Al Feynman: A physics-inspired method for symbolic regression

Silviu-Marian Udrescu¹ and Max Tegmark¹,²*

A core challenge for both physics and artificial intelligence (AI) is symbolic regression: finding a symbolic expression that matches data from an unknown function. Although this problem is likely to be NP-hard in principle, functions of practical interest often exhibit symmetries, separability, compositionality, and other simplifying properties. In this spirit, we develop a recursive multidimensional symbolic regression algorithm that combines neural network fitting with a suite of physics-inspired techniques. We apply it to 100 equations from the Feynman Lectures on Physics, and it discovers all of them, while previous publicly available software cracks only 71; for a more difficult physics-based test set, we improve the state-of-the-art success rate from 15 to 90%.

INTRODUCTION

In 1601, Johannes Kepler got access to the world’s best data tables on planetary orbits, and after 4 years and about 40 failed attempts to fit the Mars data to various ovoid shapes, he launched a scientific revolution by discovering that Mars’ orbit was an ellipse (1). This was an example of symbolic regression: discovering a symbolic expression that accurately matches a given dataset. More specifically, we are given a table of numbers, whose rows are of the form \( x_1, \ldots, x_m, y \) where \( y = f(x_1, \ldots, x_m) \), and our task is to discover the correct symbolic expression for the unknown mystery function \( f \), optionally including the complication of noise.

Growing datasets have motivated attempts to automate such regression tasks, with notable success. For the special case where the unknown function \( f \) is a linear combination of known functions of \( x_1, \ldots, x_m \), symbolic regression reduces to simply solving a system of linear equations. Linear regression (where \( f \) is simply an affine function) is ubiquitous in the scientific literature, from finance to psychology. The case where \( f \) is a linear combination of monomials in \( x_1, \ldots, x_n \) corresponds to linear regression with interaction terms, and to polynomial fitting more generally. There are countless other examples of popular regression functions that are linear combinations of known functions, ranging from Fourier expansions to wavelet transforms. Despite these successes with special cases, the general symbolic regression problem remains unsolved, and it is easy to see why: If we encode functions as strings of symbols, then the number of such strings grows exponentially with string length, so if we simply test all strings by increasing length, it may take longer than the age of our universe until we get to the function we are looking for.

This combinatorial challenge of an exponentially large search space characterizes many famous classes of problems, from code breaking and Rubik’s cube to the natural selection problem of finding those genetic codes that produce the most evolutionarily fit organisms. This has motivated genetic algorithms (2, 3) for targeted searches in exponentially large spaces, which replace the above-mentioned brute-force search by biology-inspired strategies of mutation, selection, inheritance, and recombination; crudely speaking, the role of genes is played by useful symbol strings that may form part of the sought-after formula or program. Such algorithms have been successfully applied to areas ranging from design of antennas (4, 5) and vehicles (6) to wireless routing (7), vehicle routing (8), robot navigation (9), code breaking (10), discovering partial differential equations (11), investment strategy (12), marketing (13), classification (14), Rubik’s cube (15), program synthesis (16), and metabolic networks (17).

The symbolic regression problem for mathematical functions (the focus of this paper) has been tackled with a variety of methods (18–20), including sparse regression (21–24) and genetic algorithms (25, 26). By far, the most successful of these is, as we will see in Results, the genetic algorithm outlined in (27) and implemented in the commercial Eureqa software (26).

The purpose of this paper was to further improve on this state of the art, using physics-inspired strategies enabled by neural networks. Our most important contribution is using neural networks to discover hidden simplicity such as symmetry or separability in the mystery data, which enables us to recursively break harder problems into simpler ones with fewer variables.

The rest of this paper is organized as follows. In Results, we present the results of applying our algorithm, which recursively combines six strategies, finding major improvements over the state-of-the-art Eureqa algorithm. In Discussion, we summarize our conclusions and discuss opportunities for further progress.

RESULTS

In this section, we present our results and the algorithm by which they were obtained.

Overall algorithm

Generic functions \( f(x_1, \ldots, x_n) \) are extremely complicated and near impossible for symbolic regression to discover. However, functions appearing in physics and many other scientific applications often have some of the following simplifying properties that make them easier to discover:

(1) Units: \( f \) and the variables upon which it depends have known physical units.
(2) Low-order polynomial: \( f \) (or part thereof) is a polynomial of low degree.
(3) Compositionality: \( f \) is a composition of a small set of elementary functions, each typically taking no more than two arguments.

¹Department of Physics and Center for Brains, Minds & Machines, Massachusetts Institute of Technology, Cambridge, MA 02139, USA. ²Theiss Research, La Jolla, CA 92037, USA.
*Corresponding author. Email: tegmark@mit.edu
(4) Smoothness: $f$ is continuous and perhaps even analytic in its domain.

(5) Symmetry: $f$ exhibits translational, rotational, or scaling symmetry with respect to some of its variables.

(6) Separability: $f$ can be written as a sum or product of two parts with no variables in common.

The question of why these properties are common remains controversial and not fully understood (28, 29). However, as we will see below, this does not prevent us from discovering and exploiting these properties to facilitate symbolic regression.

Property (1) enables dimensional analysis, which often transforms the problem into a simpler one with fewer independent variables. Property (2) enables polynomial fitting, which quickly solves the problem by solving a system of linear equations to determine the polynomial coefficients. Property (3) enables $f$ to be represented as a parse tree with a small number of node types, sometimes enabling $f$ or a subexpression to be found via a brute-force search. Property (4) enables approximating $f$ using a feed-forward neural network with a smooth activation function. Property (5) can be confirmed using said neural network and enables the problem to be transformed into a simpler one with one independent variable less (or even fewer for $n > 2$ rotational symmetry). Property (6) can be confirmed using said neural network and enables the independent variables to be partitioned into two disjoint sets and the problem to be transformed into two simpler ones, each involving the variables from one of these sets.

The overall algorithm (available at https://github.com/SJ001/AlFeynman) is schematically illustrated in Fig. 1. It consists of a series of modules that try to exploit each of the above-mentioned properties. Like a human scientist, it tries many different strategies (modules) in turn, and if it cannot solve the full problem in one fell swoop, it tries to transform it and divide it into simpler pieces that can be tackled separately, recursively relaunching the full algorithm on each piece. Figure 2 illustrates an example of how a particular mystery dataset (Newton’s law of gravitation with nine variables) is solved. Below, we describe each of these algorithm modules in turn.

**Dimensional analysis**

Our dimensional analysis module exploits the well-known fact that many problems in physics can be simplified by requiring the units of the two sides of an equation to match. This often transforms the problem into a simpler one with a smaller number of variables that are all dimensionless. In the best-case scenario, the transformed problem involves solving for a function of zero variables, i.e., a constant. We automate dimensional analysis as follows.

Table 3 shows the physical units of all variables appearing in our 100 mysteries, expressed as products of the fundamental units (meter, second, kilogram, kelvin, and volt) to various integer powers. We, thus, represent the units of each variable by a vector $\mathbf{u}$ of five integers as in the table. For a mystery of the form $y = f(x_1, \ldots, x_n)$, we define the matrix $\mathbf{M}$ whose $j^{th}$ column is the vector corresponding to the variable $x_j$, and define the vector $\mathbf{b}$ as the vector corresponding to $y$. We now let the vector $\mathbf{p}$ be a solution to the equation $\mathbf{M} \mathbf{p} = \mathbf{b}$, and the columns of the matrix $\mathbf{U}$ form a basis for the null space, so that $\mathbf{MU} = 0$, and define a new mystery $y' = f'(x'_1, \ldots, x'_{n'})$ where

$$x'_i \equiv \prod_{j=1}^{n} x_j^{u_{ij}}, \quad y' \equiv \frac{y}{\prod_{j=1}^{n} x_j^{p_j}}. \quad (1)$$

By construction, the new variables $x'_i$ and $y'$ are dimensionless, and the number $n'$ of new variables is equal to the dimensionality of the null space. When $n' > 0$, we have the freedom to choose any basis we want for the null space and also to replace $\mathbf{p}$ by a vector of the form $\mathbf{p} + \mathbf{Ua}$ for any vector $\mathbf{a}$; we use this freedom to set as many elements as possible in $\mathbf{p}$ and $\mathbf{U}$ equal to zero, i.e., to make the new variables depend on as few old variables as possible. This choice is useful because it typically results in the resulting powers of the dimensionless variables being integers, making the final expression much easier to find than when the powers are fractions or irrational numbers.

**Polynomial fit**

Many functions $f(x_1, \ldots, x_n)$ in physics and other sciences either are low-order polynomials, e.g., the kinetic energy $K = \frac{1}{2}m(v_x^2 + v_y^2 + v_z^2)$,
polynomial fitting, the latter after applying one of a series of simple transformations and work was trained to fit this function, which revealed two translational symmetries $\epsilon^p$ fitting error $\leq \epsilon_p$ or after a maximum runtime $t_{\text{max}}$ has been exceeded. Although this module alone could solve all our mysteries in principle, it would, in many cases, take longer than the age of our universe in practice. Our brute-force method is, thus, typically most helpful once a mystery has been transformed/broken apart into simpler pieces by the modules described below.

