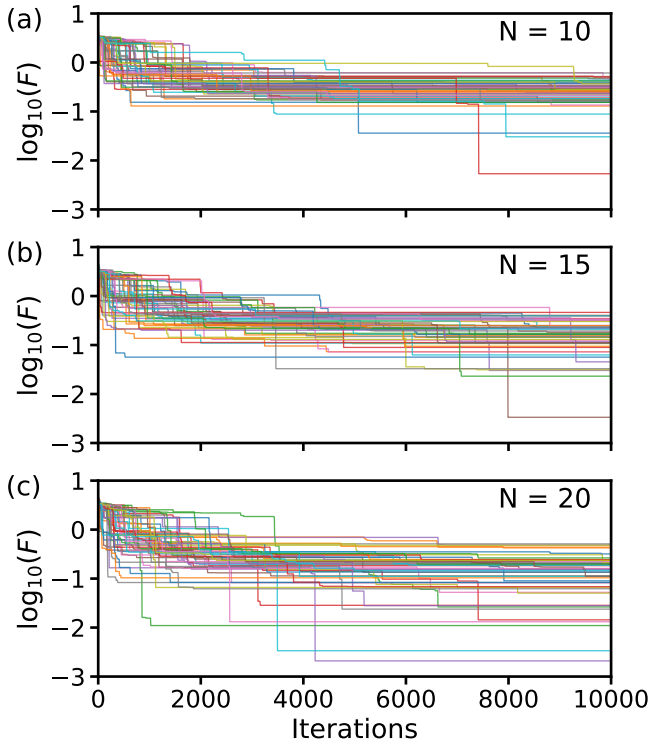

FIG. 3. Progression of error function F during 50 independent SA optimizations for different code-sizes N; note the logarithmic scale on the vertical axis.

As shown in Fig. 3, there is a clear progression in the ability of our SA optimization scheme to find sufficiently accurate codes as the code-size N increases. For $N = 10$, four of the 50 indepen-

dent simulations determined a code with $F < 0.1$, whereas the number of successes increases to 13 for $N = 15$ and 18 for $N = 20$. Although a detailed analysis of this behaviour is out of the scope of this Article, it is clear that the increased flexibility is a key factor; in general, this behaviour seems analogous to the increased flexibility provided by higher-order polynomials in regression tasks.

It is also worth noting that many of the calculations for all values of N are found to "stagnate", with little significant decrease in F as further SA iterations proceed; this behaviour suggests the presence of multiple local minima in the error landscape described by F, indicating that the optimization procedure employed here could certainly be improved. It is also worth emphasizing that a standard "brute-force" assessment of all possible codes is not possible due to the combinatorially large number of possible algorithms which can be constructed within out code set-up (Fig. 1). For example, for $N = 10$, there are around 10^{21} possible algorithms which can be constructed from the instruction sets used in our simulations; it is clearly not feasible to assess the performance of all algorithms directly. So, the improvement of our optimization scheme is an area for future work; that said, the current methodology is sufficiently successful to give a number of different proposed algorithms, as discussed below.

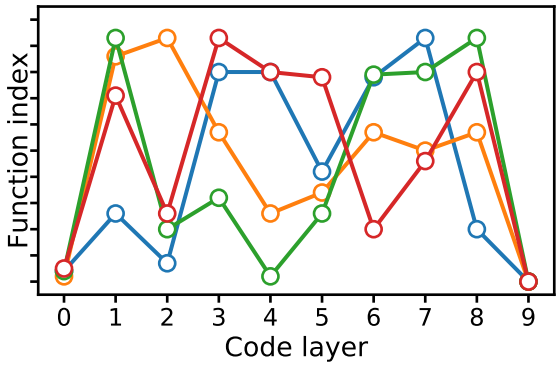

FIG. 4. Function indices at each instruction step of the four codes with $F < 0.1$ (for 10^3 random PESs) for algorithms of length $N = 10$. Here, each function in the input, internal and output functions sets has been assigned an arbitrary index for visualization.

Figure 4 shows the instruction sequences generated by our optimization approach for $N = 10$; each of the illustrated sequences has $F < 0.1$ for 10^3 randomly-generated PESs. This Figure demonstrates that a range of algorithms are generated by our SA method, with different input and internal instruction sets; however, it is noticeable that they all exhibit the same final output instruction. This aspect of our approach is discussed further below in Section III B. However, for

now, we simply note that the SA algorithm is successful in generating a variety of algorithms for further study, several of which are explored below.

B. $N = 10$ codes

We begin by discussing codes with $N = 10$ instructions; these codes contain an input definition, eight internal instructions, and an output function. We introduce the nomenclature C_N(X) when referring to different codes, where the subscript denotes the total code-length and the letter in the parentheses labels different generated codes.

A study of the four $N = 10$ codes which were flagged as having 'good' performance reveals that they are all very similar in their instruction sets, with minor variations in the constants being used or the order of operations. As such, we focus here on one specific example code, referred to as C_{10}(A); this code is shown in Fig. 5, and its performance in reproducing the correct ground-state wavefunctions for three randomly-generated PESs is highlighted in Fig. 6. The agreement with the target wavefunctions is clearly excellent, as also suggested by the fact that this algorithm had an average F value of 3.7×10^{-3} for 10^3 randomly-generated PES. It is also noticeable that the performance of the algorithm in reproducing the ground-state wavefunctions in Fig. 6 is the same regardless of the position of the PES minimum, the PES asymmetry or the fact that more than one minima is present. As a further comment, we emphasize that only $M = 10$ PES examples were used during SA code optimization; despite this, the resulting successful codes can readily generalize to give accurate wavefunctions for 10^3 further random PESs.

An important observation about the high-performing $N = 10$ algorithms which were generated in our SA optimizations is that they all employ calculation of the first eigenvector (with smallest eigenvalue) of the operating matrix **M** as the output instruction (See Table III). As a result, it is clear that these algorithms successfully identify the underlying mathematical challenge as an eigenproblem, and the input and internal instructions are providing approximations to the well-known Hamiltonian matrix. Furthermore, because the underlying problem representation relies on a grid-based representation similar to that employed in DVR schemes, it is clear that the successful algorithms are providing accurate Hamiltonian matrix approximations without explicit definition of, or integration over, an underlying finite basis set.[52,53]

To be explicit, it is well known that one can express the solutions of Eq. 1 as a linear combina-

```
import numpy as np
n = len(V)
M[:,:] = 0
for i in range(n):
    if i != n - 1:
        M[i, i + 1] = dx
    if i > 0:
        M[i, i - 1] = -dx
for i in range(n):
    for j in range(n):
        if i != j:
            M[i, j] *= 1.0 / (i - j)
M *= 4.0
for i in range(n):
    M[i,i] += V[i]
    M[i,i] /= np.pi
    M[i,i] /= 4.0
    M[i,i] *= dx
E, c = np.linalg.eig(M)
SortAscending(E,c)
w[:] = c[:, 0]
```

FIG. 5. Algorithm C_{10}(A) expressed in python. Here n is the number of grid-points, the vector $\mathbf{V}$ contains the PES evaluated on the grid and `dx` is the grid spacing. The `numpy` pacakge is used, providing the constant π as `np.pi`. The function `np.linalg.eig` calculations the eigenvectors (`c`) and eigenvalues (`E`) of $\mathbf{M}$, whereas `SortAscending()` sorts the eigenvectors in ascending order of eigenvalues.

