DISCRETE AND CONTINUOUS
DYNAMICAL SYSTEMS SERIES B
Volume **22**, Number **7**, September **2017**

doi:10.3934/dcdsb.2017133

pp. **2731–2761**

ADAPTIVE METHODS FOR STOCHASTIC DIFFERENTIAL EQUATIONS VIA NATURAL EMBEDDINGS AND REJECTION SAMPLING WITH MEMORY

Christopher Rackauckas

Department of Mathematics, Center for Complex Biological Systems
University of California, Irvine
CA 92697, USA

Qing Nie*

Department of Mathematics, Center for Complex Biological Systems
University of California, Irvine
CA 92697, USA

(Communicated by Yuan Lou)

Abstract. Adaptive time-stepping with high-order embedded Runge-Kutta pairs and rejection sampling provides efficient approaches for solving differential equations. While many such methods exist for solving deterministic systems, little progress has been made for stochastic variants. One challenge in developing adaptive methods for stochastic differential equations (SDEs) is the construction of embedded schemes with direct error estimates. We present a new class of embedded stochastic Runge-Kutta (SRK) methods with strong order 1.5 which have a natural embedding of strong order 1.0 methods. This allows for the derivation of an error estimate which requires no additional function evaluations. Next we derive a general method to reject the time steps without losing information about the future Brownian path termed Rejection Sampling with Memory (RSwM). This method utilizes a stack data structure to do rejection sampling, costing only a few floating point calculations. We show numerically that the methods generate statistically-correct and tolerance-controlled solutions. Lastly, we show that this form of adaptivity can be applied to systems of equations, and demonstrate that it solves a stiff biological model 12.28x faster than common fixed timestep algorithms. Our approach only requires the solution to a bridging problem and thus lends itself to natural generalizations beyond SDEs.

1. **Introduction.** Explicit methods for solving Ordinary Differential Equations (ODEs) with adaptive time-stepping algorithms, such as the Dormand-Prince and Cash-Karp algorithms, have become efficient and popular solvers for numerical simulations of ODEs due to their ease of use and accuracy [8, 24, 6]. These algorithms consist of two major components: an embedded higher-order and lower-order pair of temporal integrators to allow for estimating the local error (i.e. the error indicator), and an adaptive time-stepping algorithm for choosing the next stepsize and performing the update [21, 1]. Together, the algorithm is able to effectively

2010 *Mathematics Subject Classification.* Primary: 65C30, 60H10; Secondary: 68P05.

Key words and phrases. Stochastic differential equations, adaptive methods, rejection sampling, embedded algorithms, stochastic Runge-Kutta, strong approximation.

* Corresponding author.

solve most equations by automatically adapting to the solutions by adjusting the timestep.

To derive adaptive algorithms for stochastic ordinary differential equations (SODEs) of the form

$$dX_t = f(t, X_t)dt + g(t, X_t)dW_t, \tag{1}$$

where f is the drift coefficient and g is the diffusion coefficient, a natural choice is to use embedded Stochastic Runge-Kutta (SRK) methods. Kloeden and Platen developed a set of stochastic Taylor expansions which allowed for the derivation of higher order (greater than order 1.0) SRK methods [15, 13]. One interesting feature of this class of algorithms is that, since no analytical solution exists for iterated stochastic integrals, they are usually replaced by sufficiently exact approximations due to Wiktorsson [25, 23]. While strong order 2.0 SRK methods have been studied [3, 15], existing works mainly focus on strong order 1.5 Stochastic Runge-Kutta methods, such as a series of methods for Stratanovich SDEs [4, 2] and for Ito SDEs [14].

One recent advance in SRK methods is Rößler's methods for Ito SDEs [22]. This class of methods was shown to be efficient by reducing the number of function evaluations required per step. Their format via extended Butcher tableaus allows them to be easily amendable. However, these methods have not been used for adaptive algorithms. Here, we will show that for certain Rößler-SRK methods, there exists a naturally embedded order 1.0 method such that no extra function evaluations are needed in order to evaluate its difference from the order 1.5 method, presenting an efficient way to both perform the steps and approximate the error.

With the embedded SRK methods, the next challenge is to deal with the choice of time steps. For ODEs, one usually performs a trial step in order to calculate an error estimate and, using this estimate, either choose to step or reject the step and try again with a smaller timestep. The problem with such an approach for SODEs is that when performing the trial step, one takes a random number to simulate the future Brownian path, and if one rejects the timestep, this will change the sample properties of the Brownian path [5]. This is particularly an issue because larger variations in the Brownian path are correlated with more numerical error and thus such an algorithm would have a strong tendency to reduce the variance of the noise process. Lastly, it's not clear how to effectively sub-sample a solution along a Brownian path after one decides it's too coarse.

Most of the current approaches for adaptive time steps attempt to perform error calculations before stepping in order circumvent rejection sampling. One approach [9] utilizes an order 1.0 algorithm with a Brownian path stored at the h, $\frac{h}{2}$, $\frac{h}{4}$ time-points, etc. stored as a tree. The algorithm solves for an error equation using a stochastic Taylor series and then utilizes this new equation to solve a linearization backwards in time in order to produce an error estimate for a future timestep without having to do rejection sampling. This method requires using halving/doubling of stepsizes and, since it's not able to account for which timesteps the Brownian path has the largest future jumps, has to be conservative in order to achieve its chosen error estimate. It also requires the user to provide many derivatives of the drift and diffusion functions f and g, or alternatively perform a numerical estimation of these derivatives at each timestep. Lastly, in their paper they proved that any variable stepsize implementation must use a numerical method which has strong order

of at least 1, meaning that any variable stepsize method must go beyond Euler-type methods.

Another algorithm may be limited to low-order methods due to its requirement of having a small "fixed number of observations", and requires the calculation of derivatives of both f and g [19]. Lamba [16] presented an order 1.0 algorithm which used properties of the Brownian bridge restricted to a grid of multiples of $\frac{1}{3}$. It included the novel property of having separate error estimators for deterministic and noise induced error for finer control of the solution. There is one adaptive stepping algorithm which does not have restrictions on the stepsize and utilizes embedded strong order 1.0/1.5 SRK methods for Stratanovich SDEs [5] . Their embedded method requires two additional function evaluations in order to derive an error estimate, and requires solving for a dense matrix and performing a matrix multiplication each time an interval is subdivided. Their results show that general time-stepping tends to be vastly more efficient than grid-based methods from before.

The main results of the paper are the development of strong order 1.0/1.5 embedded Runge-Kutta pairs for error estimation and a new adaptive time-stepping algorithm termed Rejection Sampling with Memory (RSwM). This extension to the rejection sampling algorithm is diagrammed in Figure 1 and gives an overview of how the various developments come together to form one complete algorithm.

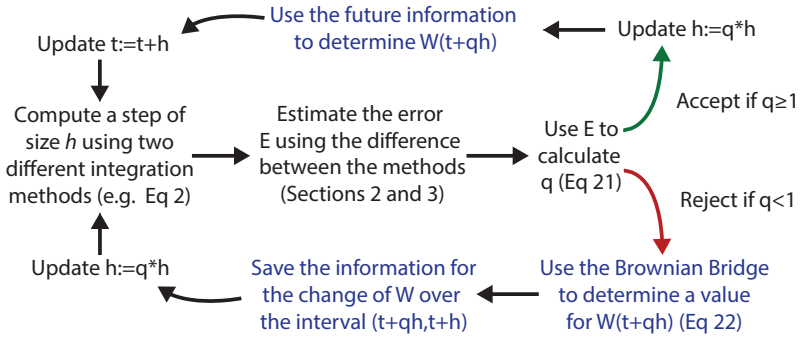

FIGURE 1. Outline of the adaptive SODE algorithm based on Rejection Sampling with Memory. This depicts the general schema for the Rejection Sampling with Memory algorithm (RSwM). The green arrow depicts the path taken when the step is rejected, whereas the red arrow depicts the path that is taken when the step is accepted. The blue text denotes steps which are specific to the stochastic systems in order to ensure correct sampling properties which are developed in section 4.

In section 2 we construct the method ESRK1 which is a strong order 1.0/1.5 embedded pair of algorithms which allows for high-order solving and error estimation. In section 3 we prove the existence of a natural order 1.0/1.5 embedded pair of Rößler SRK algorithms that allows for the efficient calculation of an error estimate for high-order SRK methods with no additional function evaluations. Using this error estimate, we construct the rejection sampling methods for SDEs which allows for efficient general adaptive time-stepping with almost no extra floating

point operations. In section 4 we first elucidate a simple solution to general adaptive time-stepping, discuss how one could attempt to improve the efficiency of the algorithms, and then develop more efficient adaptive stepping algorithms.

Lastly, we perform numerical experiments to show the effectiveness of the algorithms. In section 5, we show that our methods are able to reproduce the proper sample statistics for the stochastic differential equations and the user chosen local error parameter ϵ is able to control the global error. Then in section 6, we demonstrate the ability for the adaptive parameters to control for the coarseness of the solution to nonlinear systems of equations, and show that when applied to a large stiff system derived from biology, our method is able to stability solve a Monte Carlo experiment of 10000 simulations without diverging 12.28x faster than a few popular fixed timestep methods. We conclude by discussing how these algorithms can be generalized to other problems like Stratanovich SDEs, jump diffusions, and SPDEs.

2. Construction of an order 1.0/1.5 embedded pair.

Using a colored root tree analysis, Rößler developed a set of strong order 1.5 stochastic Runge-Kutta schemes [22]. This multi-step method resulted in less computational steps than the KPS schemes, and the number of steps grows much slower as the Ito dimension increases than in the KPS schemes [15]. The structure of Rößler methods are relatively simple, making the method both easy to implement and fast in runtime. For simplicity, we describe the equations for a single Ito dimension, though the results generalize to the case of multiple Ito dimensions.

The Runge-Kutta methods are strong order 1.5 if they satisfy a set of order conditions in A, taking the following form:

$$\begin{aligned} U_{n+1} &= U_n + \sum_{i=1}^{s} \alpha_i f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\Delta t + \\ &\sum_{i=1}^{s}\left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)}\frac{I_{(1,1)}}{\sqrt{\Delta t}} + \beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right), \end{aligned} \tag{2}$$

with stages

$$\begin{aligned} H_i^{(0)} &= U_n + \sum_{j=1}^{s} A_{ij}^{(0)} f\left(t_n + c_j^{(0)}\Delta t, H_j^{(0)}\right)\Delta t \\ &\quad + \sum_{j=1}^{s} B_{ij}^{(0)} g\left(t_n + c_j^{(1)}\Delta t, H_j^{(1)}\right)\frac{I_{(1,0)}}{\Delta t}, \end{aligned} \tag{3}$$

$$\begin{aligned} H_i^{(1)} &= U_n + \sum_{j=1}^{s} A_{ij}^{(1)} f\left(t_n + c_j^{(0)}\Delta t, H_j^{(0)}\right)\Delta t \\ &\quad + \sum_{j=1}^{s} B_{ij}^{(1)} g\left(t_n + c_j^{(1)}\Delta t, H_j^{(1)}\right)\sqrt{\Delta t}. \end{aligned} \tag{4}$$

The iterated stochastic integrals $I_{(1,1)}$, $I_{(1,0)}$, and $I_{(1,1,1)}$ are evaluated via the approximations due to Wiktorsson as noted in C. These methods are referred to as the SRI algorithms and are indicated by the tuple of 44 coefficients $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$. Similarly, for additive noise the Runge-Kutta methods are strong order 1.5 if they

satisfy the order conditions contained B, taking the form

$$\begin{aligned} U_{n+1} &= U_n + \sum_{i=1}^{s} \alpha_i f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\Delta t + \\ &\quad \sum_{i=1}^{s}\left(\beta_i^{(1)} I_{(1)} + \beta_i^{(2)}\frac{I_{(1,0)}}{\Delta t}\right) g(t_n + c_i^{(1)}\Delta t), \end{aligned} \tag{5}$$

with stages

$$H_i^{(0)} = U_n + \sum_{j=1}^{s} A_{ij}^{(0)} f\left(t_n + c_j^{(0)}\Delta t, H_j^{(0)}\right)\Delta t + \sum_{j=1}^{s} B_{ij}^{(0)} g\left(t_n + c_j^{(1)}\Delta t\right)\frac{I_{(1,0)}}{\Delta t}. \tag{6}$$

Given the efficiency of order 1.5 SRK methods for stochastic problems, we will develop Runge-Kutta-Fehlburg-like embedded algorithms [1], which step along an order 1.5 path using error estimates from an order 1.0 path.

