Applying machine learning to scientific computing problems

Problem:

Numerical algorithms approximate the solution to a continuous problem that is usually unsolvable in closed form. Due to the inherent limitations of finite precision arithmetic, these approximations are usually discrete in nature in that the continuous problem is approximated at certain (or using) discrete points in some space. For instance, time integration methods approximate the unknown solution to an initial value problem (eq 1.) at certain discrete time points in the interval of integration.

$$\frac{dy}{dt} = f(y);\ y(t_0) = y_0;\ t_0 \leq t \leq t_f \qquad (1)$$

Time stepping methods to solve (eq 1.) start with the approximation known at a time t_0 and find subsequent approximations at future time points by stepping through time. A straightforward example is the forward Euler method given below

$$y_{n+1} = y_n + h f(y_n),$$

which computes the future solution as a combination of the current known solution and the slope at the current time point scaled by a step size h.

Since these algorithms approximate the true solution, it is essential to both measure and control error to ensure that the approximation does not grow unbounded or in other words, deviate too much from the "true" solution. Several authors have addressed automatic step size control as a mechanism to control error for time integration methods[1] [2] [3]. Each time a solution is computed for a future time point, these methods build two separate estimates, each of a different accuracy level. Then they either accept or reject the solution at the future time step depending on the norm of the difference between the two estimates. This works well in practice in order to control the error. If the step is accepted, then future solutions are computed using the solution at the recently computed time step. If a step is rejected, then recomputation of the solution with a smaller time step is needed.

The drawback of these error control mechanisms is that they are built to be general and rely only on immediate information to decide the step size for future time steps. In long running

---

[1] Gear, C.W., 1971. The automatic integration of ordinary differential equations. *Communications of the ACM*, *14*(3), pp.176-179.

[2] Hairer, E., Nørsett, S.P. and Wanner, G., 1993. Solving ordinary differential equations. I, volume 8 of Springer Series in Computational Mathematics.

[3] Shampine, L.F. and Watts, H.A., 1979. The art of writing a Runge-Kutta code. II. *Applied Mathematics and Computation*, *5*(2), pp.93-121.

simulations, rejected time steps constitute wasted computations and resources. On the other hand each step acceptance or rejection can give us valuable information about the problem being solved, and this can be used to construct more reliable estimates of step size schedule.

In this project, I build a novel error control mechanism that fits a binarized error surface supported on time step size and a fraction of the states observed during the integration process. The new error control mechanism uses information from both accepted and rejected steps to implicitly estimate the largest possible safe step size that will increase the computational efficiency of the algorithm while reducing wasted computations due to rejected steps.

In an earlier work, Snoek et al. have applied machine learning to model a duration function that is trained alongside in a bayesian optimization framework. The model is subsequently used to predict expected improvement per second for each point in the search space and to select points in the exploration process that give the highest expected improvement per second[4].

Data/Algorithms:

The data for fitting the error surface is that which is generated during the time integration process making it an online algorithm that trains alongside time integration. The idea is to make use of the error estimators that are already in use to control error and obtain the training signal for the supervised learning process. As training happens in each timestep, the model is used to make a prediction about the acceptance or rejection of the future timestep. When the prediction accuracy surpasses a certain threshold over a window of past predictions, the model is repeatedly queried until the largest possible safe step size is determined.

I implemented a well-known Runge-Kutta method[5] as the time integration method. Also included in my repository is a modified implementation that does the training of the machine learning model to fit the binarized error surface. A number of control variables were introduced in the process of building the modified implementation. These are described below:

- *Number of Observations (o)*: This is the number of components of the state vector to observe while fitting the error surface, and is one less than the length of the feature vector. The observations are equally distributed across the state vector.
- *Size of the Validation Window (w)*: In order to validate the model during training, we predict if a future step will be accepted or not by querying the model using the future step size and the observations derived from the current state vector as the feature vector. The validation window is a collection of boolean entries indicating if the prediction turned out to be correct or not. The model's prediction is regarded useful only when the

[4] Snoek, J., Larochelle, H. and Adams, R.P., 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems* (pp. 2951-2959).
[5] Shampine, L.F. and Watts, H.A., 1979. The art of writing a Runge-Kutta code. II. *Applied Mathematics and Computation*, *5*(2), pp.93-121.

prediction accuracy exceeds a certain validation threshold, and in such a scenario, step size is shrunk if the prediction is a “reject”.