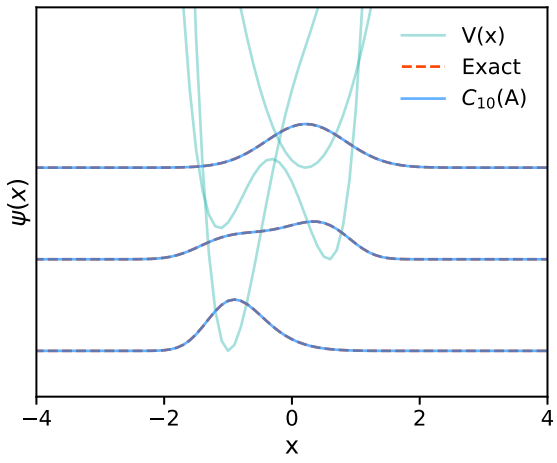

FIG. 6. Comparison of exact ground-state wavefunctions (provided by Colbert-Miller DVR method) and approximate wavefunctions given by algorithm C_{10}(A) for three random PESs.

tion of basis functions,

$$\psi_i(x) = \sum_{j=1}^{n} c_{ij}\phi_j(x), \tag{4}$$

where c_{ij} are expansion coefficients to be determine and $\phi_j(x)$ are the basis functions. Using

this approximation in Eq. 1, along with the variational principle, leads to the well-known matrix formulation of the TISE, namely

$$\mathbf{Hc} = \mathbf{ScE}, \tag{5}$$

where $\mathbf{c}$ is the matrix of unknown coefficients for each of the n solutions, $\mathbf{E}$ is a diagonal martix of corresponding eigenvalues, $\mathbf{S}$ is the overlap matrix for the basis functions (with $S_{ij} = \langle \phi_i | \phi_j \rangle$), and $\mathbf{H}$ is the Hamiltonian matrix,

$$H_{ij} = \langle \phi_i | \hat{H} | \phi_j \rangle, \tag{6}$$

where we have defined the Hamiltonian operator from Eq. 1 as

$$\hat{H} = -\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x). \tag{7}$$

In the typical DVR method, the evaluation of the Hamiltonian matrix elements are greatly simplified by transformation into a localized grid-basis, in which the potential energy operator is diagonal and the kinetic energy matrix elements can be evaluated in an underlying basis.[55,58] In this vein, the Colbert-Miller DVR method employed as the exact oracle code here, has matrix elements given by,

$$H_{ij} \equiv \begin{cases} \frac{\hbar^2 \pi^2 (-1)^{(i-j)}}{6m\Delta x^2} + V_i & \text{for } i = j \\ \\ \frac{\hbar^2 (-1)^{(i-j)}}{m\Delta x^2 (i-j)^2} & \text{for } i \neq j, \end{cases} \tag{8}$$

where the kinetic energy contribution emerges using a Fourier basis.

By analogy, we see that the generated algorithms for $N = 10$, including that shown in Fig. 5, are seeking to solve the target problem by diagonalizing the operating matrix $\mathbf{M}$; in other words, our generated codes are treating $\mathbf{M}$ as a DVR-type approximation of the Hamiltonian matrix $\mathbf{H}$, under the assumptions that the underlying basis set is orthonormal (such that the overlap matrix $\mathbf{S}$ is ignored) and the total number of basis functions is n_g (*i.e.* the size of the uniform grid and the dimension of the workspace matrix $\mathbf{M}$). It is important to note that the mathematical development given above was not hard-coded in our PS approach, but we clearly provided an output function corresponding to calculation of the first eigenvector of $\mathbf{M}$.

Based on the comments above, it is then informative to investigate exactly how a typical $N = 10$ code approximates $\mathbf{H}$. From Fig. 5, we find that the Hamiltonian matrix elements are approxi-

mated as

$$H_{ij} \equiv \begin{cases} \frac{V_i \Delta x}{4\pi} & \text{for } i = j \\ -4\Delta x & \text{for } j = i+1 \\ 4\Delta x & \text{for } j = i-1 \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

From this last equation, it is evident that the resulting Hamiltonian matrix approximation discovered in C_{10}(A) is tridiagonal, thereby offering efficiencies in computation through use of sparse matrix manipulations when compared to the Colbert-Miller Hamiltonian matrix. Upon closer inspection, and accounting for differences in the constants present, Eq. 9 is comparable to a finite-difference-type approximation resulting from discretization of the second-derivative operator on the uniform grid representation of the problem,

$$\frac{d^2\psi(x_i)}{dx^2} \simeq \frac{\psi(x_{i+1}) - 2\psi(x_i) - \psi(x_{i-1})}{\Delta x^2}, \tag{10}$$

combined with the usual diagonal potential energy term resulting in DVR approximations. However, we note that the choice of constants delivered by the optimization appears to be tied to the specific number of grid-points, n_g, used in these simulations; changing n_g means that the set of constants in Eq. 9 is no longer appropriate, although we have found that rescaling either the length of the uniform grid (such that Δx remains constant), or rescaling the Hamiltonian matrix elements themselves by a scale factor related to the change in n_g, enables one to recover accurate results. So, it is pleasing that our automated algorithm searching approach has successfully constructed codes which mimic common solution approaches for the TISE, although the choice of constants tying our solution to a specific grid-length is not desirable (but is solvable; see Section V). In the most general interpretation, we see from Fig. 5 and Eq. 9, that our PS approximation is providing an *effective Hamiltonian* approximation for which the ground-state closely approximates the true ground-state of a given PES.

To summarize, the $N = 10$ simulations have resulted several different algorithms proposed as solutions of Eq. 1; these generally fall into the category of Hamiltonian matrix approximations given in Fig. 9, using finite-difference approximation schemes. Next, we discuss $N = 15$ codes,

where some familiar algorithms emerge again, but also interesting new schemes which cannot be immediately tied to known methods.

C. $N = 15$ codes

Increasing the number of instructions available, we find that the success of our SA optimization scheme improves to around 25%, with 13 of the 50 SA simulations resulting in $F < 0.1$ for 10 target PESs (and also commonly for a further 10^3 random PESs). All 13 successful simulations used eigenvector calculation of the workspace matrix **M** as the output function to give the wavefunction approximation vector **w**; as such, all algorithms can again be viewed as DVR-type approximations to the Hamiltonian matrix, as described above.

Of the 13 successful SA simulations for $N = 15$, we find that 11 codes are essentially the same as C_{10}(A) (Fig. 9), give or take overall constants which scale the resulting eigenvalues but leave the target wavefunctions unchanged after normalization. However, the two remaining codes for $N = 15$ did not employ the standard finite-difference input matrix, and instead represent new approximations to the Hamiltonian matrix.

Pseudo-codes for these new Hamiltonian matrix approximations are shown in the Appendix. The performance of C_{15}(A) in reproducing the ground-state wavefunction for three random PESs is illustrated in Fig. 7; the performance observed for C_{15}(B) is very similar, and not shown here. In both cases, we find that the wavefunction approximations are very good, with errors for 10^3 random PESs of $F = 0.098$ for C_{15}(A) and $F = 0.07$ for C_{15}(B).