We start with the order 1.5 method SRIW1 [22] denoted by coefficients $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$. Notice that the order 1.0 conditions from A only enforce ${\beta^{(3)}}^T B^{(1)} e = 0$ and ${\beta^{(4)}}^T B^{(1)} e = 0$ on $\beta^{(3)}$ and $\beta^{(4)}$. Thus let $\tilde{\beta}^{(3)} = \tilde{\beta}^{(4)} = 0$. This gives an order 1.0 method $\left(A^{(i)}, B^{(i)}, \tilde{\beta}^{(i)}, \alpha\right)$.

Let $\bar{X}$ be the numerical solution of SRIW1 on the general SODE Equation 1, and $\tilde{X}$ be the numerical solution from the algorithm with coefficients $\left(A^{(i)}, B^{(i)}, \tilde{\beta}^{(i)}, \alpha\right)$. By Equation 2, the difference between the numerical solutions on the next step is

$$\bar{X} - \tilde{X} = \sum_{i=1}^{s}\left(\beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right). \tag{7}$$

Note that we could keep $\beta^{(3)}$ not equal to zero and still have an order 1.0 method, but zeroing $\beta^{(3)}$ gives an estimate for the size of ΔZ (the second Brownian path used to estimate $I_{(1,0)}$ as noted in C) which may need to be controlled as well.

However, this error estimate does not take into account stiffness in the drift coefficient f. Notice that the only condition on the coefficients α is $\sum \alpha_i = 1$. Thus let $\tilde{\alpha} = \left(\frac{1}{2}, \frac{1}{2}, 0, 0\right)$. This gives another order 1.0 Method $(A_0, B_0, \beta_i, \tilde{\alpha})$. If we let $\hat{X}$ be the numerical solution to Equation 1 using the coefficients $(A_0, B_0, \beta_i, \tilde{\alpha})$, then by Equation 2 notice

$$\bar{X} - \hat{X} = -\frac{\Delta t}{6} f\left(t_n + c_1^{(0)}\Delta t, H_1^{(0)}\right) + \frac{\Delta t}{6} f\left(t_n + c_2^{(0)}\Delta t, H_2^{(0)}\right). \tag{8}$$

Notice that for this algorithm the difference is only due to the stiffness of the drift coefficient. Using both the β and the α changes together gives a pair of strong order 1.0 / order 1.5 methods which is robust to errors in both coefficients. We designate this algorithm Embedded SRK 1 (ESRK1) as the set of coefficients defined in Table 1 for which steps are taken according to Equation 2, Equation 3, Equation 4.

Adding the two previous error equations Equation 7 and Equation 8, gives E, the error estimation via the difference between the order 1.5 and order 1.0 method, as

$$\begin{aligned} E &= -\frac{\Delta t}{6} f\left(t_n + c_1^{(0)}\Delta t, H_1^{(0)}\right) + \frac{\Delta t}{6} f\left(t_n + c_2^{(0)}\Delta t, H_2^{(0)}\right) \\ &\quad + \sum_{i=1}^{s}\left(\beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right). \end{aligned} \tag{9}$$

$c^{(0)}$	$A^{(0)}$	$B^{(0)}$	
$c^{(1)}$	$A^{(1)}$	$B^{(1)}$	
	α^T	${\beta^{(1)}}^T$	${\beta^{(2)}}^T$
		${\beta^{(3)}}^T$	${\beta^{(4)}}^T$
	$\tilde{\alpha}^T$	${\tilde{\beta}^{(3)}}^T$	${\tilde{\beta}^{(4)}}^T$

(A) Legend Table

0											
$\frac{3}{4}$	$\frac{3}{4}$				$\frac{3}{2}$						
0	0	0			0	0					
0	0	0	0		0	0	0				
0											
$\frac{1}{4}$	$\frac{1}{4}$				$\frac{1}{2}$						
1	1	0			-1	0					
$\frac{1}{4}$	0	0	$\frac{1}{4}$		-5	3	$\frac{1}{2}$				
	$\frac{1}{3}$	$\frac{2}{3}$	0	0	-1	$\frac{4}{3}$	$\frac{2}{3}$	0	-1	$\frac{4}{3}$	$-\frac{1}{3}$	0
					2	$-\frac{4}{3}$	$-\frac{2}{3}$	0	-2	$\frac{5}{3}$	$-\frac{2}{3}$	1
	$\frac{1}{2}$	$\frac{1}{2}$	0	0	0	0	0	0	0	0	0	0

(B) Coefficients Table

TABLE 1. ESRK1. Table (a) shows the legend for how the numbers in in Table (b) correspond to the coefficient arrays/matrices $c^{(i)}, A^{(i)}, B^{(i)}, \alpha, \beta^{(i)}, \tilde{\alpha}$, and $\tilde{\beta^{(i)}}$. For example, these tables show that $\alpha^T = (\frac{1}{3}, \frac{2}{3}, 0, 0)$. Note that the matrices $A^{(i)}$ and $B^{(i)}$ are lower triangular since the method is explicit.

Thus ESRK1 is a pair of order 1.0/order 1.5 embedded algorithms. Notice that every term in the error calculation Equation 9 is also part of the calculation for the order 1.5 timestep from Equation 2 and thus no additional function evaluations

are required for this error estimate. Thus ESRK1 is an extension to the order 1.5 algorithm which allows for an efficient error estimation.

3. **Error estimation via natural embeddings.** The construction from section 2 is part of a more general construction which we call natural embeddings in Rößler SRK methods. Take a Rößler SRK method determined by coefficients $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$. For this method, we can generate an embedded order 1.0 method via a parameter δ by changing α to $\tilde{\alpha}$ where we subtract δ from two coefficients and add δ to two coefficients. Thus by Equation 2, this gives the deterministic error estimate

$$\begin{aligned} E_D &= \Delta t\left(f\left(t_n + c_{k_1}^{(0)}\Delta t, H_{k_1}^{(0)}\right) + f\left(t_n + c_{k_2}^{(0)}\Delta t, H_{k_2}^{(0)}\right)\right) \\ &\quad - \left(f\left(t_n + c_{k_3}^{(0)}\Delta t, H_{k_3}^{(0)}\right) + f\left(t_n + c_{k_4}^{(0)}\Delta t, H_{k_4}^{(0)}\right)\right), \end{aligned} \tag{10}$$

If we then change $\beta^{(3)}$ and $\beta^{(4)}$ to zero on the indices in the index set I_2, then this gives the noise error estimate

$$E_N = \left|\sum_{i\in I_2}\left(\beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right)\right|. \tag{11}$$

By the order conditions of A, these coefficient changes together give an order 1.0 method whose difference from the order 1.5 method $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$ is a bounded error estimate which is calculated directly from the coefficients of the order 1.5 method. This result is summarized in the following theorem:

Theorem 3.1 (Natural Embedding)**.** *For any order 1.5 SRK method with coefficients $\left(A^{(i)}, B^{(i)}, \beta_i, \alpha\right)$, there exists a $\delta \in \mathbb{R}$ such that $\left(A^{(i)}, B^{(i)}, \tilde{\beta}^{(i)}, \tilde{\alpha}\right)$ defines a natural order 1.0 embedded SRK method, where $\tilde{\beta}^{(3)}$ and $\tilde{\beta}^{(4)}$ are equivalent to $\beta^{(3)}$ and $\beta^{(4)}$ respectively except the indices in I_2 are zeroed, and $\tilde{\alpha}$ differs from α by δ at each coefficient in I_1 by a sign difference of $(-1)^{\sigma(i)}$ ($\sigma(i) = 1$ for at least one i and $\sigma(i) = 2$ for at least one i), and $|I_1|$ even. The one-step difference between the methods $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$ and $\left(A^{(i)}, B^{(i)}, \tilde{\beta}^{(i)}, \tilde{\alpha}\right)$ is bounded by*

$$E = \delta E_D + E_N, \tag{12}$$

where

$$E_D = \left|\Delta t\sum_{i\in I_1}(-1)^{\sigma(i)} f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right| \le \Delta t\sum_{i\in I_1}\left|f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right|, \tag{13}$$

$$E_N = \left|\sum_{i\in I_2}\left(\beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right)\right|. \tag{14}$$

Notice that from this theorem if we take the coefficients from the SRIW1 method and let $\delta = \frac{1}{6}$ then we arrive at the ESRK1 method from section 2. Also, note that this theorem be extended to the case where $|I_1| = 3$ by letting $\tilde{\alpha}$ differ from α by δ in one coefficient and $\frac{\delta}{2}$ in the two others. However, in order to satisfy the Order 0.5 constraint $\sum \alpha_i = 1$, we cannot simply modify a single component.

This implies that there is no need to specifically derive embedded pairs of methods. Instead, for any order 1.5 Rößler SRK method that one constructs, there exists a naturally embedded order 1.0 method such that E is an error estimate, and thus the error estimator can be utilized without ever explicitly constructing the order 1.0 method. In addition, the calculation of E only uses values found in the order 1.5

timestep from Equation 2. As a result, the error estimate is naturally obtained without any additional function evaluations. Lastly, the equation Equation 12 also has a adjustable parameter δ, which allows the user to scale the relative contributions of the deterministic and noise terms to the overall error estimate.

We note that all of these equations generalize to systems by considering the variables and functions as vectors and vector functions respectively, with the absolute value generalizing to an appropriate norm. This result also generalizes to multiple Ito dimensions, i.e., SODEs of the form

$$dX_t = f(X_t, t)dt + \sum_k^d g_k(X_t, t)dW_t, \tag{15}$$

in the case where the noise function g is diagonal by a similar construction on the Rößler SRID, giving the error estimates

$$E_D = \left|\Delta t \sum_{i\in I_1}(-1)^{\sigma(i)} f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right| \leq \Delta t \sum_{i\in I_1}\left|f\left(t_n + c_i^{(0)}\Delta t, H_i^{(0)}\right)\right|, \tag{16}$$

$$E_N = \left|\sum_k^d \sum_{i\in I_2}\left(\beta_i^{(3)}\frac{I_{(1,0)}}{\Delta t} + \beta_i^{(4)}\frac{I_{(1,1,1)}}{\Delta t}\right) g_k\left(t_n + c_i^{(1)}\Delta t, H_i^{(1)}\right)\right|. \tag{17}$$

Note that in the case of SRA-type methods (additive noise), a similar theorem also holds. By Equation 5 on the multiple Ito dimension SDE Equation 15 we utilize the same construction and receive a natural embedding with the error estimation Equation 12 containing the same deterministic error estimation *Equation* 13, with a new noise-error estimation:

$$E_N = \left|\sum_k^d \sum_{i=1}^s \beta_i^{(2)}\frac{I_{(1,0)}}{\Delta t} g_k(t_n + c_i^{(1)}\Delta t)\right|. \tag{18}$$

4. Three algorithms for rejection sampling with memory.

4.1. Rejection sampling for stochastic differential equations.

As in deterministic rejection sampling algorithms [1, 21, 11], define

$$e = \sqrt{\frac{1}{n}\sum_i^n \left(\frac{E_i}{sc_i}\right)}, \tag{19}$$

where

$$sc_i = \epsilon_{abs} + X_t^{(i)}\epsilon_{rel}, \tag{20}$$

is the absolute/relative mixed error with ϵ is the user-chosen error absolute and relative tolerances respectively, E_i is the error estimation of the ith element computed with the methods from section 3, and n is the size of the system. Using this measure of the system error with respect to tolerance, it is common in deterministic systems to define

$$q = \left(\frac{1}{\gamma e}\right)^{1/(\mathcal{O}+1)}, \tag{21}$$

where γ is a penalty factor (in deterministic methods it is often taken as $\gamma = 2$) and $\mathcal{O}$ is the minimum order of the numerical methods used in the error estimators. For solving deterministic equations, the rejection sampling method for adaptive time-stepping can be summarized as:

1. If $q < 1$, reject the initial choice of h and repeat the calculation with qh

2. If $q \geq 1$, then accept the computed step and change h to $\min(h_{max}, qh)$ where h_{max} is chosen by the user.

For solving SODEs, this approach needs to be modified since rejection of future steps involves throwing away information about the future Wiener process which biases the sample statistics for the path. Our solution is to change rejection sampling so that all information about the future path is kept, and adapt the sampling of the Brownian path to accommodate for this information. This is summarized as follows:

1. If $q < 1$, reject the initial choice of h, store the values for the future steps of the Wiener process ΔW_t, and repeat the calculation with qh.
2. If $q \geq 1$, then accept the computed step and change h to $\min(h_{max}, qh)$ where h_{max} is chosen by the user. Use future information to determine the next step.