We generate the expressions to try by representing them as strings of symbols, trying first all strings of length 1, then all of length 2, etc., saving time by only generating those strings that are syntactically correct. The symbols used are the independent variables as well as a subset of those listed in Table 1, each representing a constant or a function. We minimize string length by using reverse Polish notation, so that parentheses become unnecessary. For example, $x + y$ can be expressed as the string "xy+", the number $-2/3$ can be expressed as the string "$0<<1>>/\sqrt{3}$", and the relativistic momentum formula $\nu v^2/c^2$ can be expressed as the string "$\nu v^2/c^2$".

Polynomial fit

or have parts that are, e.g., the denominator of the gravitational force $F = \frac{Gm_1m_2}{(x_2-x_1)^2 + (y_2-y_1)^2 + (z_2-z_1)^2}$. We therefore include a module that tests whether a mystery can be solved by a low-order polynomial. Our method uses the standard method of solving a system of linear equations to find the best-fit polynomial coefficients. It tries fitting the mystery data to polynomials of degree 0, 1, ..., $d_{\text{max}} = 4$ and declares success if the best-fitting polynomial gives root mean square (rms) fitting error $\leq \epsilon_p$ (we discuss the setting of this threshold below).

**Brute force**

Our brute-force symbolic regression model simply tries all possible symbolic expressions within some class, in order of increasing complexity, terminating either when the maximum fitting error drops below a threshold $\epsilon_p$ or after a maximum runtime $t_{\text{max}}$ has been exceeded. Although this module alone could solve all our mysteries in principle, it would, in many cases, take longer than the age of our universe in practice. Our brute-force method is, thus, typically most helpful once a mystery has been transformed/broken apart into simpler pieces by the modules described below.

We generate the expressions to try by representing them as strings of symbols, trying first all strings of length 1, then all of length 2, etc., saving time by only generating those strings that are syntactically correct. The symbols used are the independent variables as well as a subset of those listed in Table 1, each representing a constant or a function. We minimize string length by using reverse Polish notation, so that parentheses become unnecessary. For example, $x + y$ can be expressed as the string "xy+", the number $-2/3$ can be expressed as the string "$0<<1>>/\sqrt{3}$", and the relativistic momentum formula $\nu v^2/c^2$ can be expressed as the string "$\nu v^2/c^2$".

Inspection of Table 1 reveals that many of the symbols are redundant. For example, "$\pi$" is "$0>\pi$" and "$\times\sim$" is "$0\times\pi$". $\pi = 2$ arcsin 1, so if we drop the symbol "$\pi$", mysteries involving $\pi$ can still get solved with $P$ replaced by "$111>\pi$"—it just takes longer.

Since there are $s^d$ strings of length $n$ using an alphabet of $s$ symbols, there can be a substantial cost both from using too many symbols (increasing $s$) and from using too few symbols (increasing the required $n$ or even making a solution impossible). As a compromise, our brute-force module tries to solve the mystery using three different symbol subsets as explained in the caption of Table 1. To exploit the fact that many equations or parts thereof have multiplicative or additive constants, our brute-force method comes in two variants that automatically solves for such constants, thus allowing the algorithm to focus on the symbolic expression and not on numerical constants. Although the problem of overfitting is most familiar when searching a continuous parameter space, the same phenomenon can occur when searching our discrete space of symbol strings. To mitigate this, we follow the prescription in (30) and define the winning function to be the one with rms fitting error $\epsilon < \epsilon_b$ that has the smallest total description length

$$DL \equiv \log_2 N + \lambda \log_2 \left( \max \left( 1, \frac{\epsilon}{\epsilon_d} \right) \right)$$

where $\epsilon_d = 10^{-15}$, and $N$ is the rank of the string on the list of all strings tried. The two terms correspond roughly to the number of bits required to store the symbol string and the prediction errors, respectively, if the hyperparameter $\lambda$ is set to equal the number of data points $N_d$. We use $\lambda = N_d^{1/2}$ in our experiments below to prioritize simpler formulas. If the mystery has been generated using a neural network (see below), we set the precision threshold $\epsilon_b$ to 10 times the validation error, otherwise we set it to $10^{-5}$.

**Neural network–based tests and transformations**

Even after applying the dimensional analysis, many mysteries are still too complex to be solved by the polyfit or brute-force modules in a reasonable amount of time. However, if the mystery function $f(x_1, \ldots, x_n)$ can be found to have simplifying properties, it may be possible to transform it into one or more simpler mysteries that can be more easily solved. To search for such properties, we need to be able to evaluate $f$ at points $\{x_1, \ldots, x_n\}$ of our choosing where we
Neural network training

To obtain such an interpolating function for a given mystery, we train a neural network to predict the output given its input. We train a feed-forward, fully connected neural network with six hidden layers with soft plus activation functions, the first three having 128 neurons and the last three having 64 neurons. For each mystery, we generated 100,000 data points, using 80% as the training set and the remainder as the validation set, training for 100 epochs with learning rate 0.005 and batch size 2048. We use the rms error loss function and the Adam optimizer with soft plus activation functions, the first three having 128 neurons and the last three having 64 neurons. For each mystery, we generated 100,000 data points, using 80% as the training set and the remainder as the validation set, training for 100 epochs with learning rate 0.005 and batch size 2048. We use the rms error loss function and the Adam optimizer with weight decay of $10^{-2}$. The learning rate and momentum schedules were implemented as described in (31, 32) using the FastAI package (33), with a ratio of 20 between the maximum and minimum learning rates, and using 10% of the iterations for the last part of the training cycle. For the momentum, the maximum $\beta_1$ value was 0.95 and the minimum 0.85, while $\beta_2 = 0.99$.

If the neural network were expressive enough to be able to perfectly fit the mystery function, and the training process would never get stuck in a local minimum, then one might naively expect the rms validation error $e_{NN}^0$ to scale as $f_{rms} \varepsilon / N_{d}^{1/2}$ in the limit of ample data, with a constant prefactor depending on the number of function arguments and the function’s complexity. Here, $f_{rms}$ is the rms of the $f$ values in the dataset, $N_d$ is the number of data points, and $\varepsilon$ is the relative rms noise on the independent variable as explored in the “Dependence on noise level” section. For realistic situations, one expects limited expressibility and convergence to keep $e_{NN}^0$ above some positive floor even as $N_d \to \infty$ and $\varepsilon \to 0$. In practice, we obtained $e_{NN}^0$ values between $10^{-3}f_{rms}$ and $10^{-5}f_{rms}$ across the range of tested equations.

Table 1. Functions optionally included in brute-force search. The following three subsets are tried in turn: “+−*/><~SPLICER”, “+−*/> 0~” and “−*/><~REPLICANT50”.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Add</td>
<td>2</td>
</tr>
<tr>
<td>*</td>
<td>Multiply</td>
<td>2</td>
</tr>
<tr>
<td>−</td>
<td>Subtract</td>
<td>2</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
<td>2</td>
</tr>
<tr>
<td>⩾</td>
<td>Increment</td>
<td>1</td>
</tr>
<tr>
<td>⩽</td>
<td>Decrement</td>
<td>1</td>
</tr>
<tr>
<td>~</td>
<td>Negate</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>sqrt</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>exp</td>
<td>1</td>
</tr>
<tr>
<td>P</td>
<td>π</td>
<td>0</td>
</tr>
<tr>
<td>L</td>
<td>ln</td>
<td>1</td>
</tr>
<tr>
<td>I</td>
<td>invert</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>cos</td>
<td>1</td>
</tr>
<tr>
<td>A</td>
<td>abs</td>
<td>1</td>
</tr>
<tr>
<td>N</td>
<td>arcsin</td>
<td>1</td>
</tr>
<tr>
<td>T</td>
<td>arctan</td>
<td>1</td>
</tr>
<tr>
<td>S</td>
<td>sin</td>
<td>1</td>
</tr>
</tbody>
</table>

typically have no data. For example, to test whether a function $f$ has translational symmetry, we need to test if $f(x_1, x_2) = f(x_1 + a, x_2 + a)$ for various constants $a$, but if a given data point has its two variables separated by $x_2 - x_1 = 1.61803$, we typically have no other examples in our dataset with exactly that variable separation. To perform our tests, we thus need an accurate high-dimensional interpolation between our data point.