For C_{15}(A), the resulting Hamiltonian approximation is found to be:

$$H_{ij} \equiv \begin{cases} a + V_i & \text{for } i = j \\ \\ \frac{b e^{-(i\Delta x - j\Delta x)^2}}{(i-j)^2} & \text{for } i \neq j, \end{cases} \tag{11}$$

with $a = -\left(\frac{5}{4\pi} + 1\right)$ and $b = \frac{3}{4} - 6 - \pi$. For algorithm C_{15}(B), the Hamiltonian approximation is:

$$H_{ij} \equiv \begin{cases} \sin\left(V_i \Delta x + 4 - 2m\right) + V_i & \text{for } i = j \\ \\ \sin\left(e^{(i\Delta x - j\Delta x)^2} + 4\right) e^{-(i\Delta x - j\Delta x)^2} & \text{for } i \neq j, \end{cases} \tag{12}$$

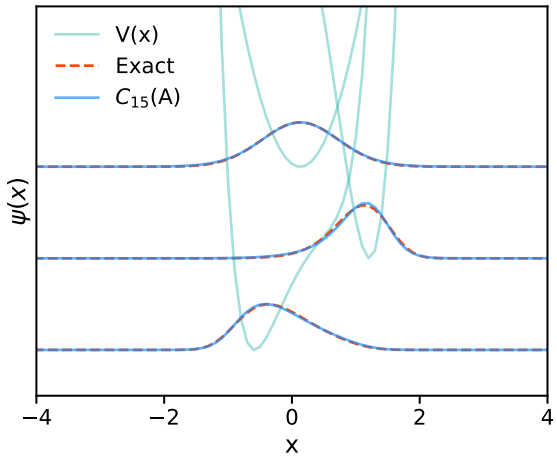

FIG. 7. Comparison of exact ground-state wavefunctions (provided by Colbert-Miller DVR method) and approximate wavefunctions given by algorithm C_{15}(A) for three random PESs.

Clearly, neither of these algorithms uses the finite-difference matrix as input; in fact, both algorithms simply use the unit vector and a matrix $H_{ij} = 1$ (for $i, j \in n_g$) as inputs. As might be expected based on the underlying uniform coordinate grid, the potential energy appears in the diagonal elements, such that the off-diagonals can be viewed as approximations to kinetic energy operator matrix elements. In C_{15}(A), the off-diagonal approximation somewhat resembles that used in the Colbert-Miller DVR method (Eq. 8). The clear difference lies in the numerator, which possesses an oscillating factor in Eq. 8 which is not included in the purely positive numerator of Eq. 11. In contrast, Eq. 12 is quite unfamiliar, combining both sine functions and exponentially-decaying functions; there is little overlap with the Colbert-Miller DVR matrix elements, other than the broad feature of the matrix elements decaying away as the relative difference between i and j increases.

In summary, the results for $N = 15$ demonstrate that it is indeed possible to discover novel algorithms which yield good approximations to the correct ground-state wavefunction if one treats the PS workspace matrix as a Hamiltonian approximation associated with an (unknown) underlying basis.

D. $N = 20$ codes

Increasing the number of instructions further still, to $N = 20$, we find that the success of our SA optimization scheme improves to around 36%. As in the case of the $N = 15$ simulations, the large majority of the successful $N = 20$ codes use the finite-difference input matrix as the

foundation of a Hamiltonian matrix approximation; as such, many of the generated codes are similar to C_{10}(A), with variations in the constants which, ultimately, do not make any difference to the output eigenvectors of the operating matrix after normalization.

However, two particular $N = 20$ codes stand out as being quite unique compared to finite-difference-based schemes; these algorithms (C_{20}(A) and C_{20}(B)) are shown in the Appendix. As in previous successful algorithms, both C_{20}(A) and C_{20}(B) use the output eigenvector operation, such that they correspond to further approximations to the Hamiltonian matrix. In C_{20}(A), the Hamiltonian matrix approximation is:

$$H_{ij} \equiv \begin{cases} \frac{aV_i}{4\pi} & \text{for } i = j \\ \\ \frac{\Delta x}{\pi} e^{-2(i\Delta x - j\Delta x)^2} - \frac{m}{\pi} e^{-3(i\Delta x - j\Delta x)^2} + \frac{\Delta x}{\pi} e^{-(i\Delta x - j\Delta x)^2} & \text{for } i \neq j, \end{cases} \tag{13}$$

where $a = ((3-m)\Delta x - m)\, m + 2\Delta x + \pi$. In contrast, the Hamiltonian matrix approximation in C_{20}(B) is

$$H_{ij} = 2\left(\sum_{k=1}^{n_g} \tilde{H}_{ik}\tilde{H}_{kj}\right) e^{-(i\Delta x - j\Delta x)^2} + \delta_{ij}V_i, \tag{14}$$

where the intermediate matrix elements $\tilde{H}_{ij}$ are given by

$$\tilde{H}_{ij} \equiv \begin{cases} 2\left(\frac{-[\cos(2)-\Delta x]+\pi+m}{\pi}\right) + \Delta x + 4 & \text{for } i = j \\ \\ -\frac{2(1-\Delta x)}{(i-j)^2} & \text{for } i \neq j. \end{cases} \tag{15}$$

These two successful Hamiltonian matrix approximations are both obviously very different, yet the performance of both algorithms is found to be similarly good for 10^3 random $V(x)$. The fact that our PS approach has identified these codes, without falling back on the standard finite-difference approximation, is very encouraging in confirming that PS can be used in the setting of quantum chemistry to drive exploration of alternative algorithms.

E. Examining new Hamiltonian approximations in more detail

At this point, it is interesting to investigate what the different Hamiltonian approximations generated in our simulations looks like, and how they compare to previous work. Figure 8 compares the Hamiltonian matrices (for the simple harmonic PES $V(x) = \frac{1}{2}x^2$) given by the Colbert-Miller

DVR method and algorithms C_{15}(B), C_{20}(A) and C_{20}(A). Here, we show the elements of the approximated Hamiltonian matrix along a single row, specifically $H_{51,i}$, as well as showing a representation of the magnitude of elements in the full matrix.

Looking at the one-dimensional slices through the Hamiltonian matrices, we see that all of the Hamiltonian matrix approximations are localized around the diagonal element $H_{51,51}$, as might have been expected based on the fact that all methods use an underlying problem representation on a uniform coordinate grid on which the potential energy operator is diagonal. However, it is clear that there are notable differences amongst these approximations. For example, the Colbert-Miller DVR Hamiltonian is highly oscillatory as a result of the appearance of the $(-1)^{(i-j)}$ factor in Eq. 8; in contrast, the Hamiltonian from C_{20}(B) is very similar to the Colbert-Miller result, but without the oscillatory nature. Furthermore, the Hamiltonian approximations of C_{15}(B) and C_{20}(A) seem to be much more delocalized in nature, with C_{15}(B) also demonstrating oscillations. These features can also be clearly observed in the contour plots showing the full Hamiltonian matrices, which serves to further emphasise the differences in the sparsity and structure in the different matrix approximations.

In the current context, where we are simply exploring whether or not the proposed PS strategy can generate new algorithms for solving Eq. 1, the features of the Hamiltonian matrix are somewhat secondary considerations when compared to overall accuracy or performance. However, in envisioned next steps here, one might be interested in biasing the PS search towards algorithms with particular properties, such as sparsity; such algorithms might obviously be numerically or computationally advantageous. This could in principle be achieved by modifying the cost-function F, guiding the search towards algorithms with particular desired numerical features. In this Article, we simply note the fact that a range of different algorithms can be generated by the simple linear PS scheme (and simple function sets) used here, which offers hope of further tuning and expanding these algorithms in the future.