The algorithm for adapting the random number sampling to account for future information is as follows. As noted in C, due to the term $I_{(1,0)}$ in the SRK algorithm, we must consider two Brownian paths: W_t, the path from the SODE, and Z_t, an independent Brownian path for the approximation. First, propose a step with ΔW^P and ΔZ^P for a timestep h. If these are rejected, change the step size to qh. Thus we need to sample a new value at $W(t_n + qh)$ using the known values of $W(t_n)$ and $W(t_n + h)$. By the properties of the Brownian Bridge [20, 17], if $W(0) = 0$, $W(h) = L$, and $q \in (0, 1)$, then

$$W(qh) \sim N\left(qL, (1-q)qh\right). \tag{22}$$

Thus propose a step of size qh and take the random numbers $\Delta W = W(qh)$ and $\Delta Z = Z(qh)$ from the distribution Equation 22.

Notice that by the Markov property of Brownian motion, the immediate future and current timestep values fully characterize the distribution of the Brownian Bridge, and using the Brownian Bridge we can interpolate the Brownian process to whatever time-point we deem necessary. Thus if the every time we take a sample of the future Brownian process (i.e. take a random number) we save the random number (even if rejecting the step), we can recover the correct distribution for interpolations of the Brownian process using Equation 22, giving all of the necessary information for computing a step of size qh.

4.2. **Computational issues.** First we deduce the proper form to save the future information upon rejection and the necessary details to save. Assume that we are at a time t and the Brownian process changed by an amount ΔW^P in the timestep of size h. If there was no future information, this would be the value given by a random number generator with the distribution $N(0, h)$. When a step is rejected, a new stepsize is chosen as qh where $q < 1$. Knowing the Brownian process' value at t and $t+h$, we can determine a value for $W(t+qh)$ using Equation 22. Labeling ΔW as the change in W over our new step of size qh, we have future knowledge that the Brownian path changes over the interval $(t+qh, t+h)$ by $\overline{\Delta W} = \Delta W^P - \Delta W$ (note that for the same reasons, we have that the secondary Brownian motion Z used as part of the high-order integrator must also satisfy $\overline{\Delta Z} = \Delta Z^P - \Delta Z$). To store this future information, we save the 3-tuple $(\overline{\Delta W}, \overline{\Delta Z}, (1-q)h)$. The first two values denote how much the Brownian motions changed and the last denotes the length of the time interval over which the changes occurred. From these values we have the information necessary for to use Equation 22 in order to interpolate the change in Brownian motion from $t + qh$ to any value in $[t + qh, t + h]$. Since we

know the value of the Brownian motions at $t + qh$, we can perform rejections using this algorithm recursively to see that we can re-reject to any value in $[t, t + qh]$ and still accurately re-create the Brownian motions at any point in the full $[t, t + h]$. Therefore saving this 3-tuple at each rejection suffices for the full reconstruction of the Brownian processes. Since this algorithm requires very few floating point operations, this algorithm has a good potential for efficiency.

At this point a naive implementation runs into many practical computational issues. The first implementation which comes to mind is to store this 3-tuple of future information into an array. The storage must be a mutable multi-valued data structure because it is possible for consecutive rejections to occur, meaning that we may have to store many pieces of future information. However, when using general stepsizes, it is possible for a step to go past some values of future information, and if this step is then rejected and the future information is added to the end of the array, the array is no longer necessarily in time order. The naive approach to solve this problem is to expand to saving a 4-tuple which also includes the timepoint, and at every timestep either search the array or re-sort the array given new inputs. While the search can be made $\mathcal{O}(\log n)$ by using bisection (where n is the number of future information points), insertion into an arbitrary point in an array still constitutes $\mathcal{O}(n)$ operations since the other elements in the array must be moved. Since this is the cost per insertion and many insertions can required before accepting a timestep, it is essential to the efficiency of the algorithm to make this process more performant.

Knowing that we will need to insert arbitrary values into the middle of the array, one may then propose using a linked list data structure. However, if the search is done on a linked list data structure, then the bisection still constitutes $\mathcal{O}(n)$ operations due to the traversal structure. To alleviate these problems, we propose an algorithm using stacks. Stacks are computationally efficient data structures which allow for $\mathcal{O}(1)$ insertion and retrieval, but with the restriction that only the value at the top of the stack is readily available. By achieving this complexity, we minimize the computational cost of rejections, allowing for one to not have to be overly conservative with the choice of stepsize. In the following subsection we will show how to achieve general timestepping with $\mathcal{O}(1)$ information insertions and retrievals while only requiring the necessary floating point operations and ordering the data in a manner which is efficient for caching.

4.3. **The RSwM algorithms.** Due to the LIFO (last in, first out) structure of the stack data structure, if the algorithm iteratively adds the future portions to the stack, then the tuple at the top of the stack always denotes the values for the time interval of the immediate future. Thus, after a successful step, if the stack is non-empty, we use the values from the top of the stack to propose our next step. If the stack is ever empty, then this indicates that the algorithm currently has no future information. In this case, take new random numbers and propose a new step in the same manner as the non-adaptive algorithm. By using the stack in such a manner, we are able to perform all of the memory operations in $\mathcal{O}(1)$ with the only additional floating point operations due to adaptation being the few operations to calculate the L_i, resulting in a computationally efficient algorithm.

A psudocode version of the algorithm is given as Algorithm 1. While intuitive, RSwM1 is not the most efficient since it does not always use the estimated "best stepsize" qh. If the stack is non-empty, then the step size is taken to be the first component from the tuple on the top of the stack (L_1) which may be significantly

Algorithm 1 RSwM1

Set the values ϵ, h_{max}, T
Set $t = 0$, $W = 0$, $Z = 0$, $X = X_0$
Take an initial h, $\Delta Z, \Delta W \sim N(0, h)$
while $t < T$ **do**
    Attempt a step with h, ΔW, ΔZ to calculate X_{temp} according to (2)
    Calculate E according to (9)
    Update q using (21)
    if $(q < 1)$ **then** ▷ *% Reject the Step*
        Take $\Delta\tilde{W} \sim N\left(q\Delta W, (1-q)qh\right)$ and $\Delta\tilde{Z} \sim N\left(q\Delta Z, (1-q)qh\right)$
        Calculate $\overline{\Delta W} = \Delta W - \Delta\tilde{W}$ and $\overline{\Delta Z} = \Delta Z - \Delta\tilde{Z}$
        Push $\left((1-q)\,h, \overline{\Delta W}, \overline{\Delta Z}\right)$ into stack S
        Update $h := qh$
        Update $\Delta W := \Delta\tilde{W}$, $\Delta Z := \Delta\tilde{Z}$
    else ▷ *% Accept the Step*
        Update $t := t + h$, $W := W + \Delta W$, $Z := Z + \Delta Z$, $X = X + X_{temp}$
        if (S is empty) **then**
            Update $c := \min(h_{max}, qh)$, $h := \min(c, T - t)$
            Take $\Delta W, \Delta Z \sim N(0, h)$
        else
            Pop the top of S as L
            Update $h := L_1$, $\Delta W := L_2$, $\Delta Z := L_3$
        end if
    end if
end while

less than qh. This constrains the algorithm to step to any timepoint t which was the value of a previously rejected timestep. Thus, RSwM1 is not an entirely general stepping algorithm.

To extend the previous algorithm, we wish to change the stepping algorithm when the stack is not empty to allow for unraveling the stack multiple times per step. To do so, let the algorithm take enough items off the stack so that the total length is close to the proposed timestep qh, and adjust for the difference.

A psudocode version of the algorithm is given as Algorithm 4 in Appendix D. The resulting algorithm, RSwM2, is more complex than RSwM1. However, the advantage of this algorithm is that if we propose a step of size $h_{new} = qh$, we will always step with a size h_{new} regardless of the number of items on the stack. In the example simulations in section 5, this algorithm is shown to be more efficient.

However, RSwM2 is not correct. The problem comes from "re-rejections". If the algorithm pops a more than one item off the stack and then rejects the step, these elements are lost. Thus when it tries to re-step with the proposed qh, it does not properly recreate our Brownian path since we will have lost some of the future information. In section 5 we see that this case is rare enough that the results are usually statistically correct, though we wish to extend this algorithm to a truly correct algorithm.

To ensure correctness we need to save the popped elements until an accepted step passes the point which they encode. The idea is to use a deque where future information is added to the back and re-popped information is added to the front.

However, whenever a step is accepted, we will wish to drop all of the elements added from the front. This means that the most natural algorithm would be similar to the common decomposition of the deque into two stacks. Instead of utilizing a deque directly, we will use one stack S_1 for the future information and another S_2 for the re-popped information. Whenever a value is popped from the future information stack S_1, it will be placed into S_2. If the step is rejected, those values will be popped back over to S_1, and if the step is accepted they will be discarded.

The algorithm RSwM3 is explained in more detail by example. Assume we reject a timestep which uses the first n elements from the stack, and that the new proposed timestep qh uses what would have only been the first k. The fix would be to add the last $n-k-1$ tuples of future information back to the stack (in reverse time order), solve the Brownian bridge problem with the k and $k+1$st items straddling the proposed step qh, and add the future information from qh to the $k+1$st items to the stack. The algorithm should then only "discard" items after they have been passed by a successful step.

If we saved the n tuples as they popped from the stack S_1 into a different stack S_2, then we can think of S_2 as the stack which contains all future information that is currently being used in a proposed timestep. Since this will add the elements to S_2 in chronological order, the elements of stack S_2 will pop out in reverse time order. Thus after rejecting the timestep, the $n-k$ elements we wish to save will be the first $n-k$ elements of the stack S_2. Therefore, after a rejection, the algorithm should pop off the first $n-k-1$ elements of S_2, re-add them to S_1, and pop one more element to solve a Brownian Bridge problem. Then after any successful timestep, the algorithm can simply discard all of the information in S_2.

A psudocode version of the algorithm is given as Algorithm 2. The resulting algorithm, RSwM3, is an extension to RSwM2 which keeps the property of always using the estimated "best timestep" qh but is now able to ensure correctness.

5. Numerical verification of correctness and efficiency.

In order to evaluate the efficiency of the adaptive methods, we implemented ESRK1 with the three rejection sampling with memory algorithms RSwM1, RSwM2, and RSwM3. We chose the three example equations Equation 31, Equation 33, and Equation 35 defined in E.

These three examples cover a variety of behaviors seen in SODEs. Example 1 is a linear stochastic differential equation. Since $\alpha > 0$, the solution tends to infinity and thus presents a case where the algorithms must decrease the timesteps in order to ensure the accuracy. Example 2 has a periodic solution and will be used to show that the error of our method decreases even in cases where the process is stable. Example 3 will be used to test the algorithm's ability to handle additive noise. For all of the tests, the relative tolerance was set to 0 and the absolute tolerance was adjusted in order to test various algorithm behaviors.