Translational symmetry and generalizations

We test for translational symmetry using the neural network as detailed in Algorithm 1. We first check if the $f(x_1, x_2, x_3, \ldots) = f(x_1 + a, x_2 + a, x_3, \ldots)$ to within a precision $\epsilon_{sym}$. If that is the case, then $f$ depends on $x_1$ and $x_2$ only through their difference, so we replace these two input variables by a single new variable $x_1' \equiv x_2 - x_1$. Otherwise, we repeat this test for all pairs of input variables and also test whether any variable pair can be replaced by its sum, product, or ratio. The ratio case corresponds to scaling symmetry, where two variables can be simultaneously rescaled without changing the answer. If any of these simplifying properties is found, the resulting transformed mystery (with one fewer input variables) is iteratively passed into a fresh instantiation of our full AI Feynman symbolic regression algorithm, as illustrated in Fig. 1. After experimentation, we chose the precision threshold $\epsilon_{sym}$ to be seven times the neural network validation error, which roughly optimized the training set performance. (If the noise were Gaussian, even a cut at 4 rather than 7 standard deviations would produce negligible false positives.)

Separability

We test for separability using the neural network as exemplified in Algorithm 2. A function is separable if it can be split into two parts with no variables in common. We test for both additive and multiplicative separability, corresponding to these two parts being added and multiplied, respectively (the logarithm of a multiplicatively separable function is additively separable).

For example, to test whether a function of two variables is multiplicatively separable, i.e., of the form $f(x_1, x_2) = g(x_1)h(x_2)$ for some univariate functions $g$ and $h$, we first select two constants $c_1$ and $c_2$; for numerical robustness, we choose $c_1$ to be the means of all the values of $x_1$ in the mystery dataset, $i = 1, 2$. We then compute the quantity

$$\Delta_{sep}(x_1, x_2) \equiv f_{rms}^{-1} \left| f(x_1, x_2) - \frac{f(x_1, c_2)f(c_1, x_2)}{f(c_1, c_2)} \right|$$

for each data point. This is a measure of nonseparability, since it vanishes if $f$ is multiplicatively separable. The equation is considered separable if the rms average $\Delta_{sep}$ over the mystery dataset is less than an accuracy threshold $\epsilon_{sep}$, which is chosen to be $N = 10$ times the neural network validation error. [We also check whether the function is multiplicatively separable up to an additive constant: $f(x_1, x_2) = a + g(x_1)h(x_2)$, where $a$ is a constant. As a backup, we retain the above-mentioned simpler test for multiplicative separability, which proved more robust when $a = 0$.]

If separability is found, we define the two new univariate mysteries $y' \equiv f(x_1, c_2)$ and $y'' \equiv f(c_1, x_2)f(c_1, c_2)$. We pass the first one, $y'$, back to fresh instantiations of our full AI Feynman symbolic regression algorithm, and if it gets solved, we redefine $y'' \equiv y'y'_{num}$, where $y_{num}$ represents any multiplicative numerical constant that appears in $y'$. We then pass $y''$ back to our algorithm, and if it gets solved, the final solution is $y = y'' / y'_{num}$. We test for additive separability analogously, simply replacing * and / by + and − above; also, $y_{num}$ will represent...
an additive numerical constant in this case. If we succeed in solving
the two parts, then the full solution to the original mystery is the
sum of the two parts minus the numerical constant. When there are
more than two variables \( x_i \), we are testing all the possible subsets
of variables that can lead to separability and proceed as above for the
newly created two mysteries.

**Setting variables equal**

We also exploit the neural network to explore the effect of setting
two input variables equal and attempting to solve the corresponding
new mystery \( y' \) with one fewer variable. We try this for all variable
pairs, and if the resulting new mystery is solved, we try solving the
mystery \( y'' \equiv y'/y' \) that has the found solution divided out.

As an example, this technique solves the Gaussian probability dis-
tribution mystery I.6.2. After making \( \theta \) and \( \sigma \) equal and dividing
the initial equation by the result, we are getting rid of the denominator,
and the remaining part of the equation is an exponential. After taking
the logarithm of this (see the below section), the resulting expression
can be easily solved by the brute-force method.

**Extra transformations**

In addition, several transformations are applied to the dependent
and independent variables, which proved to be useful for solving
certain equations. Thus, for each equation, we ran the brute force
and polynomial fit on a modified version of the equation in which
the dependent variable was transformed by one of the following
functions: square root, raise to the power of 2, log, exp, inverse, sin,
cos, tan, arcsin, arccos, and arctan. This reduces the number of symbols
needed by the brute force by one, and in certain cases, it even
allows the polynomial fit to solve the equation, when the brute
force would otherwise fail. For example, the formula for the dis-
tance between two points in the three-dimensional (3D) Euclidean
space: \( \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \), once raised to the
power of 2 becomes just a polynomial that can be easily discovered
by the polynomial fit algorithm. The same transformations are also applied
to the dependent variables, one at a time. In addition, multiplication
and division by 2 were added as transformations in this case.

It should be noted that, like most machine-learning methods, the
AI Feynman algorithm has some hyperparameters that can be
tuned to optimize performance on the problems at hand. They were
all introduced above, but for convenience, they are also summarized
in Table 2.

**The Feynman Symbolic Regression Database**

To facilitate quantitative testing of our and other symbolic regression
algorithms, we created the 6-gigabyte Feynman Symbolic Regression
Database (FSReD) and made it freely available for download at
https://space.mit.edu/home/tegmark/aifeynman.html. For each
regression mystery, the database contains the following:

1) Data table: A table of numbers, whose rows are of the form
\( (x_1, x_2, \ldots, y) \), where \( y = f(x_1, x_2, \ldots) \); the challenge is to discover the
correct analytic expression for the mystery function \( f \).

2) Unit table: A table specifying the physical units of the input
and output variables as 6D vectors of the form seen in Table 3.

3) Equation: The analytic expression for the mystery function \( f \),
for answer checking.

To test an analytic regression algorithm using the database, its task
is to predict \( f \) for each mystery taking the data table (and optionally
the unit table) as input. Of course, there are typically many symbolically
different ways of expressing the same function. For example, if the
mystery function \( f \) is \((u + v)/(1 + uv/c^2)\), then the symbolically dif-
ferent expression \((v + u)/(1 + uv/c^2)\) should count as a correct solution.
The rule for evaluating an analytic regression method is therefore
that a mystery function \( f \) is deemed correctly solved by a candidate
expression \( f' \) if algebraic simplification of the expression \( f' = f \) (say,
with the Simplify function in “Mathematica” or the simplify
function in the Python SymPy package) produces the symbol “0.”

To sample equations from a broad range of physics areas, the
database is generated using 100 equations from the seminal
Feynman Lectures on Physics (34–36), a challenging three-volume course cover-
ing classical mechanics, electromagnetism, and quantum mechanics
as well as a selection of other core physics topics; we prioritized the
most complex equations, excluding ones involving derivatives or
integrals. The equations are listed in Tables 4 and 5 and can be seen
to involve between one and nine independent variables as well as the
elementary functions \(+, −, *, /, \sqrt{\text{r}}, \exp, \log, \sin, \cos, \arctan\).
The numbers appearing in these equations are seen to be simple
rational numbers as well as \( e \) and \( \pi \).

We also included in the database a set of 20 more challenging
“bonus” equations, extracted from other seminal physics books:
Classical Mechanics by Goldstein et al. (37); Classical Electrodynamics
by Jackson (38); Gravitation and Cosmology: Principles and Applications
of the General Theory of Relativity by Weinberg (39); and Quantum
Field Theory and the Standard Model by Schwartz (40). These equations
were selected for being both famous and complicated.

The data table provided for each mystery equation contains
10^5 rows corresponding to randomly generated input variables.
These are sampled uniformly between one and five. For certain
equations, the range of sampling was slightly adjusted to avoid
unphysical result, such as division by zero, or taking the square root
of a negative number. The range used for each equation is listed in
the FSReD.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon_{br} )</td>
<td>Tolerance in brute-force module</td>
<td>( 10^{-5} )</td>
</tr>
<tr>
<td>( \epsilon_{pol} )</td>
<td>Tolerance in polynomial fit module</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>( \epsilon_{NN} )</td>
<td>Validation error tolerance for neural network use</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>( \epsilon_{sep} )</td>
<td>Tolerance for separability</td>
<td>( 10 \epsilon_{NN} )</td>
</tr>
<tr>
<td>( \epsilon_{sym} )</td>
<td>Tolerance for symmetry</td>
<td>( 7 \epsilon_{NN} )</td>
</tr>
<tr>
<td>( \epsilon_{sep_{bf}} )</td>
<td>Tolerance in brute-force module after separability</td>
<td>( 10 \epsilon_{NN} )</td>
</tr>
<tr>
<td>( \epsilon_{sep_{pol}} )</td>
<td>Tolerance in polynomial fit module after separability</td>
<td>( 10 \epsilon_{NN} )</td>
</tr>
<tr>
<td>( \lambda )</td>
<td>Importance of accuracy relative to complexity</td>
<td>( N^2 \frac{1}{2} )</td>
</tr>
</tbody>
</table>
Table 3. Unit table used for our automated dimensional analysis.