IV. COMPARISON TO PREVIOUS GP AND ML WORK

As a final point, it is interesting to compare our PS approach to previous work using GP. As noted above, GP has been used in earlier work to find functional approximations for ground-state wavefunctions for one-dimensional PESs. For example, Makarov and Metiu used GP,[50] in combination with a fitness function based on Eq. 1, to identify an approximate ground-state

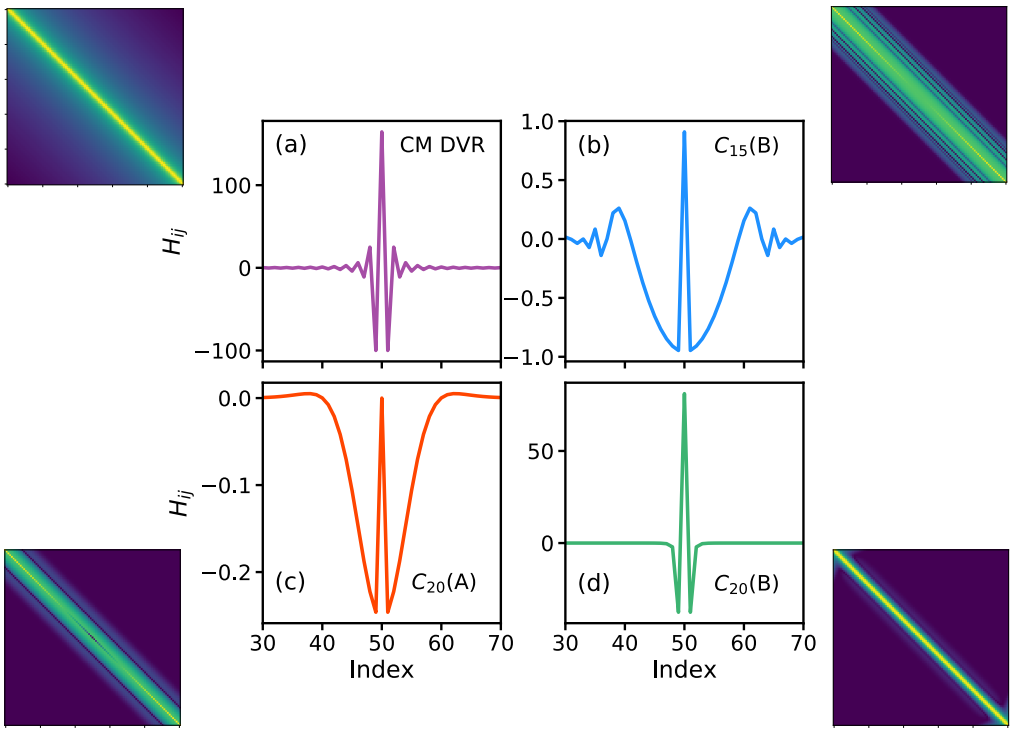

FIG. 8. Hamiltonian matrix approximations along a single row, $H_{51,i}$, for four different methods for solving Eq. 1, namely (a) Colbert-Miller DVR, (b) C_{15}(B), (C) C_{20}(A), and (d) C_{20}(B). The outer panels represent the full Hamiltonian matrix approximations; for ease of comparison of the different matrices, we plot the function $f_{ij} = \log(10^{-6} + H_{ij}^2)$.

wavefunction for the one-dimensional Eckart well, namely:

$$V(x) = -\frac{2}{[\cosh(\frac{x}{2})]^2}. \tag{16}$$

Starting from a trial wavefunction of the form

$$\psi(x) = \frac{1}{\cosh(\chi(x))}, \tag{17}$$

and using GP to evolve an appropriate function $\chi(x)$, an optimal, un-normalized wavefunction approximation was found to be given with the function,

$$\chi(x) = 1.086x + 0.023x(-1.78 + 1.086x)(1.658 + 1.086x). \tag{18}$$

Note that there are six free floating-point parameters used in this wavefunction approximation, whereas the PS strategy explored in this paper does not optimize any such free parameters.

The important point here is that the wavefunction of Eq. 18 was evolved to specifically solve the Schrödinger equation for the PES given in Eq. 16. In contrast, the PS code-generation strategy we have explored above instead generates complete *algorithms* which are not tied to any particular PES, but are instead capable of giving good wavefunction approximations for a wide variety of PESs. As such, there is a clear difference in emphasis of our PS approach and previous GP-based function approximation strategies.

This difference is further emphasized in Fig. 9. Here, we show the wavefunction approximation from the previous GP investigation (Eq. 18), the exact solution given by the Colbert-Miller DVR method, and the wavefunction approximation given by algorithm C_{20}(B). We note that algorithm C_{20}(B) did not include the PES of Eq. 16 in its target example data during optimization; despite this, it can be seen in Fig. 9 that C_{20}(B) gives an excellent reproduction of the exact wavefunction. In fact, even though the previous GP work was *specifically* targeted to reproduce $\psi(x)$ for the specific Eckart well potential of Eq. 16, we find that the performance of C_{20}(B) is better, as is particularly noticeable in the peak around $x = 0$ and in the wing regions around $x = \pm 1$.

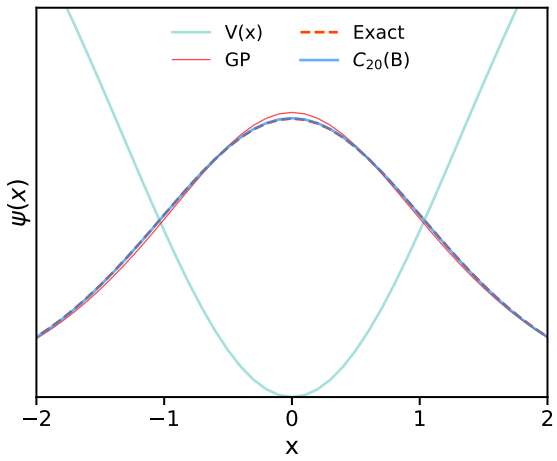

FIG. 9. Comparison of exact ground-state wavefunctions (provided by Colbert-Miller DVR method) and approximate wavefunctions given by algorithm C_{20}(B) or previous GP work for the Eckart well PES of Eq. 16.

As a final comment, we note that ML methods, most notably ANNs, have been explored as routes to generating wavefunctions for different model and molecular systems.[61–64] For example, ANNs have been used to represent wavefunctions for the vibrational Schrödinger equation for Morse potentials, harmonic oscillators, anharmonic oscillators and double-well PESs,[62–64] while

Manzhos and coworkers[61] expanded such methods to real molecular systems, using an ANN with radial basis functions to determine the first few vibrational wavefunctions for H_2O to an accuracy of 2 cm^{-1}. However, it is again worth emphasizing that the PS strategy explored in the proof-of-principle study here is fundamentally different from these previous ANN-based investigations, focussing on generation of code *via* a discrete optimization process, rather than optimization of ANN weights. These two strategies are clearly complementary, and exploring the potential of PS in more general quantum chemical applications is certainly an avenue for further investigation.

V. LIMITATIONS AND EXTENSIONS

The results above have demonstrated that PS can be used to successfully generate novel algorithms which meet the target criteria for this study, namely reproducing the ground-states of the 1-D Schrödinger equation for bound, polynomial PESs. This result is broadly encouraging as an initial proof-of-concept, but it is worth highlighting a number of current limitations, as well as possible extensions, of the PS strategy adopted here.