5.1. Correctness.

We first checked for the correctness of the algorithm. In order to study the correctness of each implementation, we ran RSwM1, RSwM2, and RSwM3 200 times on Equation 31, and tested the distribution of W_2. For Brownian motion, $W_t \sim N(0,t)$ and thus $\frac{W_2}{\sqrt{2}} \sim N(0,1)$ should be a standard normal random variable. In order to test that our rejection sampling algorithms did not change the distributions for the underlying Brownian processes, we used the Kolmogorov-Smirnov Test. The 1-Sample Kolmogorov-Smirnov test is a standard

Algorithm 2 RSwM3

Set the values ϵ, h_{max}, T
Set $t = 0$, $W = 0$, $Z = 0$, $X = X_0$
Take an initial h, $\Delta Z, \Delta W \sim N(0, h)$
while $t < T$ **do**
    Attempt a step with h, ΔW, ΔZ to calculate X_{temp} according to (2)
    Calculate E according to (9)
    Update q using (21)
    if $(q < 1)$ **then** ▷ *% Reject the Step*
        Set $h_s = 0$, $\Delta W = 0$, $\Delta Z = 0$
        while S_2 is not empty **do**
            Pop the top of S_2 as L
            if $h_s + L_1 < (1 - q)h$ **then**
                Update $h_s := h_s + L_1$
                Update $\Delta W_{tmp} := \Delta W_{tmp} + L_2$, $\Delta Z_{tmp} := \Delta Z_{tmp} + L_3$
            else
                Push L onto S_2 and break
            end if
        end while
        Set $h_K = h - h_s$, $K_2 = \Delta W - \Delta W_{tmp}$, $K_3 = \Delta Z - \Delta Z_{tmp}$
        Set $q_K = \frac{qh}{h_K}$
        Take $\Delta\tilde{W} \sim N(q_K K_2, (1 - q_K) q_K L_1)$
        Take $\Delta\tilde{Z} \sim N(q_K K_3, (1 - q_K) q_K L_1)$
        Pop $((1 - q_K)h_K, K_2 - \Delta\tilde{W}, K_3 - \Delta\tilde{Z})$ onto S_1
        Pop $(q_K L_1, \Delta\tilde{W}, \Delta\tilde{Z})$ onto S_2
        Update $\Delta W = \Delta\tilde{W}$, $\Delta Z = \Delta\tilde{Z}$, $h = qh$
    else ▷ *% Accept the Step*
        Update $t := t + h$, $W := W + \Delta W$, $Z := Z + \Delta Z$, $X = X + X_{temp}$
        Empty S_2
        Update $c := \min(h_{max}, qh)$, $h := \min(c, T - t)$
        Set $h_s = 0$, $\Delta W = 0$, $\Delta Z = 0$
        while S is not empty **do**
            Pop the top of S_1 as L
            if $(h_s + L_1 < h)$ **then** ▷ *% Temporary not far enough*
                Update $h_s := h_s + L_1$, $\Delta W := \Delta W + L_2$, $\Delta Z := \Delta Z + L_3$
                Push a copy of L onto S_2
            else ▷ *% Final part of step from stack*
                Set $q_{tmp} = \frac{h - h_s}{L_1}$
                Let $\Delta\tilde{W} \sim N(q_{tmp} L_2, (1 - q_{tmp}) q_{tmp} L_1)$
                Let $\Delta\tilde{Z} \sim N(q_{tmp} L_3, (1 - q_{tmp}) q_{tmp} L_1)$
                Push $((1 - q_{tmp})L_1, L_2 - \Delta\tilde{W}, L_3 - \Delta\tilde{Z})$ onto S_1
                Update $\Delta W := \Delta W + \Delta\tilde{W}$, $\Delta Z := \Delta Z + \Delta\tilde{Z}$
            end if
        end while
        ▷ *% Update for last portion to step. Note zero if final part is from stack*
        if $(h - h_s$ is not zero$)$ **then**
            Let $\eta_W, \eta_Z \sim N(0, h - h_s)$
            Update $\Delta W = \Delta W + \eta_W$, $\Delta Z = \Delta Z + \eta_Z$
            Push $(h - h_s, \eta_W, \eta_Z)$ onto S_2
        end if
    end if
end while

non-parametric test of distributions which is known for being very sensitive and easily rejecting the null-hypothesis of standard normality [10]. A plot of the p-values for the Kolmogorov-Smirnov test on 20 trials tolerances of $10^{-1}, 10^{-3}, 10^{-5}$ is shown in Figure 2. We note that due to multiple sampling issues, one would expect 1/20 tests to fail at a p-value of .05. Indeed, for the RSwM1 and RSwM3 algorithms our results have around the expected number of failures. We see that RSwM2 has 6 failures, which shows that it also generally passes the tests even though there is no guarantee for its correctness. However, it is clear that the issue magnifies at lower tolerances. These results indicate that the normalized Brownian process's distribution was standard normal and thus were not significantly altered by the RSwM sampling algorithms. Therefore RSwM algorithms generate statistically correct results.

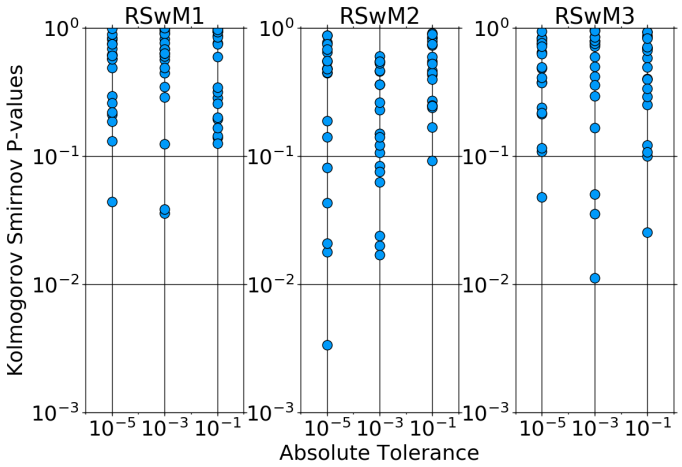

FIGURE 2. Adaptive algorithm Kolmogorov-Smirnov Tests. Equation 31 was solved from $t = 0$ to $t = 2$. Scatter plots of the p-values from Kolmogorov-Smirnov tests against the normal distribution. At the end of each run, a Kolmogorov-Smirnov test was performed on the values at end of the Brownian path for 200 simulations. The x-axis is the absolute tolerance (with relative tolerance set to zero) and the y-axis is the p-value of the Kolmogorov Smirnov tests.

As an additional verification, we included a Quantile-Quantile Plot (QQplot) of $\frac{W_2}{\sqrt{2}}$ against 10,000 standard normal random variables generated via MATLAB's randn command. The simulation results are shown in Figure 3. The estimated quantiles for $\frac{W_2}{\sqrt{2}}$ as generated by RSwM 1 through 3 line up on the $x = y$ axis, indicating that $\frac{W_2}{\sqrt{2}}$ follows a standard normal distribution. To test for edge effects, we ran the same calculation with higher/lower numbers of paths. This shows that as the number of paths increases, the number of quantiles which align also increases. These results show that although we cannot guarantee the correctness of the RSwM2, its sample statistics cannot be distinguished from the correct algorithms. This indicates that the edge case that the algorithm omits is a rare occurrence.

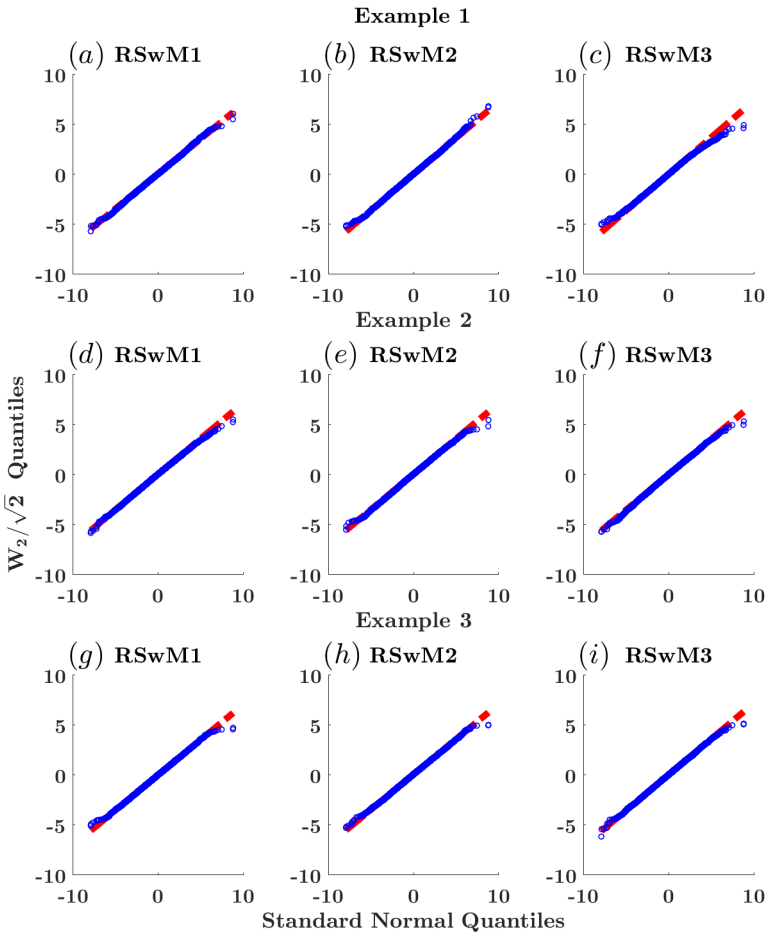

FIGURE 3. Adaptive algorithm correctness checks. QQplots of the distribution of the Brownian path at the end time $T = 2$ over 10,000 paths. The x-axis is the quantiles of the standard normal distribution while the y-axis is the estimated quantiles for the distribution of $W_2/\sqrt{2}$. Each row is for a example equation for Examples 1-3 respectively, and each column is for the algorithm RSwM1-3 respectively. $\epsilon = 10^{-4}$. The red dashed line represents $x = y$, meaning the quantiles of a 10,000 standard normal random variables equate with the quantiles of the sample. The blue circles represent the quantile estimates for $W(2)/\sqrt{2}$ which should be distributed as a standard normal.

5.2. **Accuracy and efficiency.** Next we tested the accuracy and efficiency of the three algorithms RSwM1, RSwM2, RSwM3 on the three example SDEs Equation 31, Equation 33, and Equation 35. The results, as summarized in Figure 4 and Figure 5, show that in almost every case, the improved algorithms RSwM2 and RSwM3 tend

to have lower runtimes and better convergence of the error at the final timepoint. Notice that the RSwM2 and RSwM3 algorithms tend to have error convergences much closer to log-linear, whereas RSwM1 tends to flatten out at lower tolerances. Also notice how in all cases, the RSwM2 and RSwM3 algorithms have comparable runtimes which are much smaller than those of RSwM1. This suggests that, while RSwM1 is vastly simpler to implement, the more complex RSwM2 and RSwM3 are more efficient.

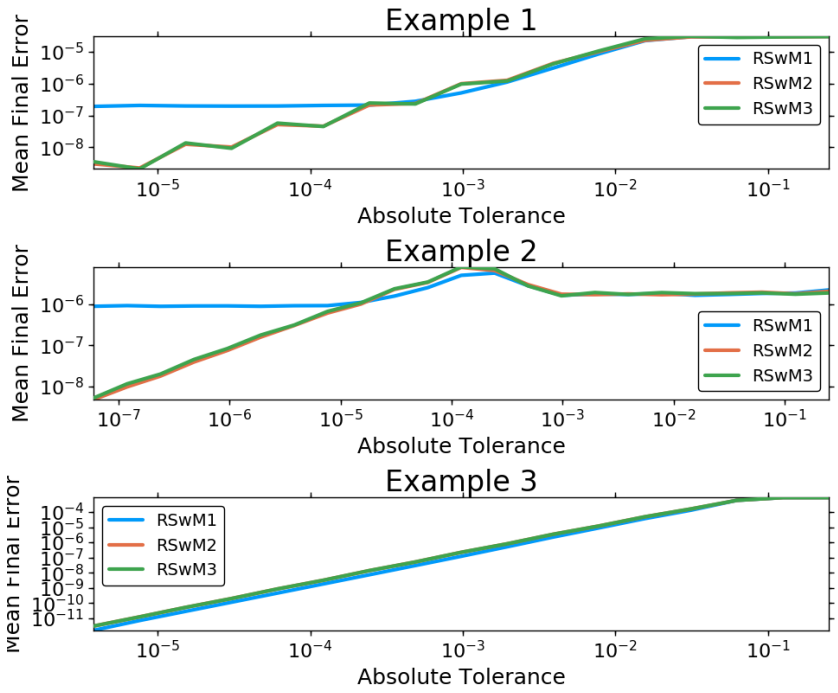

FIGURE 4. Adaptive Algorithm Error Comparison on Equation 31, Equation 33, and Equation 35. Comparison of the rejection sampling with memory algorithms on examples 1-3. Along the x-axis is ϵ which is the user chosen local error tolerance. The y-axis is the average l^2 error along the path. These values are taken as the mean for 100 runs. Both axes are log-scaled.