<table>
<thead>
<tr>
<th>Variables</th>
<th>Units</th>
<th>m</th>
<th>s</th>
<th>kg</th>
<th>T</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha, g$</td>
<td>Acceleration</td>
<td>1</td>
<td>−2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$h, h, l, j_z$</td>
<td>Angular momentum</td>
<td>2</td>
<td>−1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$A$</td>
<td>Area</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_B$</td>
<td>Boltzmann constant</td>
<td>2</td>
<td>−2</td>
<td>1</td>
<td>−1</td>
<td>0</td>
</tr>
<tr>
<td>$C$</td>
<td>Capacitance</td>
<td>2</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>−2</td>
</tr>
<tr>
<td>$q, q_1, q_2$</td>
<td>Charge</td>
<td>2</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>$j$</td>
<td>Current density</td>
<td>0</td>
<td>−3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$I, I_0$</td>
<td>Current intensity</td>
<td>2</td>
<td>−3</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>$\rho, \rho_0$</td>
<td>Density</td>
<td>−3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\theta, \theta_1, \theta_2, \alpha, \eta$</td>
<td>Dimensionless</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$g, k_B, \gamma, \beta, \alpha$</td>
<td>Dimensionless</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\rho_1, \rho_0, \delta, \beta, \mu$</td>
<td>Dimensionless</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$n_\delta, \delta, f, \mu, Z_1, Z_2$</td>
<td>Dimensionless</td>
<td>2</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$D$</td>
<td>Diffusion coefficient</td>
<td>0</td>
<td>−1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\mu_{\text{drift}}$</td>
<td>Drift velocity</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\rho_d$</td>
<td>Electric dipole moment</td>
<td>3</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>$E_r$</td>
<td>Electric field</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>Electric permittivity</td>
<td>1</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>−2</td>
</tr>
<tr>
<td>$E, K, U$</td>
<td>Energy</td>
<td>2</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$E_{\text{den}}$</td>
<td>Energy density</td>
<td>−1</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$F_E$</td>
<td>Energy flux</td>
<td>0</td>
<td>−3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$F, N_0$</td>
<td>Force</td>
<td>1</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\omega, \omega_0$</td>
<td>Frequency</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$k_G$</td>
<td>Grav. coupling</td>
<td>3</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$H$</td>
<td>Hubble constant</td>
<td>0</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$L_{\text{ind}}$</td>
<td>Inductance</td>
<td>−2</td>
<td>4</td>
<td>−1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>$n_{\text{rho}}$</td>
<td>Inverse volume</td>
<td>−3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$x, x_1, x_2, x_3$</td>
<td>Length</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$y, y_1, y_2, y_3$</td>
<td>Length</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$z, z_1, z_2, f, r, r_1, r_2$</td>
<td>Length</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\lambda, d_1, d_2, d_3, f_0, a_f$</td>
<td>Length</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$l_1, l_2, l, l_0$</td>
<td>Light intensity</td>
<td>0</td>
<td>−3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$B, B_x, B_y, B_z$</td>
<td>Magnetic field</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\mu_m$</td>
<td>Magnetic moment</td>
<td>4</td>
<td>−3</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>$M$</td>
<td>Magnetization</td>
<td>1</td>
<td>−3</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>$m, m_0, m_1, m_2$</td>
<td>Mass</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\mu_e$</td>
<td>Mobility</td>
<td>0</td>
<td>1</td>
<td>−1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$p$</td>
<td>Momentum</td>
<td>1</td>
<td>−1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$G$</td>
<td>Newton's constant</td>
<td>3</td>
<td>−2</td>
<td>−1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$p_v$</td>
<td>Polarization</td>
<td>0</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>$p$</td>
<td>Power</td>
<td>2</td>
<td>−3</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

continued on next page
Algorithm comparison

We reviewed the symbolic regression literature for publicly available software against which our method could be compared. To the best of our knowledge, the best competitor by far is the commercial Eureqa software sold by Nutonian Inc. at https://www.nutonian.com/products/eureqa, implementing an improved version of the generic search algorithm outlined in (27).

We compared the Al Feynman and Eureqa algorithms by applying them both to the Feynman Database for symbolic regression, allowing a maximum of 2 hours of central processing unit (CPU) time per mystery. Tables 4 and 5 show that Eureqa solved 71% of the 100 basic mysteries, while Al Feynman solved 100%.

For this comparison, the Al Feynman algorithm was run using the hyperparameter settings in Table 2. For Eureqa, each mystery was run on four CPUs. The symbols used in trying to solve the equations were +, −, *, /, constant, integer constant, input variable, sqrt, exp, log, sin, and cos. To help Eureqa gain speed, we included the additional functions arcsin and arccos only for those mysteries requiring them, and we used only 300 data points (since it does not use a neural network, adding additional data does not help much). The time taken to solve an equation using our algorithm, as presented in Tables 4 and 5, corresponds to the time needed for an equation to be solved using a set of symbols that can actually solve it (see Table 1). Equations I.15.3t and I.48.2 were solved using the second set of symbols, so the overall time needed for these two equations is 1 hour longer than the one listed in the tables. Equations I.15.3x and II.35.21 were solved using the third set of symbols, so the overall time taken is 2 hours longer than the one listed here.

Closer inspection of these tables reveals that the greatest improvement of our algorithm over Eureqa is for the most complicated mysteries, where our neural network enables eliminating variables by discovering symmetries and separability. The neural network becomes even more important when we rerun Al Feynman without the dimensional analysis module: It now solves 93% of the mysteries and makes very heavy use of the neural network to discover separability and translational symmetries. Without dimensional analysis, many of the mysteries retain variables that appear only raised to some power or in a multiplicative prefactor, and Al Feynman tends to recursively discover them and factor them out one by one. For example, the neural network strategy is used six times when solving

\[ F = \frac{Gm_1m_2}{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \]

without dimensional analysis: three times to discover translational symmetry that replaces \( x_2 - x_1, y_2 - y_1, \) and \( z_2 - z_1 \) by new variables, once to group together \( G \) and \( m_1 \) into a new variable \( a \), once to group together \( a \) and \( m_2 \) into a new variable \( b \), and one last time to discover separability and factor out \( b \). This shows that although dimensional analysis often provides major time savings, it is usually not necessary for successfully solving the problem.

Inspection of how Al Feynman and Eureqa make progress over time reveals interesting differences. The progress of Al Feynman over time corresponds to repeatedly reducing the number of independent variables, and every time this occurs, it is virtually guaranteed to be a step in the right direction. In contrast, genetic algorithms such as Eureqa make progress over time by finding successively better approximations, but there is no guarantee that more accurate symbolic expressions are closer to the truth when viewed as strings of symbols. Specifically, by virtue of being a genetic algorithm, Eureqa has the

<table>
<thead>
<tr>
<th>Variables</th>
<th>Units</th>
<th>m</th>
<th>s</th>
<th>kg</th>
<th>T</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )</td>
<td>Pressure</td>
<td>−1</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( R )</td>
<td>Resistance</td>
<td>−2</td>
<td>3</td>
<td>−1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>( \mu_S )</td>
<td>Shear modulus</td>
<td>−1</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( L_{rad} )</td>
<td>Spectral radiance</td>
<td>0</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \kappa_{spring} )</td>
<td>Spring constant</td>
<td>0</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \sigma_{den} )</td>
<td>Surface charge density</td>
<td>0</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>( T, T_1, T_2 )</td>
<td>Temperature</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>( \kappa )</td>
<td>Thermal conductivity</td>
<td>1</td>
<td>−3</td>
<td>1</td>
<td>−1</td>
<td>0</td>
</tr>
<tr>
<td>( t, t_1 )</td>
<td>Time</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \tau )</td>
<td>Torque</td>
<td>2</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( A_{vec} )</td>
<td>Vector potential</td>
<td>−1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( \omega, \phi, \chi, \nu )</td>
<td>Velocity</td>
<td>1</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( V, V_1, V_2 )</td>
<td>Volume</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \rho_{p, p_0} )</td>
<td>Volume charge density</td>
<td>−1</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>−1</td>
</tr>
<tr>
<td>( V_c )</td>
<td>Voltage</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( k )</td>
<td>Wave number</td>
<td>−1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( Y )</td>
<td>Young modulus</td>
<td>−1</td>
<td>−2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Table 4. Tested Feynman equations, part 1. Abbreviations in the "Methods used" column: da, dimensional analysis; bf, brute force; pf, polyfit; ev, set two variables equal; sym, symmetry; sep, separability. Suffixes denote the type of symmetry or separability (sym–, translational symmetry; sep*, multiplicative separability; etc.) or the preprocessing before brute force (e.g., bf-inverse means inverting the mystery function before bf).