1. Multidimensional systems

First, the system representation employed here, namely defining the PES and $\psi(x)$ on a pre-defined uniform grid, is clearly not optimal. In particular, as is well-known, direct extension of a uniform grid to higher-dimensional systems rapidly leads to an unmanageable number of DVR grid-points; in other words, the DVR-type schemes explored above do not address the poor scaling of computational effort with dimensionality which is common to grid-based simulation strategies.

However, it is worth noting that the DVR-type algorithms generated by our PS strategy can be readily extended to higher-dimensions, just as standard DVR schemes can be employed in multidimensional calculations. For example, for a 2-D system, DVR Hamiltonian matrix elements can be written as:

$$H_{ij,km} = T_{ik}(x)\delta_{jm} + T_{jm}(y)\delta_{ik} + V(x_i, y_j)\delta_{ik}\delta_{jm}, \tag{19}$$

where (i, j) and (k, m) label grid-points along the two distinct degrees-of-freedom, (x, y). Based on the comparison to DVR schemes described above, it is then clear that the PS-generated codes described here can equally be used in constructing multidimensional Hamiltonian matrices such as

Eq. 19. This is verified in Fig. 10, which shows the calculated and exact ground-state wavefunctions for a representative 2-D PES; in this case, the 2-D ground-state wavefunction was predicted using the 2-D generalization (Eq. 19) of algorithm $C_{15}(A)$, and it is found that agreement with the expected ground-state given by the Colbert-Miller DVR is comparable to that observed for 1-D PESs.

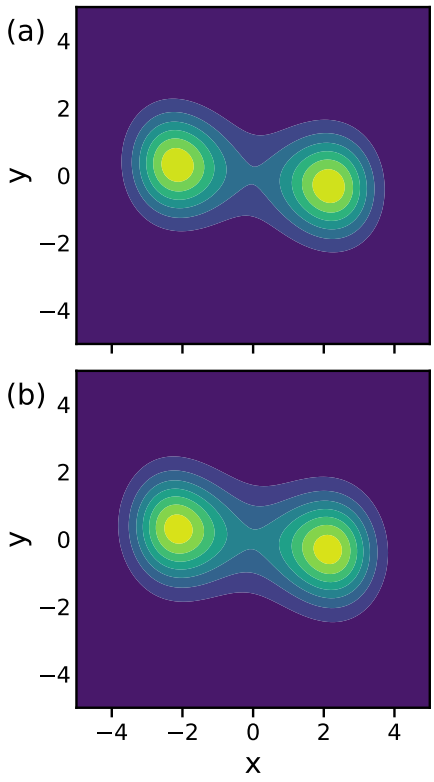

FIG. 10. Application of algorithm $C_{15}(A)$ to generate the ground-state wavefunction of a 2-D PES. Here, the PES is a double-well along the x-coordinate, linearly coupled to a harmonic oscillator in the y-coordinate; specifically, $V(x,y) = 0.046x^4 - \frac{x^2}{2} + \frac{y^2}{2} + 0.15xy$. Panel (a) shows results of standard Colbert-Miller DVR calculation, and panel (b) shows results from algorithm $C_{15}(A)$, adapted to a 2-D problem using Eq. 19. The grid-size was $n_g = 101$ along both degrees-of-freedom.

While speculative at this early stage of development, the initial PS strategy suggested here offers some possibilities in addressing the challenge of scaling with dimensionality in DVR schemes. For example, one modification would be to change the optimization cost-function (Eq. 3) to favour algorithms which reproduce fully-converged target wavefunctions (or other properties; see below) using small grid-sizes; such a strategy could help reduce the number of required DVR grid-points

per degree-of-freedom. A more ambitious strategy would be to seek to use PS to generate *both* grid-point positions *and* Hamiltonian approximations simultaneously; this could be achieved by using a tandem PS procedure, with the first stage focussing on developing optimal grid-positions (for example, given as a PS approximation to matrix elements of position operators, but without reference to an underlying basis set) and the second stage using these grid-positions to develop Hamiltonian matrix approximations. Indeed, preliminary simulations[65] indicate that this tandem PS strategy enables good convergence of energy eigenvalues for 1-D PESs with small grid-sizes, but there is further work to be done to optimize this strategy and extend it to multidimensional PESs. To summarize - the algorithms generated by our PS strategy here inherit the same limitations as standard DVR schemes in regards to multidimensional problems, but we believe that exploring the flexibility provided by different PS strategies is worth pursuing further.

2. Predicting other properties

In the PS simulations described above, we have focussed on generating algorithms which reproduce the ground-state wavefunction for bound polynomial PESs. However, as already noted above, modification of the optimization cost-function (and to the available functions and workspace objects), offers a route to generating tailored algorithms which can predict other wavefunctions and/or observable properties. For example, Eq. 3 can be straightforwardly modified to incorporate comparison of the first n_e eigenstates for example PESs, as follows:

$$F = \frac{1}{Mn_e} \sum_{j=1}^{M} \sum_{k=1}^{n_e} |\psi_j^k - \left(S_j^k \mathbf{w}_j^k\right)|. \tag{20}$$

Here, again by analogy with DVR methods, the k-th PS-predicted eigenstate for the j-th PES example, $\mathbf{w}_j^k$, can be obtained as the corresponding eigenvector of the workspace matrix $\mathbf{M}$ and compared to the target state ψ_j^k. Indeed, initial results obtained using this optimization function are shown in Fig. 11, where we illustrate the predictions made by a code generated in a single SA run using Eq. 20 with $n_e = 3$ and $M = 10$. The resulting algorithm, C_{20}(C), is shown in Fig. 17. As shown in Fig. 11, the reproduction of the first few eigenstates for previously-unseen PESs is of similar quality to that obtained when just a single target eigenstate was used (Eq. 3).

More generally, the modification of the optimization cost-function to include other target states or properties provides a relatively simple way of guiding the PS search towards algorithms which

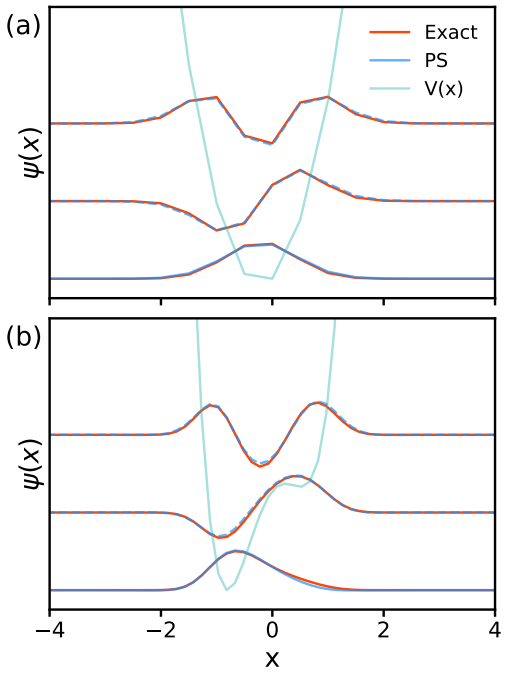

FIG. 11. Results of simulations performed using the transferrable algorithm $C_{20}(C)$ for two different randomly-generated PESs (not included in the original target examples). $C_{20}(C)$ was generated by optimizing Eq. 20 for a set of $M = 10$ target examples; in each case, $n_e = 3$ eigenfunctions were used as targets. Panel (a) shows the results of a calculation performed with $n_g = 21$ grid-points, while (b) shows a calculation with $n_g = 68$ grid-points.

obey the required symmetry properties of the target solutions; this is particularly important, for example, in reproducing nodal surfaces in higher vibrational excited states, as already highlighted in Fig. 11. As an alternative route to accounting for symmetry, we note that careful choice of the function set used during PS could also be exploited (although this is in itself a challenging problem; see below).