- *Validation Threshold (p)*: The entries in the validation window are replaced in a circular fashion, thus taking into consideration only *w*-most recent predictions when validation accuracy is computed, where *w* is the size of the validation window. The validation threshold is the proportion of entries in the validation window that needs to be correct when using the model for prediction.
- *Shrink Factor (f)*: A constant factor by which the step size is shrunk if the model is deemed to be sufficiently accurate (validation accuracy > validation threshold) and the model predicts that the subsequent step will be a “reject”.
- *Max number of Shrinks (m)*: The algorithm shrinks the step size by a shrink factor, *f*, a maximum number of times, *m*, in a loop, querying the sufficiently accurate model in each iteration whether the new step size will be accepted or not starting at the current state.
- *Number of Samples (k)*: The model is trained in each iteration using the observations and step sizes from *k* most recently successful and *k* most recently unsuccessful integration trials. This number *k* is one half the number of samples used in the training process. Currently, an equal proportion of successful and unsuccessful samples are used in training.

By using the *k* most recently successful/unsuccessful observations in training, the model adapts to the region of integration under consideration. In order to make the model more relevant to the region of integration, the entries in the validation window could be weighted with the most recent entries receiving a higher weight. This is, however, not being done at the moment, with validation accuracy computed as the mean of the validation window having 1 or 0 entries signifying a correct or incorrect prediction respectively.

Evaluation:

To evaluate the success of this strategy, I used a stiff partial differential equation as the test case. Explicit methods face a number of rejected time steps, when dealing with stiff systems, as they try to track the stiffest modes in the solution. The expected outcome is that the number of rejected time steps will be lesser with the new error estimator/step size predictor. It is possible, however, that the computation time may increase for problems that are not sufficiently big.

Results:

The test problem is the Allen-Cahn reaction diffusion equation[6] given as follows:

$$u_t = \alpha \nabla^2 u + \gamma (u - u^3), \quad (x,y) \in [0, 1] \times [0, 1], \quad t \in [0, 0.3]$$

[6] Allen, S.M. and Cahn, J.W., 1979. A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening. *Acta Metallurgica*, *27*(6), pp.1085-1095.

The first term on the right hand-side is the diffusion term and the second, the reaction term. The problem can be made stiff by increasing the resolution of the spatial grid. The spatial grid resolution was set to 128 x 128 making the total number of states equal 16384. The initial and boundary conditions across all the experiments are kept the same.

In the table below, I list the outcome of integrating the Allen-Cahn equation with a standard variable step-size Runge-Kutta method and with the modified Runge-Kutta method that trains an embedded machine learning model online. I used two different linear models and their default settings that come prepackaged with scikit-learn library: SGDClassifier and Perceptron.

	Number of observed variables (o)	Number of training samples (k)	Number of accepted steps	Number of rejected steps	Total number of steps
Runge-Kutta	-	-	10378	985	11363
Runge-Kutta with SGDClassifier	1% of 16384 ≈ 163	10	95425	7	95432
	25	10	11374	748	12122
	25	5	11064	813	11877
Runge-Kutta with Perceptron	25	10	10402	971	11373
	25	5	10491	1004	11495