5.3. **Summary of numerical correctness and efficiency.** The results of the numerical experiments can be summarized as follows. From subsection 5.1 we see that by simulating Equation 31 Equation 33, and Equation 35 by ESRK1 with the time-stepping algorithms RSwM1, RSwM2, and RSwM3, the resulting Brownain paths have the appropriate sample statistics. This indicates that in all of these cases the algorithms produce the correct results. Notably, this shows that RSwM2, whose correctness we could not guarantee from its derivation, generates results which are can be statistically correct at low tolerances. Given that the Kolmogorov-Smirnov test is a stringent test for normality, this indicates that the edge case not appropriately dealt with in RSwM2 is a rare enough occurrence that the algorithm

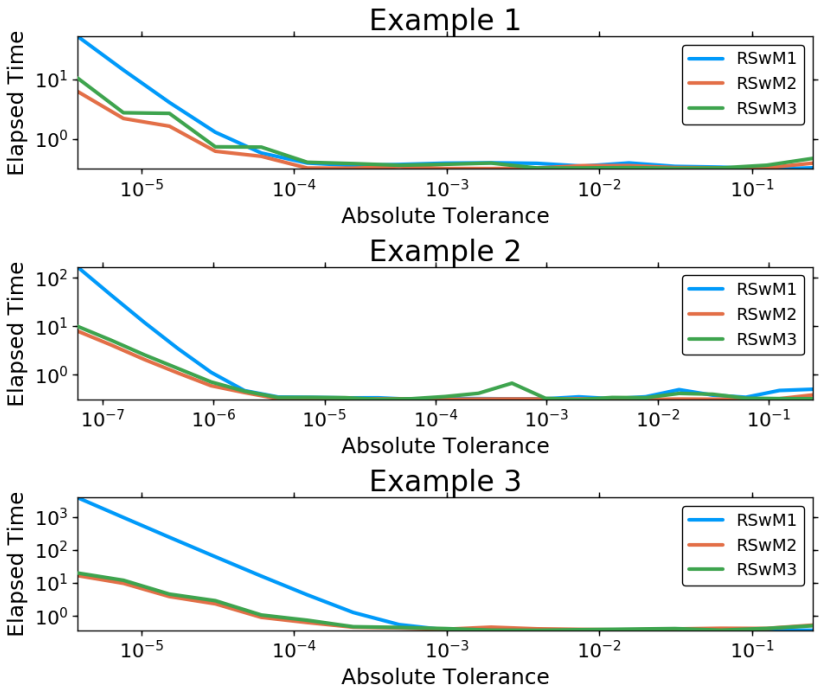

FIGURE 5. Adaptive Algorithm Timing Comparison. Comparison of the rejection sampling with memory algorithms. Along the x-axis is ϵ which is the user chosen local error tolerance. In the y-axis time is plotted (in seconds). The values are the elapsed time for a 1000 Both axes are log-scaled.

can likely be statistically indistinguishable from being correct in non demanding circumstances.

From subsection 5.2 we see that the RSwM2 and RSwM3 algorithms are the most efficient on the test equations Equation 31 and Equation 33 in terms of runtime and accuracy. Together, the results show that the three algorithms are correct and that there exists a tradeoff between them. The simplest algorithm to implement is RSwM1, but on some classes of problems it can perform slower than the other algorithms. RSwM2 is by a slight margin the fastest algorithm but is only psudo-correct. RSwM3 is an extension of RSwM2 which, at the cost of adding correctness, adds complexity in the implementation and runs with only slightly decreased efficiency.

6. **Numerical experiments with systems of SDEs.** Next we wished to show the applicability of the adaptive algorithms for solving large systems of nonlinear equations. These examples better match user cases, though no analytical solution exists for the error analysis. Instead, we wish to show how the error tolerance parameters allow for specifying the granularity of the numerical approximation, and demonstrate that this algorithm can be used to solve problem which would be computationally intractable without the adaptivity.

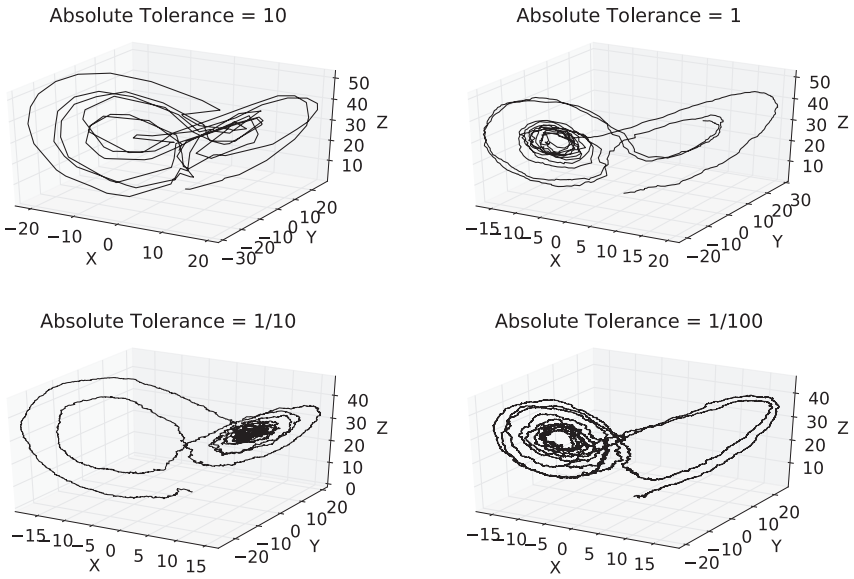

FIGURE 6. Solution to the Lorenz system Equation 39 on $t \in [0, 10]$ with additive noise and parameters $\alpha = 10$, $\rho = 28$, $\sigma = 3$, and $\beta = 8/3$ at varying tolerances. The system was solved using ESRK+RSwM3 with the relative tolerance fixed at zero and varying absolute tolerances.

6.1. **Control of numerical granularity.** To test the ability for the error tolerance parameters to control the granularity of the solution, the ESRK1 algorithm with RSwM3 was used to solve the well-known Lorenz system with additive noise. The parameters were chosen so the deterministic solution is chaotic and generates the well-known Lorenz attractor. The relative tolerance was fixed at zero and equation was solved at a variety of absolute tolerances. The solution was plotted as Figure 6. One can see that as the tolerance is decreased, the solution becomes more exact. Thus the RSwM tolerance parameters allow the users to adjust the trade-off between the exactness of the numerical approximation and computational time on nonlinear systems of equations.

6.2. **Large nonlinear system on cell differentiation.** Next we demonstrated the ability of the ESRK+RSwM algorithms to allow for solving large systems of nonlinear equations. As a test, we chose a model of stochastic cell differentiation in the epithelial-to-mesenchymal transitions of somatic cells. The system of 19 equations is given in Appendix F. This system serves as an ideal test case for adaptive algorithms since the stochastic switches give large intervals of relative stability with small intervals of extreme stiffness. Thus the ability to be able to automatically adjust the timestep depending on whether a stochastic transition is occurring is fundamental for making the system computationally tractable. In Figure 7 we show the solution on a time interval from 0 to 500 for two indicators of the cell states, Ecad and Vim, and the corresponding stepsizes the ESRK+RSwM algorithm used to solve the equations throughout the simulation. This figure shows

that the ESRK+RSwM algorithm is able to reproduce the scientific results, and that the adaptive algorithm is able to adjust the timestep between 10^{-4} and 10^{-11} to ensure accuracy and stability.

Unlike the deterministic case, whether a given solution is numerically stable for a given stepsize is dependent on the random Brownian trajectory and thus one cannot as easily classify the methods as stable for a given stepsize on a given problem. Instead, we define a method as sufficiently stable at a stepsize h if the method has no unstable trajectories in a sufficiently large ensemble. The reason for this choice is because it gives a measure for how fast a Monte Carlo simulation can correctly be computed. The condition that no trajectories can fail is because if the noisiest (and thus most unstable) trajectories are always discarded from the ensemble, the resulting Monte Carlo solution would be biased. This is a major concern for many models in practice. For example, the chosen epithelial-to-mesenchymal transition model has the most numerical instability during the stochastic switching events, but these events are the quality of interest and thus must be accurate in the resulting simulations.

Therefore, to test how efficiently the algorithms could produce statistically correct results on a stiff system, we solved cell model from Appendix F 10,000 times via the Euler-Maruyama, Runge-Kutta Milstein (RK-Mil) [15], and Rößler SRI methods using increments of $h = 2^{-i}$ and calculated the number of runs which diverged (failed) and the elapsed time for the 10,000 runs (note that the algorithm exited and denoted a run as diverged when encountering a NaN). These results are compared to the adaptive algorithm in Table 2. The adaptive algorithm solved 10,000 simulations with no failures in 187 seconds. The comparable value for fixed timestep methods, the shortest time for which there were no failures, was the Euler-Maruyama algorithm with $h = 2^{-20}$ which took 2286 seconds. This shows that the adaptive algorithm performed 12.28x as fast as the fastest fixed time-stepping algorithm, indicating that the advantages of adaptive time-stepping far out-weighted the overheads of the adaptive algorithm. Note that this testing method also far underestimates the advantages of the adaptive method: from Figure 7 we see that the most stiff portions are during the middle of stochastic transition events which would not happen during 1 second simulations. However, testing multiple solutions of the non-adaptive algorithms at such a low h in order to converge on the full $t \in [0, 500]$ was not computationally feasible. Indeed, one large advantage of the adaptive method is that the tests to determine the appropriate h are computationally expensive themselves. Also note that the tolerance for the adaptivity had to be set low in order to ensure stability. The advantage of the adaptive method could be made even larger by using a numerical method with a larger stability region. The application of the RSwM algorithms to more stable solvers for stiff systems and higher weak order solvers is a subject for future research.

7. **Implementation issues.**

For deploying these algorithms, there are a few extra implementation issues that should be addressed. First of all, the choice of q is due to calculations for the optimal stepsize for deterministic equations [7] and thus does not directly apply to stochastic systems. We instead use the deterministic q as a guideline. We heuristically modify it by noting that due to the extra overhead of the stack implementations, it may be wise to over-accelerate the stepsize changes and thus choose a higher power. In practice, we found

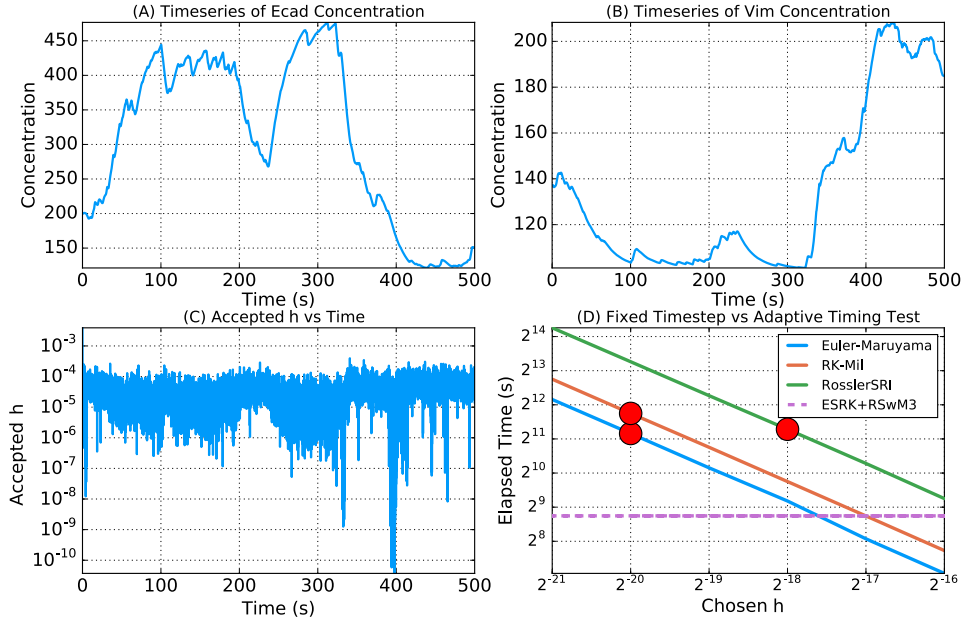

FIGURE 7. Stochastic cell differentiation model solutions. (A) Timeseries of the concentration of $[Ecad]$. The solution is plotted once every 100 accepted steps due to memory limitations. (B) Timeseries of the concentration of $[Vim]$. The solution is plotted once every 100 accepted steps due to memory limitations. (C) Accepted h over time. The h values were taken every 100 accepted steps due to memory limitations. (D) Elapsed time of the Euler-Maruyama and ESRK+RSwM3 algorithms on the stochastic cell model. Each algorithm was used to solve the model on $t \in [0, 1]$ 10,000 times. The elapsed time for the fixed timestep methods for given h's are shown as the filled lines, while the dashed line is the elapsed time for the adaptive method. The red circles denote the minimal h and times for which the method did not show numerical instability in the ensemble.

$$q = \left(\frac{1}{\gamma e}\right)^2, \tag{23}$$

to be an efficient choice. Future research should investigate this issue in more detail.