<table>
<thead>
<tr>
<th>Feynman Eq.</th>
<th>Equation</th>
<th>Solution Time (s)</th>
<th>Methods Used</th>
<th>Data Needed</th>
<th>Solved By Eureqa</th>
<th>Solved W/da</th>
<th>Noise Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>I.6.20a</td>
<td>( f = e^{-q/2}/\sqrt{2\pi} )</td>
<td>16</td>
<td>bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>I.6.20</td>
<td>( f = e^{-x^2}/\sqrt{2\pi} )</td>
<td>2992</td>
<td>ev, bf-log</td>
<td>10^2</td>
<td>No</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>I.6.20b</td>
<td>( f = e^{-a+q^2}/\sqrt{2\pi} )</td>
<td>4792</td>
<td>sym-, ev, bf-log</td>
<td>10^3</td>
<td>No</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L8.14</td>
<td>( d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2} )</td>
<td>544</td>
<td>da, pf-squared</td>
<td>10^2</td>
<td>No</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L9.18</td>
<td>( F = \frac{g m_1 m_2}{(x_1-x_2)^2 + (y_1-y_2)^2 + (z_1-z_2)^2} )</td>
<td>5975</td>
<td>da, sym-, sym-, sep*, pf-inv</td>
<td>10^6</td>
<td>No</td>
<td>Yes</td>
<td>10^{-5}</td>
</tr>
<tr>
<td>L10.7</td>
<td>( m = \frac{m_0}{\sqrt{1-v^2/c^2}} )</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L11.19</td>
<td>( A = x_1y_1 + x_2y_2 + x_3y_3 )</td>
<td>184</td>
<td>da, pf</td>
<td>10^3</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>L12.1</td>
<td>( F = \mu N_0 )</td>
<td>12</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>L12.2</td>
<td>( F = \frac{q_1 q_2}{\alpha^2 r^2} )</td>
<td>17</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>L12.4</td>
<td>( E_{ij} = \frac{q_1 q_2}{\alpha^2 r^2} )</td>
<td>12</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>L12.5</td>
<td>( F = q_2 E_j )</td>
<td>8</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>L12.11</td>
<td>( F = q(E_i + Bv \sin \theta) )</td>
<td>19</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>L13.4</td>
<td>( K = \frac{1}{2} m(v^2 + u^2 + w^2) )</td>
<td>22</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L13.12</td>
<td>( U = G m_1 m_2 \left( \frac{1}{r} - \frac{1}{r_0} \right) )</td>
<td>20</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L14.3</td>
<td>( U = mgz )</td>
<td>12</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>L14.4</td>
<td>( U = \frac{k_{	ext{max}} x^2}{2} )</td>
<td>9</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>L15.3x</td>
<td>( x_1 = \frac{x - ut}{\sqrt{1 - u^2/c^2}} )</td>
<td>22</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>No</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>L15.3t</td>
<td>( t_1 = \frac{r - \omega/c^2}{\sqrt{1 - u^2/c^2}} )</td>
<td>20</td>
<td>da, bf</td>
<td>10^2</td>
<td>No</td>
<td>No</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L15.10</td>
<td>( p = \frac{m_0 v}{\sqrt{1 - v^2/c^2}} )</td>
<td>13</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L16.6</td>
<td>( v_1 = \frac{u x - y v}{1 + u v t/c^2} )</td>
<td>18</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>L18.4</td>
<td>( r = \frac{m_1 r_1 + m_2 r_2}{m_1 + m_2} )</td>
<td>17</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>L18.12</td>
<td>( \tau = RF \sin \theta )</td>
<td>15</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>L18.16</td>
<td>( L = m r v \sin \theta )</td>
<td>17</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>L24.6</td>
<td>( E = \frac{1}{2} m (\omega^2 + \omega^2_0) x^2 )</td>
<td>22</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>L25.13</td>
<td>( V_\nu = \frac{\sigma}{c} )</td>
<td>10</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
</tbody>
</table>

continued on next page
<table>
<thead>
<tr>
<th>Feynman Eq.</th>
<th>Equation</th>
<th>Solution Time (s)</th>
<th>Methods Used</th>
<th>Data Needed</th>
<th>Solved By Eureqa</th>
<th>Solved W/o da</th>
<th>Noise Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>L26.2</td>
<td>$\theta_1 = \arcsin(n \sin \theta_2)$</td>
<td>530</td>
<td>da, bf-sin</td>
<td>$10^3$</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L27.6</td>
<td>$t_f = \frac{1}{\lambda + \frac{1}{2}}$</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L29.4</td>
<td>$k = \frac{n}{r}$</td>
<td>8</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L29.16</td>
<td>$x = \sqrt{x_1^2 + x_2^2 - 2x_1x_2 \cos(\theta_1 - \theta_2)}$</td>
<td>2135</td>
<td>da, sym-</td>
<td>$10^3$</td>
<td>No</td>
<td>No</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>L30.3</td>
<td>$l = l_0 \frac{\sin(n \pi/2)}{\sin(n \pi/2)}$</td>
<td>118</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>L30.5</td>
<td>$\theta = \arcsin(\frac{n}{n_{rad}})$</td>
<td>529</td>
<td>da, bf-sin</td>
<td>$10^2$</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>L32.5</td>
<td>$P = \frac{d^2 a^2}{6 \sin c^2}$</td>
<td>13</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L32.17</td>
<td>$P = \left(\frac{\lambda}{2} \frac{C^2}{D} \frac{B \pi r^2}{3} \frac{\alpha^2}{\alpha^2 - \alpha_0^2}\right)$</td>
<td>698</td>
<td>da, bf-sqrt</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>L34.8</td>
<td>$\omega = \frac{\alpha \delta}{\beta}$</td>
<td>13</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L34.10</td>
<td>$\omega = \frac{\alpha \delta}{\beta}$</td>
<td>13</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>L34.14</td>
<td>$\omega = \frac{1 + \sqrt{c}}{1 - \sqrt{c}}\frac{\alpha_0}{\beta}$</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>L34.27</td>
<td>$E = \frac{\alpha_0}{\beta}$</td>
<td>8</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L37.4</td>
<td>$l = l_1 + l_2 + 2l_1 l_2 \cos \delta$</td>
<td>7032</td>
<td>da, bf</td>
<td>$10^2$</td>
<td>Yes</td>
<td>No</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>L38.12</td>
<td>$r = \frac{4m \hbar^2}{\beta^2}$</td>
<td>13</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L39.10</td>
<td>$E = \frac{1}{2} p V$</td>
<td>8</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L39.11</td>
<td>$E = \frac{1}{\gamma - 1} p V$</td>
<td>13</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>L39.22</td>
<td>$P_V = \frac{n k_b T}{V}$</td>
<td>16</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>L40.1</td>
<td>$n = n_0 e^{-\frac{\alpha}{\beta}}$</td>
<td>20</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L41.16</td>
<td>$L_{rad} = \frac{\hbar a^3}{k^2 c^2 (\alpha^2 - 1)}$</td>
<td>22</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>No</td>
<td>$10^{-5}$</td>
</tr>
<tr>
<td>L43.16</td>
<td>$v = \frac{\sqrt{n_0 V_c}}{d}$</td>
<td>14</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L43.31</td>
<td>$D = \mu k_b T$</td>
<td>11</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>L43.43</td>
<td>$\kappa = \frac{1}{\gamma - 1} \frac{\alpha \delta}{\beta}$</td>
<td>16</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>L44.4</td>
<td>$E = nk_b T \ln \left(\frac{\frac{\alpha}{\beta}}{\gamma}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table><p>ight)$ | 18              | da, bf       | 10          | Yes             | Yes           | $10^{-3}$      |
| L47.23     | $c = \frac{\sqrt{\pi \gamma}}{\beta}$ | 14              | da, bf       | 10          | Yes             | Yes           | $10^{-2}$      |</p>
advantage of not searching the space of symbolic expressions blindly like our brute-force module, but rather with the possibility of a net shift toward more accurate (“fit”) equations. The flip side of this is that if Eureqa finds a fairly accurate yet incorrect formula with a quite different functional form, it risks getting stuck near that local optimum. This reflects a fundamental challenge for genetic approaches to symbolic regression: If the final formula is composed of separate parts that are not summed but combined in some more complicated way (as a ratio, say), then each of the parts may be useless fits on their own and unable to evolutionarily compete.

Dependence on data size

To investigate the effect of changing the size of the dataset, we repeatedly reduced the size of each dataset by a factor of 10 until our AI Feynman algorithm failed to solve it. As seen in Tables 4 and 5, most equations are discovered by the polynomial fit and brute-force methods using only 10 data points. One hundred data points are needed in some cases because the algorithm may otherwise overfit when the true equation is complex, “discovering” an incorrect equation that is too simple.