Finally, we emphasize that the optimization cost-function is not necessarily restricted to reproducing wavefunctions alone; other observable properties such as energy eigenvalues (as already noted above) could instead be used as targets, providing a general route to generating novel algorithms which calculate useful properties of interest.

3. Training and interpretability

As in typical applications of ANNs, inductive PS similarly requires target examples against which to judge code performance. In the present case, this is straightforward to obtain, but this will not always be the case; the example of multidimensional quantum vibrational properties, as discussed, is a good case in point. However, as described above, it is worth emphasizing that the PS approach described here is not limited to predicting ground-state wavefunctions, but can equally be applied to generate codes capable of predicting other properties and observables. For example, given existing large databases of molecules structures and DFT-calculated energies, one could anticipate using PS to generate algorithms which reproduce these target energies, in much the same way that ANNs are increasingly used to draw predictions from data. As another example, one could anticipate using PS to generate algorithms which predict the lowest-few vibrational energy eigenvalues for small-molecule systems, subsequently testing the performance against known experimental data; this approach clearly parallels the common development of empirical force-fields for complex systems based on small-molecule *ab initio* data. As described previously, we note that PS is a complementary approach to ongoing efforts in the field of ML.

On a similar note, it is worth highlighting a related aspect which is equal parts challenge and opportunity. Specifically, we have shown above that the outputs of our PS simulations can ultimately be interpreted as a set of working equations; in this regard, this approach is different from ANNs, where data patterns are usually encoded in connection weights. As such, PS offers a route to generating interpretable algorithms; however, there is no guarantee that it will be clear exactly how such algorithms are actually tackling a given problem! Ultimately, end-users of such algorithms might not be concerned with what is happening "under the hood" (as long as these algorithms are well-tested for the relevant problem domain), in a similar manner that typical ANN end-users are not usually interested in the interpretation of network weights. More broadly, we anticipate that dissecting PS-generated algorithms using function knock-out studies, as well as symbolic reduction and analysis of the generated working equations, will help to provide further insights.

Finally, we have shown above how our current PS strategy requires an input grid and corresponding grid-size, n_g; it is of course desirable to be able to generate more transferrable codes, which are not tied to a particular grid but instead work across a range of grid-sizes. As noted above, a simple way of achieving this is to modify the set of target examples to include targets with a range

of different grid-sizes; in this way, the optimization procedure should produce algorithms which work across the range of grid-sizes represented in the targets. The results shown in Fig. 11 were, in fact, generated using this approach; each of the $M = 10$ target examples was represented on a different grid-size, with n_g randomly generated for each target such that $11 < n_g \leq 71$. Figure 11(a) shows the application of a resulting algorithm C_{20}(C) (Fig. 17) to a randomly-generated polynomial PES (not originally included in the target set) with $n_g = 21$, whereas Fig. 11(b) shows application of the same algorithm to a previously-unseen PES problem with $n_g = 68$. In both cases, the reproduction of the first three wavefunctions is very good when compared to the Colbert-Miller DVR scheme. Again, these results are a simple demonstration of how modification of the targets (and the resulting optimization cost-function) enables generation of more transferrable algorithms.

4. Optimization methods and choice of functions

In the results reported here, we have used a simple SA scheme to perform algorithm optimization; it is clear, however, that there is substantial room for improvement. For example, any DVR-type scheme must incorporate the PES values in order to given sensible, problem-specific property predictions; as a result, any proposed algorithm which does not use $\mathbf{V}$ at one of the input or internal functions should be immediately rejected during optimization. As another example, any algorithm which results in numerical overflows during evaluation (for example, in seeking to take the exponent of an already-large number) can also be rejected from further consideration. Furthermore, the "linear" setup of our current PS strategy, with functions evaluated in sequence, introduces some challenges to optimization. Specifically, the choice of early-stage functions will likely impose some constraints on which of the defined internal functions are useful in generating viable algorithms; this correlation between function choices at different stages should preferably be accounted for in order to improve the success rate in generating high-performing algorithms.

Finally, we highlight the challenges associated with choosing suitable input, internal and output functions. As noted above, our initial strategy in this proof-of-concept study was to generate a large set of function possibilities which are broadly representative of quite common and generic operations; there is no guarantee that this set is in any way optimal, and there is also evidence to suggest that the presence of superfluous functions can have a negative impact on search performance and code optimization.[59] A more appropriate strategy would be to further investigate

approaches which have been developed in the context of GP; in particular, the concept of automatically defined functions (ADFs) and related strategies, in which functions are co-optimized along with the overall code during a GP calculation, offers a possible route forward.[59,60] Furthermore, a closer study of functions based on a sensitivity analysis (namely, identifying which function options commonly appear in high-performing codes) offers another way forward to be explored.

VI. CONCLUSIONS

In this Article, we have shown how a PS strategy, comparable to that used in CGP (but using a SA optimization method), can be used to generate novel algorithms which approximate the ground-state wavefunction for bound, one-dimensional PESs. We have later shown how the same system can be used to predict excited-states, and to generate transferrable algorithms which work across different grid-sizes. To the best of our knowledge, this is the first example of a PS methodology to explore new algorithms in this context.

We have given a number of different example algorithms, with total linear instruction sizes of $N = 10$, $N = 15$ and $N = 20$. In all the successful algorithms, we find that the output instruction is the evaluation of the eigenvector of the working matrix $\mathbf{M}$ with lowest eigenvalue; in other words, the matrix $\mathbf{M}$, which is generated by the operation of a sequence of functional and numerical instructions on a given input matrix, is treated as a DVR-type approximation to the Hamiltonian matrix of the system. In many cases, the algorithms which exhibit good performance against a set of random PESs are found to be essentially the same as common schemes which use a finite-difference representation of the kinetic energy operator, in combination with the usual diagonal (coordinate-space) representation of the PES operator. However, we also identified several high-performing algorithms, such as C_{20}(A) and C_{20}(B), which were novel and did not obviously correspond to previous Hamiltonian matrix approximations; for example, although these Hamiltonian matrices reproduce ground-state wavefunctions for random $V(x)$, they are not explicitly based on an underlying finite basis representation.