In addition, it is wise in practice to ensure that q takes acceptable values [11]. We found that the following tends to work well in practice:

$$q = \min(qmax, \max(q, qmin)), \tag{24}$$

where $qmax$ and $qmin$ are parameters set by the user. While for deterministic systems it's suggested that $qmax$ is between 4 and 10 [11], we note that our algorithm has a higher overhead than in the deterministic case, and thus further restricting q to reduce the number of rejections had the potential to increase the efficiency to of the algorithm. This was tested in Table 3 where we found a suitable

	Euler-Maruyama		Runge-Kutta Milstein		Rößler SRI	
Δt	Fails (/10,000)	Time (s)	Fails (/10,000)	Time (s)	Fails (/10,000)	Time (s)
2^{-16}	137	133.35	131	211.92	78	609.27
2^{-17}	39	269.09	26	428.28	17	1244.06
2^{-18}	3	580.14	6	861.01	0	2491.37
2^{-19}	1	1138.41	1	1727.91	0	4932.70
2^{-20}	0	2286.35	0	3439.90	0	9827.16
2^{-21}	0	4562.20	0	6891.35	0	19564.16

TABLE 2. Fixed timestep method fails and runtimes. The fixed timestep algorithms and ESRK+RSwM3 algorithms were used to solve the stochastic cell model on $t \in [0, 1]$ 10,000 times. Failures were detected by checking if the solution contained any NaN values. During a run, if any NaNs were detected, the solver would instantly end the simulations and declare a failure. The runtime for the adaptive algorithm (with no failures) was 186.81 seconds.

	Example 1		Example 2		Example 3		Cell Model
qmax	Time (s)	Error	Time (s)	Error	Time (s)	Error	Time (s)
$1 + 2^{-5}$	37.00	2.57e-8	60.87	2.27e-7	67.71	3.42e-9	229.83
$1 + 2^{-4}$	34.73	2.82e-8	32.40	3.10e-7	66.68	3.43e-9	196.36
$1 + 2^{-3}$	49.14	3.14e-8	132.33	8.85e-7	65.94	3.44e-9	186.81
$1 + 2^{-2}$	39.33	3.59e-8	33.90	1.73e-6	66.33	3.44e-9	205.57
$1 + 2^{-1}$	38.22	3.82e-8	159.94	2.58e-6	68.16	3.44e-9	249.77
$1 + 2^{0}$	82.76	4.41e-8	34.41	3.58e-6	568.22	3.44e-9	337.99
$1 + 2^{1}$	68.16	9.63e-8	33.98	6.06e-6	87.50	3.22e-9	418.78
$1 + 2^{2}$	48.23	1.01e-7	33.97	9.74e-6	69.78	3.44e-9	571.59

TABLE 3. *qmax* determination tests. Equation 31, Equation 33, and Equation 35 were solved using the ESRK+RSwM3 algorithm with a relative tolerance of 0 and absolute tolerance of 2^{-14}. The elapsed time to solve a Monte Carlo simulation of 100,000 simulations to $T = 1$ was saved and the mean error at $T = 1$ was calculated. The final column shows timing results for using ESRK+RSwM3 on the stochastic cell model from F solved with the same tolerance settings as in subsection 6.2 to solve a Monte Carlo simulation of 10,000 simulations.

value $qmax = 1.125$ which we used as the default throughout this paper (except in subsection 5.2 where $qmax = 10$ was used in order to better evaluate the ability to accurately unravel the stack). Further research should look into the usage of PI-controlled stepsize choices to further reduce the number of rejections.

Also, one should note that elements entering the stack should be sanitized in order to eliminate issues due to round-off error. In some cases the mathematical algorithm may specify very small intervals to go on the stack. This is particularly the case when q,q_{tmp}, or q_K are close to 1 when the step size is very small. When this item from the stack is eventually used to calculate q_{tmp}, there is the possibility of overflow which can lead to NaNs and stall the algorithm. To avoid this issue, it

is wise to discard any interval smaller than some size. We found 10^{-14} to be a good cutoff.

Lastly, it is common for many adaptive implementations to give a heuristic determination of an initial stepsize. Using the simple approximation that with 99% probability a $N(0,\sigma)$ random variable is in the interval $(-3\sqrt{\sigma}, 3\sqrt{\sigma})$, we adapt the scheme from [11] to take in account this a priori estimate, adjust due to the symmetry of the normal distribution, assume the next error power is $\frac{1}{2}$ larger, and arrive at the following 3.

Algorithm 3 Initial h Determination

Let $d_0 = \|X_0\|$
Calculate $f_0 = f(X_0, t)$ and $\sigma_0 = 3g(X_0, t)$
Let $d_1 = \|\max(|f_0 + \sigma_0|, |f_0 - \sigma_0|)\|$
if $d_0 < 10^{-5}$ or $d_1 < 10^{-5}$ **then**
    Let $h_0 = 10^{-6}$
else
    Let $h_0 = 0.01(d_0/d_1)$
end if
Calculate an Euler step: $X_1 = X_0 + h_0 f_0$
Calculate new estimates: $f_1 = f(X_1, t)$ and $\sigma_0 = 3g(X_1, t)$
Determine $\sigma_1^M = \max(|\sigma_0 + \sigma_1|), |\sigma_0 - \sigma_1|)$
Let $d_2 = \|\max(|f_1 - f_0 + \sigma_1^M|, |f_1 - f_0 - \sigma_1^M|)\|/h_0$
if $\max(d_1, d_2) < 10^{-15}$ **then**
    Let $h_1 = \max(10^{-6}, 10^{-3}h_0)$
else
    Let $h_1 = 10^{-(2+\log_1 0(\max(d_1,d_2))/(order+0.5)}$
end if
Let $h = \min(100h_0, h_1)$

Note that $\|\cdot\|$ is the norm as defined in Equation 19 with Equation 20. In practice we found this to be a conservative estimate which would "converge" to some approximate stepsize in < 10 steps, making it useful in practice.

8. **Conclusion and discussion.** In this paper we have developed a class of adaptive time-stepping methods for solving stochastic differential equations. The approach is based on a natural creation of high Strong-order embedded algorithms (with implicit error calculations) and generalized rejection sampling techniques. These algorithms are highly efficient: the error estimators require no extra evaluations of the drift and diffusion terms, while the rejection sampling with memory algorithms are $\mathcal{O}(1)$ for rejecting and interpolating the Brownian motion, with RSwM2 and RSwM3 allowing for an unconstrained choice of timestep. Through numerical testing, we demonstrated the algorithm's ability to adapt the error to the user-chosen parameter ϵ, the prescribed local error threshold. We also have shown the efficiency of the algorithms and their ability to scale to efficiently solve large stiff systems of equations.

Our results show that rejection sampling with memory can be used to speed up the solution of large systems of equations, particularly those coming from biological models. Many of these models are created to show interesting behavior which is dependent on the stochasticity, such as stochastic switching, but can be difficult to

simulate due to the stiffness associated with these properties. Using such an adaptive algorithm allows the simulation to focus its computational time to accurately capture the rare but important events. Further research into the application of these methods with stiff solvers could further improve the effectiveness. Also, the adaptive methods reduce the practical amount of time to conduct such experiments since one major time-consuming activity can be finding a small enough stepsize so that large Monte Carlo simulations can run without any path diverging and biasing the result. Further research should look into using rejection sampling with memory combined with higher order weak convergence methods to improve the computational efficiency when investigating moment properties from Monte Carlo experiments.

One of the unique features of the adaptive methods is that the embedded algorithm does not require an explicit construction of the order 1.0 method. It also does not have any extra requirements on the order 1.5 algorithm. This allows one to use any order 1.5 Rößler SRK algorithm, and thus as new coefficients for these algorithms are found to have "better" properties, such as improved stability or lower highest-order truncation error, our construction shows the existence of embedded versions for adaptive algorithms by default.

There are many implementation details that can be examined in order to more efficiency utilize these algorithms. For example, in the algorithm RSwM3, one could simulate discarding the second stack S_2 by explicitly controlling the pointer on an array and thus not have to deallocate memory every accepted timestep. Further research can investigate such implementation details for even more computationally efficient versions. Note that the calculation of the tuple for the stack requires only 3 floating point calculations and the generation of the new random numbers requires only 3x2 floating point calculations. Thus in total the algorithm requires only ≈ 10 floating point operations per rejection. Therefore the adaptive time-stepping algorithms require a minimal number of floating point operations, making them have a theoretically small impact on the computational effort.

The time-stepping method can be easily extended to other types of equations. Notice that as stated this algorithm controls the strong or pathwise error. There are many cases where one may be interested in the weak or average error of a Monte Carlo simulation. Our algorithm can be used to create a weak error estimate as well. To do so, instead of solving N independent size n systems, combine the systems into an equivalent system of size nN. Notice that by the definition of e, the error estimate obtained by this algorithm will be an average error over all simulations, and thus give a weak estimate. Using this weak estimate along with the RSwM algorithms on high weak order methods could be a path of future research.

At its core, the time-stepping algorithms are methods to sub-sample a continuous-time stochastic process as needed and effectively reproduce the sample properties at every point. Thus for any continuous-time martingale which defines a type of differential equation, this method can be applied by simply knowing the sample statistics of a bridge-type problem. Thus no part of the derivation required that the underlying process was an Ito process. Therefore, if one uses an embedded pair of methods for Stratonovich SODEs, the adaptive time-stepping algorithms still apply. Also, this includes problems like stochastic partial differential equations (SPDEs). For example, for SPDEs with space-time white noise, if one imposes a space-discretization then one arrives at a system of SODEs for which the Brownian bridge statistics are the same as any standard system of SODEs and thus our

methods apply. If one needs to adapt the spatial discretization, then this also follows the sample properties of the Brownian bridge and thus one can upsample the process along space until the desired error is met. For a Hilbert space valued Brownian motion like a Q-Wiener process, a Brownian bridge-like problem will need to be solved but the same general algorithm holds.

Notice that since the stepping algorithms only require the values on the stack, the RSwM form of adaptivity allows one to solve an equation while only storing the current values and the Brownian stack values. This is important for solving large systems of SPDEs which become memory-bound if one attempts to store the solution of the Brownain path at every timepoint. During our study of the implementation we saw the stack reaching maximum sizes of 20, and thus only used a modest amount of memory. For problems which are more memory bound, one can effectively decrease the maximum stack size by changing around some of the parameters, for example lowering the q exponent or decreasing $qmax$.

This method also provides an effective way to develop adaptive algorithms for variable-rate Markovian switching and jumps [18]. These problems are usually difficult to numerically simulate because one must effectively estimate the times for switching or else incur a large penalty due to the discontinuity. One can normally estimate whether a jump has occurred in an interval to a certain degree of accuracy, and using our adaptive algorithm one can hone in on the time at which the jump occurred, effectively solve the continuous problem to that time, and apply the jump. Therefore, the embedded time-stepping method along with the new rejection sampling algorithm developed in this work provides a general framework which one could build on for adaptive algorithms in solving many other stochastic problems.

Lastly, we wish to note that implementations of these algorithms are being released as part of a package DifferentialEquations.jl. DifferentialEquations.jl is a Julia library for solving ODEs, SDEs, DDEs, DAEs, and certain classes of PDEs and SPDEs, using efficient solvers in an easy-to-use scripting language. This package, developed by the authors, is freely available and includes additional functionalities such as parallelizing Monte Carlo experiments using the discussed methods on HPCs.