As expected, equations that require the use of a neural network to be solved need substantially more data points (between $10^2$ and $10^6$) for the network to be able to learn the mystery function accurately enough (i.e., obtaining rms accuracy better than $10^{-3}$). Note that expressions requiring the neural network are typically more complex, so one might intuitively expect them to require larger datasets for the correct equation to be discovered without overfitting, even when using alternate approaches such as genetic algorithms.

Dependence on noise level

Since real data are almost always afflicted with measurement errors or other forms of noise, we investigated the robustness of our algorithm. For each mystery, we added independent Gaussian random noise to its dependent variable $y$, of standard deviation $\epsilon y_{\text{rms}}$ where $y_{\text{rms}}$ denotes the rms $y$ value for the mystery before noise has been added. We initially set the relative noise level $\epsilon = 10^{-2}$ and then repeatedly multiplied $\epsilon$ by 10 until the AI Feynman algorithm could no longer solve the mystery. As seen in Tables 4 and 5, most of the equations can still be recovered exactly with an $\epsilon$ value of $10^{-3}$ or less, while almost half of them are still solved for $\epsilon = 10^{-2}$.

For these noise experiments, we adjusted the threshold for the brute-force and polynomial fit algorithms when the noise level changed, such that not finding a solution at all was preferred over finding an approximate solution. These thresholds were not optimized for each mystery individually, so a better choice of these thresholds might allow the exact equation to be recovered with an even higher noise level for certain equations. In future work, it will also be interesting to quantify performance of the algorithm on data with noise added to the independent variables, as well as directly on real-world data.

Bonus mysteries

The 100 basic mysteries discussed above should be viewed as a training set for our AI Feynman algorithm, since we made improvements to its implementation and hyperparameters to optimize performance. In contrast, we can view the 20 bonus mysteries as a test set, since we deliberately selected and analyzed them only after the AI Feynman algorithm and its hyperparameter settings (Table 2) had been finalized. The bonus mysteries are interesting also by virtue of being substantially more complex and difficult in order to better identify the limitations of our method.

Table 6 shows that Eureqa solved only 15% of the bonus mysteries, while AI Feynman solved 90%. The fact that the success percentage differs more between the two methods for the bonus mysteries than for the basic mysteries reflects the increased equation complexity, which requires our neural network–based strategies for a larger fraction of the cases.

To shed light on the limitations of the AI Feynman algorithm, it is interesting to consider the two mysteries for which it failed. The radiated gravitational wave power mystery was reduced to the form $y = \frac{32a^5(1 + a)}{5b^2} - \epsilon$ by dimensional analysis, corresponding to the string “aaa > > > > bbbbbb * * * * / * /” in reverse Polish notation (ignoring the multiplicative prefactor $-\frac{32}{5}$). This would require about 2 years for the brute-force method, exceeding our allotted time limit. The Jackson 2.11 mystery was reduced to the form $a - \frac{1}{4 \pi b(1 - a^2)}$ by dimensional analysis, corresponding to the string “aPO >>> > > a \backslash aa + < aa + * * / * - / * - /” in reverse Polish notation, which would require about 100 times the age of our universe for the brute-force method.

It is likely that both of these mysteries can be solved with relatively minor improvements of our algorithm. The first mystery would have been solved had the algorithm not failed to discover that $a(1 + a)/b^5$ is separable. The large dynamic range induced by the fifth power in the denominator caused the neural network to miss the separability threshold; potential solutions include temporarily limiting the parameter range or analyzing the logarithm of the absolute value (to discover additive separability).

If we had used different units in the second mystery, where $1/4 \pi \alpha$ was replaced by the Coulomb constant $k$, the costly $4\pi$ factor (requiring seven symbols “PPPP + + + +” or “PO + + + + *”) would have disappeared. Moreover, if we had used a different set of function symbols that included “$Q$” for squaring, then brute force could have discovered that $a - \frac{1}{b(1 - a^2)}$ is solved by “aabaaQ < Q *- “. Similarly, introducing a symbol $\wedge$ denoting exponentiation, enabling the string
Table 5. Tested Feynman equations, part 2 (same notation as in Table 4).

<table>
<thead>
<tr>
<th>Feynman Eq.</th>
<th>Equation</th>
<th>Solution Time (s)</th>
<th>Methods Used</th>
<th>Data Needed</th>
<th>Solved By Eureqa</th>
<th>Solved W/o da Noise</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.2.42</td>
<td>( p = \frac{a(1 - T)}{T^2} )</td>
<td>54</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.3.24</td>
<td>( f_l = \frac{p}{4\pi r_l} )</td>
<td>8</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.4.23</td>
<td>( v_e = \frac{q}{4\pi r} )</td>
<td>10</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.6.11</td>
<td>( v_e = \frac{1}{4\pi r^2} )</td>
<td>18</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.6.15a</td>
<td>( E_f = \frac{3}{4\pi r^2} \sqrt{x^2 + y^2} )</td>
<td>2801</td>
<td>da, sm, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.6.15b</td>
<td>( E_f = \frac{3}{4\pi r^2} \cos \theta \sin \theta )</td>
<td>23</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.8.7</td>
<td>( E = \frac{a^2}{4\pi r^2} )</td>
<td>10</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.8.31</td>
<td>( E_{den} = \frac{eF^2}{2} )</td>
<td>8</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.10.9</td>
<td>( E_f = \frac{\sigma_0}{1 + \frac{T}{k_B T}} )</td>
<td>13</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.11.3</td>
<td>( x = \frac{\partial \rho_0}{\partial (u_0^2 - u_0^2)} )</td>
<td>25</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.11.17</td>
<td>( n = n_0 \left(1 + \frac{\rho_0 \cos \theta}{k_B T} \right)^{-1} )</td>
<td>28</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.11.20</td>
<td>( p_e = \frac{\sigma_0 p_0 E_f}{k_B T} )</td>
<td>18</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.11.27</td>
<td>( p_e = \frac{\sigma_0}{1 - \frac{\mu}{(u_0^2)}} \epsilon )</td>
<td>337</td>
<td>da, bf-inverse</td>
<td>10^2</td>
<td>No</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.11.28</td>
<td>( \theta = 1 + \frac{\mu_0}{1 - (u_0^2)} )</td>
<td>1708</td>
<td>da, sym*, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>II.13.17</td>
<td>( B = \frac{1}{4\pi r^2} )</td>
<td>13</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.13.23</td>
<td>( p_c = \frac{\rho_0 \sqrt{1 - \frac{\mu_0^2}{c^2}}}{\sqrt{1 - v^2/c^2}} )</td>
<td>13</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>II.13.34</td>
<td>( j = \frac{\sigma_0 \sqrt{1 - v^2/c^2}}{\sqrt{1 - \frac{\mu_0^2}{c^2}} \sqrt{1 - \frac{\mu_0^2}{c^2}}} )</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>II.15.4</td>
<td>( E = -\mu_0 B \cos \theta )</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.15.5</td>
<td>( E = -\rho_0 E_f \cos \theta )</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.21.32</td>
<td>( V_e = \frac{q}{4\pi r(1 + v/c)} )</td>
<td>21</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-3} )</td>
</tr>
<tr>
<td>II.24.17</td>
<td>( k = \sqrt{\frac{u_0^2}{c^2} - \frac{v^2}{c^2}} )</td>
<td>62</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>( 10^{-5} )</td>
</tr>
<tr>
<td>II.27.16</td>
<td>( F_E = \epsilon \epsilon_0 E_f )</td>
<td>13</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
<tr>
<td>II.27.18</td>
<td>( E_{den} = \epsilon E_f^2 )</td>
<td>9</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>( 10^{-2} )</td>
</tr>
</tbody>
</table>

continued on next page
for \(a^L\) to be shortened from “\(aLb\) * E” to “\(ab\) ∧ ” would enable brute force to solve many mysteries faster, including Jackson 2.11.