While this proof-of-concept PS study has successfully demonstrated capability in identifying new algorithms which may be of interest, there are clear possibilities for expansion and improvement. In Sec. V, we have given indications of where challenges remain and progress could be made. For example, we have noted that the DVR-type schemes generated by our current PS strategy inherit the same computational scaling with dimensionality as standard DVR schemes; at the

same time, we have demonstrated (Fig. 10) that our PS-generated algorithms can be generalized to multi-dimensional systems in the same way that standard DVR schemes can. We have also highlighted possibilities for decoupling the generation of grid-points and the generation of the Hamiltonian as one approach of further interest to improve on the proof-of-concept results presented here. We have also shown how our PS strategy can be made more transferrable, for example enabling prediction of excited-state wavefunctions for systems represented on different grid-sizes (Fig. 11); furthermore, we also envisage that the same PS strategy might be useful in predicting other properties beyond wavefunctions, such as energy eigenvalues, and this is a current domain of interest. The choice of optimization strategy and the selection of input, internal and output functions have also been highlighted as current challenges, with the co-optimization of these functions along with the algorithm representing an interesting approach from the field of GP which could be employed here. Finally, we have highlighted throughout that the outputs of our PS simulations can be interpreted as working equations; in general, we find that these represent effective Hamiltonians for which the first few eigenstates reproduce those of the exact Hamiltonian matrix. For more complex problems, more diverse function sets, and alternative code structures, the interpretation of the resulting PS-generated codes would be anticipated to be much more challenging; improved interpretability, for example obtained by performing knock-out studies, is clearly desirable and worth further investigation. In summary, this proof-of-concept study has shown how PS might be brought to bear on problems from the domain of quantum chemistry; based on these promising studies, ongoing work is now aimed at tackling more challenging problems.

ACKNOWLEDGMENTS

The author gratefully acknowledges provision of high-performance computing facilities through the Scientific Computing Research Technology Platform at the University of Warwick.

AUTHOR DECLARATIONS

Conflict of interest

The authors have no conflicts to disclose.

DATA AVAILABILITY

Data from Figs. 3, 6-9 and 11 are available at wrap.warwick.ac.uk/154916.

REFERENCES

[1]C. A. Grambow, L. Pattanaik, and W. H. Green, Sci. Data **7**, 137 (2020).

[2]B. Narayanan, P. C. Redfern, R. S. Assary, and L. A. Curtiss, Chem. Sci. **10**, 7449 (2019).

[3]L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond, J. Chem. Inf. Model. **52**, 2864 (2012).

[4]R. Ramakrishnan, P. O. Dral, M. Rupp, and O. A. von Lilienfeld, Sci. Data **1** (2014).

[5]M. Rupp, A. Tkatchenko, K.-R. Müller, and O. A. von Lilienfeld, Phys. Rev. Lett. **108**, 058301 (2012).

[6]L. C. Blum and J.-L. Reymond, J. Am. Chem. Soc. **131**, 8732 (2009).

[7]M. M. Ghahremanpour, P. J. van Maaren, and D. van der Spoel, Sci. Data **5**, 180062 (2018).

[8]J. Liang, Y. Xu, R. Liu, and X. Zhu, Sci. Data **6**, 213 (2019).

[9]F. Noé, A. Tkatchenko, K.-R. Müller, and C. Clementi, Annu. Rev. Phys. Chem. **71**, 361 (2020).

[10]J. S. Smith, A. E. Roitberg, and O. Isayev, ACS Med. Chem. Lett. **9**, 1065 (2018).

[11]O. A. von Lilienfeld, Angew. Chemie Int. Ed. **57**, 4164 (2018).

[12]J. R. Kitchin, Nat. Catal. **1**, 230 (2018).

[13]P. O. Dral, J. Phys. Chem. Lett. **11**, 2336 (2020).

[14]J. P. Janet, F. Liu, A. Nandy, C. Duan, T. Yang, S. Lin, and H. J. Kulik, Inorg. Chem. **58**, 10592 (2019).

[15]R. Gómez-Bombarelli and A. Aspuru-Guzik, "Machine learning and big-data in computational chemistry," in *Handbook of Materials Modeling: Methods: Theory and Modeling*, edited by W. Andreoni and S. Yip (Springer International Publishing, Cham, 2020) pp. 1939–1962.

[16]G. B. Goh, N. O. Hodas, and A. Vishnu, J. Comp. Chem. **38**, 1291 (2017).

[17]R. Ramprasad, R. Batra, G. Pilania, A. Mannodi-Kanakkithodi, and C. Kim, NPJ Comput. Mater. **3**, 54 (2017).

[18]K. T. Butler, D. W. Davies, H. Cartwright, O. Isayev, and A. Walsh, Nature **559**, 547 (2018).

[19]Z. Qiao, M. Welborn, A. Anandkumar, F. R. Manby, and T. F. Miller, J. Chem. Phys. **153**, 124111 (2020).

[20]T. Husch, J. Sun, L. Cheng, S. J. R. Lee, and T. F. Miller, J. Chem. Phys. **154**, 064108 (2021).

[21]O. T. Unke, D. Koner, S. Patra, S. Käser, and M. Meuwly, Mach. Learn.: Sci. Technol. **1**, 013001 (2020).

[22]S. Manzhos and T. Carrington, J. Chem. Phys. **125**, 084109 (2006).

[23]P. O. Dral, A. Owens, A. Dral, and G. Csányi, J. Chem. Phys. **152**, 204110 (2020).

[24]G. Schmitz, I. H. Godtliebsen, and O. Christiansen, J. Chem. Phys. **150**, 244113 (2019).

[25]B. Jiang, J. Li, and H. Guo, J. Phys. Chem. Lett. **11**, 5120 (2020).

[26]A. P. Bartók, M. C. Payne, R. Kondor, and G. Csányi, Phys. Rev. Lett. **104**, 136403 (2010).

[27]S. Amabilino, L. A. Bratholm, S. J. Bennie, A. C. Vaucher, M. Reiher, and D. R. Glowacki, J. Phys. Chem. A **123**, 4486 (2019).

[28]G. W. Richings and S. Habershon, J. Chem. Theory Comput. **13**, 4012 (2017).

[29]A. P. Bartók and G. Csányi, Int. J. Quantum Chem. **115**, 1051 (2015).

[30]A. P. Bartók, M. J. Gillan, F. R. Manby, and G. Csányi, Phys. Rev. B **88**, 054104 (2013).

[31]M. Welborn, L. Cheng, and T. F. Miller, J. Chem. Theory Comput. **14**, 4772 (2018).

[32]S. De, A. P. Bartók, G. Csányi, and M. Ceriotti, Phys. Chem. Chem. Phys. **18**, 13754 (2016).

[33]J. S. Schreck, C. W. Coley, and K. J. M. Bishop, ACS Cent. Sci. **5**, 970 (2019).

[34]C. W. Coley, W. Jin, L. Rogers, T. F. Jamison, T. S. Jaakkola, W. H. Green, R. Barzilay, and K. F. Jensen, Chem. Sci. **10**, 370 (2019).

[35]A. L. Dewyer, A. J. Argüelles, and P. M. Zimmerman, WIRES Comput. Mol. Sci. **8**, e1354 (2018).

[36]G. N. Simm, A. C. Vaucher, and M. Reiher, J. Phys. Chem. A **123**, 385 (2019).

[37]P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. A. Hunter, C. Bekas, and A. A. Lee, ACS Cent. Sci. **5**, 1572 (2019).

[38]C. David and D. Kroening, Phil. Trans. Roy. Soc. A **375**, 20150403 (2017).

[39]P. Flener and U. Schmid, Artif. Intell. Rev. **29**, 45 (2008).

[40]A. W. Biermann, J. Symb. Comput. **1**, 119 (1985).

[41]J. Fisher and S. Woodhouse, Curr. Opin. Struct. Biol. **4**, 64 (2017).

[42]D. Cociorva, J. Wilkins, G. Baumgartner, P. Sadayappan, J. Ramanujam, M. Nooijen, D. Bernholdt, and R. Harrison, in *High Performance Computing — HiPC 2001*, edited by B. Monien, V. K. Prasanna, and S. Vajapeyam (Springer Berlin Heidelberg, Berlin, Heidelberg, 2001) pp.