Acknowledgments. This work was partially supported by NIH grants P50GM 76516 and R01GM107264 and NSF grants DMS1562176 and DMS1161621. This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1321846, the National Academies of Science, Engineering, and Medicine via the Ford Foundation, and the National Institutes of Health Award T32 EB009418. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of the NIH.

The authors would like to thank Tom Breloff for his responsiveness and development of the Plots.jl Julia package which was used extensively in this publication. We would also like to thank Catherine Ta and Tian Hong for sharing their code for the stochastic cell model. We thank Gerrit Ansmann and the reviewers for their comments. Lastly, I would like to thank the Julia community for their continuing help and support.

Appendix.

A. **Order conditions for Rößler-SRI methods.** The coefficients $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$ must satisfy the following order conditions to achieve order .5:

1. $\alpha^T e = 1$
2. $\beta^{(1)^T} e = 1$
3. $\beta^{(2)^T} e = 0$
4. $\beta^{(3)^T} e = 0$
5. $\beta^{(4)^T} e = 0$

additionally, for order 1:

1. $\beta^{(1)^T} B^{(1)} e = 0$
2. $\beta^{(2)^T} B^{(1)} e = 1$
3. $\beta^{(3)^T} B^{(1)} e = 0$
4. $\beta^{(4)^T} B^{(1)} e = 0$

and lastly for order 1.5:

1. $\alpha^T A^{(0)} e = \frac{1}{2}$
2. $\alpha^T B^{(0)} e = 1$
3. $\alpha^T \left(B^{(0)} e\right)^2 = \frac{3}{2}$
4. $\beta^{(1)^T} A^{(1)} e = 1$
5. $\beta^{(2)^T} A^{(1)} e = 0$
6. $\beta^{(3)^T} A^{(1)} e = -1$
7. $\beta^{(4)^T} A^{(1)} e = 0$
8. $\beta^{(1)^T} \left(B^{(1)} e\right)^2 = 1$
9. $\beta^{(2)^T} \left(B^{(1)} e\right)^2 = 0$
10. $\beta^{(3)^T} \left(B^{(1)} e\right)^2 = -1$
11. $\beta^{(4)^T} \left(B^{(1)} e\right)^2 = 2$
12. $\beta^{(1)^T} \left(B^{(1)} \left(B^{(1)} e\right)\right) = 0$
13. $\beta^{(2)^T} \left(B^{(1)} \left(B^{(1)} e\right)\right) = 0$
14. $\beta^{(3)^T} \left(B^{(1)} \left(B^{(1)} e\right)\right) = 0$
15. $\beta^{(4)^T} \left(B^{(1)} \left(B^{(1)} e\right)\right) = 1$

16. $$\frac{1}{2}\beta^{(1)^T} \left(A^{(1)} \left(B^{(0)} e\right)\right) + \frac{1}{3}\beta^{(3)^T} \left(A^{(1)} \left(B^{(0)} e\right)\right) = 0$$

where $f, g \in C^{1,2}(\mathcal{I} \times \mathbb{R}^d, \mathbb{R}^d)$, $c^{(i)} = A^{(i)} e$, $e = (1, 1, 1, 1)^T$ [22].

B. **Order conditions for Rößler-SRA methods.** The coefficients $\left(A^{(i)}, B^{(i)}, \beta^{(i)}, \alpha\right)$ must satisfy the conditions for order 1:

1. $\alpha^T e = 1$
2. $\beta^{(1)^T} e = 1$
3. $\beta^{(2)^T} e = 0$

and the additional conditions for order 1.5:

1. $\alpha^T B^{(0)} e = 1$
2. $\alpha^T A^{(0)} e = \frac{1}{2}$
3. $\alpha^T \left(B^{(0)} e\right)^2 = \frac{3}{2}$
4. $\beta^{(1)^T} c^{(1)} = 1$
5. $\beta^{(2)^T} c^{(1)} = -1$

where $c^{(0)} = A^{(0)} e$ with $f \in C^{1,3}(\mathcal{I} \times \mathbb{R}^d, \mathbb{R}^d)$ and $g \in C^1(\mathcal{I}, \mathbb{R}^d)$ [22]. From these conditions he proposed the following strong order 1.5 scheme found in Table 4 known as SRA1.

$c^{(0)}$	$A^{(0)}$	$B^{(0)}$	
	α^T	$\beta^{(1)^T}$	$\beta^{(2)^T}$

(A) Legend Table

0					
$\frac{3}{4}$	$\frac{3}{4}$		$\frac{1}{2}$		
	$\frac{1}{3}$	$\frac{2}{3}$	1	0	-1	1

(B) Coefficients Table

TABLE 4. SRA1. Table (a) shows the legend for how the numbers in in Table (b) correspond to the coefficient arrays/matrices $c^{(i)}, A^{(i)}, B^{(i)}, \alpha$, and $\beta^{(i)}$. Note that the matrices $A^{(i)}$ and $B^{(i)}$ are lower triangular since the method is explicit.

C. **Wiktorsson iterated stochastic integral approximations.** In the Ito expansions for the derivation of SRK methods of order greater than 1.0, iterated stochastic integrals appear. We denote these as:

$$I_{(1)} = \int_{t_n}^{t_{n+1}} dW_s, \tag{25}$$

$$I_{(1,1)} = \int_{t_n}^{t_{n+1}} \int_{t_n}^{s} dW_u dW_s, \tag{26}$$

$$I_{(1,1,\ldots,1_k)} = \int_{t_n}^{t_{n+1}} \int \ldots \int_{t_n}^{s} dW_{u_1} \ldots dW_s. \tag{27}$$

Note that for our purposes we are using a single Brownian path, though this extends to multiple Ito dimensions. For a timestep $h = t_{n+1} - t_n$, the approximation due to Wiktorsson [25, 22] is as follows:

$$I_{(1,1)} = \frac{1}{2}(I_{(1)}^2 - h), \tag{28}$$

$$I_{(1,1,1)} = \frac{1}{6}(I_{(1)}^3 - 3hI_{(1)}), \tag{29}$$

$$I_{(1,0)} = \frac{1}{2}h\left(I_{(1)} + \frac{1}{\sqrt{3}}\zeta\right). \tag{30}$$

$I_{(1)}$ is one timestep of the Brownian path W_t, which is referred to as $\Delta W \sim N(0,h)$. $\zeta \sim N(0,h)$ is a standard normal random variable which is independent of ΔW. If we collect all of the ζ from $[0,T]$, then its cumulative sum is itself an approximation to a Brownian path which we denote Z_t with discrete steps $\Delta Z = \zeta$. Z_t is independent of W_t and is the second Brownian path considered in the RSwM algorithms.

D. **RSwM2 algorithm specification.** The algorithm for RSwM2 is specified as Algorithm 4.

Algorithm 4 RSwM2

Set the values ϵ, h_{max}, T
Set $t = 0$, $W = 0$, $Z = 0$, $X = X_0$
Take an initial h, $\Delta Z, \Delta W \sim N(0, h)$
while $t < T$ **do**
    Attempt a step with h, ΔW, ΔZ to calculate X_{temp} according to (2)
    Calculate E according to (9)
    Update q using (21)
    if $(q < 1)$ **then** ▷ *% Reject the Step*
        Take $\Delta\tilde{W} \sim N\left(q\Delta W, (1-q)qh\right)$ and $\Delta\tilde{Z} \sim N\left(q\Delta Z, (1-q)qh\right)$
        Calculate $\overline{\Delta W} = \Delta W - \Delta\tilde{W}$ and $\overline{\Delta Z} = \Delta Z - \Delta\tilde{Z}$
        Push $\left((1-q)\,h, \overline{\Delta W}, \overline{\Delta Z}\right)$ into stack S
        Update $h := qh$
        Update $\Delta W := \Delta\tilde{W}$, $\Delta Z := \Delta\tilde{Z}$
    else ▷ *% Accept the Step*
        Update $t := t + h$, $W := W + \Delta W$, $Z := Z + \Delta Z$, $X = X + X_{temp}$
        Update $c := \min(h_{max}, qh)$, $h := \min(c, T - t_n)$
        Set $h_s = 0$, $\Delta W = 0$, $\Delta Z = 0$
        while S is not empty **do**
            Pop the top of S as L
            if $(h_s + L_1 < h)$ **then** ▷ *% Temporary not far enough*
                Update $h_s := h_s + L_1$, $\Delta W := \Delta W + L_2$, $\Delta Z := \Delta Z + L_3$
            else ▷ *% Final part of step from stack*
                Set $q_{tmp} = \frac{h - h_s}{L_1}$
                Let $\Delta\tilde{W} \sim N(q_{tmp}L_2, (1 - q_{tmp})q_{tmp}L_1)$
                Let $\Delta\tilde{Z} \sim N(q_{tmp}L_3, (1 - q_{tmp})q_{tmp}L_1)$
                Push $((1 - q_{tmp})L_1, L_2 - \Delta\tilde{W}, L_3 - \Delta\tilde{Z})$ onto S
                Update $\Delta W := \Delta W + \Delta\tilde{W}$, $\Delta Z := \Delta\tilde{Z}$
            end if
        end while
        ▷ *% Update for last portion to step. Note zero if final part is from stack*
        if $(h - h_s$ is not zero$)$ **then**
            Let $\eta_W, \eta_Z \sim N(0, h - h_s)$
            Update $\Delta W = \Delta W + \eta_W$, $\Delta Z = \Delta Z + \eta_Z$
        end if
    end if
end while

E. **Example equations.** The three example equations are:
Example 1.

$$dX_t = \alpha X_t dt + \beta X_t dW_t, \;\; X_0 = \frac{1}{2}, \tag{31}$$

where $\alpha = \frac{1}{10}$ and $\beta = \frac{1}{20}$. Actual Solution:

$$X_t = X_0 e^{\left(\beta - \frac{\alpha^2}{2}\right)t + \alpha W_t}. \tag{32}$$

Example 2.

$$dX_t = -\left(\frac{1}{10}\right)^2 \sin\left(X_t\right)\cos^3\left(X_t\right)dt + \frac{1}{10}\cos^2\left(X_t\right)dW_t, \;\; X_0 = \frac{1}{2}, \tag{33}$$

Actual Solution:

$$X_t = \arctan\left(\frac{1}{10}W_t + \tan\left(X_0\right)\right). \tag{34}$$

Example 3.

$$dX_t = \left(\frac{\beta}{\sqrt{1+t}} - \frac{1}{2\left(1+t\right)}X_t\right)dt + \frac{\alpha\beta}{\sqrt{1+t}}dW_t, \;\; X_0 = \frac{1}{2}, \tag{35}$$

where $\alpha = \frac{1}{10}$ and $\beta = \frac{1}{20}$. Actual Solution:

$$X_t = \frac{1}{\sqrt{1+t}}X_0 + \frac{\beta}{\sqrt{1+t}}\left(t + \alpha W_t\right). \tag{36}$$

Example 4.

$$\begin{aligned} dX_t &= \alpha(Y_t - X_t) + \sigma dW_t, & (37)\\ dY_t &= X_t(\rho - Z_t) - Y_t + \sigma dW_t, & (38)\\ dZ_t &= X_t Y_t - \beta Z_t + \sigma dW_t, & (39) \end{aligned}$$

where $X_0 = Y_0 = Z_0 = 0$, $\alpha = 10$, $\rho = 28$, $\sigma = 3$, and $\beta = 8/3$.