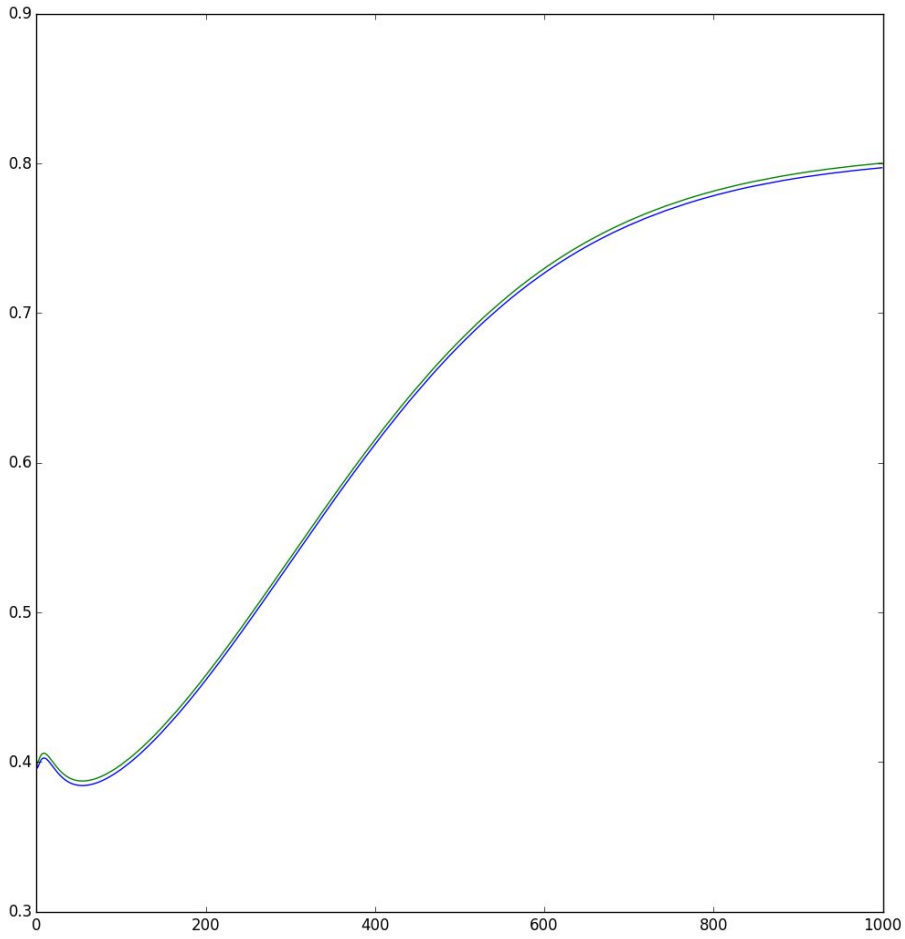
The following observations can be made from the table above:

1. The row in red corresponds to observing 163 components of the entire system, spread evenly across the state vector. Although it shows significantly lesser number of rejected steps, I had to terminate the execution because it hadn't finished integrating the system after ~10 hours of computing. Even though only about 1% of the total number of variables were being observed (feature size - 1), large feature vectors significantly hamper the performance. The time integration method presumably spent a lot of time fitting the model. Furthermore, the model turned out to be quite pessimistic, shrinking the step size every time, resulting in a large number of accepted steps and significantly lesser number of rejected steps when compared to the baseline Runge-Kutta method.
2. All of the other experiments had fewer rejected time steps than the baseline Runge-Kutta method, excluding Runge-Kutta with Perceptron as the training model and a sample size of 5, which had a higher number of rejected steps.
3. All experiments which included a training model inside the method, took more total number of steps to integrate the model than the baseline Runge-Kutta integrator. This could be attributed to the fact that the model is, currently, only used to shrink the step