237–248.

[43]J. R. Koza, Stat. Comput. **4**, 87 (1994).

[44]J. F. Miller, *Cartesian genetic programming* (Springer, 2011).

[45]J. R. Koza, *Genetic programming : on the programming of computers by means of natural selection* (MIT Press, 1992).

[46]S. Hirata, J. Phys. Chem. A **107**, 9887 (2003).

[47]S. Hirata, J. Chem. Phys. **121**, 51 (2004).

[48]C. L. Janssen and H. F. Schaefer, Theor. Chim. Acta **79**, 1 (1991).

[49]J. R. Koza, Stat. Comput. **4**, 87 (1994).

[50]D. E. Makarov and H. Metiu, J. Phys. Chem. A **104**, 8540 (2000).

[51]G. Wilson and W. Banzhaf, in *Genetic Programming*, edited by M. O'Neill, L. Vanneschi, S. Gustafson, A. I. Esparcia Alcázar, I. De Falco, A. Della Cioppa, and E. Tarantino (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008) pp. 182–193.

[52]J. V. Lill, G. A. Parker, and J. C. Light, J. Chem. Phys. **85**, 900 (1986).

[53]J. C. Light, I. P. Hamilton, and J. V. Lill, J. Chem. Phys. **82**, 1400 (1985).

[54]R. W. Heather and J. C. Light, J. Chem. Phys. **79**, 147 (1983).

[55]D. J. Tannor, *Introduction to quantum mechanics: A time-dependent perspective* (University Science Books, Sausalito, CA, USA, 2007).

[56]J. F. Miller, D. Job, and V. K. Vassilev, Genet. Program. Evolvable Mach. **1**, 7 (2000).

[57]J. F. Miller, in *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99 (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999) pp. 1135–1142.

[58]D. T. Colbert and W. H. Miller, J. Chem. Phys. **96**, 1982 (1992).

[59]J. R. Koza *et al.*, *Genetic programming II*, Vol. 17 (MIT press Cambridge, MA, 1994).

[60]J. R. Koza, D. Andre, F. H. Bennett III, and M. A. Keane, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, edited by J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo (MIT Press, Stanford University, CA, USA, 1996) pp. 132–149.

[61]S. Manzhos, K. Yamashita, and T. Carrington, Chem. Phys. Lett. **474**, 217 (2009).

[62]S. Manzhos and T. Carrington, Can. J. Chem. **87**, 864 (2009).

[63]M. Sugawara, Comp. Phys. Commun. **140**, 366 (2001).

[64]I. Lagaris, A. Likas, and D. Fotiadis, Comp. Phys. Commun. **104**, 1 (1997).

[65]S. Habershon, "In preparation," (2021).

Appendix A: Algorithm pseudo-codes

Figures 12 to 17 gives python pseudo-code for algorithms C_{15}(A), C_{15}(B), C_{20}(A), C_{20}(B) and C_{20}(C). Here, the `V` is the vector containing the PES values evaluated on the uniform grid, the `numpy` function `eig` determines the eigenvalues and eigenvectors of an input matrix, and the function `SortAscending()` sorts the eigenvectors and eigenvalues in ascending order. Furthermore, the matrix `M[:,:]` is the workspace matrix and `w[:]` is the output vector representing the output wavefunction. Finally, we note that the output ground-state wavefunction in `w[:]` is subsequently normalized using standard numerical integration.

```
import numpy as np
n = len(V)
M[:,:] = 0.0
M[:, :] = 1.0
M /= 4.0
M -= 3.0
M *= 3.0
M -= np.pi
M += 3.0
for i in range(n):
    M[i,i] += 4.0
for i in range(n):
    for j in range(n):
        if i != j:
            M[i, j] *= 1.0 / (i - j)
for i in range(n):
    M[i,i]/=np.pi
    M[i,i] += V[i]
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
for i in range(n):
    for j in range(n):
        if i != j:
            M[i, j] *= 1.0 / (i - j)
E, c = np.linalg.eig(M)
SortAscending(E,c)
w[:] = c[:, 0]
```

FIG. 12. Python pseudo-code for algorithm C_{15}(A).

```
import numpy as np
n = len(V)
y[:] = 1.0
M[:,:] = 1.0
for i in range(n):
    M[i,i] *= V[i]
    M[i,i] *= dx
    M[i,i] -= mass
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp((i*dx - j*dx)**2)
M += 4.0
for i in range(n):
    M[i,i] -= mass
M = np.sin(M)
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
for i in range(n):
    M[i,i] += V[i]
E, c = np.linalg.eig(M)
SortAscending(E,c)
w[:] = c[:, 0]
```

FIG. 13. Python pseudo-code for algorithm C_{15}(B).

```
import numpy as np
n = len(V)
M[:,:] = 0.0
for i in range(n):
    M[i,i] = 1.0
M -= mass
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
for i in range(n):
    M[i,i] += 2.0
    M[i,i] *= dx
    M[i,i] -= mass
    M[i,i] *= mass
M += dx
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)** 2)
for i in range(n):
    M[i,i] += np.pi
M += dx
for i in range(n):
    M[i,i] *= V[i]
M /= np.pi
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
for i in range(n):
    M[i,i] /= 4.0
E, c = np.linalg.eig(M)
SortAscending(E,c)
w[:] = c[:, 0]
```

FIG. 14. Python pseudo-code for Algorithm C_{20}(A).

```
import numpy as np
n = len(V)
for i in range(n):
    M[i,i] = 1.0
M *= 2.0
M = np.cos(M)
M -= dx
M = -M
for i in range(n):
    for j in range(n):
        if i != j:
            M[i, j] *= 1.0 / (i - j)
for i in range(n):
    M[i,i] += np.pi
    M[i,i] -= mass
    M[i,i] /= np.pi
M *= 2.0
for i in range(n):
    for j in range(n):
        if i != j:
            M[i, j] *= 1.0 / (i - j)
for i in range(n):
    M[i,i] += dx
    M[i,i] += 4.0
M = np.dot(M,M)
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
M *= 2.0
for i in range(n):
    M[i,i] += V[i]
E, c = np.linalg.eig(M)
SortAscending(E,c)
w[:] = c[:, 0]
```

FIG. 15. Python pseudo-code for algorithm C_{20}(B).

```
import numpy as np
n = len(V)
M[:,:] = 1.0
M += 4.0
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp((i*dx - j*dx)**2)
M -= 4.0
for i in range(n):
    M[i,i] += V[i]
M -= np.pi
M /= 3.0
for i in range(n):
    M[i,i] += 2.0
M /= 2.0
M -= dx
M -= dx
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
E, c = np.linalg.eig(M)
SortAscending(E,c)
w[:] = c[:, 0]
```

FIG. 16. Python pseudo-code for algorithm C_{20}(C).

```
import numpy as np
n = len(V)
M[:,:] = 1.0
M += 4.0
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp((i*dx - j*dx)**2)
M -= 4.0
for i in range(n):
    M[i,i] += V[i]
M -= np.pi
M /= 3.0
for i in range(n):
    M[i,i] += 2.0
M /= 2.0
M -= dx
M -= dx
for i in range(n):
    for j in range(n):
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
        M[i, j] *= np.exp(-(i*dx - j*dx)**2)
E, c = np.linalg.eig(M)
SortAscending(E,c)
w[:] = c[:, 0]
```

FIG. 17. Python pseudo-code for algorithm C_{20}(C).