Last, a powerful strategy that could ameliorate both of these failures would be to add symbols corresponding to parameters that are numerically optimized over. This strategy is currently implemented in Eureqa, but not Al Feynman, and could make a useful upgrade as long as it is done in a way that does not unduly slow down the symbolic brute-force search. In summary, the two failures of the Al

<table>
<thead>
<tr>
<th>Feynman Eq.</th>
<th>Equation</th>
<th>Solution Time (s)</th>
<th>Methods Used</th>
<th>Data Needed</th>
<th>Solved By Eureqa</th>
<th>Solved W/o da Tolerance</th>
<th>Noise Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.34.2a</td>
<td>(I = \frac{qV}{2di})</td>
<td>11</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>II.34.2</td>
<td>(\mu_M = \frac{qV}{2})</td>
<td>11</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>II.34.11</td>
<td>(\omega = \frac{g_\lambda b}{2m})</td>
<td>16</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>II.34.29a</td>
<td>(\mu_M = \frac{qL}{4aL})</td>
<td>12</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>II.34.29b</td>
<td>(E = \frac{g_\lambda LdL}{k})</td>
<td>18</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>II.35.18</td>
<td>(n = \exp(\frac{n_0}{\lambda L} + \exp(-\frac{n_0}{\lambda L})))</td>
<td>30</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>II.35.21</td>
<td>(M = n_0 \mu_M \tanh \left( \frac{\mu_M}{\lambda L} \right) )</td>
<td>1597</td>
<td>da, half-input, bf</td>
<td>10</td>
<td>Yes</td>
<td>No</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>II.36.38</td>
<td>(f = \frac{1}{4} \mu_M \mu_1 M )</td>
<td>77</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>II.37.1</td>
<td>(E = \mu_M (1 + \gamma) B)</td>
<td>15</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>II.38.3</td>
<td>(F = \frac{Y d}{\alpha} )</td>
<td>47</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>II.38.14</td>
<td>(\mu_S = \frac{Y}{(1 + n)} )</td>
<td>13</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.4.32</td>
<td>(n = \frac{1}{e^{2(\lambda L - 1)}} )</td>
<td>20</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.4.33</td>
<td>(E = \frac{\mu_M}{e^{2(\lambda L - 1)}} )</td>
<td>19</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.7.38</td>
<td>(\omega = \frac{2 \mu_M}{\lambda L} )</td>
<td>13</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>III.8.54</td>
<td>(\rho_1 = \sin \left( \frac{\rho_1}{\lambda L} \right)^2 )</td>
<td>39</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.9.52</td>
<td>(\rho_2 = \frac{\mu_M}{e^{2(\lambda L - 1)}} )</td>
<td>3162</td>
<td>da, sym-, sm, bf</td>
<td>10</td>
<td>Yes</td>
<td>No</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.10.19</td>
<td>(E = \mu_M (\frac{B_1^2 + B_2^2 + B_3^2}{\lambda L}) )</td>
<td>410</td>
<td>da, bf-squared</td>
<td>10^2</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>III.12.43</td>
<td>(L = \mu_M )</td>
<td>11</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.13.18</td>
<td>(\nu = \frac{2\pi e^k}{\lambda L} )</td>
<td>16</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>III.14.14</td>
<td>(I = \lambda L (e^{2(\lambda L - 1)} - 1) )</td>
<td>18</td>
<td>da, bf</td>
<td>10</td>
<td>No</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.15.12</td>
<td>(E = 2U(1 - \cos (kd)) )</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-4}</td>
</tr>
<tr>
<td>III.15.14</td>
<td>(m = \frac{\lambda L}{2\pi e^k} )</td>
<td>10</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
<tr>
<td>III.15.27</td>
<td>(k = \frac{2\pi m}{nd} )</td>
<td>14</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.17.37</td>
<td>(f = \beta (1 + u\cos \theta) )</td>
<td>27</td>
<td>bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-3}</td>
</tr>
<tr>
<td>III.19.51</td>
<td>(E = \frac{-m^2}{2(\lambda L - 1)^2} )</td>
<td>18</td>
<td>da, bf</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-5}</td>
</tr>
<tr>
<td>III.21.20</td>
<td>(j = \frac{-m^2}{2(\lambda L - 1)^2} )</td>
<td>13</td>
<td>da</td>
<td>10</td>
<td>Yes</td>
<td>Yes</td>
<td>10^{-2}</td>
</tr>
</tbody>
</table>
Table 6. Tested bonus equations. Goldstein 8.56 is for the special case where the vectors $\mathbf{p}$ and $\mathbf{A}$ are parallel.

<table>
<thead>
<tr>
<th>Source</th>
<th>Equation</th>
<th>Solved</th>
<th>Solved by Eureqa</th>
<th>Methods used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rutherford scattering</td>
<td>$A = \left( \frac{Z_1 Z_2 \mu_{\text{eff}}}{4 \pi \sin^2 \frac{\theta}{2}} \right)^2$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf-sqrt</td>
</tr>
<tr>
<td>Friedman equation</td>
<td>$H = \sqrt{\frac{8 k q^2 - k \mu^2}{d^4}}$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf-squared</td>
</tr>
<tr>
<td>Compton scattering</td>
<td>$U = \frac{1 - \cos \theta}{1 + \cos \theta}$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf</td>
</tr>
<tr>
<td>Radiated gravitational wave</td>
<td>$P = \frac{\alpha^2 (m_1 m_2)}{2 c^2} \frac{[m_1 + m_2]}{r^3}$</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>Relativistic aberration</td>
<td>$\theta_1 = \arccos \left( \frac{\cos \theta - 1}{1 - \cos \theta} \right)$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf-cos</td>
</tr>
<tr>
<td>N-slit diffraction</td>
<td>$I = \frac{\sin \theta (2d \sin \frac{\theta}{2})}{\sin (\theta/2)^2}$</td>
<td>Yes</td>
<td>No</td>
<td>da, sm, bf</td>
</tr>
<tr>
<td>Goldstein 3.16</td>
<td>$v = \frac{1}{\sin \theta} \left( E - U - \frac{1^2}{2m_r^2} \right)$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf-squared</td>
</tr>
<tr>
<td>Goldstein 3.55</td>
<td>$k = \frac{m_k}{1^2} \left( 1 + \left( \frac{\alpha^2}{m_k^2} \right) \cos \left( \theta_1 - \theta_2 \right) \right)$</td>
<td>Yes</td>
<td>No</td>
<td>da, sym-, bf</td>
</tr>
<tr>
<td>Goldstein 3.64 (ellipse)</td>
<td>$r = \frac{d^2 - d_0^2}{1 + \sin \alpha (d_0 - d)}$</td>
<td>Yes</td>
<td>No</td>
<td>da, sym-, bf</td>
</tr>
<tr>
<td>Goldstein 3.74 (Kepler)</td>
<td>$t = \frac{2 \alpha^2 q_0^2}{\sqrt{2m_1 m_2}}$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf</td>
</tr>
<tr>
<td>Goldstein 3.99</td>
<td>$a = \sqrt{1 + \frac{2 \alpha^2 q_0^2}{m_1 m_2}}$</td>
<td>Yes</td>
<td>No</td>
<td>da, sym+, bf</td>
</tr>
<tr>
<td>Goldstein 8.56</td>
<td>$E = \sqrt{(p - q A_{\text{eff}})^2 c^2 + m_1^2 c^4 + q V_e}$</td>
<td>Yes</td>
<td>No</td>
<td>da, sep+, bf-squared</td>
</tr>
<tr>
<td>Goldstein 12.80</td>
<td>$E = \frac{1}{2m} \left( p^2 + m_1^2 + \omega^2 x^2 \left[ 1 + \alpha \frac{\omega^2}{2} \right] \right)$</td>
<td>Yes</td>
<td>Yes</td>
<td>da, bf</td>
</tr>
<tr>
<td>Jackson 2.11</td>
<td>$F = \frac{q}{4\pi e r} \left( 4 \pi e V_e d - \frac{d_0 q r^3}{(r^2 - d_0^2)^3} \right)$</td>
<td>No</td>
<td>No</td>
<td>-</td>
</tr>
<tr>
<td>Jackson 3.45</td>
<td>$V_e = \frac{q}{r^2 + d^2 - 2dr \cos \alpha}$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf-inv</td>
</tr>
<tr>
<td>Jackson 4.60</td>
<td>$V_e = E \cos \theta \left( \frac{u - 1}{u + 2 r^2} - f \right)$</td>
<td>Yes</td>
<td>No</td>
<td>da, sep*, bf</td>
</tr>
<tr>
<td>Jackson 11.38 (Doppler)</td>
<td>$\omega_0 = \sqrt{\frac{1 - \frac{v^2}{c^2}}{1 + \frac{v^2}{c^2}}} \omega$</td>
<td>Yes</td>
<td>No</td>
<td>da, cos-input, bf</td>
</tr>
<tr>
<td>Weinberg 15.2.1</td>
<td>$p = \frac{1}{8 m_0} \left( \frac{c^2}{\alpha'} + H^2 \right)$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf</td>
</tr>
<tr>
<td>Weinberg 15.2.2</td>
<td>$p_1 = -\frac{1}{8 m_0} \left( \frac{c^2}{\alpha'} + c^2 H^2 (1 - 2\alpha) \right)$</td>
<td>Yes</td>
<td>No</td>
<td>da, bf</td>
</tr>
<tr>
<td>Schwarz 13.132 (Klein-Nishina)</td>
<td>$A = \frac{m_0 c^2}{m} \left( \frac{1}{\sin^2 \theta} \right)^2 \left[ \frac{1}{2 \alpha} + \frac{\omega^2}{2} - \sin^2 \theta \right]$</td>
<td>Yes</td>
<td>No</td>
<td>da, sym/, sep*, sin-input, bf</td>
</tr>
</tbody>
</table>
Feynman algorithm signal not unsurmountable obstacles, but motivation for further work.