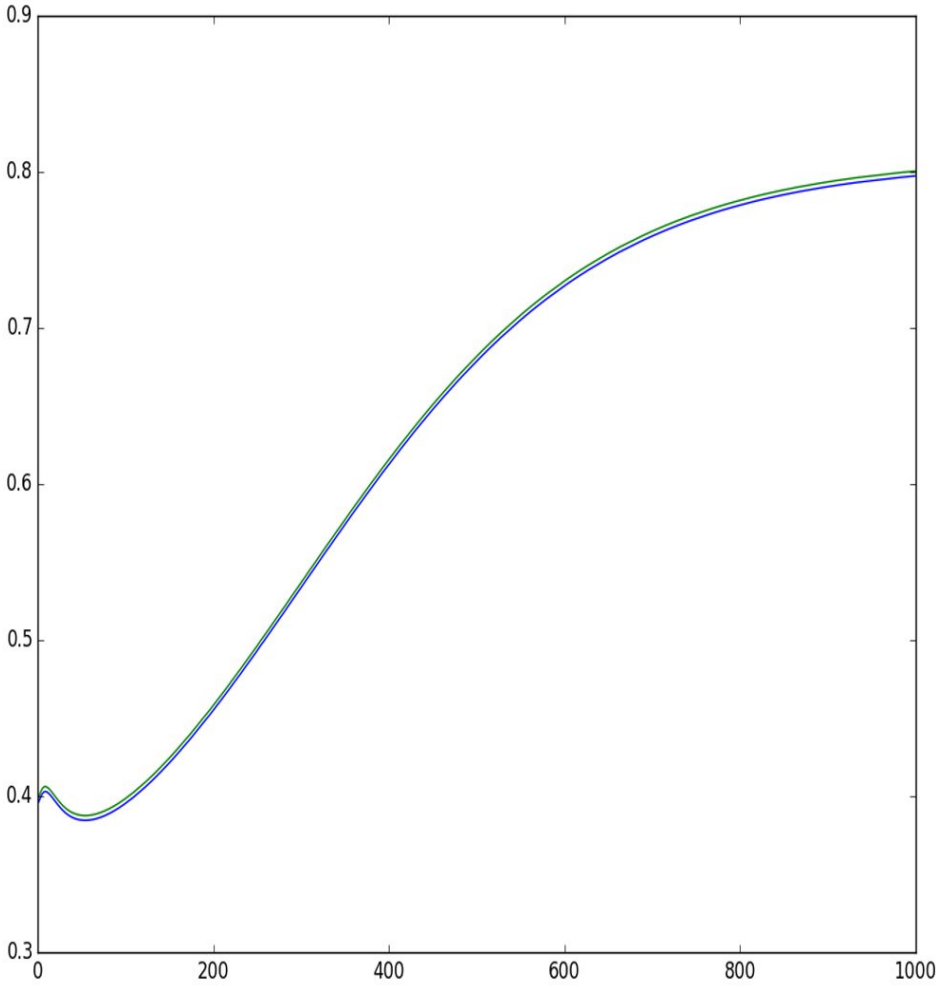
size leading to a larger number of accepted time steps. This could possibly be alleviated by allowing the integrator to also grow the step size if the model predicts that the future step may be accepted.

Plots:

Unfortunately, I didn't save the solutions from the different methods to compute the error norm relative to the baseline Runge-Kutta method. I do, however, have a plot of the first two variables of the state vector that suggest that the integrators agree to a large extent on the evolution of the first two variables, on visual inspection.



Plot of evolution of first two variables using baseline Runge-Kutta method



Plot of evolution of first two variables of state vector using Runge-Kutta with SGDClassifier, when observing 25 components of state vector and using 10 training samples in each training phase.

Plots for other integrators are quite similar to the ones shown above and I am not including them in the interest of brevity.

Conclusion:

Novel error control mechanisms were developed for time integration methods where a machine learning model is trained alongside the time integration process. Trained model is used to predict step acceptance or rejection. A window of validity of past predictions is maintained to

compute validation accuracy. When the validation accuracy of trained model exceeds a validation threshold, the model is used to shrink future step size if the model predicts that the future step may be rejected by the time integration process. Experiments reveal that fewer steps are rejected with the model in place as against the baseline method. Since the model is, currently, only used to shrink step sizes, the total number of steps increases. This could be alleviated by allowing the integration method to also grow step sizes if the model predicts that the future step will be accepted. It is then possible that fewer total number of steps may be required to perform time integration when compared to baseline method.

More experiments are required to arrive at conclusive results. This was not possible because of the number of control variables involved; and each execution was taking between 60 to 120 minutes due to the size of the problem. Reducing the size of the problem will result in simpler systems that may involve significantly fewer rejected (or even total) time steps making them less interesting to study.

Repository url: https://github.com/mod0/optml