F. **Stochastic cell differentiation model.** The stochastic cell differentiation model is given by the following system of SDEs which correspond to a chemical reaction network modeled via mass-action kinetics with Hill functions for the feedbacks. This model was introduced in [12] to model stochastic transitions between Epithelial and Mesenchymal states.

$$\begin{aligned} A &= \left(\left([TGF] + [TGF0]\right)/J0_{snail}\right)^{n0_{snail}} + \left([OVOL2]/J1_{snail}\right)^{n1_{snail}}\\ \frac{d\left[snail1\right]_t}{dt} &= k0_{snail} + k_{snail}\frac{\left(\left([TGF] + [TGF0]\right)/J0_{snail}\right)^{n0_{snail}}}{\left(1+A\right)\left(1 + [SNAIL]/J2_{snail}\right)}\\ &- kd_{snail}\left([snail1] - [SR]\right) - kd_{SR}\left[SR\right]\\ \frac{d\left[SNAIL\right]}{dt} &= k_{SNAIL}\left([snail1] - [SR]\right) - kd_{SNAIL}\left[SNAIL\right]\\ \frac{d\left[miR34\right]}{dt} &= kO_{34} + \frac{k_{34}}{1 + \left([SNAIL]/J1_{34}\right)^{n1_{34}} + \left([ZEB]/J2_{34}\right)^{n2_{34}}}\\ &- kd_{34}\left([miR34] - [SR]\right) - \left(1 - \lambda_{SR}\right)kd_{SR}\left[SR\right]\\ \frac{d\left[SR\right]}{dt} &= Tk\left(K_{SR}\left([snail1] - [SR]\right)\left([miR34] - [SR]\right) - [SR]\right)\\ \frac{d\left[zeb\right]}{dt} &= k0_{zeb} + k_{zeb}\frac{\left([SNAIL]/J1_{zeb}\right)^{n1_{zeb}}}{1 + \left([SNAIL]/J1_{zeb}\right)^{n1_{zeb}} + \left([OVOL2]/J2_{zeb}\right)^{n2_{zeb}}} \end{aligned}$$

$$
\begin{aligned}
& - kd_{zeb}\left([zeb]-\sum_{i=1}^{5}C_5^i\,[ZR]\right)-\sum_{i=1}^{5}kd_{ZR_i}C_5^i\,[ZR_i]\\
\frac{d\,[ZEB]}{dt} &= k_{ZEB}\left([zeb]-\sum_{i=1}^{5}C_5^i\,[ZR_i]\right)-kd_{ZEB}\,[ZEB]\\
\frac{d\,[miR200]}{dt} &= k0_{200}+\frac{k_{200}}{1+\left([SNAIL]/J1_{200}\right)^{n1_{200}}+\left([ZEB]/J2_{200}\right)^{n2_{200}}}\\
& - kd_{200}\left([miR200]-\sum_{i=1}^{5}iC_5^i\,[ZR_i]-[TR]\right)\\
& - \sum_{i=1}^{5}\left(1-\lambda_i\right)kd_{ZR_i}C_5^i i\,[ZR_i]-\left(1-\lambda_{TR}\right)kd_{TR}\,[TR]\\
\frac{d\,[ZR_1]}{dt} &= Tk\left(K_1\left([miR200]-\sum_{i=1}^{5}iC_5^i\,[ZR_i]-[TR]\right)\right.\\
&\quad \left.\left([zeb]-\sum_{i=1}^{5}C_5^i\,[ZR_i]\right)-[ZR_1]\right)\\
\frac{d\,[ZR_2]}{dt} &= Tk\left(K_2\left([miR200]-\sum_{i=1}^{5}iC_5^i\,[ZR_i]-[TR]\right)[ZR_1]-[ZR_2]\right)\\
\frac{d\,[ZR_3]}{dt} &= Tk\left(K_3\left([miR200]-\sum_{i=1}^{5}iC_5^i\,[ZR_i]-[TR]\right)[ZR_1]-[ZR_3]\right)\\
\frac{d\,[ZR_4]}{dt} &= Tk\left(K_4\left([miR200]-\sum_{i=1}^{5}iC_5^i\,[ZR_i]-[TR]\right)[ZR_1]-[ZR_4]\right)\\
\frac{d\,[ZR_5]}{dt} &= Tk\left(K_5\left([miR200]-\sum_{i=1}^{5}iC_5^i\,[ZR_i]-[TR]\right)[ZR_1]-[ZR_5]\right)\\
\frac{d\,[tgf]}{dt} &= k_{tgf}-kd_{tgf}\left([tgf]-[TR]\right)-kd_{TR}\,[TR]\\
\frac{d\,[TGF]}{dt} &= k0_{TGF}+k_{TGF}\left([tgf]-[TR]\right)-kd_{TGF}\,[TGF]\\
\frac{d\,[TR]}{dt} &= Tk\left(K_{TR}\left([miR200]-\sum_{i=1}^{5}iC_5^i\,[ZR_i]-[TR]\right)\left([tgf]-[TR]\right)-[TR]\right)\\
\frac{d\,[Ecad]}{dt} &= k0_E+\frac{k_{E1}}{1+\left([SNAIL]/J1_E\right)^{n1_E}}+\frac{k_{E2}}{1+\left([ZEB]/J2_E\right)^{n2_E}}-kd_E\,[Ecad]\\
B &= k_{V1}\frac{\left([SNAIL]/J1_V\right)^{n1_V}}{1+\left([SNAIL]/J1_V\right)^{n1_V}}+k_{V2}\frac{\left([ZEB]/J2_V\right)^{n2_V}}{1+\left([ZEB]/J2_V\right)^{n2_V}}\\
\frac{d\,[Vim]}{dt} &= k0_V+\frac{B}{\left(1+[OVOL2]/J3_V\right)}-kd_V\,[Vim]\\
\frac{d\,[OVOL2]}{dt} &= k0_0+k_0\frac{1}{1+\left([ZEB]/J_0\right)^{n_0}}-kd_O\,[OVOL2]
\end{aligned}
$$

where

$$
\begin{aligned}
\sum_{i=1}^{5}iC_5^i\,[ZR_i] &= 5\,[ZR_1]+20\,[ZR_2]++30\,[ZR_3]+20\,[ZR_4]+5\,[ZR_5]\,,\\
\sum_{i=1}^{5}C_5^i\,[ZR_i] &= 5\,[ZR_1]+10\,[ZR_2]+10\,[ZR_3]+5\,[ZR_4]+[ZR_5]\,.
\end{aligned}
$$

The parameter values are given in Table 5.

Parameter	Value	Parameter	Value	Parameter	Value	Parameter	Value
$J1_{200}$	3	$J1_E$	0.1	K_2	1	$k0_O$	0.35
$J2_{200}$	0.2	$J2_E$	0.3	K_3	1	kO_{200}	0.0002
$J1_{34}$	0.15	$J1_V$	0.4	K_4	1	kO_{34}	0.001
$J2_{34}$	0.35	$J2_V$	0.4	K_5	1	kd_{snail}	0.09
J_O	0.9	$J3_V$	2	K_{TR}	20	kd_{tgf}	0.1
$J0_{snail}$	0.6	$J1_{zeb}$	3.5	K_{SR}	100	kd_{zeb}	0.1
$J1_{snail}$	0.5	$J2_{zeb}$	0.9	$TGF0$	0	kd_{TGF}	0.9
$J2_{snail}$	1.8	K_1	1	Tk	1000	kd_{ZEB}	1.66
$k0_{snail}$	0.0005	$k0_{zeb}$	0.003	λ_1	0.5	$k0_{TGF}$	1.1
$n1_{200}$	3	$n1_{snail}$	2	λ_2	0.5	$k0_E$	5
$n2_{200}$	2	$n1_E$	2	λ_3	0.5	$k0_V$	5
$n1_{34}$	2	$n2_E$	2	λ_4	0.5	k_{E1}	15
$n2_{34}$	2	$n1_V$	2	λ_5	0.5	k_{E2}	5
n_O	2	$n2_V$	2	λ_{SR}	0.5	k_{V1}	2
$n0_{snail}$	2	$n2_{zeb}$	6	λ_{TR}	0.5	k_{V2}	5
k_O	1.2	k_{200}	0.02	k_{34}	0.01	k_{tgf}	0.05
k_{zeb}	0.06	k_{TGF}	1.5	k_{SNAIL}	16	k_{ZEB}	16
kd_{ZR_1}	0.5	kd_{ZR_2}	0.5	kd_{ZR_3}	0.5	kd_{ZR_4}	0.5
kd_{ZR_5}	0.5	kd_O	1.0	kd_{200}	0.035	kd_{34}	0.035
kd_{SR}	0.9	kd_E	0.05	kd_V	0.05		

TABLE 5. Table of Parameter Values for the Stochastic Cell Model.

REFERENCES

[1] R. L. Burden and J. D. Faires, *Numerical Analysis*, 9th ed., Brooks/Cole, Cengage Learning, Boston, MA, 2011.

[2] K. Burrage and P. Burrage, General order conditions for stochastic runge-kutta methods for both commuting and non-commuting stochastic ordinary differential equation systems, *Appl. Numer. Math.*, **28** (1998), 161–177.

[3] K. Burrage and P. M. Burrage, High strong order explicit runge-kutta methods for stochastic ordinary differential equations, *Applied Numerical Mathematics*, **22** (1996), 81–101.

[4] P. M. Burrage, *Runge-Kutta Methods for Stochastic Differential Equations*, Thesis, The University of Queensland Brisbane, 1999.

[5] P. M. Burrage and K. Burrage, A variable stepsize implementation for stochastic differential equations, *SIAM Journal on Scientific Computing*, **24** (2002), 848–864.

[6] J. R. Cash and A. H. Karp, A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides, *ACM Transactions on Mathematical Software*, **16** (1990), 201–222.

[7] F. Ceschino, Modification de la longueur du pas dans l'integration numerique parles methodes a pas lies, *Chiffres,* **4** (1961), 101–106.

[8] J. R. Dormand and P. J. Prince, A family of embedded runge kutta formulae, *Journal of Computational and Applied Mathematics*, **6** (1980), 19–26.

[9] J. G. Gaines and T. J. Lyons, Variable step size control in the numerical solution of stochastic differential equations, *SIAM Journal on Applied Mathematics*, **57** (1997), 1455–1484.

[10] A. Ghasemi and S. Zahediasl, Normality tests for statistical analysis: A guide for non-statisticians, *Int J Endocrinol Metab*, **10** (2012), 486–489.

[11] E. Hairer, S. P. N orsett and G. Wanner, *Solving Ordinary Differential Equations I*, 2nd rev. edn, Springer series in computational mathematics. Springer-Verlag, Berlin, New York, 1993.

[12] T. Hong, K. Watanabe, C. H. Ta, A. Villarreal-Ponce, Q. Nie and X. Dai, An ovol2-zeb1 mutual inhibitory circuit governs bidirectional and multi-step transition between epithelial and mesenchymal states, *PLoS Comput Biol*, **11** (2015), e1004569.

[13] S. M. Iacus, *Simulation and Inference for Stochastic Differential Equations: With R Examples (Springer Series in Statistics),* Springer Publishing Company, Incorporated, 2008.

[14] J. Kaneko, Explicit order 1.5 runge kutta scheme for solutions of ito stochastic differential equations, *S/urikaisekikenky/usho K/oky/uroku,* **932** (1995), 46–60.

[15] P. E. Kloeden and E. Platen, *Numerical Solution of Stochastic Differential Equations*, Springer Berlin Heidelberg, 1992.
[16] H. Lamba, An adaptive timestepping algorithm for stochastic differential equations, *Journal of Computational and Applied Mathematics*, **161** (2003), 417–430.
[17] T. Liggett, *Continuous Time Markov Processes: An Introduction*, American Mathematical Society, 2010.
[18] X. Mao and C. Yuan, *Stochastic Differential Equations with Markovian Switching*, Imperial College Press, London, 2006.
[19] N. Hofmann, T. Muller-Gronbach and K. Ritter, Optimal approximation of stochastic differential equations by adaptive step-size control, *Mathematics of Computation*, **69** (2000), 1017–1034.
[20] B. Oksendal, *Stochastic Differential Equations: An Introduction with Applications*, Springer, 2003.
[21] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing,* Cambridge University Press, 2007.
[22] A. Rößler, Runge kutta methods for the strong approximation of solutions of stochastic differential equations, *SIAM Journal on Numerical Analysis*, **48** (2010), 922–952.
[23] T. Ryden and M. Wiktorsson, On the simulation of iterated ito integrals, *Stochastic Processes and their Applications* **91** (2001), 151–168.
[24] L. F. Shampine, Some practical runge-kutta formulas, *Mathematics of Computation*, **46** (1986), 135–150.
[25] M. Wiktorsson, Joint characteristic function and simultaneous simulation of iterated ito integrals for multiple independent brownian motions, *The Annals of Applied Probability*, **11** (2001), 470–487.

Received August 2016; revised December 2016.

E-mail address: contact@chrisrackauckas.com
E-mail address: qnie@math.uci.edu