In addition, we tested the performance of our algorithm on the mystery functions presented in (41) (we wish to thank the anonymous reviewer who brought this dataset to our attention). Some equations appear twice; we included them only once. Our algorithm again outperformed Eureqa, discovering 66.7% of the equations, while Eureqa discovered 48.9%. The fact that the AI Feynman algorithm performs less well on this test set than on genuine physics formulas traces back to the fact that most of the equations presented in (41) are rather arbitrary compositions of elementary functions unlikely to occur in real-world problems, thus lacking the symmetries, separability, etc., that the neural network part of our algorithm is able to exploit.

**DISCUSSION**

We have presented a novel physics-inspired algorithm for solving multidimensional analytic regression problems: finding a symbolic expression that matches data from an unknown algebraic function. Our key innovation lies in combining traditional fitting techniques with a neural network-based approach that can repeatedly reduce a problem to simpler ones, eliminating dependent variables by discovering properties such as symmetries and separability in the unknown function. To facilitate quantitative benchmarking of our and other symbolic regression algorithms, we created a freely downloadable database with 100 regression mysteries drawn from the Feynman Lectures on Physics and a bonus set of an additional 20 mysteries selected for difficulty and fame.

**Key findings**

The preexisting state-of-the-art symbolic regression software Eureqa (26) discovered 68% of the Feynman equations and 15% of the bonus equations, while our AI Feynman algorithm discovered 100 and 90%, respectively, including Kepler’s ellipse equation mentioned in the Introduction (third entry in Table 6). Most of the 100 Feynman equations could be solved even if the data size was reduced to merely 10^2 data points or had percent-level noise added, but the most complex equations needing neural network fitting required more data and less noise.

Compared with the genetic algorithm of Eureqa, the most interesting improvements are seen for the most difficult mysteries where the neural network strategy is repeatedly deployed. Here, the progress of AI Feynman over time corresponds to repeatedly reducing the problem to simpler ones with fewer variables, while Eureqa and other genetic algorithms are forced to solve the full problem by exploring a vast search space, risking getting stuck in local optima.

**Opportunities for further work**

Both the successes and failures of our algorithm motivate further work to make it better, and we will now briefly comment on promising improvement strategies. Although we mostly used the same elementary function options (Table 1) and hyperparameter settings (Table 2) for all mysteries, these could be strategically chosen based on an automated preanalysis of each mystery. For example, observed oscillatory behavior could suggest including sin and cos, and lack thereof could suggest saving time by excluding them.

Our code could also be straightforwardly integrated into a larger program discovering equations involving derivatives and integrals, which frequently occur in physics equations. For example, if we suspect that our formula contains a partial differential equation, then the user can simply estimate various derivatives from the data (or its interpolation, using a neural network) and include them in the AI Feynman algorithm as independent variables, thus discovering the differential equation in question.

We saw how, even if the mystery data have very low noise, substantial de facto noise was introduced by imperfect neural network fitting, complicating subsequent solution steps. It will therefore be valuable to explore better neural network architectures, ideally reducing fitting noise to the 10^{-6} level. This may be easier than in many other contexts, since we do not care whether the neural network generalizes poorly outside the domain where we have data: As long as it is highly accurate within this domain, it serves our purpose of correctly factorizing separable functions, etc.

Our brute-force method can be better integrated with a neural network search for hidden simplicity. Our implemented symmetry search simply tests whether two input variables $a$ and $b$ can be replaced by a bivariate function of them, specifically $+$, $-$, $\ast$, or $/$, corresponding to length 3 strings “$ab+$”, “$ab−$”, “$ab\ast$”, and “$ab/$”. This can be readily generalized to longer strings involving two or more variables, for example, bivariate functions $ab^2$ or $e^a \cos b$.

A second example of improved brute-force use is if the neural network reveals that the function can be exactly solved after setting some variable $a$ equal to something else (say zero, one, or another variable). A brute-force search can now be performed in the vicinity of the discovered exact expression: For example, if the expression is valid for $a = 0$, the brute-force search can insert additive terms that vanish for $a = 0$ and multiplicative terms that equal unity for $a = 0$, thus being likely to discover the full formula much faster than an unrestricted brute-force search from scratch.

Last but not least, it is likely that marrying the best features from both our method and genetic algorithms can spawn a method that outperforms both. Genetic algorithms such as Eureqa perform quite well even in the presence of substantial noise, whether they output not merely one hopefully correct formula, but rather a Pareto frontier, a sequence of increasingly complex formulas that provide progressively better accuracy. Although it may not be clear which of these formulas is correct, it is more likely that the correct formula is one of them than any particular one that an algorithm might guess. When our neural network identifies separability, a so generated Pareto frontier could thus be used to generate candidate formulas for one factor, after which each one could be substituted back and tested as above, and the best solution to the full expression would be retained. Our brute-force algorithm can similarly be upgraded to return a Pareto frontier instead of a single formula.

In summary, symbolic regression algorithms are getting better and are likely to continue improving. We look forward to the day when, for the first time in the history of physics, a computer, just like Kepler, discovers a useful and hitherto unknown physics formula through symbolic regression!

**MATERIALS AND METHODS**

The materials used for the symbolic regression tests are all in the FSReD, available at https://space.mit.edu/home/tegmark/ aifeynman.html. The method by which we have implemented our algorithm is as a freely available software package made available at...
Algorithm 1 AI Feynman: Translational symmetry

Require Dataset $D = \{(x, y)\}$.
Require net: trained neural network
Require $\text{NN}_{\text{error}}$: the neural network validation error

$a = 1$

for $i$ in len($x$) do:
for $j$ in len($x$) do:
if $i < j$:
\[
x_t = x
\]
\[
x_t[i] = x_t[i] + a
\]
\[
x_t[j] = x_t[j] + a
\]
error = RMSE(net($x$),net($x_t$))
error = error/RMSE(net($x$))
if error $< 7 \times \text{NN}_{\text{error}}$
\[
x_{eq}[i] = x_{eq}[i] - x_t[j]
\]
x = delete($x_t$)
return $x_t$, $i$, $j$

Algorithm 2 AI Feynman: Additive separability

Require Dataset $D = \{(x, y)\}$
Require net: trained neural network
Require $\text{NN}_{\text{error}}$: the neural network validation error

$x_{eq} = x$

for $i$ in len($x$) do:
\[
x_{eq}[i] = \text{mean}(x[i])
\]
for $i$ in len($x$) do:
$c = \text{combinations}[1,2,\ldots,\text{len}(x)],i$
for idx1 in c do:
\[
x_1 = x
\]
\[
x_2 = x
\]
\[
idx2 = k \in [1,\text{len}(x)] \text{ not in idx1}
\]
for $j$ in idx1:
\[
x_1[j] = \text{mean}(x[j])
\]
for $j$ in idx2:
\[
x_2[j] = \text{mean}(x[j])
\]
error = RMSE(net($x$),net($x_1$) + net($x_2$)-net($x_{eq}$))
error = error/RMSE(net($x$))
if error $< 10 \times \text{NN}_{\text{error}}$
\[
x_1 = \text{delete}(x_1,\text{index}_2)
\]
x2 = delete($x_2$,$\text{index}_2$)
return $x_1$, $x_2$, $\text{index}_1$, $\text{index}_2$

REFERENCES AND NOTES

37. H. Goldstein, C. Poole, J. Safko, Classical Mechanics (Addison Wesley, 2002); https://books.google.com/books?id=4uCuQgAACAAJ.
Acknowledgments: We thank R. Domingos, Z. Dong, M. Skuhersky, A. Tan, and T. Wu for the helpful comments, and the Center for Brains, Minds, and Machines (CBMM) for hospitality.

Funding: This work was supported by The Casey and Family Foundation, the Ethics and Governance of AI Fund, the Foundational Questions Institute, the Rothenberg Family Fund for Cognitive Science, and the Templeton World Charity Foundation Inc. The opinions expressed in this publication are those of the authors and do not necessarily reflect the views of the Templeton World Charity Foundation Inc. Author contributions: Concept, supervision, and project management: M.T. Design of methodology, programming, experimental validation, data curation, data analysis, validation, and manuscript writing: S.-M.U. and M.T. Competing interests: The authors declare that they have no competing interests. Data and materials availability: All data needed to evaluate the conclusions in the paper are present in the paper, at https://space.mit.edu/home/tegmark/ai_feynman.html, and at https://github.com/SJ001/AI-Feynman. Any additional datasets, analysis details, and material recipes are available upon request.

Submitted 7 June 2019
Accepted 3 January 2020
Published 15 April 2020
10.1126/sciadv.aay2631

AI Feynman: A physics-inspired method for symbolic regression
Silviu-Marian Udrescu and Max Tegmark

Sci Adv 6 (16), eaay2631,
DOI: 10.1126/sciadv.aay2631